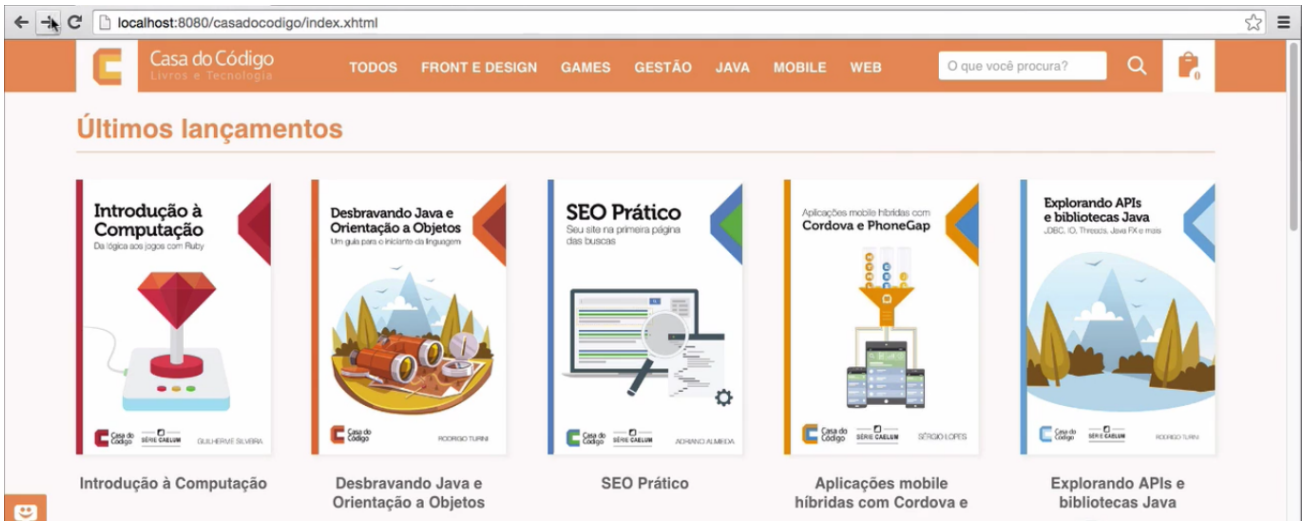
07

Criando o Detalhe do Livro

Transcrição

Nossa **HOME** está funcionando, mas ainda estamos deixando que os links envie nossos usuários para o site oficial da Casa do Código (https://www.casadocodigo.com.br/). O nosso desejo é fazer com que o usuário veja o livro dentro do nosso próprio sistema, sem ser direcionado para o site da CDC.



Vamos começar alterando o arquivo `index.xhtml` para que os links sejam locais. No `<ui:repeat>` dos *Ultimos Lançamentos*, veremos que o atributo `href` da tag `<a>` está apontando para `https://casadocodigo.com.br/....` . Removemos esse valor e o substituimos pelo nosso endereço local. Para isso, usaremos o `#{request.contextPath}` .

```
<a href="#{request.contextPath}/livro-detalhe.xhtml?id=#{livro.id}" class="livroNaVitrine-link"
                    title="#{livro.titulo}">
```

Mas ainda não fomos a lugar nenhum, pois ainda falta dizer qual é a página que exibe o detalhe do livro. No nosso caso, usaremos uma página chamada `livro-detalhe.xhtml` e além disso, precisamos informar a essa página, qual é o livro que queremos exibir. Aproveitaremos o mesmo link passando um parâmetro para o livro-detalhe que será o `ID` , que podemos obter através do livro `#{livro.id}` . Assim, nossa *URL* completa ficará assim:

```
#{request.contextPath}/livro-detalhe.xhtml?id=#{livro.id}
```

Como na parte dos *Demais Livros* será a mesma *URL*, basta copiar e dentro do `<ui:repeat>` dos *Demais Livros* e dentro da tag `<a>` alterar também o `href` para o código acima.

Estamos enviando o usuário para a página `livro-detalhe.xhtml` , no entanto essa página ainda não existe. Vamos criar nossa página de detalhe do livro.

Para não perdermos tempo digitando tudo que existe no detalhe do livro da **CDC**, baixe o *ZIP* no link a seguir, e vamos importar esse *ZIP* para dentro do nosso projeto como já fizemos anteriormente.

https://s3.amazonaws.com/caelum-online-public/java-ee-webapp/casadocodigo-javaee-detalhe.zip (https://s3.amazonaws.com/caelum-online-public/java-ee-webapp/casadocodigo-javaee-detalhe.zip)

Após baixar o arquivo acima vamos importar para o nosso projeto.

Ainda no Eclipse, clique com o botão direito em cima do projeto e selecione a opção `Import`. Vá em `General` e depois em `Archive File` e clique em `Next`. Nessa tela, clique em `Browse...` que fica logo a frente do campo `From archive file:` e navegue até o local onde você baixou o `ZIP` do livro-detalhe da CDC, e clique em OK. Você deve estar vendo no lado direito o arquivo `livro-detalhe.xhtml`. Por fim, clique em `Finish` para finalizar o processo.

Neste momento você já deve estar vendo o arquivo `livro-detalhe.xhtml`.

Já temos a nossa tela, porém se você rodar o projeto agora vai perceber um erro informando que o `Livro` não tem o ID. Na verdade nós temos o atributo `id`, o que não temos é o *get* e *set* desse id. Peça mais uma vez para o Eclipse gerar o get e set através do atalho `Ctrl + 3` e digite `ggas` para gerar os getters and setters. Selecione o `id` e clique em **OK**.

Agora que temos o get e set do ID, podemos rodar novamente nossa aplicação. Não esqueça-se de realizar um *Full Publish*.

Ainda assim, quando clicamos em alguma tela, percebemos que o livro exibido não é o livro que escolhemos. O que está acontecendo é que a imagem e título estão fixos no código ainda, sendo assim, temos que realizar a alteração para que essa informação venha do banco de dados.

Nesse ponto, precisamos analisar um ponto. O `id` que estamos enviando para o detalhe, está vindo através da *URL*, o que significa que estamos usando o método **GET** nessa requisição. Porém o **JSF** foi feito para trabalhar com **POST**, até versões anteriores era quase impossível usar **GET**. Mas para nossa sorte, com **JSF** 2.2 já é possível usar **GET** e agora podemos pegar o `id` pelo GET ao invés de fazer gambiarras para usar POST como se fosse GET como era feito anteriormente.

Como sempre, precisamos de um *ManagedBean* que representará nosso livro-detalhe.xhtml, e por isso, crie uma nova classe chamada `LivroDetalheBean` no pacote `br.com.casadocodigo.loja.beans`. Já anote esse novo Bean com `@Model` e também vamos colocar um atributo `LivroDao` injetado com `@Inject`.

Ainda dentro do nosso novo Bean, vamos criar outro método que servirá para carregar de fato o detalhe do livro específico pelo `id`. Crie o método `carregaDetalhe` como `public void` e sem receber nenhum parâmetro. Dentro do método, vamos usar o `LivroDao` para carregar o `Livro` pelo ID que recebemos, podemos fazer a seguinte chamada: `this.livro = dao.buscarPorId(id);`.

Neste momento, seu Eclipse deve estar reclamando de um monte de coisas. Primeiro, o atributo `this.livro` não existe, então vamos criá-lo. Na área de atributos, faça a declaração do livro.

```
private Livro livro;
```

Outra coisa que deve estar dando erro é o `id`. Já vamos criar esse atributo também:

```
private Integer id;
```

Peça para o Eclipse gerar os *getters and setters* do `id` e do `livro` como já ensinamos aqui.

E por fim, o método `buscarPorId` também não existe dentro do `LivroDao`. Pressione `Ctrl + 1` no Eclipse e escolha a opção `Create method ...`. O Eclipse já nos dá a estrutura necessária e faremos o código bem simples para busar pelo ID com o `EntityManager`.

```
public Livro buscarPorId(Integer id) {
    return manager.find(Livro.class, id);
}
```

Quando o clique for realizado na *Home* para nossa página de `detalhe-livro.xhtml`, o **JSF** fará a chamada da página e dentro da página começará a carregar os componentes. Pensando sobre onde devemos pegar o `id` para carregar o livro, fica claro que precisaremos fazer isso na página e a página deve chamar o método do Bean. E se percebermos, isso é o que sempre temos feito, a página chama do Bean.

No **JSF**, a ordem do **MVC** acaba ficando um pouco invertida realmente. Ao invés de irmos direto para o *Controller* (nosso Bean), vamos para a *View* (xhtml) e a *View* chama o *Controller*. Isso é mais uma particularidade do JSF.

Por isso, precisamos fazer uso da tag `<f:metadata>` que veio para resolver esse problema de carregar parâmetros que vem pelo **GET**. Dentro do `<f:metadata>` precisamos pegar o `id` que vem pela URL, faremos isso com o `<f:viewParam>` que possui dois atributos. Para especificar onde colocaremos o valor obtido, o `value` será:

```
value="livroDetalheBean.id"
```

Vamos incluir também o `name` - o nome do parâmetro que recebemos: `name="id"`.

O valor já temos. Mas como chamaremos o método `carregaDetalhe()`? O `<f:metadata>` também possui uma forma para isso, que é com a tag `<f:viewAction>` onde passamos o atributo `action` com o caminho do método:

```
action="#{livroDetalheBean.carregaDetalhe()}"
```

A tag `<f:metadata>` completa, ficará assim:

```
<f:metadata>
    <f:viewParam value="#{livroDetalheBean.id}" name="id" />
    <f:viewAction action="#{livroDetalheBean.carregaDetalhe()}" />
</f:metadata>
```

Uma vez que nosso livro já foi carregado, agora podemos fazer uso dele em toda nossa página. Vamos começar alterando a tag `<title>` logo abaixo para exibir o título realmente do livro:

```
<title>#{livroDetalheBean.livro.titulo} - Casa do Código</title>
```

No arquivo `detalhe-livro.xhtml`, vamos fazer alterações dentro da `div` de `cabecalhoProdutoLivro-tituloEAutor`. No `span` de `cabecalhoProdutoLivro-titulo-principal` (próximo da linha `272`), modificaremos o título para `#{livroDetalheBean.livro.titulo}`.

Mais abaixo temos um `<span>` com os autores. Como podemos ter mais de um autor, vamos usar o `<ui:repeat>`. Ficando assim:

```
<ui:repeat value="#{livroDetalheBean.livro.autores}" var="autor">
    #{autor.nome},
</ui:repeat>
```

Com as alterações, o trecho do código ficará assim:

```
<div class="cabecalhoProdutoLivro-tituloEAutor">
        <h1 itemprop="name" class="cabecalhoEProdutoLivro-titulo">
                <span class="cabecalhoEProdutoLivro-titulo-principal" role="presentation">
                        #{livroDetalheBean.livro.titulo}
                </span>
                <span class="cabecalhoProdutoLivro-nomeAutor">
                        <ui:repeat value="#{livroDetalheBean.livro.autores}" var="autor">
                            #{autor.nome},
                        </ui:repeat>
                </span>
</div>
```

Em seguida, queremos alterar o preço do livro de "Java SE 8".

Mais abaixo, vamos alterar o preço do nosso e-book para exibir o preço cadastrado. Dentro da tag `<span>` de `"adicionarAoCarrinho-preco-valor"`, mudaremos o valor fixo de `R$39,90` para o seguinte valor:

```
R$ #{livroDetalheBean.livro.preco}`:
```

Com as alterações, o código ficou assim:

```
<p class="adicionarAoCarrinho-preco">
                <small class="adiconarAoCarrinho-preco-promocao">
                        <del class="adicionarAoCarrinho-preco-promocao-valor">R$1099,90</del> p
                </small>
        <span class="adicionarAoCarrinho-preco-valor" itemprop="price">
            R$ #{livroDetalheBean.livro.preco}
```

```
        </span>
    </p>
```

Após descermos mais um pouco, encontraremos a `descricao`. Substituiremos o conteúdo pela descrição real do livro. No `<p class="infoSection-texto">` logo a abaixo, apague todo o conteúdo atual, substituindo pelo código abaixo:

```
<p class="infoSection-texto">
    #{livroDetalheBean.livro.descricao}
</p>
```

E por fim, temos o número de páginas. Nós queremos exibir o número que gravamos no livro. Para isso, vamos procurar pela linha:

```
<dd class="infosAdicionaisDoLivro-info-valor" itemprop="numberOfPages">
```

Iremos substituir o número fixo por:

```
#{livroDetalheBean.livro.numeroPaginas} páginas
```

O código ficará assim:

```
<dd class="infosAdicionaisDoLivro-info-valor" itemprop="numberOfPages">
    #{livroDetalheBean.livro.numeroPaginas} páginas
</dd>
```

Depois de realizar todas as alterações, faremos um *Full Publish* novamente. Veremos, então, como ficou nossa tela de detalhe do livro.

Ao clicar em algum livro na *Home*, percebemos que apenas o título da livro é exibido. Por que isso aconteceu?