

# VEM SER

**Módulo 4 - Testes de API REST**  
**Aula 4 - REST Assured (Parte 1)**



DBC

# Aula 4: REST Assured (Parte 1)





# Aula 3 - Conteúdo Programático

3.1 Introdução ao REST Assured

3.2 Requisições básicas com  
REST Assured

## 3.1 Introdução ao REST Assured



## 3.1.1 O que é REST Assured

REST Assured é uma tecnologia de código aberto (*open source*), baseada em **Java**, que nos permite **testar e validar serviços REST** de um jeito mais prático.

Integra-se perfeitamente com estruturas de teste baseadas em Java, como **JUnit**, **TestNG** e **Selenium WebDriver**.





## 3.1.2 Palavras Chave no REST Assured

O REST Assured utiliza uma sintaxe simples e intuitiva para escrever testes, onde **o código é dividido em seções para demonstrar seu comportamento.**

- **given() → [Dado]** que tenho algo. Pré condições ou informações iniciais.  
Ex: cabeçalhos, corpo, parâmetros, autenticação, etc.
- **when() → [Quando]** realizo uma ação. Método a ser utilizado e endpoint que deseja acessar. Identifica a proposta do cenário.
- **then() → [Então]** obtenho um resultado. Extrair respostas e definir assertivas.

## 3.2 Requisições Básicas





## 3.2 Requisições Básicas

POST, GET, PUT e DELETE



## 3.1.3 - 3 Passos Para Configurar Setup

**1. Instale o Java JDK 17 e configure nas variáveis do sistema.**

**2. Instale a IDE (IntelliJ) e crie um Projeto Maven:**

- a. O que é Maven?
  - i. É basicamente uma ferramenta para gerenciar e construir projetos baseados em Java;
  - ii. Criação automática de esqueleto de projeto;
  - iii. Facilidade de adição de dependências de projeto;
  - iv. Integração perfeita com CI/CD



## 3.1.3 - 3 Passos Para Configurar Setup

### 3. Configurar Rest assured no Projeto:

a. Adicionar as dependências no arquivo **pom.xml**:

i. **Rest Assured >> 5.4.0**

1. <https://mvnrepository.com/artifact/io.rest-assured/rest-assured/5.4.0>

ii. **JUnit Jupiter (Aggregator) >> 5.10.0**

1. <https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter/5.10.0>

## 3.1.4 Validando Setup

```
public class OlaMundoTest {  
    @Test  
    public void testBuscarUsuarioPorIDComSucesso() {  
        baseURI = "https://regres.in/api";  
        given() RequestSpecification  
            .log().all()  
            .when()  
            .get(s: "/users/1") Response  
        .then() ValidatableResponse  
            .log().all()  
            .assertThat()  
                .statusCode(i: 200)  
        ;  
    }  
}
```

## 3.1.5 Convenções JUnit

### Classe de Teste

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

### Método de Teste

1. Estrutura:
  - a. **@Test** >> org.junit.jupiter.api
  - b. **public void testNomeMetodo(params\*) {}**
  - c. Ao nomear um método, deve-se colocar a palavra “test” **no início**. Por exemplo: **testGetUsuarioPorIDComSucesso()**.

## 3.3 Aserções e Validações





## 3.3.1 Status Code

### Podemos validar:

- Validar o status code de **sucesso**;
- Validar o status code de **erro**:
  - Servidor inativo;
  - Requisição incorreta;
  - Recurso inexistente;
  - Erro no servidor.

```
.then() ValidatableResponse
    .assertThat()
        .statusCode(200)
;
```

## 3.3.2 Headers

Cada resposta obtida do servidor pode conter **zero ou mais headers**.

```
.assertThat()  
  .statusCode(200)  
  .header("Content-Type", "application/json") // "headerName", "headerValue"  
  .headers("Content-Type", "application/json", ...objects: "Connection", "keep-alive")
```

## 3.3.3 Body - Hamcrest

### O que é Hamcrest?

- É uma biblioteca de asserção que permite escrever asserções legíveis e expressivas em seus testes;
- Fornece uma ampla variedade de correspondentes (matchers) que podem ser usados para verificar se os resultados atendem às expectativas;



## 3.3.3 Body - Hamcrest

```
{  
  "nome": "João da Silva",  
  "idade": 32,  
  "endereço": {  
    "rua": "Rua A",  
    "cidade": "Recife"  
  },  
  "telefones": [  
    "123456789",  
    "987654321"  
  ]  
}
```

Percorrendo o JSON para extrair informações específicas

### Exemplos:

"nome"

"endereço.cidade"

"telefones[0]"

## 3.3.3 Body - Hamcrest

### Exemplos de métodos

- *lessThan()*, *greaterThan()*, *equalTo()*, *containsString()*, *notNullValue()*;

```
.time(lessThan( value: 600L)) // Medir tempo de resposta - Milliseconds
.body( s: "totalElements", greaterThan( value: 8900))
.body( s: "content[0].nome", equalTo( operand: "Miss Curt Fahey Wisozk"))
.body( s: "content[0].email", containsString( substring: "roberta.armstrong@hotmail.com"))
.body( s: "content[0].idPessoa", notNullValue())
```

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

## 3.4 Teste de “Contrato”





## 3.4.1 Teste de “Contrato”

Na verdade, o que vamos testar é o **schema-json**.

Verifica se a estrutura da resposta de uma API está de acordo com um schema predefinido.

### Por quê é importante?

- **Confiança nos dados:** Assegura que os dados retornados estão no formato esperado.
- **Prevenção de erros:** Ajuda a identificar problemas precocemente no ciclo de desenvolvimento.

## 3.4.1 Teste de “Contrato”

### Passos

#### 1. Dependência:

- a. <https://mvnrepository.com/artifact/io.rest-assured/json-schema-validator/5.4.0>

#### 2. Defina o Schema: Crie um schema JSON que representa a estrutura esperada da resposta.

- b. <https://www.liquid-technologies.com/online-json-to-schema-converter>

#### 3. Escreva o Teste

# 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://medium.com/assertqualityassurance/testando-seu-contrato-com-o-rest-assured-f0e974fb9bcb>



Let's *Tech Up Together*