CAPÍTULO 13

Mais sobre views

13.1 - HELPERS CUSTOMIZADOS

Quando criamos um controller, o Rails automaticamente cria um helper para esse controller em **app/helpers/**. Todo método escrito num helper, estará automaticamente disponível em sua view. Existe um Helper especial, o **application_helper.rb**, cujos métodos ficam disponíveis para todas as views.

13.2 - EXERCÍCIOS: FORMATANDO VALORES MONETÁRIOS

1. Crie um método no nosso Application Helper para converter um número para valor monetário:

a. Abra o arquivo **app/helpers/application_helper.rb**

b. Adicione o seguinte método:

```
def valor_formatado(number)
  number_to_currency number,
    unit: "R$",
    separator: ",",
    delimiter: "."
end
```

c. Em **app/views/qualificacoes/index.html.erb** e **app/views/qualificacoes/show.html.erb**, troque o seguinte código:

```
@qualificacao.valor_gasto
```

por:

```
valor_formatado(@qualificacao.valor_gasto)
```

2. Teste nossa melhoria acessando **http://localhost:3000/qualificacoes/** e

visualizando uma qualificação em específico.

13.3 - Partials e a opção locals

A forma como o usuário cria comentários agora é claramente contra indicativa. O usuário tem que indicar o id do restaurante que irá comentar e que é um restaurante e não uma qualificação.

Para melhorar a usabilidade iremos adicionar uma caixa de texto na página de visualização do restaurante, dessa forma, o usuário poderá comentar o restaurante da própria página do restaurante.

Partial para formulário de comentário

Para criar o formulário teríamos que adicionar o seguinte código erb na view show do restaurante:

```
<%= form_for comentario.new do |f| %>
  <%= f.hidden_field :comentavel_id, value: @restaurante.id %>
  <%= f.hidden_field :comentavel_type, value: @restaurante.class %>

  <%= f.label :conteudo %><br />
  <%= f.text_area :conteudo %>

  <%= f.submit %>
<% end %>
```

Essa alteração serviria para o caso do usuário poder comentar só o restaurante, porém no nosso sistema, uma qualificação também pode ser comentada. Logo, precisaríamos do mesmo formulário na view show de qualificação.

Logo, a solução que nos permitiria reaproveitar o erb do formulário seria extrair o formulário para um partial:

```
<%= form_for comentario.new do |f| %>
  <%= f.hidden_field :comentavel_id, value: @restaurante.id %>
  <%= f.hidden_field :comentavel_type, value: @restaurante.class %>

  <%= f.label :conteudo %><br />
  <%= f.text_area :conteudo %>

  <%= f.submit %>
<% end %>
```

Porém, esse nosso partial faz uso da variável `@restaurante`, que não existe na view **show** de uma qualificação. Precisamos de um partial mais genérico, que se utilize de um comentável qualquer. Porém com o conhecimento que temos até agora isso não é possível.

Partial mais flexível com a opção locals

Para termos nosso partial genérico, iremos utilizar a opção `locals`. Com ela podemos passar algo parecido com um parâmetro. O `locals` é utilizado na chamada do método render.

Nas view **show** de **restaurantes** iremos renderizar o partial da seguinte maneira:

```
<%= render partial: "comentarios/novo_comentario",
           locals: {comentavel: @restaurante} %>
```

Já na view **show** de **qualificacoes** iremos passar a variável `@qualificacao`:

```
<%= render partial: "comentarios/novo_comentario",
           locals: {comentavel: @qualificacao} %>
```

Dessa forma, teremos disponível no nosso partial, uma variável `comentavel` e para utilizar essa variável iremos mudar o conteúdo do partial para:

```
<%= form_for Comentario.new do |f| %>
  <%= f.hidden_field :comentavel_id, value: comentavel.id %>
  <%= f.hidden_field :comentavel_type, value: comentavel.class %>

  <%= f.label :conteudo %><br />
  <%= f.text_area :conteudo %>

  <%= f.submit %>
<% end %>
```

Agora sim, nosso partial está flexível o suficiente para ser utilizado nas duas situações. Poderíamos melhorar ainda mais o código, encapsulando a chamada ao `render` em um helper:

```
def novo_comentario(comentavel)
  render partial: "comentarios/novo_comentario",
         locals: {comentavel: comentavel}
end
```

Dessa forma poderíamos renderizar o partial simplesmente chamando o helper:

```
<%= novo_comentario @restaurante %>
```

Ou no caso da qualificação:

```
<%= novo_comentario @qualificacao %>
```

13.4 - Exercícios: Formulário para criar um novo comentário

Criar comentários usando o formulário padrão é algo pouco usual. Por isso tornaremos possível criar um comentário para um restaurante ou uma qualificação, apartir de sua página de visualização.

Ou seja, ao visualizar um restaurante ou qualificação, poderemos comentá-los através de um formulário.

1. Como utilizaremos o mesmo formulário nas views **show.html.erb** do restaurante e da qualificação, iremos escrever o ERB e uma partial.

a. Crie um novo partial **app/views/comentarios/_novo_comentario.html.erb**.

b. Insira o seguinte conteúdo:

```
<%= form_for Comentario.new do |f| %>
  <%= f.hidden_field :comentavel_id, value: comentavel.id %>
  <%= f.hidden_field :comentavel_type, value: comentavel.class %>

  <%= f.label :conteudo %><br />
  <%= f.text_area :conteudo %>

  <%= f.submit %>
<% end %>
```

2. a. Abra o arquivo **app/helpers/application_helper.rb**

b. Vamos criar um novo método nesse *Helper*. A função desse método será renderizar um *partial* contendo o formulário para criação de novos comentários. Para isso, adicione o método `novo_comentario` na sua classe `ApplicationHelper`:

```
1 def novo_comentario(comentavel)
2   render partial: "comentarios/novo_comentario",
```

```
3           locals: {comentavel: comentavel}
4 end
```

3. Posicionaremos o formulário após as informações do restaurante.

   a. Abra o arquivo `app/views/restaurantes/show.html.erb` e após o último parágrafo adicione a chamada para o nosso novo *Helper*:

```
1 <%= novo_comentario @restaurante %>
```

4. Repita os passos da questão anterior com o arquivo `app/views/qualificacoes/show.html.erb`. Agora usando a variável `@qualificacao`.

5. Para testar iremos criar um comentário pelo formulário e verificar na listagem de comentários se ele foi realmente criado.

   a. Acesse **http://localhost:3000/restaurantes/**.

   b. Entre na página de visualização de um restaurante.

   c. Comente e visualize na listagem de comentários se ele foi criado corretamente.

6. Teste o formulário para uma qualificação também.

13.5 – Partials de coleções

Nosso primeiro passo será possibilitar a inclusão da lista de comentários nas páginas de qualquer modelo que seja comentável. Para não repetir este código em todas as páginas que aceitem comentários, podemos isolá-lo em um partial:

```
<!-- app/views/comentarios/_comentario.html.erb -->
<p>
  <%= comentario.conteudo %> -
  <%= link_to '(remover)', comentario, method: :delete %>
</p>
```

Perceba que não queremos renderizar esse partial uma única vez. Para cada comentário na array, devemos renderizar o partial, no final teremos um HTML com vários parágrafos.

Para fazer algo do tipo, teríamos que ter um ERB como:

```
<% @restaurante.comentarios.each do |comentario| %>
  <%= render partial: "comentarios/comentario",
        locals: {comentario: comentario} %>
```

```
<% end %>
```

Podemos simplificar o código acima, utilizando a opção `:collection`. Dessa maneira, o partial é renderizado uma vez para cada elemento que eu tenha no meu array:

```
<%= render partial: "comentarios/comentario",
      collection: @restaurante.comentarios %>
```

Perceba que nosso objetivo aqui é *renderizar* uma coleção. Estamos usando algumas convenções do *Rails* que irão permitir simplificar ainda mais o código acima. Além de criar uma variável com o o mesmo nome do partial que vamos *renderizar*, no nosso exemplo essa variável será chamada `comentario`, o *Rails* também será capaz de decidir **qual** é o arquivo partial que deve ser utilizado para exibir os itens de uma coleção.

Como temos uma coleção de **comentários** para exibir, basta invocar o método `render` passando a coleção como parâmetro, e o *Rails* cuidará do resto. Assim será possível omitir tanto o nome do *partial* como o *hash* que passamos como parâmetro anteriormente. Vamos fazer essa simplificação em nosso código e além disso garantir que a coleção seja *renderizada* apenas se não estiver vazia, dessa forma:

```
<%= render @restaurante.comentarios %>
```


13.6 - Exercícios: Listando os comentários de um comentável

Assim como o formulário, a listagem tem que ser implementada de forma que seja reaproveitada nas views dos restaurantes e das qualificações, através de

partials.

1. Crie o arquivo **app/views/comentarios/_comentario.html.erb** com o seguinte conteúdo:

```
<p>
  <%= comentario.conteudo %> -
  <%= link_to '(remover)', comentario, method: :delete %>
</p>
```

2. a. Crie o arquivo **app/views/comentarios/_comentarios.html.erb**.

b. Seu conteúdo deve ser:

```
<div id="comentarios">
  <h3>Comentários</h3>
  <% if comentarios.empty? %>
    Nenhum comentário
  <% else %>
    <%= render comentarios %>
  <% end %>
</div>
```

3. a. Abra o arquivo **app/helpers/application_helper.rb**

b. Vamos adicionar um helper para agilizar a renderização do partial da listagem de comentários:

```
def comentarios(comentavel)
  render partial: "comentarios/comentarios",
    locals: {comentarios: comentavel.comentarios}
end
```

4. a. Abra a view de show dos restaurantes **app/views/restaurantes/show.html.erb**.

b. Antes da chamada para `novo_comentario` adicione a seguinte linha:

```
<%= comentarios @restaurante %>
```

c. Repita o exercício acima para a views de show das qualificacoes **app/views/qualificacoes/show.html.erb**. Lembrando de usar `@qualificacao` ao invés de `@restaurante`.

5. Após essas alterações deve ser possível ver os comentários na página de visualização de um restaurante ou de uma qualificação. Inicie o servidor e verifique.

13.7 - Layouts

Como vimos, quando criamos um Partial, precisamos declará-lo em todas as páginas que desejamos utilizá-los. Existe uma alternativa melhor quando desejamos utilizar algum conteúdo estático que deve estar presente em todas as páginas: o **layout**.

Cada controller pode ter seu próprio layout, e uma alteração nesse arquivo se refletirá por todas as views desse controller. Os arquivos de layout devem ter o nome do controller, por exemplo **app/views/layouts/restaurantes.html.erb**.

Um arquivo de layout "padrão" tem o seguinte formato:

```
<html>
  <head>
    <title>Um título</title>
  </head>
  <body>
    <%= yield %>
  </body>
</html>
```

Tudo o que está nesse arquivo pode ser modificado, com exceção do `<%= yield %>`, que renderiza cada view do nosso controlador.

Podemos utilizar ainda o layout **application.html.erb**. Para isso, precisamos criar o arquivo **app/views/layouts/application.html.erb** e apagar os arquivos de layout dos controladores que queremos que utilizem o layout do controlador application. Com isso, se desejarmos ter um layout único para toda nossa aplicação, por exemplo, basta ter um único arquivo de layout.

13.8 - Exercícios: Criando um menu de navegação principal

1. Vamos agora criar um menu no nosso **Layout**, para poder navegar entre Restaurante, Cliente e Qualificação sem precisarmos digitar a url:

   a. Abra o layout mais genérico: **"app/views/layouts/application.html.erb"**

   b. Dentro do `<body>` e antes do `yield` digite o ERB da lista de opções do menu:

```
<body>
  <ul>
    <% %w[clientes qualificacoes restaurantes].each do |controller_name| %>
      <li>
```

```
        <%= link_to controller_name, controller: controller_name %>
      </li>
    <% end %>
  </ul>

  <%= yield %>
</body>
```

2. Teste a navegação pelo menu: http://localhost:3000/restaurantes

13.9 - Exercícios Opcionais: Menu como partial

1. Seria interessante podermos criar outros menus no decorrer do nosso sistema, para facilitar é necessário que extraiamos o ERB do menu para um **Partial**.

a. Crie o partial "menu_principal": **"app/views/layouts/_menu_principal.html.erb"**

b. Digite o seguinte ERB:

```
<ul>
  <% opcoes.each do |controller_name| %>
    <li>
      <%= link_to controller_name, controller: controller_name %>
    </li>
  <% end %>
</ul>
```

2. Perceba que a variável `opcoes` não existe, isso ocorre, por que vamos dizer qual será o valor dessa variável ao renderizar o partial, utilizando a opção `locals`:

a. Abra o layout mais genérico: (**app/views/layouts/application.html.erb**)

b. Substitua a lista de opções por uma chamada ao método `<%=render %>`, renderizando o partial **_menu_principal.html.erb**, dessa maneira:

```
...
<body>
  <%= render "menu_principal",
    locals: {opcoes: %w(restaurantes clientes qualificacoes)} %>

  <%= yield %>
</body>
...
```

3. Teste a navegação pelo menu: **http://localhost:3000/restaurantes**

4. Para simplificar a renderização do partial, podemos criar um helper que irá fazer esse trabalho:

a. Abra o Helper "applications": **"app/helpers/application_helper.rb"**

b. Digite o seguinte método:

```
def menu_principal(controllers)
  render partial: "menu_principal", locals: {opcoes: controllers}
end
```

5. a. Abra o layout mais genérico: (**app/views/layouts/application.html.erb**)

b. Substitua a chamada ao método `<%=render %>` por uma chamada ao helper **menu**:

```
...
<body>
  <%= menu_principal %w(restaurantes clientes qualificacoes) %>

  <%= yield %>
</body>
...
```

13.10 - Aplicar CSS em apenas algumas views

Nossa listagem de comentários está com uma aparência muito desagradável, por isso iremos aplicar um CSS especial nas páginas que utilizarem nosso partial com os comentários.

É boa prática no desenvolvimento front-end, que importemos o CSS sempre dentro da tag `<head>`, porém se importarmos o CSS no **application.html.erb** esse CSS será importado desnecessariamente em todas as views de nossa aplicação.

Precisamos implementar nossas views de forma que a importação do nosso novo CSS seja feita somente se o partial **_comentarios.html.erb** for utilizado. Para fazer isso, teremos que utilizar o `yield` de uma forma diferente.

Primeiramente, iremos adicionar uma chamada ao `yield :css` dentro de **app/views/layouts/application.html.erb**, fazendo isso, estaremos tornando possível para a view inserir um conteúdo chamado `:css` naquela posição do layout. Nosso **application.html.erb** deve conter o seguinte ERB:

```
<head>
  <!-- (...) -->

  <%= stylesheet_link_tag "application", :media => "all" %>
  <%= yield :css %>

  <!-- (...) -->
</head>
```

Fazendo isso, tornamos possível que uma view ou partial possa inserir um conteúdo chamado `:css`. Agora precisamos fazer com que o partial **_comentarios.html.erb** defina um conteúdo `:css` importando o nosso CSS especial. Para isso, iremos chamar o método `content_for` em qualquer lugar do partial, dessa forma:

```
<% content_for :css do %>
  <link rel="stylesheet" href="/stylesheets/comentarios.css">
<% end %>
```

Após adicionar essas duas chamadas, nosso CSS será aplicado somente nas views que utilizam o partial **_comentarios.html.erb**.

13.11 - Exercícios: Aplicando CSS para os comentários

Vamos configurar nosso layout e partials de forma que o CSS que irá estilizar os comentários só seja aplicado nas views necessárias:

1. Como nosso foco de estudo é o Rails, iremos utilizar um CSS que já está pronto. Entre na pasta **caelum** que está em seu Desktop e copie o arquivo: **caelum/71/comentarios.css** para dentro do nosso projeto **vota_prato/public/stylesheets/comentarios.css**.

2. Agora que temos o arquivo **public/stylesheets/comentarios.css** em nosso projeto vamos, usar o Rails para que esse CSS seja aplicado somente nas views onde o partial **_comentarios.html.erb** for renderizado:

   a. Abra o partial **app/views/comentarios/_comentarios.html.erb**.

   b. Ao final do arquivo, vamos invocar o helper `content_for` envolvendo a tag `link`

associada ao nosso novo CSS:

```
<% content_for :css do %>
  <link rel="stylesheet" href="/stylesheets/comentarios.css">
<% end %>
```

3. Vamos agora indicar em que parte do layout devem ser inseridos os conteúdos referenciados pelo símbolo `:css`.

a. Vamos abrir o layout mais genérico (**app/views/layouts/application.html.erb**).

b. Adicione uma chamada ao `yield` dentro do `<head>` e logo abaixo da chamada ao helper `stylesheet_link_tag`:

```
<%= stylesheet_link_tag "application", media: "all" %>
<%= yield :css %>
```

4. Teste o novo estilo da listagem de comentários acessando: **http://localhost:3000/restaurantes/** e visualizando a página de um restaurante.

CAPÍTULO ANTERIOR:

Associações Polimórficas

PRÓXIMO CAPÍTULO:

Ajax com Rails

Você encontra a Caelum também em:

Blog Caelum

Cursos Online

Facebook

Newsletter

Casa do Código

Twitter