04

Aplicando validadores

Transcrição

[00:00] Vimos no vídeo anterior que se tentarmos fazer uma requisição para criarmos uma nova oferta passando valores inválidos, vamos tomar alguns erros na hora que tentarmos converter esses valores, que são *strings* em data e em número.

[00:13] E no Console podemos ver que o primeiro que tomamos é um "DateTimeParseException". Na hora que tentamos converter no método `toOferta`, do `RequisicaoNovaOferta`. Então aqui dentro da própria oferta temos um objeto que converte e, na hora em que ele tenta fazer essa conversão, ele toma o erro.

[00:31] Para não termos esse erro acontecendo, precisamos garantir que essa requisição seja feita. Podemos utilizar Bean Validation aqui também. Uma coisa que é muito importante e que precisamos ter é o `pedidoId`, então ele tem que ser `@NotNull`.

[00:53] Esse `pedidoId` nós que preenchemos não é o usuário, então eu não vou me preocupar com ele aqui. Eu vou colocar um `@NotNull` no valor, dizendo que é obrigatório você preencher um valor, assim como vai ser obrigatório preencher uma data de entrega.

[01:10] Depois eu vou dizer o seguinte: esse valor tem que vir em um determinado formato, e o formato que esperamos é tanto 2 como 299, como R$ 2,99 ou R$ 1,99, esse é o padrão que esperamos receber. Eu vou construir isso aqui, usando o atributo `(regexp =` e usando expressão regular, que é uma

tecnologia exatamente para encontrarmos padrões em *string* e vamos utilizar essa tecnologia para validar a *string*.

[01:40] E vamos dizer que essa *string* tem que começar com dígito. Aí você coloca `"^\d")`, que significa dígito. Só que essa barra inversa é um caractere especial e nós temos que, como falamos usualmente, "escapá-la" e usamos a própria barra invertida para fazermos isso.

[02:02] Só que ele não precisa passar só um dígito, ele pode passar, por exemplo, 299 ou pode passar vários dígitos. Para simplificarmos, podemos colocar `"^\\d+"`, o sinal de adição significa vários desse anterior ou vários dígitos.

[02:18] Depois disso, por exemplo: se o usuário passar 15.99, nós esperamos que logo em seguida desses dígitos venha um ponto, e depois desse ponto temos mais dígitos. Então vamos usar a barra invertida de novo, `"^\\d+.\\d+"`. Só que eu não vou colocar só "d+", ou seja, eu não espero vários dígitos depois da vírgula ou do ponto, eu só espero dois; então eu vou usar essas chaves aqui e colocar 2: `"^\\d+.\\d+{2}"`

[02:48] Só que esse valor, "15.99", em forma de *string*, será validado por esse `"^\\d+.\\d+{2}"`, porque ele está dizendo o seguinte: eu quero vários dígitos antes do ponto, no caso aqui tem dois, mas poderia ser mais.

[03:02] Por exemplo: isso aqui é válido também, "6546415.99", quer dizer, eu espero um ponto também. E depois do ponto eu espero dois dígitos, está aqui, dois dígitos, ou seja, essa "6546415.99", é uma *string* válida.

[03:17] Então essa nossa validação está boa, só que eu tenho que colocar mais uma barra invertida antes do ponto, `"^\\d+\\.\\d+{2}"`. Toda vez que eu quero esperar alguma coisa, eu tenho que colocar essa barra invertida e ponto final (`\.`).

[03:31] Só que o ponto e os dois números que vem depois são opcionais, então eu vou colocá-los entre parênteses: `"^\\d+(\\.\\d+{2})?"`. Vou dizer o seguinte: isso aqui pode ou não vir, ou seja, o ponto e dois dígitos pode ou não vir com o sinal de interrogação.

[03:49] E vou dizer que depois disso acabou a *string*. Eu não espero mais nada, é para isso que eu coloquei esse caractere de cifrão (`$`). Então esse é o *pattern* do `@Pattern(regexp = "^\\d+(\\.\\d+{2})?$"`.

[04:00] Já o *pattern* da `dataDaEntrega` é mais simples de construir. Veja, ele começa com dígito, só que não são vários dígitos. Deixe-me apagar aqui. Não são dois dígitos, eu espero dois dígitos, porque essa *string* da data vai vir mais ou menos assim, "10/10/2020". Vai vir nesse formato, então ela começa com dois dígitos.

[04:23] Depois vem uma barra, depois vem mais dois dígitos, então: `d{2}/\\d{4}`. Pronto, esse é o padrão da data que estamos usando aqui.

[04:44] Já colocamos o padrão para valor e data da entrega e agora eu só preciso ativar essa validação do Bean Validation com `@Valid`, do `javax.validation`. Salvo isso, agora a exceção vai mudar, não vamos tomar aqueles erros que estávamos tomando antes. Esse erro de "DateTimeParseException" nós não vamos tomar mais.

[05:06] Vou fazer a requisição de novo com valores inválidos, vou enviar esses valores. Olhe o erro que deu aqui na aba Console. Ele até deu um *warn*, mas ele deu um erro sim, que é "MethodArgumentNotValidException".

[05:22] O que vamos fazer agora? Se você olhar seguindo o que ele está imprimindo, ele está fazendo toda a descrição dos erros de validação que aconteceram e ele vai falar de todos os campos, data da entrega, valor e tudo o que veio errado ele vai mostrar - inclusive dizendo qual o valor errado, dizendo o que ele espera e uma mensagem de erro associada.

[05:47] O que vamos fazer? Precisamos transformar essa exceção e devolver para o usuário, porque olhe só: ele clica, está dando erro e nada acontece. O ideal é que mostremos que a data está errada, o usuário preencheu o valor certo. Pronto, isso é um valor correto, deu erro na data da entrega.

[06:08] Enviou a oferta, deu erro. "MethodArgumentNotValidException", está sempre tomando esse erro, só que o usuário não está sabendo o que está acontecendo.

[06:20] Precisamos dar um jeito de transformar essa exceção pegando todo o conteúdo que tem dentro dela e tornar esse conteúdo em um conteúdo que possa ser interpretado pelo Vue, para que ele possa indicar: "esse campo aqui está com erro". Então vamos fazer uma transformação de dados para apresentar mensagens de erro para o usuário.

[06:43] E isso nós vamos fazer no próximo vídeo. Até lá!