

VEM SER



Testes de API Rest com REST Assured



DBC



Aula 2 - Rest Assured (Parte 1)

Sumário

1. O que é Rest Assured
2. Primeiros Passos
3. Fazendo Requisições com Rest Assured
4. Validando Respostas com Rest Assured
5. Task RA02
6. Referências





O que é
Rest Assured




O que é Rest Assured

- Rest Assured é uma tecnologia de código aberto, baseada em Java, que nos permite testar e validar serviços REST de um jeito mais prático.
- Basicamente ele nos provê uma maneira de criar requisições HTTP, como se fossemos um cliente acessando uma API, e validar as respostas HTTP recebidas do servidor.
- Integra-se perfeitamente com estruturas de teste baseadas em Java, como JUnit, TestNG e Selenium Webdriver.

O que é REST Assured

- Suporta qualquer método HTTP, e possui suporte explícito para **GET, POST, PUT, DELETE, OPTIONS e HEAD**.
- Possibilidade do uso dos Matchers Hamcrest para realizar as validações.
- Suporta JsonPath e XmlPath que ajudam na análise da resposta em formato JSON e XML.
- Utiliza a sintaxe **given, when e then**, que é o mesmo formato da escrita BDD/Gerkin. Esse padrão é utilizado para dividir o código em seções para demonstrar seu comportamento.

Padrão REST Assured

- **given()**  **[Dado]** que tenho algo. Pré condições, como por exemplo: headers, body, portas, token de autenticação, etc.
- **when()**  **[Quando]** realizo uma ação. Método a ser executado. Trecho que identifica a proposta do cenário.
- **then()**  **[Então]** obtenho um resultado. Extrair respostas e efetuar assertivas.

Primeiros Passos

Primeiros passos: Ferramentas

- **Java JDK 8 ou superior.**
- **IntelliJ:**
 - Criar projeto com gerenciador de pacotes **Maven**.
- **Dependências:**
 - **Rest Assured >> 4.4.0**
<https://mvnrepository.com/artifact/io.rest-assured/rest-assured/4.4.0>
 - **JUnit Jupiter (Aggregator) >> 5.9.1**
<https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter/5.9.1>

Primeiros passos: Convenções JUnit

- **Classe de teste**

- O que é?
 - Qualquer classe que contém pelo menos um método de teste é considerada uma classe de teste.
- Ao nomear uma classe de teste, deve-se colocar a palavra “Test” **no final**. Por exemplo: **ProdutoTest, PessoaTest, BookStoreTest, etc.**

Primeiros passos: Convenções JUnit

- **Método de teste**
 - **Estrutura:**
 - **@Test** >> org.junit.jupiter.api
 - **public void testNomeMetodo(params*) {}**
 - Ao nomear um método, deve-se colocar a palavra “test” **no início**. Por exemplo: **testValidarTokenUsuarioAdm()**.
 - **@DisplayName** >> org.junit.jupiter.api
 - Utilizado para dar mais detalhes e informações sobre o teste.

Primeiros passos: Importes estáticos

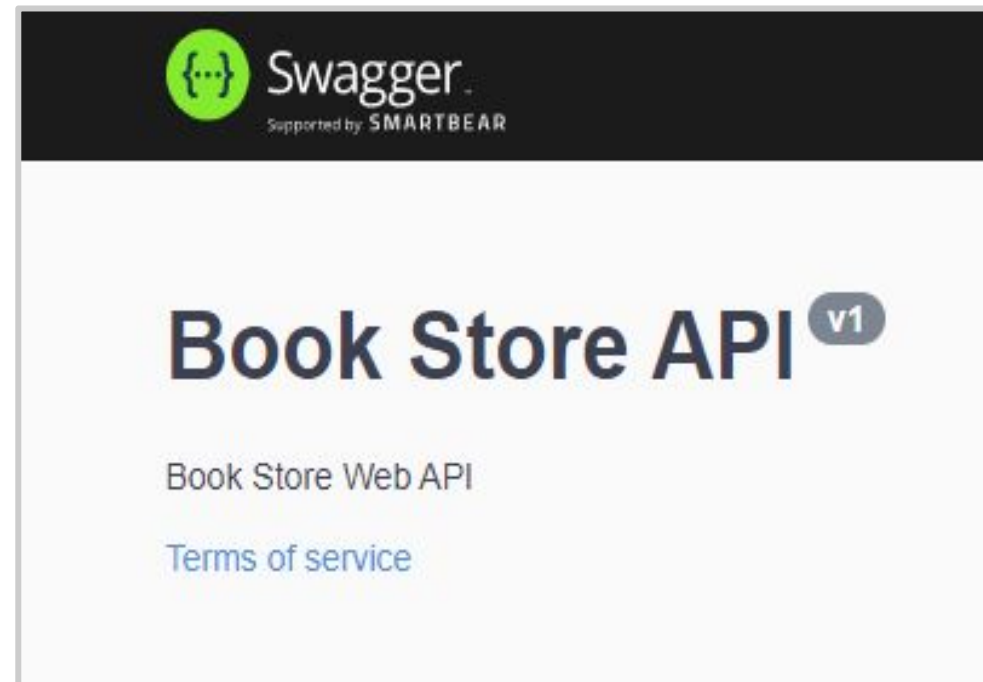
- Para uso mais eficaz do Rest Assured, a documentação recomenda fazer importes estáticos dos seguintes métodos:
 - **io.restassured.RestAssured.***
io.restassured.matcher.RestAssuredMatchers.*
org.hamcrest.Matchers.*



Fazendo
Requisições
com Rest
Assured

Fazendo Requisições com Rest Assured

- Para essa aula, vamos utilizar a Book Store API, disponível em: <https://bookstore.toolsqa.com/swagger/#/>
- Para acompanhar: <https://demoqa.com/books>





GET

Requisição GET

- **Para lembrar:** O método **GET** permite buscar informações do servidor.
- Para fazer uma requisição GET:
 - given()
 - .when()
 - **.get(url)**
 - .then();

Query Parameters

- Nem sempre teremos como objetivo buscar todos os registros de um recurso, mas apenas **um ou alguns**. É nesse contexto a importância dos **query parameters**, dado que é possível **classificar/filtrar** recursos.
- Podemos definir parâmetros de consulta **diretamente na URL**:
 - **`..when().get("/Book?ISBN=9781449325862");`**
- Ou **utilizando o método do REST Assured**:
 - `given()`
 - `.queryParams("queryParamsName", "value")`**
 - `.when()...`**

Intervalo

- 15 minutos
- Voltamos:



POST

Requisição POST


- **Para lembrar:** O método **POST** permite criar um novo recurso dentro do sistema.
- Os dados são enviados por meio do corpo da solicitação, também conhecido por **Body** ou, ainda, **Payload**.
- Para fazer uma requisição POST:
 - `given()`
 `.when()`
 `.post(url)`
 `.then();`


Autenticação e Autorização

- Autenticação e autorização são dois conceitos muito importantes no contexto de API Rest.
- No dia a dia profissional, na maioria das vezes, acessamos APIs Rest que são protegidas, ou seja, exigem fornecimento de **identificação** para o acesso a determinados recursos.
- Principais formas de identificação:
 - **Usuário e senha → Basic;**
 - **Tokens → Bearer;**

Autenticação e Autorização

- **Autenticação:** Processo para provar que você é a pessoa que pretende ser. Apresentar credenciais.





- **Autorização:** Processo de dar acesso a alguém. Diz respeito aos privilégios que são concedidos a determinado usuário ao utilizar uma aplicação.

Contas a Pagar	<input checked="" type="checkbox"/>
Contas a Receber	<input checked="" type="checkbox"/>
Controle de Estoque	<input type="checkbox"/>

Basic Authentication

- **Preemptive Basic Authentication**

- Esse esquema enviará a credencial de autenticação no cabeçalho da requisição, independentemente de ser solicitado pelo servidor.
- given()
 .auth().preemptive().basic("username", "password")
 .when()..;

 **Para saber mais:**

<https://www.toolsqa.com/rest-assured/basic-auth/>



PUT

Requisição PUT

- **Para relembrar:** O método **PUT** permite atualizar (substituir) uma representação do recurso de destino com o payload da requisição.
- Para fazer uma requisição PUT:
 - `given()`
 `.when()`
 `.put(url)`
 `.then();`

Path Parameters

- Utilizado para identificar recurso(s) específico(s). Facilita a leitura do path request, além de permitir que o mesmo seja facilmente reutilizável em muitos testes com diferentes valores de parâmetro.
- Formas de utilização:
 - **...put(url+"/{isbn}", "9781449325862");**
 - given()
 .pathParam("isbn", "9781449325862")
 .when()
 .put(url+"/{isbn}")...;

Autenticação com Token

- No Rest Assured:
 - given()
 .auth().oauth2(token)
 .when()..;
 - given()
 .header("Authorization", "Bearer " + token)
 .when()..;



DELETE

Requisição DELETE

- **Para relembrar:** O método **DELETE** exclui um recurso do servidor.
- Para fazer uma requisição DELETE:
 - given()
 .when()
 .delete(url)
 .then();

Intervalo

- 15 minutos
- Voltamos:



Validando
Respostas
com Rest
Assured

Validando Respostas

- Como visto anteriormente, uma Resposta (Response) HTTP contém:
 - Status Code;
 - Headers;
 - Body.
- No Rest Assured:
 - ...then()
;

Validando Respostas: Status Code

- Cada Resposta HTTP possui um status code. Esse valor nos informa se a requisição foi bem-sucedida ou não.
- Dessa forma, podemos:
 - Validar o status code de **sucesso**;
 - Validar o status code de **erro**:
 - Servidor inativo;
 - Requisição incorreta;
 - Recurso inexistente;
 - Erro no servidor.

Validando Respostas: Status Code

- Método Rest Assured para validar Status Code:
 - ...then()
 .statusCode(200)
 ;

Validando Respostas: Cabeçalhos

- Métodos Rest Assured:

- `..then()`

```
.header("headerName", "headerValue")  
.headers("headerName1", "headerValue1",  
"headerName2", "headerValue2")  
;
```


Validando Respostas: Corpo

- Método Rest Assured
 - ...then()
 .body()
 ;
- JsonPath e Hamcrest
 - **JsonPath:** Utilizado para consultar partes específicas do JSON de resposta.
 - **Hamcrest:** É um framework para escrever objetos matcher permitindo que regras de “match” sejam definidas declarativamente. Comumente utilizado com JUnit e testNG para fazer asserções.

Validando Respostas: Corpo



Para saber mais:

Hamcrest:

<http://hamcrest.org/JavaHamcrest/javadoc/1.3/org/hamcrest/Matchers.html>

Para refletir



Os testes são independentes entre si?



Task RA-02

TASK RA-02

Passo 1: Utilizando o que foi aprendido no módulo de Teoria (BDD), escreva casos de testes (positivos e negativos) para a BookStore API **(EM TODOS OS ENDPOINTS)**. **Entrega: 08/03 as 11:00.**

Passo 2: Desenvolver scripts de testes, com Restassured, para todos os endpoints da BookStore API, dando continuidade ao que foi desenvolvido em aula.

Desafio: Buscar a solução para que os testes sejam independentes entre si.

Referências

- <https://rest-assured.io/>
- <https://github.com/rest-assured/rest-assured/wiki/GettingStarted>
- <https://www.toolsqa.com/rest-assured/rest-assured-library/>
- <https://www.baeldung.com/rest-assured-tutorial>
- <https://blog.onedaytesting.com.br/testes-de-integracao-com-rest-assured/>
- <https://www.toolsqa.com/rest-assured/query-parameters-in-rest-assured/>
- <https://www.toolsqa.com/rest-assured/authentication-and-authorization-in-rest-webservices/>
- <https://www.treinaweb.com.br/blog/autenticacao-x-autorizacao>
- <https://www.toolsqa.com/rest-assured/validate-response-status-using-rest-assured/>



Let's *Tech Up Together*