Enterprise Java with Spring
Spring REST API
Lab 2
Exercises

1 Lab setup

Make sure you have the following items installed

- Latest LTS JDK version (at this point: JDK 21)
- Spring Tool Suite (STS) or IntelliJ IDEA
- Latest version of Maven (at this point: Maven 3.9.9)
- A free account at Postman and installed the Postman app
- A suitable text editor (Notepad ++)
- A utility to extract zip files (7-zip)

2 REST HTTP methods and response types

Start up STS. Ensure you are in the Java EE perspective.

Go to File -> New -> Other -> Spring Boot -> Spring Starter Project. Complete it with the following details:

Name: `RestMethodsExercisePt1`
Group: `com.exercise.rest`
Artifact: `RestMethodsResponsePt1`
Version: `0.0.1-SNAPSHOT`
Description: `Demo Spring Rest methods and response types`
Package: `com.exercise.rest`

Click Next.

Add the following dependencies:
`Web -> Spring Web`
`Developer Tools -> Spring Boot Dev Tools`

2.1 Testing GET and POST

In the package `com.exercise.rest` in `src/main/java`, create a class `Hero` with the following fields/properties:

- name (String)
- age (int)
- superpower (String)

Generate a constructor, getter / setter methods for the fields and toString method for the class using the Source options from the context menu in Eclipse (right click in Editor Tab).

In the @SpringBootApplication class (RestMethodsExerciseApplication), create a @Bean method that returns an ArrayList of Hero objects. This method will create 4 Hero objects with random values for their fields, adds all of them to the ArrayList and returns the ArrayList.

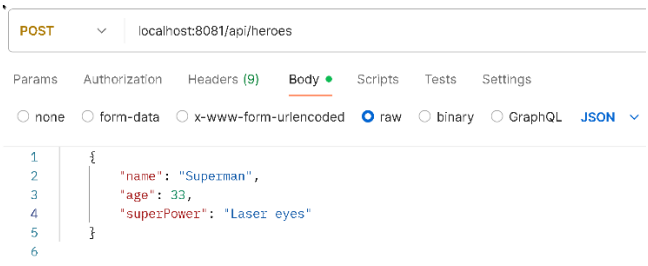
Java ArrayList sample tutorials:

https://www.w3schools.com/java/java_arraylist.asp
https://www.programiz.com/java-programming/arraylist

Create a class `HeroController` that

- Provides a top level mapping `/api`
- Initializes an ArrayList of Hero objects from the @Bean method created earlier using @Autowired
- Provides a GET mapping of `/heroes`, which returns all the objects in the ArrayList.
- Provide a POST mapping of `/heroes`, which includes a body with JSON content corresponding to all the complete fields of a Hero object, and add this object to the existing Hero ArrayList.

Test sending a POST with suitable JSON content, for e.g.



And verify that the JSON content was placed into a object that was added successfully to the ArrayList and then returned via a subsequent GET to `/heroes`

3 Accessing Path and Query parameters

Make a copy of the project `RestMethodsExercisePt1` and rename it to `RestMethodsExercisePt2`

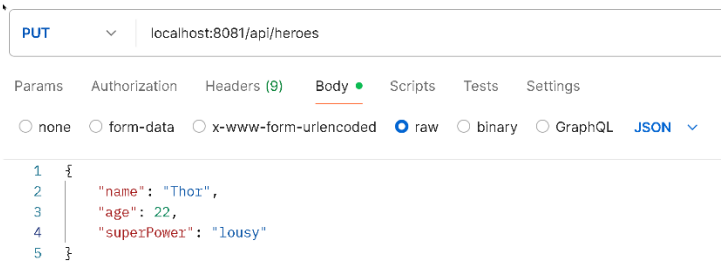
Update `HeroController` with the following mappings:

3.1 Using path parameters with GET, PUT and DELETE mapping

Add a GET mapping with one path parameter to specify the name of the Hero object to return
For e.g.
`/heroes/Thor` returns the hero with the name of Thor
`/heroes/Hulk` returns the hero with the name of Hulk

Add a GET mapping for two path parameters to specify search for a Hero by either their age or superpower.
For e.g.
`/heroes/age/33` returns the hero with the `age` field value of 33
`/heroes/age/60` returns the hero with the `age` field value of 60
`/heroes/superpower/thunder` returns the hero with the `superpower` field value of `thunder`
`/heroes/superpower/flying` returns the hero with the `superpower` field value of `flying`

Add a PUT mapping which provides a Hero object content in JSON and replaces the values of the fields of an existing Hero object with the same name.
For e.g. sending a PUT to `/heroes` with the JSON content below



will locate a Hero object with the name of `Thor` in the ArrayList and update its `age` and `superpower` field values with the values from the JSON content (22 and lousy).

Add a DELETE mapping with one path parameter to specify the name of the Hero object to delete from the ArrayList
For e.g.
`/heroes/Thor` removes the hero with the name of Thor from the ArrayList
`/heroes/Hulk` removes the hero with the name of Hulk from the ArrayList

Test that your PUT and DELETE mapping performs the require changes to the Hero objects in the ArrayList by performing a GET to `/heroes`

3.2 Using query parameters for GET mapping

Add a GET mapping for `/heroparams` that accepts two query parameters, the first is for the hero `name` and the second is for the hero `age`, where the second one is optional

For e.g.
`heroparams?name=thor` returns the hero with the name of Thor, regardless of his age
`heroparams?name=thor&age=1500` returns the hero with the name of Thor and the exact age of 1500. A hero with the name of Thor and a non-matching age value will result in nothing being returned.