04

Mensagens de erro

Transcrição

[00:00] Nós já associamos o formulário a um determinado objeto usando as tags do Thymeleaf, associamos os “inputs”. Esses “inputs” iniciais que são obrigatórios a um determinado atributo. Faltou ainda associar o “textarea” e tem um erro que temos que resolver e que vai acontecer.

[00:22] Talvez você já tenha percebido, que é quando você tentar acessar o formulário - você vai tomar um erro. Se você descer aqui para baixo, ele vai dizer assim: “A ‘requisicaoNovoPedido’ eu não consigo acessar isso como atributo do ‘request’ ”. Por que isso dá erro?

[00:40] Porque quando você faz “/pedido/formulario”, você está acessando uma outra “action”, que é essa “action” aqui. Na “action” de baixo do “novo”, o Spring automaticamente vai criar uma requisição de “novoPedido”.

[00:56] O Thymeleaf vai usar essa “requisicaoNovoPedido”. Ele vai usar isso na hora de renderizar essa página. Por quê? Porque ele associa o formulário a um objeto “requisicaoNovoPedido”. Então, nesse caso, o que acontece quando você tenta abrir o formulário? Você não tem uma “requisicaoNovoPedido”.

[01:07] Então para consertar isso, é só colocar aqui. Aí você passa a disponibilizar uma “requisicaoNovoPedido” para a tela. Então esse erro passa a não acontecer mais. Eu coloquei lá no formulário. Tento cadastrar ele de volta para cá porque a “Url” e “Imagem”, por exemplo, são obrigatórios.

[01:28] O que eu já preenchi anteriormente, ele mantém. Aquela inteligência do Thymeleaf. Mas nós precisamos apresentar mensagens de erros para o

usuário saber o que ele precisa terminar de preencher. Uma outra coisa que nós temos que fazer, antes de mais nada, é associar o “textarea” também como um “field”, que é chamado de “descricao” - e ele também não precisa de um “name”.

[01:53] Então agora está tudo bom. Esse aqui não é obrigatório. Não tem problema, mas ele, pelo menos, está associado. Então, por exemplo: se eu associar aqui e voltar, ele não dá erro. Os outros deram erros. Mas o que eu já preenchi anteriormente continua aparecendo. O que nós temos que fazer agora é apresentar mensagens de erro.

[02:11] E isso é bem interessante porque existe um padrão de nomenclatura para essas mensagens de erros que estão acontecendo. Então duas coisas estão acontecendo por baixo dos panos. Uma é quando ele valida o nome do produto, ele gera um erro. O nome desse erro é assim, é “NotBlank.requisicaoNovoProduto” com “R” minúsculo, “.nomeProduto”.

[02:38] E por padrão ele já associa a uma mensagem, a um valor para esse erro - chamado assim: “campo deve ser preenchido”, algo do tipo. Eu não sei exatamente se é esse o valor. Nós vamos ver isso agora.

[02:50] Agora como que nós acessamos esses erros que estão sendo gerados, para que possamos apresentar essas mensagens para o usuário? Existe um outro atributo do Thymeleaf chamado “th:errors” que nós podemos associar a uma nova “div”. Dentro dessa “div” vai aparecer os “Erros do nome do produto”, desse “input”.

[03:13] Então como nós fazemos? “th:errors”. E aí vamos associar e vai pedir o seguinte: “me apresente todos os erros associados ao nome do produto”. Como eu faço para acessar o campo que eu quero que ele pegue os erros?

[03:26] De novo, asterisco! Usamos essa sintaxe para acessar o “nomeProduto”, que é um atributo do objeto do formulário. Pronto! Só com isso nós já temos a

mensagens de erro sendo apresentadas - “não pode estar em branco”, isso é uma mensagem de erro que aparece aqui, “não pode estar em branco”.

[03:46] Então já é uma mensagem de erro que estamos mostrando. A mesma coisa temos que fazer para todos os outros. Tanto “UrlProduto” quanto “nomeProduto” e “UrlImagem”. Então erros do “nomeProduto” e erros da “Url” e erros da “UrlImagem”. Então, pronto!

[04:19] Agora os erros vão aparecer logo embaixo. “Não pode estar em branco”, “Não pode estar em branco” e “Não pode estar em branco”... Será que o Bootstrap tem algum tipo de estilo que possamos aplicar nas mensagens de erro?

[04:31] De repente, ele tem lá no formulário alguma coisa. Então podemos dar uma navegada e verificar se o Bootstrap já tem automaticamente alguma coisa de mensagens de erro. Mas é no “Forms” e não no “Button”. Vamos ver se ele tem. Não tem nenhum layout. Quer dizer, talvez ainda tenha. Vou procurar por “error”.

[05:05] Não, “error” não tem. Aqui já tem mensagens de erro. Então essa mensagem, qual é o estilo dela? Está associada a “City”. Tanto a mensagem quanto o “input”. Vamos ver “City”, “Please provide a valid city”. O “input” que ele está vendo aqui é o “input” de cidade e nós procuramos esse “input” aqui.

[05:33] Pronto! Então “invalid feedback”. É essa classe que nós podemos aplicar às nossas “divs”. “Invalid feedback” e “Invalid feedback”... Pronto! Então vamos ver se muda alguma coisa? Desistiu? O que acontece? Isso é engraçado, ele não mostra!

[06:06] Por que ele não está mostrando? Isso eu lembrei! Esse campo, esse “input” que está com “invalid feedback” tem que estar com o “is-invalid”. Ele tem que ter essa classe. Então, por exemplo: se eu associar essa classe a um determinado “input” - o “is invalid”, por exemplo - ele vai mostrar só para o nome do produto que foi que o que associei.

[06:31] Ele só mostra a mensagem se o “input” tiver recebido essa classe, esse “is invalid”. Só tem uma questão que nós temos que trabalhar aqui, que é a seguinte: “Thymeleaf error class”. Eu acho que é isso. Vamos lá!

[07:02] Aqui é tudo feito na hora. Vamos colocar “th:errorclass=is invalid”. Ou seja, caso tenha algum erro, ele adicionará essa classe “is invalid”. É o que eu espero que aconteça. Então, “Cadastrar”. Beleza, tudo está aparecendo! E olhe só a mensagem que aparece: não pode estar em branco. É isso que ele associa ao “input”.

[07:30] Então é aquilo que eu falei, “o campo deve ser preenchido”, na verdade é bem diferente disso. É “não pode estar em branco”, ou seja, a mensagem “default” para esse erro que está sendo gerado para esse “input” é “não pode estar em branco”.

[07:43] Ou seja, todas as mensagens “NotBlank” são isso. Ele associa por padrão essa mensagem. O que você pode fazer é mudar isso. Você pode mudar essa mensagem. Como nós mudamos da melhor forma possível? De uma maneira internacionalizada.

[07:57] Se você quiser trabalhar com várias mensagens em português e inglês, por exemplo. Uma maneira de fazer isso é você colocar o “message” no próprio atributo. Mas eu prefiro trabalhar em arquivo. Então você coloca as suas mensagens aqui e é até mais fácil para manter, “messages.properties”.

[08:15] E aí você pega - o nome da mensagem é essa e o nome do erro é esse. Nada de aspas. E aí você coloca, “campo nome do produto é obrigatório”. Pronto! Já mudei! Então nós vamos subir de novo porque quando você criar um arquivo de propriedade, você precisa reiniciar. Então, vamos lá!

[08:39] “Formulario”, “Cadastrar”, “O nome do produto é obrigatório”. Os outros continuam com o valor “default”. Mas no nome do produto ficou assim. Ele está com problema de internacionalização porque eu criei o arquivo como “ISO”. Formato Latim-1, como eles chamam.

[08:57] Nós alteramos isso, na conversão ele dá alguns erros e colocamos "obrigatório". Ajustamos! Então, "Cadastrar". Beleza! "O campo nome do produto é obrigatório". Agora se eu preencher isso, ele já não vai aparecer com aquele erro e a "Url" e "Imagem" continua com erro.

[09:17] Agora o nosso formulário está bem fácil de ser mantido. Está bem fácil de ser percebido pelo usuário, o que está acontecendo e tudo o mais. Tudo isso, nós nem tivemos que alterar tanto assim o nosso HTML. Tanto para aplicar o Bootstrap quando para aplicar o Thymeleaf. Ele ficou ainda muito legível e muito fácil de manter.

[09:39] E você continua podendo abrir o formulário como HTML puro. Você pode chegar aqui, dar "Web Browser" e ele mexerá. Então um front-end, um designer pode trabalhar em cima desse HTML e pronto. Não tem problema! Já vai aplicar! Não precisa subir a aplicação para poder renderizar a página, para poder fazer as alterações e depois chamar de novo para ver como ficou. Ele só abre o HTML puro!