02

Adicionando comentários via AJAX

Adicionando comentários via AJAX

No último capítulo aprendemos a identificar e solucionar os problemas de performance mais comuns em aplicações Rails: *consultas N+1* e *counter caches*. Terminamos discutindo sobre comentários na aplicação, e hoje vamos continuar falando sobre eles, tornando o envio de comentários mais dinâmico utilizando AJAX. Vamos lá!

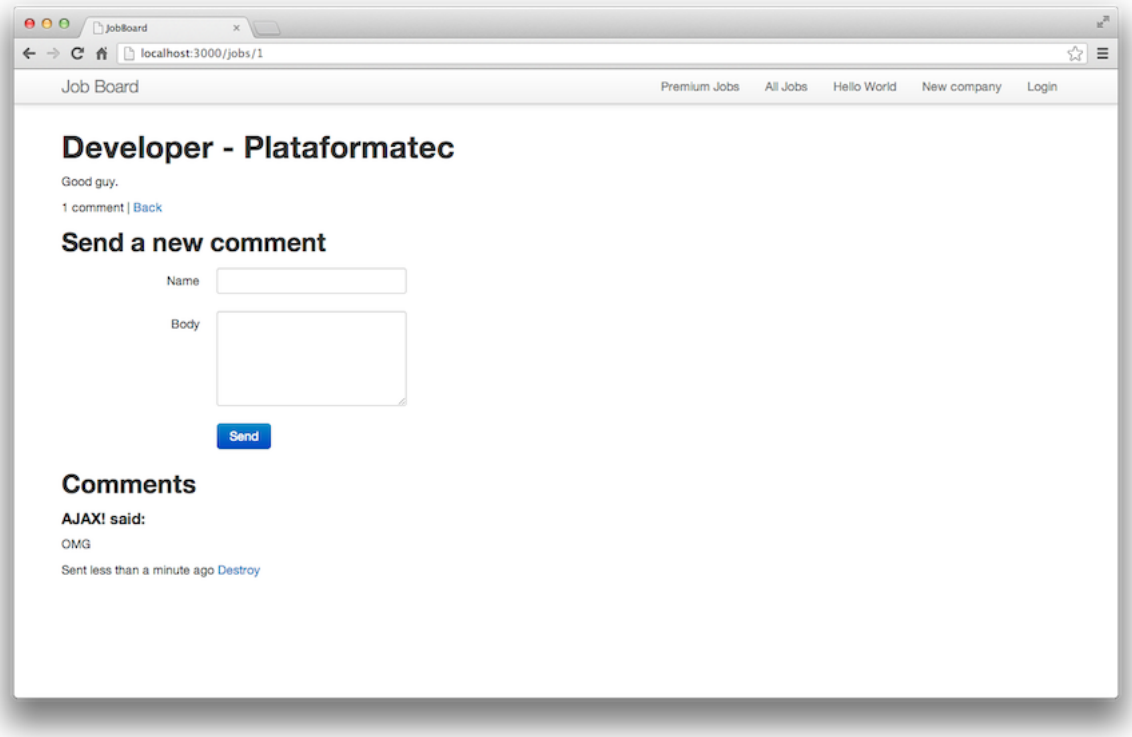
Formulários remotos

O principal motivo para tornar o formulário de envio de comentário mais dinâmico é a melhoria na usabilidade da aplicação para o usuário final, já que a página não será recarregada a cada comentário enviado. Desta forma, o usuário tem uma percepção de que a aplicação responde muito mais rapidamente, enquanto utilizamos AJAX através de JavaScript para enviar os dados do formulário para o servidor.

Nosso primeiro passo será indicar para o Rails que o formulário de comentários deverá ser submetido via AJAX ao invés de uma requisição normal. Abra o formulário de envio de comentários em *app/views/comments/_form.html.erb*, e adicione a opção `remote: true` ao `form_for`:

```
<%= form_for [@job, Comment.new], html: { class: 'form-horizontal' }, remote: true do |f| %>
```

Vamos testar o que acontece pelo navegador, tenha a certeza de estar rodando o servidor do Rails e acesse a página de um *job*, como *http://localhost:3000/jobs/1 (http://localhost:3000/jobs/1)*. Preencha os campos do comentário e clique em `Send`: aparentemente nada aconteceu, certo? Agora recarregue a página e olhe para o final da lista de comentários: seu novo comentário deverá estar aparecendo lá!



Se checarmos o log do servidor, veremos que foi recebida uma requisição `POST` em `/jobs/1/comments`, para criar o novo comentário:

```
Started POST "/jobs/1/comments" for 127.0.0.1 at 2013-02-19 10:37:35 -0300
Processing by CommentsController#create as JS
  Parameters: {"utf8"=>"", "authenticity_token"=>"q9MdFOkB8fSefI9tkIA0dO1oOZw+Z+byKao5M4Yufqs="
  Job Load (0.1ms)  SELECT "jobs".* FROM "jobs" WHERE "jobs"."id" = ? LIMIT 1  [["id", "1"]]
   (0.0ms)  begin transaction
  SQL (0.5ms)  INSERT INTO "comments" ("body", "created_at", "job_id", "name", "updated_at") VAI
  Job Load (0.1ms)  SELECT "jobs".* FROM "jobs" WHERE "jobs"."id" = 1 LIMIT 1
  SQL (0.1ms)  UPDATE "jobs" SET "comments_count" = COALESCE("comments_count", 0) + 1 WHERE "jol
   (6.4ms)  commit transaction
Redirected to http://localhost:3000/jobs/1
Completed 302 Found in 14ms (ActiveRecord: 7.2ms)
```

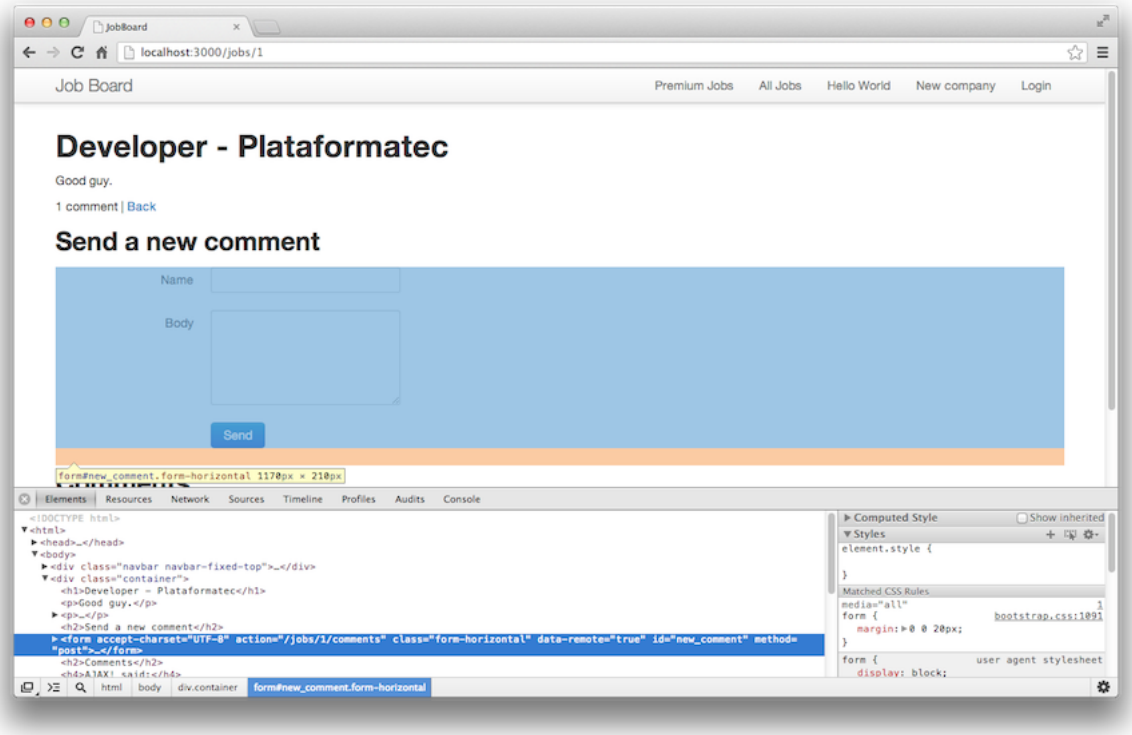
Perceba que a segunda linha do log, `Processing by CommentsController#create as JS`, identifica que o Rails reconheceu essa requisição como `JS`, ou seja, através de uma requisição AJAX, e criou o comentário no banco de dados normalmente. Maravilha! Não tivemos que escrever nenhum JavaScript para que nosso formulário seja automaticamente enviado via AJAX. Mas fica a dúvida: como isso funciona?

Rails, jQuery, e Unobtrusive JavaScript

O Rails já possui algumas funcionalidades integradas com JavaScript para tornar nossa vida mais fácil, e uma delas é o envio de formulários através de requisições AJAX apenas utilizando a opção `remote: true`.

Para entendermos um pouco melhor como isso funciona por baixo dos panos, vamos utilizar o painel de desenvolvedor do Google Chrome como no capítulo 2. Acesse a mesma página do *job* que acessamos antes, agora pelo Chrome, e abra o painel (**Ctrl + Shift + C** para PC, ou **Cmd + Option + C** para Mac). Clique na aba *Elements*, que mostra o HTML completo da página da forma como ele é visto pelo navegador, e então localize o botão com uma lupa na parte inferior do painel: este botão serve para inspecionar um elemento, ou seja, encontrar este elemento na página para que possamos verificar seu HTML. Clique na lupa, e então mova o cursor do mouse para a página: conforme você for movendo o cursor sobre os elementos, eles serão selecionados, e você pode clicar para inspecionar o elemento. Localize o formulário de envio de comentários e clique nele, ele deverá ser marcado no HTML sendo exibido na aba *Elements*, e deve ser algo como:

```
<form accept-charset="UTF-8" action="/jobs/1/comments" class="form-horizontal" data-remote="true
```



Note que existe um atributo chamado `data-remote` com o valor `true`. Este atributo é adicionado pelo Rails quando passamos a opção `remote: true` para o formulário, e é através dessa indicação que o Rails sabe que deve manipular o envio deste formulário por AJAX. Mas como essa manipulação é feita?

O Rails utiliza por padrão a biblioteca *jQuery* (http://jquery.com/), muito utilizada em aplicações web, para manipulação de elementos na página através de JavaScript. Além disso, o Rails possui um JavaScript próprio, especialmente escrito para se integrar com o *jQuery*, que adiciona funcionalidades como o envio de formulários com AJAX através da opção `remote`. Esse JavaScript próprio do Rails é conhecido como o **adaptador jQuery para Rails**, ou simplesmente **jquery-rails** (http://github.com/rails/jquery-rails), e existem outros adaptadores que se integram com bibliotecas diferentes.

Abra o *Gemfile* e você verá uma linha que contém a *gem* `jquery-rails`, que traz o *jQuery* e o JavaScript do Rails para a nossa aplicação. Você pode conferir essas duas bibliotecas sendo importadas no arquivo *app/assets/javascripts/application.js*, que contém as linhas:

```
//= require jquery
//= require jquery_ujs
```

Esse arquivo adiciona o *jquery* e o *jquery_ujs* (o adaptador) à nossa aplicação, e é referenciado no *layout* *app/views/layouts/application.html.erb*, ao final da página:

```
<%= javascript_include_tag "application" %>
```

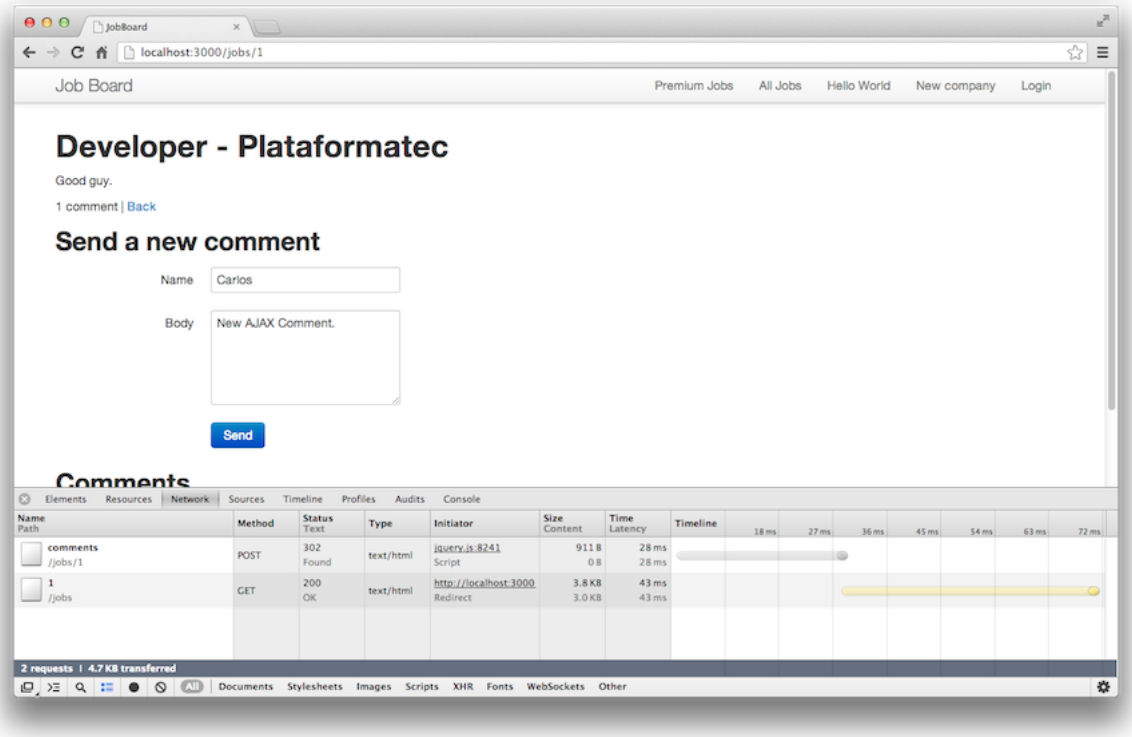
Isto inclui o *application.js* em todas as nossas páginas, e com isso, quando a página do *job* é exibida, o JavaScript do Rails identifica que nosso formulário possui o atributo `data-remote=true`, e altera o envio dele para que seja feito através de AJAX. Desta forma, o Rails não precisa encher o HTML com código JavaScript para manipular o formulário, pois o próprio JavaScript se encarregará de integrar a funcionalidade nos formulários que forem identificados com a opção `data-remote`. Essa técnica é conhecida como **Unobtrusive JavaScript**, e permite que mantenhamos nossas páginas mais limpas contendo somente HTML, e utilizemos JavaScript para adicionar funcionalidades que melhoram a

usabilidade para o usuário final. Também é devido ao título **Unobtrusive JavaScript** que o nome do arquivo JavaScript do Rails é *jquery_ujs*.

Ótimo, agora que entendemos como o Rails modificou o envio do nosso formulário para usar AJAX, precisamos dar feedback para o usuário de que o comentário foi enviado com sucesso ou se alguma validação falhou, caso contrário ele pode pensar que a aplicação está com problemas e seu comentário não está sendo enviado.

Respondendo à requisições AJAX no controller com respond_to

Antes de fazermos qualquer alteração, precisamos descobrir como o Rails está respondendo à nossa requisição AJAX. Abra a aba *Network* no painel do Chrome, que exibe todas as requisições que a página executa, e então envie um novo comentário através do formulário. Você deverá ver duas requisições acontecendo: a primeira é o envio do comentário em si, uma requisição *POST* para `/jobs/1/comments`, que foi completada e redirecionou para a página do *job*, `/jobs/1`, causando a segunda requisição.



Perfeito, a requisição AJAX está acontecendo, porém temos essa segunda requisição que é necessária quando enviamos o comentário normalmente, mas se faz desnecessária quando enviamos via AJAX. Isto quer dizer que o Rails está recebendo com sucesso a requisição AJAX como observamos no log, porém ele não sabe responder a essa requisição de maneira específica, então ele responde da mesma forma que uma requisição normal, causando o redirecionamento.

Vamos tratar isso adicionando um bloco `respond_to` à action responsável por criar comentários, que permitirá definirmos uma resposta específica para a requisição AJAX, dessa forma evitando esse redirecionamento desnecessário. Você deve lembrar de ter visto o `respond_to` em outros *controllers*, pois ele é automaticamente gerado pelo Rails através do *scaffold*. Por exemplo, abra o *app/controllers/jobs_controller.rb* e localize a *action index*:

```
def index
  @jobs = Job.most_recent.includes(:company).all

  respond_to do |format|
    format.html # index.html.erb
```

```
      format.json { render json: @jobs }
    end
  end
```

O `respond_to` recebe um bloco com o argumento `format`, que neste caso responde para `html` (o padrão quando não há `respond_to`), e também `json`. No caso de comentários, vamos responder para `html` da mesma forma que respondemos atualmente, e também para `js` de *JavaScript*, para manipularmos a resposta à requisição AJAX. Dessa forma garantimos que o envio de comentários funcionará como antes quando não for possível utilizar JavaScript. Abra o *app/controllers/comments_controller.rb*, e altere a *action create* conforme abaixo:

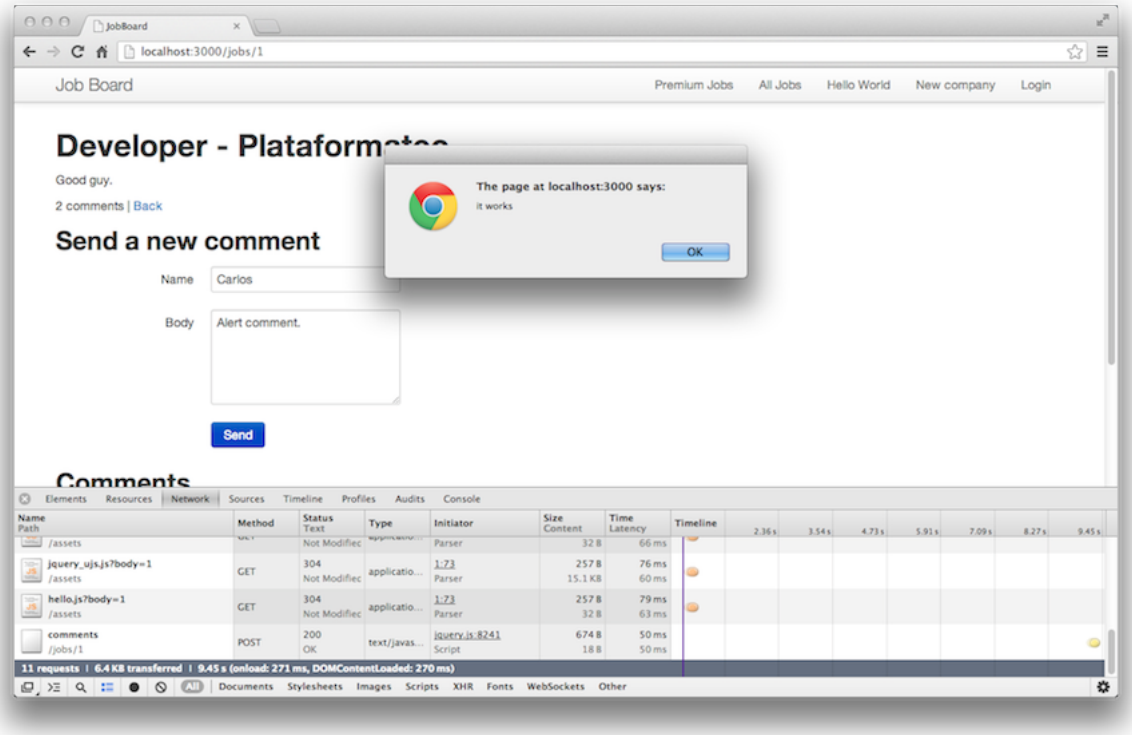
```
def create
  @job = Job.find(params[:job_id])
  @comment = @job.comments.build(params[:comment])

  respond_to do |format|
    format.html do
      if @comment.save
        flash[:notice] = "Comment was created with success!"
      else
        flash[:alert] = "Please fill in all fields to create a comment."
      end
      redirect_to @job
    end
    format.js do
      @comment.save
    end
  end
end
```

Mantemos a resposta `html` com o código anterior que vai efetuar o *redirect*, e no caso da requisição AJAX, apenas salvamos o comentário. Porém, apenas salvar o comentário evitando o redirecionamento não é suficiente: precisamos dar feedback para o usuário se o comentário foi criado com sucesso ou não. Para isso, vamos criar uma nova *view* em *app/views/comments*, chamada *create.js.erb*, que será automaticamente renderizada pela nossa *action create* quando responder por `js`, com o seguinte conteúdo para fins de teste:

```
alert('it works!');
```

Perceba que o nome da *view* contém `.js` ao invés de `.html`, pois esse é o formato da requisição que estamos respondendo. Note também que escrevemos JavaScript nesta *view* e não HTML, pois é o formato de resposta esperado pela requisição AJAX, e dessa forma conseguiremos escrever código JavaScript para manipular o conteúdo da página. Vamos testar isto no navegador, atualize a página do *job* e tente enviar um comentário preenchendo todos os campos: você deverá receber a mensagem `alert`, indicando que tudo funcionou com sucesso!



Perfeito! Basta agora escrevermos o JavaScript necessário para dar algum feedback ao usuário. Modifique o conteúdo da *view* conforme abaixo:

```
<% if @comment.errors.empty? %>
  $('#comments').append('<%=j render(@comment) %>');
  $('#new_comment')[0].reset();
  alert('Comment was created with success!');
<% else %>
  alert('Please fill in all fields to create a comment.');
<% end %>
```

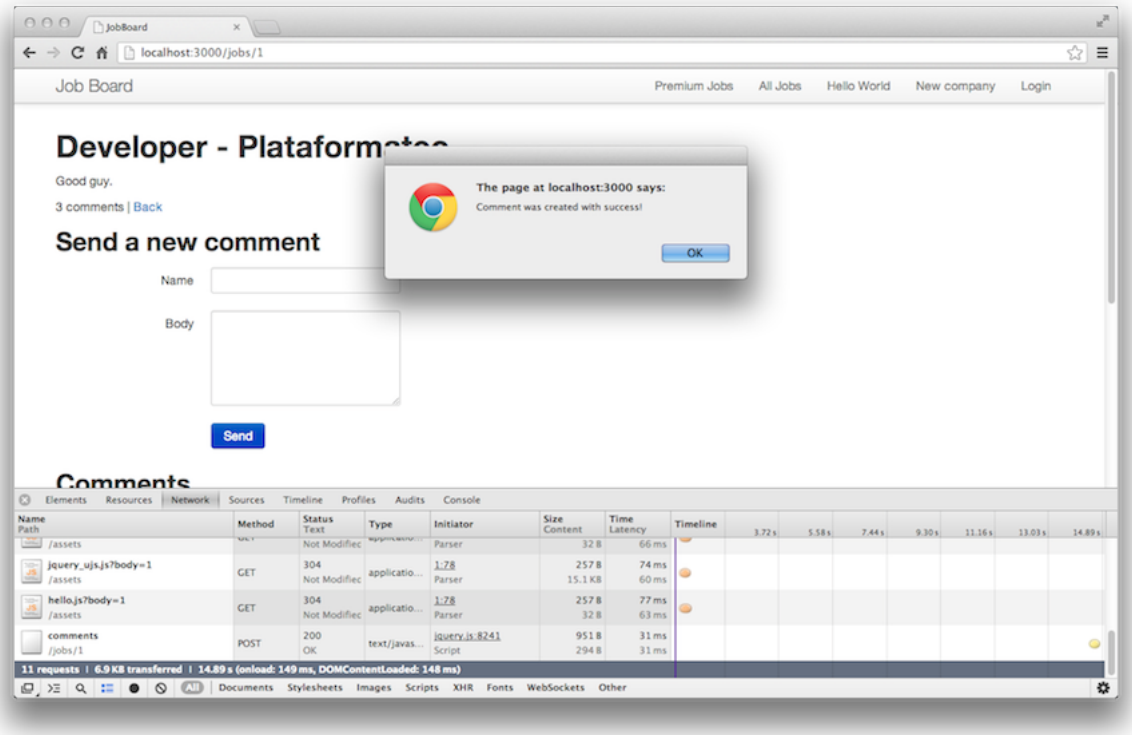
O código é simples: primeiro checamos se o comentário possui algum erro, e caso não, renderizamos o comentário com `render(@comment)`, que vai renderizar a *partial app/views/comments/_comment.html.erb*, e adicionamos esse conteúdo HTML à lista de comentários que já existirá na página. Precisamos utilizar o *helper* `j`, que é um simples atalho para `escape_javascript`, e que tem como objetivo escapar o conteúdo HTML, como aspas por exemplo, para que possa ser inserido corretamente no meio do comando JavaScript. Depois resetamos o conteúdo do formulário para limpar os campos, e mostramos uma mensagem para o usuário informando que o comentário foi criado com sucesso. Caso o comentário não tenha sido salvo, mostramos uma mensagem pedindo que todos os campos sejam preenchidos.

Agora só precisamos adicionar o `div` com o id `#comments` à página, pois ele ainda não existe, para que o nosso comentário seja inserido no lugar correto pelo JavaScript. Abra *app/views/jobs/show.html.erb* e altere a seção de comentários como abaixo:

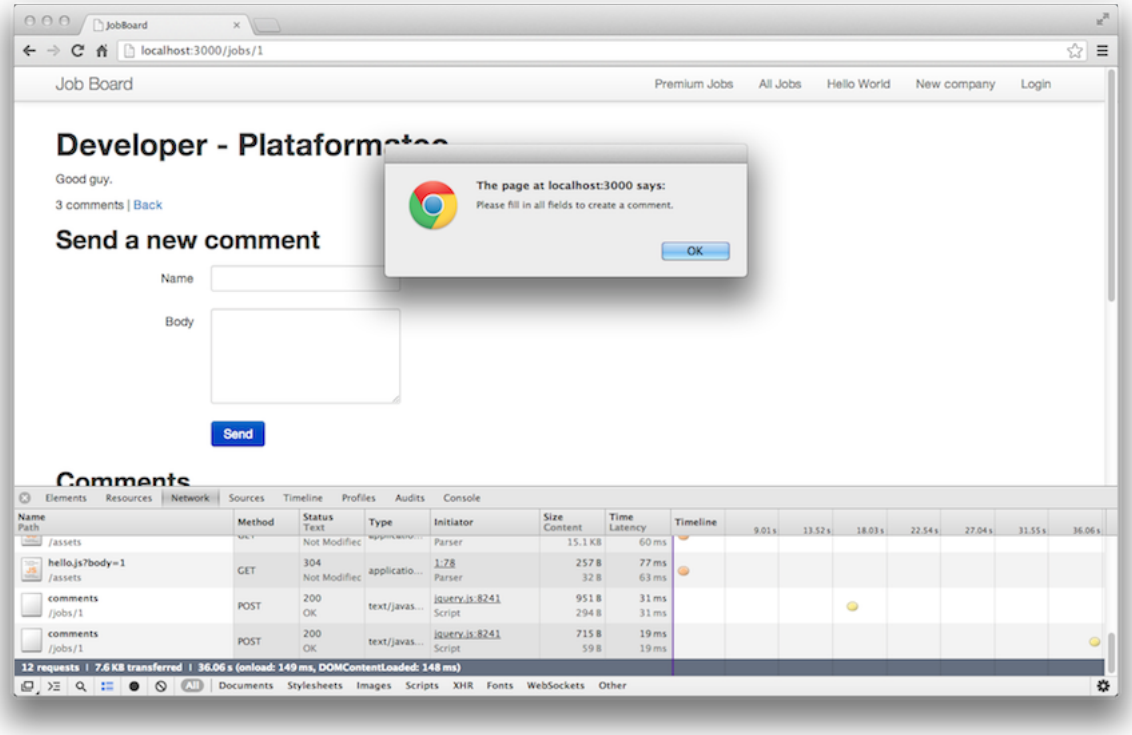
```
<h2>Comments</h2>
<div id="comments">
  <%= render @job.comments %>
</div>
```

Pronto! Vamos testar se tudo está funcionando no navegador. Atualize a página do *job* e mantenha a aba *Network* aberta no painel de desenvolvedor. Agora tente enviar um novo comentário preenchendo todos os campos: você deve receber

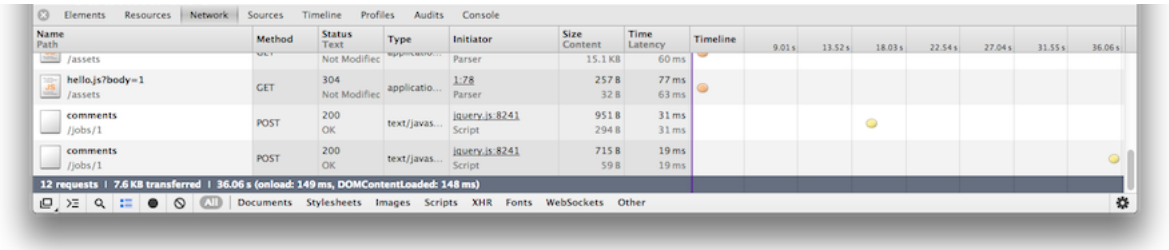
uma mensagem de sucesso, e o comentário deve aparecer na listagem logo abaixo, indicando que tudo ocorreu bem.



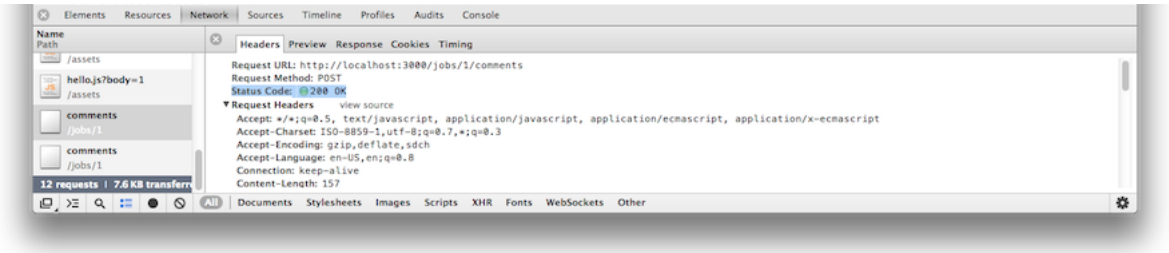
E se você tentar enviar um comentário sem preencher algum campo, receberá uma mensagem de erro. Excelente!



Verifique novamente o painel na aba *Network*, você perceberá que apenas uma requisição foi feita para cada tentativa de enviar um comentário, ou seja, eliminamos o redirecionamento adicional que acontecia com sucesso!

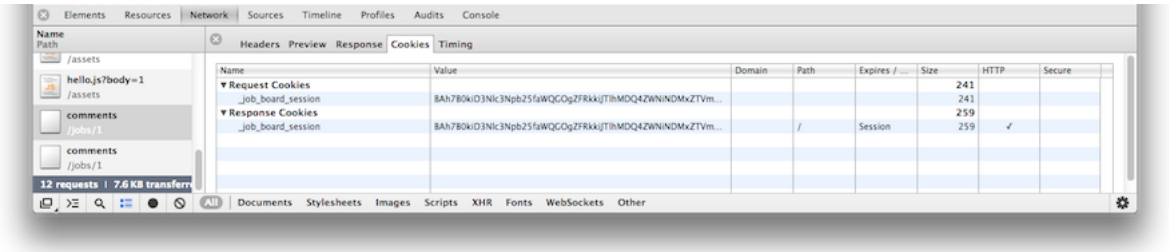


Ainda no painel, se você clicar sobre a primeira requisição para */jobs/1/comments*, poderá observar uma série informações importantes, como por exemplo na aba *Headers* podemos ver o método que foi utilizado: *POST* e o *status* da resposta: *200*, que indica sucesso.



E também na aba *Response* podemos ver o conteúdo que o servidor retornou, nesse caso o código JavaScript que escrevemos e que foi executado no navegador ao receber essa resposta.

Além disso, a aba *Cookies* exibe os *cookies* que são enviados pelo navegador e recebidos de volta do servidor, neste caso apenas o `_job_board_session`, como vimos no capítulo 2.



Podemos incrementar bastante o feedback para o usuário de acordo com a nossa imaginação, como por exemplo removendo as mensagens `alert` e adicionando mensagens `flash`, mas isso ficará como um exercício para você.

E isso encerra o nosso capítulo. Aprendemos como enviar formulários via AJAX, e como o Rails faz uso de *Unobtrusive JavaScript* integrando-se à biblioteca *jQuery* para nos dar algumas funcionalidades sem muito esforço. Também descobrimos como manipular a resposta com base no tipo de requisição que estamos recebendo utilizando `respond_to`, e a retornar código JavaScript em requisições AJAX, que será executado no navegador, permitindo que manipulemos a página melhorando o feedback para o usuário. No capítulo seguinte notificaremos empresas por email quando comentários forem criados, não perca!

Para saber mais

-
- Em nossa aplicação estamos exibindo a listagem de comentários de um *job* na ordem em que eles estão sendo criados, contudo é bastante comum que comentários sejam exibidos na ordem inversa, ou seja, os comentários mais recentes aparecem no topo da lista. Você está encarregado de fazer essa alteração, basta adicionar uma cláusula aos comentários sendo exibidos, ordenando-os através do campo `id` de forma *decrescente*, como abaixo:

```
@job.comments.order('id desc')
```

Isso deve ser feito na *view jobs/show.html.erb*.

-
- Agora que os comentários são corretamente exibidos em ordem inversa, temos que alterar o JavaScript que é executado ao criar um novo comentário, pois ele está sempre inserindo novos comentários **ao final** da lista, lembra-se? Modifique-o para inserir os comentários **no começo** da lista, alterando a *view comments/create.js.erb* para chamar `prepend` ao invés de `append` na listagem de comentários. Pronto!
-
- Com os comentários ordenados, descobrimos que é importante procurarmos não deixar consultas ao Active Record como essa dentro de *views*, pois isso pode facilmente deixar nossas *views* bagunçadas e mais difíceis de serem compreendidas. A maneira correta é movermos essas consultas para os *controllers*, que são os responsáveis por esse tipo de trabalho, e utilizar variáveis de instância para enviar as informações necessárias para as *views*. Então faça isso: mova o código que busca os comentários da *view jobs/show.html.erb* para a variável `@comments` no *controller jobs*, *action show*, dessa forma:

```
def show
  @job = Job.find(params[:id])
  @comments = @job.comments.order('id desc')

  respond_to do |format|
    format.html # show.html.erb
    format.json { render json: @job }
  end
end
```

E utilize agora essa variável na *view*. Pronto! O código está refatorado e agora a *view* está mais limpa e simples de se entender.

-
- Na *partial jobs/_job.html.erb*, você poderá ver que o link *Destroy* possui uma opção `data: { confirm: 'Are you sure?' }`. Essa opção funciona de forma similar ao `remote: true`, o JavaScript do Rails identifica links com o atributo `data-confirm` e os manipula para mostrar a mensagem de confirmação.
-
- O *helper* `link_to` também possui a opção `remote: true`, que funciona de forma similar ao `form_for`, alterando o clique do link para enviar uma requisição AJAX. [Cheque a documentação do método (http://api.rubyonrails.org/classes/ActionView/Helpers/UrlHelper.html#method-i-link_to)](http://api.rubyonrails.org/classes/ActionView/Helpers/UrlHelper.html#method-i-link_to) para mais informações.
-

- Veja a documentação do método `respond_to` (http://api.rubyonrails.org/classes/ActionController/MimeResponds.html#method-i-respond_to) para mais exemplos de uso.