Spring MVC - Custom Validation

Last Updated : 23 Jul, 2025

Validating user input is essential for any web application to ensure the processing of valid data. The **Spring MVC framework** supports the use of **validation** API. The validation API puts constraints on the user input using annotations and can validate both client-side and server-side. It provides standard predefined validators like @**Min**, @**Max**, @**Size**, @**Pattern**, and @**NotNull**. In case we have to process a more specific input, spring MVC also provides the concept of user-defined **validators** with custom validation logic.

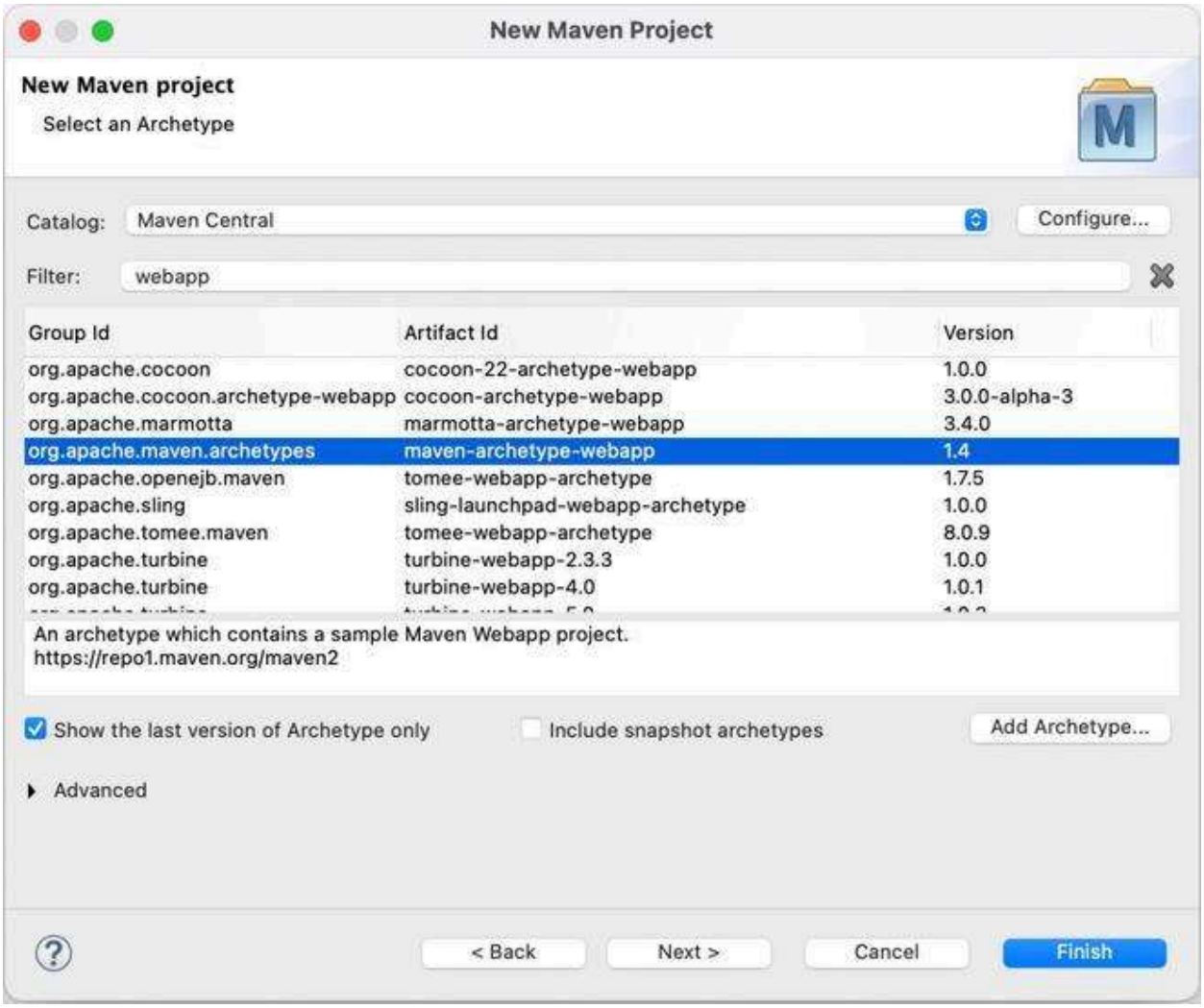
Prerequisites: Spring MVC, Spring MVC Validation

Setting Up the Project

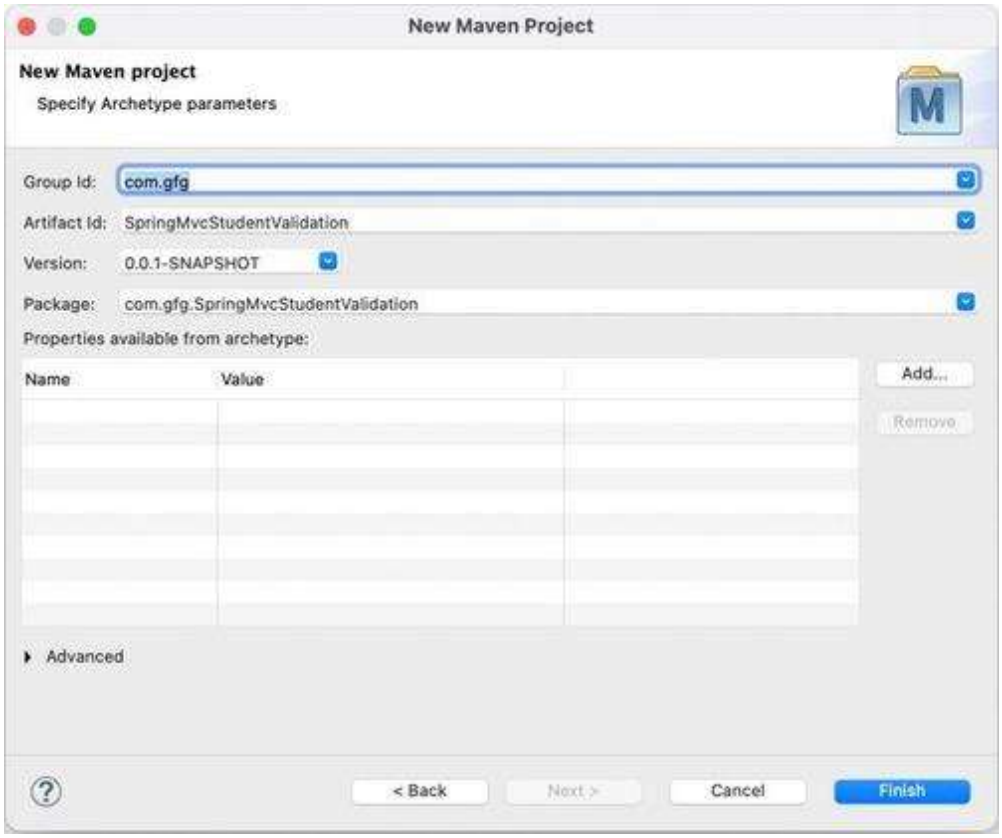
Now, we'll create a custom validator to validate the address of a student in a student portal.

Step 1: Create a Maven Web Application Project

Firstly, we need to create a maven web app project. In this tutorial, we'll use **Eclipse IDE.** Now, choose to create a maven while creating a new project and add a **maven webapp archetype**. Enter the GroupId and the ArtifactId for your project and click Finish.

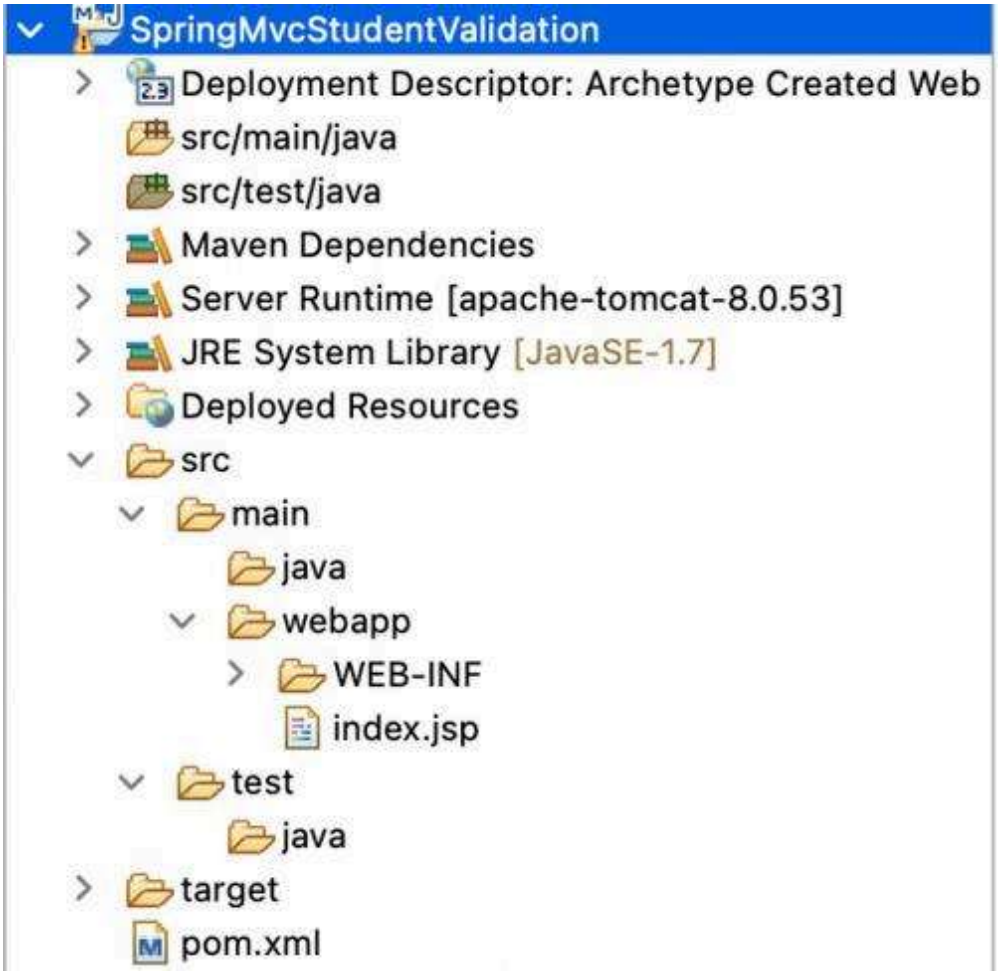


Provide all the metadata:



Project Structure

A maven web project would be created with a **pom.xml** configuration file. The project structure would look something like this:



Step 2: Configure the pom.xml File

Now, let's configure the **pom.xml** configuration file to add dependencies. Maven will get and manage all the dependencies defined in this file.

pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="https://maven.apache.org/POM/4.0.0"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="https://maven.apache.org/POM/4.0.0
                             https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.gfg</groupId>
    <artifactId>SpringMvcStudentValidation</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>

    <name>SpringMvcStudentValidation Maven Webapp</name>
```

```
    <url>http://www.example.com</url>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <spring.version>5.3.29</spring.version>

<hibernate.validator.version>6.2.0.Final</hibernate.validator.version>
        <servlet.api.version>4.0.1</servlet.api.version>
        <jstl.version>1.2</jstl.version>
        <junit.version>4.13.2</junit.version>
    </properties>

    <dependencies>
        <!-- JUnit for testing -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>

        <!-- Spring MVC -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <!-- Servlet API -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>${servlet.api.version}</version>
            <scope>provided</scope>
        </dependency>

        <!-- JSTL -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>${jstl.version}</version>
        </dependency>

        <!-- Hibernate Validator -->
        <dependency>
            <groupId>org.hibernate.validator</groupId>
            <artifactId>hibernate-validator</artifactId>
            <version>${hibernate.validator.version}</version>
        </dependency>

        <!-- Jakarta Validation API -->
        <dependency>
            <groupId>javax.validation</groupId>
            <artifactId>validation-api</artifactId>
```

```
            <version>2.0.1.Final</version>
        </dependency>
    </dependencies>

    <build>
        <finalName>SpringMvcStudentValidation</finalName>
        <pluginManagement>
            <plugins>
                <plugin>
                    <artifactId>maven-clean-plugin</artifactId>
                    <version>3.1.0</version>
                </plugin>
                <plugin>
                    <artifactId>maven-resources-plugin</artifactId>
                    <version>3.0.2</version>
                </plugin>
                <plugin>
                    <artifactId>maven-surefire-plugin</artifactId>
                    <version>2.22.1</version>
                </plugin>
                <plugin>
                    <artifactId>maven-war-plugin</artifactId>
                    <version>3.2.2</version>
                </plugin>
                <plugin>
                    <artifactId>maven-install-plugin</artifactId>
                    <version>2.5.2</version>
                </plugin>
                <plugin>
                    <artifactId>maven-deploy-plugin</artifactId>
                    <version>2.8.2</version>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.8.1</version>
                    <configuration>
                        <source>${maven.compiler.source}</source>
                        <target>${maven.compiler.target}</target>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.apache.tomcat.maven</groupId>
                    <artifactId>tomcat7-maven-plugin</artifactId>
                    <version>2.2</version>
                    <configuration>
                        <path>/</path>
                        <contextReloadable>true</contextReloadable>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
</project>
```

Step 3: Configure the web.xml File

The web.xml file in the "**webapp/WEB-INF/web.xml**" defines mapping with different URLs and servlets to handle requests for those URLs. In this, we have defined the servlet XML file to be named as **gfg.** The **URL-pattern** is set as "/", which means any request with "/" will be sent to the dispatcher servlet to be handled.

```
<web-app
xmlns="http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/index.html"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"

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

http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/web-app_3_0.xsd"
    version="3.0">

    <display-name>To do List</display-name>

    <welcome-file-list>
        <welcome-file>login.do</welcome-file>
    </welcome-file-list>

    <servlet>
        <servlet-name>gfg</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/gfg-servlet.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>gfg</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```

Step 4: Create the gfg-servlet.xml Configuration

The **gfg-servlet.xml** file is located in "**/src/main/webapp/WEB-INF/gfg.servlet.xml**". This file is named after the servlet-name we have mentioned in the web.xml file, it handles all HTTP requests for the web applications. The annotation-driven enable the spring annotation classes. The bean configuration helps in identifying and scanning the jsp located in the views folder. The component scan locates and allocated beans according to the defined annotation. It's a pretty simple file to handle incoming HTTP calls.

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

http://www.springframework.org/schema/context//spring-context.xsd">

    <context:component-scan base-package="com.gfg" />
     <bean

class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix">
            <value>/WEB-INF/views/</value>
        </property>
        <property name="suffix">
            <value>.jsp</value>
        </property>
    </bean>
    <mvc:annotation-driven />

</beans>
```

Step 5: Create the Student Model Class

Now, we'll create a student class in the **com.gfg.model** package, for our student portal we'll show four attributes firstName, lastName, rollNo, and address. The first two fields have @**Size** validation annotation as a constrain of minimum size as one, so they can't be empty. For roll number, we have used @**Min** which is also a predefined validator. Now, for address, we have used @**Address** which is a custom validator that we are going to define and configure.

```
package com.gfg.model;

import javax.validation.constraints.Min;
import javax.validation.constraints.Size;
import com.gfg.validationconfig.Address;

public class Student {
    @NotNull(message = "Student first name can't be empty")
    @Size(min = 1, message = "Student first name can't be empty")
    private String firstName;

    @NotNull(message = "Student last name can't be empty")
    @Size(min = 1, message = "Student last name can't be empty")
    private String lastName;

    @Min(value = 1000, message = "Roll number must be a four-digit number")
    private int rollNo;

    @NotNull(message = "Address cannot be null")
    @Address(message = "Address must contain 'India'")
    private String address;

    // Default constructor
    public Student() {}

    // Getters and Setters
    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; }

    public int getRollNo() { return rollNo; }
    public void setRollNo(int rollNo) { this.rollNo = rollNo; }

    public String getAddress() { return address; }
    public void setAddress(String address) { this.address = address; }
}
```

Step 6: Create the Student Controller

The Controller class handles the incoming requests by redirecting them to the appropriate view page, any URL must be defined in the controller class in order to send a request. In this project, we have defined a **StudentController** class in **com.gfg.controller** package. In this class we have two methods for two request, the first one simply redirects to the login portal and simply add a new **student** object to match the inputs into the form using the model's **addAttribute**. The second method redirects to the welcome page, but before that, it checks for any **validation errors** using the **BindingResult**. If there exists an error it redirects to the portal else to the welcome page.

```
 package com.gfg.controller;

import javax.validation.Valid;

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

import com.gfg.model.Student;

@Controller
public class StudentController {

    @RequestMapping("/login")
    public String showForm(Model theModel) {
        theModel.addAttribute("student", new Student());
        return "portal";
    }

    @RequestMapping("/welcome")
    public String processForm(@Valid @ModelAttribute("student") Student
student, BindingResult result) {
        if (result.hasErrors()) {
            return "portal";
        }
        else {
            return "welcome";
        }
    }
}
```

Step 7: Create the custom Address Validator

The **AddressValidator** class in the **com.gfg.validationcongfig** package defines the constraint for which the object needs to be checked, this class implements the **ConstraintValidator** which defines the logic to validate a given constrain. We override a method class **isValid** from the ConstrainValidator interface, where we define our validation logic.

```
package com.gfg.validationconfig;

import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;

public class AddressValidator implements ConstraintValidator<Address,
String> {

    @Override
    public boolean isValid(String address, ConstraintValidatorContext
context) {
        if (address == null) {
            return false;
        }
        return address.toLowerCase().contains("india");
    }
}
```

Step 8: Define the Address Annotation

Now, we create an Address annotation using the @**interface,** This class must have three overridden methods in order for the validation logic to work. The first method defines the error message to be shown, the second method represents a group of constraints, the third method represents additional information about the annotation. The @**Constraint**, @**Target**, and @**Retention** annotation define the validation logic, the element to be passed, and additional information.

```
package com.gfg.validationconfig;
import java.lang.annotation.ElementType;
```

```
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import javax.validation.Constraint;
import javax.validation.Payload;

@Constraint(validatedBy = AddressValidator.class)
@Target( { ElementType.METHOD, ElementType.FIELD } )
@Retention(RetentionPolicy.RUNTIME)
public @interface Address {

    public String message() default "You address must contains india";

        public Class<?>[] groups() default {};
        public Class<? extends Payload>[] payload() default {};
}
```

Step 9: Create the portal.jsp Form

The **portal.jsp** page the "**/webapp/WEB-INF/views/portal.jsp**" defines the student login portal. We have used the form configuration to format our form. This is a pretty simple form that defines four fields.

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

 <html>
<head>
    <style>
        .error {color:red}
    </style>
</head>
<body>
    <h1>Student Portal</h1>
    <form:form action="welcome" modelAttribute="student">
        <label>First name:</label>
         <form:input path="firstName" />
         <form:errors path="firstName" cssClass="error" /><br><br>

         <label>Last name:</label>
         <form:input path="lastName" />
        <form:errors path="lastName" cssClass="error" /><br><br>

         <label>Roll No:</label>
          <form:input path="rollNo" />
         <form:errors path="rollNo" cssClass="error" /><br><br>

         <label>Address:</label>
         <form:textarea path="address" />
```

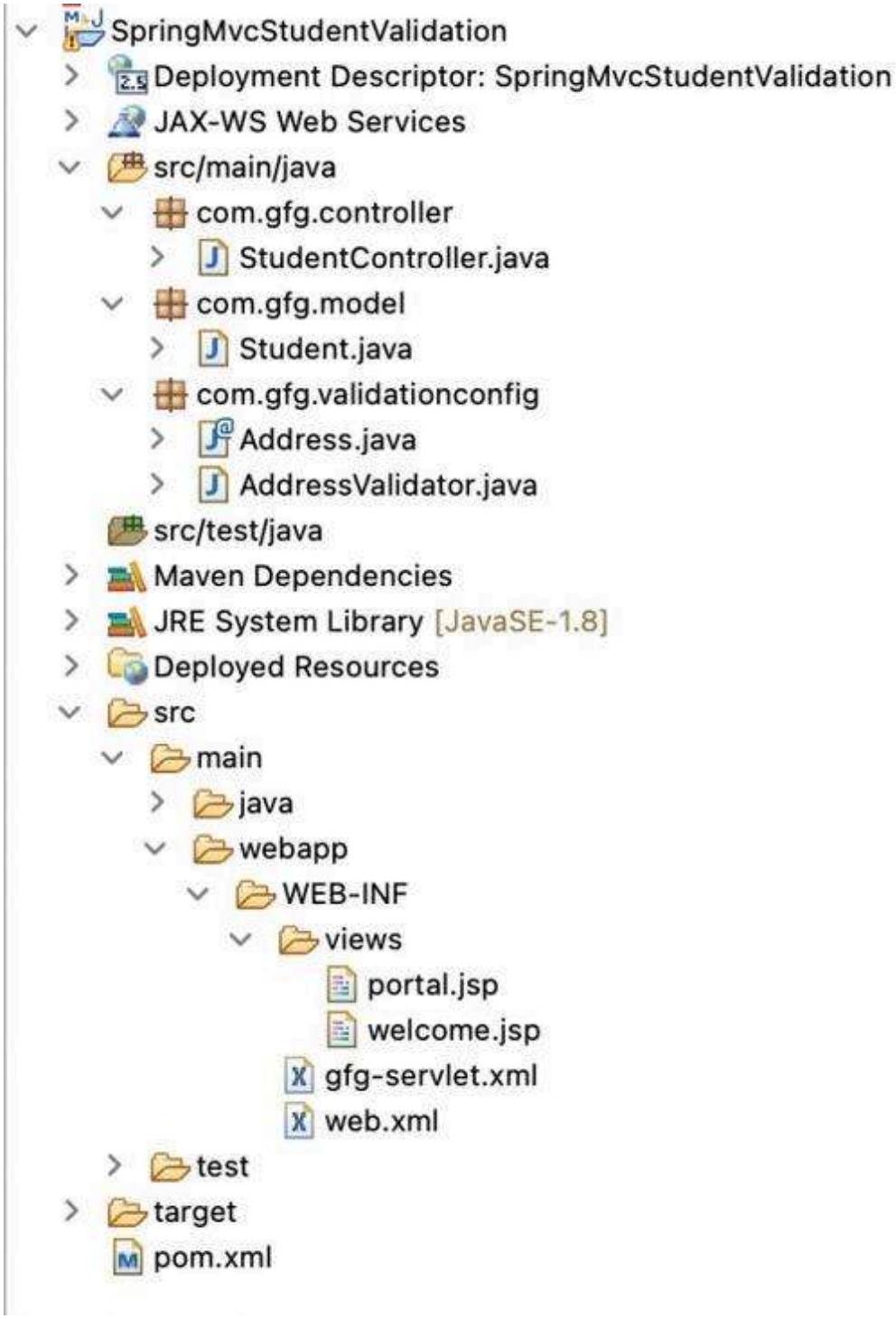
```
        <form:errors path="address" cssClass="error" /><br><br>

        <input type="submit" value="Submit" />
    </form:form>
</body>
</html>
```

Step 10: Create the welcome.jsp View

The **welcome.jsp** page in the "**/webapp/WEB-INF/views/welcome.jsp**" is the view page that is shown after successful login.

```
<%-- <%@ taglib prefix="c"
uri="http://www.oracle.com/technetwork/java/index.html" %>
 --%>
<!DOCTYPE html>
<html>
    <body>
        <div>
            <h2>Welcome ${student.firstName} ${student.lastName} to Student
portal<br><br>
            Your Roll Number is ${student.rollNo} and you live in India.
</h2>
        </div>
    </body>
</html>
```

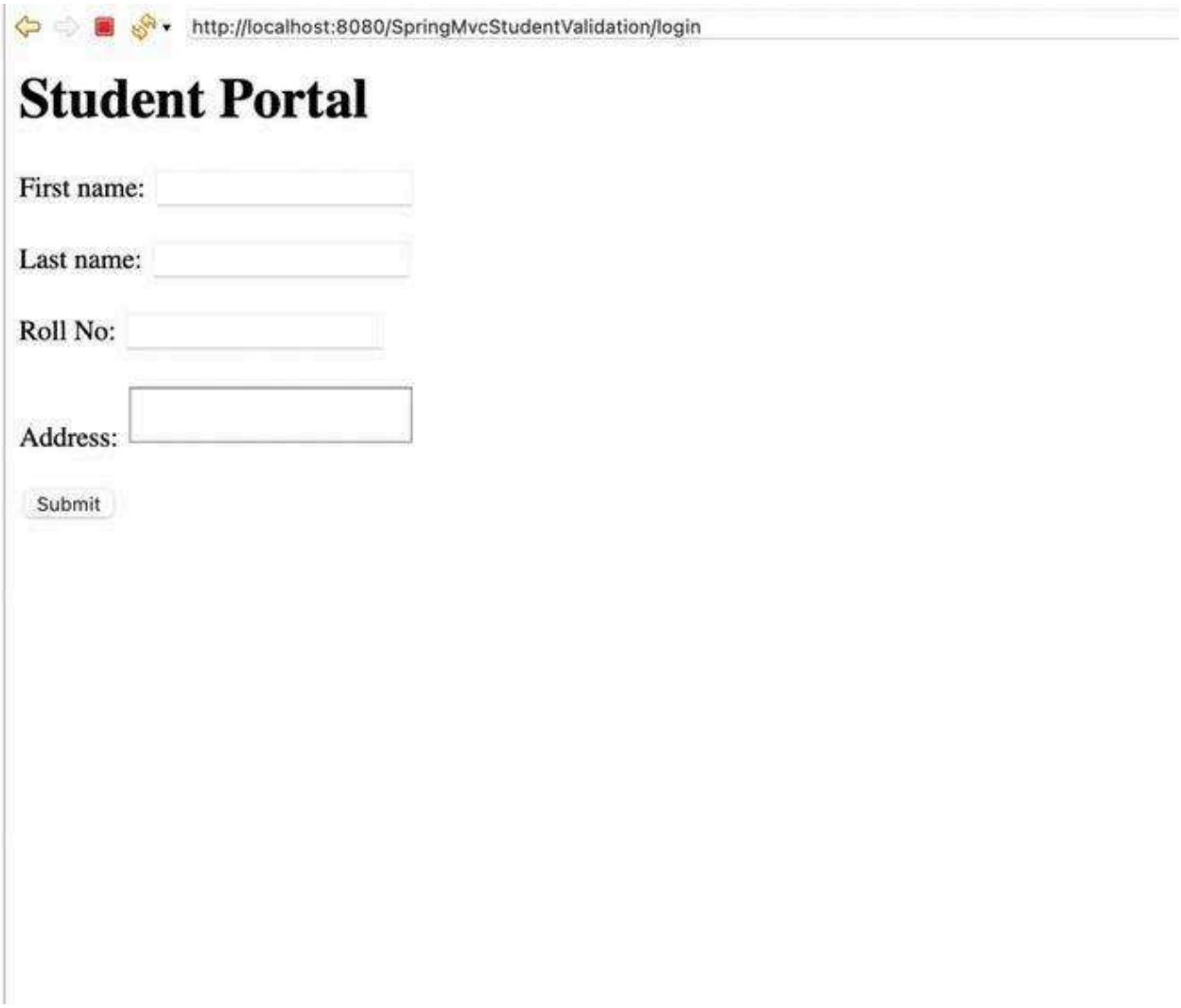
Now, that our Spring MVC project has been completed with all the configuration files and classes. The Structure of your project should look like this:



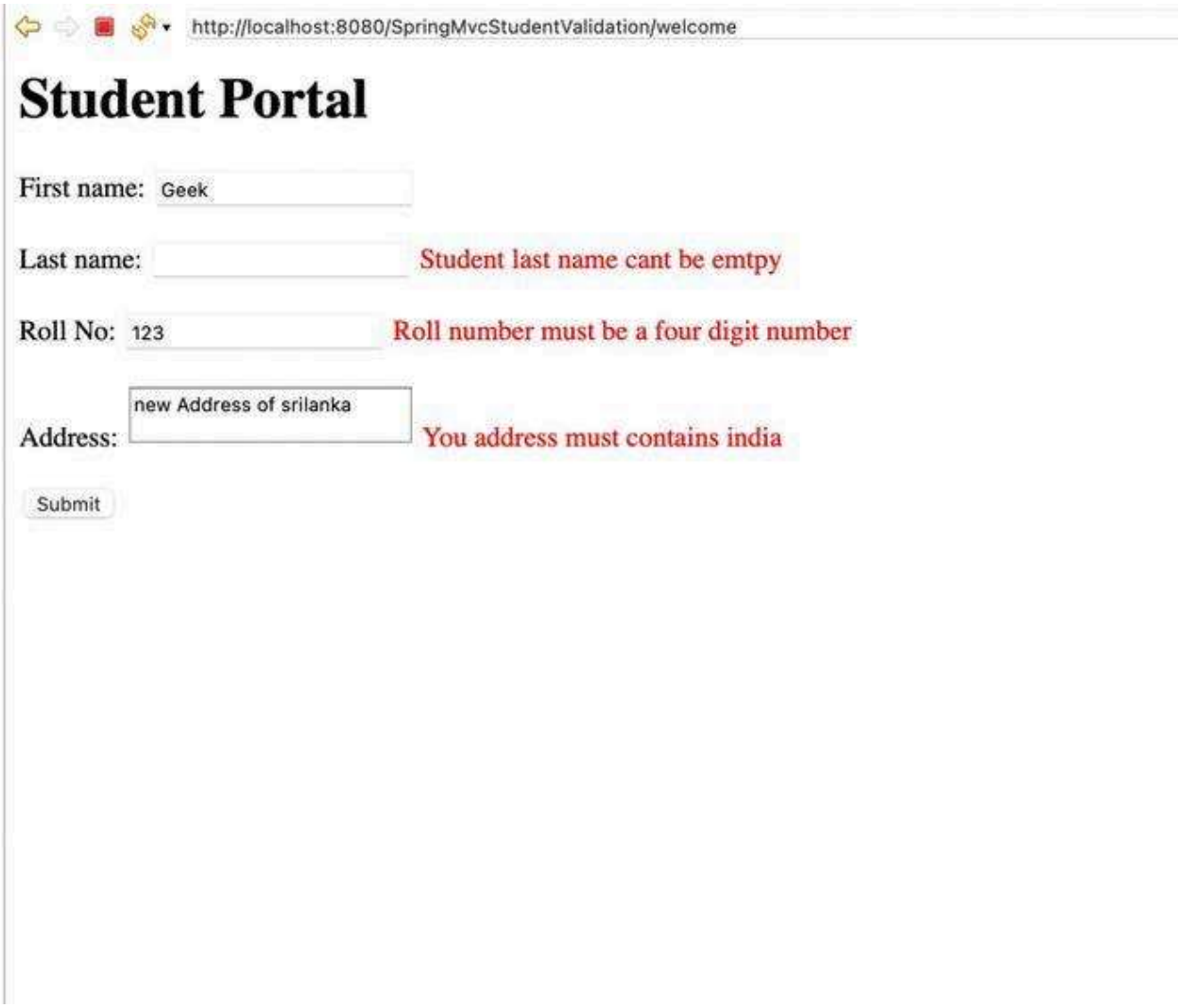
Output:

It's time to run your web application on the tomcat server. I am assuming you know how to run a tomcat server. After successfully running the tomcat server enter **http://localhost:8080/SpringMvcStudentValidation/welcome** in your favorite browser.

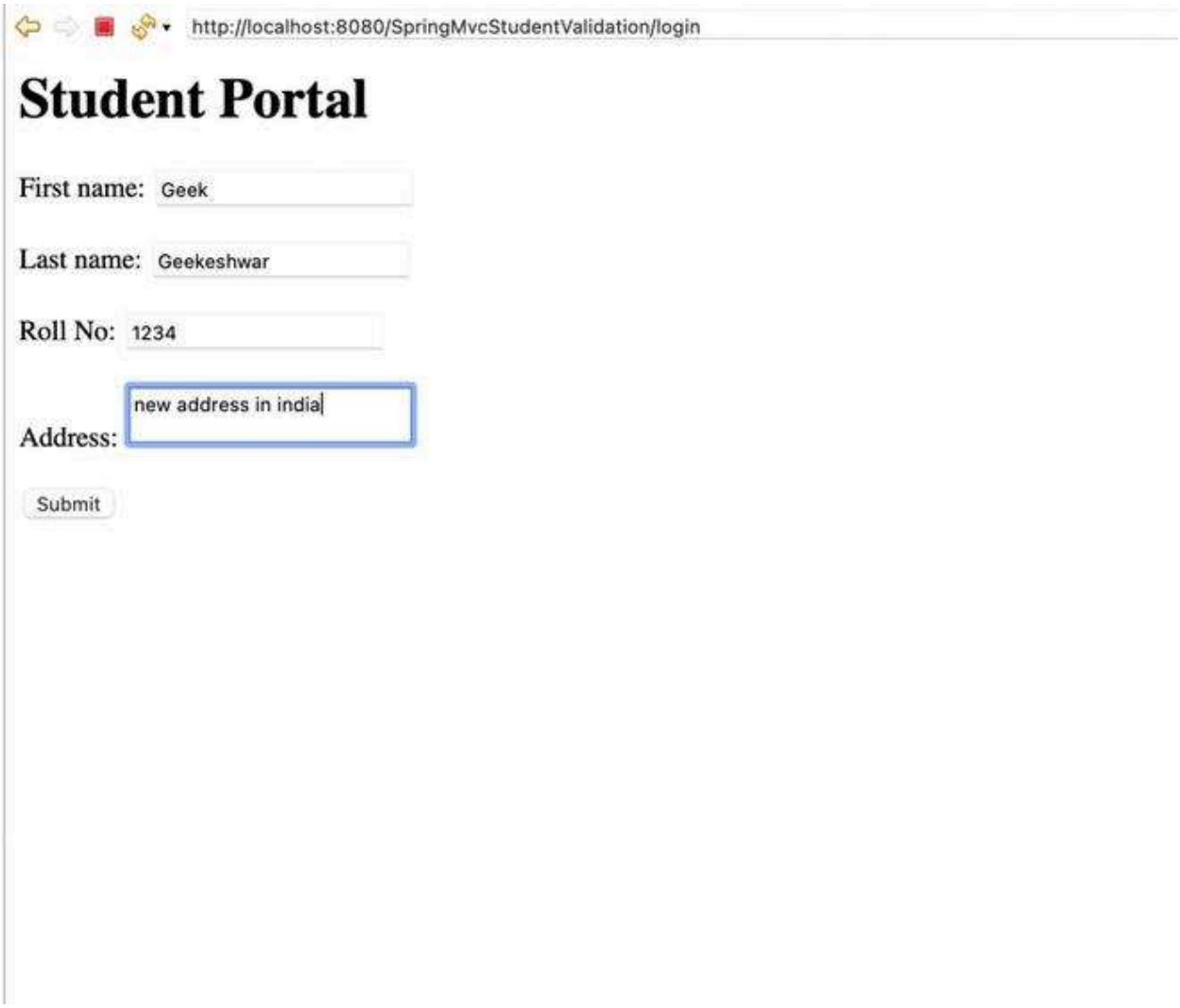
The image below demonstrates the Student Portal form with fields for entering the first name, last name, roll number, and address (this is the initial form before validation takes place).



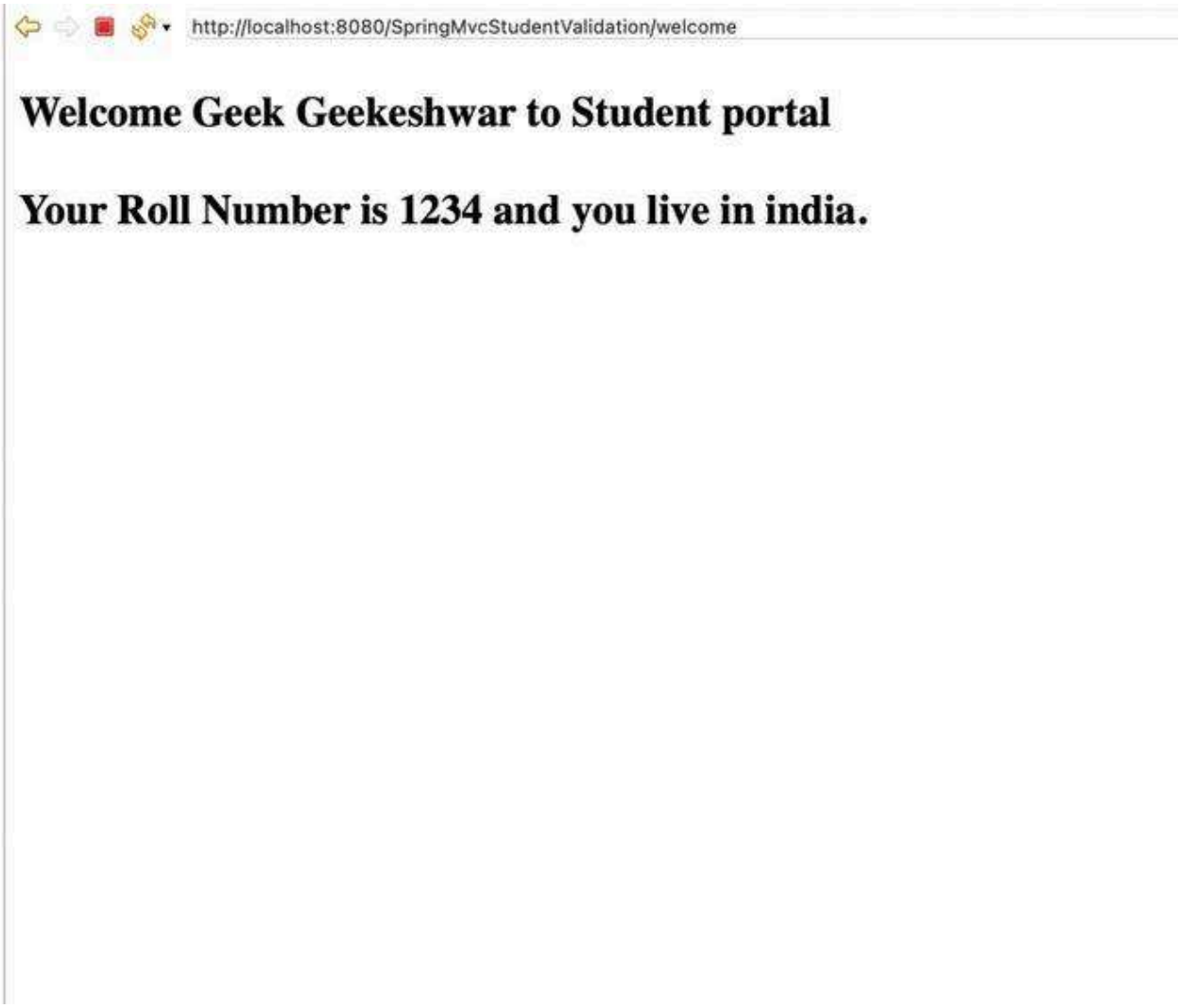
The below image demonstrates the validation errors as per the custom validation.



The below image demonstrates that the form has been filled correctly.



Upon successful submission, the Welcome page is shown with a confirmation message. The user's first name, last name, roll number, and address (with "India") are displayed.



So, we have created a Spring MVC project with our own validation and used it in a student portal form.