

CETEJ35 - Java Web - JAVA_XXX (2024_01)

[Meus cursos](#) / [CETEJ35 - Web \(2024_01\)](#) / [Semana 05: 30/09 a 06/10](#) / [Validação de Dados](#)

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.

VALIDANDO MÉTODO criar()

Apesar de termos inserido as anotações de validação, isso não é o suficiente para a validação acontecer de forma automática. Precisamos indicar aos métodos que eles precisam usar a validação quando receberem a classe **Cidade** como parâmetro de uma solicitação. Para fazer isso, adicione a anotação **javax.validation.Valid** antes do parâmetro **cidade**, na classe **CidadeController**.

```
32 @PostMapping("/criar")
33 public String criar(@Valid Cidade cidade) {
```



Isso fará com que o Spring Boot acione a validação no método **criar()** quando ele receber um objeto do tipo **Cidade**. A validação será realizada de acordo com as anotações na classe **Cidade**. Portanto, a nulidade e tamanho dos atributos **nome** e **estado** serão validadas.

Quando a **Cidade** for validada, os possíveis erros gerados são enviados para um objeto do tipo **org.springframework.validation.BindingResult**. Por isso, precisamos definir esse objeto como um parâmetro no método **criar()**, conforme mostra a Figura.

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 enquanto, simplesmente mude a chamada do método **criar()**, no corpo de **alterar()**, para **criar(cidade, null)**.

```
32 @PostMapping("/criar")
33 public String criar(@Valid Cidade cidade, BindingResult validacao) {
34
```

Agora, tudo que precisamos fazer é usar os métodos disponíveis em **org.springframework.validation.BindingResult** para verificar pela existência de erros. Em caso de erros, vamos imprimir os erros e as mensagens correspondentes na console do sistema. Observe que a criação de uma nova cidade agora está condicionada à não existência de problemas de validação.

```
32 @PostMapping("/criar")
33 public String criar(@Valid Cidade cidade, BindingResult validacao) {
34
35     if (validacao.hasErrors()) {
36         validacao
37             .getFieldErrors()
38             .forEach(error ->
39                 System.out.println(
40                     String.format("O atributo %s emitiu a seguinte mensagem %s",
41                         error.getField(),
42                         error.getDefaultMessage()
43                 )
44             );
45     } else {
46         cidades.add(cidade);
47     }
48
49     return "redirect:/";
50 }
51 }
```

O método `hasErrors()`, na linha 35, retorna true caso exista algum erro de validação. Se não houver erros, o código na linha 47 é executado e uma nova cidade é inserida.

Se houverem erros, o método `getFieldErrors()` retorna uma lista de erros (linha 37). Nesse código, usamos um `forEach` para iterar sobre a lista e imprimir os erros na linha de comando. Usamos dois métodos de `FieldError`, representado pela variável `error`. Esse objeto fornece dois métodos que permitem identificar o atributo onde o erro foi detectado (`getField()`, linha 41) e a mensagem de erro associada (`getDefaultMessage()`, linha 42).



Se você executar a aplicação e tentar inserir uma cidade sem informar os valores nome e estado, vai perceber a seguinte mensagem na console do sistema:

```
O atributo nome emitiu a seguinte mensagem O nome da cidade deve ter entre 5 e 60 caracteres
O atributo estado emitiu a seguinte mensagem A sigla do estado está limitada a dois caracteres
O atributo estado emitiu a seguinte mensagem Sigla do estado deve ser informado
O atributo nome emitiu a seguinte mensagem O nome da cidade deve ser informado
```

Contudo, não é bem isso que um usuário espera, não é mesmo? A mensagem precisa aparecer na página Web, e não na console do sistema. Por isso, vamos precisar do Freemarker e do Bootstrap para ajustar nossa página para mostrar as mensagens de erro.

O código desenvolvido nesta Seção está disponível no [Github](#), na branch `semana04-30-validacao-javax-completa`.

[Retroceder](#)[Avançar](#)[◀ Verificação de aprendizado - Criando, Alterando e Excluindo](#)[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

