Unit 1: Introduction to Server-Side Development

LECTURER: SURAJ PANDEY, CCT COLLEGE

Topics:

- 1.1. Overview of Web-I
- 1.2. HTTP request message, HTTP response Message
- 1.3. Differences between client-side and server-side technologies
- 1.4. Web Clients
- 1.5. Web Server
- 1.5.1. Role of the server in a web application
- 1.5.2. Overview of web servers - Apache - Nginx - IIS
- 1.6. Server-side architecture
- 1.6.1. Monolithic
- 1.6.2. Micro-services
- 1.7. Introduction to backend programming languages (Python, PHP, Java)
- 1.8. Understanding MVC (Model-View-Controller) architecture

1.1. Overview of Web-I

- The **World Wide Web (Web-I)** is a collection of websites and online services accessible via the **Internet**. It works using:
- **Clients** (Web browsers like Chrome, Firefox)
- **Servers** (Computers hosting websites, e.g., Apache, Nginx)
- **Protocols** (Rules for communication, e.g., HTTP, HTTPS)
- Web-I, also known as Web 1.0, was the first generation of the World Wide Web. It consisted mainly of static web pages that were created using HTML. Users could view information, but interaction was minimal or non-existent.

Characteristics:

- Static content
- Read-only web pages
- No interactivity
- Limited to displaying text and images
- Centralized content management

- **Advantages:**
- ✓ **Accessible worldwide** – Anyone with internet can use it.
  ✓ **Supports multimedia** – Videos, images, interactive content.
  ✓ **Platform-independent** – Works on Windows, Mac, Android, etc.
- **Disadvantages:**
- ✕ **Security risks** – Hackers can attack websites.
  ✕ **Requires internet** – No connectivity means no access.

1.2. HTTP Request & Response Messages

- **HTTP Request (From Client to Server)**
- **HTTP Request:**
- A message sent by the client (browser) to request a resource (HTML file, image, etc.) from the server.
- **Components:**
- Request Line (e.g., GET /index.html HTTP/1.1)
- Headers (Host, User-Agent, Accept)
- Body (optional, used in POST/PUT)

- When you visit a website, your browser sends an **HTTP Request** like this:
- **Example:**
- http
- Copy
- GET /home.html HTTP/1.1
- Host: www.website.com
- User-Agent: Chrome/Windows
- Accept-Language: en-US

- **Method:** GET (fetch data), POST (send data), PUT (update), DELETE (remove).
- **Path:** /home.html (the file requested).
- **Headers:** Extra info (browser type, language).

- **HTTP Response (From Server to Client)**
- A message sent from the server back to the client with the requested resource or error message.
- **Components:**
- Status Line (e.g., HTTP/1.1 200 OK)
- Headers (Content-Type, Content-Length)
- Body (HTML or data)
- The server replies with:
- **Example:**
- http
- Copy
- HTTP/1.1 200 OK
- Content-Type: text/html
- Content-Length: 500

- **Status Code:**
  - 200 = Success ✓
  - 404 = Page not found ✗
  - 500 = Server error 🔧
- **Headers:** Describe the response (Content-Type, Server).
- **Body:** The actual webpage (HTML, JSON, etc.).
- **Advantages:**
- ✓ Standard way for browsers and servers to talk.
  ✓ Works for all websites.
- **Disadvantages:**
- ✗ Without HTTPS, data can be intercepted.
  ✗ Too many requests can slow down a website

1.3. Client-Side vs Server-Side Technologies

Feature	Client-Side Technologies	Server-Side Technologies
Definition	Runs on the user's device (browser).	Runs on a remote server.
Languages	HTML, CSS, JavaScript (React, Angular, Vue).	Python, PHP, Java, Node.js, Ruby, C#.
Execution	Executed in the browser.	Executed on the web server.
Access to Resources	Limited to browser APIs (e.g., localStorage, DOM).	Full access to databases, file systems, and APIs.
Security	Less secure (code is visible to users).	More secure (logic is hidden from users).
Performance	Faster for UI interactions (no server round-trip).	Slower (depends on server processing and network).
Dependency	Requires browser compatibility.	Depends on server configuration and resources.
Examples	Form validation, animations, dynamic content updates.	User authentication, database queries, payment processing.

Advantages & Disadvantages

- **Client-Side**
- **✓ Pros:**
- Faster response times (no server delay).
- Reduces server load.
- Offline capabilities (e.g., Progressive Web Apps).
- **✕ Cons:**
- Limited functionality (no direct database access).
- Security risks (code is exposed).

- **Server-Side**
- **✓ Pros:**
- Handles complex logic (e.g., e-commerce checkout).
- Secure (sensitive operations stay hidden).
- **✕ Cons:**
- Slower (requires round-trip to server).
- Higher server costs (needs hosting).

1.4. Web Clients

- A **web client** is any software that accesses web content.
- **Types:**
- **Web Browsers** (Chrome, Firefox) – Renders HTML.
- **Mobile Apps** (Facebook, Instagram) – Uses APIs.
- **Command-Line Tools** (cURL, Postman) – Tests APIs.
- **Example:**
- When you open **Google Chrome** and search something, Chrome is the **client**.

1.5. Web Server

- **Different Web Servers**
- A **web server** is software that handles HTTP requests and serves web pages. Popular ones include:
- **1. Apache HTTP Server**
- **An open-source web server software, widely used and known for its flexibility and customization options.**
- **Developed By:** Apache Software Foundation
- **Best For:** Small to medium websites, WordPress hosting.
- **Pros:**
  ✓ Open-source & free.
  ✓ Highly customizable with modules.
- **Cons:**
  ✗ Slower under heavy traffic compared to Nginx.

- **2. Nginx (Engine-X)**
- **Another popular open-source web server, known for its high performance and efficiency, often used as a reverse proxy and load balancer.**
- **Developed By:** Igor Sysoev
- **Best For:** High-traffic sites, reverse proxying, load balancing.
- **Pros:**
  ✅ Faster than Apache for static content.
  ✅ Uses less memory.
- **Cons:**
  ❌ Harder to configure for beginners.

- **3. Microsoft IIS (Internet Information Services)**
- A web server developed by Microsoft, primarily used in Windows environments.
- **Developed By:** Microsoft
- **Best For:** Windows-based applications (.NET, ASP).
- **Pros:**
  ✓ Tight integration with Windows.
  ✓ Good for enterprise applications.
- **Cons:**
  ✗ Not open-source (paid license for Windows Server).

Comparison Table

Feature	Apache	Nginx	IIS
Speed	Moderate	Very Fast	Moderate
Config	Easy (.htaccess)	Complex	GUI-based
OS	Linux, Windows	Linux, Windows	Windows Only
Use Case	WordPress, PHP	High-traffic sites	.NET apps

Key Differences in Performance

Action	Apache	Nginx	IIS
Handle 10k requests	Slower (process-based)	Faster (event-driven)	Moderate (Windows opt.)
Static file delivery	Good	Excellent	Good
Config file	`.htaccess`	`nginx.conf`	GUI / XML

- **Why This Matters?**
- **For Developers:**
  - Choosing Nginx → Better for high-traffic sites.
  - Choosing Apache → Easier to configure plugins.
- **For Users:**
  - Faster servers (Nginx) → Pages load quicker.
  - Stable servers (Apache) → Fewer crashes.

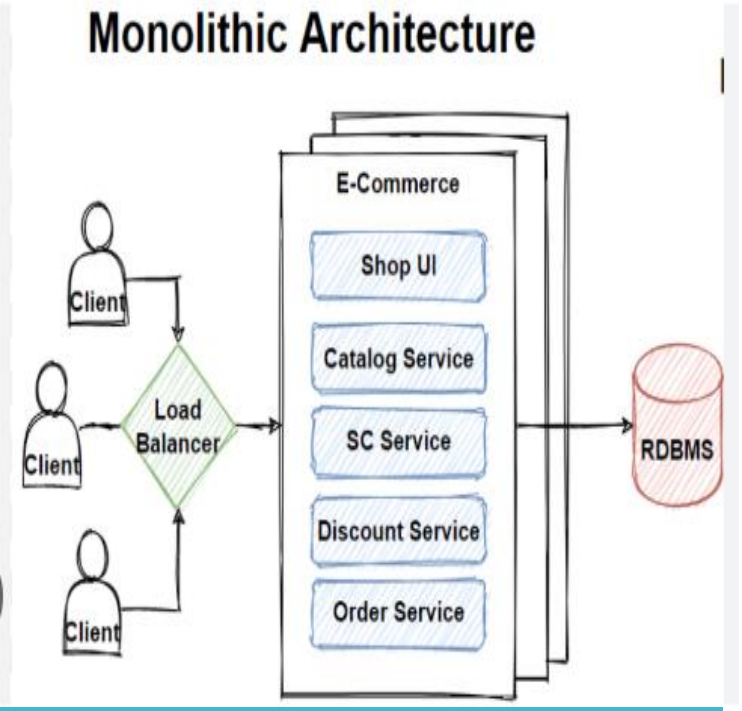
Web Server:

- A **web server** is a computer that stores and delivers websites.
- **1.5.1 Role of the Server in a Web Application:**
- The web server is responsible for receiving HTTP requests from web clients, processing them, and sending back HTTP responses. It acts as the host for web applications and their resources.
- **Role of a Web Server**
- Stores website files (HTML, CSS, images).
- Runs server-side code (PHP, Python).
- Connects to databases (MySQL).
- Sends responses to browsers.

Server Architectures & Backend Programming Languages:

- **Server Architectures**

- Server-side architectures like monolithic and microservices offer different approaches to building and managing applications. Monolithic architecture, a traditional approach, uses a single codebase for all application functionalities. In contrast, microservices break down applications into smaller, independently deployable services.

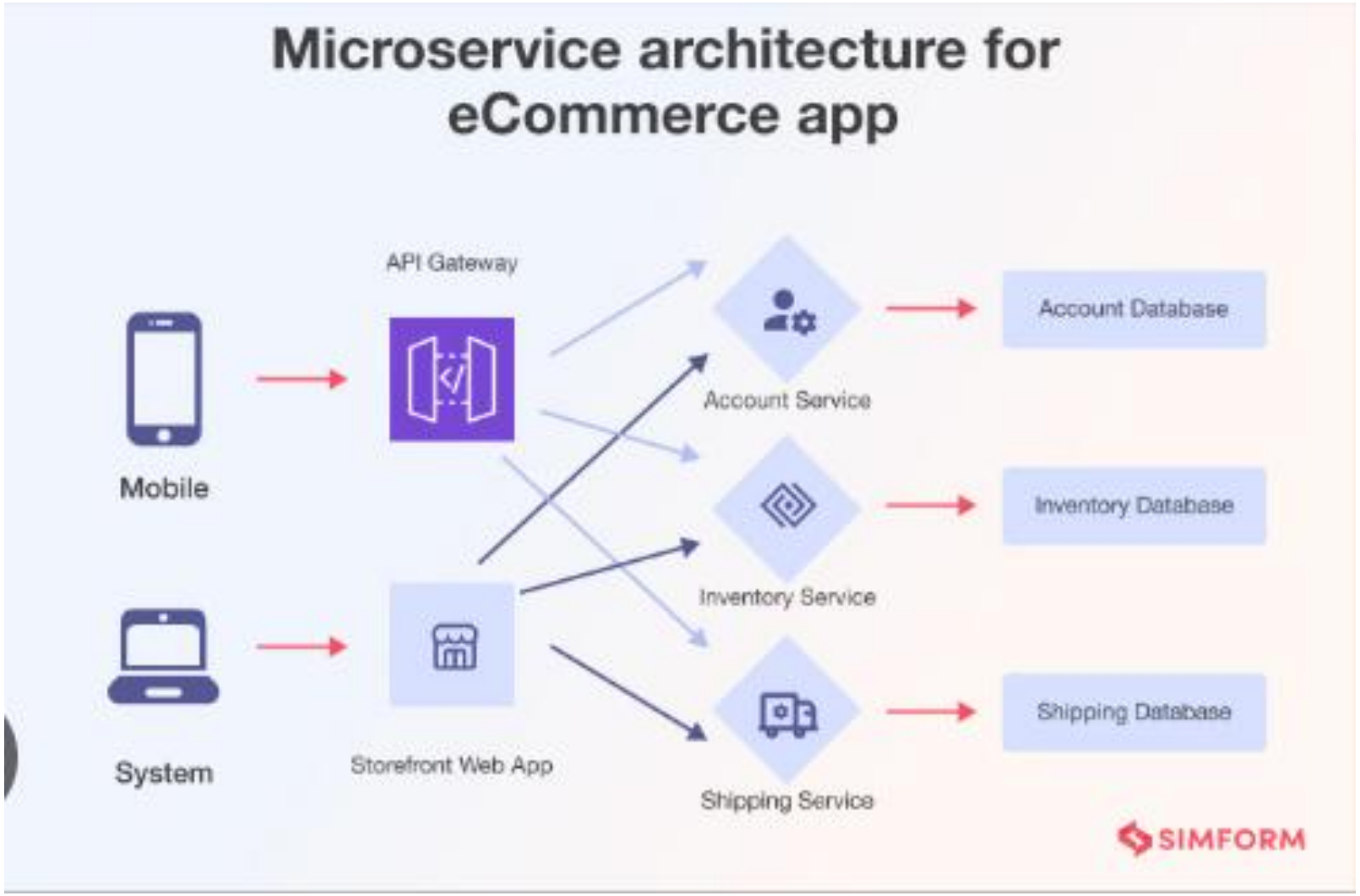


- 1.6.1. Monolithic Architecture:
- **Definition:**
- Monolithic architecture in software development refers to an application's structure where all components, including the user interface, business logic, and data access layers, are packaged and deployed as a single, unified unit. This contrasts with other architectures, like microservices, where the application is broken down into smaller, independently deployable services.
- **Example:** A WordPress blog where:
- Frontend (theme)
- Backend (PHP code)
- Database (MySQL)
  all run as one application.

- ✓ **Pros:**
- Simple to develop and deploy
- Easier debugging (all code in one place)
- Good for small projects
- ✕ **Cons:**
- Difficult to scale (must deploy entire app)
- Single point of failure
- Technology lock-in (hard to change frameworks)
- **When to use:** Small applications, MVPs, simple websites

- **Characteristics:**
  - **Centralized:** All parts of the application are tightly coupled and reside within the same codebase.
  - **Single Deployment:** The entire application is deployed as a single entity.
  - **Less Scalable:** Scaling requires deploying the entire application, potentially impacting performance and cost-effectiveness.
  - **Development Speed:** Can be slower due to the need to coordinate changes across the entire codebase.
  - **Maintenance:** Changes to one part of the application can impact other parts, making maintenance and debugging more challenging.
- **Example:**
- A traditional web application where all components, including the user interface, business logic, and database access, are within the same code base.

- 1.6.2. Microservices Architecture:
- **Definition:**
- Microservices architecture divides an application into smaller, independently deployable services that communicate with each other.
- **Example:** Netflix uses microservices for:
- User authentication service
- Recommendation engine
- Video streaming service
- Payment processing


Microservice architecture for eCommerce app
API Gateway
Mobile
System
Storefront Web App
Account Service
Inventory Service
Shipping Service
Account Database
Inventory Database
Shipping Database
SIMFORM

- ✓ **Pros:**
- Scalable (can scale individual services)
- Fault isolation (one service fails, others work)
- Technology flexibility (each service can use different languages)
- ✕ **Cons:**
- Complex to manage
- Requires DevOps expertise
- Network latency between services
- **When to use:** Large-scale applications, cloud-native apps, enterprise systems

- **Characteristics:**
  - **Decentralized:** Each service is self-contained and can be developed, deployed, and scaled independently.
  - **Loosely Coupled:** Services have minimal dependencies on each other.
  - **Independent Deployments:** Services can be deployed and updated independently without affecting other services.
  - **Scalability:** Individual services can be scaled independently, allowing for efficient resource allocation.
  - **Faster Development:** Smaller, independent services allow for faster development cycles and easier integration of new features.
  - **Fault Isolation:** If one service fails, it is less likely to bring down the entire application.
- **Example:**
- A modern e-commerce application where functionalities like product catalog, user authentication, payment processing, and order management are separated into individual microservices.

Backend Programming Languages

- **Python:**
- A versatile and widely used programming language known for its readability and extensive libraries, often used for building web applications and backends.
- **PHP:**
- A scripting language primarily used for web development, particularly on the server-side, and commonly used with web frameworks.
- **Java:**
- A powerful and object-oriented language, commonly used for building robust and scalable web applications and enterprise-level systems.

Comparison of Popular Backend Languages

Language	Best For	Frameworks	Pros	Cons
JavaScript (Node.js)	Real-time apps, APIs	Express.js, NestJS	Full-stack with one language, fast I/O	Callback hell, single-threaded
Python	Data science, ML, web apps	Django, Flask	Easy to learn, rich libraries	Slower execution, not ideal for mobile
Java	Enterprise apps, banking	Spring, Jakarta EE	High performance, strong typing	Verbose code, steep learning curve
PHP	Web development, CMS	Laravel, Symfony	Great for web, many hosting options	Inconsistent standard library

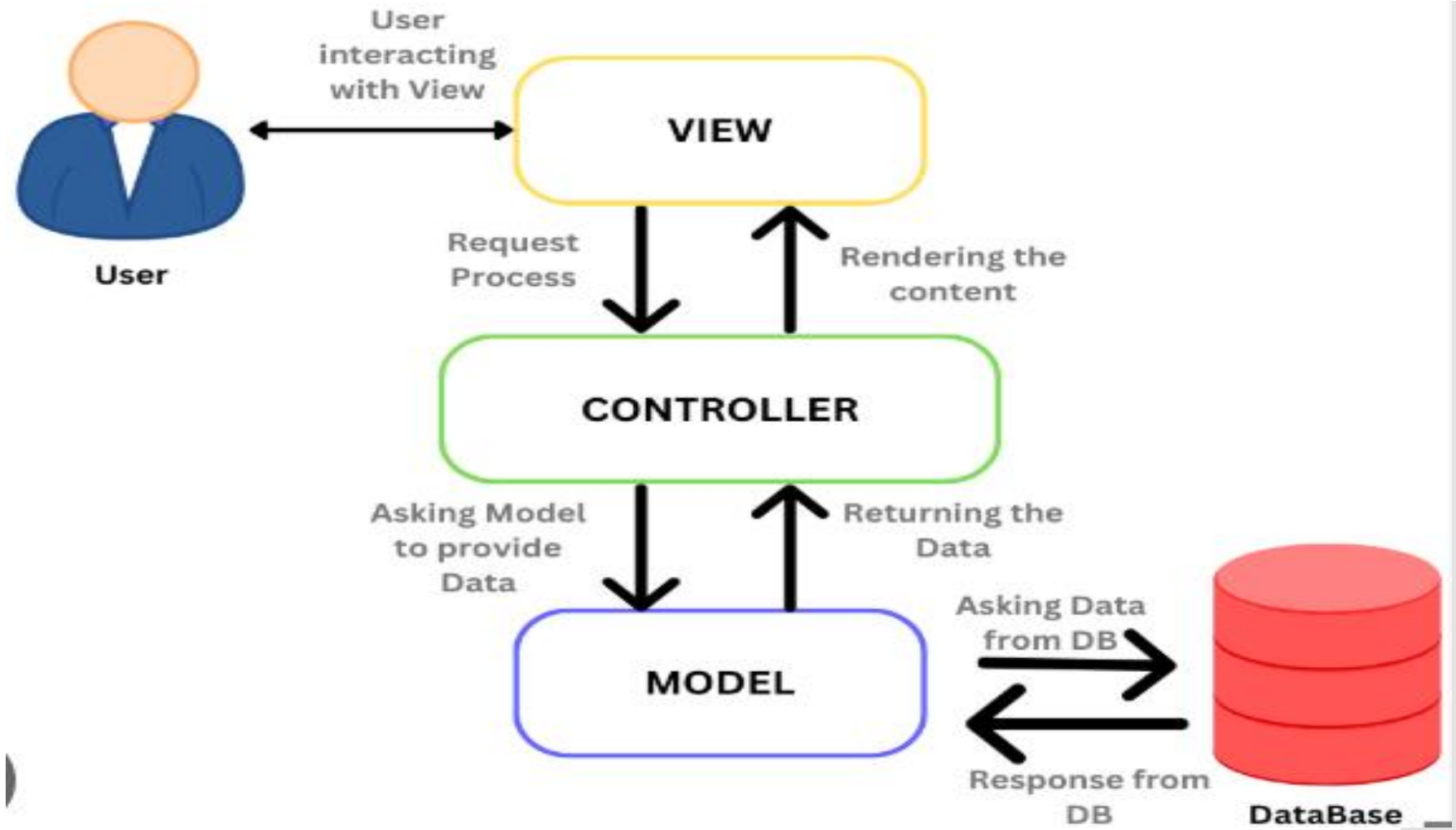
Go (Golang)	Cloud services, CLI tools	Gin, Echo	Fast compilation, great concurrency	Younger ecosystem, less flexible
C#	Windows apps, games	ASP.NET Core	Great for Windows, strong typing	Mostly Windows-centric

1.7. Backend Programming Languages

Language	Used For	Popular Frameworks	Pros & Cons
Python	AI, Web Dev, Data	Django, Flask	✅ Easy to learn ❌ Slower than Java
PHP	WordPress, CMS	Laravel, Symfony	✅ Great for web ❌ Old syntax
Java	Enterprise apps	Spring, Jakarta EE	✅ Very fast ❌ Complex

1.8. MVC Architecture (Model-View-Controller)

- MVC (Model-View-Controller) is a software architectural pattern that separates an application into three interconnected components: the Model, the View, and the Controller. This separation of concerns promotes code maintainability, testability, and scalability


User interacting with View
VIEW
User
Request Process
Rendering the content
CONTROLLER
Asking Model to provide Data
Returning the Data
MODEL
Asking Data from DB
Response from DB
DataBase

- **The Model:**
- Manages the application's data and business logic.
- Includes data storage interaction (e.g., databases) and data validation.
- Ensures data integrity and consistency.
- **The View:**
- Responsible for presenting data to the user.
- Represents the user interface (UI) and how data is displayed.
- Does not contain any business logic.
- **The Controller:**
- Acts as an intermediary between the Model and the View.
- Handles user input, interacts with the Model, and determines which View to display.
- Updates the Model and View accordingly.

Benefits of MVC:

- **Separation of Concerns:**
- Each component has a specific responsibility, making the code easier to understand and maintain.
- **Reusability:**
- Components can be reused in other parts of the application or even in different projects.
- **Testability:**
- Individual components can be tested independently.
- **Maintainability:**
- Changes in one component (e.g., a new feature) do not necessarily affect other components.
- **Scalability:**
- MVC makes it easier to expand the application without significantly altering existing code.
- **Collaboration:**
- MVC allows developers to work on different components simultaneously, reducing code conflicts.

How It Works?

- **Model** → Manages data (e.g., database queries).
- **View** → Displays UI (e.g., HTML templates).
- **Controller** → Handles logic (e.g., user login).
- **Example (User Login):**
- **View:** Shows login form (HTML).
- **Controller:** Checks username/password.
- **Model:** Fetches user data from the database.
- ✓ **Clean code** (easier to maintain).
  ✕ **Extra setup** needed.