02

Utilizando JAX-RS para realizar o pagamento

Transcrição

Vamos implementar o código para que a requisição seja executada através de um serviço REST de dentro da nossa aplicação para uma externa, não importa onde. Para isso utilizaremos o JAX-RS, uma especificação Java EE que nos diz como devemos fazer uma requisição REST, sendo que o JAX-WS é para SOAP.

Primeiro precisamos de um cliente que faça tal requisição. Então, no `CarrinhoCompras.java`, dentro do `finalizar()`:

```
Client client = ClientBuilder.newClient();
```

Vamos pegar esse client do JAX-RS e realizar o pagamento:

```
Client client = ClientBuilder.newClient();
Pagamento pagamento = new Pagamento(getTotal());
```

Já já vamos trabalhar em cima desse `Pagamento`. Agora iremos dizer qual o *target* dele, que seria o endereço que utilizamos no DHC:

```
Client client = ClientBuilder.newClient();
Pagamento pagamento = new Pagamento(getTotal());
String target = "http://book-payment.herokuapp.com/payment"
```

A Classe "Entity" do JAX-RS é quem faz a transformação de um determinado objeto para Json para assim ser enviado:

```
Client client = ClientBuilder.newClient();
Pagamento pagamento = new Pagamento(getTotal());
String target = "http://book-payment.herokuapp.com/payment";
Entity<Pagamento> json = Entity.json(pagamento);
WebTarget webTarget = client.target(target);
```

para que possamos fazer a ligação do alvo (*target*) com o Json, precisamos de uma requisição (*request*):

```
Client client = ClientBuilder.newClient();
Pagamento pagamento = new Pagamento(getTotal());
String target = "http://book-payment.herokuapp.com/payment";
Entity<Pagamento> json = Entity.json(pagamento);
WebTarget webTarget = client.target(target);
Builder request = webTarget.request();
request.post(json, String.class);
```

Ok, a primeira parte já foi. Criemos, enfim, a Classe "Pagamento", dentro e Models. A única coisa que ela precisa ter, de fato, é o `value` que estamos fazendo na requisição:

```
package br.com.casadocodigo.loja.models;

import java.math.BigDecimal;

public class Pagamento {

    private BigDecimal value;
}
```

E os getters e setters:

```
public class Pagamento {

    private BigDecimal value;

    public BigDecimal getValue() {
        return value;
    }

    public void setValue(BigDecimal value) {
        this.value = value;
    }
}
```

Podemos gerar também um construtor:

```
public class Pagamento {

    private BigDecimal value;

    public Pagamento(BigDecimal value) {
        this.value = value;
    }

    //...
}
```

Lembra que lá em cima, ao fazermos a requisição, recebemos o status 201 e uma mensagem "Pagamento efetuado com sucesso". Podemos mexê-la pelo código também. O ´request.post `retorna uma String como sendo` response`. Vamos também pedir para imprimir:

```
//...
Builder request = webtarget.request();
String response = request.post(json, String.class);
System.out.println(response);
```

Como sempre, selecionamos um livro e finalizamos a compra. E no console podemos ver que além do Json ter sido gerado e os dados inseridos, aparece a mesma mensagem do cliente DHC:

```
[{"titulo":"SEO Prático","preco":59.00,"quantidade":1,"total":59.00}]
Hibernate:
    insert
    into
        Usuario
        (email, nome, senha)
    values
        (?, ? ,?)
Hibernate:
    insert
    into
        Compra
        (itens, usuario_id)
    values
        (?, ?)
Pagamento efetuado com sucesso
```