Tutorial SpringBoot com JPA, Hibernate, H2 e Rest

Parte 3

Glauco Todesco

Download:

https://github.com/glaucotodesco/CursoSpringBootComAngular/tree/Tutorial-Parte3/HelloSpringBootRest

Nessa terceira parte do tutorial iremos inserir um simples teste de unidade, introduzindo o conceito de TDD (Desenvolvimento Guiado por Testes)

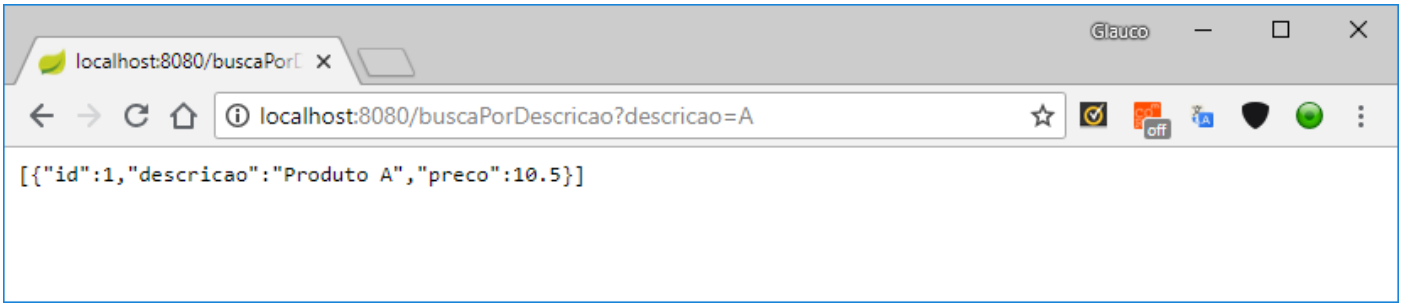
1. Vamos criar um método no repositório, alterando a classe ProdutoRepositorio.java:

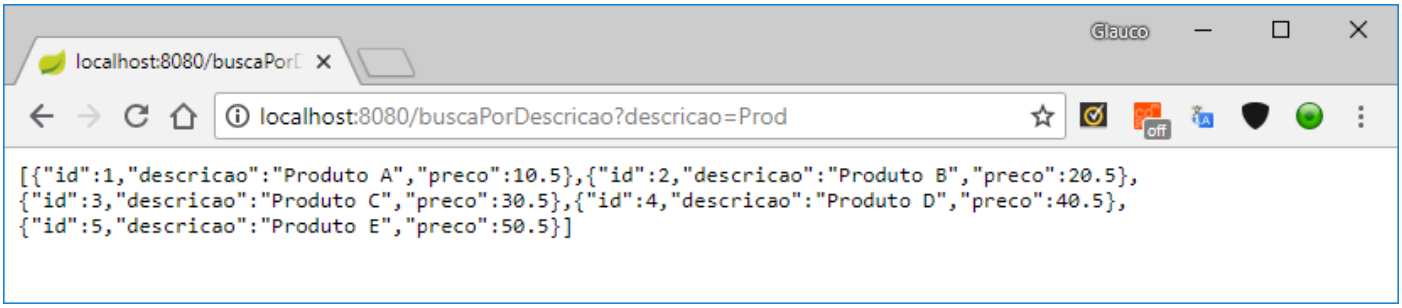
```
package com.abutua.springboot.HelloSpringBootRest;

import java.util.List;

import org.springframework.data.repository.CrudRepository;

public interface ProdutoRepositorio extends CrudRepository <Produto, Long> {

	public List <Produto> findByDescricaoLikeIgnoreCase(String descricao);

}
```

2. Vamos acrescentar um novo serviço ao Rest, alterando o arquivo Cotroller.java para usar o novo método do repositório:

```
package com.abutua.springboot.HelloSpringBootRest;


import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class Controller {

	@Autowired
	ProdutoRepositorio rep;

	@RequestMapping("/")
	public Iterable<Produto> index() {
		return rep.findAll();
	}

	@RequestMapping("/buscaPorDescricao")
	public List<Produto> buscaPorDescricao(@RequestParam("descricao") String descricao){
		return rep.findByDescricaoLikeIgnoreCase("%" + descricao + "%");
	}

}
```

3. Agora podemos executar uma chamada ao novo serviço de Rest criado pelo navegador. Note que o parâmetro descrição pode mudar a cada chamada:



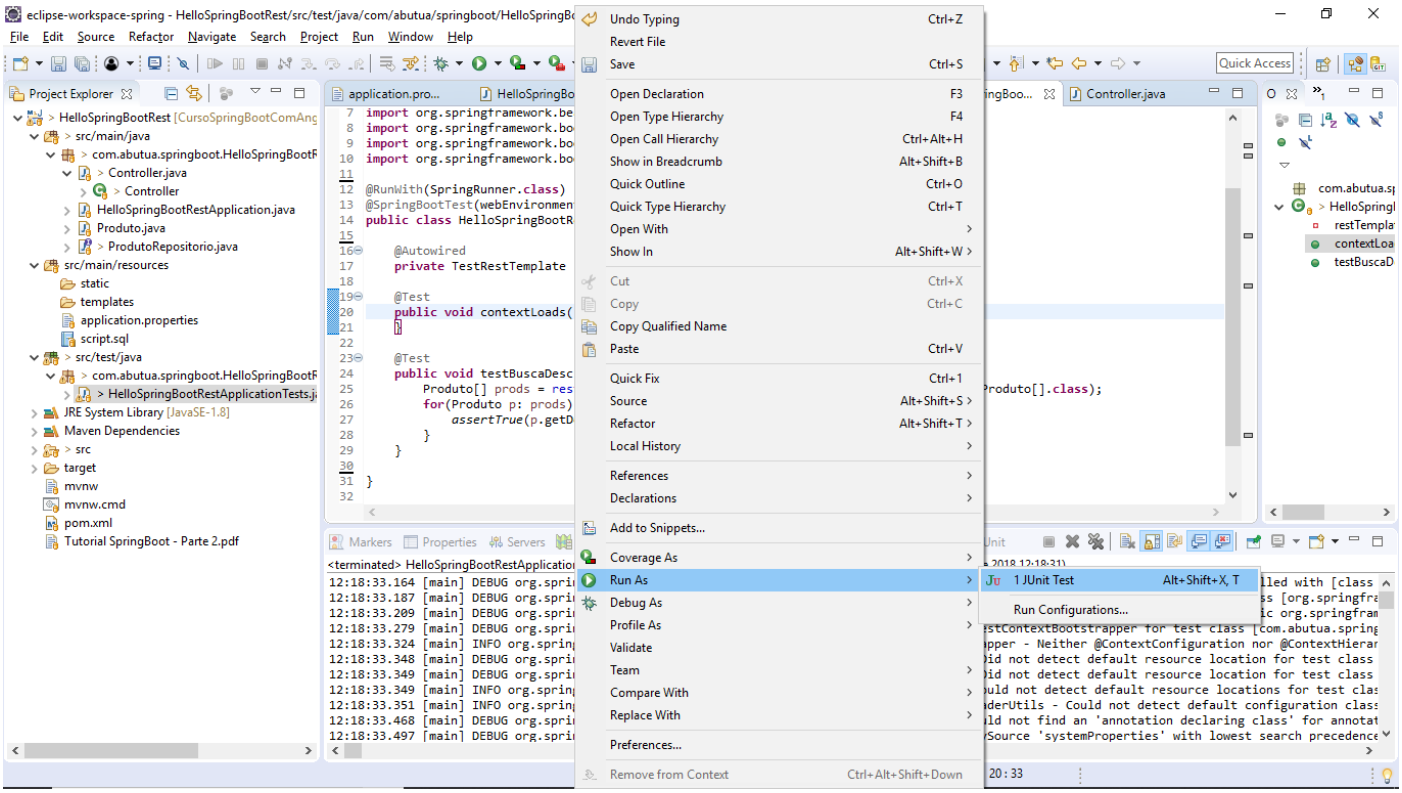


4. A próxima etapa é inserir um exemplo simples de um teste de unidade alterando a classe HelloSpringBootRestApplicationTests.java. O teste apresentado verifica que o tamanho da descrição de um produto é maior do que 2 caracteres.

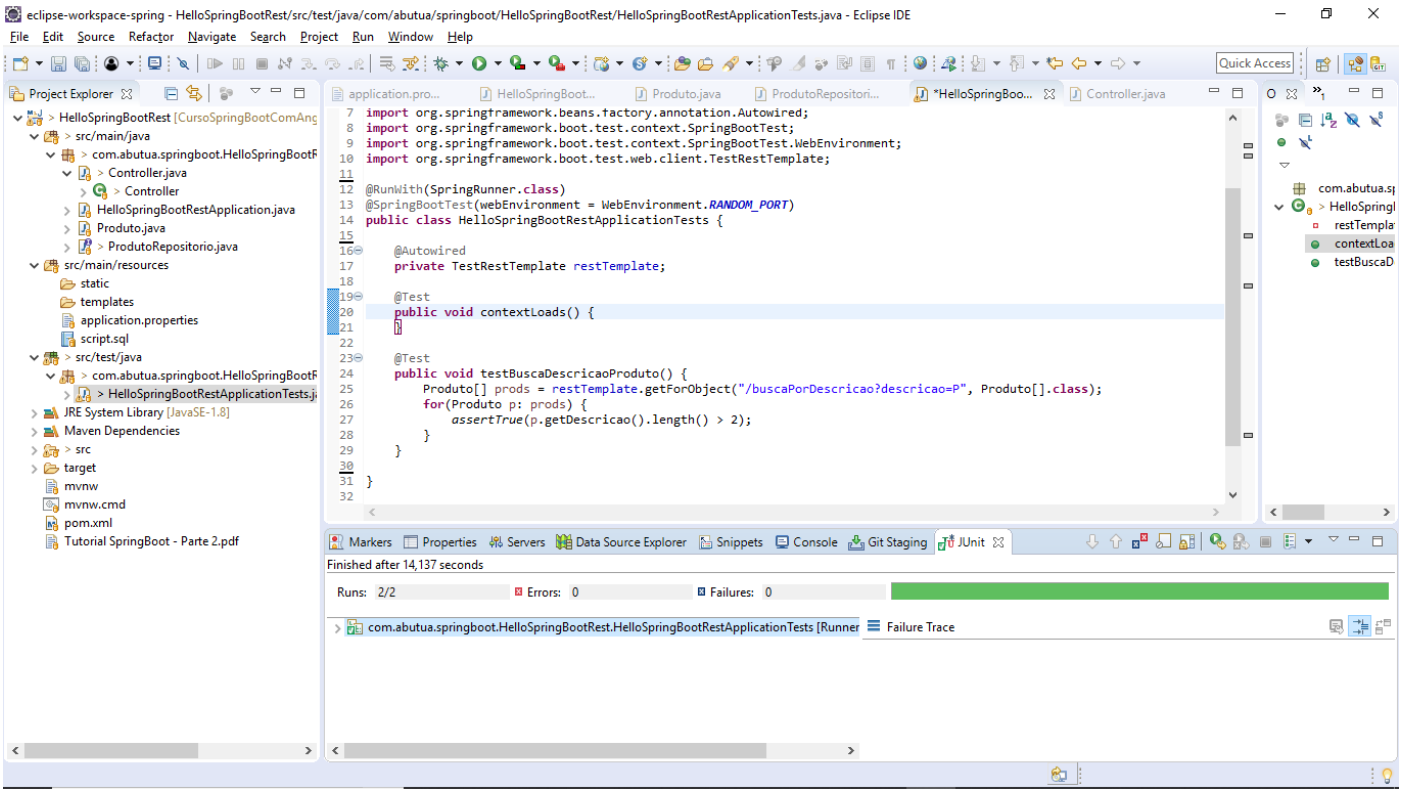
```
package com.abutua.springboot.HelloSpringBootRest;

import static org.junit.Assert.assertTrue;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
import org.springframework.boot.test.web.client.TestRestTemplate;

@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
public class HelloSpringBootRestApplicationTests {

	@Autowired
	private TestRestTemplate restTemplate;

	@Test
	public void contextLoads() {
	}

	@Test
	public void testBuscaDescricaoProduto() {
		Produto[] prods = restTemplate.
				getForObject("/buscaPorDescricao?descricao=P", Produto[].class);
		for(Produto p: prods) {
			assertTrue(p.getDescricao().length() > 2);
		}
	}
```

}

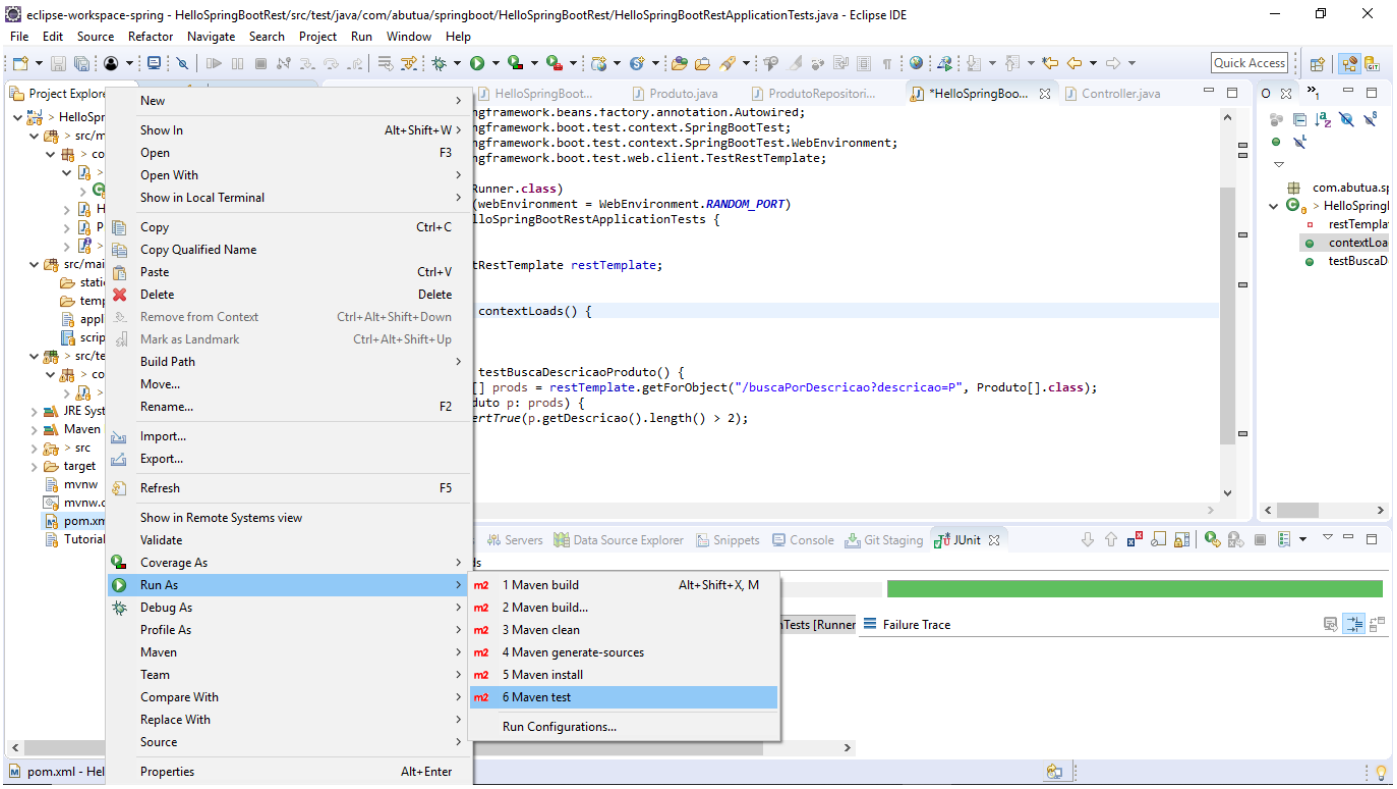
5. Para executar o teste iremos apresentar duas formas. A primeira é executando pelo Eclipse o teste:



Para ver o resultado do teste acesse a janela do JUnit. Na imagem abaixo temos 0 erros e 0 falhas.



A segunda forma de executar é pelo Maven. Selecione o arquivo pom.xml, botão direito, Run -> Maven Test.



No final da execução serão apresentados a quantidade de testes, erros e falhas: