UNIVERSITY OF ECONOMICS & FINANCE

JAVA TECHNOLOGY

Abstract

This practical manual is designed to guide students in building dynamic web applications using Java based on the Spring MVC architecture, with JSP for the user interface and JDBC for connecting to a Microsoft SQL Server database. Through a series of hands-on labs, students will practice managing data flow across the MVC layers (Controller – Model – View), performing CRUD operations (Create, Read, Update, Delete) on real-world databases, and organizing project code logically following the MVC pattern.

The material also includes instructions on: Setting up the SQL Server JDBC Driver; Creating a sample database schema; Common troubleshooting techniques when working with SQL

Hoang Van Hieu, Msc.

FACULTY OF INFORMATION TECHNOLOGY

Internal circulation, 2025

TABLE OF CONTENTS

LAB 1. INTRODUCTION TO SPRING MVC

1. OBJECTIVE

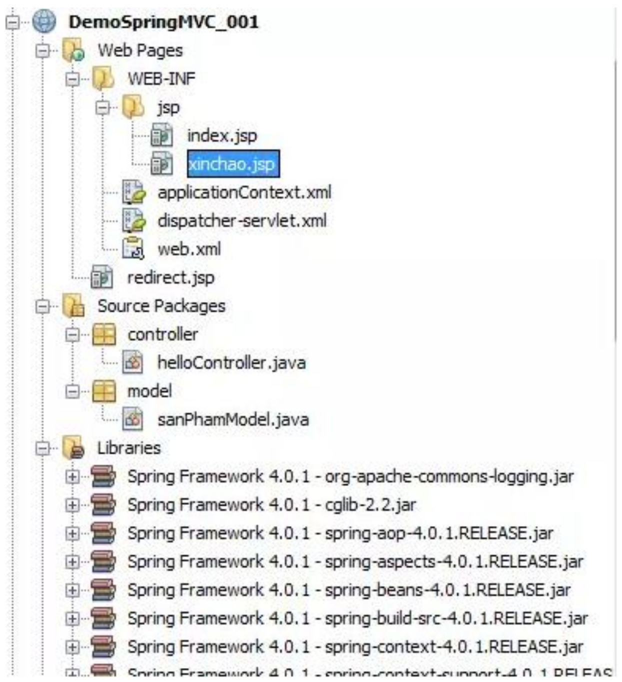
After the lesson, students will be able to:

- Understand the Spring MVC architecture.
- Create a Maven project with Spring MVC.
- Configure DispatcherServlet and ViewResolver.
- Write a Controller to handle data via URL (GET) and Form (POST).
- Display user data on a JSP interface.

2. CREATE A SPRING MVC PROJECT

2.1. Create a Maven Web Project

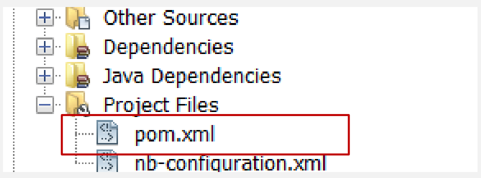
- Open NetBeans
- Navigate to: **File → New Project → Maven → Web Application**
- Set project name: **DemoSpringMVC_001**
- The generated structure should look like:



```
DemoSpringMVC_001
  Web Pages
    WEB-INF
      jsp
        index.jsp
        xinchao.jsp
      applicationContext.xml
      dispatcher-servlet.xml
      web.xml
    redirect.jsp
  Source Packages
    controller
      helloController.java
    model
      sanPhamModel.java
  Libraries
    Spring Framework 4.0.1 - org-apache-commons-logging.jar
    Spring Framework 4.0.1 - cglib-2.2.jar
    Spring Framework 4.0.1 - spring-aop-4.0.1.RELEASE.jar
    Spring Framework 4.0.1 - spring-aspects-4.0.1.RELEASE.jar
    Spring Framework 4.0.1 - spring-beans-4.0.1.RELEASE.jar
    Spring Framework 4.0.1 - spring-build-src-4.0.1.RELEASE.jar
    Spring Framework 4.0.1 - spring-context-4.0.1.RELEASE.jar
    Spring Framework 4.0.1 - spring-context-support-4.0.1.RELEAS
```

2.2. Configure pom.xml

Open the file **pom.xml** and insert the following into the <dependencies> section:



```
<dependencies>
        <!-- Spring MVC -->
        <dependency>
           <groupId>org.springframework</groupId>
           <artifactId>spring-webmvc</artifactId>
           <version>5.3.9</version>
        </dependency>
        <!-- Servlet API (provided by the container, such as Tomcat) -->
        <dependency>
           <groupId>javax.servlet</groupId>
           <artifactId>javax.servlet-api</artifactId>
           <version>4.0.1</version>
           <scope>provided</scope>
        </dependency>
        <!-- JSTL -->
        <dependency>
           <groupId>javax.servlet</groupId>
           <artifactId>jstl</artifactId>
           <version>1.2</version>
        </dependency>
</dependencies>
```

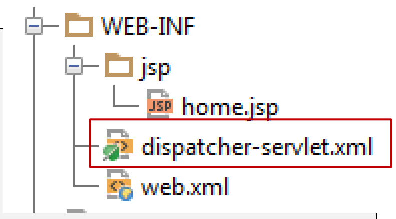
Save the file and wait for the IDE to download the libraries. If not, **right-click** → **Reload Maven Project.**

3. CONFIGURE SPRING MVC

3.1. Create dispatcher-servlet.xml

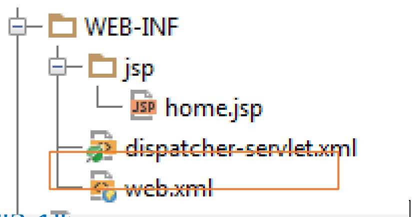
Create a new file: **WEB-INF/dispatcher-servlet.xml**. You can use a different name instead of using the name **"dispatcher",** follow the naming conventions.

After creating the file, replace all current content with the following content:



```
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:context="http://www.springframework.org/schema/context"
     xmlns:mvc="http://www.springframework.org/schema/mvc"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd">

  <!-- Scan Java classes with annotations such as @Controller -->
  <context:component-scan base-package="com.example.controller"/>

  <!--Enable the use of annotations such as @RequestMapping -->
  <mvc:annotation-driven/>

  <!-- Configure the view resolver -->
  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/"/>
    <property name="suffix" value=".jsp"/>
  </bean>

</beans>
```

3.2. Configure web.xml



- Open file: **WEB-INF/web.xml**
- Replace all current content with the following content:

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="3.1">

        <servlet>
           <servlet-name>dispatcher</servlet-name>
           <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
           <init-param>
             <param-name>contextConfigLocation</param-name>
             <param-value>/WEB-INF/dispatcher-servlet.xml</param-value>
           </init-param>
           <load-on-startup>1</load-on-startup>
        </servlet>
        <!-- DispatcherServlet handles all requests at "/" -->
        <servlet-mapping>
           <servlet-name>dispatcher</servlet-name>
           <url-pattern>/</url-pattern>
        </servlet-mapping>

</web-app>
```

- It uses the dispatcher-servlet.xml file to configure controllers, views, beans, etc."

4. CREATE THE CONTROLLER

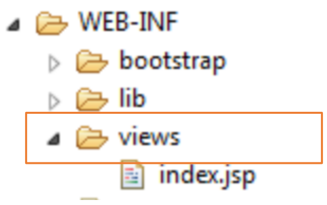
- Create a package: **com.example.controller** (when creating a package containing a Controller, you can use a different prefix; it doesn't have to be "com.example")
- Create class: **HelloController.java**.
- After creating the file, replace all current content with the following content:

```
package com.example.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller  // @Controller: Marks this as a controller
public class HelloController {

   @RequestMapping("/hello") //Handles requests to /hello.
   public String sayHello(Model model) {
      model.addAttribute("message", "Hi! Spring MVC!"); // Passes data to the view.
      return "hello"; // Will look for the hello.jsp file
   }
   // Display the form
   @RequestMapping(value = "/showForm", method = RequestMethod.GET)
   public String showForm() {
      return "input-form";
   }

   // Handle POST submission
   @RequestMapping(value = "/processForm", method = RequestMethod.POST)
   public String processForm(@RequestParam("name") String name, Model model) {
      String message = "Hello " + name + "!";
      model.addAttribute("message", message);
      return "greet";
   }

   // (Optional) Handle GET with query param: /greet?name=An
   @RequestMapping(value = "/greet", method = RequestMethod.GET)
   public String greetUser(@RequestParam("name") String name, Model model) {
      model.addAttribute("message", "Hello " + name + "!");
      return "greet";
   }
}
```

5. CREATE JSP VIEWS

5.1. Create views directory

– Create folder: **/WEB-INF/views**



5.2. Create View

❖ **Create a new hello.jsp file - View**

After creating the file, replace all current content with the following content:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html lang="en-US">
        <head>
            <title>Welcome</title>
        </head>
        <body>
            <h2>${message}</h2>
        </body>
</html>
```

- **Create a new input-form.jsp file - View**

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html lang="en-US">
        <head>
            <title>Enter your name</title>
        </head>
        <body>
            <h2>Please enter your name:</h2>
            <form action="processForm" method="post">
                <input type="text" name="name" required>
                <button type="submit">Submit</button>
            </form>
        </body>
</html>
```

- **Create a new greet.jsp file – View**

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html lang="en-US">
        <head>
            <title>Result</title>
        </head>
        <body>
            <h2>${message}</h2>
        </body>
</html>
```

6. RUN & TEST

- Successfully build the project.
- Deploy it to Apache Tomcat.
- Open browser → test GET and POST requests (e.g., /showForm).

LAB 1. INTRODUCTION TO SPRING MVC

6.1. Add Apache Tomcat to NetBeans

– Go to **Tools** → **Servers**

– Click Add Server

– Choose **Apache Tomcat** → click **Next**

– Browse and select your Tomcat installation folder **(e.g., C:\apache-tomcat-9.x)**

– Set admin username/password if prompted.

– Click **Finish**

If you haven't installed Tomcat yet, download it here: https://tomcat.apache.org/download-90.cgi

6.2. Configure Maven to build WAR file

– Open pom.xml

– Add the following section if not present:

```
<build>
   <finalName>DemoSpringMVC_001</finalName>
   <plugins>
      <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-war-plugin</artifactId>
         <version>3.3.2</version>
      </plugin>
   </plugins>
</build>
```

This ensures the deployed WAR file is named DemoSpringMVC_001, so your URL will be: **http://localhost:8080/DemoSpringMVC_001/**

6.3. Clean and Build, Run the Project

❖ **Clean and Build**

– Right-click the project → select **Clean and Build**

– Check the **Output** window → make sure there are **no build errors**

❖ **Run the Project on Tomcat**

– Right-click your project → select **Run**

– NetBeans will start Apache Tomcat (it may take a few seconds)

– Your default browser should open: http://localhost:8080/**DemoSpringMVC_001**/

– Then manually go to: http://localhost:8080/SpringHelloApp/showForm or http://localhost:8080/SpringHelloApp/hello

❖ **Test the Application**

– A form appears asking for a name.

– You enter “An” and click Submit.

– The next page shows: "Hello An!"

❖ **You continue to implement the additional requirements:**

– Add another field for email and display both name and email.

– Create a form with name and age, and output: ***"Hello An, you are 20 years old!"***

6.4. Common Errors & Fixes

Error	Cause & Solution
HTTP 404 Not Found	Incorrect URL (check /showForm), servlet not mapped properly
HTTP 500 Internal Server Error	Syntax error in JSP or missing JSTL library
Tomcat not showing up	Tomcat server not added/configured in NetBeans
Missing web.xml	Ensure web.xml is located at: /src/main/webapp/WEB-INF/web.xml

7. Advanced Practice Exercises

Exercise 1. Validate empty name and redirect back to form

If the name input is empty:

– Do not display greeting

– Show error message: “Name cannot be empty”

– Reload form with message.

❖ **Update HelloController**

```
@Controller
public class HelloController {

//……
   @RequestMapping(value = "/processForm", method = RequestMethod.POST)
   public String processForm(@RequestParam("name") String name, Model model) {
      if (name == null || name.trim().isEmpty()) {
         model.addAttribute("error", "Name cannot be empty.");
         return "input-form"; // stay on the form page
      }
```

```
        model.addAttribute("message", "Hello " + name + "!");
        return "greet";
    }
//……
}
```

@RequestParam("name") is an **annotation** in Spring MVC used to retrieve the value of a parameter from the request URL or from an HTML form.

❖ **Update input-form.jsp**

– Add JSTL dependency (if not yet), in pom.xml:

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>
```

– Also add this to the top of your input-form.jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
    <head>
        <title>Enter your name</title>
    </head>
    <body>
        <h2>Please enter your name:</h2>

        <!-- Show error if exists -->
        <c:if test="${not empty error}">
            <p style="color: red;">${error}</p>
        </c:if>

        <form action="processForm" method="post">
            <input type="text" name="name" required>
            <button type="submit">Submit</button>
        </form>
    </body>
</html>
```

Exercise 2. Greeting with Name and Age

❖ Requirements:

– Create a form with two fields: name (text), age (number)

– After submission via POST, the application must:

– Check if the name is not empty.

- Display the result: “Hello [Name], you are [Age] years old!”
- If the name is empty:
  - Stay on the form page.
  - Display the error message: “Name cannot be empty.”

Exercise 3. Greeting with Name and Email

❖ Requirements:

- Create a form with the following fields: name (text), email (text)
- After form submission via **POST**, your app must display a message like: **“Hello [Name]! We’ve sent a confirmation to: [Email].”**
- Add basic validation, if name or email is empty:
  - Stay on the form page.
  - Show: **“Please fill in all required fields.”**

The end

LAB 2. INTRODUCTION TO SPRING MVC