04

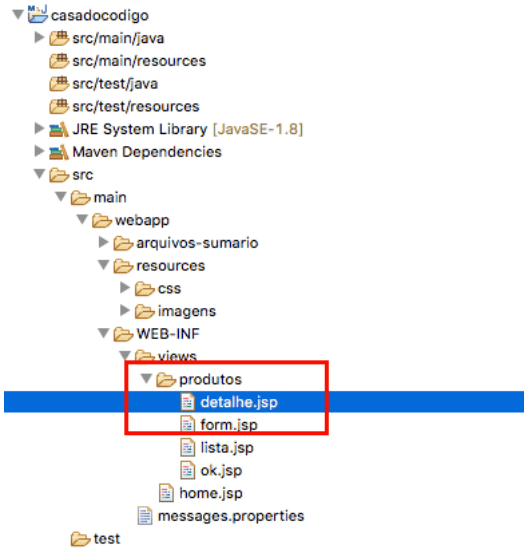
Mãos na massa: URLs amigáveis

Chegou a hora de você pôr em prática o que foi visto na aula. Para isso, execute os passos listados abaixo.

1) Se você ainda não baixou, faça agora o download do código fonte da página JSP aqui (https://s3.amazonaws.com/caelum-online-public/spring-mvc-1-criando-aplicacoes-web/springmvc-arquivos-extras-aula10.zip). Após baixar, extraia o ZIP. Você encontrará dois arquivos JSP e uma pasta com os recursos CSS:

- O arquivo JSP limpo mas **sem *expression languages*** (**detalhe-limpo.jsp**).
- O arquivo JSP finalizado com **todas as alterações** já aplicadas nessa aula (**detalhe-pronto.jsp**).
- Uma pasta **resources**, com os arquivos CSS e imagem da Casa do Código para adicionar no projeto localmente.

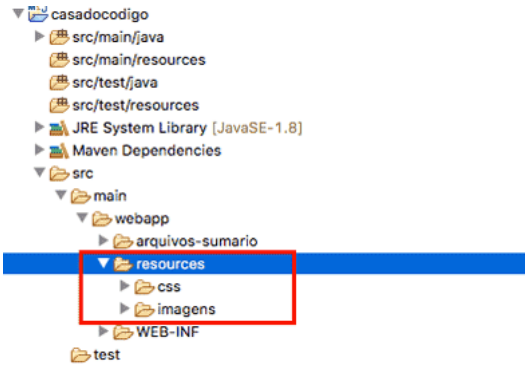
2) Escolha qual JSP usar, renomeie-o para **detalhe.jsp** e copie-o para a pasta **src/main/webapp/WEB-INF/views/produtos** do seu projeto:



3) Se você usou o arquivo **detalhe-limpo.jsp**, ainda é preciso incluir as *expression languages* nos lugares indicados. Lembre-se de alterar os locais onde deseja que as informações de *título*, *id*, *descrição*, *número de páginas*, *sumário*, *data de lançamento*, além do `<c:forEach />` para listar os preços do livro:

```
<form action="/carrinho/add" method="post" class="container">
    <input type="hidden" value="${produto.id}" name="produtoId" >
    <ul id="variants" class="clearfix">
        <c:forEach items="${produto.precos}" var="preco">
            <li class="buy-option">
                <input type="radio" name="tipoPreco" class="variant-radio"
                    id="tipoPreco" value="${preco.tipo}" checked="checked" />
                <label class="variant-label" >${preco.tipo}</label>
                <small class="compare-at-price">R$ 39,90</small>
                <p class="variant-price">${preco.valor}</p>
            </li>
        </c:forEach>
    </ul>
    <button type="submit" class="submit-image icon-basket-alt"
        title="Compre Agora ${produto.titulo}"></button>
</form>
```

4) Agora, copie toda a pasta **resources** para a pasta **src/main/webapp** do seu projeto:



5) Por padrão, o Spring MVC nega o acesso à pasta **resources**. Consequentemente, o Tomcat não pode carregar os arquivos CSS (e a página fica sem design). Para liberar o acesso, é preciso fazer duas alterações na classe `AppWebConfiguration`:

- A classe deve estender a classe `WebMvcConfigurerAdapter`:

```
@EnableWebMvc
@ComponentScan(basePackageClasses= {HomeController.class, ProdutoDAO.class,
    FileSaver.class})
public class AppWebConfiguration extends WebMvcConfigurerAdapter {

    // restante do código omitido

}
```

- A classe deve implementar o método `configureDefaultServletHandling` para liberar o acesso:

```
@Override
public void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer) {
    configurer.enable();
}
```

6) Ajuste o código Java começando pelo `ProdutoDAO`. Escreva um novo método, chamado `find`, que recebe o `id` do produto por parâmetro e retorna o produto com todas as informações preenchidas, inclusive os preços. O método deve executar uma *query* planejada através do `EntityManager`, que vai realizar o `join` entre as tabelas `Produto` e `precos`:

```
public Produto find(Integer id) {
    return manager.createQuery("select distinct(p) from Produto p " +
        "join fetch p.precos precos where p.id = :id", Produto.class)
            .setParameter("id", id).getSingleResult();
}
```

7) Ajuste o *controller*, para carregar a JSP com os dados do produto. Primeiramente, crie o método `detalhe`, que recebe o `id` do produto por parâmetro, retorna para a JSP de detalhe e carrega o objeto `ModelAndView` com um produto:

```
@RequestMapping("/detalhe")
public ModelAndView detalhe(Integer id){
```

```
    ModelAndView modelAndView = new ModelAndView("produtos/detalhe");
    Produto produto = produtoDao.find(id);
    modelAndView.addObject("produto", produto);

    return modelAndView;
}
```

8) O Spring já traz um sistema de URLs amigáveis, que são URLs que parecem texto e não ficam poluídas com `?` e `&`. Então edite o método `detalhe`, adicionando uma variável na rota e usando a anotação `@PathVariable` para mapear essa variável a um parâmetro:

```
@RequestMapping("/detalhe/{id}")
public ModelAndView detalhe(@PathVariable("id") Integer id){

    ModelAndView modelAndView = new ModelAndView("/produtos/detalhe");
    Produto produto = produtoDao.find(id);
    modelAndView.addObject("produto", produto);

    return modelAndView;
}
```

9) Por fim, adicione um link na listagem de produtos (**lista.jsp**) que irá te direcionar para a página de detalhe do produto. Para construir a URL, use o método `mvcUrl`. Além disso, invoque o método `arg` para adicionar um parâmetro à URL. Portanto, o link vai ficar com a seguinte estrutura:

```
<c:forEach items="${produtos}" var="produto">
    <tr>
        <td>
            <a href="${s:mvcUrl('PC#detalhe').arg(0,produto.id).build()}">
                ${produto.titulo}
            </a>
        </td>
        <td>${produto.descricao}</td>
        <td>${produto.paginas}</td>
    </tr>
</c:forEach>
```

Lembre-se que é preciso fazer o *import* da *taglib* para poder usar o `mvcUrl`:

```
<%@ taglib uri="http://www.springframework.org/tags" prefix="s"%>
```