DIGITALHOUSE

GET, POST, UPDATE Y DELETE CON REST ASSURED

Certified Tech Developer
The Ultimate Degree

Índice

Rest Assured

- Es una biblioteca Java o API para probar servicios web RESTful.
- Se utiliza para probar servicios web basados en XML y JSON.
- Admite diferentes métodos HTTP: GET, POST, PUT, DELETE, OPTIONS, PATCH, HEAD.
- Se puede integrar con marcos de prueba como JUnit, TestNG.

01 GET

GET

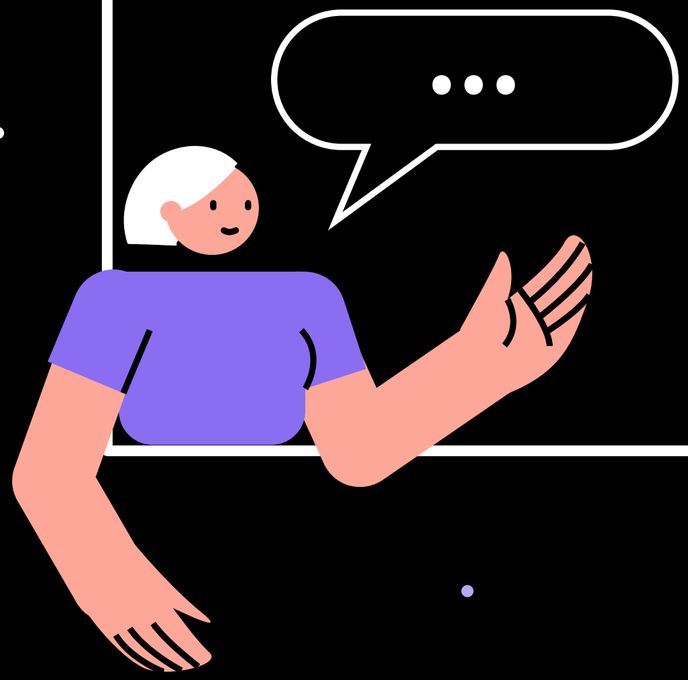
1. Lo primero que debemos hacer es realizar la petición GET al endpoint correspondiente y lo almacenamos en una variable.
2. Como vemos en este apartado podemos mostrar diferente información obtenida desde el endpoint consultado.
3. El siguiente paso es realizar las validaciones necesarias para verificar que la información obtenida sea la deseada, validar los error code HTTP, etc. Podemos utilizar diferentes Assert() para validar la información.

```
public class Test01_GET {

 @Test

 void test01() {

      Response response =

①     RestAssured.get("https://reqres.in/api/users?page=2");

    System.out.println(response.getBody().asString());

    System.out.println(response.asString());

②   System.out.println(response.getStatusCode());

    System.out.println(response.getHeader("content-type"));

    System.out.println(response.getTime());


 // Validar status code

       int statusCode = response.getStatusCode();

③      Assert.assertEquals(statusCode, 200);

 // Validar contenido body

       String body = response.getBody().asString();

       Assert.assertEquals(body.data.id[0].email,

       “michael.lawson@reqres.in”);

 }}
```

Rest Assured+Gherkin

Rest Assured nos permite utilizar Gherkin.
Gherkin define un lenguaje comprensible por humanos y por ordenadores, con el que vamos a describir las funcionalidades, definiendo el comportamiento del software, sin entrar en su implementación.
Sólo nos hace falta conocer 5 palabras para empezar a utilizar BDD.

- FEATURE: Nombre de la funcionalidad que vamos a probar
- SCENARIO: Describe cada escenario que vamos a probar.
- GIVEN: Contexto para el escenario en que se va a ejecutar el test.
- WHEN: Conjunto de acciones que lanzan el test.
- THEN: Especifica el resultado esperado en el test.



GET

1. Podemos utilizar la nomenclatura de Gherkin para escribir nuestros tests. De manera que nuestro código sea más legible por otras personas.

```
public class Test01_GET {
 @Test
 void test01() {
       given().
       get("https://reqres.in/api/users?page=2").
       then().
       statusCode(200).body("data.id[0]", equalTo(7));
 }}
```

① get("https://reqres.in/api/users?page=2").

02 POST

POST

1. Lo primero que debemos hacer es crear un objeto del tipo JSONObject(), el cual enviará toda la información que viajará por el body de nuestra request.
2. Como vemos, estamos utilizando Gherkin, de manera tal que en nuestro given() vamos a configurar todos los parámetros necesarios para enviar nuestra petición.
3. Hacemos una petición POST a nuestra API.
4. Finalmente realizamos todas las validaciones necesarias para verificar que la información obtenida sea la deseada, validar los error code HTTP, etc

```
public class Test01_POST {
 @Test
 void test01() {
      Jt request = new JSONObject();
①     request.put("name"SONObjec, "Ajeet");
      request.put("Job", "Teacher");
          given().
          header("Content-type","application/json").
②         contentType(ContentType.JSON).
          body(request.toJSONString()).
          when().
③         post("https://reqres.in/api/users").
          then().
④         statusCode(201).log().all();
 }}
```

03 PUT

PUT

1. Lo primero que debemos hacer es crear un objeto del tipo JSONObject(), el cual enviará toda la información que viajará por el body de nuestra request.
2. Configuramos todos los parámetros necesarios para enviar nuestra petición.
3. Hacemos una petición PUT a nuestra API.
4. Finalmente realizamos todas las validaciones necesarias para verificar que la información obtenida sea la deseada, validar los error code HTTP, etc.

```
public class Test01_PUT {
 @Test
 void test01() {
    JSONObject request = new JSONObject();
①    request.put("name", "Amann");
     request.put("Job", "Teacher");
②        given().
         header("Content-type","application/json").
         contentType(ContentType.JSON).
         body(request.toJSONString()).
         when().
③        put("https://reqres.in/api/users/2").
④        then().
         statusCode(201).log().all();
 }}
```

04 DELETE

DELETE

1. Realizamos la petición DELETE al endpoint correspondiente.
2. Finalmente realizamos todas las validaciones necesarias para verificar que la información obtenida sea la deseada, validar los error code HTTP, etc.

```
public class Test01_DELETE {
 @Test
 void test01() {
          given().
①        delete("https://reqres.in/api/users/2").
          then().
②        statusCode(204).log().all();
 }}
```

DIGITALHOUSE

¡Muchas gracias!