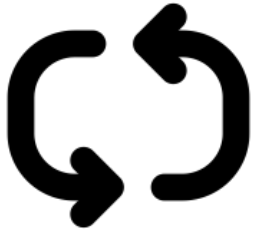
REST APIs

Part 2

MARIANO CECCATO (mariano.ceccato@univr.it)

SOFIA MARI (sofia.mari@univr.it)

Table of contents



Recap

REST Assured

Class Exercise

Homework

A brief recap

Get Books

Create a Book

Get a specific book

Update a book

Delete a Book

A brief recap

GET /books

Create a Book

Get a specific book

Update a book

Delete a Book

A brief recap

GET /books

Get a specific book

Delete a Book

POST /book + query parameter

Update a book

A brief recap

GET /books

POST /book + query parameter

GET /book/{bookId}

Update a book

Delete a Book

A brief recap

GET /books

GET /book/{bookId}

Delete a Book

POST /book + query parameter

PUT /book + request body

A brief recap

GET /books

GET /book/{bookId}

DELETE /book + query parameter

POST /book + query parameter

PUT /book + request body

Testing REST APIs

POSTMAN

REST ASSURED

REST Assured

- Testing is a **crucial** process for deploying **high-quality** software.
- REST Assured is a framework to perform **end-to-end testing** for a Springboot Application.
- REST Assured is a is a **Java library** for testing and validating Restful web services.
- Provides **primitives** to launch HTTP requests, read the contents of responses, and make assertions.

Example

```
@Test
void shouldCreateArticle(){
    given()
    .contentType(ContentType.JSON)
    .body("""
                {
                    "name": "Test Title"
                }
            """
    )
    .header("Authorization", "Bearer " + token)
    .when()
    .post("/api/articles")
    .then()
    .statusCode(201)
    .body("id", notNullValue())
    .body("name", equalsTo("Test Title"));
}
```

`given()`: is used to specify request headers, body, cookies, and authentication.

`when()`: is used to specify the HTTP method (GET, POST, PUT, PATCH, DELETE) of the request.

`then()`: contains the response to be validated.

Setting up REST-assured

In the build.gradle file, add the REST-assured library as a dependency.

```
dependencies {
        implementation('org.springframework.boot:spring-boot-starter-web')
        implementation('org.springframework.boot:spring-boot-starter-data-jpa')
        runtimeOnly 'com.h2database:h2'
        testImplementation 'org.springframework.boot:spring-boot-starter-test'
        testImplementation 'io.rest-assured:rest-assured:5.3.0'
        testImplementation 'org.junit.jupiter:junit-jupiter:5.10.0’

}
```

TestRest Class

Set up the infrastructure (e.g. launching the server)

```
@ExtendWith(SpringExtension.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.DEFINED_PORT)

public class RestTest {
	@BeforeAll
	public static void setBaseUri() {
		RestAssured.baseURI = "http://localhost:8080";
	}

	. . .
```

Configure REST-assured with the baseURI of our REST API

Test case for *create book*

```
@Test
@DirtiesContext(methodMode =
DirtiesContext.MethodMode.AFTER_METHOD)

public void createBookTest(){
      final String TITLE = "My book";
      final String AUTHOR = "Sofia";
      final float PRICE = 13.50F;

      given()
          .queryParam("title", TITLE)
          .queryParam("author", AUTHOR)
          .queryParam("price", PRICE)
      .when()
          .post("/book")
      .then()
          .statusCode(200)
          .body("id", Matchers.greaterThan(0))
          .body("title", Matchers.is(TITLE))
          .body("author", Matchers.is(AUTHOR))
          .body("price", Matchers.is(PRICE));
}
```

Reset the database after the test to avoid side effects

given this input data ...

when performing this action

Assert this requirements

```
@Test
@DirtiesContext(methodMode =
DirtiesContext.MethodMode.AFTER_METHOD)

public void retrieveBookTest() {
     final String TITLE = "My book";
     final String AUTHOR = "Sofia";
     final float PRICE = 13.50F;
     int id =
       given()
        .queryParam("title", TITLE)
        .queryParam("author", AUTHOR)
        .queryParam("price", PRICE)
       .when()
        .post("/book")
       .then()
        .statusCode(200)
        .body("id", Matchers.greaterThan(0))
        .body("title", Matchers.is(TITLE))
        .body("author", Matchers.is(AUTHOR))
        .body("price", Matchers.is(PRICE))
        .extract().path("id");
. . .
```

The value of the property *id* is extracted and saved into the variable *id*

Test case for *retrieve book*

(1/2)

Test case for *retrieve book*

(2/2)

```
        . . .
        given()
            .pathParam("bookId", id)
        .when()
            .get("/book/{bookId}")
        .then()
            .statusCode(200)
            .body("id", Matchers.is(id))
            .body("title", Matchers.is(TITLE))
            .body("author", Matchers.is(AUTHOR))
            .body("price", Matchers.is(PRICE));
}
```

Test case for *missing book*

```
@Test
@DirtiesContext(methodMode =
DirtiesContext.MethodMode.AFTER_METHOD)

public void missingBookTest() {
     given()
        .pathParam("bookId", 123)
     .when()
        .get("/book/{bookId}")
     .then()
        .body(Matchers.is("null"));
}
```

Exercise

Starting from your own implementation of the bookstore implement the following test cases using REST-assured.

TEST CASE 1:

- Create a book.
- Update the book with new values.
- Retrieve the same book with a `GET` request.
- Check if the retrieved book has the new values.

TEST CASE 2:

- Create a book.
- Delete the book.
- Try to retrieve the deleted book with a `GET` request.
- Check that the response body contains “null”.

When sending JSON objects as request bodies, remember to set the content type correctly in the HTTP header (*`given().contentType("application/json").body("{JSON object}")`*)

Homework

- Implement a REST API service to handle e-books based only on the provided test cases written in RESTAssured.
- From the test cases you can get the information you need for implementation such as **available operations**, **input parameters**, and **response formats**.
- Download the template with the test cases from: https://github.com/SofiaMari/Progettazione-Validazione-Sistemi-Software