Handling Form Submission

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 processing and to display results. This project is based on "Handling Form Submission" 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

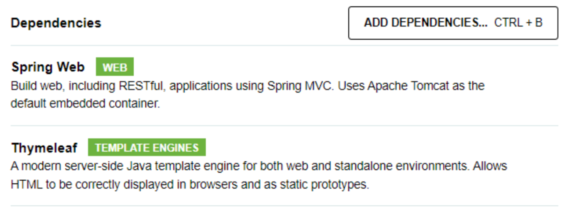
For a list of key terms and definitions used throughout this and various Spring Boot demo documents see the document titled “Appendix 01 Glossary”.

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



Dependencies
ADD DEPENDENCIES... CTRL + B

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.

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

“**Group**” use “**com.example**”
“**Artifact**” use “**handling-form-submission**”
“**Name**” use “**handling-form-submission**”
“**Package name**” use “**com.example.handling-form-submission**”

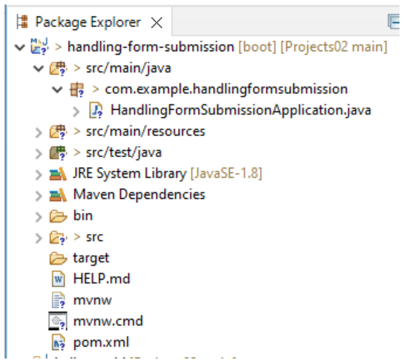
For other items in the “**Project Metadata**” use the defaults. Follow the instructions to extract the files from the zip file into the Spring Tool Suite (STS) 4 workspace.

Import the Spring Boot Download

Open the Spring Tool Suite 4 IDE.

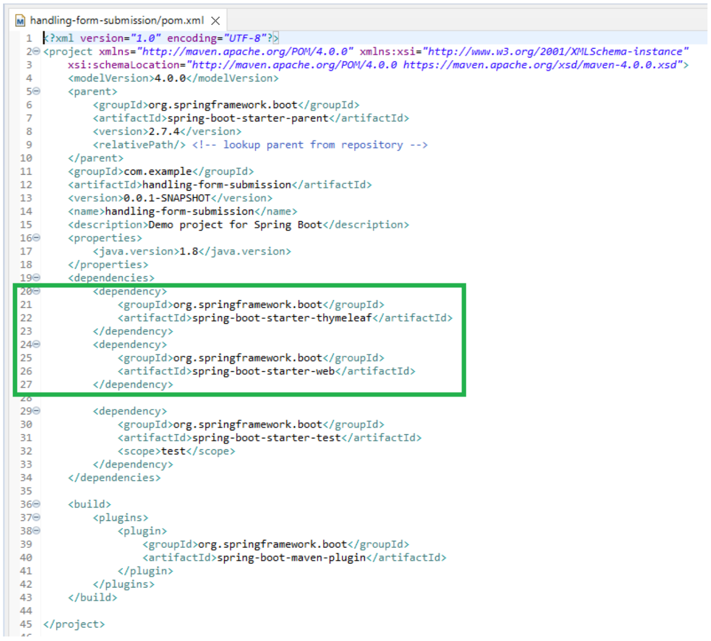
Import the project: “handling-form-submission”

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



```
Package Explorer
v > handling-form-submission [boot] [Projects02 main]
  v > src/main/java
    v > com.example.handlingformsubmission
      > HandlingFormSubmissionApplication.java
  > > src/main/resources
  > > src/test/java
  > JRE System Library [JavaSE-1.8]
  > Maven Dependencies
  > bin
  > > src
    target
    HELP.md
    mvnw
    mvnw.cmd
    pom.xml
```

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



handling-form-submission/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>handling-form-submission</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>handling-form-submission</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-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 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: HandlingFormSubmissionApplication

The class file "HandlingFormSubmissionApplication" 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: Greeting

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.handlingformsubmission" named: **Greeting**.

This model class is very basic. Copy and replace the code shell generated by the STS IDE.

```
package com.example.handlingformsubmission;

public class Greeting {

  private long id;
  private String content;

  public long getId() {
    return id;
  }

  public void setId(long id) {
    this.id = id;
  }

  public String getContent() {
    return content;
  }

  public void setContent(String content) {
    this.content = content;
  }

}
```

The `greetingForm()` method uses a `Model` object to expose a new `Greeting` to the view template. The `Greeting` object in the following code (from `src/main/java/com/example/handlingformsubmission/Greeting.java`) contains fields such as `id` and `content` that correspond to the form fields in the `greeting` view and are used to capture the information from the form. -- “Handling Form Submission”

Create Controller Class: GreetingController

Create a class under folder "src/main/java" in package "com.example.handlingformsubmission" named: GreetingController.

This controller class contain one get and one post endpoints. Copy and replace the code shell generated by the STS IDE.

```
package com.example.handlingformsubmission;

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

@Controller
public class GreetingController {

  @GetMapping("/greeting")
  public String greetingForm(Model model) {
    model.addAttribute("greeting", new Greeting());
    return "greeting";
  }

  @PostMapping("/greeting")
  public String greetingSubmit(@ModelAttribute Greeting greeting, Model model) {
    model.addAttribute("greeting", greeting);
    return "result";
  }

}
```

The mapping annotations let you map HTTP requests to specific controller methods. The two methods in this controller are both mapped to `/greeting`. You can use `@RequestMapping` (which, by default, maps all HTTP operations, such as `GET`, `POST`, and so forth). However, in this case, the `greetingForm()` method is specifically mapped to `GET` by using `@GetMapping`, while `greetingSubmit()` is mapped to `POST` with `@PostMapping`. This mapping lets the controller differentiate the requests to the `/greeting` endpoint. -- "Handling Form Submission"

Create View Component: greeting.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. The HTML page contains the form that this project processes.

The implementation of the method body relies on a view technology to perform server-side rendering of the HTML by converting the view name (`greeting`) into a template to render. In this case, we use Thymeleaf, which parses the `greeting.html` template and evaluates the various template expressions to render the form. The following listing (from `src/main/resources/templates/greeting.html`) shows the `greeting` template. -- "Handling Form Submission"

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

```
<!DOCTYPE HTML>
<html xmlns:th="https://www.thymeleaf.org">
<head>
    <title>Getting Started: Handling Form Submission</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
        <h1>Form</h1>
    <form action="#" th:action="@{/greeting}" th:object="${greeting}" method="post">
        <p>Id: <input type="text" th:field="*{id}" /></p>
         <p>Message: <input type="text" th:field="*{content}" /></p>
         <p><input type="submit" value="Submit" /> <input type="reset" value="Reset" /></p>
    </form>
</body>
</html>
```

The `th:action="@{/greeting}"` expression directs the form to POST to the `/greeting` endpoint, while the `th:object="${greeting}"` expression declares the model object to use for collecting the form data. The two form fields, expressed with `th:field="{id}"` **and** `th:field="{content}"`, correspond to the fields in the `Greeting` object. -- "Handling Form Submission"

Create View Component: result.html

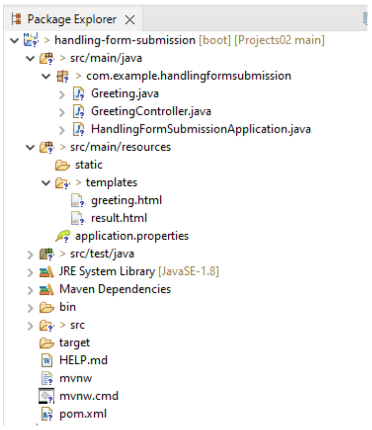
The HTML page contains the results from processing information from the form.

Create a file under src/main/resources/templates named: “result.html”
Add the following code to the file:

```
<!DOCTYPE HTML>
<html xmlns:th="https://www.thymeleaf.org">
<head>
    <title>Getting Started: Handling Form Submission</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
        <h1>Result</h1>
    <p th:text="'id: ' + ${greeting.id}" />
    <p th:text="'content: ' + ${greeting.content}" />
    <a href="/greeting">Submit another message</a>
</body>
</html>
```

Run the Application

The final project structure looks like the following with the three class files under the main package and two HTML resources:

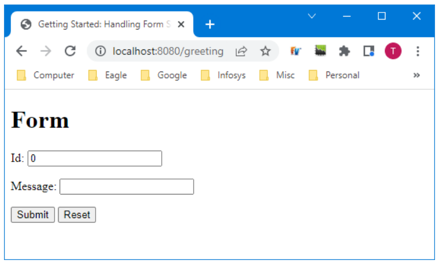


```
Package Explorer
v > handling-form-submission [boot] [Projects02 main]
  v > src/main/java
    v > com.example.handlingformsubmission
      > Greeting.java
      > GreetingController.java
      > HandlingFormSubmissionApplication.java
  v > src/main/resources
      static
    v > templates
        greeting.html
        result.html
      application.properties
  > > src/test/java
  > JRE System Library [JavaSE-1.8]
  > Maven Dependencies
  > bin
  > > src
    target
    HELP.md
    mvnw
    mvnw.cmd
    pom.xml
```

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

Enter the URL in a browser window:

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

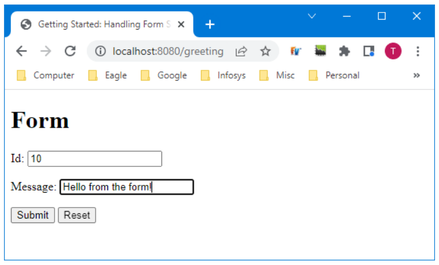


Form

Id: 0

Message:

Submit Reset



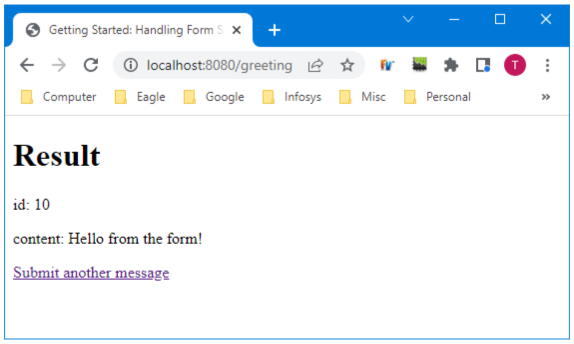
Form

Id: 10

Message: Hello from the form|

Submit Reset

Enter form data and submit.



Result

id: 10

content: Hello from the form!

Submit another message

The results are shown in the result.html resource file.
A link is provided to go back to the form and enter data.

Project Conclusion

This concludes the simple web page form to submit data project. The data submitted was simply echo back in a results page. The next form project "Spring Boot 110 Form Input Validation" will introduce data validation.