A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.

A Spring MVC provides an elegant solution to use MVC in spring framework by the help of **DispatcherServlet**. Here, **DispatcherServlet** is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.

Spring Web Model-View-Controller

* **Model** - A model contains the data of the application. A data can be a single object or a collection of objects.
* **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
* **View** - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.
* **Front Controller** - In Spring Web MVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.

Understanding the flow of Spring Web MVC

* As displayed in the figure, all the incoming request is intercepted by the DispatcherServlet that works as the front controller.
* The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller.
* The controller returns an object of ModelAndView.
* The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.

Advantages of Spring MVC Framework

Let's see some of the advantages of Spring MVC Framework:-

* **Separate roles** - The Spring MVC separates each role, where the model object, controller, command object, view resolver, DispatcherServlet, validator, etc. can be fulfilled by a specialized object.
* **Light-weight** - It uses light-weight servlet container to develop and deploy your application.
* **Powerful Configuration** - It provides a robust configuration for both framework and application classes that includes easy referencing across contexts, such as from web controllers to business objects and validators.
* **Rapid development** - The Spring MVC facilitates fast and parallel development.
* **Reusable business code** - Instead of creating new objects, it allows us to use the existing business objects.
* **Easy to test** - In Spring, generally we create JavaBeans classes that enable you to inject test data using the setter methods.
* **Flexible Mapping** - It provides the specific annotations that easily redirect the page.

## Spring Web MVC Framework Example

Let's see the simple example of a Spring Web MVC framework. The steps are as follows:

* Load the spring jar files or add dependencies in the case of Maven
* Create the controller class
* Provide the entry of controller in the web.xml file
* Define the bean in the separate XML file
* Display the message in the JSP page
* Start the server and deploy the project

### Create the controller class

To create the controller class, we are using two annotations @Controller and @RequestMapping.

The @Controller annotation marks this class as Controller.

The @Requestmapping annotation is used to map the class with the specified URL name.

**HelloController.java**

1. **package** com.javatpoint;
2. **import** org.springframework.stereotype.Controller;
3. **import** org.springframework.web.bind.annotation.RequestMapping;
4. @Controller
5. **public** **class** HelloController {
6. @RequestMapping("/")
7. **public** String display()
8. {
9. **return** "index";
10. }
11. }

### Provide the entry of controller in the web.xml file

In this xml file, we are specifying the servlet class DispatcherServlet that acts as the front controller in Spring Web MVC. All the incoming request for the html file will be forwarded to the DispatcherServlet.

**web.xml**

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd" id="WebApp\_ID" version="3.0"**>**
3. **<display-name>**SpringMVC**</display-name>**
4. **<servlet>**
5. **<servlet-name>**spring**</servlet-name>**
6. **<servlet-class>**org.springframework.web.servlet.DispatcherServlet**</servlet-class>**
7. **<load-on-startup>**1**</load-on-startup>**
8. **</servlet>**
9. **<servlet-mapping>**
10. **<servlet-name>**spring**</servlet-name>**
11. **<url-pattern>**/**</url-pattern>**
12. **</servlet-mapping>**
13. **</web-app>**

### Define the bean in the xml file

This is the important configuration file where we need to specify the View components.

The context:component-scan element defines the base-package where DispatcherServlet will search the controller class.

This xml file should be located inside the WEB-INF directory.

**spring-servlet.xml**

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans** xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:context="http://www.springframework.org/schema/context"
5. xmlns:mvc="http://www.springframework.org/schema/mvc"
6. xsi:schemaLocation="
7. http://www.springframework.org/schema/beans
8. http://www.springframework.org/schema/beans/spring-beans.xsd
9. http://www.springframework.org/schema/context
10. http://www.springframework.org/schema/context/spring-context.xsd
11. http://www.springframework.org/schema/mvc
12. http://www.springframework.org/schema/mvc/spring-mvc.xsd"**>**
14. <!-- Provide support for component scanning -->
15. **<context:component-scan** base-package="com.javatpoint" **/>**
17. <!--Provide support for conversion, formatting and validation -->
18. **<mvc:annotation-driven/>**
20. **</beans>**

### Display the message in the JSP page

This is the simple JSP page, displaying the message returned by the Controller.

**index.jsp**

1. **<html>**
2. **<body>**
3. **<p>**Welcome to Spring MVC Tutorial**</p>**
4. **</body>**
5. **</html>**

# Spring MVC Multiple View page Example

[**next →**](https://www.javatpoint.com/spring-mvc-multiple-controller-example)[**← prev**](https://www.javatpoint.com/spring-mvc-tutorial)

# Spring MVC Multiple View page Example

Here, we redirect a view page to another view page.

Let's see the simple example of a Spring Web MVC framework. The steps are as follows:

* Load the spring jar files or add dependencies in the case of Maven
* Create the controller class
* Provide the entry of controller in the web.xml file
* Define the bean in the separate XML file
* Create the other view components
* Start the server and deploy the project

## Directory Structure of Spring MVC

### 1. Add dependencies to pom.xml

1. <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
2. **<dependency>**
3. **<groupId>**org.springframework**</groupId>**
4. **<artifactId>**spring-webmvc**</artifactId>**
5. **<version>**5.1.1.RELEASE**</version>**
6. **</dependency>**
8. <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
9. **<dependency>**
10. **<groupId>**javax.servlet**</groupId>**
11. **<artifactId>**servlet-api**</artifactId>**
12. **<version>**3.0-alpha-1**</version>**
13. **</dependency>**

### 2. Create the request page

Let's create a simple jsp page containing a link.

**index.jsp**

1. **<html>**
2. **<body>**
3. **<a** href="hello"**>**Click here...**</a>**
4. **</body>**
5. **</html>**

### 3. Create the controller class

Let's create a controller class that returns the JSP pages. Here, we pass the specific name with a @Requestmapping annotation to map the class.

**HelloController.java**

1. **package** com.javatpoint;
2. **import** org.springframework.stereotype.Controller;
3. **import** org.springframework.web.bind.annotation.RequestMapping;
4. @Controller
5. **public** **class** HelloController {
6. @RequestMapping("/hello")
7. **public** String redirect()
8. {
9. **return** "viewpage";
10. }
11. @RequestMapping("/helloagain")
12. **public** String display()
13. {
14. **return** "final";
15. }
16. }

### 4. Provide the entry of controller in the web.xml file

**web.xml**

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd" id="WebApp\_ID" version="3.0"**>**
3. **<display-name>**SpringMVC**</display-name>**
4. **<servlet>**
5. **<servlet-name>**spring**</servlet-name>**
6. **<servlet-class>**org.springframework.web.servlet.DispatcherServlet**</servlet-class>**
7. **<load-on-startup>**1**</load-on-startup>**
8. **</servlet>**
9. **<servlet-mapping>**
10. **<servlet-name>**spring**</servlet-name>**
11. **<url-pattern>**/**</url-pattern>**
12. **</servlet-mapping>**
13. **</web-app>**

### 5. Define the bean in the xml file

Now, we also provide view resolver with view component.

Here, the InternalResourceViewResolver class is used for the ViewResolver.

The prefix+string returned by controller+suffix page will be invoked for the view component.

This xml file should be located inside the WEB-INF directory.

**spring-servlet.xml**

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans** xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:context="http://www.springframework.org/schema/context"
5. xmlns:mvc="http://www.springframework.org/schema/mvc"
6. xsi:schemaLocation="
7. http://www.springframework.org/schema/beans
8. http://www.springframework.org/schema/beans/spring-beans.xsd
9. http://www.springframework.org/schema/context
10. http://www.springframework.org/schema/context/spring-context.xsd
11. http://www.springframework.org/schema/mvc
12. http://www.springframework.org/schema/mvc/spring-mvc.xsd"**>**
14. <!-- Provide support for component scanning -->
15. **<context:component-scan** base-package="com.javatpoint" **/>**
17. <!--Provide support for conversion, formatting and validation -->
18. **<mvc:annotation-driven/>**
19. <!-- Define Spring MVC view resolver -->
20. **<bean** id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>**
21. **<property** name="prefix" value="/WEB-INF/jsp/"**></property>**
22. **<property** name="suffix" value=".jsp"**></property>**
23. **</bean>**
24. **</beans>**

### 6. Create the other view components

**viewpage.jsp**

1. **<html>**
2. **<body>**
3. **<a** href="helloagain"**>**Javatpoint Tutorials**</a>**
4. **</body>**
5. **</html>**

**final.jsp**

1. **<html>**
2. **<body>**
3. **<p>**Welcome to Spring MVC Tutorial**</p>**
4. **</body>**
5. **</html>**

# Spring MVC Multiple Controller Example

In Spring MVC, we can create multiple controllers at a time. It is required to map each controller class with **@Controller** annotation. Here, we see a Spring MVC example of multiple controllers.

### Create the request page

Let's create a simple JSP page containing two links.

**index.jsp**

1. **<html>**
2. **<body>**
3. **<a** href="hello1"**>**Spring MVC**</a>** ||
4. **<a** href="hello2"**>**Spring Boot**</a>**
5. **</body>**
6. **</html>**

### 3. Create the controller class

Let's create two controller classes, where each returns the particular view page.

**HelloController1.java**

1. package com.javatpoint;
2. import org.springframework.stereotype.Controller;
3. import org.springframework.web.bind.annotation.RequestMapping;
4. @Controller
5. public class HelloController1 {
6. @RequestMapping("/hello1")
7. public String display()
8. {
9. return "viewpage1";
10. }
11. }

**HelloController2.java**

1. package com.javatpoint;
2. import org.springframework.stereotype.Controller;
3. import org.springframework.web.bind.annotation.RequestMapping;
4. @Controller
5. public class HelloController2 {
6. @RequestMapping("/hello2")
7. public String display()
8. {
9. return "viewpage2";
10. }
11. }

### 4. Provide the entry of controller in the web.xml file

**web.xml**

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd" id="WebApp\_ID" version="3.0"**>**
3. **<display-name>**SpringMVC**</display-name>**
4. **<servlet>**
5. **<servlet-name>**spring**</servlet-name>**
6. **<servlet-class>**org.springframework.web.servlet.DispatcherServlet**</servlet-class>**
7. **<load-on-startup>**1**</load-on-startup>**
8. **</servlet>**
9. **<servlet-mapping>**
10. **<servlet-name>**spring**</servlet-name>**
11. **<url-pattern>**/**</url-pattern>**
12. **</servlet-mapping>**
13. **</web-app>**

### 5. Define the bean in the xml file

**spring-servlet.xml**

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans** xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:context="http://www.springframework.org/schema/context"
5. xmlns:mvc="http://www.springframework.org/schema/mvc"
6. xsi:schemaLocation="
7. http://www.springframework.org/schema/beans
8. http://www.springframework.org/schema/beans/spring-beans.xsd
9. http://www.springframework.org/schema/context
10. http://www.springframework.org/schema/context/spring-context.xsd
11. http://www.springframework.org/schema/mvc
12. http://www.springframework.org/schema/mvc/spring-mvc.xsd"**>**
14. <!-- Provide support for component scanning -->
15. **<context:component-scan** base-package="com.javatpoint" **/>**
17. <!--Provide support for conversion, formatting and validation -->
18. **<mvc:annotation-driven/>**
19. **<bean** id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>**
20. **<property** name="prefix" value="/WEB-INF/jsp/"**></property>**
21. **<property** name="suffix" value=".jsp"**></property>**
22. **</bean>**
23. **</beans>**

### 6. Create the other view components

**viewpage1.jsp**

1. **<html>**
2. **<body>**
3. **<p>**Welcome to Spring MVC Tutorial**</p>**
4. **</body>**
5. **</html>**

**viewpage1.jsp**

1. **<html>**
2. **<body>**
3. **<p>**Welcome to Spring Boot Tutorial**</p>**
4. **</body>**
5. **</html>**

# Spring MVC Model Interface

In Spring MVC, the model works a container that contains the data of the application. Here, a data can be in any form such as objects, strings, information from the database, etc.

It is required to place the **Model** interface in the controller part of the application. The object of **HttpServletRequest** reads the information provided by the user and pass it to the **Model** interface. Now, a view page easily accesses the data from the model part.

## Methods of Model Interface

|  |  |
| --- | --- |
| **Method** | **Description** |
| Model addAllAttributes(Collection<?> arg) | It adds all the attributes in the provided Collection into this Map. |
| Model addAllAttributes(Map<String,?> arg) | It adds all the attributes in the provided Map into this Map. |
| Model addAllAttribute(Object arg) | It adds the provided attribute to this Map using a generated name. |
| Model addAllAttribute(String arg0, Object arg1) | It binds the attribute with the provided name. |
| Map<String, Object> asMap() | It return the current set of model attributes as a Map. |
| Model mergeAttributes(Map< String,?> arg) | It adds all attributes in the provided Map into this Map, with existing objects of the same name taking precedence. |
| boolean containsAttribute(String arg) | It indicates whether this model contains an attribute of the given name |

## Spring MVC Model Example

### 2. Create the request page

Here, we create the login page to receive name and password from the user.

**index.jsp**

1. **<html>**
2. **<body>**
3. **<form** action="hello"**>**
4. UserName : **<input** type="text" name="name"**/>** **<br><br>**
5. Password : **<input** type="text" name="pass"**/>** **<br><br>**
6. **<input** type="submit" name="submit"**>**
7. **</form>**
8. **</body>**
9. **</html>**

### 3. Create the controller class

In controller class:

* The **HttpServletRequest** is used to read the HTML form data provided by the user.
* The **Model** contains the request data and provides it to view page.

**HelloController.java**

1. **package** com.javatpoint;
2. **import** javax.servlet.http.HttpServletRequest;
3. **import** org.springframework.stereotype.Controller;
4. **import** org.springframework.ui.Model;
5. **import** org.springframework.web.bind.annotation.RequestMapping;
7. @Controller
8. **public** **class** HelloController {
10. @RequestMapping("/hello")
11. **public** String display(HttpServletRequest req,Model m)
12. {
13. //read the provided form data
14. String name=req.getParameter("name");
15. String pass=req.getParameter("pass");
16. **if**(pass.equals("admin"))
17. {
18. String msg="Hello "+ name;
19. //add a message to the model
20. m.addAttribute("message", msg);
21. **return** "viewpage";
22. }
23. **else**
24. {
25. String msg="Sorry "+ name+". You entered an incorrect password";
26. m.addAttribute("message", msg);
27. **return** "errorpage";
28. }
29. }
30. }

### 4. Provide the entry of controller in the web.xml file

**web.xml**

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd" id="WebApp\_ID" version="3.0"**>**
3. **<display-name>**SpringMVC**</display-name>**
4. **<servlet>**
5. **<servlet-name>**spring**</servlet-name>**
6. **<servlet-class>**org.springframework.web.servlet.DispatcherServlet**</servlet-class>**
7. **<load-on-startup>**1**</load-on-startup>**
8. **</servlet>**
9. **<servlet-mapping>**
10. **<servlet-name>**spring**</servlet-name>**
11. **<url-pattern>**/**</url-pattern>**
12. **</servlet-mapping>**
13. **</web-app>**

### 5. Define the bean in the xml file

**spring-servlet.xml**

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans** xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:context="http://www.springframework.org/schema/context"
5. xmlns:mvc="http://www.springframework.org/schema/mvc"
6. xsi:schemaLocation="
7. http://www.springframework.org/schema/beans
8. http://www.springframework.org/schema/beans/spring-beans.xsd
9. http://www.springframework.org/schema/context
10. http://www.springframework.org/schema/context/spring-context.xsd
11. http://www.springframework.org/schema/mvc
12. http://www.springframework.org/schema/mvc/spring-mvc.xsd"**>**
14. <!-- Provide support for component scanning -->
15. **<context:component-scan** base-package="com.javatpoint" **/>**
17. <!--Provide support for conversion, formatting and validation -->
18. **<mvc:annotation-driven/>**
19. **<bean** id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>**
20. **<property** name="prefix" value="/WEB-INF/jsp/"**></property>**
21. **<property** name="suffix" value=".jsp"**></property>**
22. **</bean>**
23. **</beans>**

### 6. Create the other view components

To run this example, the following view components must be located inside the WEB-INF/jsp directory.

**viewpage.jsp**

1. **<html>**
2. **<body>**
3. ${message}
4. **</body>**
5. **</html>**

**errorpage.jsp**

1. **<html>**
2. **<body>**
3. ${message}
4. **<br><br>**
5. **<jsp:include** page="/index.jsp"**></jsp:include>**
6. **</body>**
7. **</html>**

# Spring MVC Form Tag Library

The Spring MVC form tags are the configurable and reusable building blocks for a web page. These tags provide JSP, an easy way to develop, read and maintain.

The Spring MVC form tags can be seen as data binding-aware tags that can automatically set data to Java object/bean and also retrieve from it. Here, each tag provides support for the set of attributes of its corresponding HTML tag counterpart, making the tags familiar and easy to use.

## Configuration of Spring MVC Form Tag

The form tag library comes under the spring-webmvc.jar. To enable the support for form tag library, it is required to reference some configuration. So, add the following directive at the beginning of the JSP page:

1. **<**%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %**>**

## List of Spring MVC Form Tags

Let's see some of the frequently used Spring MVC form tags.

|  |  |
| --- | --- |
| **Form Tag** | **Description** |
| form:form | It is a container tag that contains all other form tags. |
| form:input | This tag is used to generate the text field. |
| form:radiobutton | This tag is used to generate the radio buttons. |
| form:checkbox | This tag is used to generate the checkboxes. |
| form:password | This tag is used to generate the password input field. |
| form:select | This tag is used to generate the drop-down list. |
| form:textarea | This tag is used to generate the multi-line text field. |
| form:hidden | This tag is used to generate the hidden input field. |

## The form tag

The Spring MVC form tag is a container tag. It is a parent tag that contains all the other tags of the tag library. This tag generates an HTML form tag and exposes a binding path to the inner tags for binding.

### Syntax

1. **<form:form** action="nextFormPath" modelAttribute=?abc**?>**

# Spring MVC Form Text Field

The Spring MVC form text field tag generates an HTML input tag using the bound value. By default, the type of the input tag is text.

### Syntax

1. **<form:input** path="name" **/>**

Here, **path** attribute binds the form field to the bean property.

The Spring MVC form tag library also provides other input types such as email, date, tel, etc.

### For email:

1. **<form:input** type=?email? path="email" **/>**

### For date:

1. **<form:input** type=?date? path="date" **/>**

## Example of Spring MVC Form Text Field

Let's see an example to create a railway reservation form using form tag library.

### 1. Add dependencies to pom.xml file.

1. <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
2. **<dependency>**
3. **<groupId>**org.springframework**</groupId>**
4. **<artifactId>**spring-webmvc**</artifactId>**
5. **<version>**5.1.1.RELEASE**</version>**
6. **</dependency>**
8. <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
9. **<dependency>**
10. **<groupId>**javax.servlet**</groupId>**
11. **<artifactId>**servlet-api**</artifactId>**
12. **<version>**3.0-alpha-1**</version>**
13. **</dependency>**
15. <!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
16. **<dependency>**
17. **<groupId>**javax.servlet**</groupId>**
18. **<artifactId>**jstl**</artifactId>**
19. **<version>**1.2**</version>**
20. **</dependency>**
21. <!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jasper -->
22. **<dependency>**
23. **<groupId>**org.apache.tomcat**</groupId>**
24. **<artifactId>**tomcat-jasper**</artifactId>**
25. **<version>**9.0.12**</version>**
26. **</dependency>**

### 2. Create the bean class

Here, the bean class contains the variables (along setter and getter methods) corresponding to the input field exist in the form.

**Reservation.java**

1. **package** com.javatpoint;
3. **public** **class** Reservation {
5. **private** String firstName;
6. **private** String lastName;
8. **public** Reservation()
9. {
10. }
11. **public** String getFirstName() {
12. **return** firstName;
13. }
14. **public** **void** setFirstName(String firstName) {
15. **this**.firstName = firstName;
16. }
17. **public** String getLastName() {
18. **return** lastName;
19. }
20. **public** **void** setLastName(String lastName) {
21. **this**.lastName = lastName;
22. }
23. }

### 3. Create the controller class

**ReservationController.java**

1. **package** com.javatpoint;
2. **import** org.springframework.stereotype.Controller;
3. **import** org.springframework.ui.Model;
4. **import** org.springframework.web.bind.annotation.ModelAttribute;
5. **import** org.springframework.web.bind.annotation.RequestMapping;
7. @RequestMapping("/reservation")
8. @Controller
9. **public** **class** ReservationController {
10. @RequestMapping("/bookingForm")
11. **public** String bookingForm(Model model)
12. {
13. //create a reservation object
14. Reservation res=**new** Reservation();
15. //provide reservation object to the model
16. model.addAttribute("reservation", res);
17. **return** "reservation-page";
18. }
19. @RequestMapping("/submitForm")
20. // @ModelAttribute binds form data to the object
21. **public** String submitForm(@ModelAttribute("reservation") Reservation res)
22. {
23. **return** "confirmation-form";
24. }
25. }

### 4. Provide the entry of controller in the web.xml file

**web.xml**

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd" id="WebApp\_ID" version="3.0"**>**
3. **<display-name>**SpringMVC**</display-name>**
4. **<servlet>**
5. **<servlet-name>**spring**</servlet-name>**
6. **<servlet-class>**org.springframework.web.servlet.DispatcherServlet**</servlet-class>**
7. **<load-on-startup>**1**</load-on-startup>**
8. **</servlet>**
9. **<servlet-mapping>**
10. **<servlet-name>**spring**</servlet-name>**
11. **<url-pattern>**/**</url-pattern>**
12. **</servlet-mapping>**
13. **</web-app>**

### 5. Define the bean in the xml file

**spring-servlet.xml**

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans** xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:context="http://www.springframework.org/schema/context"
5. xmlns:mvc="http://www.springframework.org/schema/mvc"
6. xsi:schemaLocation="
7. http://www.springframework.org/schema/beans
8. http://www.springframework.org/schema/beans/spring-beans.xsd
9. http://www.springframework.org/schema/context
10. http://www.springframework.org/schema/context/spring-context.xsd
11. http://www.springframework.org/schema/mvc
12. http://www.springframework.org/schema/mvc/spring-mvc.xsd"**>**
13. <!-- Provide support for component scanning -->
14. **<context:component-scan** base-package="com.javatpoint" **/>**
15. <!--Provide support for conversion, formatting and validation -->
16. **<mvc:annotation-driven/>**
17. <!-- Define Spring MVC view resolver -->
18. **<bean** id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>**
19. **<property** name="prefix" value="/WEB-INF/jsp/"**></property>**
20. **<property** name="suffix" value=".jsp"**></property>**
21. **</bean>**
22. **</beans>**

### 6. Create the requested page

**index.jsp**

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<title>**Railway Registration Form**</title>**
5. **</head>**
6. **<body>**
7. **<a** href="reservation/bookingForm"**>**Click here for reservation.**</a>**
8. **</body>**
9. **</html>**

### 7. Create other view components

**reservation-page.jsp**

1. **<**%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %**>**
2. <!DOCTYPE html**>**
3. **<html>**
4. **<head>**
5. **<title>**Reservation Form**</title>**
6. **</head>**
7. **<h3>**Railway Reservation Form**</h3>**
8. **<body>**
9. **<form:form** action="submitForm" modelAttribute="reservation"**>**
10. First name: **<form:input** path="firstName" **/>**
11. **<br><br>**
12. Last name: **<form:input** path="lastName" **/>**
13. **<br><br>**
14. **<input** type="submit" value="Submit" **/>**
15. **</form:form>**
16. **</body>**
17. **</html>**

### Note - The value passed with the @ModelAttribute annotation should be the same to the modelAttribute value present in the view page.

**confirmation-page.jsp**

1. <!DOCTYPE html**>**
2. **<html>**
3. **<body>**
4. **<p>**Your reservation is confirmed successfully. Please, re-check the details.**</p>**
5. First Name : ${reservation.firstName} **<br>**
6. Last Name : ${reservation.lastName}
7. **</body>**
8. **</html>**

# Spring MVC CRUD Example

CRUD (Create, Read, Update and Delete) application is the most important application for creating any project. It provides an idea to develop a large project. In spring MVC, we can develop a simple CRUD application.

Here, we are using **JdbcTemplate** for database interaction.

## Create a table

Here, we are using emp99 table present in the MySQL database. It has 4 fields: id, name, salary, and designation. Here, id is auto incremented which is generated by the sequence.

## Spring MVC CRUD Example

### 1. Add dependencies to pom.xml file.

**pom.xml**

1. <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
2. **<dependency>**
3. **<groupId>**org.springframework**</groupId>**
4. **<artifactId>**spring-webmvc**</artifactId>**
5. **<version>**5.1.1.RELEASE**</version>**
6. **</dependency>**
8. <!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jasper -->
9. **<dependency>**
10. **<groupId>**org.apache.tomcat**</groupId>**
11. **<artifactId>**tomcat-jasper**</artifactId>**
12. **<version>**9.0.12**</version>**
13. **</dependency>**
14. <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
15. **<dependency>**
16. **<groupId>**javax.servlet**</groupId>**
17. **<artifactId>**servlet-api**</artifactId>**
18. **<version>**3.0-alpha-1**</version>**
19. **</dependency>**
20. <!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
21. **<dependency>**
22. **<groupId>**javax.servlet**</groupId>**
23. **<artifactId>**jstl**</artifactId>**
24. **<version>**1.2**</version>**
25. **</dependency>**
26. <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
27. **<dependency>**
28. **<groupId>**mysql**</groupId>**
29. **<artifactId>**mysql-connector-java**</artifactId>**
30. **<version>**8.0.11**</version>**
31. **</dependency>**
32. <!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
33. **<dependency>**
34. **<groupId>**org.springframework**</groupId>**
35. **<artifactId>**spring-jdbc**</artifactId>**
36. **<version>**5.1.1.RELEASE**</version>**
37. **</dependency>**

### 2. Create the bean class

Here, the bean class contains the variables (along setter and getter methods) corresponding to the fields exist in the database.

**Emp.java**

1. **package** com.javatpoint.beans;
3. **public** **class** Emp {
4. **private** **int** id;
5. **private** String name;
6. **private** **float** salary;
7. **private** String designation;
9. **public** **int** getId() {
10. **return** id;
11. }
12. **public** **void** setId(**int** id) {
13. **this**.id = id;
14. }
15. **public** String getName() {
16. **return** name;
17. }
18. **public** **void** setName(String name) {
19. **this**.name = name;
20. }
21. **public** **float** getSalary() {
22. **return** salary;
23. }
24. **public** **void** setSalary(**float** salary) {
25. **this**.salary = salary;
26. }
27. **public** String getDesignation() {
28. **return** designation;
29. }
30. **public** **void** setDesignation(String designation) {
31. **this**.designation = designation;
32. }
34. }

### 3. Create the controller class

**EmpController.java**

1. **package** com.javatpoint.controllers;
2. **import** java.util.List;
3. **import** org.springframework.beans.factory.annotation.Autowired;
4. **import** org.springframework.stereotype.Controller;
5. **import** org.springframework.ui.Model;
6. **import** org.springframework.web.bind.annotation.ModelAttribute;
7. **import** org.springframework.web.bind.annotation.PathVariable;
8. **import** org.springframework.web.bind.annotation.RequestMapping;
9. **import** org.springframework.web.bind.annotation.RequestMethod;
10. **import** com.javatpoint.beans.Emp;
11. **import** com.javatpoint.dao.EmpDao;
12. @Controller
13. **public** **class** EmpController {
14. @Autowired
15. EmpDao dao;//will inject dao from XML file
17. /\*It displays a form to input data, here "command" is a reserved request attribute
18. \*which is used to display object data into form
19. \*/
20. @RequestMapping("/empform")
21. **public** String showform(Model m){
22. m.addAttribute("command", **new** Emp());
23. **return** "empform";
24. }
25. /\*It saves object into database. The @ModelAttribute puts request data
26. \*  into model object. You need to mention RequestMethod.POST method
27. \*  because default request is GET\*/
28. @RequestMapping(value="/save",method = RequestMethod.POST)
29. **public** String save(@ModelAttribute("emp") Emp emp){
30. dao.save(emp);
31. **return** "redirect:/viewemp";//will redirect to viewemp request mapping
32. }
33. /\* It provides list of employees in model object \*/
34. @RequestMapping("/viewemp")
35. **public** String viewemp(Model m){
36. List<Emp> list=dao.getEmployees();
37. m.addAttribute("list",list);
38. **return** "viewemp";
39. }
40. /\* It displays object data into form for the given id.
41. \* The @PathVariable puts URL data into variable.\*/
42. @RequestMapping(value="/editemp/{id}")
43. **public** String edit(@PathVariable **int** id, Model m){
44. Emp emp=dao.getEmpById(id);
45. m.addAttribute("command",emp);
46. **return** "empeditform";
47. }
48. /\* It updates model object. \*/
49. @RequestMapping(value="/editsave",method = RequestMethod.POST)
50. **public** String editsave(@ModelAttribute("emp") Emp emp){
51. dao.update(emp);
52. **return** "redirect:/viewemp";
53. }
54. /\* It deletes record for the given id in URL and redirects to /viewemp \*/
55. @RequestMapping(value="/deleteemp/{id}",method = RequestMethod.GET)
56. **public** String delete(@PathVariable **int** id){
57. dao.delete(id);
58. **return** "redirect:/viewemp";
59. }
60. }

### 4. Create the DAO class

Let's create a DAO class to access the required data from the database.

**EmpDao.java**

1. **package** com.javatpoint.dao;
2. **import** java.sql.ResultSet;
3. **import** java.sql.SQLException;
4. **import** java.util.List;
5. **import** org.springframework.jdbc.core.BeanPropertyRowMapper;
6. **import** org.springframework.jdbc.core.JdbcTemplate;
7. **import** org.springframework.jdbc.core.RowMapper;
8. **import** com.javatpoint.beans.Emp;
10. **public** **class** EmpDao {
11. JdbcTemplate template;
13. **public** **void** setTemplate(JdbcTemplate template) {
14. **this**.template = template;
15. }
16. **public** **int** save(Emp p){
17. String sql="insert into Emp99(name,salary,designation) values('"+p.getName()+"',"+p.getSalary()+",'"+p.getDesignation()+"')";
18. **return** template.update(sql);
19. }
20. **public** **int** update(Emp p){
21. String sql="update Emp99 set name='"+p.getName()+"', salary="+p.getSalary()+",designation='"+p.getDesignation()+"' where id="+p.getId()+"";
22. **return** template.update(sql);
23. }
24. **public** **int** delete(**int** id){
25. String sql="delete from Emp99 where id="+id+"";
26. **return** template.update(sql);
27. }
28. **public** Emp getEmpById(**int** id){
29. String sql="select \* from Emp99 where id=?";
30. **return** template.queryForObject(sql, **new** Object[]{id},**new** BeanPropertyRowMapper<Emp>(Emp.**class**));
31. }
32. **public** List<Emp> getEmployees(){
33. **return** template.query("select \* from Emp99",**new** RowMapper<Emp>(){
34. **public** Emp mapRow(ResultSet rs, **int** row) **throws** SQLException {
35. Emp e=**new** Emp();
36. e.setId(rs.getInt(1));
37. e.setName(rs.getString(2));
38. e.setSalary(rs.getFloat(3));
39. e.setDesignation(rs.getString(4));
40. **return** e;
41. }
42. });
43. }
44. }

### 5. Provide the entry of controller in the web.xml file

**web.xml**

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd" id="WebApp\_ID" version="3.0"**>**
3. **<display-name>**SpringMVC**</display-name>**
4. **<servlet>**
5. **<servlet-name>**spring**</servlet-name>**
6. **<servlet-class>**org.springframework.web.servlet.DispatcherServlet**</servlet-class>**
7. **<load-on-startup>**1**</load-on-startup>**
8. **</servlet>**
9. **<servlet-mapping>**
10. **<servlet-name>**spring**</servlet-name>**
11. **<url-pattern>**/**</url-pattern>**
12. **</servlet-mapping>**
13. **</web-app>**

### 6. Define the bean in the xml file

**spring-servlet.xml**

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans** xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:context="http://www.springframework.org/schema/context"
5. xmlns:mvc="http://www.springframework.org/schema/mvc"
6. xsi:schemaLocation="
7. http://www.springframework.org/schema/beans
8. http://www.springframework.org/schema/beans/spring-beans.xsd
9. http://www.springframework.org/schema/context
10. http://www.springframework.org/schema/context/spring-context.xsd
11. http://www.springframework.org/schema/mvc
12. http://www.springframework.org/schema/mvc/spring-mvc.xsd"**>**
13. **<context:component-scan** base-package="com.javatpoint.controllers"**></context:component-scan>**
15. **<bean** class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>**
16. **<property** name="prefix" value="/WEB-INF/jsp/"**></property>**
17. **<property** name="suffix" value=".jsp"**></property>**
18. **</bean>**
20. **<bean** id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource"**>**
21. **<property** name="driverClassName" value="com.mysql.jdbc.Driver"**></property>**
22. **<property** name="url" value="jdbc:mysql://localhost:3306/test"**></property>**
23. **<property** name="username" value=""**></property>**
24. **<property** name="password" value=""**></property>**
25. **</bean>**
27. **<bean** id="jt" class="org.springframework.jdbc.core.JdbcTemplate"**>**
28. **<property** name="dataSource" ref="ds"**></property>**
29. **</bean>**
31. **<bean** id="dao" class="com.javatpoint.dao.EmpDao"**>**
32. **<property** name="template" ref="jt"**></property>**
33. **</bean>**
34. **</beans>**

### 7. Create the requested page

**index.jsp**

1. **<a** href="empform"**>**Add Employee**</a>**
2. **<a** href="viewemp"**>**View Employees**</a>**

### 8. Create the other view components

**empform.jsp**

1. **<**%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%**>**
2. **<**%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%**>**
4. **<h1>**Add New Employee**</h1>**
5. **<form:form** method="post" action="save"**>**
6. **<table** **>**
7. **<tr>**
8. **<td>**Name : **</td>**
9. **<td><form:input** path="name"  **/></td>**
10. **</tr>**
11. **<tr>**
12. **<td>**Salary :**</td>**
13. **<td><form:input** path="salary" **/></td>**
14. **</tr>**
15. **<tr>**
16. **<td>**Designation :**</td>**
17. **<td><form:input** path="designation" **/></td>**
18. **</tr>**
19. **<tr>**
20. **<td>** **</td>**
21. **<td><input** type="submit" value="Save" **/></td>**
22. **</tr>**
23. **</table>**
24. **</form:form>**

**empeditform.jsp**

Here "/SpringMVCCRUDSimple" is the project name, change this if you have different project name. For live application, you can provide full URL.

1. **<**%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%**>**
2. **<**%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%**>**
4. **<h1>**Edit Employee**</h1>**
5. **<form:form** method="POST" action="/SpringMVCCRUDSimple/editsave"**>**
6. **<table** **>**
7. **<tr>**
8. **<td></td>**
9. **<td><form:hidden**  path="id" **/></td>**
10. **</tr>**
11. **<tr>**
12. **<td>**Name : **</td>**
13. **<td><form:input** path="name"  **/></td>**
14. **</tr>**
15. **<tr>**
16. **<td>**Salary :**</td>**
17. **<td><form:input** path="salary" **/></td>**
18. **</tr>**
19. **<tr>**
20. **<td>**Designation :**</td>**
21. **<td><form:input** path="designation" **/></td>**
22. **</tr>**
24. **<tr>**
25. **<td>** **</td>**
26. **<td><input** type="submit" value="Edit Save" **/></td>**
27. **</tr>**
28. **</table>**
29. **</form:form>**

**viewemp.jsp**

1. **<**%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%**>**
2. **<**%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%**>**
4. **<h1>**Employees List**</h1>**
5. **<table** border="2" width="70%" cellpadding="2"**>**
6. **<tr><th>**Id**</th><th>**Name**</th><th>**Salary**</th><th>**Designation**</th><th>**Edit**</th><th>**Delete**</th></tr>**
7. **<c:forEach** var="emp" items="${list}"**>**
8. **<tr>**
9. **<td>**${emp.id}**</td>**
10. **<td>**${emp.name}**</td>**
11. **<td>**${emp.salary}**</td>**
12. **<td>**${emp.designation}**</td>**
13. **<td><a** href="editemp/${emp.id}"**>**Edit**</a></td>**
14. **<td><a** href="deleteemp/${emp.id}"**>**Delete**</a></td>**
15. **</tr>**
16. **</c:forEach>**
17. **</table>**
18. **<br/>**
19. **<a** href="empform"**>**Add New Employee**</a>**