How to Create Your First View in Spring MVC?

Last Updated : 23 Jul, 2025

Spring MVC is a powerful Web MVC Framework for building web applications. It is designed around the Model-View-Controller (MVC) pattern, which separates the application into three main components:

- **Model:** Represents the data of the application. It can be a single object or a collection of objects.
- **View:** Responsible for displaying the data to the user in a specific format. Spring supports various view technologies like JSP, Thymeleaf, Freemarker, and Velocity.
- **Controller:** This handles the logical part of the application. Classes marked with the @Controller annotation act as controllers in Spring MVC.
- **Front Controller:** Manages the flow of the web application. In Spring MVC, the DispatcherServlet acts as the front controller.

In this article, we will discuss the steps to create and run our first **view** in a **Spring MVC** application using the **Spring Tool Suite (STS) IDE.**

> ***Note:*** *Views are nothing, they are just web pages.*

Prerequisites:

Before it, certain requirements are needed, as follows:

- Eclipse (EE version)/STS IDE
- Spring JAR Files
- Tomcat Apache's latest version

Note: We are going to use Spring Tool Suite 4 IDE for this project. Please refer to this article to install STS in your local machine: How to Download and Install Spring Tool Suite (Spring Tools 4 for Eclipse) IDE?

Step-by-Step Implementation

Step 1: Create a Dynamic Web Project

You may refer to this article, How to Create a Dynamic Web Project in Spring Tool Suite?

Step 2: Adding Dependencies

Use Maven/Gradle (Recommended): Maven is the preferred approach for managing project dependencies. If you are using Maven, add the following dependencies to your **pom.xml** file

pom.xml:

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.3.23</version>
    </dependency>
    <dependency>
        <groupId>jakarta.servlet</groupId>
        <artifactId>jakarta.servlet-api</artifactId>
        <version>5.0.0</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

If you are using Gradle, add the following to your **build.gradle** file:

```
dependencies {
    implementation 'org.springframework:spring-webmvc:5.3.23'
    compileOnly 'jakarta.servlet:jakarta.servlet-api:5.0.0'
}
```

Step 3: Configure the Tomcat Server with the Application

Next step is to Configure Apache Tomcat Server. Now we are ready to go.

Configuring Dispatcher Servlet

Refer to this article What is Dispatcher Servlet in Spring? and read more about Dispatcher Servlet which is a very very important concept to understand. Now we are going to configure Dispatcher Servlet with our Spring MVC application.

Step 4: Create web.xml File

Go to the **src > main > webapp > WEB-INF > web.xml** file.

web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"

xmlns="http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/index.html"

xsi:schemaLocation="http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/index.html

http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/index.html/web-app_4_0.xsd"
    id="WebApp_ID" version="4.0">

    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

Explanation: The DispatcherServlet is the entry point of the application. We can think of a DispatcherServlet as the gatekepper, it handles all the

incoming requests. When a request comes, it passes to the DispatcherServlet and then it decides which controller should handle it.

Note: Here, the **<url-pattern>/</url-pattern>** tells the spring that this servlet handle every request, no matter what URL the user visits.

Step 5: Create dispatcher-servlet.xml File

Now go to the **src > main > webapp > WEB-INF** and create an XML file. Actually, this is a Spring Configuration file like beans.xml file. And the name of the file must be in this format.

YourServletName-servlet.xml

For example, for this project, the name of the file must be:

dispatcher-servlet.xml

So either you can create a Spring Configuration File or you can just create a simple XML file add the below lines of code inside that file.

dispatcher-servlet.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans/"
       xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context/"
       xsi:schemaLocation="http://www.springframework.org/schema/beans/
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context/
        https://www.springframework.org/schema/context/spring-context.xsd">

    <!-- Scans for @Controller annotations -->
    <context:component-scan base-package="com.demo.controllers" />

    <!-- Configures JSP ViewResolver -->
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
```

```
    </bean>
</beans>
```

Explanation: The **<context:component-scan>** tells the spring to look through the com.demo.controllers package for any classes marked with @Controller. The classes will handle the requests. The **<InternalResourceViewResolver>** helps Spring find the right JSP file when a view name like "demo" is returned by a controller.

Step 6: Creating Spring MVC Controller

Now, let's create some controllers. Go to the src/main/java and create a new controllers package (For ex. com.demo.controllers) as per your choice. And inside that create a Java class and name the class as **DemoController**. Now how to tell the Spring that this is our controller class. So the way we are going to tell the Spring is by marking it with a @Controller annotation.

```
@Controller
public class DemoController {}
```

Note: Spring will automatically initialize the class having a @Controller annotation and register that class with the spring container.

Now let us create a simple method inside the Controller class and use @GetMapping annotation before the method something like this.

```
// Annotation
@GetMapping("/hello")
// Method
public String helloWorld()
{
    return "demo";
}
```

Now in the return statement, we have to return some views (web pages), so whenever the endpoint '/hello' is invoked we can see our result on the web page. So let's create our first View.

Creating First View

Go to the **src > main > webapp > WEB-INF > right-click > New > Folder** and name the folder as **views.** Then **views > right-click > New > JSP File** and name your first view. Here we have named it as **demo.jsp** file. Below is the code for the **demo.jsp** file. We have created a simple web page inside that file.

demo.jsp:

```
<!DOCTYPE html>
<html>
<body bgcolor="green">
    <h1>Hello GeeksforGeeks!</h1>
</body>
</html>
```

Explanation: Here, we have created a jsp file, which contains a basic html code and outputs "Hello GeeksforGeeks" in the body with the green background.

Now go to the DemoController class and inside the helloWorld() method we have to return a value something like this and we are done.

return "demo";

We have just been given the path for our view. So the complete code for the DemoController.java is given below.

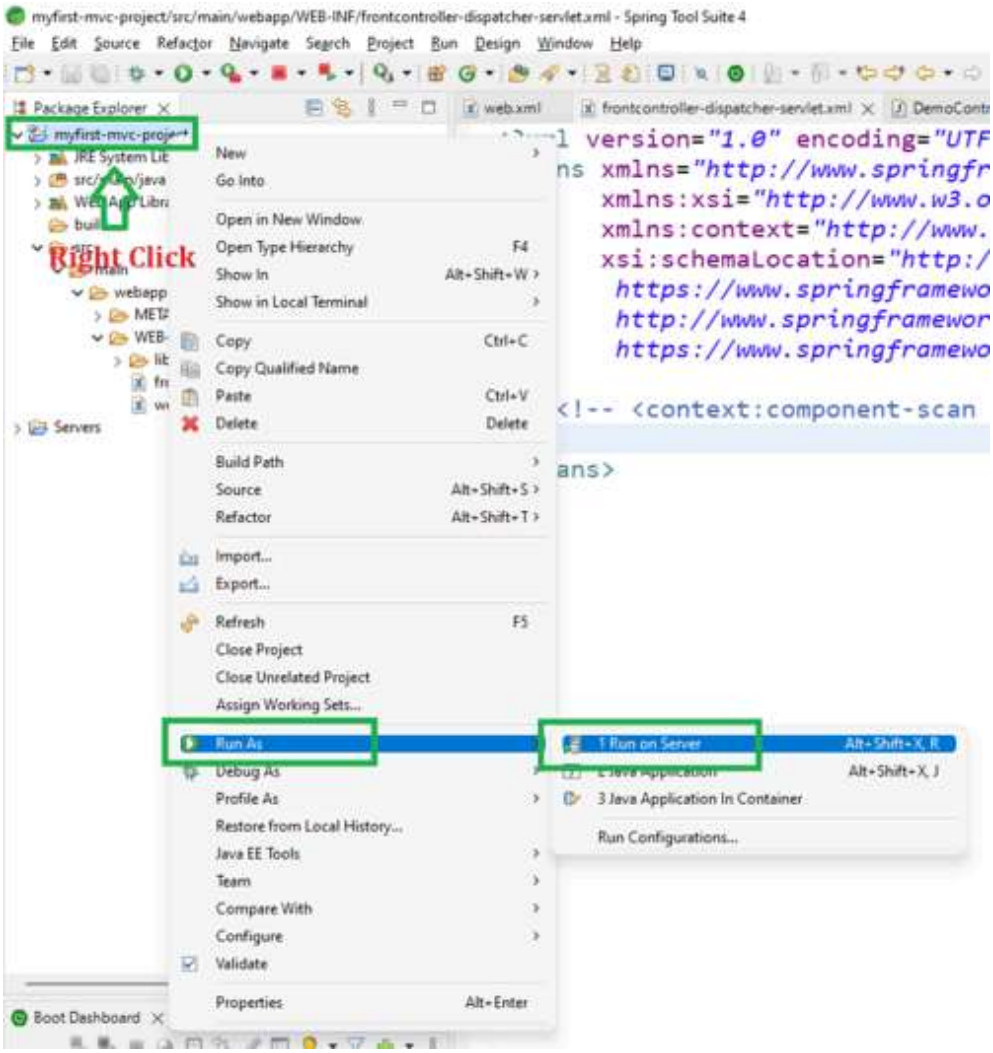
DemoController.java:

```
package com.demo.controllers;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class DemoController {
```

```
    @RequestMapping("/hello")
    public String helloWorld() {
        return "demo";
    }

}
```

Step 7: Run Spring MVC Application

To run your Spring MVC Application **right-click on your project > Run As > Run on Server** and run your application as shown in the below image.



After that use the following URL to run your controller:

http://localhost:8080/springmvc-view-resolver/hello

Output:

Comment | More info | Advertise with us