02

Autenticação de empresas com BCrypt

Autenticação de empresas com BCrypt

No capítulo anterior nós criamos uma página para cadastro de empresas. Além do nome e e-mail, cada empresa deve cadastrar uma senha, que é criptografada antes de ser armazenada no banco de dados. Neste capítulo criaremos uma página de login, onde empresas informarão seu e-mail e senha cadastrados e ganharão acesso ao sistema caso os dados estejam corretos.

Validação da senha

Para validar uma senha qualquer, nós vamos adicionar um novo método ao modelo `Company` definido em *app/models/company.rb* que retorna `true` se o *password* recebido é igual ao *password* armazenado de forma encriptada no banco de dados:

```
def valid_password?(password_to_validate)
  BCrypt::Password.new(encrypted_password) == password_to_validate
end
```

O método recebe um *password* para validar (`password_to_validate`) e compara esse *password* com o password encriptado no banco dados (`encrypted_password`) com a ajuda do `BCrypt` . Vamos utilizar o console do Rails para verificar que o nosso método funciona corretamente:

Nota: se você já tiver um console do Rails rodando, apenas chame `reload!` para recarregar o console.

```
>> company = Company.new(password: "123456")
=> #<Company id: nil, ...>
>> company.valid_password?("invalid")
=> false
>> company.valid_password?("123456")
=> true
```

Excelente! Agora que sabemos como validar um *password*, vamos adicionar a tela de login.

Rotas, controllers, views e formulários

O primeiro passo é adicionar as rotas para a nossa tela de login. Assim como no cadastro de empresas, vamos adicionar as rotas necessárias ao arquivo *config/routes.rb*:

```
get  "/companies/login", to: "login#new"
post "/companies/login", to: "login#create"
```

As duas rotas possuem o mesmo endereço, porém utilizam verbo HTTP diferentes. Ao acessarmos a página */companies/login*, o *controller* e *action login#new* serão acessados, e vão renderizar um formulário para preencher e-mail e senha. Ao preencher e enviar o formulário, será enviada uma requisição **POST** para a mesma URL, mas que agora

será recebida por *login#create* e deverá executar o trabalho de logar a empresa. Vamos criar o nosso novo controller em *app/controllers/login_controller.rb*:

```
class LoginController < ApplicationController
  def new
  end

  def create
    company = Company.find_by_email(params[:company][:email])

    if company && company.valid_password?(params[:company][:password])
      session[:company_id] = company.id
      redirect_to root_path
    else
      flash.now[:alert] = "Invalid e-mail or password."
      render action: "new"
    end
  end
end
```

Dessa vez, a *action create* é um pouco diferente das que criamos anteriormente. Primeiro, observe que estamos buscando uma empresa por e-mail. Para tal, utilizamos o método `Company.find_by_email` e passamos o e-mail que recebemos do formulário como argumento. O método `find_by_email` é chamado de **método dinâmico**, porque o Rails define esse método automaticamente baseado nos campos de cada modelo. Por exemplo, o modelo `Job` possui um campo chamado `title`, então o Rails disponibiliza um método chamado `find_by_title` nesse modelo.

O método `find_by_email` vai retornar uma instância do modelo `Company` ou `nil` caso nenhuma empresa exista com aquele e-mail no banco de dados. Em seguida, nós verificamos se a senha enviada através do formulário é igual a senha encriptada no banco de dados, com a ajuda do método `valid_password?` que criamos anteriormente.

Caso o e-mail e senha sejam válidos, nós armazenamos o `id` da empresa na sessão e redirecionamos para a página inicial (vamos entender um pouco mais sobre sessões logo mais). Caso contrário, mostramos uma mensagem de erro e exibimos novamente o formulário. Observe como usamos o método `flash.now` ao invés de `flash`, pois dessa forma a mensagem flash será exibida **agora**, durante a renderização do formulário devido ao erro, e não na próxima requisição.

Agora, vamos criar as *views* para exibir o formulário de login. Como no capítulo anterior, vamos primeiro criar uma *view* em *app/views/login/new.html.erb* que vai apenas renderizar o formulário:

```
<h1>Company login</h1>

<%= render 'form' %>

<%= link_to 'Back', root_path %>
```

O formulário será criado em *app/views/login/_form.html.erb*:

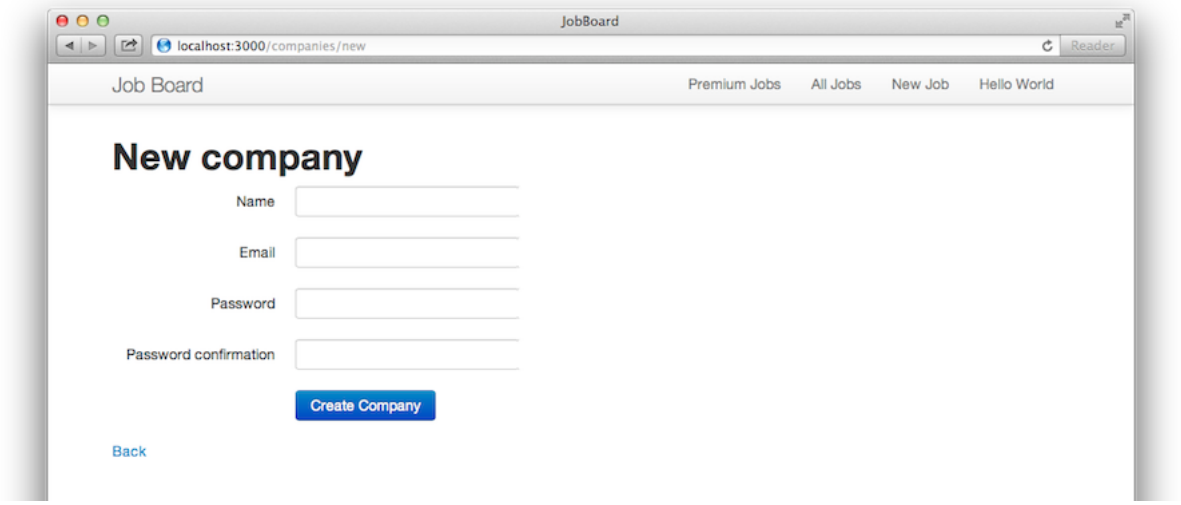
```
<%= form_for(Company.new, url: companies_login_path, html: { class: 'form-horizontal' }) do |f|
  <% if alert %>
    <p class="alert alert-error">
      <%= alert %>
    </p>
```

```
  <% end %>

  <div class="control-group">
    <%= f.label :email, class: 'control-label' %>
    <div class="controls">
      <%= f.text_field :email %>
    </div>
  </div>
  <div class="control-group">
    <%= f.label :password, class: 'control-label' %>
    <div class="controls">
      <%= f.password_field :password %>
    </div>
  </div>
  <div class="control-group controls">
    <%= f.submit "Login", class: 'btn btn-primary' %>
  </div>
<% end %>
```

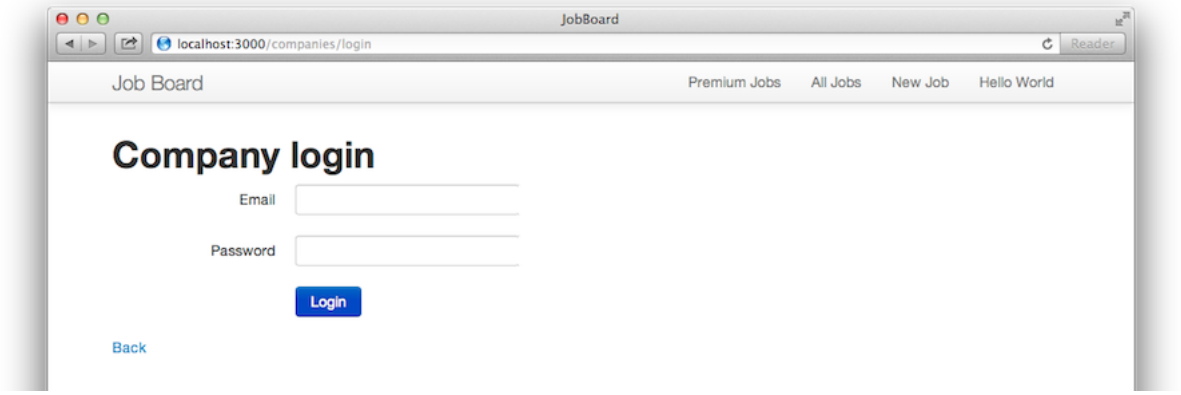
O formulário é bastante similar ao do capítulo anterior com duas principais diferenças:

1.
2. Na primeira linha, nós passamos a URL para qual queremos enviar o formulário explicitamente através da opção `url: companies_login_path` ;
3.
4. Ao invés de mostrarmos as mensagens de erro que existem no modelo `Company` , dessa vez nós vamos simplesmente mostrar o conteúdo da mensagem de `alert` que setamos no controller;

Com as *views* criadas, estamos prontos para verificar se a nossa feature de login funciona. Primeiramente, vamos visitar a página *http://localhost:3000/companies/new (http://localhost:3000/companies/new)* e criar uma nova empresa com e-mail e senha:



Após criar a empresa, vamos para a página de login em *http://localhost:3000/companies/login (http://localhost:3000/companies/login)* e informar o mesmo e-mail e senha:



Se o e-mail e senha estiverem corretos, seremos redirecionados para a página inicial, indicando que o login funciona como esperado! Porém, existem algumas melhorias que podem ser feitas, por exemplo, seria interessante exibirmos no canto direito superior se estamos logados, juntamente com o nome da empresa, certo? Vamos fazer isso na próxima seção!

Layouts no Rails

Agora que empresas conseguem logar no nosso sistema, seria útil se pudéssemos mostrar no canto superior direito da página o nome da empresa atualmente logada. Porém, nós queremos mostrar essa informação em *todas* as páginas e não apenas em uma página ou outra. E nós sabemos que seria bastante trabalhoso duplicar essa informação em todas as páginas manualmente, além de ir contra uma das convenções do Rails, que diz que não devemos nos repetir (**DRY**). Por esse motivo, uma aplicação Rails possui **layouts**.

Os **layouts** são onde escrevemos a estrutura geral de todas as nossas páginas HTML, como por exemplo cabeçalhos e rodapés que normalmente não mudam entre as páginas. As *views* da nossa aplicação, como as criadas na seção anterior, são inseridas em um local que determinamos dentro desse *layout*. Dessa forma conseguimos manter nossas views muito mais *DRY*, eliminando assim a necessidade de copiar a estrutura HTML geral da aplicação em todas as páginas.

O *layout* principal da aplicação fica em *app/views/layouts/application.html.erb*, onde podemos observar o código que gera os links que estão no topo da nossa página e, mais ao final, também uma chamada **yield**, que é exatamente onde será inserido o conteúdo da *view* que estamos renderizando.

Em outras palavras, qualquer alteração que fizermos em *app/views/layouts/application.html.erb* será refletida em todas as páginas, o que torna esse arquivo o lugar ideal para colocarmos a informação da empresa que está logado. Para isso, vamos editar o *layout* e adicionar um novo item ao menu. O menu no HTML é dado pela tag `ul` com atributo `id` igual a `app-menu`:

```
<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 "New Job", new_job_path %></li>
  <li><%= link_to "Hello World", hello_world_path %></li>

  <% if current_company %>
    <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>
```

Com o *layout* alterado, nós agora checamos se existe uma empresa definida através do método `current_company`. Caso exista, mostramos que estamos logado como tal empresa, caso contrário, exibimos links para cadastrar uma empresa ou fazer login.

Compartilhamento de helpers entre o controller e a view

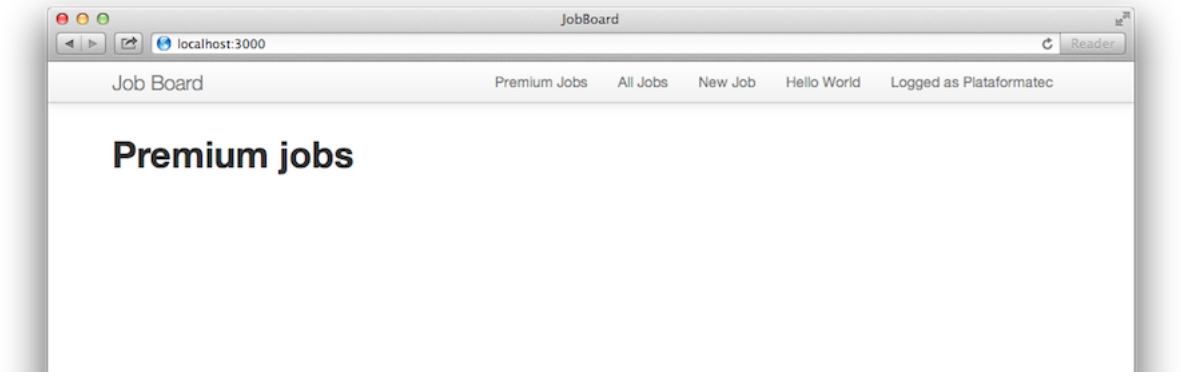
Estamos quase prontos, o último passo é criar esse método `current_company` que utilizamos no *layout*. Como esse método será usado em todas as páginas e precisaremos dele também nos *controllers* posteriormente para verificar se existe uma empresa logada, nós vamos definí-lo no `ApplicationController`, localizado em *app/controllers/application_controller.rb*:

```
class ApplicationController < ActionController::Base
  protect_from_forgery

  private

  def current_company
    @current_company ||= if session[:company_id]
      Company.find_by_id(session[:company_id])
    end
  end
  helper_method :current_company
end
```

Esse método simplesmente verifica se existe um `:company_id` armazenado na sessão, e usa ele para encontrar a empresa logada atualmente, armazenando-a na variável `@current_company`. Observe que utilizamos `Company.find_by_id` ao invés de `Company.find`, já que o `find` gera uma exceção caso o `id` da empresa não exista e, neste caso, não queremos uma exceção, apenas que `nil` seja retornado. Note também que estamos utilizando `||=` para armazenar a empresa na variável `@current_company`, com isso evitamos que `Company.find_by_id` seja executado toda vez que o método `current_company` for chamado, pois o valor será cacheado na variável após a primeira execução.

Como todos os *controllers* herdam de `ApplicationController`, eles terão acesso ao `current_company`. Porém, as *views* ainda não sabem que esse método existe, e precisamos dele no *layout*, então chamamos o método `helper_method` para dizer ao Rails que o método `:current_company` deve estar disponível também nas *views*, sendo visto assim como qualquer outro *helper*.

Carregue novamente a aplicação e você verá no canto direito o nome da empresa que está atualmente logada no sistema. Excelente!



Entendendo sessões e cookies

Depois que fazemos login com uma empresa, todas as páginas que acessarmos mostrarão no canto superior direito o nome dessa empresa. Isso acontece porque, ao fazermos login no *controller* e *action login#create*, nós executamos:

```
session[:company_id] = company.id
```

O comando acima armazena o identificador da empresa logada na sessão. Por padrão, a sessão do Rails é armazenada em um **cookie** que é enviado de volta para o navegador. A cada nova requisição, o navegador envia uma cópia desse *cookie* para o Rails, o Rails converte esse *cookie* novamente para uma sessão e assim conseguimos acessar o `session[:company_id]` em cada página, para calcular o `current_company` que usamos no nosso *layout*.

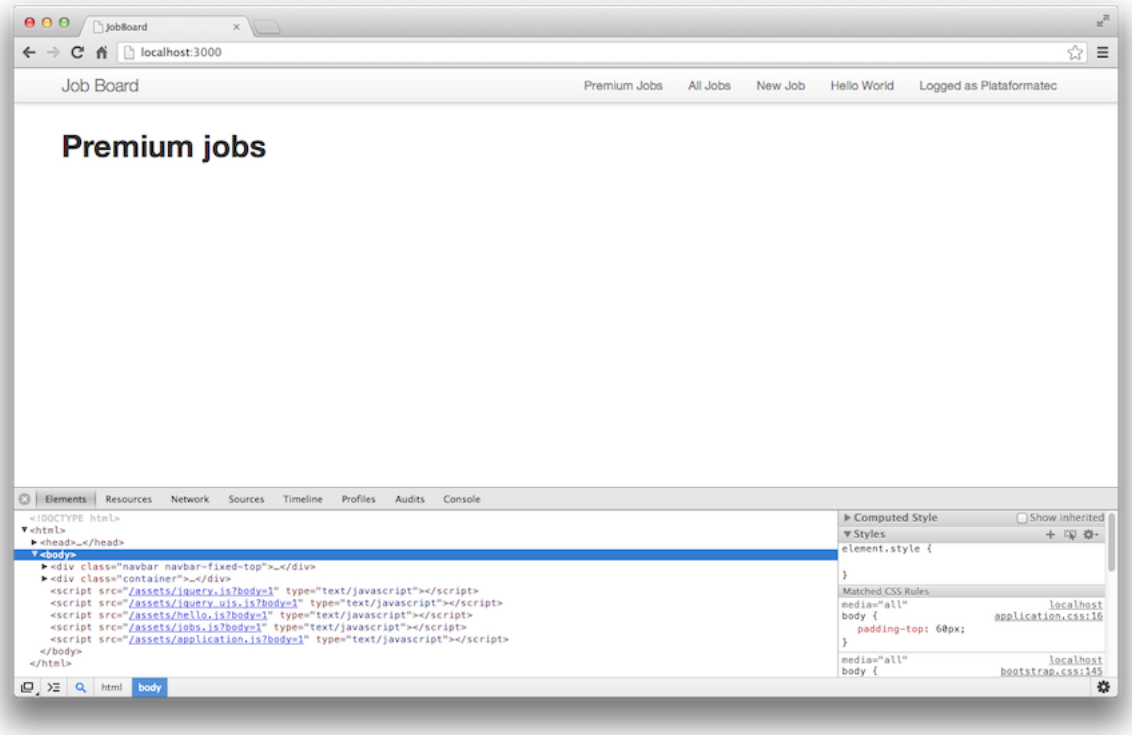
O Rails é responsável por gerenciar as sessões para que não precisemos nos preocupar com esses detalhes. De qualquer forma, o Rails nos fornece algumas opções para configurar como as sessões funcionam. Para tal, verifique o arquivo em *config/initializers/session_store.rb*:

```
# Be sure to restart your server when you modify this file.

JobBoard::Application.config.session_store :cookie_store, key: '_job_board_session'

# Use the database for sessions instead of the cookie-based default,
# which shouldn't be used to store highly confidential information
# (create the session table with "rails generate session_migration")
# JobBoard::Application.config.session_store :active_record_store
```

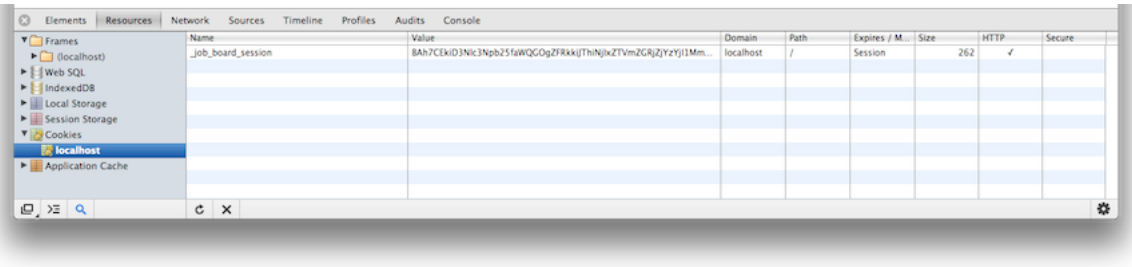
O arquivo simplesmente configura a sessão para ser armazenada em um *cookie* e a chave (*key*) do cookie é `_job_board_session` . Como o *cookie* é sempre enviado de volta ao navegador, nós **nunca** devemos colocar informação importante nele, como número de cartão de crédito, a senha do usuário, etc.

Para verificarmos que a sessão é realmente armazenada em um *cookie*, vamos utilizar o navegador Google Chrome (http://www.google.com/chrome), que possui excelentes ferramentas para desenvolvedores, para nos mostrar todos os *cookies* definidos em uma página e utilizá-lo para remover esses *cookies*.

Abra o Chrome, caso ele não esteja aberto, e faça *login* utilizando o e-mail e senha da empresa criada na seção anterior. Agora, clique **Control + Shift + C** (ou **Cmd + Option + C** se você utiliza Mac) e o Chrome abrirá um painel:



Clique na aba Recursos (`Resources`) e veja que no lado esquerdo temos um item **Cookies** e dentro dele **localhost**, clique em **localhost** e ele mostrará todos os cookies definidos pela aplicação:



Nessa lista, você verá o cookie da nossa sessão, com nome `_job_board_session` e o seu valor, que é gerenciado pelo Rails. Ao clicarmos com o botão direito e deletarmos esse cookie, a informação da sessão será perdida, e portanto seremos deslogados do site. Recarregue a página e verifique que não estamos mais logados, .

Com isso, finalizamos o segundo capítulo! Nós adicionamos login à nossa aplicação ao mesmo tempo em que aprendemos como *layouts* e sessões funcionam no Rails.

Para saber mais

-
- É bastante comum que aplicações web também ofereçam uma ação para fazer *logout*. Essa ação poderia ser adicionada ao `LoginController` e simplesmente limparia a sessão, apagando todos os valores existentes previamente, redirecionando então para a página inicial:

```
class LoginController < ApplicationController
  def destroy
    session.clear
    redirect_to root_path
  end
end
```

Você está encarregado de terminar essa funcionalidade. Tudo que precisa ser feito é adicionar uma nova rota e alterar o *layout* para mostrar o *link* de *logout*.