10.1. Introduça o aos Testes de Integra ̧ o e Testes de APIs

terça-feira, 7 de março de 2023 09:42

Nesse módulo você vai aprender a implementar testes de integração e testes de APIs usando JUnit, REST Assured, etc.

Escrever bons testes pode te ajudar a evoluir os seus projetos sem correr o risco de quebrar funcionalidades que já estavam funcionando ou quebrar contratos da API que já estavam pré-estabelecidos.

Mas como qualquer outra coisa, precisa de um investimento em tempo para codificar esses testes.

Por isso, nós vamos implementar os testes apenas nesse módulo. Não continuaremos escrevendo testes nos módulos seguintes.

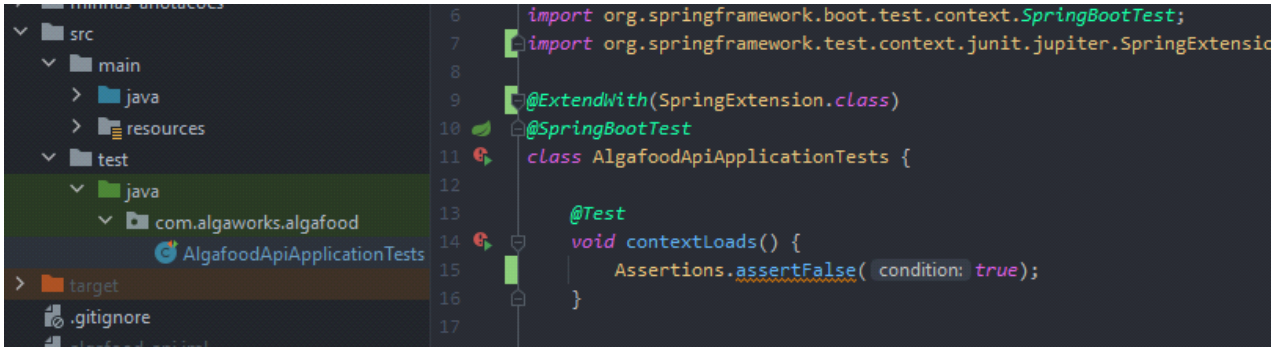
A ideia é que você aprenda a fazer isso e possa, de acordo com a sua decisão, escrever testes para as suas APIs.

Teste de integração: Valida o funcionamento dos softwares em conjunto, não isolamos as dependências. ex: a camada do controlador se comunica com a de serviço para executar uma query no repositório e persistir a entidade no banco de dados. Todos esses módulos são testados.

Testes de APIs: Tipo de teste de integração: chamadas HTTP e esperar uma resposta. Esperamos fazer todo o caminho que a requisição se garanta fazer. end-to-end

10.2. Preparando o projeto para testes de integração

terça-feira, 7 de março de 2023 09:43



Nota para atualização de versão
Atualização para o JUnit 5
Este documento irá auxiliá-lo no uso do JUnit 5.

Essas alterações são necessárias apenas para quem utiliza uma versão do Spring Boot superior a 2.4.0, a qual utiliza o JUnit 5 com padrão ao invés do JUnit 4.

Dependências no pom.xml
O parent deve estar descrito da seguinte forma, utilizando a versão 2.4.x ou superior, como por exemplo a 2.7.4:

```
<parent>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-parent</artifactId>
 <version>2.7.4</version>
 <relativePath/>
</parent>
```

A dependência do spring-boot-starter-test continua sem alteração:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

Mas a versão 2.4.0 necessita da dependência do spring-boot-starter-validation declarada de forma explicita, já que a mesma não é mais incluída por padrão:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
```

```
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Alterações nos imports
A versão 5 sofreu diversas alterações nas annotations, tanto nos nomes quanto na localidade (package), a tabela abaixo exibe as diferenças entre as versões:

JUnit 4	JUnit 5
org.junit.Test	org.junit.jupiter.api.Test
org.junit.Before	org.junit.jupiter.api.BeforeEach
org.junit.After	org.junit.jupiter.api.AfterEach
org.junit.BeforeClass	org.junit.jupiter.api.BeforeAll
org.junit.AfterClass	org.junit.jupiter.api.AfterAll
org.junit.runner.RunWith*	org.junit.jupiter.api.extension.ExtendWith

Nota: A annotation @RunWith foi removida, foi adicionado uma similar a @ExtendWith

Atualizando a classe de testes
Agora vamos atualizar a nossa classe de testes a CadastroCozinhaIntegrationTests.

Seguindo a tabela de alterações, vamos atualizar os imports de org.junit.Test para import org.junit.Test.

```
import org.junit.jupiter.api.Test;
```

Não é necessário substituir a annotation @RunWith(SpringRunner.class) por @ExtendWith(SpringExtension.class), já que a @SpringBootTest já a inclui.

```
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.test.context.junit.jupiter.SpringExtension;

//@ExtendWith(SpringExtension.class) //Desnecessário
@SpringBootTest
public class CadastroCozinhaIntegrationTests {
  ...
}
```

- para utilizar asserts, importar a classe import org.junit.jupiter.api.Assertions;

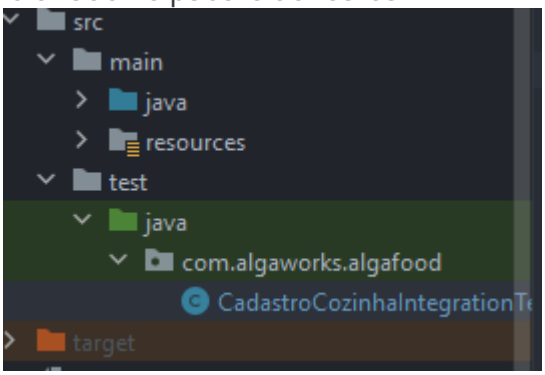
- A classe SpringRunner era compatível com a versão 4 do JUnit, e a anotação @RunWith. No JUnit 5 usamos a classe SpringExtension: @ExtendWith(SpringExtension.class)

por mais que a Annotation @ExtendWith tenha o comportamento similar ao da @RunWith do JUnit 4, ela não se torna necessária quando usamos o @SpringBootTest.

10.3. Criando e rodando um teste de integração com Spring Boot, JUnit e AssertJ

terça-feira, 7 de março de 2023 10:23

Todo código de teste é adicionado no pacote de testes



Adicionar um nome descritivo ao método de teste.
Anotar os métodos de testes com @Test e a classe de teste com @SpringBootTest.
Para aumentar a legibilidade do teste e das asserções, é recomendável importar estaticamente a respectiva classe. Dessa forma, podemos nos referir diretamente ao próprio método de asserção sem a classe representativa como um prefixo.

```
import static org.assertj.core.api.Assertions.assertThat;
```

O código de teste é divido em 3 partes:

- Cenário
- Ação
- Validação

O código do teste pode ser caracterizado por Happy Path ou caminho feliz/positivo para determinar códigos que fazem exatamente o que o método se propõe a fazer. ex: Devemos escrever uma lógica para o método testarCadastroUsuario que de fato cadastre um usuário, sem simulação de erro. esperamos que faça o caminho para qual foi definido e não definimos nenhum erro.

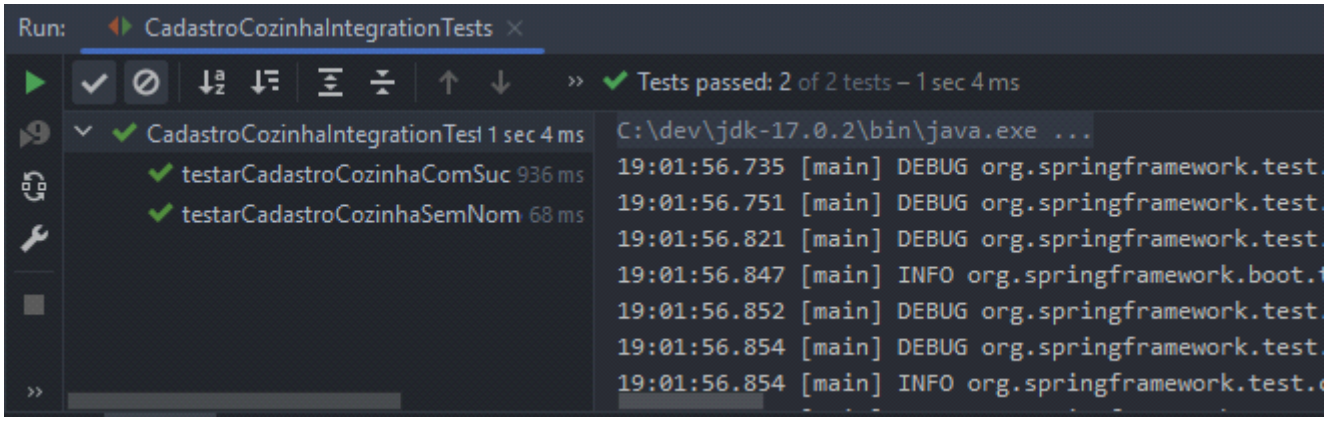
```java
import static org.assertj.core.api.Assertions.assertThat;

@SpringBootTest
class CadastroCozinhaIntegrationTests {

    1 usage
    @Autowired
    private CozinhaService cozinhaService;

    @Test()

    public void testarCadastroCozinhaComSucesso(){
        // cenário
        Cozinha novaCozinha = new Cozinha();
        novaCozinha.setNome("Chinesa");

        // ação
        novaCozinha = cozinhaService.salvar(novaCozinha);

        // validação
        assertThat(novaCozinha).isNotNull();
        assertThat(novaCozinha.getId()).isNotNull();
    }
}
```

Podemos fazer quantas asserções forem necessárias, não há um limite, mas tem que ser discutido com as regras da empresa. O método tem que ser coerente com a proposta e coerente com as asserções. Vai de bom senso e com objetos similares.

Há uma convenção para definir nome de métodos Java, mas não se aplica em métodos de Teste, incluindo utilizar underscore/underline.

```java
@Test
/*bad path*/
public void testarCadastroCozinhaSemNome(){
    // cenário
    Cozinha novaCozinha = new Cozinha();
    novaCozinha.setNome(null);

    // ação
    Assertions.assertThrows(ConstraintViolationException.class,
            () -> cozinhaService.salvar(novaCozinha));
}
```

Agora em um método que esperamos que seja lançada exceções, temos que tratar elas com o próprio jUnit usando bibliotecas internas como assertThrows da classe Assertions do pacote org.junit.jupiter.api.Assertions; o 1º argumento espera a exceção que esperamos e o 2º argumento um Executable utilizando lambda.


Run:
CadastroCozinhaIntegrationTests
Tests passed: 2 of 2 tests – 1 sec 4 ms
CadastroCozinhaIntegrationTest 1 sec 4 ms
testarCadastroCozinhaComSuc 936 ms
testarCadastroCozinhaSemNom 68 ms
C:\dev\jdk-17.0.2\bin\java.exe ...
19:01:56.735 [main] DEBUG org.springframework.test
19:01:56.751 [main] DEBUG org.springframework.test
19:01:56.821 [main] DEBUG org.springframework.test
19:01:56.847 [main] INFO org.springframework.boot.
19:01:56.852 [main] DEBUG org.springframework.test
19:01:56.854 [main] DEBUG org.springframework.test
19:01:56.854 [main] INFO org.springframework.test.

10.4. Escrevendo bons nomes de testes

terça-feira, 7 de março de 2023 18:47

```java
@Test
public void deveAtribuirId_QuandoCadastrarCozinhaComDadosCorretos() {
    Cozinha novaCozinha = new Cozinha();
    novaCozinha.setNome("Chinesa");

    novaCozinha = cadastroCozinha.salvar(novaCozinha);

    assertThat(novaCozinha).isNotNull();
    assertThat(novaCozinha.getId()).isNotNull();
}

@Test(expected = ConstraintViolationException.class)
public void deveFalhar_QuandoCadastrarCozinhaSemNome() {
    Cozinha novaCozinha = new Cozinha();
    novaCozinha.setNome(null);

    novaCozinha = cadastroCozinha.salvar(novaCozinha);
}
```

10.5. Desafio- escrevendo testes de integração

terça-feira, 7 de março de 2023 19:15

Desafio: implementar testes bad path com os nomes `deveFalhar_QuandoExcluirCozinhaEmUso` e `deveFalhar_QuandoExcluirCozinhaInexistente`.

```java
@Test
/* bad path */
public void deveFalhar_QuandoExcluirCozinhaEmUso() {
    final Long cozinhaId = restauranteService.buscarOuFalhar( restauranteId: 1L).getCozinha().getId();

    final EntidadeEmUsoException erroEsperado = Assertions.
            assertThrows(EntidadeEmUsoException.class, () -> cozinhaService.deletar(cozinhaId));
    assertThat(erroEsperado).isNotNull();
}
```

```java
@Test
/* bad path */
public void deveFalhar_QuandoExcluirCozinhaInexistente() {
    final Long cozinhaId = 55L;

    final ConstraintViolationException erroEsperado = Assertions
            .assertThrows(ConstraintViolationException.class, () -> cozinhaService.deletar(cozinhaId));
    System.out.println("antes do assertThaht");

    assertThat(erroEsperado).isNotNull();
}
```

o método `deveFalhar_QuandoExcluirCozinhaEmUso` resgata um id de uma cozinha existente e utilizada em outra entidade e tenta deletar, o método Assertions.assertThrows espera realmente que uma exceção aconteça para o método cumprir sua responsabilidade, falhar quando excluir uma cozinha em uso, o teste é checado e não contém falhas.

```
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 14.051
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-failsafe-plugin:3.0.0-M9:verify (default) @ algafood-api ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  20.005 s
[INFO] Finished at: 2023-03-08T18:04:26-03:00
[INFO] ------------------------------------------------------------------------
PS C:\Users\Guto1\stsworkspace\AlgaWorks\algafood-api>
```

10.6. Rodando os testes pelo Maven

quarta-feira, 8 de março de 2023 10:58

./mvnw test
./mvnw clean package
Ao utilizar comandos pelo maven, ele realiza automaticamente os testes de integração e o ideal para os testes automáticos são os unitários.

10.7. Configurando Maven Failsafe Plugin no projeto

quarta-feira, 8 de março de 2023 15:53

Plugin do Maven para executar testes de integração

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-failsafe-plugin</artifactId>
<version>3.0.0-M9</version>
</plugin>
```

Como foi visto em 10.6. Rodando os testes pelo Maven ao fazer um build do projeto, ele executa os testes de integração. Os testes de integração não devem ser feitos automaticamente.

Toda classe de teste de integração deve seguir uma convenção de nome com sufixo IT de Integration Test.

para rodar os testes de integração é necessário

./mvnw verify

10.8. Implementando Testes de API com REST Assured e validando o código de status HTTP

quarta-feira, 8 de março de 2023 18:02

A importação da anotação LocalServerPort foi alterada de
import org.springframework.boot.web.server.LocalServerPort;
para
import org.springframework.boot.test.web.server.LocalServerPort;

Faz uma chamada na API utilizando a biblioteca RestAssured uma biblioteca de Teste e Validação com uma linguagem fluente

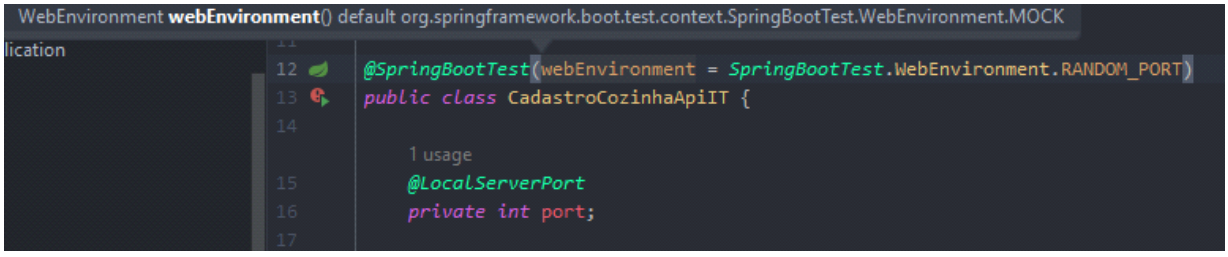
- importação
  <dependency>
  <groupId>io.rest-assured</groupId>
  <artifactId>rest-assured</artifactId>
  <scope>test</scope>
  </dependency>

► Não é uma biblioteca para produção, apenas para testes.

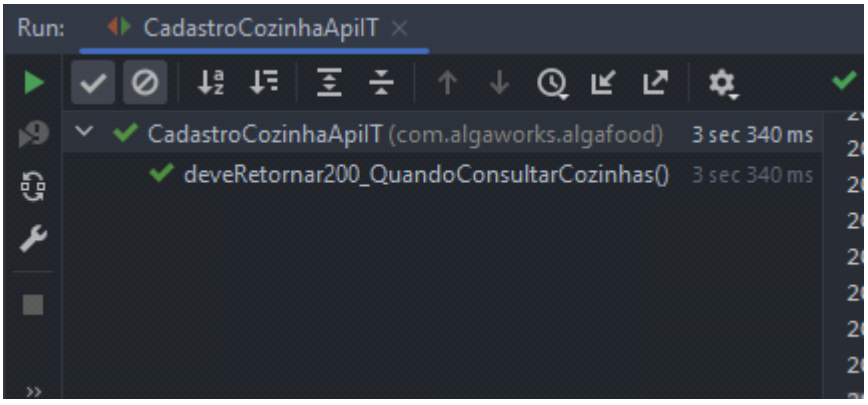
```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class CadastroCozinhaApiIT {

    1 usage
    @LocalServerPort
    private int port;

    @Test
    public void deveRetornar200_QuandoConsultarCozinhas(){

        RestAssured.enableLoggingOfRequestAndResponseIfValidationFails();

        RestAssured.given() RequestSpecification
                .basePath( s: "/cozinhas")
                .port(port)
                .accept(ContentType.JSON)
            .when()
                .get() Response
            .then() ValidatableResponse
                .statusCode(HttpStatus.OK.value());
    }
}
```

chamando given() de RestAssured dado que eu tenho um basepath em "/cozinhas" na porta com o valor da variável com o tipo de retorno tipo JSON (io.restassured.http.ContentType) quando fizer uma requisição get então o statuscode precisa ser 200.

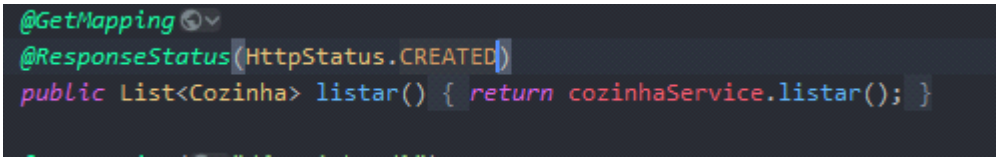
A anotação a nível de classe permite o método levantar um container web com a aplicação somente para os testes. Podemos configurar isso com a propriedade webEnvironment.



por padrão, webEnvironment é do tipo MOCK e podemos trocar para RANDOM_PORT, uma porta aleatória, porém, injetamos a porta em uma variável com a anotação @LocalServerPort e passamos como argumento de given().



Podemos forçar um erro na requisição ao fazer retornar um statuscode diferente do esperado



o método na linha 21 enableLoggingOfRequestAndResponseIfValidationFails(); habilita o log de erro da requisição e resposta se a validação falha.

```
Request method: GET
Request URI:    http://localhost:57183/cozinhas
Proxy:          <none>
Request params: <none>
Query params:   <none>
Form params:    <none>
Path params:    <none>
Headers:        Accept=application/json, application/javascript, text/javascript, text/json
Cookies:        <none>
Multiparts:     <none>
Body:           <none>

HTTP/1.1 201
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 09 Mar 2023 12:19:44 GMT
Keep-Alive: timeout=60
Connection: keep-alive

[
    {
        "id": 1,
        "nome": "Tailandesa"
    },
    {
        "id": 2,
        "nome": "Indiana"
    },
    {
        "id": 3,
        "nome": "Argentina"
    },
```

```
java.lang.AssertionError: 1 expectation failed.
Expected status code <200> but was <201>.
```

10.9. Validando o corpo da resposta HTTP

quinta-feira, 9 de março de 2023 09:50

Podemos validar o corpo da mesma maneira da aula 10.8, porém, mudando a linguagem fluente: Dado que (given) quando(when) então(then).

```
@Test
public void deveConter4Cozinhas_QuandoConsultarCozinhas(){

    enableLoggingOfRequestAndResponseIfValidationFails();

    given() RequestSpecification
            .basePath( s: "/cozinhas")
            .port(port)
            .accept(ContentType.JSON)
        .when()
            .get() Response
        .then() ValidatableResponse
            .body( s: "", Matchers.hasSize(8))
            .body( s: "nome", Matchers.hasItems("Indiana", "Tailandesa"));
}
```

Vamos validar o corpo da resposta com alguns parâmetros de validação do teste chamado .body (corpo) e utilizando a biblioteca org.hamcrest.

- given(): dado que tempos um basepath, port e tipo aceito na resposta
- when(): quando fazer requisição get
- then(): então validar se o corpo como uma propriedade vazia, contém 8 itens json
    - e então, no corpo, a propriedade "nome" dos objetos, conter os itens "Indiana" E "Tailandesa"

- .body("", Matchers.hasSize(8))
    - 1º argumento: branco. 2º argumento: Se o array de objetos JSON contém é de tamanho 8(contém 8 objetos no array)
    - Quando passamos uma string vazia, retorna um array de objetos JSON.
    - Não é necessário filtrar somente algum dos atributos se for contar a quantidade de objetos retornados na requisição. Passando uma string vazia retorna todo o objeto, e o hasSize conta.

```
java.lang.AssertionError: 1 expectation failed.
JSON path  doesn't match.
Expected: a collection with size <94>
  Actual: <[{id=1, nome=Tailandesa}, {id=2, nome=Indiana}, {id=3, nome=Argentina}, {id=4, nome=Brasileira}]>
```

- .body("nome", Matchers.hasItems("Indiana", "Tailandesa"));
    - 1º argumento: "nome": retorna um array de valores do atributo "nome" do array json de objetos.
    - 2º argumento: itera sobre cada um dos itens e verifica se existem um valor com os argumentos em String - "Indiana", "Tailandesa"

```
java.lang.AssertionError: 1 expectation failed.
JSON path nome doesn't match.
Expected: (a collection containing "teste")
  Actual: <[Tailandesa, Indiana, Argentina, Brasileira]>
```

Fazendo consulta em restaurantes (7)

```
51      @Test
52      public void deveConter7Restaurantes_QuandoConsultarRestaurantes() {
53          RestAssured.enableLoggingOfRequestAndResponseIfValidationFails();
54
55          RestAssured.given() RequestSpecification
56                  .basePath("/restaurantes")
57                  .port(port)
58                  .accept(ContentType.JSON)
59                  .when()
60                  .get() Response
61                  .then() ValidatableResponse
62                  .body( path: "", Matchers.hasSize(7))
63                  .body( path: "nome", Matchers.hasItems("Thai Gourmet"))
64                  .body( path: "taxaFrete", Matchers.hasItems(Matchers.greaterThan( value: 5.0f)));
65
66      }
```

- Na linha 64, retorna um array de valores, no 2º argumento itera sobre os itens e para cada item verifica se o valor é maior que 5.0f. caso não há ocorrerias, o teste falha.

Verifica fazendo uma requisição em todos os restaurantes, validando se o corpo da requisição possui um objeto com a propriedade nome contendo "Frete" e o campo taxaFrete sendo igual a 9.5

```
67      @Test
68      public void deveConterRestauranteComFreteGratis_e_nomeComFreteGratis_QuandoConsultarRestaurantes() {
69          RestAssured.enableLoggingOfRequestAndResponseIfValidationFails();
70
71          RestAssured.given() RequestSpecification
72                  .basePath("/restaurantes")
73                  .port(port)
74                  .accept(ContentType.JSON)
75              .when()
76                  .get() Response
77              .then() ValidatableResponse
78                  .body( path: "nome", Matchers.hasItems(Matchers.containsString( substring: "Frete")))
79                  .body( path: "taxaFrete", Matchers.hasItems(Matchers.equalTo( operand: 9.5f)));
80      }
```

```
@Test
public void deveConter1RestauranteComFreteGratis_e_nomeComFreteGratis_QuandoConsultarRestaurantes() {
    RestAssured.enableLoggingOfRequestAndResponseIfValidationFails();

    RestAssured.given() RequestSpecification
            .basePath("/restaurantes")
            .port(port)
            .accept(ContentType.JSON)
            .when()
            .get( path: "/1") Response
            .then() ValidatableResponse
            .body( path: "taxaFrete", Matchers.equalTo( operand: 10.0f));

}
```

fazendo get e filtrando se o campo "taxafrete" é igual a 10.0f

Hamcrest

sábado, 11 de março de 2023 09:20

Framework conhecido por auxiliar em testes de unidade e de API em java. Ele é agrupado no Junit e simplesmente usa predicates para fazer as asserções, predicates esses chamados de matchers.

referências e exemplos: https://www.baeldung.com/java-junit-hamcrest-guide

REST-assured

sábado, 11 de março de 2023 09:42

Foi projetado para auxiliar nos testes e validações de regras de API e é altamente influenciado por técnicas de testes usadas em linguagens dinâmicas como Groovy e Ruby.

Oferece suporte sólido para HTTP.

O REST-assured aproveita o poder dos matchers Hamcrest para executar suas asserções.

Exemplos interessantes:

referências: https://www.baeldung.com/rest-assured-tutorial

10.10. Criando um método para fazer setup dos testes

quinta-feira, 9 de março de 2023 15:13

Podemos criar um método que pode ser executado antes dos testes com @Test.

```
@BeforeEach
public void SetUp(){
    enableLoggingOfRequestAndResponseIfValidationFails();
     RestAssured.basePath = "/cozinhas";
     RestAssured.port = port;
}
```

Usando a anotação @BeforeEach (jUnit5)

10.11. Entendendo o problema da ordem de execução dos testes

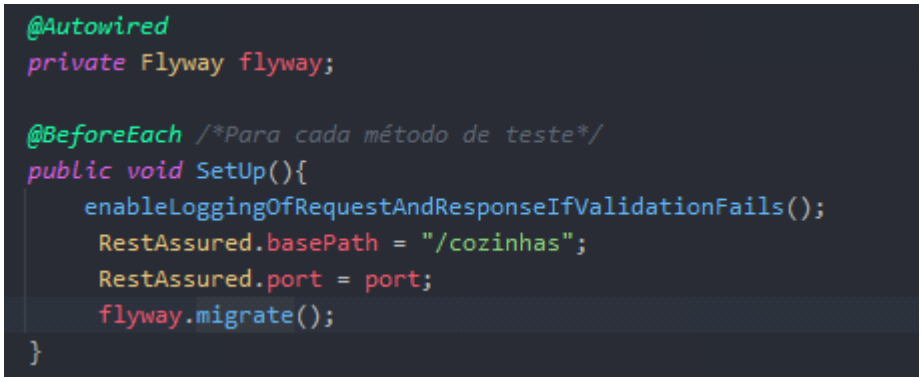
quinta-feira, 9 de março de 2023 15:24

- Os métodos de testes não podem depender de outros métodos de teste.
- Métodos de teste não devem afetar outros métodos de teste.

Podemos executar um arquivo de callback para reiniciar a massa de dados sempre que um método de teste for executado como em 10.12. Voltando o estado inicial do banco de dados para cada execução de teste com callback do Flyway

10.12. Voltando o estado inicial do banco de dados para cada execução de teste com callback do Flyway

quinta-feira, 9 de março de 2023 16:03



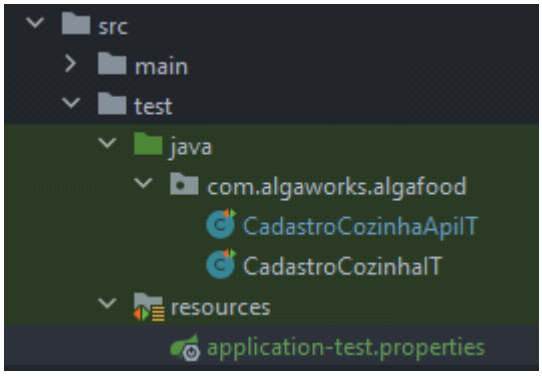
```java
@Autowired
private Flyway flyway;

@BeforeEach /*Para cada método de teste*/
public void SetUp(){
    enableLoggingOfRequestAndResponseIfValidationFails();
     RestAssured.basePath = "/cozinhas";
     RestAssured.port = port;
     flyway.migrate();
}
```

Mas o ideal é ter um afterMigrate de testes

10.13. Configurando um banco de testes e usando @TestPropertySource

quinta-feira, 9 de março de 2023 16:40

A ideia é isolar a base de testes da base de desenvolvimento e conter seu próprio banco de dados. Além de conter o properties de test



src/test/resources
application-test.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/algafood_test?createDatabaseIfNotExist=true&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=1042

#10.13. Configurando um banco de testes e usando @TestPropertySource
#caminho do afterMigrate deletado
#O banco de teste vai começar vazio ao rodar os testes de integração
#o Spring vai continuar usando as configurações de application.properties
# mas somente as configurações encontradas neste arquivo serão substituídas
spring.flyway.locations=classpath:db/migration

spring.datasource.hikari.maximum-pool-size=1
```

- Base de dados: algafood_test
- pool de conexões do hikari: 1 pois não é necessário várias conexções para testes
- localização do arquivo do flyway: as migrations padrão serão executadas no ambiente de teste mas os arquivos do afterMigrate não.

```
#7.10 Adicionando dados de testes com callback do Flyway
#indicando ao Flyway as migrations e os dados de teste
spring.flyway.locations=classpath:db/migration,classpath:db/data
```

  - application.properties

Na classe de testes, anotar com @TestPropertySource e passando o path do novo application-test.properties

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@TestPropertySource("/application-test.properties")
public class CadastroCozinhaApiIT {
```

Os testes usarão as propriedades de application.properties padrão, porém, serão substituídas somente as configurações encontradas em application-test.properties

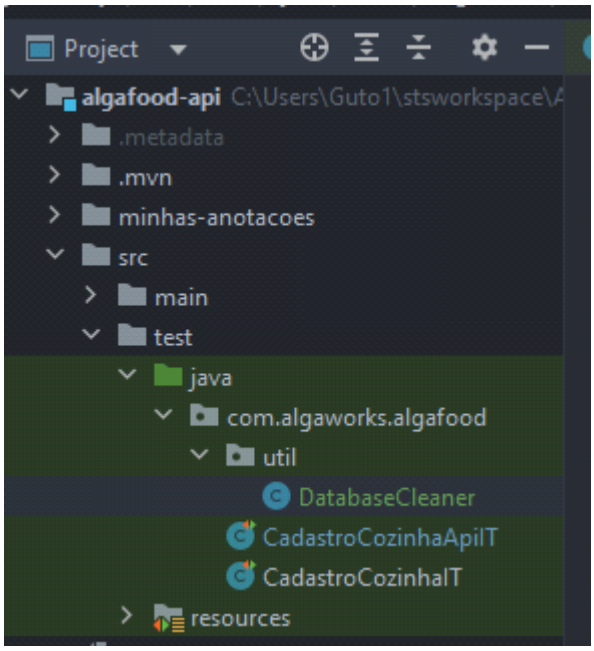
Agora configuramos uma nova base de dados de teste e isolamos a base de teste.

Todos os testes de alteração da base de teste não serão redefinidos, a alteração fica na base de testes. Na próxima página é visto como popular a base sem o uso do Flyway

10.14. Limpando e populando o banco de dados de teste

quinta-feira, 9 de março de 2023 19:16

Vamos usar uma classe para deletar os registros do banco de dados

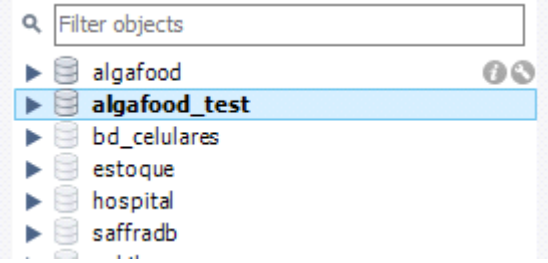



```
1 usage
    private void prepararDados(){
        Cozinha cozinha = new Cozinha();
        cozinha.setNome("Americana");
        Cozinha cozinha2 = new Cozinha();
        cozinha2.setNome("Portuguesa");

        cozinhaRepository.saveAll(Arrays.asList(cozinha, cozinha2));
    }
}
```

```java
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@TestPropertySource("/application-test.properties")
public class CadastroCozinhaApiIT {

    1 usage
    @LocalServerPort
    private int port;

    @Autowired
    private Flyway flyway;

    /*Classe para limpar base de dados*/
    1 usage
    @Autowired
    private DatabaseCleaner databaseCleaner;

    1 usage
    @Autowired
    private CozinhaRepository cozinhaRepository;

    @BeforeEach /*Executa sempre antes de um método*/
    public void SetUp(){
        enableLoggingOfRequestAndResponseIfValidationFails();
         RestAssured.basePath = "/cozinhas";
         RestAssured.port = port;
         databaseCleaner.clearTables();
         prepararDados();
    }
```

Agora para cada um dos testes, antes de executar, chama os métodos

databaseCleaner.clearTables();
prepararDados();
para limpar e adicionar dados no banco de dados de teste

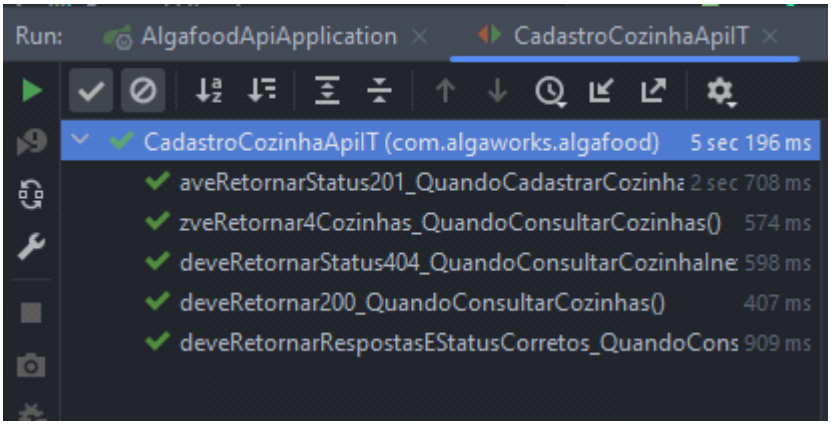


10.15. Testando endpoint passando parâmetro de URL

quinta-feira, 9 de março de 2023 19:20

```java
@Test
public void deveRetornarRespostasEStatusCorretos_QuandoConsultarCozinhaExistente(){
    given() RequestSpecification
            .accept(ContentType.JSON)
            .pathParams( firstParameterName: "cozinhaId", firstParameterValue: 2)
        .when()
            .get( path: "/{cozinhaId}") Response
        .then() ValidatableResponse
            .statusCode(HttpStatus.OK.value())
            .body( path: "nome", Matchers.equalTo( operand: "Portuguesa"));
}
```

usando pathParams



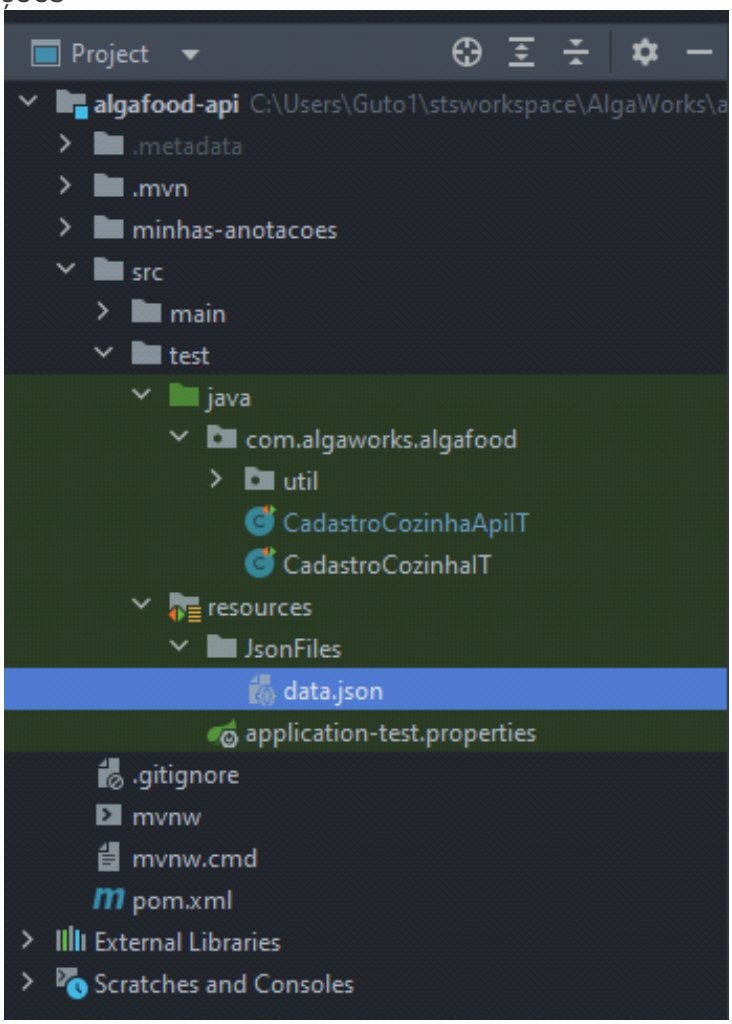
10.16. Desafio refatorando o código de testes

quinta-feira, 9 de março de 2023 20:47

- Deixar o método de teste de contagem de registros com uma variável dinâmica de acordo com o número total de registros no banco.

```
1 usage
private long quantidadeDeCozinhas(){
    return cozinhaRepository.count();
}
```

- criar um arquivo json para servir de referência para adição de recursos no corpo de requisições



```java
@Test
public void aveRetornarStatus201_QuandoCadastrarCozinha(){
    given() RequestSpecification
            .accept(ContentType.JSON)
            .contentType(ContentType.JSON)
            .body(ResourceUtils.getContentFromResource(FILES_DATA_JSON))
        .when()
            .post() Response
        .then() ValidatableResponse
            .statusCode(HttpStatus.CREATED.value());
}
```

```java
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@TestPropertySource("/application-test.properties")
public class CadastroCozinhaApiIT {

    1 usage
    public static final int ID = 100;
    1 usage
    public static final String FILES_DATA_JSON = "/JsonFiles/data.json";
    1 usage
    @LocalServerPort
    private int port;

    @Autowired
    private Flyway flyway;

    /*Classe para Limpar base de dados*/
```

- adicionar uma instância de uma entidade para fazer testes dinâmicos

```java
    1 usage
    public static final int ID = 100;
    1 usage
```

10.17. Desafio escrevendo testes de API

quinta-feira, 9 de março de 2023 22:04

Devemos escrever testes agregando valor.
e devemos escrever testes baseados em regras e contratos da API.
Funcionalidades do endpoint para testes da api.
Testar caminhos infelizes.

Escrever testes para Restaurantes

- [x] Preparar dados de testes
- [x] Criar classe que transforma texto de arquivo em variável de String
- [x] Preparar classe que deleta a base de dados a cada teste
- [x] Preparar método de SetUp com @BeforeEach para rodar a cada teste
- [x] Habilitar log de falhas para falhas de teste do pacote io.restassured.RestAssured
  enableLoggingOfRequestAndResponseIfValidationFails

- [x] Usar os métodos de io.restassured.RestAssured para fazer requisições de teste
- [x] Utilizar Hamcrest e RestAssured