Spring MVC without SpringBoot

Introduction to Spring MVC (Without Spring Boot)

- This module focuses on **Spring MVC** using the **Spring Framework directly**, without relying on **Spring Boot**.
- Previously, **Spring Boot MVC** was used to simplify development through auto-configuration and embedded servers.
- In this section, the same MVC flow is followed, but with **manual configuration**.
- The goal is to understand what Spring Boot abstracts away by handling configuration automatically.

Recap: What Was Achieved Using Spring Boot MVC

- Built a working **web application** using:
  - Controllers
  - JSP-based UI
  - Property files
  - Configuration handled by Spring Boot
- Application runs successfully when accessed through the browser (home page loads correctly).
- Used **embedded Tomcat**, which required no external server setup.

Why Learn Spring MVC Without Spring Boot

- Spring Boot simplifies development, but it hides many underlying configurations.
- Spring MVC (without Boot) requires:
  - Explicit configuration
  - Manual setup of components
- Learning this approach helps in understanding:
  - Core Spring MVC architecture
  - How controllers, views, and configuration work internally
- No change in application flow:
  - Controllers remain the same
  - Views remain the same
  - Only **configuration changes**

Key Difference: Embedded vs External Tomcat

- **Spring Boot**:
  - Uses an **embedded Tomcat server**
  - No need to install or configure Tomcat separately
- **Spring Framework (Spring MVC)**:
  - Requires an **external Tomcat server**
  - Application is deployed to the server manually

Requirement 1: External Tomcat Setup

- External Tomcat is mandatory when not using Spring Boot.
- Steps to obtain Tomcat:
  - Search for **Apache Tomcat download** on Google
  - Download **Tomcat version 10** (used in this module)
  - Choose the version based on the operating system
  - Download the ZIP file and extract it
- After extraction, Tomcat is available as a standalone server (e.g., in the Downloads folder).

Tomcat Version Considerations (Java EE vs Jakarta EE)

- **Tomcat 10** introduces a major change:
  - Migration from **Java EE** to **Jakarta EE**
- Key impact:
  - Package names change from javax.* to jakarta.*
- Options available:
  - Use **Tomcat 10** → must use jakarta packages
  - Use **Tomcat 9** → continues to support javax packages
- Either option is acceptable for learning:
  - Choice depends on whether Jakarta EE compatibility is desired

---

Requirement 2: IDE Selection

- **IntelliJ IDEA Community Edition**:
  - Does **not support external servers** like Tomcat by default
  - External Tomcat requires **IntelliJ Ultimate**
- **Eclipse IDE**:
  - Free and open-source

    - Fully supports **external Tomcat integration**
- Decision made in this module:
    - Use **Eclipse** to keep the setup free and accessible

Development Approach in This Module

- Reuse the **same project structure and code logic** where possible
- Focus areas:
    - Manual configuration
    - External server deployment
- Non-focus areas:
    - Controller logic (already covered)
    - MVC flow (already understood)

Summary

- Spring MVC without Spring Boot requires:
    - External Tomcat server
    - Manual configuration
    - IDE support for server deployment
- Core MVC concepts remain unchanged:
    - Controllers
    - Views
    - Request handling
- Primary learning objective:
    - Understand how Spring MVC works **without auto-configuration**
    - Gain clarity on what Spring Boot simplifies

Setting Up Spring MVC Project Without Spring Boot (Eclipse + Maven)

Choosing the IDE (Eclipse)

- Eclipse is used instead of IntelliJ because:
    - Eclipse (free) supports **external servers** like Tomcat.
    - IntelliJ Community Edition does **not** support external servers.
- Required Eclipse version:
    - **Eclipse IDE for Enterprise Java and Web Developers** (Java EE)

- Not the standard “Eclipse for Java Developers”.

Workspace Setup

- On launching Eclipse, select a workspace.
- Example workspace name used: **spring**.
- Workspace name can be anything; it does not affect functionality.

Creating a Maven Web Project (Without Spring Boot)

- Since Spring Boot is not used, **Spring Initializr is not used**.
- Steps to create project:
    1. Go to **File → New → Maven Project**
    2. Click **Next**
    3. Select **Internal** catalog
    4. Choose **maven-archetype-webapp** (Web Application archetype)
    5. Click **Next**
    6. Provide project details:
        - **Group ID:** com.telusko
        - **Artifact ID:** spring-mvc-demo
    7. Click **Finish**
- Result:
    - A Maven-based **Spring MVC web application structure** is created.

External Apache Tomcat Setup

- Spring MVC (without Boot) uses an **external Tomcat server**.
- Apache Tomcat version used:
    - **Apache Tomcat 10.1.16**
- Key Tomcat directory:
    - **bin/** folder
        - Contains startup and shutdown scripts.
- Important note:
    - Tomcat 10 uses **Jakarta EE** instead of Java EE.
    - Package names change from javax.* to jakarta.*.
    - If Javax is preferred, Tomcat 9 can be used instead.

Default Project Structure Analysis

- Generated structure includes:

        - src/main/webapp
            - Contains default index.jsp
        - No src/main/java folder initially

Cleanup and Folder Preparation

- Delete default index.jsp (not required).
- Manually create:
    - src/main/java folder
    - This folder will hold all Java source code.

Copying Code from Spring Boot Project

- Reused components from existing Spring Boot project:
    - Alien.java
    - HomeController.java
- Main (Spring Boot starter) class is **not copied**.

Creating Package Structure

- Create package inside src/main/java:
    - com.telusko.springmvcdemo
- Ensure package name matches across all Java files.

Adding JSP Views

- JSP files are placed inside:
    - src/main/webapp/views
- Copied entire views folder from Spring Boot project.
- CSS files are ignored for now to avoid extra configuration.
- Focus remains on backend and Spring MVC setup.

Understanding Initial Errors

- Errors appear after copying Java files because:
    - Spring MVC dependencies are missing.
    - Jakarta / Servlet APIs are not yet configured.
- Servlet-related imports not immediately required can be removed.

Adding Required Maven Dependencies

1. Spring Web MVC Dependency

- Required dependency:
  - **spring-webmvc**
- Steps:
  - Search for **Spring Web MVC** on Maven Repository.
  - Use version **6.1.0** (or a commonly used recent version).
  - Add dependency to pom.xml.

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>6.1.0</version>
</dependency>
```

- After adding:
  - Maven downloads all required Spring libraries.
  - Errors in HomeController are resolved.

Current Project Status

- Completed so far:
  - Maven Web Project created
  - External Tomcat identified
  - Java source folder created
  - Controller and model classes copied
  - JSP views added
  - Spring MVC dependency configured
- Pending:
  - Configure **Tomcat server inside Eclipse**
  - Deploy and run the application

Setting Up Apache Tomcat in Eclipse and Connecting a Spring MVC Project

Overview

This section explains how to configure **Apache Tomcat** in **Eclipse IDE** and associate an existing **Spring MVC (non-Spring Boot)** project with the server. It also highlights a common runtime issue encountered after setup.

Accessing the Servers View in Eclipse

- Eclipse provides a **Servers** tab used to manage application servers.
- If no server is configured, Eclipse displays:
  - **"No servers are available"**
- Clicking this prompt initiates the server configuration wizard.

Adding an Apache Tomcat Server

- From the server selection list:
  - Choose **Apache**
  - Select **Tomcat 10.1**
- Click **Next** to proceed.

Configuring the Tomcat Installation Path

- Eclipse requires the local **Tomcat installation directory**.
- Two options are available:
  - **Download and install** Tomcat directly from Eclipse
  - **Browse** to an existing Tomcat installation
- In this case:
  - A pre-downloaded Tomcat directory is selected (from the Downloads folder).
- After selecting the path, click **Next**.

Deploying the Project to Tomcat

- Eclipse prompts whether to add projects to the server.
- To ensure the application runs:
  - Move the **Spring MVC project** to the configured Tomcat server
- Click **Finish**
- Result:
  - Tomcat server is added in a **Stopped** state

- The Spring MVC project appears under the server node

Starting the Tomcat Server

- Attempting to start Tomcat may show:
  - **Port already in use**
- Cause:
  - Another server (e.g., embedded Tomcat from IntelliJ) is already running on the same port
- Resolution:
  - Restart Tomcat from Eclipse using **Restart / Relaunch**
- After restart:
  - Tomcat starts successfully

Running the Project on Tomcat

- Steps:
  - Right-click the project
  - Select **Run As → Run on Server**
  - Choose the configured Tomcat server
  - Click **Next → Finish**
- Eclipse automatically opens a browser window

Application Access Attempt

- The application is accessed using:
  - http://localhost:8080
- Observed behavior:
  - Browser displays **“Resource not available”**
- Outcome:
  - Application does not load successfully

Conclusion and Next Step

- Tomcat is:
  - Properly configured
  - Successfully started
  - Correctly linked to the project
- However:
  - The application does not respond as expected

- The reason for this issue and its resolution are addressed in the **next section/video**, focusing on missing or incorrect configuration.

Title: Spring MVC Maven Project Setup and Initial Issue

- A **Spring MVC project** was created as a **Maven project**, not a Spring Boot project.
- When the project was run, the application did **not work** even though:
  - A controller existed.
  - A mapping for the home page was defined.
  - The controller attempted to return index.jsp.
- The browser returned **404 – Not Found**.
- Root cause identified:
  - This is **not a Spring Boot project**.
  - Adding Spring dependencies alone does not make Tomcat aware of Spring controllers.

Title: Role of Tomcat in a Non-Spring Boot Project

- The application is deployed on **Tomcat**, which acts as a **Servlet Container**.
- Tomcat is responsible for executing **servlets**, not Spring controllers directly.
- For Tomcat to process requests correctly:
  - There must be a **mapping between a request and a servlet**.
- Although a Spring controller exists:
  - Tomcat has **no knowledge** of it without proper servlet configuration.

Title: Need for a Front Controller in Spring MVC

- In Spring Framework:
  - Multiple controllers can exist.
  - Each controller handles different requests.
- Client requests **do not go directly** to controllers.
- Instead, requests are first sent to a **Front Controller**.
- Responsibilities of the Front Controller:
  - Receive all incoming requests.

- Identify which controller should handle a specific request.

Title: DispatcherServlet as the Front Controller

- In Spring MVC, the Front Controller is called **DispatcherServlet**.
- DispatcherServlet:
  - Is the **first servlet** that receives client requests.
  - Routes requests to the appropriate controller based on mappings.
- Developers **do not create** DispatcherServlet manually.
- Spring provides it by default.
- Developers must **configure** it so Tomcat knows about it.

Title: Configuring DispatcherServlet Using web.xml

- Tomcat always looks for **web.xml** when starting a web application.
- web.xml is used to:
  - Define servlets.
  - Map URL patterns to servlets.
- Two required tags:
  - <servlet>
  - <servlet-mapping>

Title: Servlet Definition in web.xml

- The <servlet> tag specifies:
  - A custom **servlet name**.
  - The fully qualified class name of **DispatcherServlet**.
- DispatcherServlet class is found in:
  - Maven dependency: **spring-webmvc**
  - Package: org.springframework.web.servlet.DispatcherServlet
- Example configuration:

```
<servlet>
  <servlet-name>springmvcdemo</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
</servlet>
```

Title: Servlet Mapping for All Requests

- The <servlet-mapping> tag connects:
  - A URL pattern
  - To the defined servlet name
- Mapping / means:
  - **All requests** are routed to DispatcherServlet.
- Servlet name in <servlet> and <servlet-mapping> must be the **same**.
- Example:

```
<servlet-mapping>
  <servlet-name>springmvcdemo</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

Title: Result After DispatcherServlet Configuration

- After configuring web.xml:
  - Tomcat successfully started.
  - 404 error was resolved.
- Application now returned a **500 – Internal Server Error**.
- This indicated:
  - DispatcherServlet was found and invoked.
  - A new configuration issue existed.

Title: Build Path and Runtime Configuration Issues

- Additional setup was required in the IDE:
  - Java version needed to be updated to **Java 21**.
  - Tomcat runtime library needed to be added.
- Steps performed:
  - Right-click project → Build Path → Configure Build Path.
  - Update **JRE System Library** to Java 21.
  - Add **Server Runtime** → Apache Tomcat.
- These steps are required in **normal Spring MVC projects**.
- This configuration is typically done **only once per project**.

Title: Understanding the 500 Internal Server Error

- The 500 error indicated:

        - DispatcherServlet was executing.
        - DispatcherServlet could not complete request processing.
    - Root cause from stack trace:
        - java.io.FileNotFoundException
        - Missing file: /WEB-INF/springmvcdemo-servlet.xml

Title: Purpose of springmvcdemo-servlet.xml

- DispatcherServlet expects a configuration file named:
    - <servlet-name>-servlet.xml
- In this case:
    - Servlet name: springmvcdemo
    - Expected file: springmvcdemo-servlet.xml
- This file is required to:
    - Configure DispatcherServlet.
    - Enable it to detect controllers and request mappings.
- Without this file:
    - DispatcherServlet cannot identify controllers.

Title: Controller Implementation Overview

- The controller is annotated with **@Controller**.
- Handles multiple request mappings using **@RequestMapping**.

Home Page Mapping

```
@RequestMapping("/")
public String homePage() {
   return "index";
}
```

- Returns logical view name index.

Title: Calculator Request Mappings

- Addition:

```
@RequestMapping("add")
```

- Subtraction:

@RequestMapping("subtract")

- Multiplication:

@RequestMapping("multiply")

- Division:

@RequestMapping("divide")

- Uses **ModelAndView** to:
  - Pass input values.
  - Pass result.
  - Handle errors.

Title: Alien Form Mappings

- Alien home page:

@RequestMapping("/alien")

- Alien submission:

@RequestMapping("addAlien")

- Uses **@ModelAttribute** to bind request data.

Title: Global Model Attribute Usage

- A common attribute is added using:

```
@ModelAttribute("course")
public String courseName() {
   return "Java...";
}
```

- This attribute is available across views.

Title: Maven Dependencies Used

- **JUnit** for testing.
- **Tomcat Jasper** for JSP compilation.
- **JSTL API** for JSP tag support.
- **Spring Web MVC** dependency:

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>7.0.3</version>
</dependency>
```

Title: Key Takeaway

- In a **non-Spring Boot Spring MVC project**:
  - DispatcherServlet must be explicitly configured.
  - web.xml is mandatory.
  - <servlet-name>-servlet.xml configuration file is required.
- Missing any of these leads to:
  - 404 errors or
  - 500 internal server errors.

Configuring the DispatcherServlet Using XML (springmvcdemo-servlet.xml)

- The **DispatcherServlet** must be explicitly configured when using **Spring MVC without Spring Boot**.
- Configuration is performed using an XML file instead of Java-based annotation configuration.
- XML configuration is used to understand what happens behind the scenes.

Location and Naming of the Dispatcher Servlet Configuration File

- The configuration file must be placed inside the **WEB-INF** directory.
- The file name follows a strict convention:
  - <servlet-name>-servlet.xml
- Since the servlet name in web.xml is **springmvcdemo**, the file name must be:
  - springmvcdemo-servlet.xml
- If a different servlet name were used (e.g., t1, t2), the file name would change accordingly.

Purpose of springmvcdemo-servlet.xml

- This file is used to configure how the **DispatcherServlet** works.
- It defines how Spring should:

        - Locate controllers
        - Detect annotations
        - Process incoming requests

Avoiding Manual Controller Mappings

- Controllers already use annotations such as:
    - @Controller
    - @RequestMapping
- Because annotations are used:
    - Manual request-to-method mappings in XML are not required
- Instead, the DispatcherServlet must be informed that:
    - Controllers exist in a specific package
    - Annotation-based configuration is being used

Bean and Namespace Definitions

- XML tags must be declared and defined before use.
- Required namespaces include:
    - beans
    - context (ctx)
    - mvc
- These namespaces allow the use of tags such as component scanning and annotation-driven MVC.

Enabling Component Scanning

- Component scanning tells Spring where to find controllers and other components.
- Configuration:

```
<ctx:component-scan base-package="com.springmvcdemo" />
```

- This instructs Spring to scan the specified package for annotated components.

Enabling Annotation-Based MVC Configuration

- Annotation-driven configuration enables support for annotations such as:
    - @RequestMapping
    - @Controller
- Configuration:

```
<mvc:annotation-driven />
```

- This allows the DispatcherServlet to process annotation-based mappings.

Result After DispatcherServlet Configuration

- The DispatcherServlet successfully:
  - Locates the configuration file
  - Scans the specified package
  - Detects annotated controllers
- Console output confirms that controller methods are invoked.

Remaining Issue: View Resolution Error

- Although the controller method is executed:
  - The application still returns **404 – No endpoint found** for the view name
- The controller returns logical view names such as index.
- No configuration exists to map logical view names to actual JSP files.

Root Cause of the View Error

- Unlike Spring Boot:
  - No default view resolver is auto-configured
- The application does not know:
  - Where JSP files are located
  - What file extension to use

Next Required Configuration

- The **Internal Resource View Resolver** must be configured.
- This resolver maps logical view names to physical JSP files.
- View resolver configuration will be addressed in the next step.

Configuring the Internal Resource View Resolver in Spring MVC

Problem: View Name Cannot Be Resolved

- Controller methods return logical view names such as index
- Spring MVC does **not know**:
  - Where the view files are located
  - What file extension to use (e.g., .jsp)
- This results in **404 errors** even though the controller method is

executed

Where View Resolver Configuration Belongs

- All communication with **DispatcherServlet** happens in:
  - springmvcdemo-servlet.xml (servlet-name + -servlet.xml)
- This XML file is located under:
  - WEB-INF/

InternalResourceViewResolver

- Spring MVC uses **InternalResourceViewResolver** to:
  - Map logical view names to actual JSP files
- Example:
  - Returned view name: index
  - Resolved path: /views/index.jsp

Finding the View Resolver Class

- Navigate through Maven dependencies:
  - org.springframework.web.servlet
  - org.springframework.web.servlet.view
  - InternalResourceViewResolver.class
- Copy the **fully qualified class name**
- Use it **without .class extension** in XML

Configuring the View Resolver Bean

Required Configuration

- Must define a <bean> in springmvcdemo-servlet.xml
- Two mandatory properties:
  - prefix
  - suffix

Configured Bean

```
<bean
class="org.springframework.web.servlet.view.InternalResourceVie
wResolver">
```

```
    <property name="prefix" value="/views/" />
    <property name="suffix" value=".jsp" />
</bean>
```

Meaning

- Spring will resolve:
  - index → /views/index.jsp
  - result → /views/result.jsp

Why Errors Occurred Initially

- Properties were incorrectly written
- Correct approach:
  - Use value="..." attribute
  - Not inline text or brackets
- After fixing:
  - Application successfully loads the JSP

Successful Page Rendering

- Application reload shows:
  - Home page loads correctly
  - Form submission works
- Controller logic executes correctly

Issue: JSP Output Not Displaying Model Data

Observed Problem

- Output page loads
- Static text appears
- Model attributes (e.g., Alien data, course name) do **not render**

Expected Data

- Alien object passed via @ModelAttribute
- Global model attribute:

```
@ModelAttribute("course")
public String courseName() {
    return "Java...";
}
```

Root Cause: JSP Expression Language Ignored

- JSP Expression Language (EL) was **ignored**
- This prevents ${} expressions from being evaluated

Fix: Enable Expression Language in JSP

- Add the following attribute at the top of the JSP:

isELIgnored="false"

Effect

- Enables evaluation of:
  - ${alien.name}
  - ${course}
- Model data renders correctly

Final Result

- Application fully functional:
  - Controllers mapped
  - DispatcherServlet configured
  - Views resolved correctly
  - Model data displayed in JSP
- Project successfully converted from:
  - **Spring Boot MVC**
  - To **Spring MVC without Spring Boot**

Static Resource Handling (Mentioned Configuration)

<mvc:resources mapping="/css/**" location="/css/" />

- Enables access to static resources
- Not deeply covered in this transcript

Key Takeaways

- Spring MVC (without Spring Boot) requires **manual configuration**
- Essential components:
    1. web.xml → DispatcherServlet mapping
    2. *-servlet.xml → Controller scanning + MVC config
    3. InternalResourceViewResolver → View resolution
    4. JSP EL configuration → Proper data rendering
- Spring Boot simplifies all of the above via auto-configuration

---

Recap: Building a Spring MVC Application Without Spring Boot

1. Project Creation and Dependency Setup

- The project is created as a **Maven project**
- This is a **Spring MVC project**, **not Spring Boot**
- Required dependency:
    - **Spring Web MVC**
    - Version used: **6.1**
- Version can vary depending on when the project is built

---

2. Creating the Controller Layer

- A controller class (e.g., **HomeController**) is created
- Multiple handler methods exist inside the controller
- **Annotations** are used:
    - @Controller
    - @RequestMapping
- These annotations define request mappings directly in the class

---

3. Why the Application Does Not Work Initially

- The project runs on **Apache Tomcat**
- Tomcat is a **Servlet Container**
- By default, Tomcat:
    - Does **not know** this is a Spring MVC project

        - Does **not know** about Spring controllers

---

4. Role of DispatcherServlet (Front Controller)

- Spring MVC uses a **Front Controller pattern**
- The front controller in Spring is:
    - **DispatcherServlet**
- Responsibilities:
    - Receives all incoming requests
    - Identifies the correct controller
    - Dispatches the request to the correct handler method

---

5. Configuring DispatcherServlet in web.xml

Purpose of web.xml

- web.xml is used to **talk to Tomcat**
- It tells Tomcat:
    - Which servlet to invoke
    - For which URL patterns

Key Configuration

- All requests (/) are mapped to **DispatcherServlet**
- DispatcherServlet is provided by Spring Framework

---

6. DispatcherServlet Configuration File

Why Another XML File Is Needed

- DispatcherServlet:
    - Exists
    - But does not know about controllers or annotations yet

Naming Convention

- Based on servlet name in web.xml
- Format:

<servlet-name>-servlet.xml

- Example:

        - Servlet name: springmvcdemo
        - Config file: springmvcdemo-servlet.xml
    - Location:
        - WEB-INF/

---

7. Configuring DispatcherServlet (*-servlet.xml)

What Is Configured Here
- This file is used to **talk to DispatcherServlet**
- Key configurations:
    - Enable **annotation-based configuration**
    - Specify **package scanning**

Key Points
- Annotation configuration is enabled
- Package specified:
    - com.telescope
- This allows Spring to:
    - Discover controllers
    - Process @RequestMapping

---

8. View Resolution Problem
- Controller returns logical view names:
    - Example: index
- DispatcherServlet:
    - Does not know where the JSP file is
    - Does not know the file extension

---

9. Configuring Internal Resource View Resolver

Purpose
- Maps logical view names to actual JSP files

Configuration Details
- **InternalResourceViewResolver** is configured
- Two mandatory properties:
    - **Prefix** → location of views
    - **Suffix** → file extension

Example

- Prefix: /views/
- Suffix: .jsp
- Result:
    - index → /views/index.jsp

---

10. How the Request Flow Works End-to-End

1. Client sends a request
2. Tomcat forwards request to **DispatcherServlet**
3. DispatcherServlet:
    - Scans controllers
    - Finds matching handler method
4. Controller returns logical view name
5. View Resolver:
    - Resolves JSP location
6. JSP is rendered and returned to client

---

11. Configuration Scope and Effort

- Most configuration:
    - Is done **once**
    - Applies to the entire project
- As features grow:
    - More configuration may be added
- Core setup remains unchanged

---

12. Spring MVC vs Spring Boot

Spring MVC (Without Boot)

- Requires:
    - Manual configuration
    - More control
- Suitable for:
    - Fine-grained customization
    - Legacy or specific enterprise setups

Spring Boot

- Reduces:

        - XML configuration
        - Setup time
- Preferred for:
        - Faster development
        - Most modern projects