Lecture 5

Software Architecture Patterns (MVI, MVVM, MVP, and MVC)

- The study of architecture patterns focuses on ensuring separation of concerns, maintainability, and testability in applications.

Model-View-Controller (MVC)

- The second half of this module focuses particularly on the **Model** component of MVC.

Component	Role
Model	Responsible for managing the data, business logic (e.g., "check stock" or "calculate total"), and rules of the application. It interacts with the database or other data sources.
View	Displays information, often through JSP pages or HTML templates.
Controller	Handles HTTP requests, processes user input, updates the Model, and selects the appropriate View. It manages the **bidirectional relationship** between the user and service provider.
Data Flow	Typically **bidirectional** between Model and View through the Controller.
Use Case	Traditional web applications, such as those built with Spring MVC.

Model-View-Presenter (MVP)

- MVP is a derivative of MVC aimed at improving separation of concerns.

Component	Role	Key Characteristics
Model	Manages the data and business logic.	
View	Handles the display and user interactions.	**Passive View:** It is passive and does not know about the Model.
Presenter	Acts as a **mediator** between Model and View, handling presentation	Holds the state and logic, updating the View accordingly.

	logic.	
Data Flow	Typically **unidirectional** from the Presenter to the View; Presenter acts as the mediator.	
Testability	**High testability**; the Presenter can be tested without the View or Model.	Before MVVM, MVP was widely used in Android to improve testability.

Model-View-ViewModel (MVVM)

- MVVM separates the development of the graphical user interface from the business or back-end logic.

Component	Role	Key Characteristics
Model	Data and business logic.	
View	The UI elements.	Differs from the View in MVC/MVP. View binds to the ViewModel.
ViewModel	An **abstraction of the View**, containing state and logic to manage the View.	Exposes data streams to which the View binds.
Data Flow	**Unidirectional** flow from Model to ViewModel to View.	
Data Binding	Strong data binding support is a major feature (e.g., frameworks like WPF or Angular). The View uses **Data Binding** to the ViewModel.	
Use Case	Rich client applications requiring robust data binding, such as desktop apps or complex UIs (e.g., iOS apps with SwiftUI).	

Model-View-Intent (MVI)

- MVI is a pattern popular in **reactive programming**.

Component	Role	Key Characteristics
Model (State)	Represents the **entire state** of the View. It is the **single source of truth** for the UI.	The Model processes the intent and emits a new state.
View	Renders the UI based on the current state.	Interacts by triggering an Intent.

Intent	User actions or events that request state changes.	Manipulates the Model to update to a new state.
Data Flow	**Unidirectional, cyclical flow**: View → Intent → Model → ViewState → View.	MVI emphasizes unidirectional data flow and **immutable state**.
Complexity	High; requires an understanding of reactive streams and unidirectional data flow.	
Use Case	Applications requiring high scalability and responsiveness, often used in Android development with reactive frameworks (e.g., RxJava) or ReactJS applications following similar unidirectional data flow (like Redux).	

Modern Front-End and JavaScript

- Modern development has seen a shift towards **client-side rendering** using JavaScript frameworks.

JavaScript (JS)

- JavaScript is a **high-level, interpreted programming language** primarily used for creating interactive web pages.
- It runs mainly in the browser (**client-side**) but can also run server-side using Node.js.
- Each browser uses its own **JavaScript Engine** to interpret and compile code into machine language quickly, enabling real-time, dynamic functionality. Examples include **V8** (Chrome), **SpiderMonkey** (Firefox), and **JavaScriptCore/Nitro** (Safari).
- JS runs in a secure environment called a **sandbox**, which limits its access to the computer, ensuring safety without needing additional installations like Flash.

Single-Page Applications (SPAs)

- **Definition:** SPAs are web applications that load a **single HTML page** (and CSS) and dynamically update content without refreshing the entire page.

- **Benefits:** They offer an improved user experience with faster interactions and reduce server load due to fewer HTTP requests.

Modern Front-End Frameworks

These frameworks facilitate the shift towards client-side interactivity and component-based architecture.

- **React.js (Meta/Facebook):** A JavaScript library focusing on building **reusable UI components**. It supports **unidirectional data flow**.
- **Angular (Google):** A complete framework known for two-way data binding.
- **Vue.js:** Lightweight and incrementally adoptable.

React and the Virtual DOM (VDOM)

React improves performance and optimizes updates through the use of the Virtual DOM.

- **Document Object Model (DOM):** The DOM is an interface that treats an HTML or XML document as a **tree structure**, where every part of the document (elements, attributes, text, comments) is represented as a node.
- **Virtual DOM Process:**

  1. When underlying data changes, the entire UI is first re-rendered in the **Virtual DOM representation**.

  2. The difference (the "diff") between the previous VDOM and the new VDOM is calculated.

  3. Finally, the real DOM is updated **only with the specific elements that have actually changed**, which optimizes performance.

Integrating Spring MVC with Modern Front-End

The traditional role of Spring MVC as a server-side renderer has evolved when integrating with modern frameworks like React.

- **Shift in Rendering:** While Spring MVC traditionally rendered views server-side (e.g., JSP), modern frameworks shift the view rendering entirely to the **client-side**.
- **API-Centric Backend:** When integrating with React, the Spring Boot application typically serves as the **backend**, providing data via **REST APIs**.

- **Streamlined Controllers:** The Spring Controller becomes streamlined, often using a **single @RequestMapping** (or `@RestController`) to return product data (e.g., JSON).
- **Client Control:** The client-side (React) uses technologies like `fetch` or `axios` to handle HTTP requests, dynamically requesting data from the backend to update the UI without needing full page reloads (SPA functionality).

This integration model leverages Spring MVC's strength as a robust backend framework while utilizing modern JavaScript frameworks for client-side interactivity and real-time updates.