CETEJ35 - Java Web - JAVA_XXX (2024_01)

Validação de Dados

✓ **Feito:** Ver ✓ **Feito:** Gastar pelo menos 20 minutos na atividade **A fazer:** Passar pela atividade até o fim

Fecha: segunda-feira, 2 dez. 2024, 00:00

Nesta aula, vamos ver como usar recursos do *Bean Validation Framework* juntamente com o Spring Boot, Freemarker e Bootstrap para garantir que o usuário consiga visualizar os erros e corrigir os dados sempre que necessário.

EXIBINDO ERROS NA TELA

Nosso usuário espera ser avisado caso erros impeçam alguma operação de ser executada. Por isso, vamos usar o Bootstrap e Freemarker para mostrar esses erros na tela. Além disso, precisamos de um pequeno ajuste no `CidadeController` para enviar os erros para página Web.

Vamos começar pelo controlador. Abra a classe `CidadeController` e adicione o parâmetro `memoria`, do tipo `org.springframework.ui.Model`, no método `criar()`. Assim como já fizemos antes, esse objeto é responsável p carregar dados em uma solicitação – seja ela indo para a GUI ou vindo da GUI.

> Mais uma vez, observe que o método `alterar()` usa o método `criar()`. Ao mudar o número de parâmetros do método, seu editor/IDE deve reclamar um erro. Por isso, assim como fizemos antes, simplesmente mude a chamada do método `criar()`, no corpo de `alterar()`, para `criar(cidade, null, null)`.

```
32      @PostMapping("/criar")
33      public String criar(@Valid Cidade cidade, BindingResult validacao, Model memoria) {
```

Em seguida, em vez de imprimir os erros na console do sistema, vamos usar a mesma repetição para adicionar os erros como atributos da variável memoria.

```
35          if (validacao.hasErrors()) {
36              validacao
37                  .getFieldErrors()
38                  .forEach(error ->
39                          memoria.addAttribute(
40                              error.getField(),
41                              error.getDefaultMessage())
42                          );
```

Assim como antes, o método `getFieldErrors()`, na linha 37, retorna uma lista de erros caso eles existam (linha 35). A linha 38 faz uma iteração sobre esses erros, usando a variável error como referencia para cada erro encontrado durante a iteração. Em vez de imprimir o erro na console do sistema, usamos a variável memoria para guardar os erros como atributos. Observe que o método `getField()` retorna o atributo onde o erro ocorreu. Por exemplo, se o nome da cidade

não foi informado, então o `getField()` retorna nome. O `getDefaultMessage()` retorna a mensagem de erro definida para o erro ocorrido. Por exemplo, dizendo que o nome não foi informado.

> Lembre-se que as mensagens de erro foram definidas no arquivo `messages.properties`.

Por fim, precisamos adicionar a lista de cidades à `memoria` antes de redirecionar o usuário para a página `crud.ftl` novamente. Veja como ficou o método completo após essas mudanças.

```
32      @PostMapping("/criar")
33      public String criar(@Valid Cidade cidade, BindingResult validacao, Model memoria) {
34
35          if (validacao.hasErrors()) {
36              validacao
37                  .getFieldErrors()
38                  .forEach(error ->
39                          memoria.addAttribute(
40                              error.getField(),
41                              error.getDefaultMessage())
42                          );
43
44                  memoria.addAttribute("listaCidades", cidades);
45
46                  return ("/crud");
47          } else {
48              cidades.add(cidade);
49          }
50
51          return "redirect:/";
52      }
```

Agora, precisamos ajustar a página `crud.ftl` para que ela exiba as mensagens de erro caso existam. Vamos começar ajustando os atributos dos formulários de criação e alteração. Na tag `form`, precisamos adicionar o atributo novalidate para evitar que o formulário use as mensagens padrão do HTML. Também precisamos adicionar a classe `needs-validation` do Bootstrap. Essa classe especifica que esse formulário tem validações.

```
21              <#if cidadeAtual??>
22                  <form action="/alterar" method="POST" class="needs-validation" novalidate>
23                  <input type="hidden" name="nomeAtual" value="${(cidadeAtual.nome)!}"/>
24                  <input type="hidden" name="estadoAtual" value="${(cidadeAtual.estado)!}"/>
25              <#else>
26                  <form action="/criar" method="POST" class="needs-validation" novalidate>
27              </#if>
```

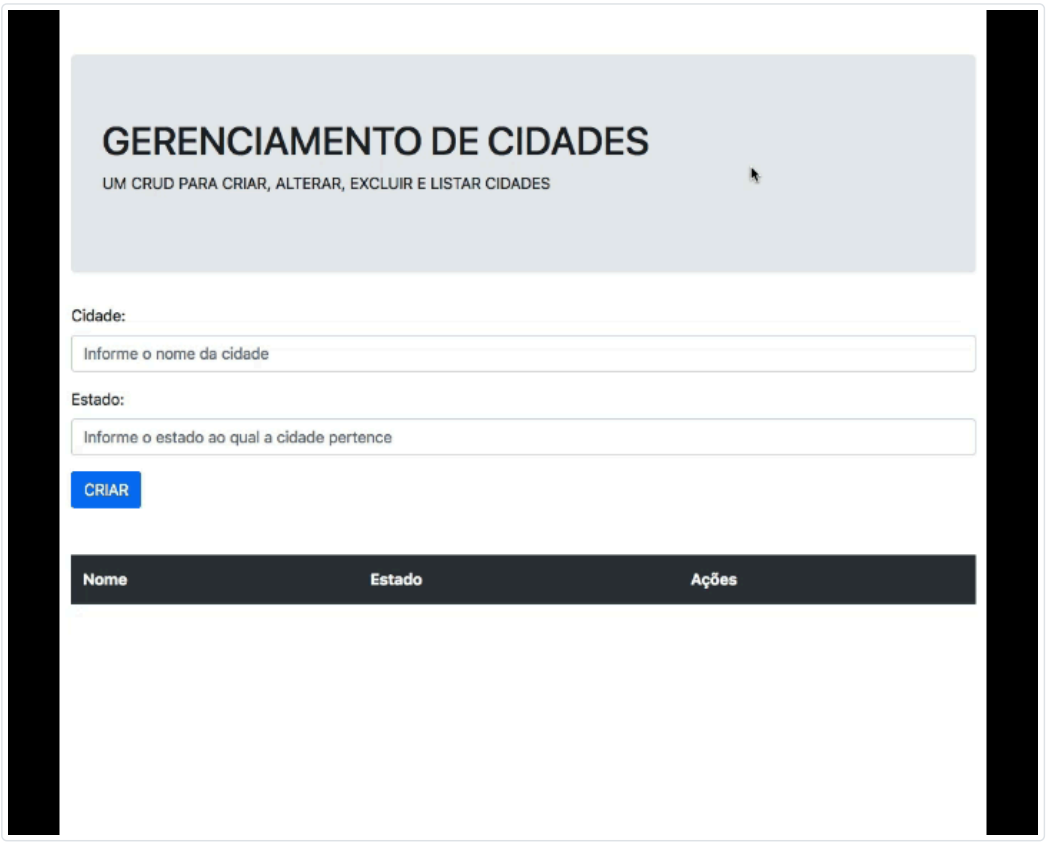
Em seguida, vamos criar dois elementos para exibir as mensagens de erro. Esses elementos estarão logo abaixo dos elementos de entrada de dados, conforme mostra a Figura a seguir.

```
29              <div class="form-group">
30                  <label for="nome">Cidade:</label>
31                  <input
32                      value="${(cidadeAtual.nome)!}"
33                      name="nome"
34                      type="text"
35                      class="form-control ${(nome??)?then('is-invalid', '')}"
36                      placeholder="Informe o nome da cidade"
37                      id="nome">
38
39                  <div class="invalid-feedback">
40                      ${nome!}
41                  </div>
42              </div>
43
44              <div class="form-group">
45                  <label for="estado">Estado:</label>
46                  <input
47                      value="${(cidadeAtual.estado)!}"
48                      name="estado"
49                      type="text"
50                      class="form-control ${(estado??)?then('is-invalid', '')}"
51                      placeholder="Informe o estado ao qual a cidade pertence"
52                      id="estado">
53
54                  <div class="invalid-feedback">
55                      ${estado!}
56                  </div>
57              </div>
```

Observe que os novos elementos usam uma formatação de erro definida pelo Bootstrap (`invalid-feedback`). Dentro desses elementos nós apresentamos a mensagem de erro acessando o atributo definido no controlador, desde que ele exista. Para isso, usamos a sintaxe do Freemarker (`${}`), o nome do atributo que colocaremos na variável `memoria` (`nome` e `estado`), e o marcador do Freemarker (`!`), que apenas apresenta um valor caso ele exista.

Outra alteração que já pode ser observada na Figura anterior está nas linhas 35 e 50. Além da classe `form-control`, já definida em aulas anteriores, adicionamos uma expressão que avalia a existência do atributo `nome` (linha 35) e `estado` (linha 50) na primeira parte. A segunda parte da expressão (`then`), adiciona a classe `is-invalid` caso nome ou estado existam na variável memoria. Isso é usado pelo Bootstrap para marcar a caixa de texto em vermelho, indicando que existe um erro.

Execute o código e tente enviar o formulário sem informar o valores. Você verá as mensagens de erro. Depois, tente colocar um valor, mas não outro. Também tente colocar espaços em branco e veja o que acontece.



O código desenvolvido nesta Seção está disponível no **Github**, na branch `semana04-40-validacao-web`.

Retroceder Avançar

◄ Verificação de aprendizado - Criando, Alterando e Excluindo

Seguir para...

Integração ►

Contate o suporte do site

Você acessou como RAFAEL ROCHA DA SILVA PROENCA (Sair)
CETEJ35 - Web (2024_01)

Tema
Adaptable
Boost
Clássico

Campus
Apucarana
Campo Mourão
Cornélio Procópio
Curitiba
Dois Vizinhos
Francisco Beltrão
Guarapuava
Londrina
Medianeira
Pato Branco
Ponta Grossa
Reitoria
Santa Helena

Toledo

UTFPR

Ajuda

Chat UTFPR

Calendário Acadêmico

Biblioteca

e-Mail

Nuvem (OwnCloud)

Produção Acadêmica

Secretaria Acadêmica

Sistemas Corporativos

Sistema Eletrônico de Informação - SEI

Suporte ao usuário

Criação de curso

Comunidade

Português - Brasil (pt_br)

Deutsch (de)

English (en)

Português - Brasil (pt_br)

Resumo de retenção de dados

Baixar o aplicativo móvel.

Dê um feedback sobre este software

Universidade Tecnológica Federal do Paraná - UTFPR

Suporte ao usuário