

Spring MVC.

Create spring boot project with dependencies web, dev tools , JPA

// This is for integration with hibernate

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

// This is for spring core

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

// This is for integration of jstl in jsp pages.

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

// This is for hibernate + mysql

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.6</version>
</dependency>
```

// This is for jsp engine for compiling and running a jsp

```
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-jasper</artifactId>
  <version>9.0.45</version>
</dependency>
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

// This is for developer help for deploying and restarting application automatically

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
  <optional>true</optional>
</dependency>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<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/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.6.1</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.javabykiran</groupId>
  <artifactId>springmvcexample</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>springmvcexample</name>
  <description>Demo project for Spring Boot MVC</description>
```

```
<properties>
  <java.version>1.8</java.version>
</properties>
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.6</version>
  </dependency>
  <dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-jasper</artifactId>
    <version>9.0.45</version>
  </dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

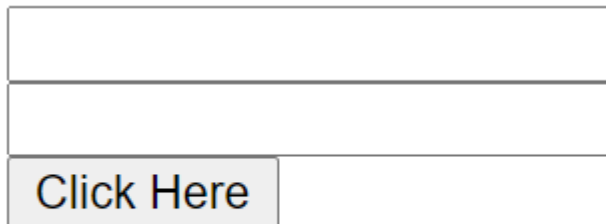
```
</project>
```

If you have the static welcome page (**index.html**) for the application, put in under the **src/main/resources/static** directory. For example, create the **index.html** file with the following content:

```
<form action="login">
  <input type="text" name="uname"/><br>
  <input type="text" name="passwd"/><br>
  <input type="submit" value="Click Here"/><br>
</form>
```

Now open a browser

<http://localhost:8989/>



Click Here

*Till now we have
created setup for
spring MVC. Now it's
time to customize flow.*

javabyKiran
java | selenium | python

**1) First write controller and change default page navigation with “
/testjsp ”**

```
package com.javabykiran;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;  
  
@Controller  
  
public class StudentController {  
  
    @RequestMapping("/testjsp")  
  
    public String welcome() {  
        return "index.html";  
    }  
}
```

Now open <http://localhost:8989/testjsp>

Same page will be opening.

How to use JSP with Spring Boot

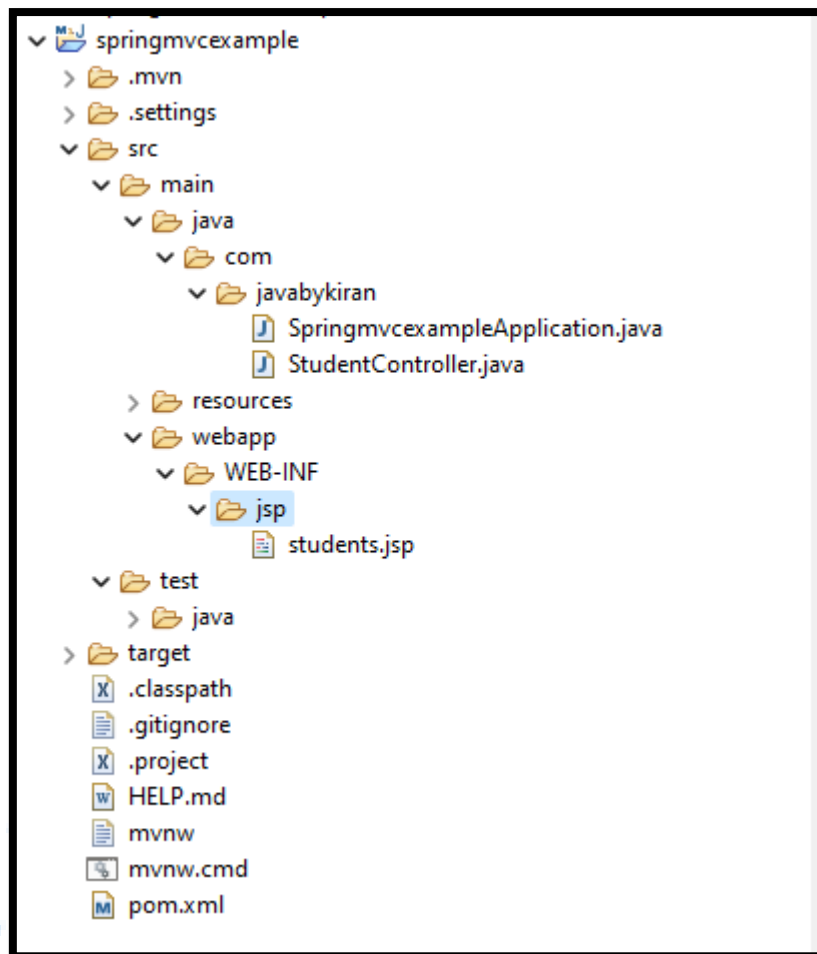
We will create JSP page with JSTL to display the contact list

The dependency **tomcat-embed-jasper** is JSP Engine for Tomcat. It is required to enable JSP with Spring Boot.

By default, JSP pages are looked up inside **/webapp directory.** So under the **src/main** directory, create a new directory named webapp. If no view resolvers are configured, a handler method should return the actual view file name,

for example: Go to navigator so that its easy to create folders.

To follow common convention and to protect JSP pages, create directories **WEB-INF/jsp** inside **webapp**. And create contact.jsp file under **src/main/webapp/WEB-INF/jsp** as shown in the following screenshot:



As we do not use here by default webapp directory for placing jsps we need to configure view resolver in this case.

Go to application.properties and configure as below.

```
server.port=8989
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
```


In controller create method for returning all students.

Make sure you have Student class with rollnumber, and name is created. / getter setter/ constructor should be present

Add below code in controller

```
@RequestMapping("/allstudents")
    public ModelAndView fetchStudents() {

        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("students");
        ArrayList<Student> list = new ArrayList<Student>();
        list.add(new Student(123, "jbk"));
        list.add(new Student(345, "jbkiran"));
        list.add(new Student(567, "javabykiran"));
        modelAndView.addObject("stulist", list);
        return modelAndView;
    }
```

Now we have to write code in jsp

```
<% @ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"% >
<% @ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"% >
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Contact List - Spring Boot Web Application Example</title>
</head>
<body>
```

```
<h1 align="center">My Student List</h1>
<br />
<table border="1" cellpadding="10">
  <tr>
    <th>Roll Number</th>
    <th>Name</th>
  </tr>
  <c:forEach var="student" items="${stulist}">
    <tr>
      <td>${student.rollNumber}</td>
      <td>${student.name}</td>
    </tr>
  </c:forEach>
</table>
</body>
</html>
```

As you can see, this JSP page uses JSTL's `forEach` tag to iterate over Student objects in the collection named `stulist` in the model, and generate HTML code to display the contact list.

Stop and re-run the main program. Once you opens a browser

<http://localhost:8989/allstudents>

My Student List

| Roll Number | Name |
|-------------|-------------|
| 123 | jbk |
| 345 | jbkiran |
| 567 | javabykiran |

Explanation of JSTL Code from jsp

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

This is tag library we have used here to get predefined features such as for loop. Same prefix we will be using in below code. We can change this.

```
<c:forEach var="student" items="${stulist}">

    <tr>

        <td>${student.rollNumber}</td>

        <td>${student.name}</td>

    </tr>

</c:forEach>
```

Stulist - > key in controller against which data is stored.

var="student" -- > This is a variable we will use to retrieve data.it can be anything

```
<td>${student.rollNumber}</td>
```

```
<td>${student.name}</td>
```

These lines because of variable names. Here student is not a class name but it is var name while defining for loop

Homework:

1. Use hibernate in controller to fetch data.
2. Do add update delete operation from jsp
3. Separate Database code from controller to other class and annotate that class with @Repository and autowire that class in controller to call method.
4. Separate session factory code from the configuration file that is starter class to some other class and annotate that class with @Configuration.
5. Above 2 points will separate code between 2 layers.