CAPÍTULO 14

Ajax com Rails

"O Cliente tem sempre razão"
— Selfridge, H.

Nesse capítulo, você verá como trabalhar com AJAX de maneira não obstrusiva no Rails.

14.1 – UTILIZANDO AJAX PARA REMOÇÃO DE COMENTÁRIOS

Se observarmos a listagem de comentários perceberemos que a experiência do usuário não é agradável, após remover um comentário o usuário é redirecionado para a listagem de comentários. Podemos utilizar AJAX para uma experiência mais marcante no uso do site pelos usuários.

link_to realizando requisições AJAX

Ao remover um comentário, queremos que o usuário continue na página, porém iremos remover o parágrafo do comentário excluido. Primeiramente, teremos que fazer com que o link **(remover)** realize uma requisição AJAX ao invés de uma requisição convencional, impedindo assim que o usuário seja lançado para outra página.

Ao abrirmos o nosso partial **app/views/comentarios/_comentario.html.erb** veremos que há uma chamada ao helper `link_to`, que é responsável por criar o link **(remover)**. Esse helper, já nos disponibiliza uma opção para realização de uma requisição AJAX. Para obter esse comportamento, devemos utilizar a opção `remote: true`, da seguinte forma:

```
1 <!-- /app/views/comentarios/_comentario.html.erb -->
2 <p>
3   <%= comentario.conteudo %> -
4   <%= link_to '(remover)', comentario, method: 'delete',
5         remote: true %>
```

```
6 </p>
```

Após realizar essa alteração, poderemos remover um comentário sem sair da página atual. Porém o comentario em si continuará na tela. Se recarregarmos a página o comentário não aparecerá pois ele realmente foi excluido no banco de dados.

Utilizando JavaScript o parágrafo do comentário excluido

A action `ComentariosController#destroy` está sendo chamada de forma assíncrona (*Ajax*), porém a página não foi atualizada. Por isso precisaremos utilizar um código JavaScript que irá apagar o comentário logo após a action **destroy** responder a requisição AJAX com sucesso.

É uma boa prática no desenvolvimento front-end que importemos os nossos JavaScripts sempre antes de `</body>`. Para fazer isso, utilizaremos uma solução muito parecida com a do CSS, utilizando `yield` e `content_for`.

Vamos começar alterando o **app/views/layouts/application.html.erb** para que as nossas views e partials possam inserir um conteúdo JavaScript antes do **</body>**

```
1 <!-- /app/views/layouts/application.html.erb -->
2
3   <!-- (...) -->
4
5     <script type="text/javascript">
6       <%= yield :js %>
7     </script>
8   </body>
9 </html>
```

Utilizaremos um código JavaScript que irá apagar o comentário assim que a requisição AJAX for respondida como bem sucedida:

```
$('#remove_comentario_<%= comentario.id %>').bind('ajax:success', function(){
  $('#comentario_<%= comentario.id %>').remove();
}
);
```

Agora vamos alterar o arquivo *app/views/comentarios/_comentario.html.erb* para colocar esse código JavaScript no lugar do 'yield :js'.

```
<p id="comentario_<%= comentario.id %>">
  <%= comentario.conteudo %>
  <%= link_to 'remover', comentario, method: :delete,
          remote: true,
          id: "remove_comentario_#{comentario.id}"%>
```

```
</p>

<% content_for :js do%>
  $('#remove_comentario_<%= comentario.id %>').bind('ajax:success',
function(){
    $('#comentario_<%= comentario.id %>').remove();
  });
<%end%>
```

Perceba que também adicionamos **id** no parágrafo e no link, adicionamos esse atributo pois o nosso código JavaScript necessita que cada comentário tenha um parágrafo com id **comentario_ID-DO-COMENTARIO** e um link com id **remove_comentario_ID-DO-COMENTARIO**.

Dessa forma, teremos o HTML gerado será algo como:

```
<div id='comentarios'><h3><Comentarios></h3>
  <p id="comentario_1">
    comentario 1
    <a href="/comentarios/1" data-method="delete" data-remote="true"
      id="remove_comentario_1" rel="nofollow">remover</a>
  </p>

  <p id="comentario_2">
    Comentario 2
    <a href="/comentarios/2" data-method="delete" data-remote="true"
      id="remove_comentario_2" rel="nofollow">remover</a>
  </p>

  <!-- Continuação do HTML -->

<script>
  $('#remove_comentario_1').bind('ajax:success', function(){
    $('#comentario_1').remove();
  });

  $('#remove_comentario_2').bind('ajax:success', function(){
    $('#comentario_2').remove();
  });
</script>
```

Após implementar, nossa página deve conter o código JavaScript ao final do código fonte da página. Para verificar, abra a página de visualização de um restaurante qualquer, pressione **CTRL + U** e verifique no código fonte da página se o nosso novo código JavaScript foi adicionado antes da tag body.

Mesmo com o código JavaScript sendo adicionado corretamente, o parágrafo do comentário ainda não será removido. Isso ocorrerá pois nosso código JavaScript assume que a resposta da requisição AJAX será de sucesso quando na verdade, ela é uma mensagem de falha pois não preparamos nosso controller para receber uma requisição AJAX.

Preparando o controller para uma requisição AJAX

Quando uma requisição AJAX é recebida pela action `ComentariosController#destroy`, a mesma é tratada como de formato **js** ao invés do convencional **html**.

Portanto, para preparar nossa action **destroy** basta adicionarmos um novo formato ao `respond_to` que só responda como bem sucedida. Para isso utilizaremos o método `head` passando o simbolo `:ok`:

```
1 def destroy
2   @comentario = Comentario.find(params[:id])
3   @comentario.destroy
4
5   respond_to do |format|
6     format.js { head :ok }
7   end
8 end
```

Após realizar todas essas alterações, poderemos finalmente desfrutar de um link remover que além de excluir no banco de dados, irá remover dinamicamente o parágrafo do comentário.

> **Para saber mais: AJAX em versões antigas do Rails**
>
> O Rails, antes da versão 3, tentava facilitar o desenvolvimento de código javascript com recursos como o RJS Templates, que produzia código javascript a partir de código Ruby.
>
> Contudo, o código gerado era acoplado ao Prototype, o que dificultava o uso de outras bibliotecas populares como o jQuery. No Rails 3 optou-se por essa nova forma, não obstrutiva, de se trabalhar com javascript, permitindo que os desenvolvedores tenham controle absoluto do código criado, e podendo, inclusive, escolher a biblioteca que será usada (Prototype, jQuery, Moo-tools ou qual quer outra, desde que um driver exista para essa biblioteca).
>
> Outra mudança interessante que mostra que o *Rails* é um framework em constante evolução foi a adoção do framework *jQuery* como padrão a partir da versão 3.1.

14.2 - Exercícios: Link de remover comentário usando AJAX

1. Vamos configurar o link **remover** de forma que ele realize uma requisição AJAX.

   a. Abra o partial que renderiza um único comentário (**app/views/comentarios/_comentario.html.erb**):

   b. Adicione a opção **remote** com o valor **true** na chamada ao `link_to`:

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

2. Apesar de ser feita uma requisição AJAX, o comentário continua aparecendo. Vamos adicionar um código JavaScript que irá apagar o parágrafo quando o link for clicado:

   a. Abra novamente o partial que renderiza um único comentário (**app/views/comentarios/_comentario.html.erb**):

   b. Altere-o adicionando o código JavaScript e os **id**s que ele necessita. Ao final, o partial deve estar parecido com o código abaixo:

```
<p id="comentario_<%= comentario.id %>">
  <%= comentario.conteudo %> -
  <%= link_to '(remover)', comentario, method: :delete,
        remote: true,
        id: "remove_comentario_#{comentario.id}" %>
</p>

<% content_for :js do %>
  $('#remove_comentario_<%= comentario.id %>').bind('ajax:success',
    function(){
      $('#comentario_<%=comentario.id%>').remove();
    }
  );
<% end %>
```

3. Como queremos que todo o JavaScript esteja posicionado antes do `</body>` é necessário que alteremos o layout `application.html.erb`.

   a. Abra o arquivo **app/views/layouts/application.html.erb**.

   b. Adicione as seguintes linhas logo abaixo do `<%= yield %>`:

```
<script>
  <%= yield :js %>
</script>
```

4. Precisamos também preparar a action **destroy** do nosso controller de

comentários para responder à requisições AJAX.

a. Altere as seguintes linhas da action **destroy** do controller **comentarios_controller.rb**

```
def destroy
  @comentario = Comentario.find(params[:id])
  @comentario.destroy

  respond_to do |format|
    format.js { head :ok }
  end
end
```

5. Teste em **http://localhost:3000/restaurantes**, selecionando um restaurante e removendo um de seus comentários.

14.3 - Adicionando comentários dinamicamente

Ao criar um novo comentário, fica claro que temos o mesmo problema que tinhamos no link **(remover)**. Após criar um comentário somos redirecionados para uma outra página. Portanto resolveremos esse problema fazendo com que o formulário envie uma requisição AJAX.

Formulário enviando requisições AJAX

Assim como o `link_to`, o helper `form_for` também tem a opção `remote` para ser utilizadas em formulários que devem realizar requisições AJAX.

Logo, precisaremos realizar uma pequena alteração no nosso partial **app/views/comentarios/_novo_comentario.html.erb** para que o formulário do

mesmo passe a realize requisições AJAX:

```
<!-- /app/views/comentarios/_novo_comentario.html.erb -->
<%= form_for Comentario.new, remote: true do |f| %>

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

Após utilizar a opção `remote: true`, conseguiremos criar um comentário, porém o mesmo não aparecerá na listagem de comentários.

JavaScript para atualizar a lista

Para atualizar a lista após a resposta da requisição AJAX iremos utilizar o seguinte código JavaScript:

```
$('form').bind('ajax:complete', function(){
  $('#comentarios').replaceWith(result.responseText);
  $('textarea').val("");
});
```

Vamos nos aproveitar do `yield :js` que já foi inserido no layout, ou seja, só precisaremos adicionar a chamada ao método `content_for` em qualquer local do nosso partial **app/views/comentarios/_novo_comentario.html.erb**:

```
<% content_for :js do %>
  $('form').bind('ajax:complete', function(){
    $('#comentarios').replaceWith(result.responseText);
    $('textarea').val("");
  });
<% end %>
```

Após realizar essas alterações verificaremos que ao tentar criar um comentário, a listagem de comentários não é atualizada. Ao invés disso, a listagem é substituida por uma outra página. Isso ocorrerá, pois nosso código JavaScript substitui a listagem pela resposta da requisição.

Isso ocorre pois nossa action `ComentariosController#create` não está preparada para responder com a listagem de comentários atualizada.

Preparando view para requisição AJAX

Até agora trabalhamos com arquivos `html.erb`, que são utilizados na resposta para requisições do formato **html**.

Porém, nossa action `ComentariosController#create` precisa responder com um conteúdo para o formato **js** que é o utilizado para requisições AJAX. Para fazer isso, iremos criar um arquivo de extensão **js.erb**. Como a action que irá responder é a

create iremos criar o arquivo **app/views/comentarios/create.js.erb** que irá simplesmente renderizar a listagem de comentários novamente:

```
<%= comentarios @comentario.comentavel %>
```

Note que nossas views anteriores utilizavam o layout **application.html.erb**, porém nossa nova view irá utilizar o layout de acordo com seu próprio formato, ou seja, o arquivo **app/views/layouts/application.js.erb**. Como a resposta terá somente a lista e nada mais, nosso novo arquivo de layout deverá ter somente duas chamadas para `yield`. Uma para renderizar o conteúdo da view e a outra para renderizar o código JavaScript:

```
<%= yield %>

<script>
  <%= yield :js %>
</script>
```

Respondendo AJAX com as views preparadas

Após criar as novas views, precisaremos declarar na action **create** que ela deve dar suporte ao formato **js**. Para isso, basta adicionar uma chamada à `format.js` dentro do `respond_to`:

```
def create
  @comentario = Comentario.new comentario_params

  respond_to do |format|
    if @comentario.save
      format.js
    end
  end
end
```

Após realizar as alterações acima, poderemos criar comentários sem ter que esperar a página inteira ser carregada. Pois a listagem de comentários será atualizada dinamicamente.

14.4 – Exercícios: AJAX no formulário de comentários

1. Primeiramente iremos configurar nosso formulário para criação de comentários de forma que ele realize requisições AJAX:

   a. Abra o partial do formulário para criação de comentários: (**app/views/comentarios/_novo_comentario.html.erb**)

b. Na chamada do `form_for` utilize a opção `remote` como `true`:

```
<%= form_for Comentario.new, remote: true do |f| %>
```

c. No final do partial, adicione o código JavaScript que irá lidar com a resposta da requisição AJAX:

```
<% content_for :js do %>
  $('form').bind('ajax:complete', function(xhr, result){
    $('#comentarios').replaceWith(result.responseText);
    $('textarea').val("");
  });
<% end %>
```

2. Para finalizar o processo temos que alterar o nosso controller de comentários para responder com a lista de comentários do nosso comentável.

a. Altere o método create no **app/controllers/comentarios_controller.rb**

```
if @comentario.save
  format.js
else
```

b. E agora crie o arquivo **views/comentarios/create.js.erb**

```
<%= comentarios @comentario.comentavel %>
```

c. Para que nosso partial exiba corretamente os javascripts precisamos criar um layout para colocar o bloco de yield dos javascripts.

d. Crie o arquivo **app/views/layouts/application.js.erb** com o seguinte conteúdo

```
<%= yield %>

<script>
  <%= yield :js %>
</script>
```

e. Agora já é possível cadastrar um comentário direto do show dos comentáveis

CAPÍTULO ANTERIOR:

Mais sobre views

PRÓXIMO CAPÍTULO:

Algumas Gems Importantes

Você encontra a Caelum também em:

Blog Caelum

Cursos Online

Facebook

Newsletter

Casa do Código

Twitter