Web Programming with Spring Boot and Thymeleaf

1 Overview

In this assignment, students will learn the basics of web programming with Spring Boot and Thymeleaf. The accompanying source code illustrates how to create a web page that responds on specific Uniform Resource Locators (URLs), how to control the output of these web pages, by passing variables to the model, how to create HyperText Markup Language (HTML) tables, and how to create objects with different scopes, request, session, and application, e.g., to share between different users of the web application or to keep information for the life of an interactive session.

2 Creating and Running a Project

While we will make use of the source code distributed with this assignment, students should, nonetheless, know how to create their own Spring Boot project from scratch. One way of doing this is by creating and downloading a project from the Spring Initializer web page: `https://start.spring.io`. In this initializer page, students may select the following dependencies:

- Spring Web.
- Thymeleaf.
- Spring Boot DevTools.

Students may pick the latest version of Java and `jar` packaging. This will result in a `pom.xml` file with four dependencies:

- `spring-boot-starter-thymeleaf`.
- `spring-boot-starter-web`.
- `spring-boot-devtools`.
- `spring-boot-starter-test`.

To run the Spring Boot application, students should execute the following command on the command line on the root of the source code tree:

```
./mvnw spring-boot:run
```

The main class of the program has the annotation `@SpringBootApplication`. Among other things, this annotation adds an annotation called `@ComponentScan` that looks for controllers in the `com.example` package. This lets Spring discover our controller in the class `GreetingController`, which is the heart of our application.

3 Setting and Using Variables for the Model

Let us start by looking at the method `greeting()` of the controller. It has an annotation `@GetMapping("/greeting")`, which, in addition to defining the URL of the resource, tells the controller that the access to the `greeting()` method happens via the HyperText Transfer Protocol (HTTP) `GET` method. This method sets two variables for the model and returns a string with the name of the model to display, in this case, `"greeting"`. Students may find the corresponding template named in `greeting.html` and the way it uses the variables that we set up in the `greeting()` method.

Students should also look at the method `redirect()`, which returns an instruction for Spring to redirect the browser request to the `/greeting` URL. One should notice that the browser actually displays the final URL after the redirection.

Finally, we have the method `atable()` to illustrate the creation of tables. In this case, the template `table.html` will show the content of a list of objects of the class `Employee.java` in the form of a table. The controller must create the list of employees and, as in the previous cases, store it as a variable in the model object.

4 Forms

The methods `createProjectForm()` and `saveProjectSubmission()`, together with their respective templates and supporting classes, provide an example on how to use forms. To access the form, students should point their browsers to `http://localhost:8080/create-project`, which is the entry point of the example. The method `createProjectForm()` will create an object of the class `Project` to be filled and submitted in the form defined in the template `create-project.html`. This template provides different types of inputs to be submitted to the `/save-project` URL. Here, we use the `POST` method, because this submission will usually have side effects on the server. This destination URL receives the object that was set at form submission time and may use the object of class `Project` to do whatever it deems as necessary, like creating an entry in the database with a new project. In the end the `saveProjectSubmission()` method returns a template, `result.html`, that displays the created object of class `Project`.

5 Different Scopes for Variables

The method `counters()`, together with the three different `@Bean` annotations in the class illustrates some of the scopes available for variables in a Spring Boot web application, namely

request, session, and application. A variable with **request** scope changes every single request, this meaning that the Spring will invoke the `requestScopedNumberGenerator()` method every single request. A variable with **session** scope survives for the entire life of the session, while a variable with **application** scope is unique for the entire application and for all the clients using the application. The code also illustrates how to retrieve an `HttpSession` object, which provides a way of storing and retrieving session-scoped objects.

To understand the importance of variable scopes, students should open two different browsers and point them at `http://localhost:8080/counters`.

6 Servlets

Servlets play a crucial role in many Java frameworks. They offer methods like `doGet()` and `doPost()` that enable programmatic response to HTTP, respectively, `GET` and `PUT` requests. While using these methods is usually not a very good idea, because they mix business functionality with presentation, they are often used as the basis for other technologies, like Java Server Pages (JSPs). They are the base for template processing in Thymeleaf as well. Spring Boot also enables the use of servlets. In the attached example, students have access to two servlets, `Example` and `ThymeleafServlet`. While the former builds a straightforward `HTML` programmatically, the latter connects to a Thymeleaf template and enables the use of variables. Both servlets are registered in the `main` class of the application, with indication of the URL they serve. To make this example run, the `pom.xml` file needs a dependency for the Object Graph Navigation Library (OGNL), as follows (the version number may change):

```
<dependency>
    <groupId>ognl</groupId>
    <artifactId>ognl</artifactId>
    <version>3.1.12</version>
</dependency>
```

7 Available `URLs`

The following is the list of all the URLs available once students start the attached code successfully:

- `http://localhost:8080/exampleServlet/AnnotationExample`
- `http://localhost:8080/thymeleafServlet/hellofromservlet.html`
- `http://localhost:8080/thymeleafServlet/urls.html`
- `http://localhost:8080/`
- `http://localhost:8080/greeting`

- `http://localhost:8080/givemeatable`
- `http://localhost:8080/create-project`
- `http://localhost:8080/counters`

8 Exercises

Consider the following exercises:

- Could one get the value of a session variable using the expression `${session.counter}`?
- Create a form that serves for a user to log in.
- Think of a mechanism that could prevent the user from accessing resources if not logged in (search for servlet filters).