02

Criando um cadastro de produtos

Para que nossa aplicação de controle de estoque de produtos funcione como esperamos precisamos adicionar mais alguns comportamentos, como por exemplo a adição de produtos, remoção, entre outras operações importantes pra este dominio.

Vamos adicionar esses comportamentos em nossa classe controladora de `Produtos` , a `ProdutoController` !

Por sinal, é interessante saber que é uma prática comum separar os comportamentos de nossas regras de negócio em `controller` s específicos para cada recurso que estamos manipulando. Por exemplo, em nosso `ProdutoController` eu adiciono todos os comportamentos relacionados ao meu modelo `Produto` . Mas e se eu for adicionar um formulário de cadastro de usuários, faria sentido adicionar esse comportamento na mesma classe `ProdutoController` ? O código até funcionaria, mas para que fique melhor organizado a melhor opção seria criar uma classe `UsuarioController` que isola as operações com o modelo `Usuario` .

Lembrando que para que uma classe seja um `Controller` do VRaptor, tudo que você precisa fazer é adicionar a anotação `@Controller` . Por convenção, também adicionamos o sufixo Controller ao nome da classe.

Criando um cadastro de produtos

Para criar a funcionalidade de cadastro de produtos, primeiro precisamos fornecer ao usuário uma página com o formulário que deverá ser preenchido com as informações necessárias para esse cadastro.

Então vamos começar criando a página `formulario.jsp` dentro da pasta `/WEB-INF/jsp/produto/` , com três `input` s para o usuário adicionar o `nome` , `valor` e `quantidade` do produto que será criado:

```
<html>
    <body>
        <form action="">
            Nome: <input type="text" />
            Valor: <input type="text" />
            Quantidade: <input type="text" />
           <input type="submit" value="Adicionar" />
        </form>
    </body>
</html>
```

Para que o usuário possa acessar esse formulário, precisamos adicionar esse mapeamento na classe controladora de nossos produtos. Quero que ao acessar a URL `/produto/formulario` o VRaptor redirecione o pedido do usuário para esse `formulario.jsp` que acabamos de criar. Para isso vamos adicionar o método `formulario()` com a anotação `@Path` indicando o caminho.

```
@Path("/produto/formulario")
public void formulario(){

}
```

Apenas lembrando algumas convenções, é muito importante que esse método se chame `formulario`, pois assim o VRaptor irá procurar pelo arquivo `formulario.jsp`, e como já vimos, esse arquivo precisa estar dentro da pasta `/WEB-INF/jsp/produto/`.

Estamos quase lá, só precisamos agora criar uma ação para o nosso formulário, para que alguma lógica de nosso controller seja utilizada quando o usuário clicar em `Adicionar`. Vamos adicionar um novo método na classe `ProdutoController`, chamado `adiciona`:

```
@Path("/produto/adiciona")
public void adiciona() {
    EntityManager em = JPAUtil.criaEntityManager();
    em.getTransaction().begin();
    ProdutoDao dao = new ProdutoDao(em);
    dao.adiciona(??); // preciso de um produto aqui!!!
    em.getTransaction().commit();
}
```

Repare que agora, além de criar um `EntityManager`, precisamos abrir uma transação e commitar ao final do processo. Operação como adição e remoção de produtos sempre precisam estar envolvidas em uma transação, para que só seja concluída caso o proceso todo tenha acontecido com sucesso. Logo veremos mais a respeito e melhorar esse processo de trabalhar com as transações do JPA.

Vamos adicionar essa nova ação no campo *action* do nosso formulário:

```
<html>
    <body>
        <form action="<c:url value='/produto/adiciona'/>">
            Nome: <input type="text" />
            Valor: <input type="text" />
            Quantidade: <input type="text" />
           <input type="submit" value="Adicionar" />
        </form>
    </body>
</html>
```

Repare que usamos o `c:url` na `action` de nosso formulário. Essa `taglib` tem como função buscar o caminho relativo e é uma prática comum utilizá-la em links e endereços.

Recebendo parâmetros da Requisição

Mas como podemos receber essas informações do formulário em nosso método `adiciona`? Isso é muito simples com VRaptor! Poderíamos começar adicionando um **name** para cada campo do nosso formulário, e adicionar parâmetros com esses mesmos nomes em nosso método adiciona. Por exemplo, vamos adicionar os *names* como **nome**, **valor** e **quantidade**:

```
<html>
    <body>
        <form action="<c:url value='/produto/adiciona'/>">
            Nome: <input type="text" name="nome" />
            Valor: <input type="text" name="valor" />
```

```
            Quantidade: <input type="text" name="quantidade" />
            <input type="submit" value="Adicionar" />
        </form>
    </body>
</html>
```

Agora em nosso método adiciona, basta adicionar os parâmetros com os mesmos nomes que demos no atributo *name* dos campos:

```
@Path("/produto/adiciona")
public void adiciona(String nome, double valor, int quantidade) {
    // restante do código omitido
}
```

Agora, tudo que precisamos fazer é transformar isso tudo em um `Produto`. Podemos criar esse objeto e *setar* os valores:

```
@Path("/produto/adiciona")
public void adiciona(String nome, double valor, int quantidade) {
    Produto produto = new Produto();
    produto.setNome(nome);
    produto.setValor(valor);
    produto.setQuantidade(quantidade);
    EntityManager em = JPAUtil.criaEntityManager();
    em.getTransaction().begin();
    ProdutoDao dao = new ProdutoDao(em);
    dao.adiciona(produto);
    em.getTransaction().commit();
}
```

Achou trabalhoso? Então podemos simplificar mais! Repare que o **nome**, **valor** e **quantidade** são representados pela classe `Produto` em nosso sistema, podemos apenas pedir ao VRaptor um `Produto` com as informações que passamos como parâmetro na requisição! Para isso basta adicionar o parâmetro `Produto` no método adiciona:

```
@Path("/produto/adiciona")
public void adiciona(Produto produto) {
    EntityManager em = JPAUtil.criaEntityManager();
    em.getTransaction().begin();
    ProdutoDao dao = new ProdutoDao(em);
    dao.adiciona(produto);
    em.getTransaction().commit();
}
```

Agora só precisamos modificar o atributo **name** nos `input`s do nosso formulário, para `produto.nomeDoCampo`:

```
<html>
    <body>
        <form action="<c:url value="/produto/adiciona"/>">
            Nome: <input type="text" name="produto.nome"/>
            Valor: <input type="text" name="produto.valor"/>
            Quantidade: <input type="text" name="produto.quantidade"/>
            <input type="submit" value="Adicionar" />
```

```
        </form>
    </body>
</html>
```

Essa é outra convenção do VRaptor. O nome do atributo que passamos no método `adiciona` é **produto**, portanto no `name` dos `input` s HTML eu passo o valor `produto.nomeDoCampo` . Dessa forma, quando pedimos um atributo chamado `produto` como parâmetro ao VRaptor, ele procura na requisição os valores de texto que começam com `produto.algumValor` e depois tenta preencher esse valor usando o seu `setter` .

Por exemplo, quando eu uso o *name* `produto.nome` o VRaptor chama o método `setNome(String nome)` passando o texto que eu digitei no campo de texto do HTML como valor. Basta ir separando por pontos que o VRaptor sabe que deve setar esses valores, e caso não encontre o campo, apenas ignora e não preenche nada.

Concluindo o cadastro de produtos

Ao invocar o método `adiciona(Produto produto)` do nosso controller, a sua lógica será executada e o VRaptor vai tentar redirecionar a requisição do usuário para a página `adiciona.jsp` ! Portanto, para concluir nosso cadastro precisamos criar essa página:

```
<html>
  <body>
    Produto adicionado com sucesso!
  </body>
</html>
```

Lembrando, para que tudo funcione como esperado esse arquivo precisa estar no diretório `/WEB-INF/jsp/produto/` .

Vamos testar essa nova funcionalidade! Após restartar o servidor, vamos abrir o formulário de produtos na url http://localhost:8080/vraptor-produtos/produto/formulario (http://localhost:8080/vraptor-produtos/produto/formulario) e adicionar um novo produto!

Estilizando o formulário

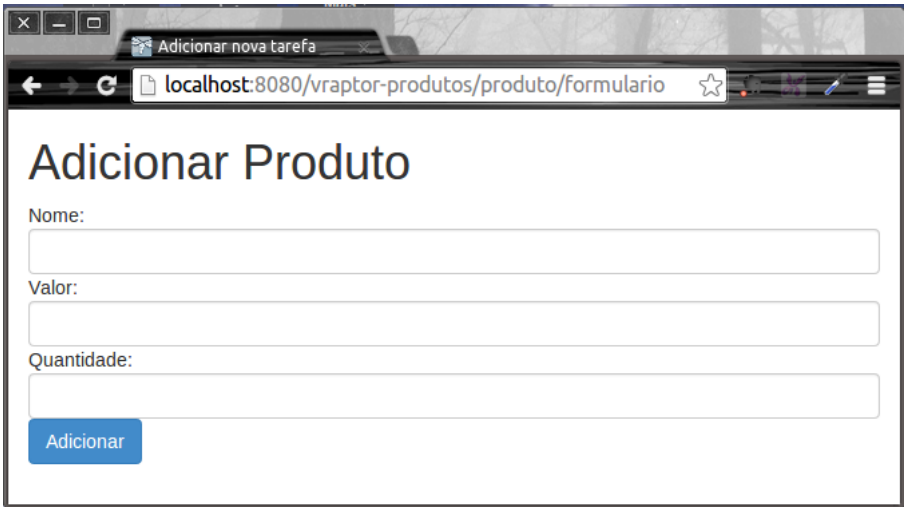
Vamos adicionar as classes e estrutura do twiter bootstrap para deixar o nosso formulário com uma aparência melhor. Para isso, basta adicionar suas classes seletoras. No final, nosso jsp deve ficar assim:

```
<div class="container">

  <h1>Adicionar Produto</h1>

  <form action="<c:url value='/produto/adiciona'/>" >
    Nome:
    <input class="form-control" type="text" name="produto.nome" value="${produto.nome}"/>
    Valor:
    <input class="form-control" type="text" name="produto.valor" value="${produto.valor}"/>
    Quantidade:
    <input class="form-control" type="text" name="produto.quantidade" value="${produto.quantida
    <button type="submit" class="btn btn-primary">Adicionar</button>
  </form>
```

```
</div>
```


Adicionar nova tarefa
localhost:8080/vraptor-produtos/produto/formulario
Adicionar Produto
Nome:
Valor:
Quantidade:
Adicionar