

2. TESTES AUTOMATIZADOS DE API

Objetivo: Avaliar a capacidade de automatizar testes de API e interpretar os resultados.

Tarefas Práticas:

- Tarefa 1 : Usando ferramentas como Rest Assured, Cypress ou Playwright, crie um teste para validar endpoints de uma API de exemplo. O teste deve incluir verificações de resposta (status codes, headers e corpo).

- o Avaliação: Uso correto de ferramentas, clareza dos testes e cobertura de cenários importantes (testes positivos, negativos, etc.).

- Tarefa 2: Automatizar testes para múltiplos endpoints da API, validando diferentes métodos HTTP (GET, POST, PUT, DELETE), e realizar validações de status codes, headers e corpo de resposta para cada um dos métodos.

- o Relatório: Gerar e apresentar um relatório detalhado com os resultados dos testes.

INFORMAÇÕES GERAIS

Projeto: Teste API com TestGetPosts, TestPostCreate, TestPutUpdate e TestDeletePost, 400 (BadRequest)

401 (Unauthorized), 403 (Forbidden), 404 (NotFound) e 500 (Server Error)

Ferramenta: Rest Assured 5.4.0, JUnit 5.10.2, Allure 2.13.9, Maven 3.9 e Java 11

Base URL: <https://jsonplaceholder.typicode.com>

Base URL simulando erros: <https://httpstat.us>

Sistema Operacional: Windows 10

Data de Execução: 17/05/2025

Responsável: [Bruno Nascimento]

EVIDÊNCIAS

Estrutura dos testes:

TestGetPosts: valida GET /posts

TestPostCreate: valida criação via POST

TestPutUpdate: valida atualização via PUT

TestDeletePost: valida exclusão via DELETE

Teste	Método HTTP	Endpoint	Descrição	Resultado esperado
TestGetPosts	GET	/posts	Valida retorno status 200 e lista não vazia	Status 200, JSON com > 0 itens
TestPostCreate	POST	/posts	Cria novo post com título, corpo e userId	Status 201, retorno com dados criados
TestPutUpdate	PUT	/posts/1	Atualiza título e corpo do post 1	Status 200, título atualizado
TestDeletePost	DELETE	/posts/1	Deleta post 1	Status 200 ou 204

Executando o comando mvn clean test no bash:

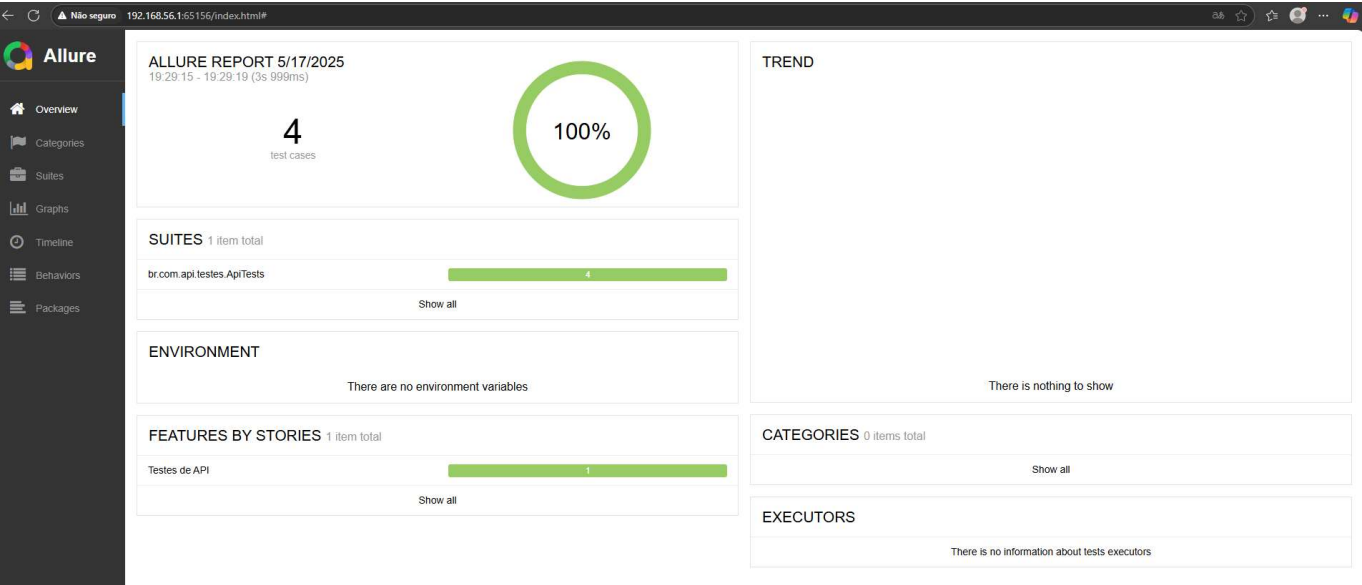
Com esse comando geramos os tests results:

```
MINGW64/c/testeapi/teste-api
hell@DESKTOP-FSRAT80 MINGW64 /c/testeapi/teste-api
$ mvn clean test
[INFO] Scanning for projects...
[INFO]
[INFO] -----< br.com.api.testes:teste-api >-----
[INFO] Building teste-api 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO]
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- clean:3.2.0:clean (default-clean) @ teste-api ---
[INFO] Deleting C:\testeapi\teste-api\target
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ teste-api ---
[INFO] skip non existing resourceDirectory C:\testeapi\teste-api\src\main\resources
[INFO]
[INFO] --- compiler:3.13.0:compile (default-compile) @ teste-api ---
[INFO] Recompiling the module because of changed source code.
[INFO] Compiling 1 source file with javac [debug target 11] to target\classes
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ teste-api ---
[INFO] skip non existing resourceDirectory C:\testeapi\teste-api\src\test\resources
[INFO]
[INFO] --- compiler:3.13.0:testCompile (default-testCompile) @ teste-api ---
[INFO] Recompiling the module because of changed dependency.
[INFO] Compiling 1 source file with javac [debug target 11] to target\test-classes
[INFO]
[INFO] --- surefire:3.0.0-M5:test (default-test) @ teste-api ---
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[INFO]
[INFO] T E S T S
[INFO]
[INFO] SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
[INFO] SLF4J: Defaulting to no-operation (NOP) logger implementation
[INFO] SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
[INFO] Running br.com.api.testes.ApiTests
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 4.455 s - in br.com.api.testes.ApiTests
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 9.173 s
[INFO] Finished at: 2025-05-17T19:29:19-03:00
[INFO]
hell@DESKTOP-FSRAT80 MINGW64 /c/testeapi/teste-api
$ allure serve target/allure-results
```

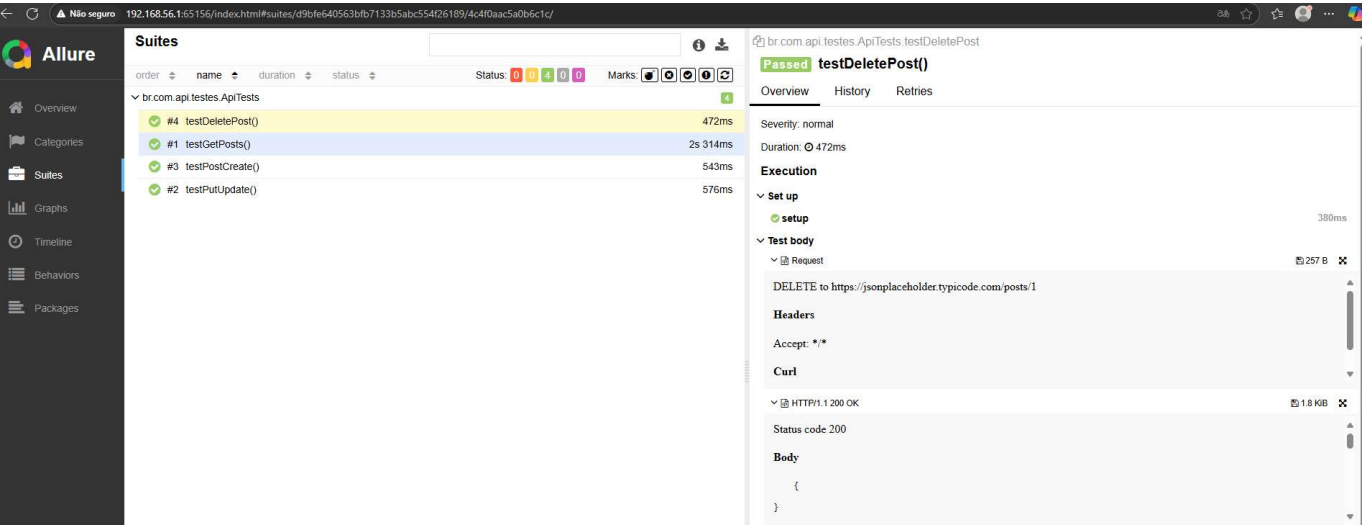
Executando o comando do report allure após gerar o teste corretamente:
allure serve target/allure-results

```
Dell@DESKTOP-FSRAT80 MINGW64 /c/testeapi/teste-api
$ allure serve target/allure-results
Generating report to temp directory...
Report successfully generated to C:\Users\Dell\AppData\Local\Temp\11739275443976481236\allure-report
Starting web server...
2025-05-17 19:33:57.924:INFO::main: Logging initialized @3008ms to org.eclipse.jetty.util.log.StdErrLog
Server started at <http://192.168.56.1:65156/>. Press <Ctrl+C> to exit
```

Após executar o comando acima, automaticamente abre o Allure Report:



testDeletPost()



TestGetPost()

Allure

Overview

Categories

Suites

Graphs

Timeline

Behaviors

Packages

192.168.56.1:65156/index.html#suites/d9bfe640563bfb7133b5abc554f26189/8c18c951aae51b6f/

Suites

order

name

duration

status

Status:

Marks:

br.com.api.tests.ApiTests

#4 testDeletePost()

#1 testGetPosts()

#3 testPostCreate()

#2 testPutUpdate()

472ms

2s 314ms

543ms

576ms

br.com.api.tests.ApiTests.testGetPosts

Passed testGetPosts()

OverviewHistoryRetries

Severity: normal

Duration: 2s 314ms

Description

Teste que valida se o endpoint GET /posts/1 retorna o status 200 e dados corretos.

Execution

Set up

setup

380ms

Test body

Request

247 B

GET to https://jsonplaceholder.typicode.com/posts

Headers

Accept: */*

Curl

HTTP/1.1 200 OK

36.5 KB

Status code 200

Body

```
{
  "userId": 1,

```

TestPostCreat()

Allure

Overview

Categories

Suites

Graphs

Timeline

Behaviors

Packages

192.168.56.1:65156/index.html#suites/d9bfe640563bfb7133b5abc554f26189/7b8f291709d2609d/

Suites

order

name

duration

status

Status:

Marks:

br.com.api.tests.ApiTests

#4 testDeletePost()

#1 testGetPosts()

#3 testPostCreate()

#2 testPutUpdate()

472ms

2s 314ms

543ms

576ms

br.com.api.tests.ApiTests.testPostCreate

Passed testPostCreate()

OverviewHistoryRetries

Severity: normal

Duration: 543ms

Execution

Set up

setup

380ms

Test body

Request

651 B

POST to https://jsonplaceholder.typicode.com/posts

Body

```
{
  "title": "foo",
  "body": "bar",
  "userId": 1
}
```

HTTP/1.1 200 OK

2 KB

Status code 201

Body

```
{
  "title": "foo",
  "body": "bar",
  "userId": 1,

```

TestPostUpdate()

Allure

Overview

Categories

Suites

Graphs

Timeline

Behaviors

Packages

192.168.56.1:65156/index.html#suites/d9bfe640563bfb7133b5abc554f26189/3ac44ad72fd34f43/

Suites

order

name

duration

status

Status:

Marks:

br.com.api.tests.ApiTests

#4 testDeletePost()

#1 testGetPosts()

#3 testPostCreate()

#2 testPutUpdate()

472ms

2s 314ms

543ms

576ms

br.com.api.tests.ApiTests.testPutUpdate

Passed testPutUpdate()

OverviewHistoryRetries

Severity: normal

Duration: 576ms

Execution

Set up

setup

380ms

Test body

Request

725 B

PUT to https://jsonplaceholder.typicode.com/posts/1

Body

```
{
  "id": 1,
  "title": "updated",
  "body": "updated body",
}
```

HTTP/1.1 200 OK

19 KB

Status code 200

Body

```
{
  "id": 1,
  "title": "updated",
  "body": "updated body",
}
```

SIMULANDO API COM ERROS

400 (BadRequest)
401 (Unauthorized)
403 (Forbidden)
404 (NotFound)
500 (Server Error)

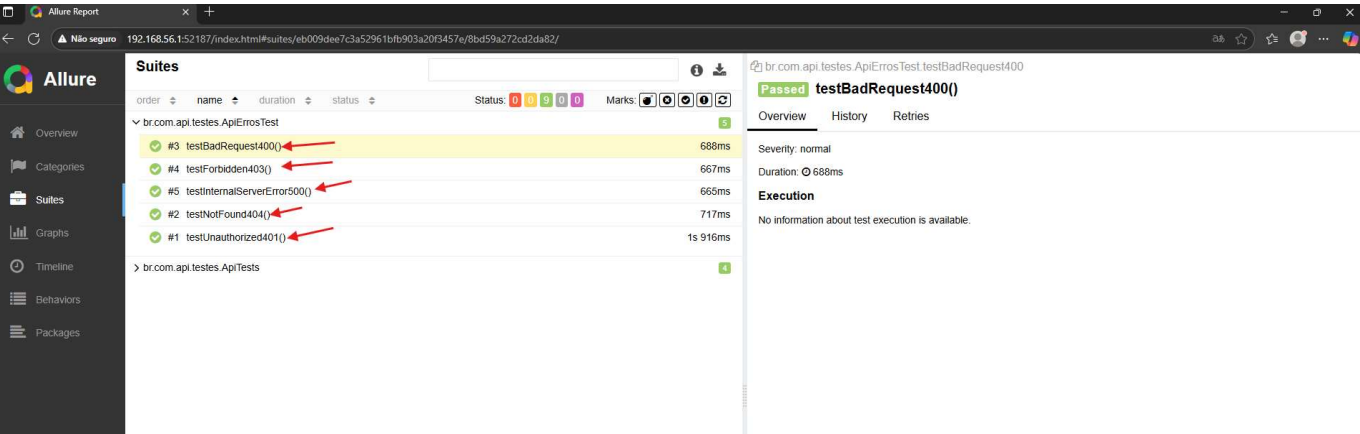
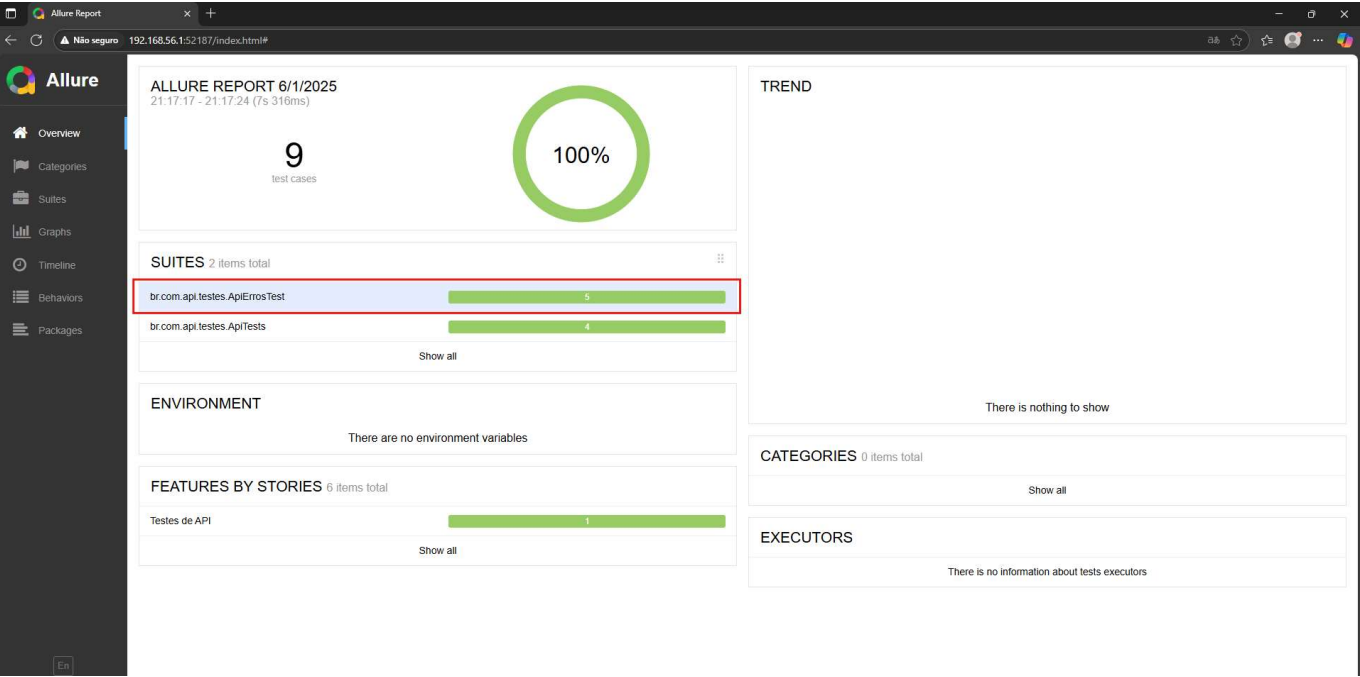
```
src > test > java > br > com > api > testes > ApiErrosTest.java > ApiErrosTest > testBadRequest400()

18 public class ApiErrosTest {
19     public void testBadRequest400() {
20         when()
21             .get(path: "/400") // Simula Bad Request
22             .then()
23             .statusCode(expectedStatusCode: 400);
24     }
25
26     @Test
27     public void testUnauthorized401() {
28         when()
29             .get(path: "/401") // Simula Unauthorized
30             .then()
31             .statusCode(expectedStatusCode: 401);
32     }
33
34     @Test
35     public void testForbidden403() {
36         when()
37             .get(path: "/403") // Simula Forbidden
38             .then()
39             .statusCode(expectedStatusCode: 403);
40     }
41
42     @Test
43     public void testNotFound404() {
44         when()
45             .get(path: "/404") // Simula Not Found
46             .then()
47             .statusCode(expectedStatusCode: 404);
48     }
49
50     @Test
51     public void testInternalServerError500() {
52         when()
53             .get(path: "/500") // Simula Internal Server Error
54             .then()
55             .statusCode(expectedStatusCode: 500);
56     }
57 }
```

Execução:

```
[INFO] -----
[INFO] T E S T S
[INFO] -----
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
[INFO] Running br.com.api.testes.ApiErrosTest
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 5.166 s - in br.com.api.testes.ApiErrosTest
[INFO] Running br.com.api.testes.ApiTests
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.56 s - in br.com.api.testes.ApiTests
[INFO] Results:
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 12.925 s
[INFO] Finished at: 2025-06-01T21:17:24-03:00
[INFO] -----
PS C:\TesteAPI> allure serve target/allure-results
Generating report to temp directory...
```

Results:



CONCLUSÃO

A automação de testes foi conduzida com sucesso, validando os principais métodos HTTP (**GET**, **POST**, **PUT**, **DELETE**) da API pública JSONPlaceholder, além de simular e verificar respostas com **códigos de erro** (**400**, **401**, **403**, **404**, **500**) utilizando a API httpstat.us.

Todos os testes foram desenvolvidos em **Java**, utilizando a biblioteca **Rest Assured** e o framework **JUnit 5**.

A integração com o **Allure Report** permitiu a geração de relatórios detalhados, com logs das requisições, dados das validações, status dos testes e evidências visuais claras, facilitando a rastreabilidade e análise de falhas (caso ocorram).

Todos os testes executaram conforme o esperado, sem apresentar erros ou falhas, demonstrando que:

- Os endpoints estão operacionais e respondem de forma consistente;
- A estrutura de automação está funcional, modular, reutilizável e de fácil manutenção;
- O projeto **já está integrado ao GitHub Actions**, possibilitando a execução automatizada dos testes a cada push ou pull request, fortalecendo a confiabilidade do processo de entrega contínua.

Este cenário comprova que a estratégia de testes automatizados adotada é eficaz tanto para **execução local** quanto para **pipelines de CI/CD**, agregando valor ao processo de validação de APIs REST e contribuindo para uma cultura de qualidade contínua.