

DESUENDANDO **REST** ASSURED

A FORÇA DOS TESTES JAVA



PRISCILA PEREIRA



Introdução AO Rest Assured

No universo da programação Java, a automação de testes de integração é essencial para garantir que todas as partes de uma aplicação funcionem harmoniosamente. O Rest Assured é uma ferramenta poderosa que facilita a criação e execução de testes automatizados para APIs RESTful. Ele permite aos desenvolvedores realizar requisições HTTP e validar respostas de maneira simples e eficiente.

Por que usar Rest Assured?

Rest Assured se destaca por sua simplicidade e poder. Com ele, é possível escrever testes de forma intuitiva, sem a necessidade de configurar muitos detalhes. Além disso, sua integração com frameworks populares de teste, como JUnit e TestNG, torna o processo de automação ainda mais fluido.

01

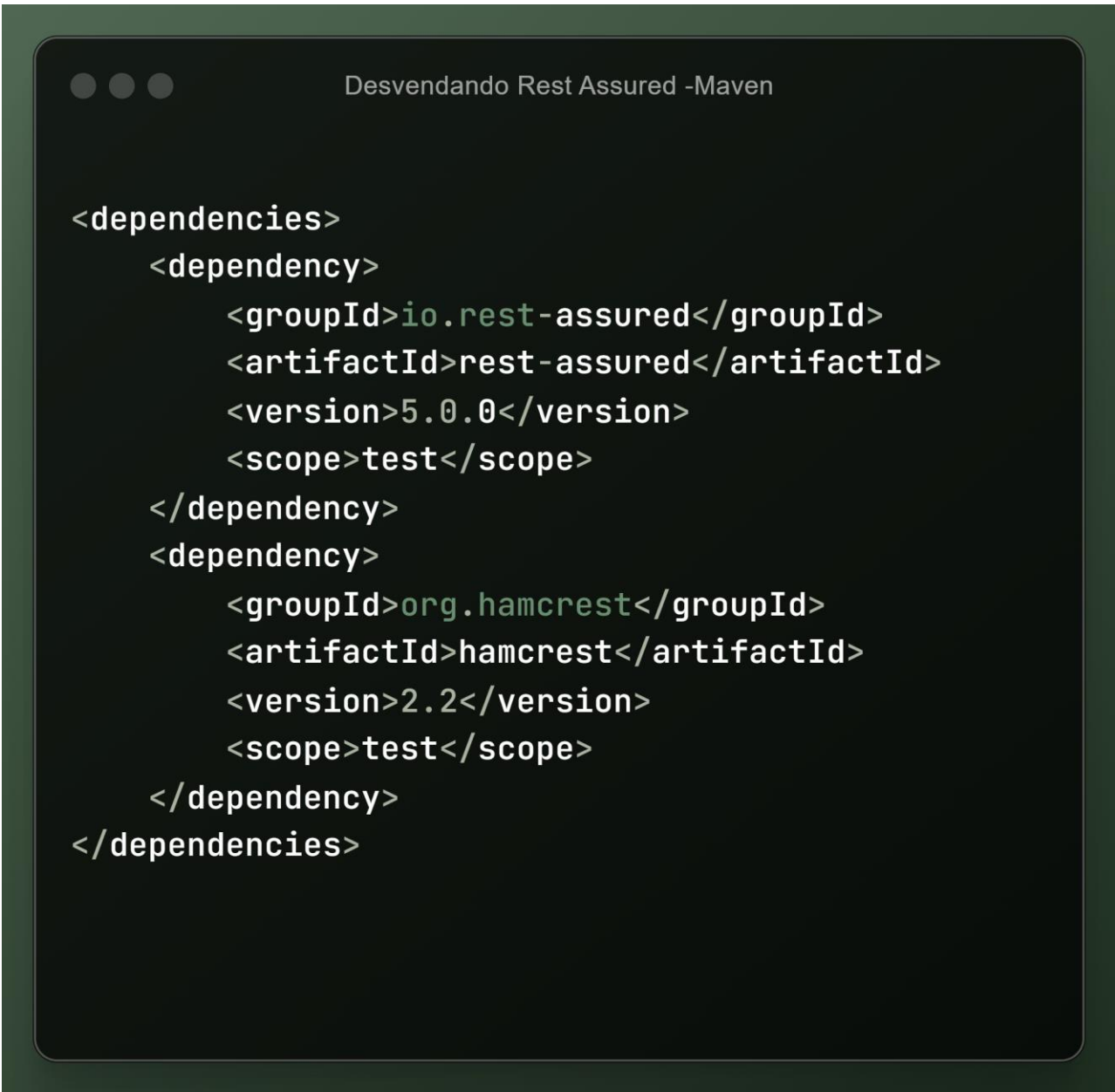
INSTALAÇÃO DO REST ASSURED

Instalando o Rest Assured

Para começar a usar o Rest Assured, você precisa adicionar as dependências do Maven ou Gradle ao seu projeto. Aqui está um guia passo a passo para instalação:

Usando Maven

1. Abra o arquivo pom.xml do seu projeto.
2. Adicione as seguintes dependências:




```
Desvendando Rest Assured -Maven

<dependencies>
  <dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>5.0.0</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest</artifactId>
    <version>2.2</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Instalando o Rest Assured

Usando Gradle

1. Abra o arquivo build.gradle do seu projeto.
2. Adicione as seguintes dependências:



```
dependencies {  
    testImplementation 'io.rest-assured:rest-assured:5.0.0'  
    testImplementation 'org.hamcrest:hamcrest:2.2'  
}
```

02

Requisições HTTP (GET, POST, PUT, DELETE)



Requisições HTTP (GET, POST, PUT, DELETE)

O Rest Assured é uma biblioteca poderosa para testes de integração de APIs RESTful em Java. Com ele, é possível realizar e validar requisições HTTP de forma simples e eficiente. Vamos explorar as requisições mais comuns: GET, POST, PUT e DELETE, com exemplos de código Java.

Requisição GET

A requisição GET é uma das operações mais comuns usadas em APIs RESTful. Ela é utilizada para recuperar dados de um servidor. Quando você faz uma requisição GET para um determinado endpoint, você está basicamente solicitando ao servidor que lhe envie uma representação dos dados armazenados nesse endpoint.

Requisições HTTP (GET, POST, PUT, DELETE)

Estrutura de uma Requisição GET

Aqui está um exemplo de como fazer uma requisição GET usando Rest Assured:

```
Desvendando Rest Assured - Requisição GET

import io.restassured.RestAssured;
import io.restassured.response.Response;
import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;

public class ApiTest {
    @Test
    public void testGet() {
        RestAssured.baseURI = "https://api.exemplo.com";

        Response response = given()
            .when()
            .get("/users/1")
            .then()
            .statusCode(200)
            .extract()
            .response();

        System.out.println("Response: " + response.asString());
    }
}
```

Neste exemplo:

RestAssured.baseURI: Define a URL base para as requisições.

given(): Configura a requisição, onde você pode adicionar headers, parâmetros, etc.

when(): Inicia a execução da requisição.

get("/users/1"): Especifica que estamos fazendo uma requisição GET para o endpoint /users/1.

then(): Permite fazer validações na resposta.

statusCode(200): Verifica se o servidor retornou o código de status 200.

extract().response(): Extrai a resposta da requisição para uso posterior.

Requisições HTTP (GET, POST, PUT, DELETE)

Requisição POST

A requisição POST é utilizada para enviar dados ao servidor. É geralmente usada para criar novos recursos. Quando fazemos uma requisição POST, estamos enviando uma "carga útil" que o servidor deve processar e, em muitos casos, adicionar aos dados existentes.

Estrutura de uma Requisição POST

Aqui está um exemplo de como fazer uma requisição POST usando Rest Assured:

```
Desvendando Rest Assured - Requisição GET

import io.restassured.RestAssured;
import io.restassured.response.Response;
import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;

public class ApiTest {
    @Test
    public void testGet() {
        RestAssured.baseURI = "https://api.exemplo.com";

        Response response = given()
            .when()
            .get("/users/1")
            .then()
            .statusCode(200)
            .extract()
            .response();

        System.out.println("Response: " + response.asString());
    }
}
```

Requisições HTTP (GET, POST, PUT, DELETE)

Neste exemplo anterior:

RestAssured.baseURI: Define a URL base para as requisições.

given(): Configura a requisição, onde você pode adicionar headers e o corpo da requisição.

header("Content-Type", "application/json"): Define o tipo de conteúdo da requisição como JSON.

body(...): Define o corpo da requisição, que neste caso é um JSON contendo os dados do novo usuário.

post("/users"): Especifica que estamos fazendo uma requisição POST para o endpoint /users.

then(): Permite fazer validações na resposta.

statusCode(201): Verifica se o servidor retornou o código de status 201.

body("name", equalTo("John Doe")): Valida que o nome retornado no corpo da resposta é "John Doe".

Requisições HTTP (GET, POST, PUT, DELETE)

Requisição PUT

A requisição PUT é usada para atualizar dados existentes no servidor. A diferença entre uma requisição PUT e uma POST é que a PUT é idempotente, o que significa que se você enviar a mesma requisição várias vezes, o resultado será o mesmo (o recurso será atualizado da mesma forma).

Estrutura de uma Requisição PUT

Aqui está um exemplo de como fazer uma requisição PUT usando Rest Assured:

```
Desvendando Rest Assured - Requisição PUT

import io.restassured.RestAssured;
import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;
import static org.hamcrest.Matchers.equalTo;

public class ApiTest {
    @Test
    public void testPut() {
        RestAssured.baseURI = "https://api.exemplo.com";

        given()
            .header("Content-Type", "application/json")
            .body("{ \"name\": \"Jane Doe\", \"email\": \"janedoe@example.com\" }")
        .when()
            .put("/users/1")
        .then()
            .statusCode(200)
            .body("name", equalTo("Jane Doe"));
    }
}
```

Requisições HTTP (GET, POST, PUT, DELETE)

No exemplo anterior:

RestAssured.baseURI: Define a URL base para as requisições.

given(): Configura a requisição, onde você pode adicionar headers e o corpo da requisição.

header("Content-Type", "application/json"): Define o tipo de conteúdo da requisição como JSON.

body(...): Define o corpo da requisição, que neste caso é um JSON contendo os dados atualizados do usuário.

put("/users/1"): Especifica que estamos fazendo uma requisição PUT para o endpoint /users/1.

then(): Permite fazer validações na resposta.

statusCode(200): Verifica se o servidor retornou o código de status 200, indicando que a atualização foi bem-sucedida.

body("name", equalTo("Jane Doe")): Valida que o nome retornado no corpo da resposta é "Jane Doe".

Requisições HTTP (GET, POST, PUT, DELETE)

Requisição DELETE

A requisição DELETE é usada para remover recursos do servidor. Quando você faz uma requisição DELETE, está solicitando ao servidor que elimine o recurso especificado.

Estrutura de uma Requisição DELETE

Aqui está um exemplo de como fazer uma requisição DELETE usando Rest Assured:

```
Desvendando Rest Assured - Requisição DELETE

import io.restassured.RestAssured;
import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;

public class ApiTest {
    @Test
    public void testDelete() {
        RestAssured.baseURI = "https://api.exemplo.com";

        given()
            .when()
            .delete("/users/1")
            .then()
            .statusCode(204);
    }
}
```



Requisições HTTP (GET, POST, PUT, DELETE)

No exemplo anterior:

RestAssured.baseURI: Define a URL base para as requisições.

given(): Configura a requisição, onde você pode adicionar headers, se necessário.

when(): Inicia a execução da requisição.

delete("/users/1"): Especifica que estamos fazendo uma requisição DELETE para o endpoint /users/1.

then(): Permite fazer validações na resposta.

statusCode(204): Verifica se o servidor retornou o código de status 204."

03

Validação de Respostas

Validação de Respostas com Rest Assured

A validação de respostas é um aspecto crucial dos testes de integração com Rest Assured, garantindo que as respostas das APIs estejam corretas e atendam às expectativas. Vamos explorar como validar o código de status, os headers e o corpo da resposta.

Validação do Código de Status

Validar o código de status HTTP é essencial para garantir que a API está retornando o status correto, indicando o resultado esperado da operação (por exemplo, 200 OK, 201 Created, 204 No Content, etc.).

```
Desvendando Rest Assured - Validação do Código de Status

import io.restassured.RestAssured;
import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;

public class StatusCodeValidationTest {
    @Test
    public void testStatusCode() {
        RestAssured.baseURI = "https://api.exemplo.com";

        given()
            .when()
            .get("/users/1")
            .then()
            .statusCode(200); // Verifica se o código de status é 200 (OK)
    }
}
```


Validação de Respostas com Rest Assured

Validação dos Headers

Os headers da resposta contêm informações adicionais importantes, como o tipo de conteúdo (Content-Type), codificação (Content-Encoding), entre outros. Validar esses headers ajuda a garantir que a resposta está sendo entregue no formato esperado.

```
Desvendando Rest Assured - Validação dos Headers

import io.restassured.RestAssured;
import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;
import static org.hamcrest.Matchers.equalTo;

public class HeaderValidationTest {
    @Test
    public void testHeaders() {
        RestAssured.baseURI = "https://api.exemplo.com";

        given()
            .when()
            .get("/users/1")
            .then()
            .header("Content-Type", equalTo("application/json")) // Verifica se
// o header Content-Type é JSON
            .header("Content-Encoding", equalTo("gzip")); // Verifica se o
// header Content-Encoding é gzip
    }
}
```

Validação de Respostas com Rest Assured

Validação do Corpo da Resposta

A validação do corpo da resposta é crucial para garantir que os dados retornados pela API estejam corretos e completos. Utilizando JsonPath ou XmlPath, é possível extrair e validar campos específicos dentro do JSON ou XML retornado.

```
Desvendando Rest Assured - Validação Corpo da Resposta

import io.restassured.RestAssured;
import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;
import static org.hamcrest.Matchers.equalTo;

public class BodyValidationTest {
    @Test
    public void testBody() {
        RestAssured.baseURI = "https://api.exemplo.com";

        given()
            .when()
            .get("/users/1")
            .then()
            .body("name", equalTo("John Doe")) // Verifica se o campo 'name' é
            'John Doe'
            .body("email", equalTo("johndoe@example.com")); // Verifica se o
            campo 'email' é 'johndoe@example.com'
    }
}
```

04

Validação de Esquemas JSON e XML

Validação de Esquemas JSON e XML

A validação de esquemas é uma prática essencial para garantir que as respostas das APIs RESTful estejam conforme os formatos esperados e que os dados sigam uma estrutura predefinida. No Rest Assured, a validação de esquemas pode ser aplicada tanto para JSON quanto para XML, assegurando que a integridade dos dados seja mantida.

Validação de Esquema JSON

A validação de esquema JSON garante que a resposta JSON de uma API esteja de acordo com um esquema especificado, que define a estrutura dos dados, tipos de dados e restrições.

```
Desvendando Rest Assured - Validação JSON

import io.restassured.RestAssured;
import org.junit.jupiter.api.Test;

import static
io.restassured.module.json.JsonSchemaValidator.matchesJsonSchemaInClasspath;
import static io.restassured.RestAssured.given;

public class JsonSchemaValidationTest {
    @Test
    public void testJsonSchemaValidation() {
        RestAssured.baseURI = "https://api.exemplo.com";

        given()
            .when()
            .get("/users/1")
            .then()
            .statusCode(200)
            .body(matchesJsonSchemaInClasspath("user-schema.json"));
    }
}
```


Validação de Respostas com Rest Assured

Validação de Esquema XML

A validação de esquema XML garante que a resposta XML de uma API esteja de acordo com um esquema XSD (XML Schema Definition) especificado.

```
Desvendando Rest Assured - Validação XML

import io.restassured.RestAssured;
import org.junit.jupiter.api.Test;

import static io.restassured.module.xml.XmlSchemaValidator.matchesXsdInClasspath;
import static io.restassured.RestAssured.given;

public class XmlSchemaValidationTest {
    @Test
    public void testXmlSchemaValidation() {
        RestAssured.baseURI = "https://api.exemplo.com";

        given()
            .when()
            .get("/users/1")
            .then()
            .statusCode(200)
            .body(matchesXsdInClasspath("user-schema.xsd"));
    }
}
```

05

Integração com Frameworks de Teste (JUnit, TestNG)

Integração com Frameworks de Teste (JUnit, TestNG)

Passo 1: Configuração do Projeto

Primeiro, certifique-se de que seu projeto está configurado com as dependências necessárias no Maven.

```
Desvendando Rest Assured - JUNIT

<dependencies>
  <dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>5.0.0</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.8.2</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest</artifactId>
    <version>2.2</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.8.2</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Integração com Frameworks de Teste (JUnit, TestNG)

Passo 2: Estrutura do Teste com JUnit

Vamos criar uma classe de teste que verifica se a API está retornando os dados corretos para um usuário específico.

```
Desvendando Rest Assured - JUNIT

import io.restassured.RestAssured;
import io.restassured.response.Response;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import static io.restassured.RestAssured.given;
import static org.hamcrest.Matchers.equalTo;

public class RestAssuredJUnitTest {

    @BeforeAll
    public static void setup() {
        // Define a URL base para as requisições
        RestAssured.baseURI = "https://api.exemplo.com";
    }

    @Test
    public void testGetUser() {
        // Realiza uma requisição GET para o endpoint /users/1
        Response response = given()
            .when()
            .get("/users/1")
            .then()
            .statusCode(200) // Verifica se o código de status é 200 (OK)
            .body("id", equalTo(1)) // Verifica se o campo 'id' é 1
            .body("name", equalTo("John Doe")) // Verifica se o campo 'name' é
            'John Doe'
            .body("email", equalTo("johndoe@example.com")) // Verifica se o
            campo 'email' é 'johndoe@example.com'
            .extract()
            .response();

        // Exibe a resposta no console (opcional)
        System.out.println("Response: " + response.asString());
    }
}
```


Integração com Frameworks de Teste (JUnit, TestNG)

Passo 2: Estrutura do Teste com JUnit

Sobre código anterior:

@BeforeAll: Anotação do JUnit que indica que o método setup deve ser executado uma vez antes de todos os testes. Usamos este método para definir a URL base da API.

@Test: Anotação do JUnit que indica que o método testGetUser é um teste.

given(): Configura a requisição.**when():** Inicia a execução da requisição.

get("/users/1"): Especifica que estamos fazendo uma requisição GET para o endpoint /users/1.

then(): Permite fazer validações na resposta.**statusCode(200):** Verifica se o código de status da resposta é 200 (OK).

body("id", equalTo(1)): Valida que o campo id na resposta é igual a 1.

body("name", equalTo("John Doe")): Valida que o campo name na resposta é igual a "John Doe".

body("email", equalTo("johndoe@example.com")): Valida que o campo email na resposta é igual a "johndoe@example.com".**extract():**

response(): Extrai a resposta da requisição para uso posterior (opcional).

Integração com Frameworks de Teste (JUnit, TestNG)

TestNG

TestNG é outro popular framework de teste para Java, conhecido por sua flexibilidade e funcionalidades avançadas, como a capacidade de definir métodos de configuração e a execução paralela de testes.

No exemplo de teste abaixo podemos ver que é muito similar ao JUnit, onde **@Test** é uma anotação do TestNG que indica que o método `testGetUser` é um teste. A estrutura do teste é similar ao exemplo do JUnit, mostrando a compatibilidade do Rest Assured com ambos os frameworks.

```
Desvendando Rest Assured - TestNG

import io.restassured.RestAssured;
import io.restassured.response.Response;
import org.testng.annotations.Test;
import static io.restassured.RestAssured.given;
import static org.hamcrest.Matchers.equalTo;

public class RestAssuredTestNGTest {

    @Test
    public void testGetUser() {
        RestAssured.baseURI = "https://api.exemplo.com";

        given()
            .when()
            .get("/users/1")
            .then()
            .statusCode(200)
            .body("name", equalTo("John Doe"));
    }
}
```

06

Projeto de teste de API REST básico



Projeto de teste de API REST básico

Vamos criar um exemplo prático de um projeto de teste de API REST básico usando Rest Assured para testes de integração em Java na IDE Eclipse. Suponha que estamos testando uma API de gerenciamento de usuários. O objetivo será realizar operações básicas como criar um novo usuário, buscar um usuário existente, atualizar informações do usuário e excluir um usuário.

Rodando Testes no Eclipse

Passo 1: Instalar Eclipse e Configurar o Projeto

- Baixe e instale o Eclipse IDE for Java Developers.
- Abra o Eclipse e crie um novo projeto Maven.
- Adicione as dependências do Rest Assured e JUnit no arquivo pom.xml.

Projeto de teste de API REST básico

Rodando Testes no Eclipse

Passo 2: Configurar as dependências Maven

- Primeiro, adicione as seguintes dependências ao arquivo pom.xml do seu projeto:

```
Desvendando Rest Assured - Projeto Básico Eclipse

<dependencies>
  <dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>5.0.0</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.8.2</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest</artifactId>
    <version>2.2</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.8.2</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Projeto de teste de API REST básico

Rodando Testes no Eclipse

Passo 3: Estrutura do Projeto

- Crie uma classe de teste UserApiTest que contém os métodos CRUD(create,update,delete e read) de teste para as operações da API.

```
Desvendando Rest Assured - Projeto Básico Eclipse - UserApiTest

import io.restassured.RestAssured;
import io.restassured.response.Response;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Order;
import org.junit.jupiter.api.TestMethodOrder;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.MethodOrderer.OrderAnnotation;

import static io.restassured.RestAssured.given;
import static org.hamcrest.Matchers.equalTo;

@TestMethodOrder(OrderAnnotation.class)
public class UserApiTest {

    private static int userId;

    @BeforeAll
    public static void setup() {
        RestAssured.baseURI = "https://api.exemplo.com";
    }

    @Test
    @Order(1)
    public void testCreateUser() {
        Response response = given()
            .header("Content-Type", "application/json")
            .body("{ \"name\": \"John Doe\", \"email\": \"johndoe@example.com\" }")
            .when()
            .post("/users")
            .then()
            .statusCode(201) // Verifica se o código de status é 201 (Created)
            .body("name", equalTo("John Doe"))
            .body("email", equalTo("johndoe@example.com"))
            .extract()
            .response();

        userId = response.jsonPath().getInt("id");
        System.out.println("Created User ID: " + userId);
    }

    @Test
    @Order(2)
    public void testGetUser() {
        given()
            .when()
            .get("/users/" + userId)
            .then()
            .statusCode(200) // Verifica se o código de status é 200 (OK)
            .body("id", equalTo(userId))
            .body("name", equalTo("John Doe"))
            .body("email", equalTo("johndoe@example.com"));
    }

    @Test
    @Order(3)
    public void testUpdateUser() {
        given()
            .header("Content-Type", "application/json")
            .body("{ \"name\": \"Jane Doe\", \"email\": \"janedoe@example.com\" }")
            .when()
            .put("/users/" + userId)
            .then()
            .statusCode(200) // Verifica se o código de status é 200 (OK)
            .body("id", equalTo(userId))
            .body("name", equalTo("Jane Doe"))
            .body("email", equalTo("janedoe@example.com"));
    }

    @Test
    @Order(4)
    public void testDeleteUser() {
        given()
            .when()
            .delete("/users/" + userId)
            .then()
            .statusCode(204); // Verifica se o código de status é 204 (No Content)
    }
}
```




Projeto de teste de API REST básico

Rodando Testes no Eclipse

Explicação do Código:

@BeforeAll: Configura a URL base da API para todas as requisições.

@TestMethodOrder(OrderAnnotation.class): Define a ordem de execução dos testes.

@Order(x): Especifica a ordem dos testes, onde x é o número da ordem.

testCreateUser(): Testa a criação de um novo usuário e verifica se o código de status é 201 (Created).

testGetUser(): Testa a recuperação de um usuário existente e verifica se o código de status é 200 (OK) e os dados do usuário estão corretos.

testUpdateUser(): Testa a atualização dos dados de um usuário existente e verifica se o código de status é 200 (OK) e os dados atualizados estão corretos.

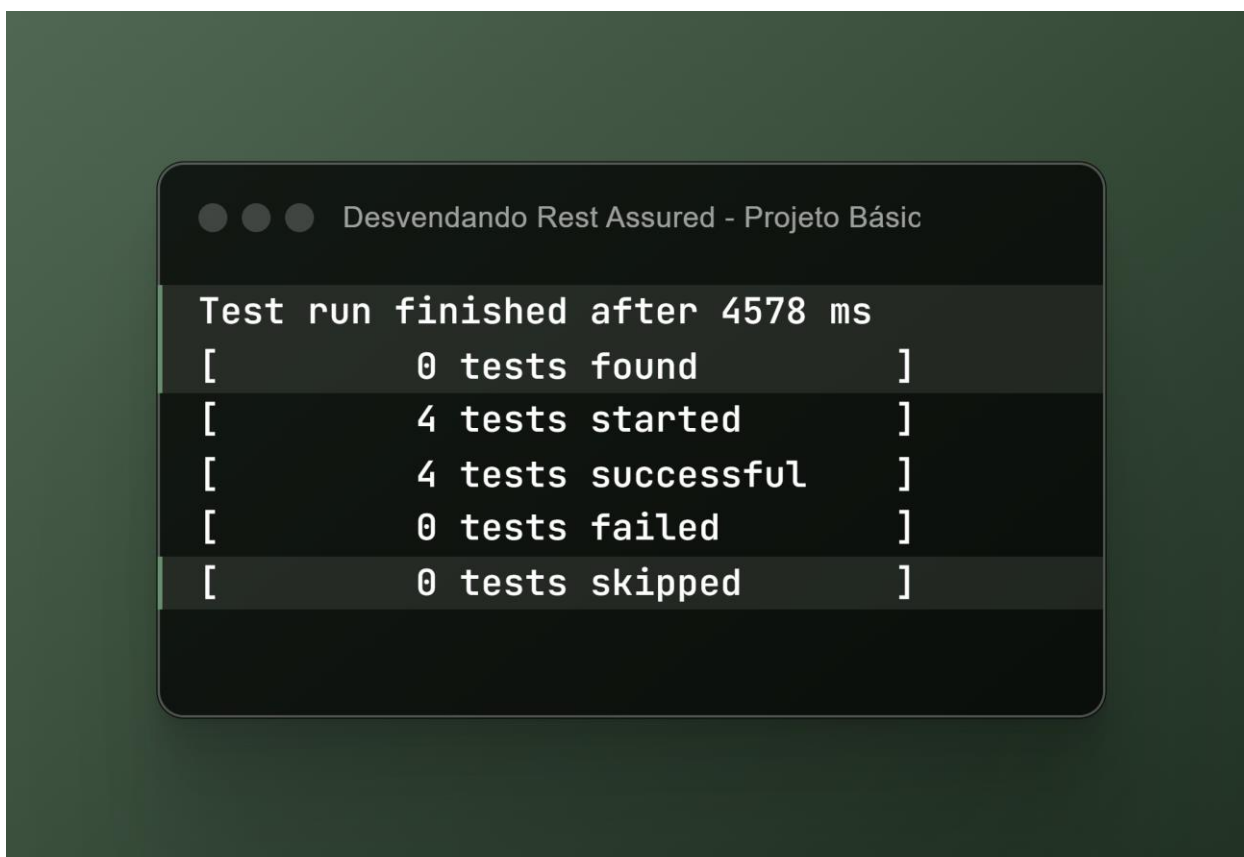
.testDeleteUser(): Testa a exclusão de um usuário existente e verifica se o código de status é 204 (No Content).

Projeto de teste de API REST básico

Rodando Testes no Eclipse

Passo 4: Executar Testes

- Navegue até o arquivo da classe de teste (UserApiTest.java) no Package Explorer.
- Clique com o botão direito no arquivo e selecione Run As > JUnit Test.
- O Eclipse irá compilar e executar seus testes, exibindo os resultados na aba "JUnit"



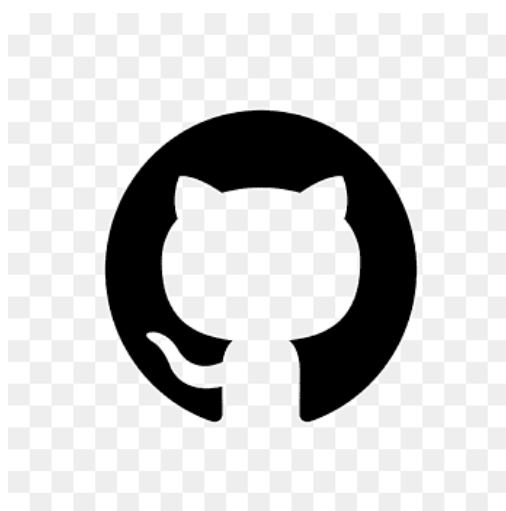
Agradecimentos

Obrigada por ler até aqui

Este e-book foi gerado por IA, editado e diagramado por humano.

Foi criado para fins didáticos de construção e pode conter informações não revisadas pelo uso de IA.

Espero que gostem do conteúdo e seja colabore na sua jornada dos testes de integração!



<https://github.com/prittypcs/ebook-rest-assured>

