# Validating Form Input

**Description:** Spring Boot is an open source, micro service-based Java web framework. The Spring Boot framework creates a fully production-ready environment that is completely configurable using its prebuilt code within its own codebase.

**Project:** Build a simple web page containing a form to submit data for data validation, processing and to display results. This project is based on "Validating Form Input" and just expands on the screenshots and step by step instructions. This project might also sequences building the application in attempt to reduce dependency errors.

**Technology:** This project uses the following technology:

- Integrated Development Environment (IDE):
  - Spring Tool Suite 4 (Version: 4.11.0.RELEASE)
- Java Development Kit (JDK):
  - Oracle's JDK 8 (1.8)

## Table of Contents

## Glossary of Terminology

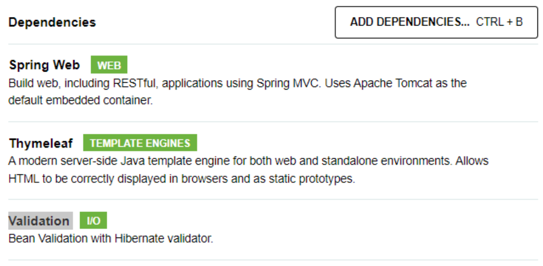
For a list of key terms and definitions used throughout this and various Spring Boot demo documents see the document titled "Appendix 01 Glossary".

## Generate Spring Boot Download

Follow the instructions in the document title "Appendix 02 Spring Initializr" to generate a spring boot download for this project.

When the document talks about adding dependencies add only these:

**Spring Web**
**Thymeleaf**
**Validation**



Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Thymeleaf TEMPLATE ENGINES
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

Validation I/O
Bean Validation with Hibernate validator.

When the document talks about the items in the "**Project Metadata**" use the values shown below:

"**Group**" use "**com.example**"
"**Artifact**" use "**validating-form-input**"
"**Name**" use "**validating-form-input**"
"**Package name**" use "**com.example.validating-form-input**"

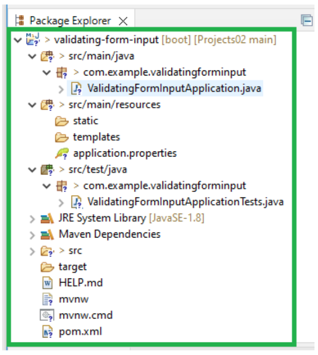
For other items in the "**Project Metadata**" use the defaults.  Follow the instructions to extract the files from the zip file into the Spring Tool Suite (STS) 4 workspace.

## Import the Spring Boot Download

Open the Spring Tool Suite 4 IDE.

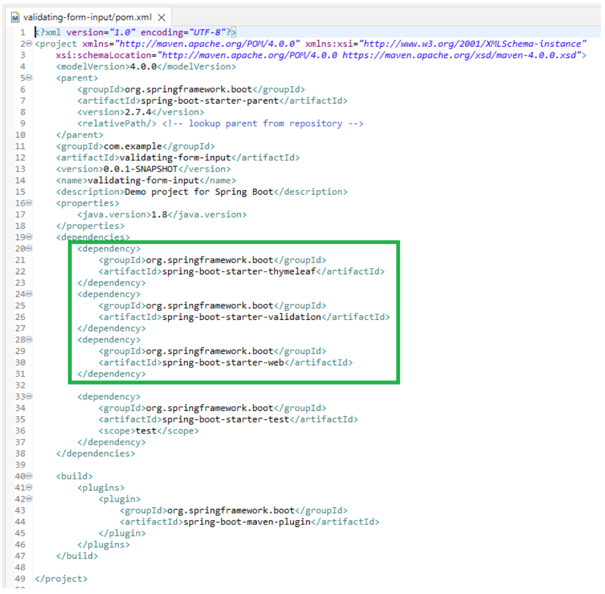
### Import the project: "validating-form-input"

Follow the instructions in the document title "Appendix 03 Import Spring Tool Suite Project" and import the "**validating-form-input**" project that was created with Spring Initializr.  After importing the project, it should look like the following using the IDE "Package Explorer".



```
Package Explorer
validating-form-input [boot] [Projects02 main]
  src/main/java
    com.example.validatingforminput
      ValidatingFormInputApplication.java
  src/main/resources
    static
    templates
    application.properties
  src/test/java
    com.example.validatingforminput
      ValidatingFormInputApplicationTests.java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

Look at the pom.xml and find the added dependencies for this project.



validating-form-input/pom.xml

```xml
<?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 https://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>2.7.4</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>validating-form-input</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>validating-form-input</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

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

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

## Project Discussion

This project will create simple web page containing a form to submit data for validation, processing and to display results. The project will be tested within the IDE using the Spring Boot built in Tomcat server.

## Create the Project from the Import

Use the STS 4 IDE to modify the code after importing the project.

### Main Application Class: ValidatingFormInputApplication

The class file "ValidatingFormInputApplication" containing the startup method generated by Spring Initializr tool is used without modification.

Main application class @SpringBootApplication annotation discussion.

`@SpringBootApplication` is a convenience annotation that adds all of the following:

- `@Configuration`: Tags the class as a source of bean definitions for the application context.
- `@EnableAutoConfiguration`: Tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings. For example, if `spring-webmvc` is on the classpath, this annotation flags the application as a web application and activates key behaviors, such as setting up a `DispatcherServlet`.
- `@ComponentScan`: Tells Spring to look for other components, configurations, and services in the `com/example` package, letting it find the controllers.

The `main()` method uses Spring Boot's `SpringApplication.run()` method to launch an application. Did you notice that there was not a single line of XML? There is no `web.xml` file, either. This web application is 100% pure Java and you did not have to deal with configuring any plumbing or infrastructure. -- "Handling Form Submission"

## Create Model / Entity Class: PersonForm

A model or entity class is used to describe real world objects and data and many time describe database tables. It is part of the MVC architecture. Create a class under folder "src/main/java" in package "com.example.validatingforminput" named: PersonForm.

This model class contains validation annotation from javax. Copy and replace the code shell generated by the STS IDE.

```
package com.example.validatingforminput;

import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

public class PersonForm {

          @NotNull
          @Size(min=2, max=30)
          private String name;

          @NotNull
          @Min(18)
          private Integer age;

          public String getName() {
                    return this.name;
          }

          public void setName(String name) {
                    this.name = name;
          }

          public Integer getAge() {
                    return age;
          }

          public void setAge(Integer age) {
                    this.age = age;
          }

          public String toString() {
                    return "Person(Name: " + this.name + ", Age: " + this.age + ")";
          }
}
```

The `PersonForm` class has two attributes: `name` and `age`. It is flagged with a few standard validation annotations:

- `@Size(min=2, max=30)`: Allows names between 2 and 30 characters long.
- `@NotNull`: Does not allow a null value, which is what Spring MVC generates if the entry is empty.
- `@Min(18)`: Does not allow the age to be less than 18.

-- "Validating Form Input"

## Create Controller Class: WebController

Controller annotation discussion.  Using @RestController buys us more flexibility than using @Controller.  @Controller is used to identify classes as Spring MVC Controller.  @RestController does the same and plus provides functionality as if using @ResponseBody separately.  @RestController is used in building RESTful Web services.  For the controller layer in the MVC architecture, controller classes handles the initial request and response from a view layer.

For this project the "Validating Form Input" website uses @Controller.  This project will also use @Controller and not @RestController.

Create a class under folder "src/main/java" in package "com.example.validatingforminput" named: WebController.
This controller class contains one get and one post endpoints.  Copy and replace the code shell generated by the STS IDE.

```
package com.example.validatingforminput;

import javax.validation.Valid;

import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;


@Controller
public class WebController implements WebMvcConfigurer {

	@Override
	public void addViewControllers(ViewControllerRegistry registry) {
		registry.addViewController("/results").setViewName("results");
	}

	@GetMapping("/")
	public String showForm(PersonForm personForm) {
		return "form";
	}

	@PostMapping("/")
	public String checkPersonInfo(@Valid PersonForm personForm, BindingResult bindingResult) {

		if (bindingResult.hasErrors()) {
			return "form";
		}

		return "redirect:/results";
	}
}
```

This controller has a GET method and a POST method. Both methods are mapped to `/`.

The `showForm` method returns the `form` template. It includes a `PersonForm` in its method signature so that the template can associate form attributes with a `PersonForm`.

The `checkPersonInfo` method accepts two arguments:

- A `personForm` object marked with `@Valid` to gather the attributes filled out in the form.
- A `bindingResult` object so that you can test for and retrieve validation errors.

You can retrieve all the attributes from the form, which is bound to the `PersonForm` object. In the code, you test for errors. If you encounter an error, you can send the user back to the original `form` template. In that situation, all the error attributes are displayed.

If all of the person's attribute are valid, it redirects the browser to the final `results` template.

-- "Validating Form Input"

## Create View Component: form.html

This project uses a simple HTML page as a view component within the STS 4 IDE. The view component is created in the resources folder “src/main/resources/templates”. The view components will incorporate Thymeleaf the dependency added in the pom.xml.

Create a file under src/main/resources/templates named: “form.html”
Right click on “templates” folder ➔ “New” ➔ “File”
Add the following code to the file:

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
    <body>
        <form action="#" th:action="@{/}" th:object="${personForm}" method="post">
            <table>
                <tr>
                    <td>Name:</td>
                    <td><input type="text" th:field="*{name}" /></td>
                    <td th:if="${#fields.hasErrors('name')}" th:errors="*{name}">Name Error</td>
                </tr>
                <tr>
                    <td>Age:</td>
                    <td><input type="text" th:field="*{age}" /></td>
                    <td th:if="${#fields.hasErrors('age')}" th:errors="*{age}">Age Error</td>
                </tr>
                <tr>
                    <td><button type="submit">Submit</button></td>
                </tr>
            </table>
        </form>
    </body>
</html>
```

The page contains a simple form, with each of its field in a separate cell in a table. The form is geared to post to `/`. It is marked as being backed up by the `personForm` object that you saw in the `GET` method in the web controller. This is known as a “bean-backed form”. There are two fields in the `PersonForm` bean, and you can see that they are tagged with `th:field="*{name}"` and `th:field="*{age}"`. Next to each field is a secondary element that is used to show any validation errors.
Finally, you have a button that submits the form. In general, if the user enters a name or age that violates the `@Valid` constraints, it bounces back to this page with the error message displayed. If a valid name and age is entered, the user is routed to the next web page.
-- “Validating Form Input”

## Create View Component: results.html

Create the results web page.

Create a file under src/main/resources/templates named: “results.html”
Right click on “templates” folder ➔ “New” ➔ “File”
Add the following code to the file:

```
<html>
          <body>
                    Congratulations! You are old enough to sign up for this site.
          </body>
</html>
```

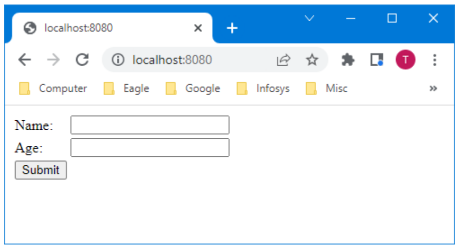
## Run the Application

Start the application inside the STS 4 IDE as a spring boot application.
Right click inside ValidatingFormInputApplication class ➔ “Run As” ➔ “Spring Boot App”

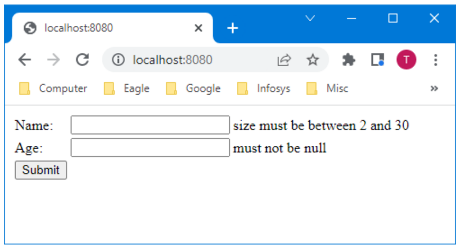
Enter the URL in a browser window:

```
http://localhost:8080
```

The initial web page is displayed. Submitting the form without any values display the field information on the screen.

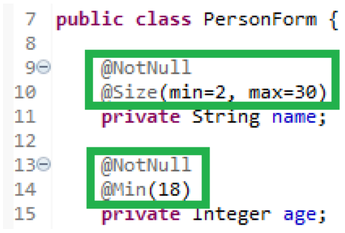


Name:
Age:



Name: size must be between 2 and 30
Age: must not be null

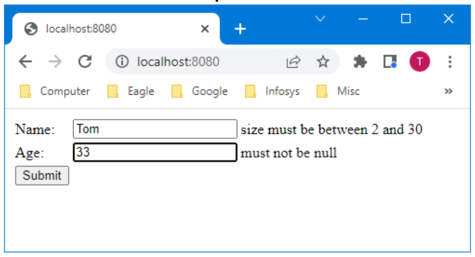
The field validation is set in the PersonForm model.



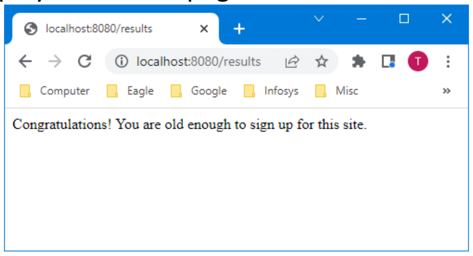
```
public class PersonForm {

    @NotNull
    @Size(min=2, max=30)
    private String name;

    @NotNull
    @Min(18)
    private Integer age;
```

If you search the code for portion of the messages being displayed, “must be between” or “must not be” they will not be found. This must be a function of the Validation dependency in the pom.xml.

Enter values that pass validation will display the results page.



Name: Tom size must be between 2 and 30
Age: 33 must not be null



Congratulations! You are old enough to sign up for this site.

## Project Conclusion

This concludes the simple web page form to submit and validate data project. The data submitted was validated to allow access to the results page. The field validation relied on the Validation dependency in the pom.xml and the field values to validate is found in the class model.