Creating and consuming a simple web API

*Note: Look up the meanings of the terms in **bold.***

Introduction

Last week, a simple introduction to spring and spring boot was made. This week, we will create two simple programs. One to serve greetings over a web API when called and the other to consume this service.

What is an API?

APIs are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols (Source). In this particular case, it is via http.

Prerequisites

- To have completed the practical sheet 4 – Introduction to Spring framework.
- Optional: Install Postman API platform tool.

Part 1: Creating the API

1. Create a new Spring Boot project using the *Spring initializr* through VS code. For this exercise, select only the *Spring Web* dependency. Name the project as *GreetingAPI*. Please refer the previous week's practical sheet for more information on creating a new project.

2. The resource with which we are going to respond to a call to our API is a Greeting. The greeting should consist of a greeting along with a unique id for each greeting. Create a new file called *Greeting.java* to represent a model greeting we are going to respond with. Inside this file, create a new class called *Greeting.* It is called a *resource representation class.*

Greeting.java

```java
package com.example.greetingapi;

// a record class for a greeting response.
// read more: https://docs.oracle.com/en/java/javase/15/language/records.html
public record Greeting(long id, String content) { }
```

Greeting class

3. You may notice that the structure of the class is quite different from a normal class structure in Java that is familiar to you. This type of class is called a ***Record class*** and they are intended to act as "data carriers". Functionally it is still similar to a regular class in Java, but more concise. Read more at this link.

4. In Spring's approach to building web APIs, HTTP requests are handled by a controller. These components are identified by the *@RestController* annotation. A Controller is simply a class

which has methods and logic to handle calls made to predefined **endpoints** in our web API. Create new file called *GreetingController.java*. Implement the *GreetingController* class in this file. Make sure to include the appropriate dependencies when writing code for the class.

```java
// controller for Greeting releated endpoints
@RestController
public class GreetingController {

    private static final String template = "Hello, %s!";

    private final AtomicLong count = new AtomicLong();

    // endpoint for responding to calls for /greeting
    @GetMapping("/greeting")
    public Greeting greeting() {
        return new Greeting(count.incrementAndGet(), String.format(template, "World"));
    }

    // endpoint for responding to calls for /greeting/name?name=<your_name>
    @GetMapping("/greeting/name")
    public Greeting greeting(@RequestParam(value = "name", defaultValue = "<Your name>") String name) {
        return new Greeting(count.incrementAndGet(), String.format(template, name));
    }

}
```

GreetingController class – observe the coding style. @RestController/ @GetMapping annotations were discussed in the previous lab sheet.

5. Save everything and run the project.

6. Once the project is running, navigate to http://localhost:8080/greeting through the web browser and observe the response.

7. Then navigate to http://localhost:8080/greeting/name?name=<your_name_here> and observe the response.

Note: For steps 6 and 7, the Postman tool may be used instead of the web browser.

Part 2: Creating the API consuming application

It's nice to be able to access a resource through a browser or through a tool, but it is not very useful. A more useful way of consuming a resource is via a program which is going to be the next step.

1. Create a new Spring Boot project via the *Spring initializr* through VS code. Name the project as Getgreetings.

2. To contain the data that is to be received from the GreetingAPI, we need to have a *domain class*. Create *Greeting.java* and implement the Greeting class. This again is a *record* class.

Greeting.java

```java
package com.example.getgreetings;

// a record class for a greeting response.
// read more: https://docs.oracle.com/en/java/javase/15/language/records.html
public record Greeting(long id, String content) { }

```

Greeting class

3. Then, in the *GetgreetingsApplication* class, add the methods *getHttpClient(), getGreeting(), getGreetingByName(), makeCalls()* to consume the services in API as below. A comment above each method explains what each of them is for.

```java
@SpringBootApplication
public class GetgreetingsApplication {

    // client for performing HTTP requests
    private static RestTemplate httpClient = null;

    // base url for remote calls
    private static String baseURL = "http://localhost:8080/";

    // endpoints for remote calls
    private static String defaultGreetingURL = "greeting";
    private static String namedGreetingURL = "greeting/name?name=<your_name_here>";

    // main method
    public static void main(String[] args) {…
    }

    // singleton pattern implemented to get a single instnace of the http client
    private static RestTemplate getHttpClient() {…
    }

    // call the default endpoint and get the response
    private static Greeting getGreeting(String url) {…
    }

    // call the named endpoint and get the response
    private static Greeting getGreetingByName(String url) {…
    }

    // call the endpoints, receive the responses and print them on the console
    private static void makeCalls() {…
    }
```

Overall structure of GetgreetingsApplication class - observe the coding style.

4. The main method once expanded looks like this. It only contains a method called *makeCalls().*

```
    // main method
    public static void main(String[] args) {
        SpringApplication.run(GetgreetingsApplication.class, args);

        makeCalls();
    }
```

5. The *makeCalls()* method looks like this. It makes calls to *getGreeting()* and *getGreetingByName()* methods, then prints the contents of each response.

```
    // call the endpoints, receive the responses and print them on the console
    private static void makeCalls() {
        Greeting receivedGreeting1 = GetgreetingsApplication.getGreeting(defaultGreetingURL);
        Greeting receivedGreeting2 = GetgreetingsApplication.getGreetingByName(namedGreetingURL);

        String content1 = receivedGreeting1.content();
        System.out.println(content1);

        String content2 = receivedGreeting2.content();
        System.out.println(content2);
    }
```

6. *getGreeting()* and *getGreetingByName()* methods look like this. Both of them call *getHttpClient()* which returns an object of type ***RestTemplate***. Read about *RestTemplate* here.

```
    // call the default endpoint and get the response
    private static Greeting getGreeting(String url) {
        RestTemplate restmp = getHttpClient();
        Greeting response = restmp.getForObject(baseURL + "/" + url, Greeting.class);

        return response;
    }

    // call the named endpoint and get the response
    private static Greeting getGreetingByName(String url) {
        RestTemplate restmp = getHttpClient();
        Greeting response = restmp.getForObject(baseURL + "/" + url, Greeting.class);

        return response;
    }

```

7. The *getHttpClient()* method looks like this.

```
    // singleton pattern implemented to get a single instnace of the http client
    private static RestTemplate getHttpClient() {

        if (httpClient == null) {
            httpClient = new RestTemplate();
        }
        return httpClient;
    }
```

8. Complete the class and try to run the program. Please include the dependencies as required.

9. Does the program run? If it does not, try and identify the cause of the issue and fix it.

10. Once the issue is fixed, try running it again.

Part 3: Self-learning activity

1. Add a new endpoint in the API program to send today's date along with the greeting.

2. Modify the Getgreetings application to receive this new type of greeting as well.

3. Can the API you developed be considered as a REST API? Find out.

Note: The original API endpoints also should work along with the new endpoint.