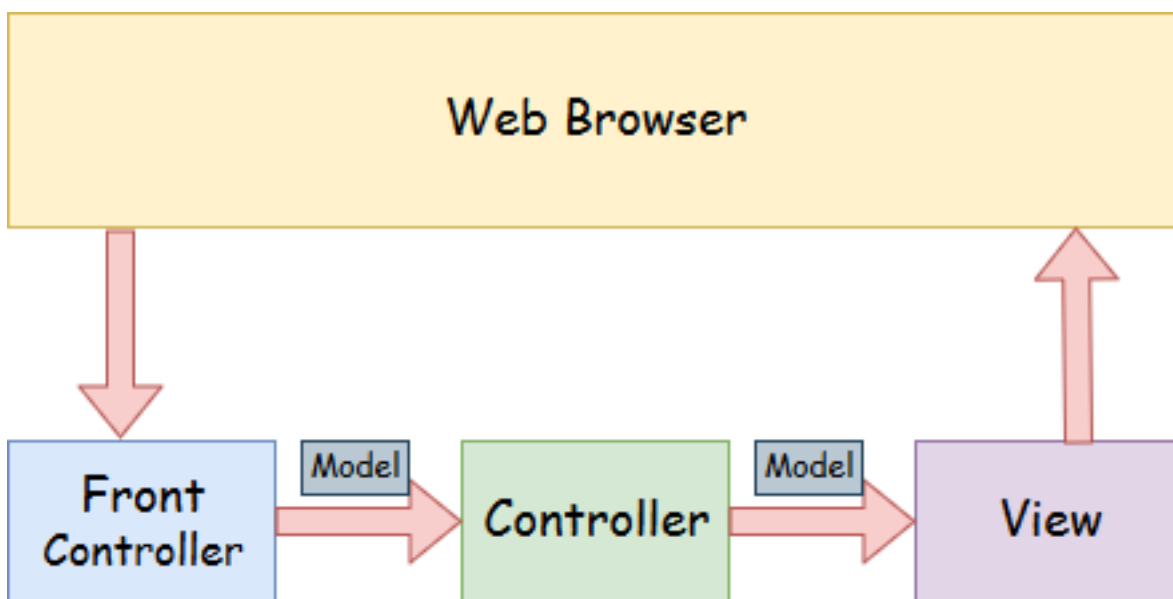


# Spring MVC

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.

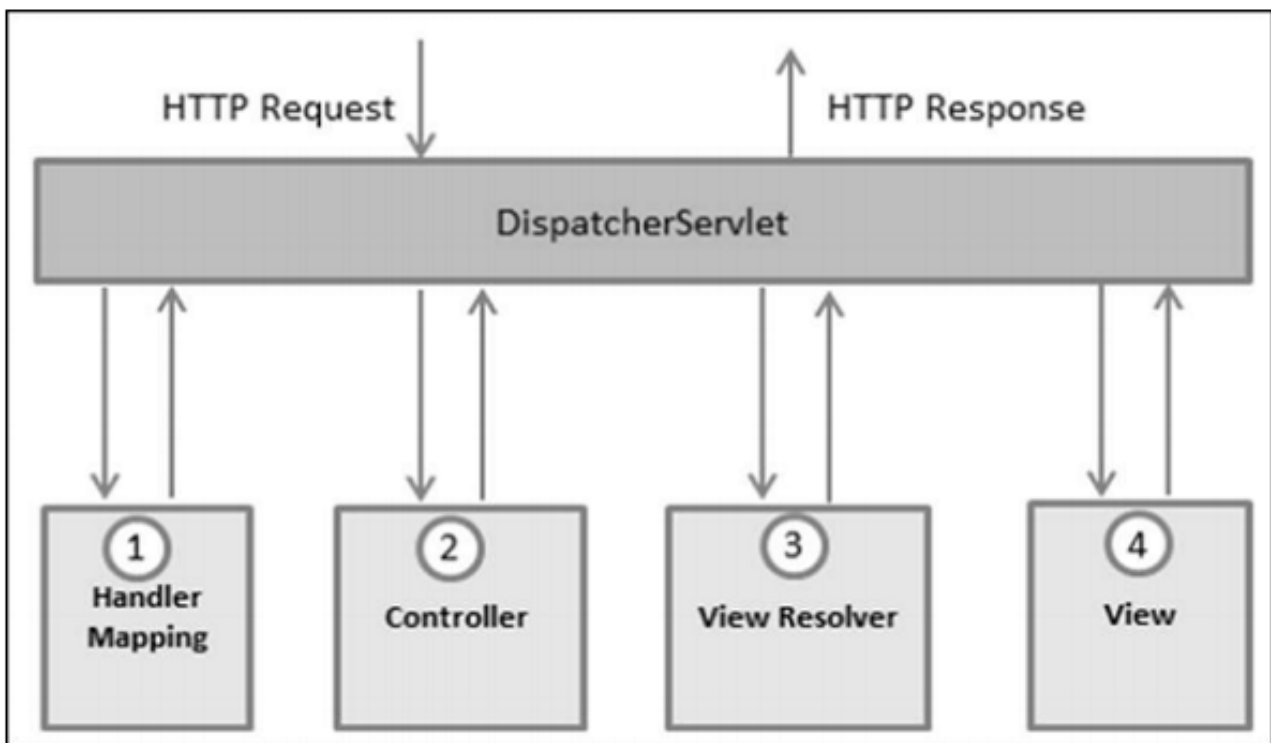
**Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.

**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.

## The DispatcherServlet

The Spring Web model-view-controller (MVC) framework is designed around a DispatcherServlet that handles all the HTTP requests and responses. The request processing workflow of the Spring Web MVC DispatcherServlet is shown in the following illustration.



Following is the sequence of events corresponding to an incoming HTTP request to DispatcherServlet –

After receiving an HTTP request, DispatcherServlet consults the **HandlerMapping** to call the appropriate Controller.

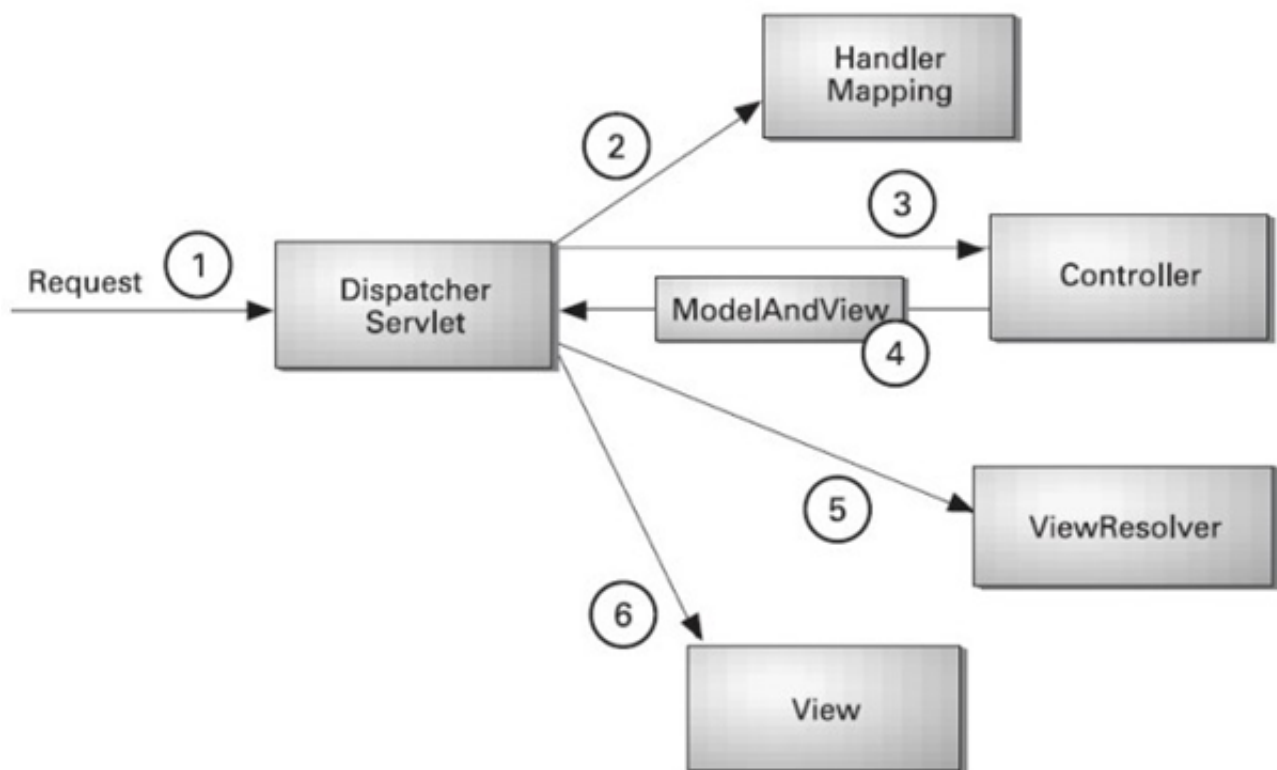
The Controller takes the request and calls the appropriate service methods based on used **GET** or **POST method**. The service method will set model data based on defined business logic and returns view name to the DispatcherServlet.

The DispatcherServlet will take help from **ViewResolver** to pick up the defined view for the request.

Once view is finalized, The DispatcherServlet passes the model data to the view, which is finally rendered, on the browsers.

All the above-mentioned components, i.e. HandlerMapping, Controller and ViewResolver are parts of **WebApplicationContext**, which is an extension of the plain **ApplicationContext** with some extra features necessary for web applications.

## 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

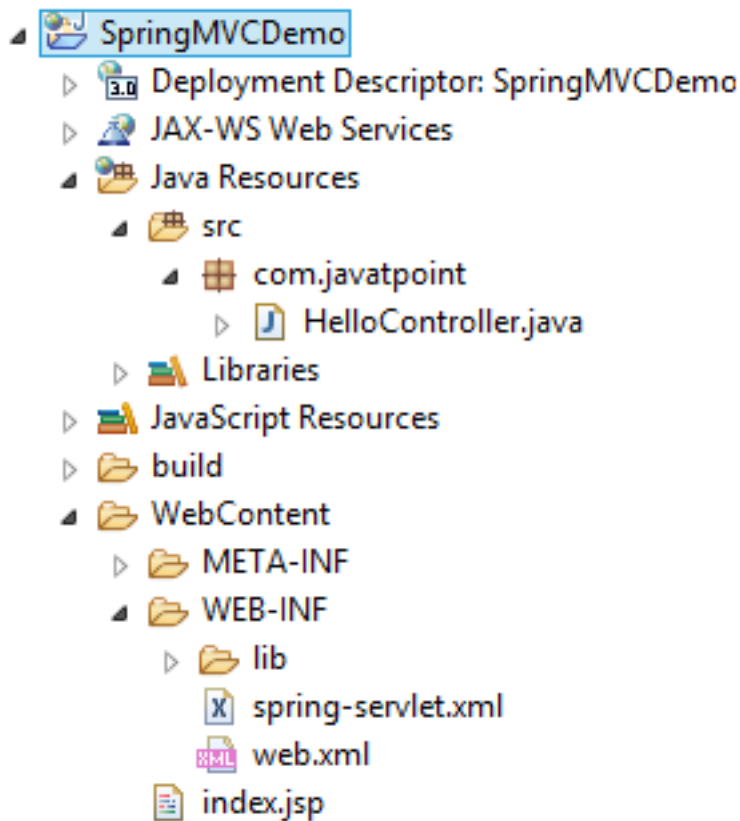
Provide the entry of controller in the web.xml file

Define the bean in the separate XML file

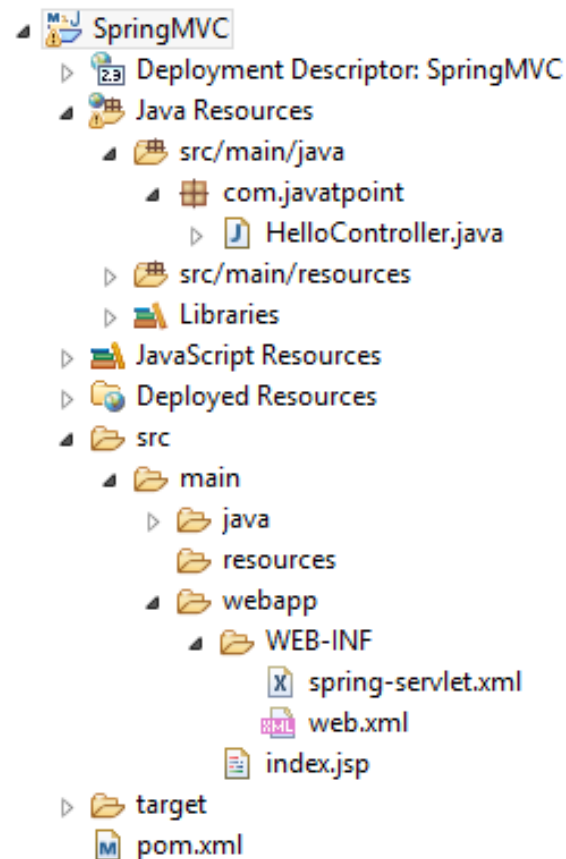
Display the message in the JSP page

Start the server and deploy the project

## Directory Structure of Spring MVC



## Directory Structure of Spring MVC using Maven



# Required Jar files or Maven Dependency

To run this example, you need to load:

Spring Core jar files

Spring Web jar files

JSP + JSTL jar files (If you are using any another view technology then load the corresponding jar files).

1. Provide project information and configuration in the pom.xml file.

**pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.springMVC</groupId>
  <artifactId>SpringMVC</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>SpringMVC Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
```

```

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>5.1.1.RELEASE</version>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>servlet-api</artifactId>
  <version>3.0-alpha-1</version>
</dependency>

</dependencies>
<build>
  <finalName>SpringMVC</finalName>
</build>
</project>

```

## 2. Create the controller class

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

## Defining a Controller

The DispatcherServlet delegates the request to the controllers to execute the functionality specific to it. The **@Controller** annotation indicates that a particular class serves the role of a controller. The **@RequestMapping** annotation is used to map a URL to either an entire class or a particular handler method.

The @Controller annotation marks this class as Controller.

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

The **@Controller** annotation defines the class as a Spring MVC controller. Here, the first usage of **@RequestMapping** indicates that all handling methods on this controller are relative to the **/hello** path.

The next annotation **@RequestMapping (method = RequestMethod.GET)** is used to declare the **printHello()** method as the controller's default service method to handle HTTP GET request. We can define another method to handle any POST request at the same URL.

### HelloController.java

```
package com.springMVC;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
;
@Controller
public class HelloController {
    @RequestMapping("/")
    public String display()
    {
        return "index";
    }
}
```

### 3. 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

```
<?xml version="1.0" encoding="UTF-8"?>
<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">
  <display-name>SpringMVC</display-name>
  <servlet>
    <servlet-name>spring</servlet-name>
    <servlet-
class>org.springframework.web.servlet.DispatcherServlet</
servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>

```

## 4. 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

```

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

```
    <!-- Provide support for component scanning -->  
    <context:component-scan base-package="com.springMVC" /  
>  
  
    <!--Provide support for conversion, formatting and validation --  
>  
    <mvc:annotation-driven/>
```

```
</beans>
```

## 5. Display the message in the JSP page

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

## Creating JSP Views

Spring MVC supports many types of views for different presentation technologies. These include - **JSPs, HTML, PDF, Excel Worksheets, XML, Velocity Templates, XSLT, JSON, Atom** and **RSS** feeds, **JasperReports**, etc. However, the most common ones are the JSP templates written with JSTL

Here **`${message}`** Here is the attribute, which we have setup inside the Controller. You can have multiple attributes to be displayed inside your view.

**index.jsp**

```
<html>  
<body>  
<p>Welcome to Spring MVC Tutorial</p>  
</body>  
</html>
```

# Spring MVC Multiple View page 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
- Create the other view components
- Start the server and deploy the project

Step 1: create a project named as SpringMVCMultipleViewPage and check the tomcat server version in runtime

step2: Check Generate web.xml and click the finish button

step 3: Right click the project and click convert to Maven

step 4: GroupId: com.springmultiple  
ArtifactId: spring-multiple-mvc

Name: SpringMultiplePage

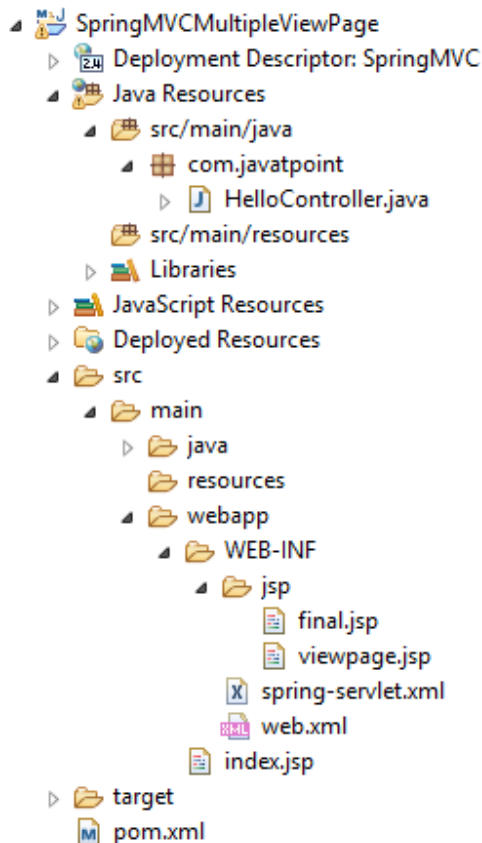
Description: Spring Multiple Page Example

step 5: Generate HelloController.java class file inside the folder com.springmultiple

step 6: Generate index.jsp and spring-servlet.xml

step 7: Generate viewpage.jsp and final.jsp inside the folder jsp in WEB-INF

# Directory Structure of Spring MVC



## 1. Add dependencies to pom.xml

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
```

```
<dependency>
```

```
  <groupId>org.springframework</groupId>
```

```
  <artifactId>spring-webmvc</artifactId>
```

```
  <version>5.1.1.RELEASE</version>
```

```
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
```

```
<dependency>
```

```
  <groupId>javax.servlet</groupId>
```

```
  <artifactId>servlet-api</artifactId>
```

```
  <version>3.0-alpha-1</version>
```

`</dependency>`

## 2. Create the request page

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

**index.jsp**

```
<html>
<body>
<a href="hello">Click here...</a>
</body>
</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**

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
;
@Controller
public class HelloController {
    @RequestMapping("/hello")
    public String redirect()
    {
        return "viewpage";
    }
    @RequestMapping("/helloagain")
    public String display()
    {
        return "final";
    }
}
```

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

## web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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">
  <display-name>SpringMVC</display-name>
  <servlet>
    <servlet-name>spring</servlet-name>
    <servlet-
class>org.springframework.web.servlet.DispatcherServlet</
servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</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

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

```

xmlns:context="http://www.springframework.org/schema/
context"
xmlns:mvc="http://www.springframework.org/schema/mvc"
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">

```

```

<!-- Provide support for component scanning -->
<context:component-scan base-package="com.springMVC" /
>

<!--Provide support for conversion, formatting and validation --
>
<mvc:annotation-driven/>
<!-- Define Spring MVC view resolver -->
<bean id="viewResolver" class="org.springframework.web.servlet.
view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/"></
property>
    <property name="suffix" value=".jsp"></property>
</bean>
</beans>

```

## 6. Create the other view components

viewpage.jsp

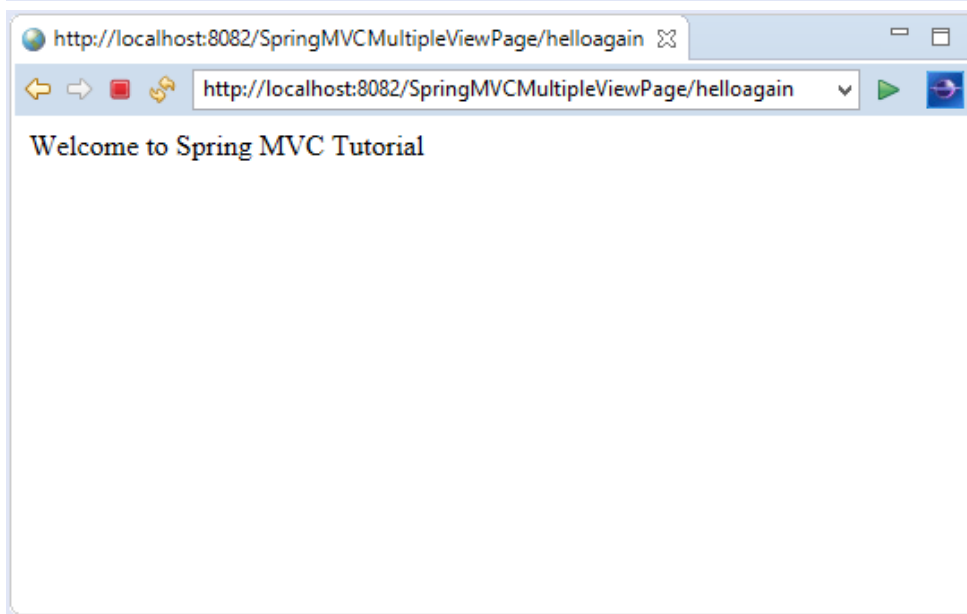
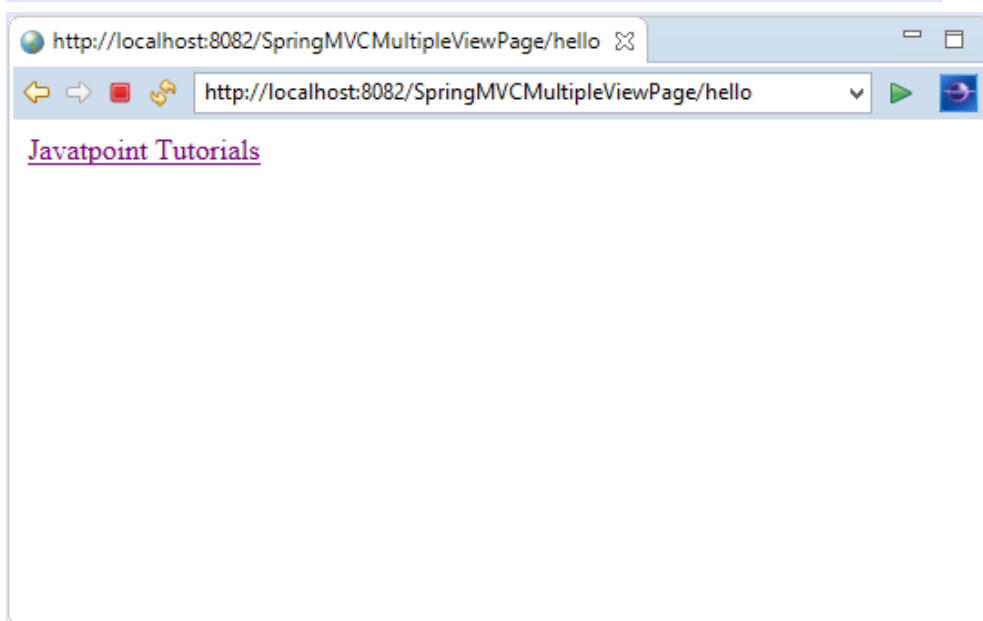
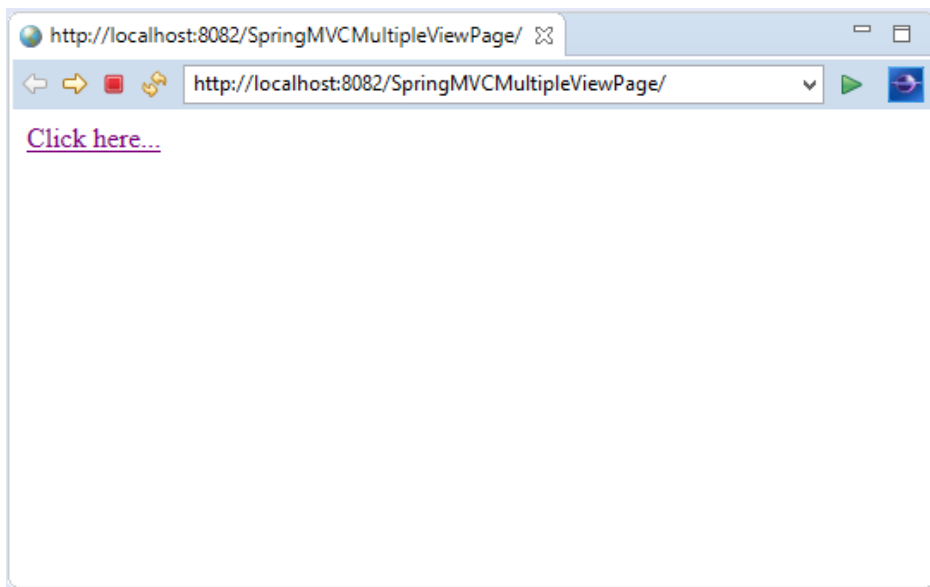
```

<html>
<body>
<a href="helloagain">Javatpoint Tutorials</a>
</body>
</html>
final.jsp

```

```
<html>
<body>
<p>Welcome to Spring MVC Tutorial</p>
</body>
</html>
```





# 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

```
<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:

```
<form:input type=?email? path="email" />
```

## For date:

```
<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.

```
<!-- https://mvnrepository.com/artifact/org.springframework/
spring-webmvc -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>5.1.1.RELEASE</version>
</dependency>
```

```

        <!-- https://mvnrepository.com/artifact/javax.servlet/
javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>3.0-alpha-1</version>
</dependency>

    <!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-
jasper -->
<dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-jasper</artifactId>
    <version>9.0.12</version>
</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

```

public class Reservation {

    private String firstName;
    private String lastName;

    public Reservation()
    {
    }

    public String getFirstName() {
        return firstName;
    }
}

```

```

    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}

```

### 3. Create the controller class

#### ReservationController.java

```

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

@RequestMapping("/reservation")
@Controller
public class ReservationController {
    @RequestMapping("/bookingForm")
    public String bookingForm(Model model)
    {
        //create a reservation object
        Reservation res=new Reservation();
        //provide reservation object to the model
        model.addAttribute("reservation", res);
        return "reservation-page";
    }
    @RequestMapping("/submitForm")
    // @ModelAttribute binds form data to the object
    public String submitForm(@ModelAttribute("reservation") Reservati
on res)
    {
        return "confirmation-form";
    }
}

```

```
}  
}  
}
```

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

##### web.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<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">  
  <display-name>SpringMVC</display-name>  
  <servlet>  
    <servlet-name>spring</servlet-name>  
                                <servlet-  
class>org.springframework.web.servlet.DispatcherServlet</  
servlet-class>  
    <load-on-startup>1</load-on-startup>  
  </servlet>  
  <servlet-mapping>  
    <servlet-name>spring</servlet-name>  
    <url-pattern>/</url-pattern>  
  </servlet-mapping>  
</web-app>
```

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

##### spring-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xmlns:context="http://www.springframework.org/schema/  
context"  
  xmlns:mvc="http://www.springframework.org/schema/mvc"  
  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">
    <!-- Provide support for component scanning -->
    <context:component-scan base-package="com.springMVC" /
>
    <!--Provide support for conversion, formatting and validation --
>
    <mvc:annotation-driven/>
    <!-- Define Spring MVC view resolver -->
    <bean id="viewResolver" class="org.springframework.web.serv
let.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"></
property>
        <property name="suffix" value=".jsp"></property>
    </bean>
</beans>

```

## 6. Create the requested page

index.jsp

```

<!DOCTYPE html>
<html>
<head>
    <title>Railway Registration Form</title>
</head>
<body>
<a href="reservation/bookingForm">Click here for reservation.</
a>
</body>
</html>

```

## 7. Create other view components

## reservation-page.jsp

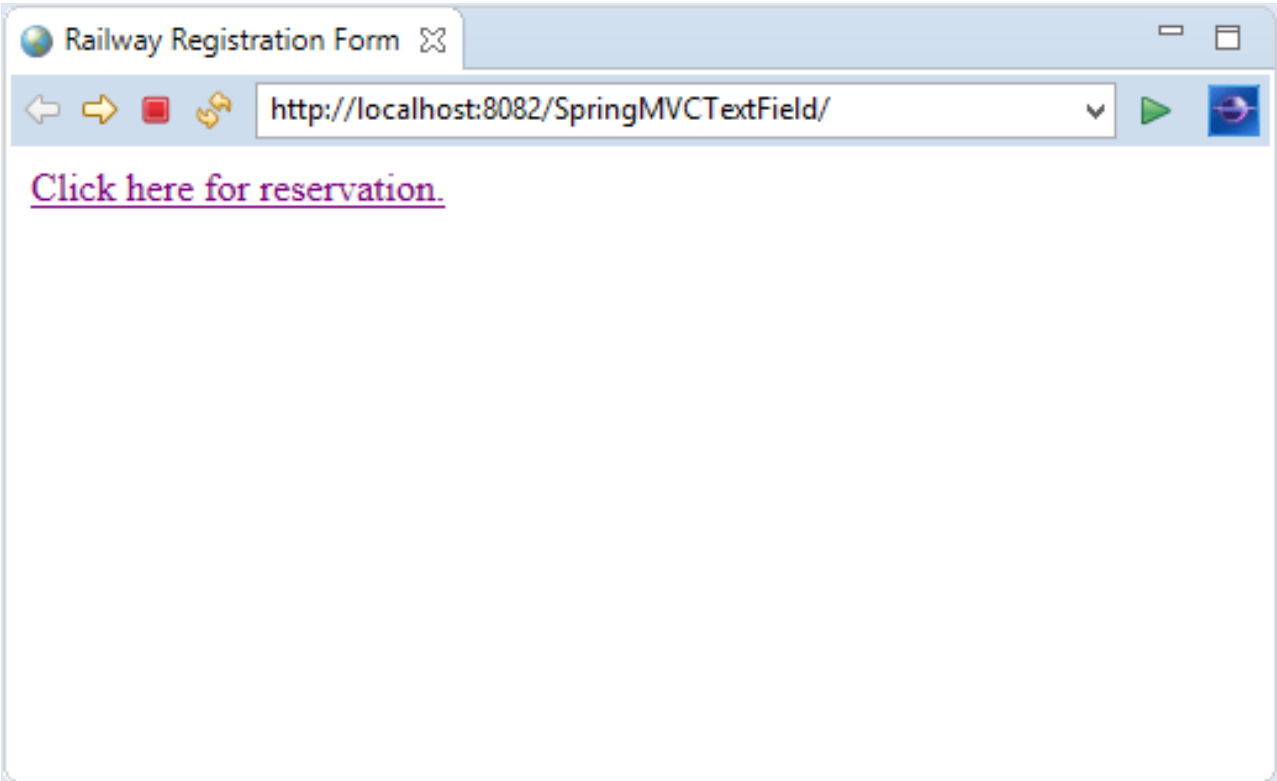
```
<%@ taglib prefix="form" uri="http://www.springframework.org/
tags/form" %>
<!DOCTYPE html>
<html>
<head>
  <title>Reservation Form</title>
</head>
<h3>Railway Reservation Form</h3>
<body>
  <form:form action="submitForm" modelAttribute="reservation"
>
  First name: <form:input path="firstName" />
  <br><br>
  Last name: <form:input path="lastName" />
  <br><br>
  <input type="submit" value="Submit" />
  </form:form>
</body>
</html>
```

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

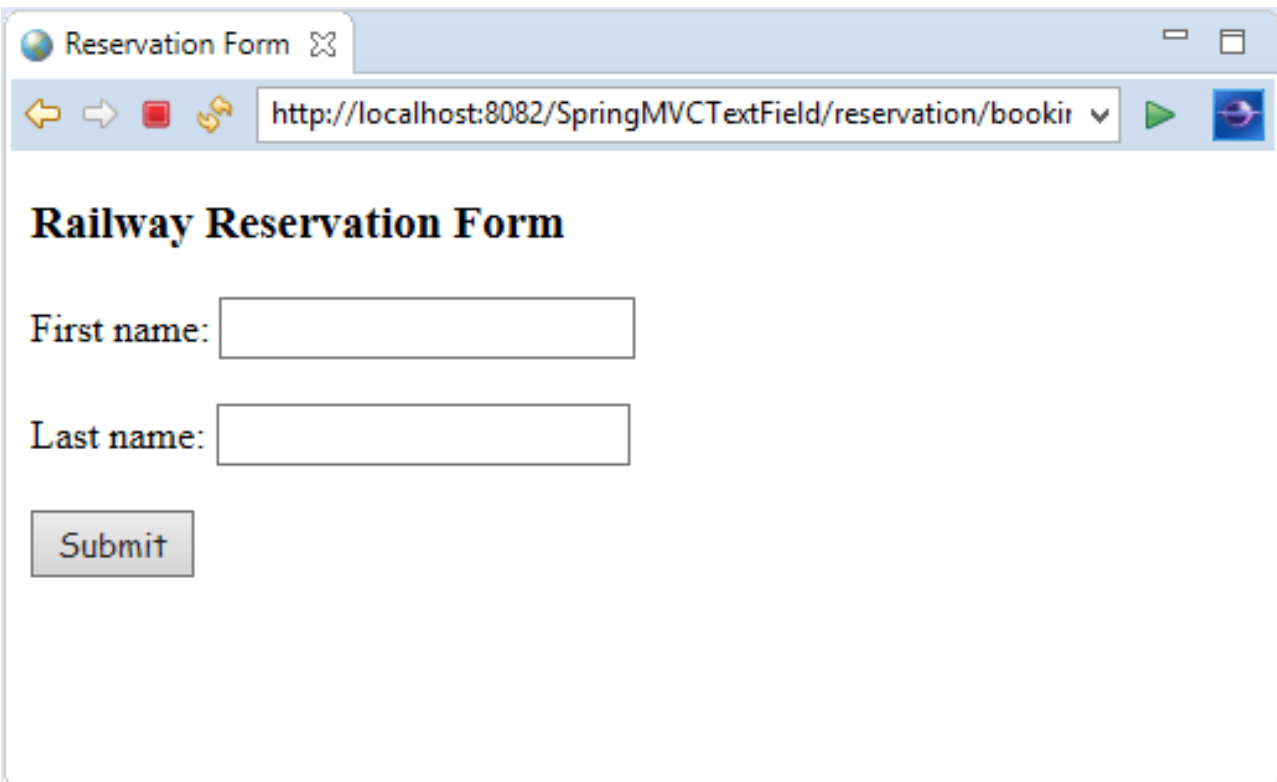
## confirmation-page.jsp

```
<!DOCTYPE html>
<html>
<body>
<p>Your reservation is confirmed successfully. Please, re-
check the details.</p>
First Name : ${reservation.firstName} <br>
Last Name : ${reservation.lastName}
</body>
</html>
```

## Output:



The screenshot shows a web browser window with the title "Railway Registration Form". The address bar displays the URL "http://localhost:8082/SpringMVCTextField/". The main content area contains a single link: [Click here for reservation.](#)



The screenshot shows a web browser window with the title "Reservation Form". The address bar displays the URL "http://localhost:8082/SpringMVCTextField/reservation/bookir". The main content area displays the "Railway Reservation Form" with the following fields and button:

**Railway Reservation Form**

First name:

Last name:



The image displays two sequential browser window screenshots. The top window, titled 'Reservation Form', shows a web page with the heading 'Railway Reservation Form'. It contains two text input fields: 'First name:' with the value 'Gaurav' and 'Last name:' with the value 'Chawla'. A 'Submit' button is positioned below these fields. The browser's address bar shows the URL 'http://localhost:8082/SpringMVCTextField/reservation/bookir'. The bottom window shows the result of the submission. The address bar now points to 'http://localhost:8082/SpringMVCTextField/reservation/submitForm'. The page content displays a confirmation message: 'Your reservation is confirmed successfully. Please, re-check the details.' followed by the submitted details: 'First Name : Gaurav' and 'Last Name : Chawla'.

Reservation Form

**Railway Reservation Form**

First name:

Last name:

http://localhost:8082/SpringMVCTextField/reservation/submitForm

Your reservation is confirmed successfully. Please, re-check the details.

First Name : Gaurav  
Last Name : Chawla

## 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.

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER	No	-	1
NAME	VARCHAR2(4000)	Yes	-	-
SALARY	NUMBER	Yes	-	-
DESIGNATION	VARCHAR2(4000)	Yes	-	-
				1 - 4

## Spring MVC CRUD Example

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

**pom.xml**

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
```

```
<dependency>
```

```
    <groupId>org.springframework</groupId>
```

```
    <artifactId>spring-webmvc</artifactId>
```

```
    <version>5.1.1.RELEASE</version>
```

```
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jasper -->
```

```
<dependency>
```

```
    <groupId>org.apache.tomcat</groupId>
```

```

    <artifactId>tomcat-jasper</artifactId>
    <version>9.0.12</version>
</dependency>
    <!-- https://mvnrepository.com/artifact/javax.servlet/
javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>3.0-alpha-1</version>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
    <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-
java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.11</version>
</dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/
spring-jdbc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.1.1.RELEASE</version>
</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

```
package com.springMVC.beans;

public class Emp {
private int id;
private String name;
private float salary;
private String designation;

public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public float getSalary() {
    return salary;
}
public void setSalary(float salary) {
    this.salary = salary;
}
public String getDesignation() {
    return designation;
}
public void setDesignation(String designation) {
    this.designation = designation;
}

}
```

## 3. Create the controller class

## EmpController.java

```
package com.springMVC.controllers;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;

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

import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping
;
import org.springframework.web.bind.annotation.RequestMethod;

import com.springMVC.beans.Emp;
import com.springMVC.dao.EmpDao;
@Controller
public class EmpController {
    @Autowired
    EmpDao dao;//will inject dao from XML file

    /
    *It displays a form to input data, here "command" is a reserved req
    uest attribute
    *which is used to display object data into form
    */
    @RequestMapping("/empform")
    public String showform(Model m){
        m.addAttribute("command", new Emp());
        return "empform";
    }

    /
    *It saves object into database. The @ModelAttribute puts request d
    ata
    * into model object. You need to mention RequestMethod.POS
    T method
    * because default request is GET*/
}
```

```

        @RequestMapping(value = "/
save",method = RequestMethod.POST)
    public String save(@ModelAttribute("emp") Emp emp){
        dao.save(emp);
        return "redirect:/viewemp";//
will redirect to viewemp request mapping
    }
    /* It provides list of employees in model object */
    @RequestMapping("/viewemp")
    public String viewemp(Model m){
        List<Emp> list=dao.getEmployees();
        m.addAttribute("list",list);
        return "viewemp";
    }
    /* It displays object data into form for the given id.
    * The @PathVariable puts URL data into variable.*/
    @RequestMapping(value="/editemp/{id}")
    public String edit(@PathVariable int id, Model m){
        Emp emp=dao.getEmpById(id);
        m.addAttribute("command",emp);
        return "empeditform";
    }
    /* It updates model object. */
        @RequestMapping(value = "/
editsave",method = RequestMethod.POST)
    public String editsave(@ModelAttribute("emp") Emp emp){
        dao.update(emp);
        return "redirect:/viewemp";
    }
    /* It deletes record for the given id in URL and redirects to /
viewemp */
        @RequestMapping(value = "/deleteemp/
{id}",method = RequestMethod.GET)
    public String delete(@PathVariable int id){
        dao.delete(id);
        return "redirect:/viewemp";
    }
}

```

#### 4. Create the DAO class

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

## EmpDao.java

```
package com.springMVC.dao;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import org.springframework.jdbc.core.BeanPropertyRowMapper;

import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import com.springMVC.beans.Emp;

public class EmpDao {
    JdbcTemplate template;

    public void setTemplate(JdbcTemplate template) {
        this.template = template;
    }

    public int save(Emp p){
        String sql="insert into Emp99(name,salary,designation) values('"
        +p.getName()+"'," +p.getSalary()+"'," +p.getDesignation()+"')";
        return template.update(sql);
    }

    public int update(Emp p){
        String sql="update Emp99 set name='"+p.getName()
        +"', salary='"+p.getSalary()+"',designation='"+p.getDesignation()
        +"' where id="+p.getId()+"";
        return template.update(sql);
    }

    public int delete(int id){
        String sql="delete from Emp99 where id="+id+"";
        return template.update(sql);
    }

    public Emp getEmpById(int id){
        String sql="select * from Emp99 where id=?";
```

```

        return template.queryForObject(sql, new Object[]
{id},new BeanPropertyRowMapper<Emp>(Emp.class));
    }
    public List<Emp> getEmployees(){
        return template.query("select * from Emp99",new RowMapper<
Emp>(){
            public Emp mapRow(ResultSet rs, int row) throws SQLExcep
tion {
                Emp e=new Emp();
                e.setld(rs.getInt(1));
                e.setName(rs.getString(2));
                e.setSalary(rs.getFloat(3));
                e.setDesignation(rs.getString(4));
                return e;
            }
        });
    }
}

```

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

### web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<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">
    <display-name>SpringMVC</display-name>
    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</
servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>

```



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

## 6. Define the bean in the xml file

### spring-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/
context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       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">
  <context:component-scan base-
package="com.springMVC.controllers"></context:component-
scan>

  <bean class="org.springframework.web.servlet.view.InternalResou
rceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/"></property>
    <property name="suffix" value=".jsp"></property>
  </bean>

  <bean id="ds" class="org.springframework.jdbc.datasource.Driver
ManagerDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Drive
r"></property>
```

```
<property name="url" value="jdbc:mysql://localhost:3306/test"></property>
<property name="username" value=""></property>
<property name="password" value=""></property>
</bean>
```

```
<bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate"
">
<property name="dataSource" ref="ds"></property>
</bean>
```

```
<bean id="dao" class="com.springMVC.dao.EmpDao">
<property name="template" ref="jt"></property>
</bean>
</beans>
```

## 7. Create the requested page

index.jsp

```
<a href="empform">Add Employee</a>
<a href="viewemp">View Employees</a>
```

## 8. Create the other view components

empform.jsp

```
<%@ taglib uri="http://www.springframework.org/tags/
form" prefix="form"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

```
    <h1>Add New Employee</h1>
    <form:form method="post" action="save">
    <table >
    <tr>
    <td>Name : </td>
    <td><form:input path="name" /></td>
    </tr>
    <tr>
    <td>Salary :</td>
```

```

        <td><form:input path="salary" /></td>
    </tr>
    <tr>
        <td>Designation :</td>
        <td><form:input path="designation" /></td>
    </tr>
    <tr>
        <td> </td>
        <td><input type="submit" value="Save" /></td>
    </tr>
</table>
</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.

```

<%@ taglib uri="http://www.springframework.org/tags/
form" prefix="form"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

```

```

    <h1>Edit Employee</h1>
    <form:form method="POST" action="/
SpringMVCCRUDSimple/editsave">
    <table >
    <tr>
    <td></td>
    <td><form:hidden path="id" /></td>
    </tr>
    <tr>
    <td>Name : </td>
    <td><form:input path="name" /></td>
    </tr>
    <tr>
    <td>Salary :</td>
    <td><form:input path="salary" /></td>
    </tr>

```

```

<tr>
  <td>Designation :</td>
  <td><form:input path="designation" /></td>
</tr>

<tr>
  <td> </td>
  <td><input type="submit" value="Edit Save" /></td>
</tr>
</table>
</form:form>

```

viewemp.jsp

```

<%@ taglib uri="http://www.springframework.org/tags/
form" prefix="form"%>
  <%@ taglib uri="http://java.sun.com/jsp/jstl/
core" prefix="c"%>

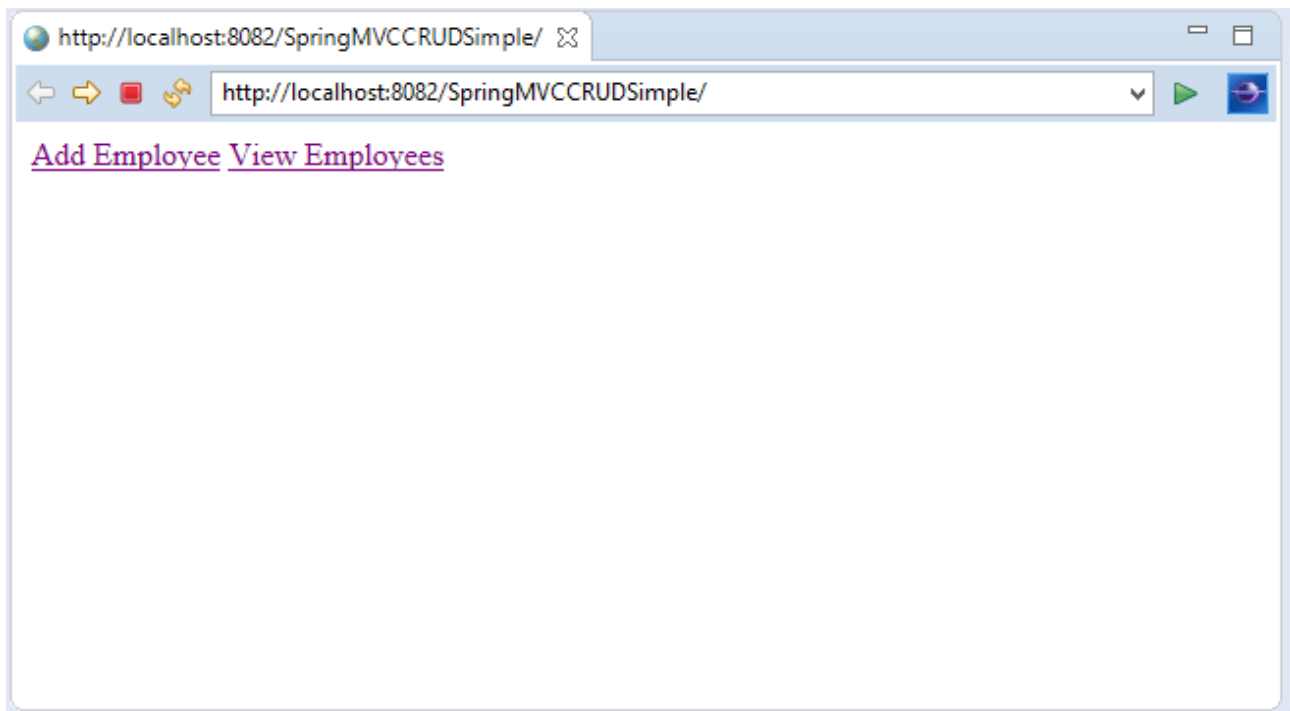
```

```

<h1>Employees List</h1>
<table border="2" width="70%" cellpadding="2">
<tr><th>Id</th><th>Name</th><th>Salary</
th><th>Designation</th><th>Edit</th><th>Delete</th></tr>
  <c:forEach var="emp" items="${list}">
    <tr>
      <td>${emp.id}</td>
      <td>${emp.name}</td>
      <td>${emp.salary}</td>
      <td>${emp.designation}</td>
      <td><a href="editemp/${emp.id}">Edit</a></td>
      <td><a href="deleteemp/${emp.id}">Delete</a></td>
    </tr>
  </c:forEach>
</table>
<br/>
<a href="empform">Add New Employee</a>

```

Output:



On clicking **Add Employee**, you will see the following form.

A screenshot of a web browser window showing the 'Add New Employee' form. The address bar shows the URL 'http://localhost:8082/SpringMVCCRUDSimple/empform'. The form has a title 'Add New Employee' in a large, bold, black serif font. Below the title, there are three input fields: 'Name' with the value 'Sonoo Jaiswal', 'Salary' with the value '85000', and 'Designation' with the value 'Team Leader'. Below the 'Designation' field, there is a 'Save' button.

Fill the form and **click Save** to add the entry into the database.

http://localhost:8082/SpringMVCCRUDSimple/viewemp

http://localhost:8082/SpringMVCCRUDSimple/viewemp

## Employees List

<b>ID</b>	<b>Name</b>	<b>Salary</b>	<b>Designation</b>	<b>Edit</b>	<b>Delete</b>
8	Sonoo Jaiswal	85000.0	Team Leader	<a href="#">Edit</a>	<a href="#">Delete</a>
9	Ashutosh Kumar	50000.0	Enginner	<a href="#">Edit</a>	<a href="#">Delete</a>
10	Yash Tyagi	25000.0	Data Entry	<a href="#">Edit</a>	<a href="#">Delete</a>
11	John William	75000.0	Manager	<a href="#">Edit</a>	<a href="#">Delete</a>

[Add New Employee](#)