

Capítulo 8 - Layouts

Formulário

Vamos implementar um formulário para poder cadastrar novos produtos na nossa lista. O que fazíamos até agora era por meio do console do *Rails*. Precisamos de uma interface dentro do próprio site.

Como o formulário servirá para adicionar novos produtos, daremos ao arquivo HTML o nome de "new.html.erb" (convenção), dentro do diretório "views/produtos". Começemos a implementar:

```
<html>
<body>
<form>    //formulário

Nome <input type="text" name="nome"/><br/>    //"nome" é tipo texto

Descrição <textarea name="descricao"/><br/>    //"descricao" é tipo textarea

Quantidade <input type="number" name="quantidade"/><br/>    //"quantidade" é tipo number

Preço <input type="number" name="preco" step="0.01"/><br/>    //"preço" também e crescerá de 0.01 er

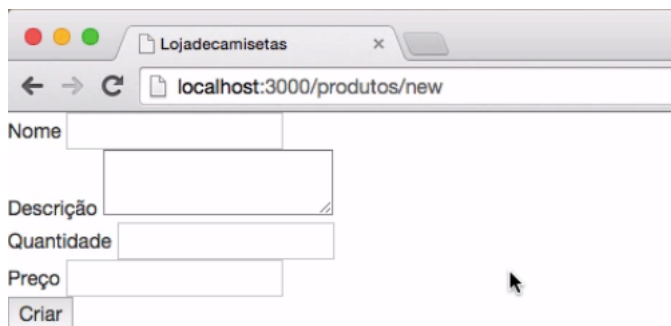
<button type="submit">Criar</button>    //botão para criar um novo produto

</form>
</body>
</html>
```

Devemos editar a rota acrescentando a linha:

```
get "/produtos/new" => "produtos#new"
```

Se acessarmos localhost:3000/produtos/new :



The screenshot shows a web browser window with the title "Lojadecamisetas". The address bar displays "localhost:3000/produtos/new". The form contains the following fields and a button:

- Nome: A text input field.
- Descrição: A text area input field.
- Quantidade: A number input field.
- Preço: A number input field with a step attribute.
- Criar: A submit button.

O principal da interface está feito, porém queremos que, quando digitarmos nos campos as características dos produtos, eles sejam adicionados nas nossas tabelas. Como está, se preencheremos os campos, os parâmetros irão parar na URL, voltará para a mesma página, mas não será criado um novo produto. Acrescentamos uma `action` (para onde iremos depois de inputar um novo produto) e um `method` (a criação propriamente dita por meio do *post*) dentro do `<form>` :

```
<form action="/produto" method="post">
```

Ainda não conseguimos criar o produto pois o navegador não achará esse método *post* para `/produto`. Então, nas rotas:

```
post "/produto" => "produtos#[alguma coisa]"
```

Mas que "coisa" é essa? É o método que agora, de fato, criará um novo produto, o método `create`. É uma nova *regra de negócio* dentro do *controller*. Para conseguirmos pegar os parâmetros, o *Rails* possui um objeto que os representa, o *params*:

```
def create
  nome = params["nome"]
  descricao = params["descricao"]
  quantidade = params["quantidade"]
  preco = params["preco"]
end
```

E fazemos a rota do `post` :

```
post "/produto" => "produtos#create"
```

Como a Classe "ProdutosController" é filho de "ApplicationController", ela deve seguir algumas regras. Uma delas é a regra de segurança do `post`. Toda vez que fazemos um `post`, precisamos enviar um *Token*, o qual confirma que tal formulário está recebendo a requisição e uma única vez. O próprio *Rails* implementa isso para nós. Ao invés de escrevermos o formulário, existe um método que já cria a estrutura do formulário automaticamente:

```
<html>
<body>

<%= form_for Produto.new do |f| %>

Nome <input type="text" name="nome"/><br/>

Descrição
<textarea name="descricao"></textarea><br/>

Quantidade
<input type="number" name="quantidade"/><br/>

Preço
<input type="number" name="preco" step="0.01"/><br/>

<button type="submit">Criar</button>

<% end %>
```

```
</body>
</html>
```

O `<%= form_for Produto.new do |f| %>` criará tanto a URL `/produto` quanto o método `post`, por isso excluimos o `<form ...>`.

Se deixarmos como está, ainda não funciona. O navegador acusará que não definimos o método `produtos_path`. Na verdade só precisamos seguir o padrão e, no caso, colocar tudo no plural:

```
post "/produtos" => "produtos#create"
```

Ainda assim, apesar da rota nos levar primeiro para a regra de negócio `create`, não existe um template com esse nome. Criemos uma página que confirma a postagem do produto, a `create.html.erb`:

```
<html>
Criado!
</html>
```

Os produtos podem ser criados a partir de agora, mas só serão impressos no log do console. Eles não serão realmente salvos. Então faremos no método `create`:

```
def create
  valores = params.require(:produto).permit!
  produto = Produto.create valores
end
```

Dessa maneira passamos todos os parâmetros usando o método `require`, que faz a solicitação do produto. E, no formulário:

```
<%= form_for Produto.new do |f| %>

Nome <input type="text" name="produto[nome]"/><br/>

Descrição
<textarea name="produto[descricao]"/></textarea><br/>

Quantidade
<input type="number" name="produto[quantidade]"/><br/>

Preço
<input type="number" name="produto[preco]" step="0.01"/><br/>

<button type="submit">Criar</button>

<% end %>
```

Assim cada parâmetro fica ligado a seu objeto. É uma convenção do *Rails*. Agora, enfim, os produtos podem ser incluídos na lista:

Nome camiseta azul

Descrição

Quantidade 30

Preço 30

Criar

Criado!

Nome	Descrição	Preço	Quantidade
Bermuda do big bang	Bermuda para combinar com a camiseta. Compre também a camiseta	120.3	13
Camiseta de gastronomia	Camiseta super bacana gourmet	37.66	8
Camiseta do Big Bang Theory	Camiseta super bacana do bbt	70.5	10
camiseta azul	camiseta azul	30.0	30

Permissões

Do jeito que está, corremos o risco de alguma pessoa mal intencionada mudar o código do formulário e passar parâmetros que não deveriam existir. O problema está no `permit!` dentro do *controller*, que libera a inserção de quaisquer atributos. O que vamos fazer é uma *whitelist*:

```
def create
  valores = params.require(:produto).permit :nome, :preco, :descricao, :quantidade
  produto = Produto.create valores
end
```

Agora só serão aceitos os parâmetros passados.

Formulário pelo Rails

Vamos ainda dar uma polida no formulário em HTML. Não precisamos ficar repetindo o padrão `<input ... "produto[atributo]">`. O próprio Rails pode gerar essas linhas para nós com código *Ruby*:

```
<%= form_for Produto.new do |f| %>

Nome
<%= f.text_field :nome %><br/>    //campo de texto para o nome

Descrição
<%= f.text_area :descricao %><br/>    //campo de texto maior para a descrição

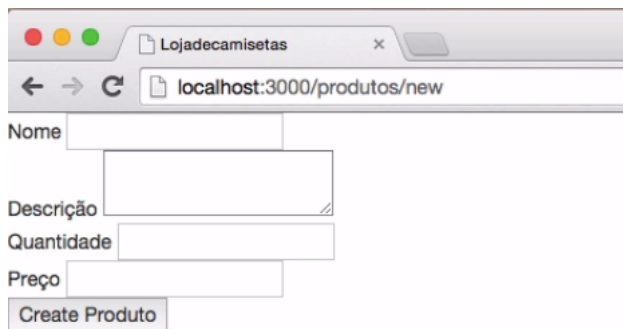
Quantidade
<%= f.number_field :quantidade %><br/>    //campo numérico para a quantidade

Preço
<%= f.number_field :preco %><br/>    //campo numérico para o preço

<%= f.submit %>    //botão para submeter o novo produto

<% end %>
```

O *Rails* também atribui um *id* para cada campo. Este é o resultado final:



The screenshot shows a web browser window with the title 'Lojadecamisetas'. The address bar shows 'localhost:3000/produtos/new'. The form contains the following elements:

- Nome: A text input field.
- Descrição: A text area input field.
- Quantidade: A numeric input field.
- Preço: A numeric input field.
- Create Produto: A submit button.

O `number_field` aceita tanto números inteiros quanto decimais, mas não queremos isso para a "quantidade". Passaremos para o atributo um parâmetro a mais. Todos esses comandos que estamos utilizando até agora podem ser encontrados no site do *Rails* em api.rubyonrails.org (<http://api.rubyonrails.org>). Se procurarmos por "form_for", encontramos o parâmetro que queremos: `step`.

```
Quantidade
<%= f.number_field :quantidade, step: 1 %><br/>
```

Ou seja, o campo "Quantidade" cresce de um em um. Se tentarmos armazenar um número decimal, aparece uma mensagem de erro:

Vamos mudar também a cara do botão, para ele aparecer em português:

```
<%= f.submit "Criar o produto" %>
```

Todos os `<%= f. ... %>` são chamados de *form helpers*.

Layout do formulário

Da mesma forma que fizemos com as tabelas, vamos utilizar as Classes do Bootstrap para melhorar a cara do nosso formulário. Para os campos usamos a Classe "form-control":

```
<%= form_for Produto.new do |f| %>
```

Nome

```
<%= f.text_field :nome, class: "form-control" %><br/>
```

Descrição

```
<%= f.text_area :descricao, class: "form-control", rows: 4 %><br/> // quatro linhas para dar um €
```

Quantidade

```
<%= f.number_field :quantidade, step: 1, class: "form-control" %><br/>
```

Preço

```
<%= f.number_field :preco, class: "form-control" %><br/>
```

```
<%= f.submit "Criar o produto" %>
```

```
<% end %>
```

localhost:3000/produtos/new

Nome

Descrição

Quantidade

Preço

Criar o produto

E, para o botão, usamos a Classe "btn btn-primary" que provê um *layout* básico:

```
<%= f.submit "Criar o produto", class: "btn btn-primary" %>
```

Conseguimos melhorar ainda mais a cara do formulário adicionando algumas *features*, como *labels*.

```
<%= form_for Produto.new do |f| %>

<%= f.label :nome %>
<%= f.text_field :nome, class: "form-control" %><br/>

<%= f.label :descricao %>
<%= f.text_area :descricao, class: "form-control", rows: 4 %><br/>

<%= f.label :quantidade %>
<%= f.number_field :quantidade, step: 1, class: "form-control" %><br/>

<%= f.label :preco %>
<%= f.number_field :preco, class: "form-control" %><br/>

<%= f.submit "Criar o produto", class: "btn btn-primary" %>

<% end %>
```

Elas são, inclusive, mapeadas pelo *id* de cada campo. Inspeccionando o elemento observamos:

```
<label for="produto_nome">Nome</label>
<input class="form-control" type="text" name="produto[nome]" id="produto_nome" />
```

As mudanças aqui não são muito aparentes, deixando os textos em **negrito** e ao clicarmos no nome, a área para escrita é selecionada. Podemos também agrupar os campos como acharmos melhor. Aqui, cada grupo terá apenas um campo:

```
<div class="form-group">
  <%= f.label :nome %>
```

```

<%= f.text_field :nome, class: "form-control" %>
</div>

<div class="form-group">
  <%= f.label :descricao %>
  <%= f.text_area :descricao, class: "form-control", rows: 4 %>
</div>

<div class="form-group">
  <%= f.label :quantidade %>
  <%= f.number_field :quantidade, step: 1, class: "form-control" %>
</div>

<div class="form-group">
  <%= f.label :preco %>
  <%= f.number_field :preco, class: "form-control" %>
</div>

<%= f.submit "Criar o produto", class: "btn btn-primary" %>

<% end %>

```

The screenshot shows a web browser window at the URL `localhost:3000/produtos/new`. The form contains the following elements:

- Nome:** A single-line text input field.
- Descricao:** A multi-line text area input field.
- Quantidade:** A single-line number input field.
- Preco:** A single-line number input field.
- Submit Button:** A blue button with the text "Criar o produto".

Perceba que excluímos os `
`. Para os campos de "Quantidade" e "Preco" e o botão, vamos deixá-los na mesma linha, um pouco menores, afinal não precisamos de uma linha inteira para digitar os números.

```

<div class="form-inline">
  <div class="form-group">
    <%= f.label :quantidade %>
    <%= f.number_field :quantidade, step: 1, class: "form-control" %>
  </div>

  <div class="form-group">
    <%= f.label :preco %>
    <%= f.number_field :preco, class: "form-control" %>
  </div>

  <%= f.submit "Criar o produto", class: "btn btn-primary" %>
</div>

```


Nos resta colocar a barra de navegação na página do formulário:

```
<html>
<body>

<%= nav_bar brand: "Kunstee", brand_link: root_url %>

<%= form_for Produto.new do |f| %>
```

Perceba que até agora estamos apenas copiando o código de uma página para outra. Se tivéssemos diversas, este trabalho seria considerado uma má prática. No futuro veremos como otimizar isso.

Containers e application

Vamos trabalhar um pouco mais com o *layout* da nossa página. As tabelas poderiam estar melhor distribuídas, sem ficarem coladas nas extremidades da tela. Vamos criar margens inserindo um *container* do Bootstrap para centralizar melhor o conteúdo.

```
<html>
<body>

<%= nav_bar brand: "Kunstee", brand_link: root_url %>

<div class="container">
<table class="table table-bordered">
  <thead>
    <tr>
      <th>Nome</th>
      <th>Descrição</th>
      <th>Preço</th>
      <th>Quantidade</th>
    </tr>
  </thead>
  <tbody>
    <% @produtos_por_nome.each do |produto| %>
    <tr>
      <td><%= produto.nome %></td>
      <td><%= produto.descricao %></td>
      <td><%= produto.preco %></td>
      <td><%= produto.quantidade %></td>
    </tr>
    <% end %>
```

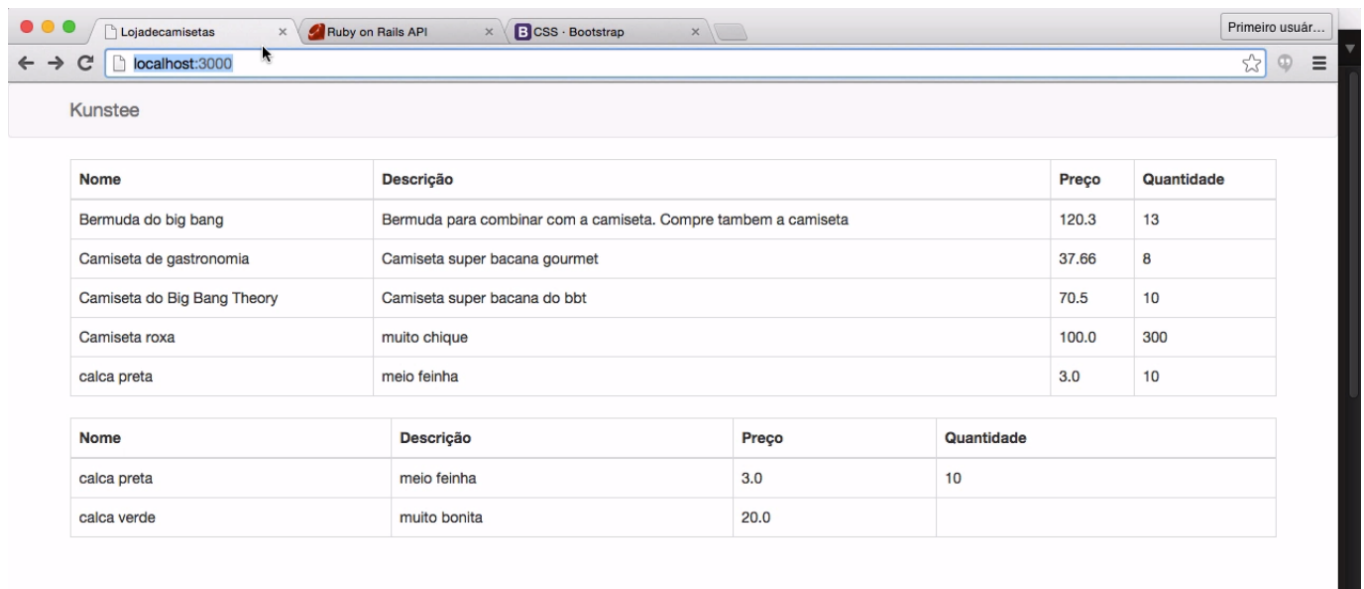
```

    </tbody>
  </table>

  // segunda tabela
</div>

</body>
</html>

```



Porém, precisaremos desse *container* em outras páginas, como na do formulário. Não é uma boa prática ficar simplesmente copiando o comando *container* (ou qualquer outro) porque, muito provavelmente, iremos querer aplicá-lo em todas as páginas, e serão muitas. Isso também serve para a barra de navegação.

Dentro do diretório "app/views/layouts" encontra-se o arquivo "application.html.erb", que já importa, por padrão, todos os CSS's, HTML's e javascripts. Todas as páginas são copiadas para dentro desse arquivo. Então, a primeira coisa que podemos fazer para deixar o código mais limpo é excluir

```

<html>
<body>
e
</body>
</html>

```

de todas as páginas. Também excluímos a *nav_bar* e o *container*, mas estes copiamos para o arquivo "application.html.erb" dessa forma:

```

// parâmetros
<body>

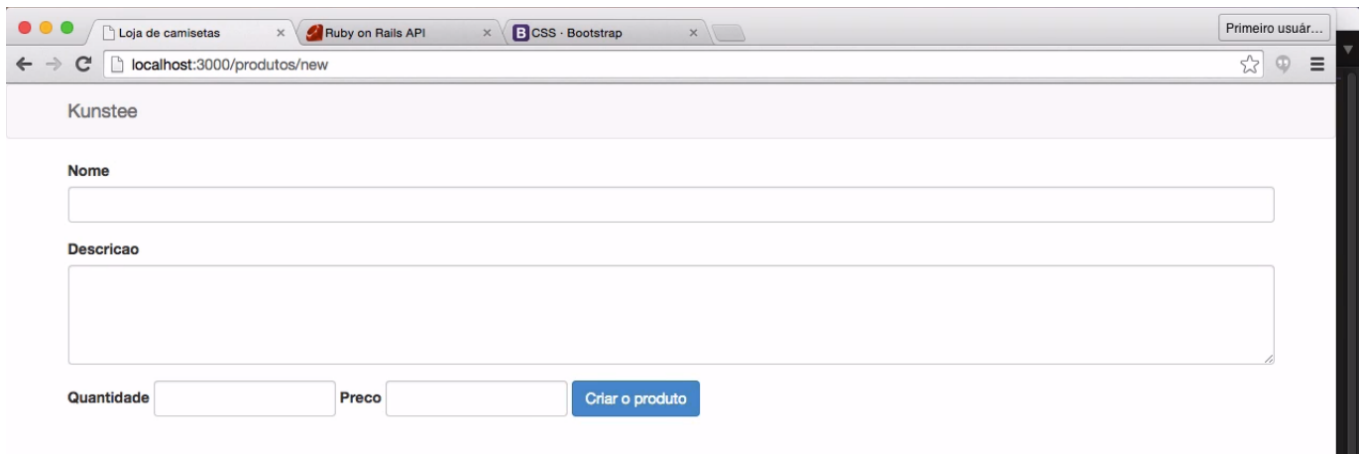
<%= nav_bar brand: "Kunstee", brand_link: root_url %>

<div class="container">
  <%= yield %>
</div>

</body>

```

Assim, não precisaremos nos preocupar com o *layout* padrão de todas as páginas. Até a de formulário foi atualizada:



The screenshot shows a web browser window with three tabs: 'Loja de camisetas', 'Ruby on Rails API', and 'CSS - Bootstrap'. The address bar displays 'localhost:3000/produtos/new'. The page content includes a header 'Kuntee' and a form for creating a new product. The form consists of a 'Nome' field, a 'Descricao' field, and two input fields for 'Quantidade' and 'Preco', followed by a blue button labeled 'Criar o produto'.

Os *layouts* são de extrema importância para não ficarmos repetindo código em todas as páginas e darmos margem a erros.