04

Internacionalização da validação

Capítulo 11 - Internacionalização da validação

Neste momento conseguimos adicionar novos produtos em nossa lista sem haver uma validação dos valores de quantidade ou preço. Se quisermos, podemos armazenar produtos sem uma quantidade determinada.

Devemos achar um modo de garantir que os produtos sejam armazenados com valores plausíveis, cada campo possui características que devem ser seguidas.

Podemos passar no *Rails* que tipo de validação queremos que cada campo possua. Começaremos pela "Quantidade". Fazemos isso dentro do arquivo "app/models/produto.rb". O *Rails* possui um método, chamado de `Validates` que nos ajuda nesse processo:

```
class Produto < ActiveRecord::Base

    validates :quantidade, presence: true

end
```

Estamos aqui validando a presença da quantidade. Dessa maneira, se não passarmos valor para esse campo na hora de armazenar um novo produto, ele não aparece na lista. Mas o que realmente aconteceu? O método `create` dentro do *controller* não acusou erro.

Simulando a mesma situação no console, tentando armazenar um item sem algum campo, esta ação é concluída e o item é inserido no banco de dados. Isso aconteceu pois não recarregamos o console após termos feito a alteração de validação, então precisamos usar o comando `reload!` . Agora sim, o console irá reclamar a falta dos campos:

```
irb(main):001:0> produto = Produto.create nome: "Blusa xadrez"
   (0.1ms)  begin transaction
   (0.1ms)  rollback transaction
   ...
```

Acontece um *rollback*, ou seja, o banco de dados descartou a transação. Para que o erro se mostre mais aparente podemos fazer `produto = Produto.create! nome: "Blusa xadrez"` (com o "!"), o que retorna:

```
...
   (0.1ms)  begin transaction
   (0.1ms)  rollback transaction
ActiveRecord::RecordINvalid: Validation failed: Quantidade can't be blank
...
```

De fato, a Quantidade não pode ter valor nulo. Outra maneira de perceber que o produto não pode ser salvo é primeiramente instanciá-lo e depois tentar salvá-lo com `Produto.save` . Retornará `false` .

No nosso site não aparece nenhuma mensagem de erro. Precisamos alertar quem estiver inserindo o produto que algo saiu errado. Uma forma amigável de mostrar esse erro é substituir o comando no *controller*:

```
def create
    valores = params.require(:produto).permit :nome, :preco, :descricao, :quantidade
    produto = Produto.new valores
    if produto.save
        redirect_to root_url
    else
        render :new
    end
end
```

Se o produto for salvo, seremos redirecionados para a *root*, se não, renderiza-se a página de criação de produtos. Desse jeito ainda não temos muita ideia do que poderia ter acontecido. Nada é mostrado para dizer que o produto não foi adicionado por algum motivo.

Método *errors*

O produto que criamos anteriormente pelo console ficou gravado no banco de dados, mesmo possuindo erros. Podemos verificá-los fazendo `produto.errors` , o que devolve todos os erros inerentes a ele. Para ler as mensagens de erro, podemos fazer `produto.errors.full_messages` . No nosso caso:

```
irb(main):001:0> produto.errors.full_messages
=> ["Quantidade can't be blank"]
```

Implementando a mensagem de erro

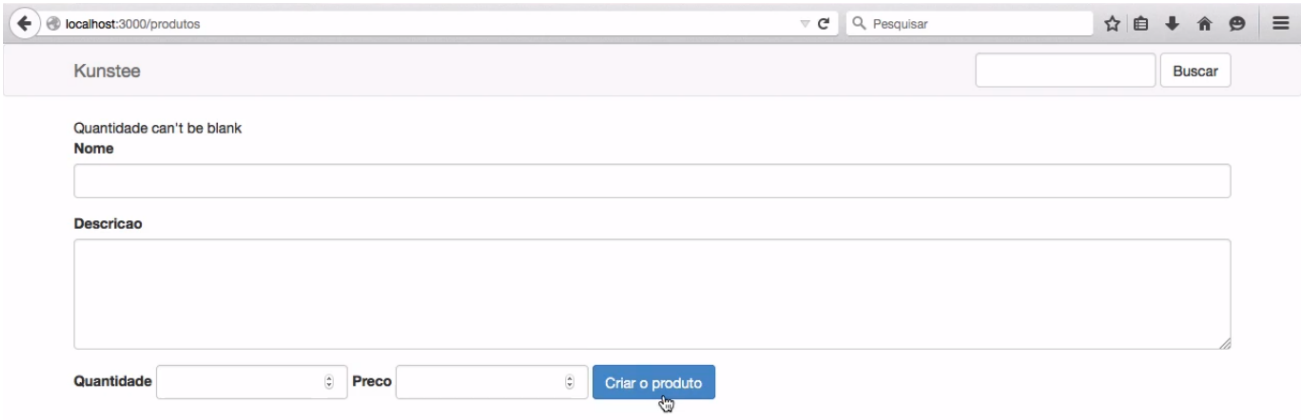
Vamos implementar a mensagem de erro direto na página "new". Para disponibilizar o produto lá, usamos o símbolo de variável de instância:

```
def create
    valores = params.require(:produto).permit :nome, :preco, :descricao, :quantidade
    @produto = Produto.new valores
    if @produto.save
        redirect_to root_url
    else
        render :new
    end
end
```

Agora a variável `@produto` está disponível para a página de criação. Lá fazemos:

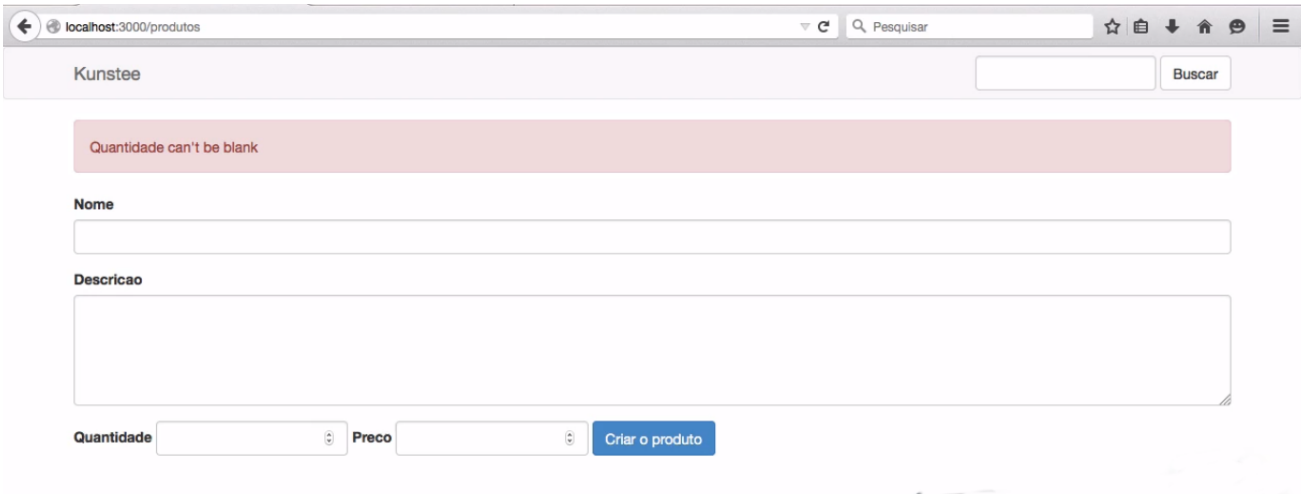
```
<% @produto.errors.full_messages.each do |message|%>
    <%= message %>
<% end %>
...
```

Agora, ao não passarmos a quantidade, a página nos mostra



Podemos melhorar a estética dessa mensagem usando o Bootstrap:

```
<% @produto.errors.full_messages.each do |message|%>
    <div class="alert alert-danger" role="alert"><%=message%></div>
<% end %>
```



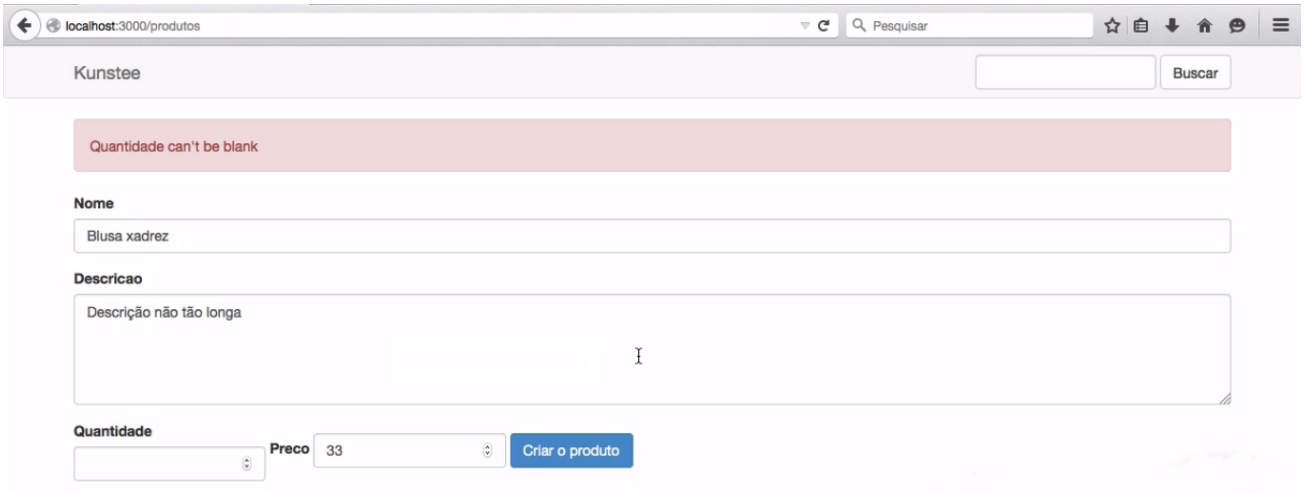
Perceba que não importa o que escrevemos na página de criação de produtos, sempre perderemos dados que podem ser importantes se houver um erro. Se erramos um campo, os outros, os quais estavam corretos, se perdem. Para consertar isso, na página de criação substituímos o `Produto.new` por `@produto` :

```
<%= form_for @produto do |f| %>
```

Porém, para isso funcionar, precisamos instanciar a variável `@produto` dentro de um método `new` no *controller* logo depois do `index` :

```
def new
    @produto = Produto.new
end
```

O que é até melhor, pois estaremos realmente instanciando um objeto. Agora se tentarmos salvar um produto sem quantidade aparecerá o erro, mas não perderemos as outras informações:



Um outro aviso interessante de aparecer para nós é o aviso de que realmente funcionou o armazenamento de um novo produto. Devemos tomar cuidado pois, ao implementarmos essa mensagem, ela durará apenas uma requisição (de criação do produto) e não será mostrada. Então usamos o método `flash` para que ela dure uma a mais:

```
def create
    valores = params.require(:produto).permit :nome, :preco, :descricao, :quantidade
    @produto = Produto.new valores
    if @produto.save
        flash[:notice] = "Produto salvo com sucesso"
        redirect_to root_url
    else
        render :new
    end
end
```

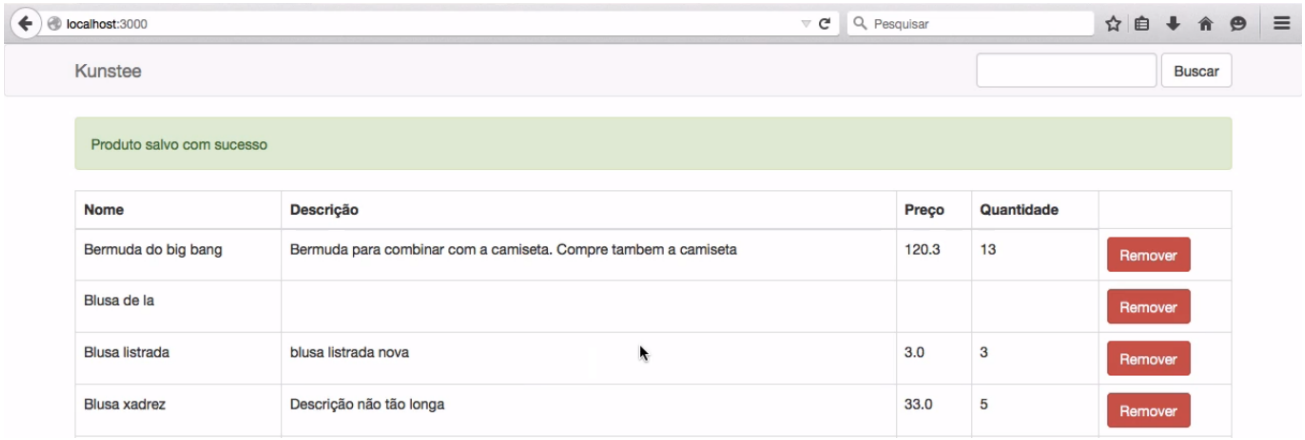
Não podemos esquecer:

> as variáveis de instância duram apenas uma requisição na Web.

E no `index`, com o alerta do Bootstrap:

```
<% if flash[:notice] %>
    <div class="alert alert-success" role="alert"><%= flash[:notice] %></div>
<% end %>
...
```

A condicional se faz necessária para que a mensagem de sucesso apareça apenas quando adicionamos um novo produto. E, de fato:



Validando o nome e traduzindo as mensagens

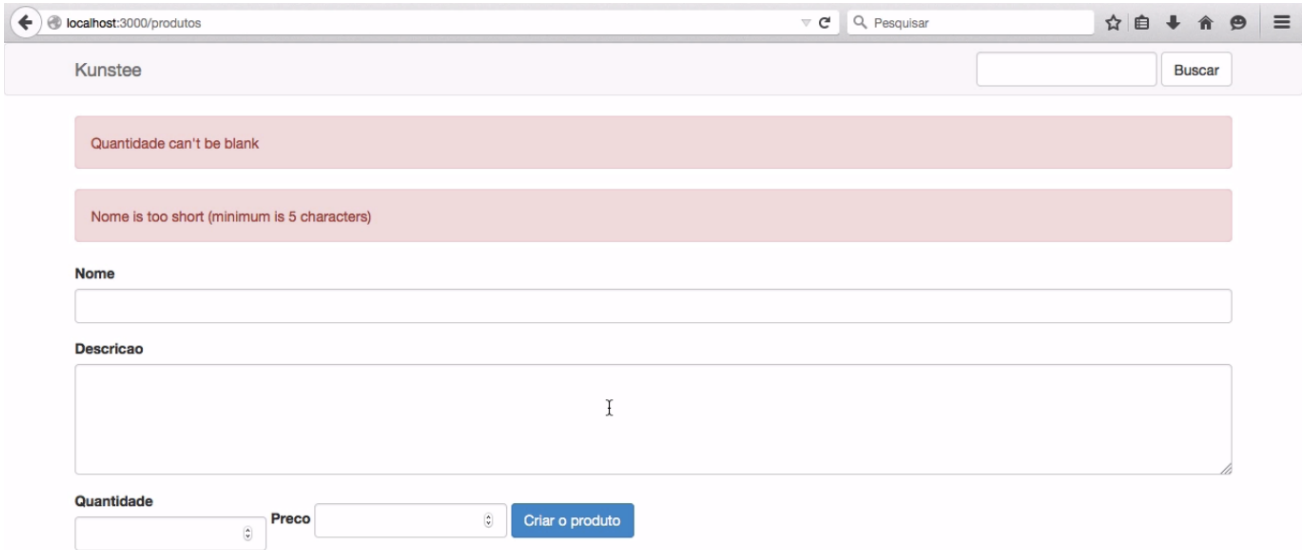
Vamos validar agora o campo do nome. Queremos que ele tenha um mínimo de 5 caracteres. Será parecido com o que fizemos para a quantidade:

```
class Produto < ActiveRecord::Base

    validates :quantidade, presence: true
    validates :nome, length: { minimum: 5 }

end
```

Ao tentarmos criar um produto, digamos, com todos os campos vazios:



A validação funciona. Porém perceba que as mensagens estão com termos em inglês. Vamos traduzí-las. Existe uma biblioteca que traduz todas elas para português, a *rails-i18n*. Adicionamos no gemfile:

```
gem 'rails-i18n', '~> 4.0.0'
```

E rodamos o `bundle install` . Para solicitarmos a liguagem em português, fazemos no arquivo "/config/application.rb":

```
module Lojadecamisetas
    class Application < Rails::Application
```

```
        config.i18n.default_locale = "pt-BR"
    end
  end
```

Reinicializamos o servidor e, de fato: