02

Criando recursos e o POST

Até agora o meu cliente acessou uma URI no meu servidor para pedir informações sobre um recurso e o servidor devolveu uma representação XML, Json, seja lá o que for, deste recurso. Mas e se eu quero criar um carrinho?

Ao invés de buscar informações, com um GET, eu quero criar informações, criar um carrinho. Ao invés de acessar um GET, eu quero usar o verbo, o método POST, o POST que vai criar um recurso no meu servidor. Quando falamos de APIs HTTP com REST, costumamos utilizar o método POST para criar recursos.

Isso significa que dentro do nosso `CarrinhoResource` teremos um novo método, o `adiciona` que não será GET e sim POST:

```
@POST
public String adiciona() {

}
```

O POST permite que eu envie uma representação do cliente para o servidor: eu envio como cliente a representação do carrinho para o servidor, é o caminho contrário do que fazíamos até agora. Antes o cliente pedia os dados e o servidor devolvia a representação de um carrinho, agora o cliente envia a requisição com os dados, com a representação do carrinho, e o servidor confirma a criação do carrinho.

Criarei qual carrinho? O 3? O 15? Estranho... qual URI devo utilizar para criar *qualquer* carrinho? Quero criar um carrinho qualquer, então farei o POST para `/carrinhos` . Como é só `/carrinhos` , não preciso anotar o método.

Preciso dizer que a inclusão foi um sucesso, então retornamos essa informação:

```
@POST
@Produces(MediaType.APPLICATION_XML)
public String adiciona() {
    return "<status>sucesso</status>";
}
```

Tem algo de estranho aqui, um XML que devolve o status? Veremos adiante como podemos melhorar esse código. Mas preciso receber o conteúdo que quero deserializar em um `Carrinho` para poder inserir no meu banco. Como recebo dados em um método? Através de um parâmetro, então adicionamos o parâmetro:

```
public String adiciona(String conteudo) {
    return "<status>sucesso</status>";
}
```

E precisamos deserializar o carrinho:

```
public String adiciona(String conteudo) {
    Carrinho carrinho = (Carrinho) new XStream().fromXML(conteudo);
    return "<status>sucesso</status>";
}
```

Legal, tenho o meu carrinho, mas quero adicionar ele no meu banco de dados... para isso crio o `CarrinhoDAO` e mando adicionar:

```
@POST
@Produces(MediaType.APPLICATION_XML)
public String adiciona(String conteudo) {
    Carrinho carrinho = (Carrinho) new XStream().fromXML(conteudo);
    new CarrinhoDAO().adiciona(carrinho);
    return "<status>sucesso</status>";
}
```

Poderíamos usar aqui qualquer biblioteca de deserialização, assim como qualquer ferramenta de persistência ao invés do nosso DAO que simula o banco em memória. Agora gostaria de testar nosso servidor, como fazer um POST no navegador? Será chato ficar criando algum tipo de ferramenta no navegador para testar o POST, portanto vamos usar a linha de comando. Em um terminal linux, mac ou windows utilizamos o programa `curl` para executar requisições web. Se você não o possui basta instalar com seu programa de package manager, ou no caso do windows usar o site oficial do curl http://curl.haxx.se/download.html (http://curl.haxx.se/download.html)

Para acessar a uri que desejamos, buscando o primeiro carrinho:

```
curl http://localhost:8080/carrinhos/1
```

Legal, ele traz o xml para nós:

```
<br.com.alura.loja.modelo.Carrinho>
  <produtos>
    <br.com.alura.loja.modelo.Produto>
      <preco>4000.0</preco>
      <id>6237</id>
      <nome>Videogame 4</nome>
      <quantidade>1</quantidade>
    </br.com.alura.loja.modelo.Produto>
    <br.com.alura.loja.modelo.Produto>
      <preco>60.0</preco>
      <id>3467</id>
      <nome>Jogo de esporte</nome>
      <quantidade>2</quantidade>
    </br.com.alura.loja.modelo.Produto>
  </produtos>
  <rua>Rua Vergueiro 3185, 8 andar</rua>
  <cidade>S?o Paulo</cidade>
  <id>1</id>
</br.com.alura.loja.modelo.Carrinho>
```

O curl é o grande amigo de APIs HTTP Rest para fazermos testes manuais. Para fazer um POST ao invés de um GET, devemos enviar um xml para o servidor e dizer ao curl que devemos fazer um POST. Dentro de um editor de texto eu digito o xml que é um carrinho novo, um que possui somente um videogame:

```
<br.com.alura.loja.modelo.Carrinho>
  <produtos>
    <br.com.alura.loja.modelo.Produto>
      <preco>4000.0</preco>
      <id>6237</id>
      <nome>Videogame 4</nome>
      <quantidade>1</quantidade>
    </br.com.alura.loja.modelo.Produto>
  </produtos>
  <rua>Rua Vergueiro 3185, 8 andar</rua>
  <cidade>São Paulo</cidade>
  <id>1</id>
</br.com.alura.loja.modelo.Carrinho>
```

Devo copiar tudo isso sem as quebras de linha:

```
<br.com.alura.loja.modelo.Carrinho>  <produtos>    <br.com.alura.loja.modelo.Produto>      <prec
```

E é isso tudo que quero enviar para o servidor:

```
curl -d "<br.com.alura.loja.modelo.Carrinho>  <produtos>    <br.com.alura.loja.modelo.Produto>
```

E o resultado é de sucesso:

```
<status>sucesso</status>
```

Será que existe o carrinho número 2? Testamos o curl para o carrinho 2:

```
curl http://localhost:8080/carrinhos/2
```

E o resultado, com o carrinho no nosso servidor:

```
<br.com.alura.loja.modelo.Carrinho>
  <produtos>
    <br.com.alura.loja.modelo.Produto>
      <preco>4000.0</preco>
      <id>6237</id>
      <nome>Videogame 4</nome>
      <quantidade>1</quantidade>
    </br.com.alura.loja.modelo.Produto>
  </produtos>
  <rua>Rua Vergueiro 3185, 8 andar</rua>
  <cidade>São Paulo</cidade>
  <id>2</id>
</br.com.alura.loja.modelo.Carrinho>
```

Para darmos o suporte a criação de recursos foi só criarmos um método no Resource que suportasse o POST, recebendo como parâmetro do tipo `String` o conteúdo que o cliente enviará. Podemos deserializar usando qualquer biblioteca e fazer o que quiser com este conteúdo, no nosso caso salvamos o carrinho no banco.

Claro, queremos garantir que nosso cliente Java (a classe de teste) seja capaz de fazer tudo isso:

```
Client client = ClientBuilder.newClient();
        WebTarget target = client.target("http://localhost:8080");
```

Precisamos criar um carrinho e transforma-lo em XML para realizar o post:

```
Carrinho carrinho = new Carrinho();
        carrinho.adiciona(new Produto(314L, "Tablet", 999, 1));
        carrinho.setRua("Rua Vergueiro");
        carrinho.setCidade("Sao Paulo");
        String xml = carrinho.toXML();
```

Agora que temos o XML e sabemos que o media type que enviaremos é `application/xml` , precisamos representar isso de alguma maneira. Utilizaremos a classe `Entity` do próprio JAX-RS, para criar tal representação - o conteúdo e o media type - bastando importá-la em nosso projeto com o CTRL+SHIFT+O:

```
        Entity<String> entity = Entity.entity(xml, MediaType.APPLICATION_XML);

        Response response = target.path("/carrinhos").request().post(entity);
        Assert.assertEquals("<status>sucesso</status>", response.readEntity(String.class));
```

A `Entity` é utilizada para representar o que será enviado.