### 🔹 ****Spring MVC Overview (In Simple Words)****

1. **What is Spring MVC?**  
   Spring MVC is a part of the Spring framework. It is used to build **web applications** (like websites or web services).
2. **Spring + Spring Boot = Auto Configuration**
   * When we use **Spring** and **Spring Boot** together, we get something called **Auto Configuration**.
   * Auto Configuration means most of the setup is **done automatically** by Spring Boot.
   * As developers, we don’t need to write a lot of configuration code manually. This saves time and reduces errors.
3. **How Spring MVC helps in Web Application Development**
   * Spring MVC lets you build web apps in two main ways:
     + Using **Servlets** (old style Java web programming).
     + Using **Spring Web MVC** (modern and easier way with Spring features).
   * You can choose the one that fits your project needs.
4. **Spring Version and Web MVC Support**
   * In **Spring 1.x**, we had:
     + **Spring MVC** (early version of web framework).
     + **Spring Web MVC** was used for basic web projects.
   * In **Spring 2.x**, we mainly used **Spring Web MVC** for web applications. It had better features and improvements.

## ✅ **Distributed Application vs Web Application (Simple & Clear Explanation)**

### 🔷 ****1. What is a Distributed Application?****

* A **Distributed Application** is not fully located in one place.
* Different parts of the application run on **different systems or servers**.
* These parts **talk to each other using a network** (like the internet).

Mobile App (C) → API Gateway (PB) → Bank Server (B)

Mobile App = Client

API Gateway = Middle layer (PB)

Bank Server = Business Logic (Backend)

 **PB** actually **represents a real part** in architecture — the **middle layer**.

 That **middle layer** can be:

* A **Proxy** (like an API Gateway, Load Balancer)
* A **Broker** (like a Message Queue)
* A **Service layer** that handles routing or transformation

#### 📌 Example:

Mobile OS → GPay App → Banking Server

* The **OS** runs GPay (Client).
* GPay connects to the **Bank's Server (Business Logic)**.
* The flow is **distributed across different systems**.

#### 🛠 Technologies Used in Distributed Applications:

* ✅ **SOAP (Simple Object Access Protocol)** – XML-based communication, used in older systems
* ✅ **RESTful Services** – More modern, based on HTTP methods (GET, POST, etc.)

#### 🧠 Key Idea:

* Different components are **spread out** and **talk to each other through the internet**.
* The app is **not tightly connected** in one place.

### 🔷 ****2. What is a Web Application?****

* A **Web Application** runs in a **web browser** (like Chrome or Firefox).
* It has a **frontend (Client)** and a **backend (Server)**.
* The **client directly communicates** with the backend using HTTP.

#### 🔁 Flow Diagram:

Client ↔ Backend

* The **Client (C)** sends requests.
* The **Backend (B)** responds directly — there’s **no middle layer like PB**.

✅ So, **C ≡ B** means **direct connection** between client and business logic .

#### 📌 Example:

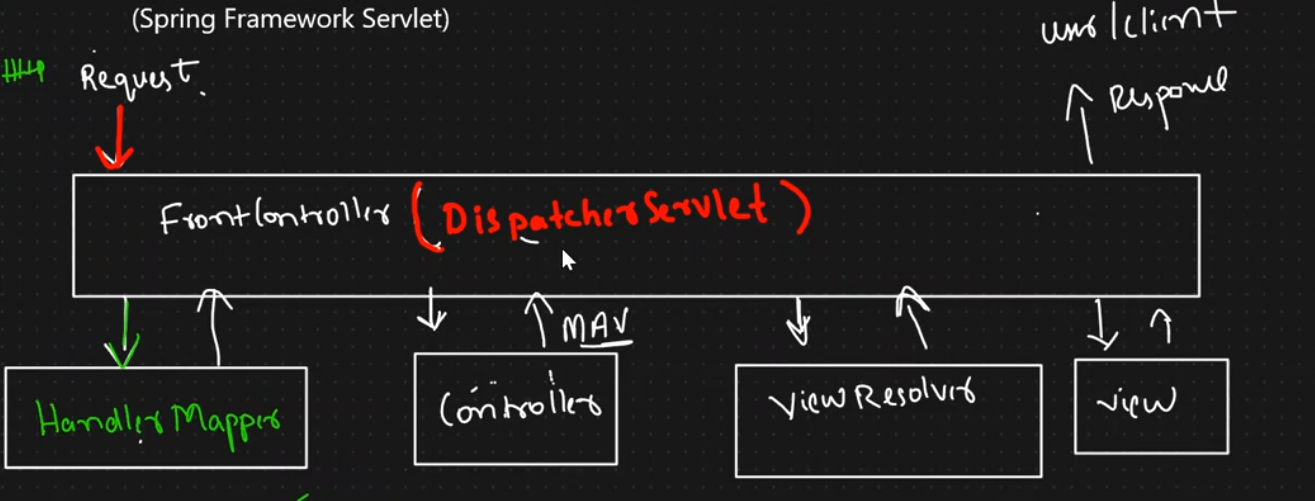
Browser → Online Shopping Website

* The browser sends a request (like viewing a product).
* The server responds with data (like product info).

#### 🛠 Technologies Used:

* **UI/UX** – Designs how the app looks and feels.
* **i18n (Internationalization)** – Supports multiple languages and regions.
* **Spring Web MVC** – Java framework for building web apps using Model-View-Controller pattern.

## ✅ **Spring Web MVC Architecture (Simple Explanation)**



### 🔷 Spring Web MVC is built on two design patterns:

#### 1️⃣ **MVC Design Pattern**

* **MVC** stands for **Model - View - Controller**.
* It helps **separate the code** based on responsibilities:
  + **Model** = The data or logic (e.g., Student info, business rules).
  + **View** = What the user sees (e.g., HTML, JSP, UI screens).
  + **Controller** = Handles user requests and controls the flow (e.g., button click goes to a method).

✅ **Why MVC?**  
It makes the code **cleaner, easier to manage**, and **easy to update without affecting everything else**.

#### 2️⃣ **Front Controller Design Pattern**

* This means **one single controller** (entry point) handles **all the user requests**.
* In Spring, that is the **DispatcherServlet**.
* It acts like a **traffic controller** – it receives every request and decides where to send it.

✅ **Why Front Controller?**  
It gives **centralized control** and **reduces repeated code** in different places.

## 🔶 **Spring MVC Main Components (Explained Simply)**

### 1️⃣ ****Front Controller →**** DispatcherServlet

* This is the **main entry point** of a Spring MVC application.
* It receives **all the incoming requests** (like /login, /saveStudent).
* It decides **which controller method should handle** that request.

✅ Think of it like a **receptionist** who receives every customer and sends them to the right department.

## ✅ **Pre-Processing by Front Controller (DispatcherServlet)**

🔹 Pre-processing means doing tasks **before** the controller handles the request.

### 🔁 What is Pre-Processing?

Pre-processing is all the **initial steps** the Front Controller does **when a request first comes in**, before handing it over to the appropriate controller method.

### ✅ Examples of Pre-Processing (Explained Simply)

### 1️⃣ ****Capturing incoming request data****

* The user might submit a form (like a login form or registration).
* The Front Controller **receives that request** and **reads all the data** (like username, password, etc.).

✅ **Example:**  
When a user submits a form with name and email, the Front Controller **collects that form data** and prepares it for the controller.

### 2️⃣ ****Identifying which controller should handle the request****

* The URL of the request tells the Front Controller where to go.
* It uses a component called **HandlerMapper** to find the right controller method.

✅ **Example:**  
If the user hits /saveStudent, the Front Controller checks:  
"Which controller has a method mapped to /saveStudent?"

### 3️⃣ ****Creating required objects for processing****

* It creates required helper objects, like:
  + HttpServletRequest (to hold request data)
  + HttpServletResponse (to prepare the response)
* These are passed into the controller method for further handling.

✅ So the controller doesn’t need to worry about setting these up — the Front Controller does it **beforehand**.

### 4️⃣ ****Applying filters or interceptors (if any)****

* If there are any filters (like checking login status, security, etc.), they are **executed before** the request goes to the controller.

✅ **Example:**  
If a page requires login, a filter will check login status and block unauthenticated users **before the controller is reached**.

### 🧠 Summary of Pre-Processing (Step-by-Step):

1. Front Controller **receives the request** from the client.
2. It **captures input data** (like form fields, parameters).
3. It **uses HandlerMapper** to find the matching controller method.
4. It **creates and prepares objects** needed for request handling.
5. It **runs filters/interceptors** (if any) to apply additional logic (like login check).
6. Then it **passes the request** to the selected controller method.

## ✅ **Post-Processing by Front Controller (DispatcherServlet)**

🔹 After the request is handled by the controller and the result is ready, the **Front Controller does post-processing** before sending the final output back to the user.

### 🔁 What is Post-Processing?

Post-processing means doing tasks **after** the controller has finished its work and is ready to send a response.

### ✅ Examples of Post-Processing (Explained Simply)

### 1️⃣ ****Sending the final response back to the client****

* Once the controller returns a result (like success message or data), the Front Controller **collects it** and **sends it to the browser or client app**.
* It ensures the response is in a format the client can understand (like HTML, JSON, XML).

✅ **Example:**  
If the controller returns a list of students, the Front Controller makes sure it is **converted into HTML or JSON** and sent to the browser.

### 2️⃣ ****Resolving the view name into an actual view file****

* The controller may return a view name like "studentList".
* The Front Controller uses the **ViewResolver** to find the exact file like studentList.jsp.

✅ Then, it **loads that JSP file**, fills it with the model data (like student names), and **generates the final web page**.

### 3️⃣ ****Rendering the view (final output)****

* After getting the view, it is **processed to create the final content**.
* This may include:
  + Adding dynamic values to HTML
  + Filling form fields
  + Showing messages

✅ Finally, the **rendered page is shown to the user** in the browser.

### 🧠 Summary of Post-Processing (Step-by-Step):

1. Controller returns ModelAndView (data + view name).
2. Front Controller uses **ViewResolver** to find the real view file.
3. The **view is rendered** with model data.
4. The **final output is sent to the browser or client** in a proper format.

## ✅ Understanding DispatcherServlet in Spring Web MVC

### 🔹 Statement 1:

**Spring Web MVC based application → uses DispatcherServlet as a Front Controller**

### ✅ What it means (Simple Explanation):

* In a **Spring Web MVC** application, the **DispatcherServlet** acts as the **central controller**.
* It is responsible for handling **all the incoming web requests**.
* Every request (like clicking a link or submitting a form) first goes to **DispatcherServlet**.
* This is based on the **Front Controller Design Pattern**, where **one servlet controls the entire request flow**.

✅ **Key Point:**  
DispatcherServlet is the **main starting point** in any Spring MVC web application.

### 🔹 Statement 2:

**DispatcherServlet → Servlet provided by Spring MVC module**

### ✅ What it means (Simple Explanation):

* DispatcherServlet is a **special servlet** that comes **built-in** with the Spring MVC framework.
* You don’t need to create it manually — just declare it in web.xml or let **Spring Boot configure it automatically**.
* It is part of the **Spring Web MVC module**, and it knows how to:
  + Read configuration
  + Handle URLs
  + Call the right controller
  + Resolve views
  + Manage pre-processing and post-processing

✅ **Key Point:**  
DispatcherServlet is **not a regular servlet** — it is a **Spring-provided servlet** that adds MVC logic and controls the full request-response cycle.

### 2️⃣ ****Handler Mapper****

* After receiving the request, the DispatcherServlet asks the **Handler Mapper**:  
  "**Which controller and method should handle this URL?**"
* The Handler Mapper matches the **URL pattern to a specific controller method** using annotations like @RequestMapping.

✅ Think of it like a **map** that tells where to go based on the address.

## ✅ **Understanding Handler Mapper in Spring Web MVC**

### 🔹 1. ****Handler Mapper is a Predefined Class in Spring Web MVC****

* Spring Web MVC provides a built-in component called **Handler Mapper**.
* You don’t need to create it manually — Spring configures it for you.

✅ **Key Point:**  
**Handler Mapper** is already available in the Spring framework. It helps connect incoming requests to the correct controller.

### 🔹 2. ****Handler Mapper is used to Identify the Request Handler (Controller)****

* When a request comes in (like /saveStudent), the **DispatcherServlet** asks the **Handler Mapper**:

"Which controller should handle this URL?"

* The Handler Mapper then **searches for the matching controller method** using annotations like @RequestMapping.

✅ **Key Point:**  
Handler Mapper’s job is to **find the correct controller method** for the incoming request.

### 🔹 3. ****Request Handler = Controller****

* In Spring MVC, the term **Request Handler** means the same thing as a **Controller**.
* A Controller is the class that has methods like saveStudent() or showForm() to handle user actions.

✅ **Key Point:**  
Whenever you see “Request Handler,” you can simply think of it as a **Controller**.

### 🔹 4. ****What Happens Internally****

* A user sends a request (e.g., /addStudent)
* DispatcherServlet receives it
* DispatcherServlet asks the **Handler Mapper**
* Handler Mapper looks at all available controllers and **finds the matching one**
* It **returns the details** (which controller and method) back to the DispatcherServlet

✅ **Key Point:**  
Handler Mapper **connects the request URL to the right controller method** and tells the DispatcherServlet what to call next.

### 🧠 Final Summary (Step-by-Step):

1. **User sends a request**
2. **DispatcherServlet** receives the request
3. **Handler Mapper** checks which controller can handle it
4. **Handler Mapper** returns the matching controller method
5. DispatcherServlet forwards the request to that controller

### 3️⃣ ****Controller****

* A **Controller** is a Java class where you write the code to **handle the request**.
* It processes the data, calls the service/business logic, and prepares the response.

✅ It’s like the **main worker** who handles the job and gives back the result.

## ✅ Detailed Explanation of Controller in Spring Web MVC

### 🔷 1. ****What is a Controller?****

* A **Controller** is a **Java class** used in Spring Web MVC to **handle user requests**.
* When a user performs any action in the browser (like clicking a button, submitting a form, or typing a URL), the request is sent to the **DispatcherServlet**.
* The **DispatcherServlet** then passes this request to the appropriate **Controller**.
* The **Controller takes care of processing the request**, preparing data (if needed), and sending a response (like a web page or message).

✅ **Key Point:**  
Controller is the **brain of a Spring Web Application**. It receives requests, processes them, and sends back responses.

### 🔷 2. ****Why is it Called a “Controller”?****

* It **controls the request flow** between the user (client), the business logic (service), and the view (JSP/HTML page).
* It acts like a **middleman** that connects the **client** to the **backend logic**.

✅ **Example in Real Life:**  
Like a receptionist who receives your request, understands it, passes it to the right department, and then gives you the final response.

### 🔷 3. ****What Does a Controller Do?****

A controller performs these 3 main tasks:

1. **Accepts user input (HTTP Request)**  
   → Like /register, /save, /updateStudent, etc.
2. **Processes that input using business logic**  
   → Calls a service or DAO to handle database or business logic.
3. **Returns a response back to the user**  
   → Usually returns a **view name** (like JSP page) or **data** (like JSON).

✅ **Key Point:**  
Controller receives the input, calls service/DAO for logic, and returns output to user.

### 🔷 4. ****How to Create a Controller in Spring MVC?****

* Use @Controller annotation on your class
* Use @RequestMapping or @GetMapping / @PostMapping on methods

✅ **Example:**

@Controller

public class StudentController {

@RequestMapping("/showForm")

public String showForm() {

return "student-form.jsp"; // returns view page

}

@PostMapping("/saveStudent")

public String saveStudent(Student student) {

// process the student data

return "success.jsp";

}

}

* The class is a Controller because of @Controller.
* The method showForm() is a request handler for URL /showForm.

✅ **Key Point:**  
Controller class can have **many methods**, and each method handles a **specific URL/request**.

### 🔷 5. ****What is a Request Handler?****

* A **Request Handler** is a method inside a Controller.
* It handles a specific type of request (GET, POST, etc.)
* In Spring MVC, these methods are annotated with @RequestMapping, @GetMapping, or @PostMapping.

✅ **Key Point:**  
**Controller = Class**,  
**Request Handler = Method** that handles a specific request.

### 🔷 6. ****Types of Controllers****

There are 2 types of Controllers in Spring:

#### a) **Predefined Controllers (Old Style)**

* Classes like SimpleFormController, AbstractController, MultiActionController
* Used in **Spring 1.x and 2.x** (older versions)
* Required a lot of **XML configuration**
* Now **mostly outdated**

✅ **Key Point:**  
Old controllers required extending Spring’s classes and writing XML — not preferred today.

#### b) **Annotation-based Controllers (Modern Style)**

* Use @Controller annotation
* Add methods with @RequestMapping, @GetMapping, etc.
* Introduced in **Spring 2.5+**
* **Much easier and cleaner** than the old approach

✅ **Key Point:**  
**Modern Spring MVC uses annotation-based Controllers** — they are simple, flexible, and easy to maintain.

### 🔷 7. ****Where Does the Controller Fit in the MVC Pattern?****

In the MVC (Model-View-Controller) design:

* **Model** = Data (like Student object)
* **View** = UI (like JSP page)
* **Controller** = Connects the two, controls the flow

✅ **Key Point:**  
Controller takes data from the user, sends it to the model, and then returns the view.

### 4️⃣ ****ModelAndView****

* The controller usually returns a **ModelAndView** object.
* **Model** = Holds the data to show in the view (like a list of students).
* **View** = The name of the page to display (like studentList.jsp).

✅ It’s like a **package** that contains:

* What to show (data)
* Where to show it (view page)

## ✅ What is ModelAndView in Spring Web MVC?

### 🔷 1. ****What is**** ModelAndView****?****

* ModelAndView is a **predefined class** in Spring Web MVC.
* It is used by the **Controller** to return **both data (Model)** and **view name (View)** together.
* This object is given back to the **DispatcherServlet** after the request is processed.

✅ **Key Point:**  
ModelAndView = **Model (data)** + **View (page name)**

### 🔷 2. ****Where is**** ModelAndView ****used?****

* Inside the **Controller**, after processing a request (like saving form data), you need to:
  + Pass some data to the next page
  + Also tell **which page** (JSP, HTML) should be shown next
* You can use ModelAndView to do **both in one step**

✅ **Key Point:**  
Controller uses ModelAndView to send **data** and **page name** back to DispatcherServlet.

### 🔷 3. ****What does the “Model” mean?****

* **Model** means the **data** you want to send to the view (like JSP).
* It can be anything: a string, number, list, object (like Student, Employee).
* This data will be displayed on the webpage.

✅ **Example:**  
Sending a message like "Registered Successfully" or a Student object to the JSP page.

### 🔷 4. ****What does the “View” mean?****

* **View** means the **name of the web page** you want to show to the user.
* Usually, this is a **logical view name** like "success", "result", "studentDetails".
* Spring will use this name to find the actual JSP or HTML file using ViewResolver.

✅ **Example:**  
If view name is "success", Spring may render success.jsp.

### 🔷 5. ****How to Create a**** ModelAndView ****Object?****

✅ **Example Code:**

ModelAndView mv = new ModelAndView();

mv.setViewName("success"); // view = success.jsp

mv.addObject("msg", "Registration Done"); // model = data to show

return mv;

Or in one step:

return new ModelAndView("success", "msg", "Registration Done");

✅ **Key Point:**  
You can **add data** with addObject(key, value)  
And set the **view name** with setViewName("pageName")

### 🔷 6. ****What Happens After Controller Returns ModelAndView?****

1. **Controller** returns ModelAndView to **DispatcherServlet**
2. DispatcherServlet:
   * Takes the **model (data)**
   * Uses **ViewResolver** to find the actual **view (JSP page)**
3. DispatcherServlet sends the **data to the view**
4. Final **output is shown** to the user

✅ **Key Point:**  
ModelAndView helps pass both **data** and **page name** from controller to the frontend.

### 5️⃣ ****ViewResolver****

* This component **takes the view name** from ModelAndView and **finds the actual view file** (like a JSP file).
* Example: If the controller returns studentList, the ViewResolver may convert it to WEB-INF/views/studentList.jsp.

✅ It works like a **translator** that finds the real path of the view page.

## ✅ What is ViewResolver in Spring Web MVC?

### 🔷 1. ****What is ViewResolver?****

* **ViewResolver** is a **predefined component** in Spring Web MVC.
* It helps the application **find the correct view file** (like JSP, HTML).
* It maps a **logical view name** (given by the Controller) to the **actual view file** (like success.jsp).

✅ **Key Point:**  
**Controller gives just a name**, and **ViewResolver finds the actual page.**

### 🔷 2. ****Why Do We Need ViewResolver?****

* In your controller, you don’t need to write full file names like "/WEB-INF/views/success.jsp".
* You just return a simple name like "success".
* **ViewResolver will automatically add the path and file extension.**

✅ **Example:**  
If controller returns "success"  
➡️ ViewResolver will convert it to /WEB-INF/views/success.jsp

### 🔷 3. ****What Does ViewResolver Do Exactly?****

* It **adds prefix (folder path)** and **suffix (file extension)** to the view name.
* Then it checks if that file exists and sends it to the DispatcherServlet.

✅ **Example Configuration (in XML):**

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/views/" />

<property name="suffix" value=".jsp" />

</bean>

* This means:
  + "success" → /WEB-INF/views/success.jsp
  + "home" → /WEB-INF/views/home.jsp

## ✅ ViewResolver in Annotation-Based Approach

### 🔷 1. ****Is there an annotation for ViewResolver?****

* ❌ There is **no direct annotation like @ViewResolver**.
* ✅ But in annotation-based approach, we use **Java Configuration Class** (with @Configuration) to define and register a **ViewResolver bean**.

### 🔷 2. ****How do we configure ViewResolver without XML?****

* In annotation-based config, we **write a Java class** instead of an XML file.
* We **use @Configuration and @Bean annotations** to declare the InternalResourceViewResolver.

✅ Example:

@Configuration

public class WebMvcConfig implements WebMvcConfigurer {

@Bean

public InternalResourceViewResolver viewResolver() {

InternalResourceViewResolver resolver = new InternalResourceViewResolver();

resolver.setPrefix("/WEB-INF/views/");

resolver.setSuffix(".jsp");

return resolver;

}

}

### 🔷 3. ****Simple Explanation of the Code Above:****

| **Line** | **Meaning** |
| --- | --- |
| @Configuration | Tells Spring: This class contains configuration code |
| public class WebMvcConfig | Java class to hold view resolver and other web-related setup |
| @Bean | Tells Spring: Create a bean for this method’s return type |
| InternalResourceViewResolver | This object helps in locating JSP files |
| setPrefix("/WEB-INF/views/") | Views are inside this folder |
| setSuffix(".jsp") | All views are JSP files (like home.jsp, welcome.jsp) |

### 🔷 4. ****Where Do We Put This Configuration Class?****

* Place this class in a package that gets **scanned** by Spring.
* If using Spring Boot, it gets picked up automatically.

### 🔷 Final Summary:

| **Topic** | **Details** |
| --- | --- |
| **ViewResolver with XML** | Used in older, XML-based Spring MVC projects |
| **ViewResolver with Java** | ✅ Used in real-time, annotation-based Spring MVC |
| Direct Annotation? | ❌ No @ViewResolver — we use @Bean inside @Configuration class |
| Advantage | No need for XML files; configuration is in pure Java and easier to manage |

### 🔷 4. ****Types of ViewResolvers in Spring****

There are many built-in ViewResolver classes in Spring:

### 🔷 Commonly Used ViewResolver Classes in Spring MVC

| **ViewResolver Class** | **Description** | **Real-Time Usage** |
| --- | --- | --- |
| **InternalResourceViewResolver** | ✅ Most commonly used in real-time projects. It resolves **JSP files** located inside the project using prefix and suffix. | ✅ **Used the most** |
| **XmlViewResolver** | Takes view names and maps them to actual views using an **external XML file**. | ❌ **Rarely used** in modern apps |
| **UrlBasedViewResolver** | Supports **multiple view types** (JSP, PDF, Excel, etc.) using URL patterns. | 🔶 Used in **advanced or multi-view** apps |
| **BeanNameViewResolver** | Resolves view by **matching bean name** in the config file or code. | ❌ **Rarely used** |

✅ **Key Point:**  
We usually use **InternalResourceViewResolver** in basic web apps (JSP-based).

### 🔷 5. ****When Does ViewResolver Get Involved?****

1. **Controller** returns ModelAndView with a **logical view name**.
2. **DispatcherServlet** gives that name to the **ViewResolver**.
3. ViewResolver finds the **actual JSP/HTML file** using prefix and suffix.
4. DispatcherServlet renders that page to the **user**.

✅ **Key Point:**  
ViewResolver works **after the controller finishes** its job, to show the correct page.

### 6️⃣ ****View****

* This is the **final HTML or JSP page** shown to the user.
* It uses the **model data** to display dynamic content.

✅ It’s like the **result screen** that the user sees in the browser.

## 🔷 **View (in Spring Web MVC)**

### ✅ What is a View?

* A **View** is the final output page that the **user sees** in the browser.
* It shows the **data returned by the controller**, in a format that is easy to understand.
* It is the **presentation layer** in the **MVC design pattern**.

### ✅ Where does the View come in?

| **Flow Step** | **Explanation** |
| --- | --- |
| Controller processes data | After business logic, Controller returns data |
| Controller returns view name | Controller gives a **logical view name** like "welcome" |
| ViewResolver steps in | It finds the actual **view file** (like welcome.jsp) |
| View is rendered | Final output is shown to the user |

### ✅ Technologies used as View in Spring MVC:

| **View Type** | **Description** | **Real-Time Use** |
| --- | --- | --- |
| **JSP** | ✅ Most commonly used in Spring Web MVC | ⭐ Yes |
| Thymeleaf | Used in Spring Boot projects, supports HTML5 | 🔶 Sometimes |
| PDF View | Used for generating downloadable PDFs | 🔶 Special cases |
| Excel View | Used for exporting Excel files | 🔶 Special cases |
| FreeMarker | Template engine alternative to JSP | ❌ Rarely used |

### ✅ Example of a View File (JSP):

<!-- File: /WEB-INF/views/welcome.jsp -->

<html>

<body>

<h2>Welcome, ${userName}!</h2>

</body>

</html>

| **Part** | **Explanation** |
| --- | --- |
| ${userName} | Dynamic data passed from Controller (Model) |
| welcome.jsp | File shown to user when view name is "welcome" |

### ✅ How is data sent to the View?

* The **Controller** uses a Model or ModelAndView to **send data**.
* This data is accessed using **${}** syntax in JSP.

### ✅ Folder structure (common in real-time projects):

/WEB-INF/

└── views/

└── welcome.jsp

└── dashboard.jsp

| **Folder/File** | **Purpose** |
| --- | --- |
| /WEB-INF/views/ | Stores all JSP view files |
| welcome.jsp | View file matched to logical view name "welcome" |

### ✅ Final Summary:

| **Concept** | **Description** |
| --- | --- |
| View | Final output shown to user |
| View file | JSP, Thymeleaf, etc. |
| Comes from | Controller returns logical name → ViewResolver → View file |
| Most used | ✅ JSP (with InternalResourceViewResolver) |
| View format | Uses HTML + dynamic data (${}) from controller |

## 🔁 Full Flow Summary (Step-by-Step):

1. **User sends a request** (e.g., clicking on a link or submitting a form).
2. **DispatcherServlet** receives the request.
3. **Handler Mapper** finds the correct controller method.
4. **Controller** processes the request and returns **ModelAndView**.
5. **ViewResolver** finds the correct view file.
6. **View** is rendered and shown to the user with the data.