Spring Boot (BOOT)

Exercise BOOT –Spring Boot Application

The Setup :

Create a copy of your W3D2-VAL project and call it **W3D3-Boot**. If you don't like your validation solution, you can actually also just copy the W3D1-MVC project, but it won't have all the features that were added in the last couple of days.

We're going to make some significant changes to the pom.xml, the first of which should of course be changing the <artifactId> to W3D3-Boot.

Then change the <packaging> to jar, delete the <endorsed.dir> tag from <properties>, and delete all the contents of the <dependencies> tag and of the <plugins> tag.

Right after the <properties> tag, and before the <dependencies> tag add the following <parent>:

```
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.3.5</version>
    </parent>
```

Inside <dependencies> add the following:

```
        <!-- automatically reloads app when development files change -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <optional>true</optional>
        </dependency>
        <!-- Spring Boot Starters -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>
```

```xml
        <!-- needed for JSP / JSTL -->
        <dependency>
            <groupId>org.apache.tomcat.embed</groupId>
            <artifactId>tomcat-embed-jasper</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>jakarta.servlet.jsp.jstl</groupId>
            <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
            <version>3.0.0</version>
        </dependency>
        <dependency>
            <groupId>org.glassfish.web</groupId>
            <artifactId>jakarta.servlet.jsp.jstl</artifactId>
            <version>3.0.1</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-taglibs</artifactId>
        </dependency>
        <!-- needed for MySQL -->
        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
        </dependency>
```

Inside the <plugins> add:

```xml
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
```

The Exercise:

Delete the MyWebAppInitializer.java and Config.java classes (don’t delete your SecurityConfig). Then create the following App.java class that configures and starts Spring Boot: (see next page)

```java
package cs544;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class App {

    public static void main(String[] args) {
        SpringApplication.run(App.class, args);
    }
}
```

Lastly create an application.properties file inside the project's resources folder with:

```
spring.datasource.url = jdbc:mysql://localhost/cs544?
useSSL=false&serverTimezone=America/Chicago

spring.datasource.username = root
spring.datasource.password = root

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.ddl-auto = create-drop
spring.jpa.open-in-view=true

spring.mvc.view.prefix=/WEB-INF/view/
spring.mvc.view.suffix=.jsp

logging.level.root=WARN
```

Run your Application class to see the application running. Remember that it will start an embedded Tomcat server, which means that you should make sure that you don't have any other Webserver running on port 8080.

You should be able to run your spring boot application simply by running **main()** or by going to the terminal and then executing the command inside your project folder: **`mvn spring-boot:run`**

I'm not 100% sure why, but for Spring Boot I have to update my security config to allow it to use the JSP pages, like so:

```java
.requestMatchers(HttpMethod.GET, "/WEB-INF/view/**").hasRole("USER")
```

If you get errors related to the MySQL timezone (I did when I tried to run the application). Then be sure to set the Timezone on the datasource.url (see properties file above).

REST (RST)

Exercise RST1 –Restful Web Service

The Setup:

The purpose of this exercise is to add a Restful API to our Book application.

Start by copying your Spring Boot application, and renaming it to **W3D3-REST**. Be sure to update the <artifactId> inside the pom.xml as well.

Add the following dependency to your pom:

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
</dependency>
```

Remove the spring-boot-starter-security from your pom:

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

The Exercise:

Part A, creating a web-service:

- Delete your SecurityConfig.java file. Security for Restful webservices is generally done through an OAuth token, which is unfortunately outside the scope of this course. So for now we'll go without security.

- Then create a BookRestController with the BookService injected and the following methods. Be sure to add the required RequestMappings.

```
public List<Book> getAll() {
public Book get(@PathVariable int id) {
public RedirectView add(@RequestBody Book book) {
public void update(@RequestBody Book book) {
public void delete(@PathVariable int id) {
```

- Note: the JSON converter cannot use the OpenEntityManagerInView filter, which means that it will break if it needs to do any lazy loading. Because of this I recommend having the get() method on the BookService use findById() on the BookDao instead of .getById().

- You can test the GET mappings of your web-service with a web browser, and I recommend downloading and installing Postman to test your POST/PUT/DELETE mapped methods.

See the next page for part B

Part B: creating a web-service-client:

- Create a new Spring Boot project called **W3D3-REST-Client.** For this part you can more or less copy the RestTemplate slides. You will only need the following dependencies (remember to also include the spring-boot plugin)

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
</dependency>
```

- Also create an application.properties file with:

```
spring.main.web-application-type=NONE
```

- Create an @SpringBootApplication App.java that has a @Bean for the RestTemplate

- Copy the book class from **W3D3-REST** and remove all the annotations (no need for them on this side) so that we can send and receive Book objects with the same properties. You may also want to put a .toString() method on as it can come in handy.

- If you want you can make an interface for BookService and have that use the RestTemplate, but it's not required.

- Create an Client.java that implements CommandLineRunner, and uses the RestTemplate to test the methods of our web service. I recommend the sequence of calls shown below.

    - make a call to whatever url you mapped getAll() on and print the result
    - make a call to the url that you mapped add() on (giving it a new book)
    - make a call to the url that you mapped update() on (giving it an updated version)
    - make a call to the url that you mapped delete() on (deleting one of the books)
    - make a call to the url that you mapped getAll() on and print the result again
    - make a call to the url that you mapped get() to check that you can get one book

- Test your client to see if everything works

JSON Web Token (JWT)

Exercise JWT1

The Setup:
The purpose of this exercise is to give you the experience of using a JWT, and provide you with the code to be able to add JWTs to your own projects.

Start by downloading the JWT project, it should already have all the dependencies inside its pom.xml.

The Exercise:
Run the spring boot application and then use a API request tool like Postman or Advanced Rest Client to send a POST request to: http://localhost:8080/adsweb/api/v1/login

With the folling in the request body (raw input):

```
{
"username": "user1",
"password": "password"
}
```

If you open **cs544/jwt/App.java** you will see the code that creates all the users and a variety of other entities. This code was originally used for a lab in a different course and still has some of the stuff for that lab in it (I plan to clean it up in the near future).

Once you've sent a succesful login request it should give you a token in the response body.

Next create a GET request to: http://localhost:8080/adsweb/api/v1/patients

And in the Authorization section select "Bearer" to provide your token for authentication and authorization.

If everything goes well you should get a JSON object containing the data for 4 patients.

To submit simply let me know in the textfield that everything worked for you.