Material descargable

egg

API Testing

¿Qué es Rest Assured?

Rest Assured es una biblioteca de pruebas automatizadas en Java, diseñada específicamente para realizar pruebas de APIs RESTful. Proporciona una interfaz sencilla para enviar solicitudes HTTP a una API, validar las respuestas y realizar varias comprobaciones, como verificar el contenido de las respuestas, el estado de la API, y mucho más.

BDD (Behavior-Driven Development)

BDD es una metodología de desarrollo que enfoca las pruebas en el comportamiento del sistema, usando un lenguaje comprensible tanto para desarrolladores como para no técnicos. BDD utiliza una estructura que facilita la colaboración y la comprensión de los requisitos del sistema a través de ejemplos concretos, usualmente utilizando la sintaxis *Given-When-Then*. Esta estructura también se aplica a las pruebas, proporcionando claridad y mejor documentación sobre lo que se espera que suceda en cada escenario de prueba.

- **Given (Dado):** Describe las condiciones previas o el estado inicial del sistema antes de realizar cualquier acción.
- **When (Cuando):** Define la acción que se lleva a cabo, es decir, la solicitud o evento que estamos probando.
- **Then (Entonces):** Especifica lo que se espera que suceda como resultado de la acción. Normalmente, esto incluye la validación de la respuesta.

.

¿Cuáles son las diferencias entre Postman y Rest Assured?

Aunque ambas herramientas están diseñadas para probar APIs, tienen diferencias en su uso y propósitos:

- **Postman:** Es una herramienta gráfica de interfaz para pruebas manuales, que permite enviar solicitudes a una API y ver las respuestas. También se pueden escribir pruebas en JavaScript, pero su enfoque principal es en pruebas manuales y exploratorias.

- **Rest Assured:** Es una biblioteca para Java que permite realizar pruebas automatizadas en APIs RESTful. A diferencia de Postman, Rest Assured se usa para crear pruebas de integración automatizadas, lo que lo hace ideal para ser integrado en pipelines de CI/CD y en procesos automatizados.

Incorporación de dependencias necesarias en tu proyecto Maven

En tu proyecto Maven, debes incorporar las siguientes dependencias para poder utilizar Rest Assured y realizar pruebas automatizadas. Asegúrate de incluir tanto la dependencia de **JUnit** como la de **Rest Assured** para garantizar el correcto funcionamiento de las pruebas.

```
<!-- Dependencia de Rest Assured -->
        <dependency>
            <groupId>io.rest-assured</groupId>
            <artifactId>rest-assured</artifactId>
            <version>5.4.0</version> <!-- Verifica la
última versión en el repositorio de Maven -->
            <scope>test</scope>
        </dependency>
```

Algunos métodos de Rest Assured, como los que permiten realizar verificaciones más específicas, como `statusCode()`, pueden requerir la dependencia de Hamcrest para las aserciones.

```
<dependency>
            <groupId>org.hamcrest</groupId>
            <artifactId>hamcrest</artifactId>
            <version>2.2</version> <!-- O la última versión
disponible -->
            <scope>test</scope>
</dependency>
```

Estructura de los Métodos de Rest Assured

Al crear pruebas con Rest Assured, usamos los métodos dentro de los bloques **`given()`**, **`when()`** y **`then()`** para estructurar las pruebas siguiendo el enfoque BDD. Aquí se presentan algunos de los métodos más comunes utilizados en estos bloques:

Existen diversas formas de declarar métodos para implementar el uso de Rest Assured, y esto puede variar según las necesidades específicas de tu proyecto. A continuación, te mostramos una estructura básica y práctica para declarar un método de prueba, que puedes adaptar según los requisitos del proyecto en el que estés trabajando:

```
@Test
public void nombreDelTest() {
    given() // Métodos específicos para la preparación de la
solicitud
        .when() // Métodos que definen la acción a realizar
        .then() // Métodos que validan la respuesta obtenida
}
```

1. . `given()`

Inicia la construcción de la solicitud HTTP. Aunque puedes configurar diversos aspectos de la solicitud, como encabezados, parámetros, cookies y más, no es necesario configurar nada adicional si no es requerido por la prueba específica. Algunos métodos comunes que puedes usar en **given()** son:

- **contentType("application/json")**: Especifica el tipo de contenido que se espera en la solicitud, por ejemplo, JSON. Esto es útil cuando se está enviando información en el cuerpo de la solicitud. Ejemplo:

```
given().contentType("application/json");
```

- **body(requestBody)**: Establece el cuerpo de la solicitud, donde puedes incluir datos que serán enviados al servidor (como un objeto JSON o una cadena de texto). Ejemplo:

```
given().body(requestBody);
```

- **header("Authorization", "Bearer token")**: Para agregar encabezados personalizados, como un token de autorización. Ejemplo:

```
given().header("Authorization", "Bearer token");
```

- **param("userId", 1)**: Para agregar parámetros a la URL de la solicitud. Ejemplo:

```
given().param("userId", 1);
```

- **cookie("session", "123abc")**: Para agregar cookies a la solicitud.

```
given().cookie("session", "123abc");
```

- **baseUri("https://api.example.com")**: Para definir una URL base, si es necesario.

```
given().baseUri("https://api.example.com");
```

2. . when()

Define la acción o el tipo de solicitud HTTP que se realizará sobre un endpoint específico. Dependiendo de la operación que quieras probar, puedes utilizar métodos como los siguientes para especificar la acción que se llevará a cabo sobre el recurso:

- **get("URL")**:Para realizar una solicitud GET, que recupera datos de un endpoint. Ejemplo:

```
when().get("https://jsonplaceholder.typicode.com/posts");
```

- **post("URL")**: Para realizar una solicitud POST, que envía datos a un endpoint para crear un nuevo recurso.

```
when().post("https://jsonplaceholder.typicode.com/posts");
```

- **put("URL")**: Para realizar una solicitud PUT, que actualiza un recurso existente en el endpoint.

```
when().put("https://jsonplaceholder.typicode.com/posts/1");
```

- **delete("URL")**: Para realizar una solicitud DELETE, que elimina un recurso en el endpoint.

```
when().delete("https://jsonplaceholder.typicode.com/posts/1");
```

3. . then()

Es utilizado para validar la respuesta de la solicitud. En este paso, verificamos diferentes aspectos de la respuesta recibida, como el código de estado, el cuerpo de la respuesta y otros detalles.

Puedes usar varios métodos dentro de **then()** para realizar diferentes tipos de validaciones. Algunos ejemplos comunes son:

- **statusCode(int code)**: Verifica que el código de estado de la respuesta sea igual al valor especificado. El código de estado puede variar según la acción realizada y lo que se espera de la solicitud. Por ejemplo, **200** indica una solicitud exitosa, pero otros códigos como **404** (No encontrado) o **500** (Error del servidor) también pueden ser relevantes dependiendo de lo que estés probando. Ejemplo:

```
then().statusCode(200);  // Verifica que el código de estado sea 200
```

- **body(String path, Matcher<?> matcher)**: Permite validar el contenido del cuerpo de la respuesta. En este caso, se puede verificar el tamaño de una lista o el valor de un campo específico, dependiendo de la estructura de la respuesta. Este tipo de validación es útil cuando se espera una lista de resultados o un campo específico en la respuesta.

```
then().body("size()", greaterThan(5));  // Verifica que el tamaño de
la lista sea mayor a un valor especificado
```

- **body("field", equalTo(value))**: Verifica que el valor de un campo en el cuerpo de la respuesta sea igual a un valor específico.

```
then().body("title", equalTo("Post 1"));  // Verifica que el campo
'title' sea igual a "Post 1"
```

- **contentType("application/json")**: Valida que el tipo de contenido de la respuesta sea **application/json**, común en APIs que devuelven datos en formato JSON.

```
then().contentType("application/json");  // Verifica que el tipo de
contenido sea "application/json"
```

Aclaración sobre Métodos Usados en `given()` y `then()`

Algunos métodos, como **contentType**, pueden ser utilizados tanto en **given()** como en **then()**, pero con diferentes propósitos:

- En **given()**, **contentType("application/json")** se usa para establecer el tipo de contenido de la solicitud, es decir, cómo enviarás los datos al servidor.
- En **then()**, **contentType("application/json")** se usa para validar que el tipo de contenido de la respuesta sea el esperado, es decir, cómo el servidor te está respondiendo.

Lo mismo ocurre con otros métodos, como **header()**, **body()**, y **param()**, que tienen diferentes roles según el bloque en el que se utilicen.

Ejemplos de implementaciones

Implementación solicitudes POST

```
@Test
public void postRequest() {
    String requestBody = "{\"title\": \"foo\", \"body\": \"bar\", \"userId\": 1}";

     given()
        .contentType(contentType:"application/json")
        .body(requestBody)
    .when()
        .post(path:"https://jsonplaceholder.typicode.com/posts")
    .then()
        .statusCode(expectedStatusCode:201)
        .body(path:"title", equalTo("foo"))
        .body(path:"body", equalTo("bar"))
        .body(path:"userId", equalTo(1));
}
```

Implementación solicitudes GET

```java
@Test
public void verifyStatusCode() {
    given()
    .when()
        .get(path:"https://jsonplaceholder.typicode.com/posts/1")
    .then()
        .statusCode(expectedStatusCode:200);
}
```

```java
    @Test
public void verifyUserId() {
    given()
    .when()
        .get(path:"https://jsonplaceholder.typicode.com/posts/1")
    .then()
        .body(path:"userId", equalTo(1));
}
```

Te invitamos a explorar el **sitio oficial** para obtener más información sobre su uso e implementación. Allí encontrarás ejemplos detallados y recursos que te ayudarán a comprender mejor cómo utilizarlo en tus proyectos.