04

Mãos na massa: Mais validações

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

1) Para adicionar mensagens de validação no projeto, crie o arquivo que define essas mensagens, conhecido como **messages.properties**. Ele segue um formato bem parecido com chave/valor. Por exemplo, caso você queira adicionar mensagens para campos que são obrigatórios, basta preencher a chave `field.required` com o valor **Campos obrigatórios**. Você também pode definir mensagens mais específicas. Então, na pasta **WEB-INF**, crie o arquivo **messages.properties** com o seguinte conteúdo, por exemplo:

```
field.required = Campo obrigatório
field.required.produto.titulo = O título é obrigatório
field.required.produto.paginas = Informe o número de páginas
field.required.produto.descricao = A descrição é obrigatória
typeMismatch = O tipo de dado é inválido
typeMismatch.produto.paginas = Digite um valor válido. Exemplo: "100"
```

2) Assim que o arquivo for salvo, clique nele com o botão direito do mouse e selecione ***Properties***. Em ***Text file encoding***, verifique se o *encoding* é **UTF-8**, caso não seja, altere-o.

3) Para o Spring encontrar o arquivo **messages.properties**, crie um novo método na classe `AppWebConfiguration` , que irá retornar o arquivo com algumas configurações. Crie um novo método como o exemplo abaixo:

```
@Bean
public MessageSource messageSource() {
    ReloadableResourceBundleMessageSource messageSource =
        new ReloadableResourceBundleMessageSource();

    messageSource.setBasename("/WEB-INF/messages");
    messageSource.setDefaultEncoding("UTF-8");
    messageSource.setCacheSeconds(1);

    return messageSource;
}
```

4) Para adicionar as mensagens no formulário **form.jsp**, adicione uma nova `taglib` :

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

5) Além disso, substitua a *tag* `<form>` para:

```
<form:form action="/casadocodigo/produtos" method="POST" commandName="produto">
```

Caso esteja usando dependências atualizadas e der o erro ***Unable to find setter method for attribute: [commandName]***, troque o `commandName` para `modelAttribute` .

6) Adicione as mensagens de erro para os campos de título, descrição e páginas:

```
<form:form action="/casadocodigo/produtos" method="POST" commandName="produto">
    <div>
        <label>Título</label>
        <input type="text" name="titulo">
        <form:errors path="titulo" />
    </div>
    <div>
        <label>Descrição</label>
        <textarea rows="10" cols="20" name="descricao"></textarea>
        <form:errors path="descricao" />
    </div>

    <div>
        <label>Páginas</label>
        <input type="text" name="paginas">
        <form:errors path="paginas" />
    </div>

    <c:forEach items="${tipos}" var="tipoPreco" varStatus="status">
        <div>
            <label>${tipoPreco}</label>
            <input type="text" name="precos[${status.index}].valor">
            <input type="hidden" name="precos[${status.index}].tipo" value="${tipoPreco}">
        </div>
    </c:forEach>
    <button type="submit">Cadastrar</button>
</form:form>
```

7) Por fim, para evitar de ficar mexendo na *action* do formulário, peça para o Spring preencher a URL para você. Para isso, adicione a `taglib` abaixo:

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

8) E use o método `mvcUrl("controller#metodo").build()` para criar a URL:

```
<form:form action="${s:mvcUrl('PC#gravar').build()}" method="POST" commandName="produto">
```

O *controller* é referenciado pelas suas letras maiúsculas, então `ProdutosController` vira `PC`.

9) Por fim, para que a URL possa ser construída de forma correta, separando os contextos, modifique o `@RequestMapping` do `ProdutosController` para `/produtos`:

```
@Controller
@RequestMapping("/produtos")
public class ProdutosController {

    // restante do código omitido

}
```