02

Validações com Bean Validation

Transcrição

[00:00] Continuando nosso curso. Já temos o cadastro funcionando, aprendemos na última aula a como testar requisição do tipo POST, mandando cabeçalhos e verificando o que foi devolvido como resposta, usando o Postman. Agora, na aula de hoje, vamos aprender sobre validação. Não colocamos nenhum tipo de validação, então, na teoria posso cadastrar um tópico sem título, sem mensagem, sem o nome do curso. Só que isso não deveria ser permitido, até porque se eu não mandar o nome do curso, na hora de fazer a consulta para carregar o objeto `curso` do banco de dados, vai dar um erro, uma *exception*.

[00:35] Como posso fazer para validar essas informações? No geral, o jeito mais simples de validar seria diretamente no `TopicosController.java`. Nele, estou recebendo um `form` com os dados que foram enviados pelo cliente.

```
@PostMapping
public ResponseEntity<TopicoDto> cadastrar(@RequestBody
TopicoForm form, UriComponentsBuilder uriBuilder) {

    Topico topico = form.converter(cursoRepository);
    topicoRepository.save(topico);

    URI uri =
uriBuilder.path("/topicos/{id}").buildAndExpand(topico.getId()

    return ResponseEntity.created(uri).body(new
```

```
TopicoDto(topico));
}
```

COPIAR CÓDIGO

Poderia colocar `if-else`. Por exemplo, `if(form.getTitulo() == null )` (ou, se for vazio, eu devolvo um código 400 de requisição inválida), que é uma clássica validação com `if-else`.

```
@PostMapping
public ResponseEntity<TopicoDto> cadastrar(@RequestBody
TopicoForm form, UriComponentsBuilder uriBuilder) {
    if (form.getTitulo() == null ) {
}
```

COPIAR CÓDIGO

[01:12] O problema é que vou colocar essa lógica dentro do `Controller`, começo a poluir o `Controller` com coisas que não deveriam estar lá, e, para cada campo, existirá um `if`. Esse código vai ficar grande. Embora funcione, não é a melhor maneira possível de fazer validação.

[01:35] Para fazer esse tipo de validação de formulários, de campo obrigatório, tamanho mínimo, tamanho máximo, letra, número e etc., existe uma especificação do Java, que é o tal do Bean Validation. E o Spring se integra com essa especificação. Podemos utilizá-lo e a validação será toda feita por anotações, fica muito mais fácil.

[02:03] No nosso método `cadastrar()`, o objeto que estamos recebendo é o tal do `TopicoForm`. O cliente manda o JSON e o Spring chama o Jackson para pegar e converter no `TopicoForm`. Na classe `TopicoForm.java` que estão os atributos, os campos que quero validar. Então, no `TopicoForm.java` mesmo, no próprio objeto `Form`, em cima dos atributos, posso colocar as anotações do Bean Validation: `@NotNull`, para dizer ao Spring que o campo título não pode ser

nulo e `@NotEmpty` para dizer que esse campo não pode ser vazio. Posso também indicar um número mínimo de caracteres com `@Length(min = 5)` .

```
package br.com.alura.forum.controller.form;

import javax.validation.constraints.NotEmpty;

public class TopicoForm {

    @NotNull @NotEmpty @Length(min = 5)
    private String titulo;
    private String mensagem;
    private String nomeCurso;

    public String getTitulo(){
        return titulo;
}
```

COPIAR CÓDIGO

[03:13] Essas anotações fazem parte do Bean Validation que posso utilizar para fazer a validação. Posso colocar, por exemplo, que a mensagem tem que ter no mínimo 10 caracteres, `@NotNull @NotEmpty @Length(min = 10)` , o título no mínimo 5 caracteres, `@NotNull @NotEmpty @Length(min = 5)` . Tudo como `@NotNull` ,não nulo, e `@NotEmpty` , não vazio.

```
package br.com.alura.forum.controller.form;

import javax.validation.constraints.NotEmpty;

public class TopicoForm {

    @NotNull @NotEmpty @Length(min = 5)
    private String titulo;
    @NotNull @NotEmpty @Length(min = 10)
    private String mensagem;
```

```
    @NotNull @NotEmpty
    private String nomeCurso;

}
```

COPIAR CÓDIGO

A ideia é essa: vou anotando os atributos com as anotações do Bean Validation. Tem anotações para String, para campo decimal, campo inteiro, data e outras anotações.

[03:45] O Bean Validation também é flexível. Você pode criar uma nova anotação, caso não tenha. Por exemplo, você quer validar um campo CPF. Você pode criar uma anotação `@CPF` e ensinar como é essa validação. É uma especificação bem bacana e simples de trabalhar.

[04:05] Só que não basta apenas anotar os atributos da classe. Além de anotar os atributos, preciso dizer para o Spring: quero que você chame o Bean Validation e gere um erro caso algum parâmetro esteja inválido, de acordo com as anotações que eu coloquei.

[04:23] Para fazer isso, no nosso `TopicosController.java`, no `Topicoform`, além de estar anotado o `@RequestBody` - para dizer para o Spring que o `TopicoForm` está vindo no corpo da requisição - temos que colocar mais uma anotação, o `@Valid` - que é do próprio Bean Validation - para avisar para o Spring: quando você for injetar o `TopicoForm`, puxando os dados que estão vindo na requisição, rode as validações, `@Valid`, do Bean Validation. Nessa classe tem anotações e eu quero que você verifique se o objeto chegando é válido, conforme as anotações que eu coloquei.

```
@Post Mapping
public ResponseEntity<TopicoDto> cadastrar(@RequestBody
@Valid TopicoForm form, UriComponentsBuilder uriBuilder) {
//...
}
```

COPIAR CÓDIGO

[04:56] O próprio Spring vai rodar as validações para nós. Se estiver tudo ok, ele vai executar linha por linha do método `cadastrar()`. Se alguma coisa for inválida, ele nem vai entrar no método, vai devolver o código 400, que é o código de "Bad Request", indicando que o cliente mandou uma requisição inválida (com algum parâmetro inválido).

[05:27] Vamos testar se a validação vai funcionar corretamente. Vou voltar no Postman. Estou com a requisição da última aula para cadastrar um novo tópico. Vou testar ela, que está com tudo preenchido certinho. Vou apenas incluir o número "2" para diferenciar.

```
{
    "título":"Dúvida Postman 2",
    "mensagem":"Texto da mensagem 2",
    "nomeCurso":"Spring Boot"
}
```

COPIAR CÓDIGO

Depois disso, vou enviar pressionando o botão "Send" que está na parte centro direita da tela. Seguindo, confirmo se veio o código 201 e a resposta.

```
{
    "id": 4,
    "titulo": "Dúvida Postman 2",
    "mensagem": "Texto da mensagem 2",
    "dataCriacao": "2019-05-14T14:40:30.2972032"
}
```

COPIAR CÓDIGO

Apareceu o código quatro, `"id": 4`, porque, quando fiz a alteração no código que está no `TopicosController.java` e salvei, ele reiniciou o servidor. E aí toda

vez que ele reinicia, ele apaga os registros e cria tudo que está no `data.sql`. Ele voltou a ter apenas três registros no banco, três tópicos, disparei a requisição e ele criou o quarto. Disparando de novo, ele cria o registro 5.

```
{
    "id": 5,
    "titulo": "Dúvida Postman 2",
    "mensagem": "Texto da mensagem 2",
    "dataCriacao": "2019-05-14T14:40:58.2047724"
}
```

COPIAR CÓDIGO

[06:20] Funcionou com sucesso, só que agora vou testar uma requisição mandando o campo título vazio, `"titulo":""` que deveria dar erro. Pressionando "Send", ele me mostra o erro no "Status": "HTTP 400 Bad Request". Significa que o Spring detectou que era inválido, devolveu o código 400 e não cadastrou no banco de dados.

[06:43] O que ele devolve quando dá esse erro? Devolve o código 400 e, no corpo da resposta, o Spring me manda um JSON com várias informações: a data, `"timestamp"`; status, `"status"`; erro, `"erro"`.

```
{
    "timestamp": "2019-05-14T17:41:11.978+0000",
    "status": 400,
    "error": "Bad Request",
    "errors": [
//...
}
```

COPIAR CÓDIGO

Para cada campo inválido, ele manda um tal de `"codes"`, `"NotEmpty"`, qual é o campo, o tipo de validação, qual foi o argumento passado.

```
"codes": [
    "NotEmpty.topicoForm.titulo",
        "NotEmpty.titulo",
        "NotEmpty.java.lang.String",
        "NotEmpty"
//...
```

COPIAR CÓDIGO

Ele devolve um JSON gigantesco, contendo várias informações sobre o erro que aconteceu.

[07:26] Ele validou, está funcionando. O problema é só que a resposta, o corpo da resposta vem com um JSON monstruoso, enorme. Poderia vir algo simplificado, só falando, por exemplo, que o campo `"título"` é obrigatório, campo `"mensagem` " deve ter no mínimo 5 caracteres.

[07:45] Já vimos que está funcionando. Na próxima aula vamos aprender a simplificar esse JSON, porque esse é o padrão enviado pelo Spring, mas posso personalizar.