02

Limitando acesso aos jobs através de filtros

Limitando acesso aos *jobs* através de filtros

No capítulo anterior adicionamos o login de empresas, enquanto aprendemos mais sobre layouts e sessões no Rails. Neste capítulo vamos estender a parte de autenticação com autorização, permitindo que apenas empresas cadastradas possam criar novos *jobs* e editar seus *jobs* enviados.

Nota: É importante entendermos a diferença entre *autenticação* e *autorização* de usuários: *autenticação* refere-se a identificar quem está acessando a aplicação, enquanto *autorização* é sobre quais itens um usuário tem acesso dentro da aplicação, o que geralmente acontece depois de autenticado.

Autorização com filtros de controller

Até então nossa aplicação permite criar novos *jobs* a qualquer momento, estando ou não logado com uma empresa. O nosso primeiro passo será garantir que a página de criação de novos *jobs* seja apenas acessível quando há uma empresa logada, pois só assim saberemos a que empresa o *job* pertence quando ele for criado.

Para fazermos isso vamos modificar o *jobs controller*, responsável por gerenciar o CRUD de *jobs*, para adicionar um filtro que vai fazer o trabalho de checar se existe uma empresa autenticada no sistema, e caso não exista, redirecionar para a página inicial informando que a empresa precisa autenticar-se para criar novos *jobs*. Abra o arquivo *app/controllers/jobs_controller.rb* e adicione o seguinte código:

```
class JobsController < ApplicationController
  before_filter :authorize_company, only: [:new, :create]

  # ...

  private

  def authorize_company
    unless current_company
      redirect_to root_path, alert: "You need to login to continue."
    end
  end
end
```

O filtro que utilizamos é um *before_filter*, que nada mais é do que a indicação de um método que desejamos executar **antes** da(s) *action(s)* que definirmos. Ou seja, estamos dizendo que o método *authorize_company* deverá ser chamado **apenas** antes das *actions new* e *create*, e não deverá ser executado para as outras *actions* desse *controller*. Filtros são ótimos para esse tipo de controle de autorização e até mesmo para eliminar duplicação de código entre algumas *actions* de um mesmo *controller*.

Nota: O argumento *only* pode ser omitido, nesse caso o filtro é executado para todas as *actions* desse *controller*, ou é possível utilizar a opção *except* no lugar de *only*, para identificar em que *actions* o filtro **não** deve ser executado (porém normalmente damos preferência por utilizar *only* ao invés de *except*, pois é mais difícil esquecermos de adicionar alguma *action* na lista).

Isto vai garantir que ninguém acesse a página de criação de *jobs* sem estar logado. Inicie o servidor do Rails e tente acessar *http://localhost:3000/jobs/new (http://localhost:3000/jobs/new)*: você deverá ser redirecionado de volta para a página inicial. Contudo, existe um pequeno problema: nenhuma mensagem *flash* foi exibida. Acontece que até então sempre exibimos as mensagens *flash* nas *views* conforme necessário, como no form de login, e nunca precisamos de mensagens na página inicial, então ela não está preparada para exibí-las. Como é bastante comum utilizarmos mensagens em várias páginas diferentes, vamos mover o código que exibe mensagens das *views* para o *layout*.

Abra o arquivo *app/views/jobs/show.html.erb*, e recorte o código de mensagens *flash* existente lá:

```
<% if notice %><p class="alert alert-success"><%= notice %></p><% end %>
<% if alert %><p class="alert alert-error"><%= alert %></p><% end %>
```

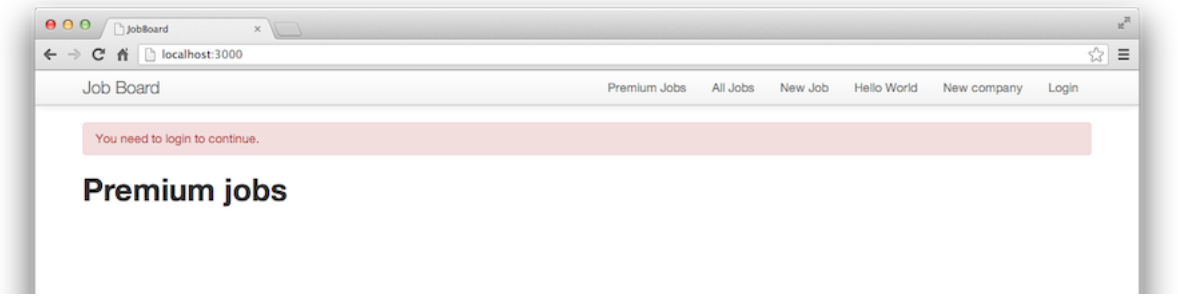
Cole esse código dentro do nosso *layout app/views/layouts/application.html.erb*, logo antes da chamada *yield*:

```
<div class="container">
  <% if notice %><p class="alert alert-success"><%= notice %></p><% end %>
  <% if alert %><p class="alert alert-error"><%= alert %></p><% end %>
  <%= yield %>
</div>
```

Precisamos também remover a mensagem *alert* que adicionamos anteriormente ao form de login quando um erro acontece, caso contrário essa mensagem será exibida duas vezes: uma através do *layout*, e outra pela *view*. Abra *app/views/login/_form.html.erb* e apague o trecho de código relacionado:

```
<% if alert %>
  <p class="alert alert-error">
    <%= alert %>
  </p>
<% end %>
```

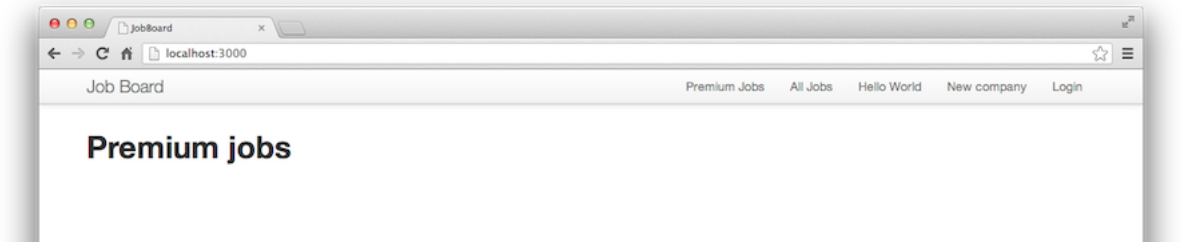
Pronto! Agora volte ao navegador e tente acessar novamente *http://localhost:3000/jobs/new (http://localhost:3000/jobs/new)*: você deverá ser redirecionado para a página inicial como antes, mas agora a mensagem de erro será exibida com sucesso!



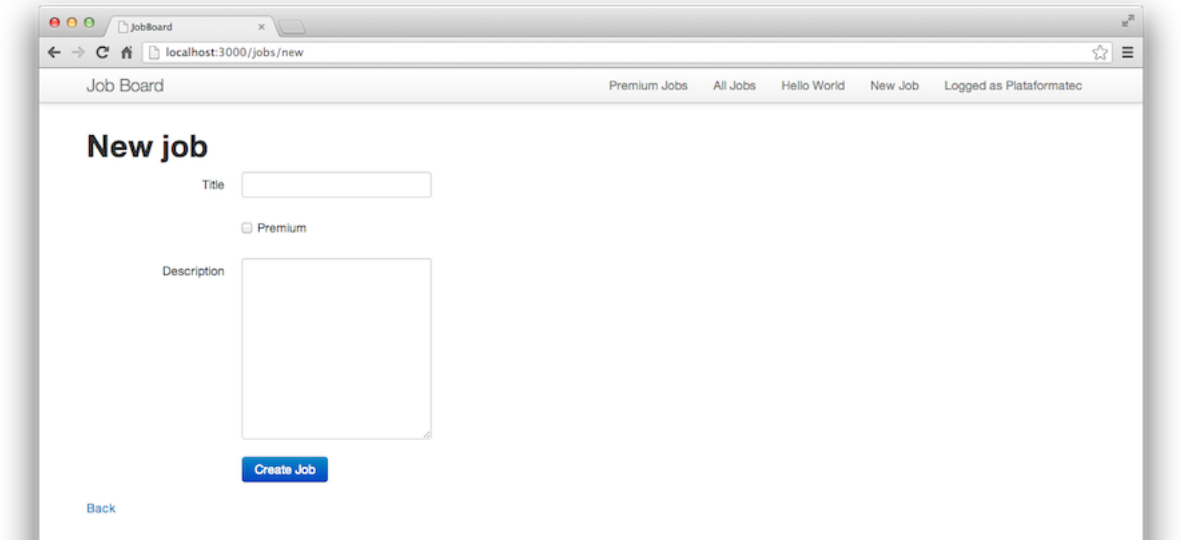
Porém falta um pequeno detalhe que deixamos passar: o link `New Job` está completamente visível para todos os usuários, quer estejam logados ou não. Como somente uma empresa logada poderá cadastrar um novo *job*, não faz sentido este link ser exibido lá quando não há ninguém logado, certo? Vamos então esconder este link, abra novamente o *layout app/views/layouts/application.html.erb*, e altere a seção de links conforme abaixo:

```
<ul id="app-menu" class="nav pull-right">
  <li><%= link_to "Premium Jobs", root_path %></li>
  <li><%= link_to "All Jobs", jobs_path %></li>
  <li><%= link_to "Hello World", hello_world_path %></li>

  <% if current_company %>
    <li><%= link_to "New Job", new_job_path %></li>
    <li><a>Logged as <%= current_company.name %></a></li>
  <% else %>
    <li><%= link_to "New company", new_company_path %></li>
    <li><%= link_to "Login", companies_login_path %></li>
  <% end %>
</ul>
```

Perceba que apenas movemos o link `New Job` para dentro da condição de existir uma empresa logada. Atualize a página no navegador e você verá que o link não está mais lá, perfeito!



Agora basta verificarmos se essa página é realmente acessível quando estamos logados com uma empresa: navegue para a página de acesso através do link *Login* no topo, e autentique-se com o e-mail e senha cadastrados no capítulo anterior. Note que o link `New Job` agora está disponível, e clicando nele conseguimos acessar o formulário de cadastro de *job* com sucesso, excelente!



Nota: se você não lembrar o e-mail e senha que cadastrou anteriormente, basta criar um novo cadastro com um e-mail qualquer para teste.

Pronto, agora que sabemos que o acesso à página para criar *jobs* requer uma empresa autenticada, precisamos associar os *jobs* criados com a empresa atual.

Escopos e associações para controle de acesso

Quando um *job* é criado, devemos armazenar o código da empresa juntamente com ele para podermos posteriormente identificar os *jobs* de cada empresa. Para fazermos isso, vamos gerar uma *migration* adicionando o campo *company_id* à tabela *jobs*, executando no terminal:

```
$ rails g migration add_company_id_to_jobs company_id:integer
```

Antes de atualizar o banco de dados, precisamos fazer uma pequena alteração nesta *migration* para adicionar um índice nesse campo *company_id*.

Índices em migrações

Índices são importantes quando criamos relacionamentos entre tabelas, como no caso de *companies* e *jobs*, pois eles farão com que nossas consultas através desses relacionamentos sejam otimizadas. Neste caso, nós vamos criar um índice no campo *company_id* que estamos adicionando à tabela *jobs*, dessa forma toda vez que precisarmos consultar todos os *jobs* relacionados a uma empresa, esse índice do banco de dados será utilizado e portanto a consulta terá melhor performance. Abra a *migration* que acabou de gerar, e altere conforme abaixo:

```
class AddCompanyIdToJobs < ActiveRecord::Migration
  def change
    add_column :jobs, :company_id, :integer
    add_index  :jobs, :company_id
  end
end
```

Além do comando `add_column` que foi gerado pela *migration* para adicionar o campo *company_id* à tabela *jobs*, criamos também o comando `add_index` para adicionar um índice para este mesmo campo.

Podemos agora executar a *migration* atualizando nosso banco de dados:

```
$ rake db:migrate
```

Então podemos configurar as associações nos modelos. Abra *app/models/company.rb* e adicione:

```
has_many :jobs
```

E abra também *app/models/job.rb*, adicionando:

```
belongs_to :company
```

E aproveite para adicionar também a validação de presença do atributo *:company_id* ao *job*, pois não queremos mais vagas de emprego sem uma empresa relacionada:

```
validates_presence_of :description, :title, :company_id
```

Com as associações configuradas, podemos testar se tudo está funcionando corretamente através do console do Rails:

```
>> company = Company.last
=> #<Company id: 2, name: "Plataformatec", #...>
>> company.jobs
=> []
>> job = company.jobs.create title: "Developer", description: "Good guy."
=> #<Job id: 1, title: "Developer", description: "Good guy.", created_at: "2013-02-18 01:24:11"
>> job.company
=> #<Company id: 2, name: "Plataformatec", #...>
```

Maravilha! Vamos agora utilizar estas associações ao criar *jobs* pela aplicação, associando corretamente a empresa que está criando o *job*. Abra o *app/controllers/jobs_controller.rb* e altere a primeira linha das *actions new* e *create* conforme abaixo:

```
def new
  @job = current_company.jobs.build

  # more...
end

def create
  @job = current_company.jobs.build(params[:job])

  # more...
end
```

A única mudança necessária é que ao invés de utilizarmos `Job.new` para inicializarmos um novo *job*, fazemos isto através da associação utilizando o `current_company`, que está disponível para nós no `ApplicationController`. Com isso, o *job* já estará automaticamente ligado à empresa logada.

Antes de partirmos para o navegador para testarmos tudo isto, vamos alterar o *helper* que exibe o título completo do *job* para mostrar também o nome da empresa a que ele pertence, tornando a navegação mais amigável para o usuário. Abra *app/helpers/jobs_helper.rb*, e altere o *helper* `job_title` conforme abaixo:

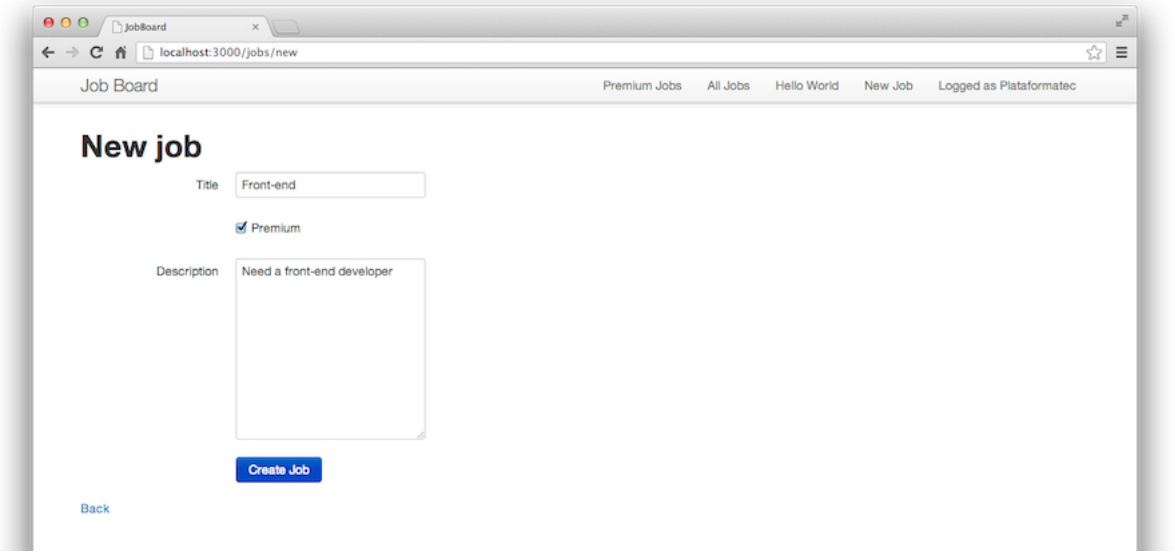
```
def job_title(job)
  title = raw("#{h(job.title)} - #{h(job.company.name)}")

  if job.premium
    title + content_tag(:strong, " (premium)")
  else
    title
  end
end
```

Nota: precisamos utilizar o *helper* `raw` para informar ao Rails que aquele conteúdo é considerado seguro, para que seja então exibido corretamente, pois estamos concatenando strings. Caso isto não seja feito, o Rails escapará o HTML deste *helper* e as tags serão exibidas como texto escapado e não interpretadas como HTML. Perceba que o título do *job* e o nome da empresa são manualmente escapados utilizando o *helper* `h`, que é um atalho para `html_escape`, garantindo que nenhum conteúdo malicioso seja gerado em nossa página, como uma tag javascript por exemplo.

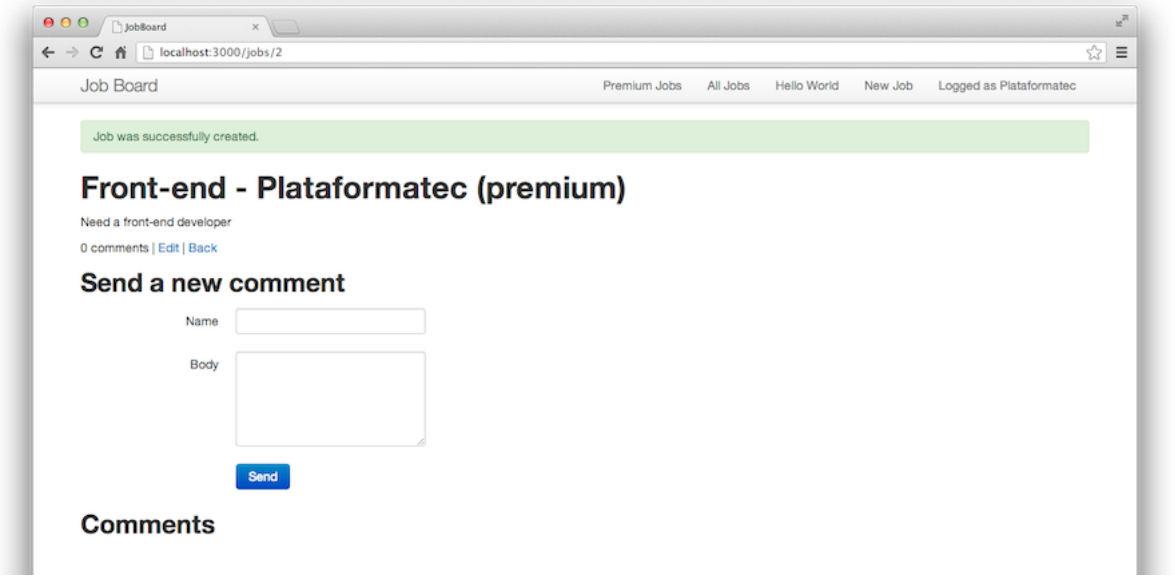
Este *helper* é atualmente utilizado na *view app/views/jobs/show.html.erb*, e vamos alterar agora a *partial app/views/jobs/_job.html.erb* (compartilhada entre ambas as listagens de *jobs* e *jobs premium*) para também utilizá-lo, modificando o cabeçalho desta forma:

```
<h3><%= job_title job %></h3>
```

A alteração nesta *partial* havia ficado como exercício para o aluno durante a primeira parte do curso, por isto ela estava utilizando o código anterior ao invés do *helper*. Agora sim, vamos verificar no navegador. Tenha a certeza de estar logado com alguma empresa, e acesse a página para criar um novo *job*, informando todos os dados e confirmando.



Após criar este novo *job*, já podemos ver o nome da empresa atual sendo exibido corretamente junto com o nome do *job*.



Tudo funcionando como esperado, ótimo! Contudo nossa aplicação ainda possui algumas brechas de segurança: qualquer pessoa, estando logada ou não, pode editar ou remover um *job*. Vamos fazer um teste: através do console, crie uma nova empresa e algumas vagas de emprego:

```
>> company = Company.create name: "Caelum", email: "contato@caelum.com.br", password: "123456",
=> #<Company id: 3, name: "Caelum", email: "contato@caelum.com.br", encrypted_password: "$2a$10
```

```
>> company.jobs.create title: "Support", description: "Help people"
=> #<Job id: 3, title: "Support", description: "Help people", created_at: "2013-02-18 02:06:54"
>> company.jobs.create title: "Teacher", description: "Teach people"
=> #<Job id: 4, title: "Teacher", description: "Teach people", created_at: "2013-02-18 02:07:04
```

Agora faça o teste pelo navegador: estando logado, tente acessar a tela para editar qualquer um destes *jobs* que acabamos de criar, como por exemplo *http://localhost:3000/jobs/4/edit (http://localhost:3000/jobs/4/edit)*, e você verá que isto é possível, da mesma forma que é possível remover um *job* qualquer. Devemos garantir que apenas a própria empresa que criou o *job* possa executar estas ações, então mãos a obra!

Controlando o acesso aos jobs da empresa

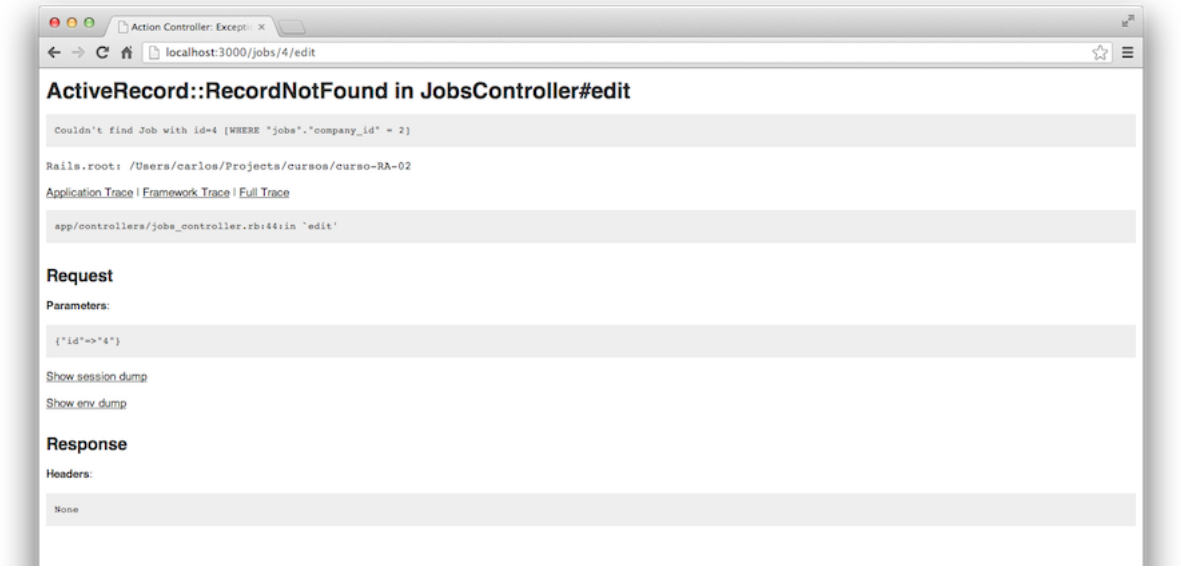
Podemos aplicar a mesma lógica que utilizamos antes para filtrarmos o acesso às demais *actions* necessárias no *jobs controller*: **edit**, **update** e **destroy**. Primeiro alteramos o *before_filter* para adicionar estas *actions*, garantindo assim que exista ao menos uma empresa logada no sistema:

```
before_filter :authorize_company, only: [:new, :create, :edit, :update, :destroy]
```

Depois alteramos estas *actions* para buscar o *job* pelo `id` através da associação com `current_company`, garantindo que somente a empresa que criou o *job* poderá executar tais ações:

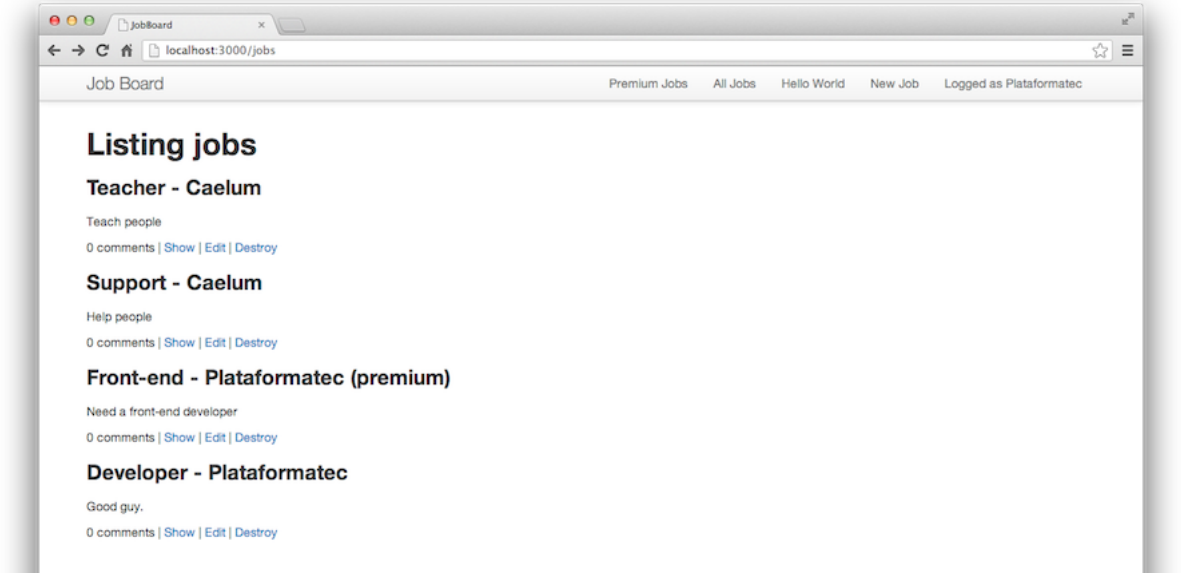
```
def edit
  @job = current_company.jobs.find(params[:id])
end

def update
  @job = current_company.jobs.find(params[:id])

  # more...
end

def destroy
  @job = current_company.jobs.find(params[:id])

  # more...
end
```

Pronto, volte para o navegador, acesse novamente a mesma página de edição, e você deverá ver uma tela como essa:



Indicando que não é mais possível acessarmos a página para editar um *job* que não pertence à empresa atual, mesmo que ele exista no banco de dados. Sucesso!

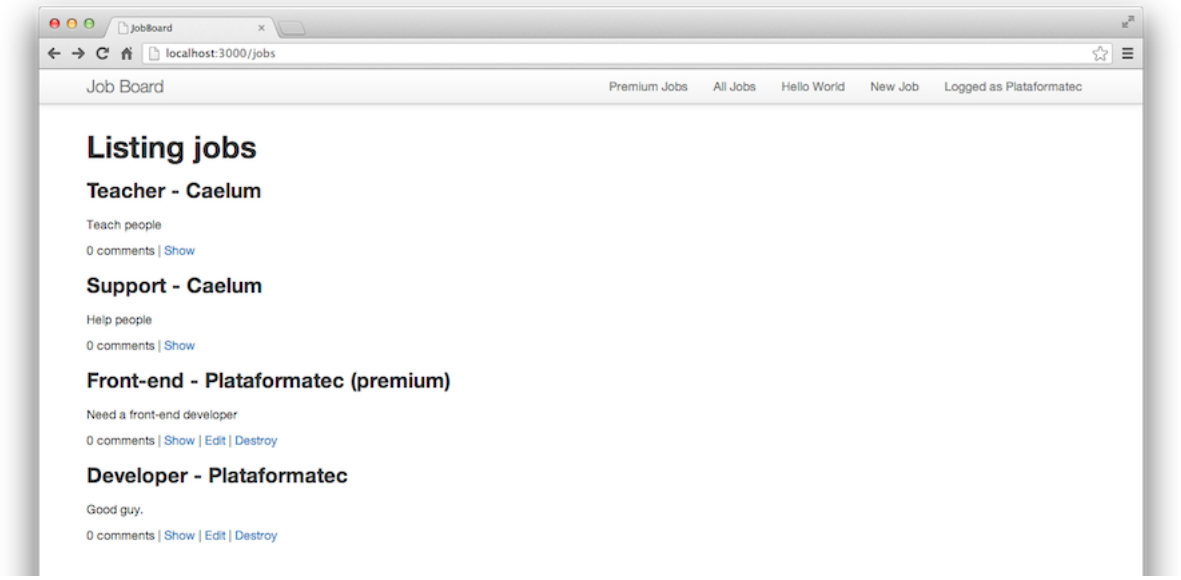
Agora precisamos apenas exibir corretamente os links `Edit` e `Destroy` que aparecem na listagem de *jobs*, somente se o *job* pertencer a empresa atualmente logada, pois não faz sentido exibirmos estes links para *jobs* que são de outra empresa já que não conseguiremos acessá-los. Verifique a listagem de *jobs* em *http://localhost:3000/jobs (http://localhost:3000/jobs)* para ver todos eles contendo os links:



Abra novamente a *partial app/views/jobs/_job.html.erb* e adicione a checagem pela empresa atual nos links, como abaixo:

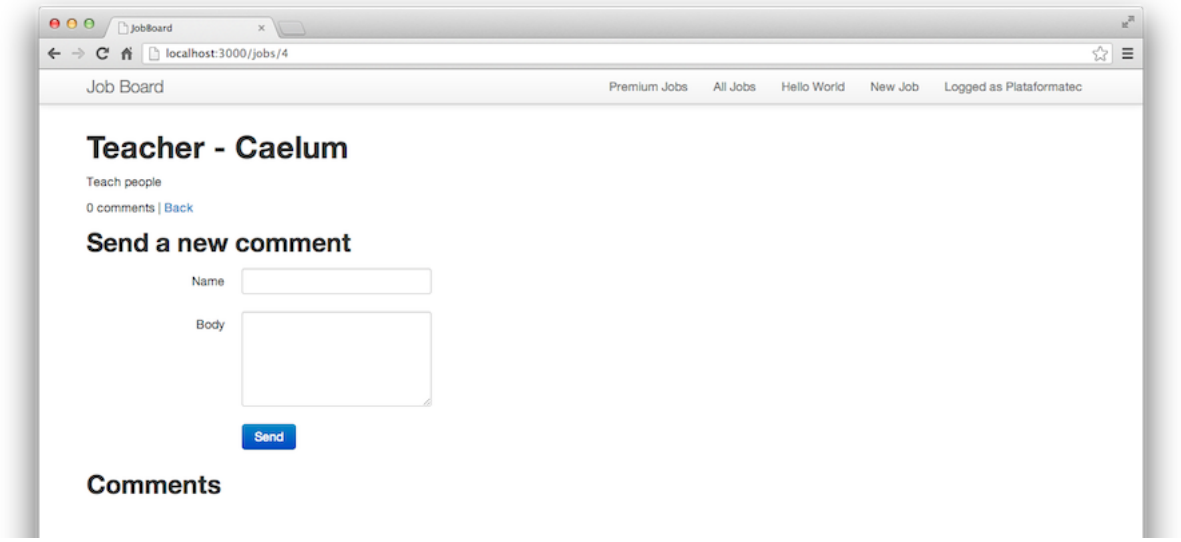
```
<p>
  <%= pluralize(job.comments.size, "comment") %> |
  <%= link_to 'Show', job %>
  <% if current_company == job.company %>
    | <%= link_to 'Edit', edit_job_path(job) %>
    | <%= link_to 'Destroy', job, method: :delete, data: { confirm: 'Are you sure?' } %>
  <% end %>
</p>
```

Volte para o navegador e recarregue a página: os links `Edit` e `Destroy` devem sumir nos *jobs* que são de outras empresas, e permanecerem nos *jobs* que são da empresa que você está acessando.



Ótimo! Porém, a visualização do *job* ainda possui o link `Edit`, então vamos corrigir lá também, alterando a seção de links na *view app/views/jobs/show.html.erb*:

```
<p>
  <%= pluralize(@job.comments.size, "comment") %> |
  <% if current_company == @job.company %>
    <%= link_to 'Edit', edit_job_path(@job) %> |
  <% end %>
  <%= link_to 'Back', jobs_path %>
</p>
```



Perfeito! Temos uma estrutura de autorização em mãos, garantindo que apenas empresas logadas podem cadastrar novos *jobs*, e que somente poderão alterar ou excluir os *jobs* criados por elas mesmas.

Contudo agora temos uma pequena questão de performance em mãos: ao exibirmos a lista de *jobs*, para cada *job* estamos consultando a empresa a que ele pertence, e isso gera uma série de consultas no banco de dados, que podemos

otimizar facilmente através das facilidades do Active Record. Esse é um problema de performance bem comum, conhecido como **N+1**, e vamos entender um pouco mais sobre como identificar e otimizar isso no próximo capítulo!

Neste capítulo aprendemos a controlar o acesso para que apenas empresas cadastradas possam acessar determinadas páginas da nossa aplicação. Também garantimos que *jobs* somente possam ser editados ou removidos pela empresa que os cadastrou, criando um sistema de autorização completo em torno dos *jobs*. Até o próximo capítulo, onde vamos discutir um pouco sobre performance e otimizações, não perca!

Para saber mais

-
- Além do `before_filter`, o Rails possui outros tipos de filtros que podem ser configurados nos *controllers*, como `after_filter` e `around_filter`. Consulte mais informações sobre os filtros de controllers no Rails Guide sobre o Action Controller (http://guides.rubyonrails.org/action_controller_overview.html#filters)
-
- As associações `has_many` possuem uma série de métodos, como o `build` e o `find` que utilizamos aqui, que criam e encontram itens com base no escopo da associação. Conheça mais sobre a associação `has_many` no Rails Guide sobre Associações (http://guides.rubyonrails.org/association_basics.html), e veja também a documentação do `has_many` (http://api.rubyonrails.org/classes/ActiveRecord/Associations/ClassMethods.html#method-i-has_many) para descobrir todos os métodos disponibilizados através de associações.
-
- Neste capítulo criamos um índice para o campo *company_id* em *jobs*. Você pode também criar uma nova *migration* adicionando um índice para *job_id* em *comments*.