**Learnings:**

Day 1- MVN

**MVC (Model–View–Controller)**

It’s a **software architectural pattern** used to separate an application into three interconnected parts.  
This separation makes code **organized, reusable, and easy to maintain**.

**1️ Model**

* Represents the **data** and the **business logic** of the application.
* Talks to the **database** or **data source**.
* Contains rules on how data can be created, stored, and changed.

Example: A Student class that fetches data from a database.

**2️ View**

* Represents the **User Interface (UI)** — what the user sees.
* Displays data from the **Model**.
* Should not contain business logic, only **presentation logic**.

Example: An HTML page, JSP file, or frontend component.

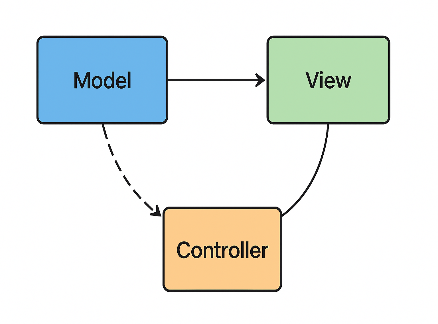
**3️ Controller**

* Acts as a **middleman** between Model and View.
* Handles **user input** (like button clicks, form submissions).
* Updates the Model and tells the View what to show.

Example: A servlet in Java, or a Spring Boot controller.

**Flow of MVC**

1. User interacts with **View** (clicks button, submits form).
2. **Controller** handles input and processes it.
3. **Model** is updated or queried.
4. Data from Model goes back to the **View**.
5. View renders the updated UI.



What is Encapsulation?

* Encapsulation means binding data (fields/variables) and methods (functions) that operate on that data into a single unit (class) and restricting direct access to some components.
* It helps achieve data hiding and ensures controlled access via getters and setters.

What is getter and setter

### ****Getter****

* A method that **returns the value** of a private field.
* Provides **read-only access**.

Example:

public String getName() {

return name;

}

### 2️ ****Setter****

* A method that **updates/sets the value** of a private field.
* Provides **write-only access**.

Example:

public void setName(String name) {

this.name = name;

}