03

Estendendo o job board com cadastro de empresas

Seja bem-vindo ao curso de Rails 3 do alura

Recomendamos esse curso para aquelas pessoas que já conheçam o rails3 ou trabalha em algum projeto legado do mesmo. Para quem é novo e deseja começar a aprender do rails agora recomendamos o curso de rails 4: https://www.alura.com.br/course/ruby-on-rails-4-do-zero (https://www.alura.com.br/course/ruby-on-rails-4-do-zero)

Estendendo o job board com cadastro de empresas

Este curso é a continuação do curso **RA-01**, sobre desenvolvimento de aplicações web com Ruby on Rails.

No curso anterior, nós começamos a construir um "Classificado de Empregos", onde empresas podem cadastrar novas vagas de emprego, e candidatos podem interagir com as vagas. Ao construirmos essa aplicação, nós aprendemos sobre a estrutura do Rails, discutimos rotas, *views*, *models* e *controllers*, além de associações, mensagens *flash*, paginação e outros.

Neste curso continuaremos o desenvolvimento desta aplicação e trabalharemos diversos aspectos que são comuns em aplicações para web, como por exemplo: cadastro e autenticação de empresas para enviarem novos jobs, envio de comentários utilizando AJAX, envio de emails para notificar empresas de novos comentários, internacionalização para utilizar a aplicação em português e inglês, além de algumas informações importantes sobre segurança e performance, entre outros.

Para os alunos que não concluíram o curso **RA-01**, uma cópia da aplicação desenvolvida no curso anterior e que vamos utilizar como base neste curso está disponível aqui (https://github.com/plataformatec/curso-RA-02). Você pode baixar uma cópia da aplicação utilizando Git, ou como um arquivo zip (https://github.com/plataformatec/curso-RA-02/zipball/master).

Para quem participou da primeira parte, esta cópia da aplicação teve seu layout melhorado para a continuação deste curso, contudo as funcionalides que foram desenvolvidas no primeiro curso não foram alteradas, e o código em geral permanece como estava.

Para o layout, nós utilizamos o Twitter Bootstrap (http://twitter.github.com/bootstrap/) que é um conjunto de arquivos CSS e imagens que adicionam um estilo elegante à nossa aplicação. Nós modificamos o HTML das páginas de listagem, visualização de jobs e dos formulários para utilizar o *Bootstrap*, deixando o layout mais bonito e integrado. Você pode conferir essas alterações no repositório Git que disponibilizamos. Apesar de não ser totalmente obrigatório, recomendamos que os alunos que desenvolveram a aplicação no primeiro curso também baixem essa nova aplicação para continuar.

Após obter a aplicação, vamos garantir que ela está atualizada e funciona corretamente. Acesse o diretório da aplicação através da linha de comando e digite:

```
$ gem install bundler
$ bundle install --without production
```

Lembre-se que o **Bundler** é a ferramenta responsável por gerenciar as nossas **gems** e dependências. Primeiro instalamos a gem do bundler e depois com o comando `bundle install`, pedimos ao **Bundler** para instalar qualquer dependência que não esteja ainda disponível na nossa máquina, exceto as dependências de produção.

Após instalar essas dependências, devemos criar o banco de dados e executar as migrações existentes, garantindo que o banco de dados está atualizado:

```
$ rake db:create db:migrate
```

Perceba que executamos as duas tarefas `rake` em sequência, ao invés de rodarmos cada uma separadamente. E finalmente podemos iniciar o servidor web:

```
$ rails s
```

Observe que iniciamos o servidor utilizando apenas `rails s` invés de `rails server`. Acontece que `rails s` é simplesmente um atalho. Digite apenas `rails` e você verá todos os comandos disponíveis e seus atalhos.

Com a aplicação rodando, verificamos que podemos criar ofertas de emprego, que mapeiam para **jobs** no banco de dados, com sucesso. Porém, como discutimos a pouco, queremos limitar essa ação apenas para empresas cadastradas, que poderão se autenticar no sistema. É isto que vamos desenvolver durante os próximos capítulos.

Modelos e BCrypt

O primeiro passo é permitir que empresas se cadastrem. Para isso, nós vamos criar o modelo `Company` com os campos *name*, *email* e *encrypted_password*, todos de tipo *string*:

```
$ rails g model company name:string email:string encrypted_password:string
```

Nota: Observe que utilizamos o atalho `rails g` invés de `rails generate`.

Após gerar o modelo, vamos rodar as migrações para atualizar o banco de dados:

```
$ rake db:migrate
```

Nós armazenaremos o nome e o e-mail da empresa nos campos *name* e *email*. No campo *encrypted_password*, salvaremos a senha de acesso da empresa cadastrada, porém faremos isso de forma encriptada. É muito comum para aplicações web armazenarem informação sensível do usuário, como códigos de acesso, senhas, etc, e por questões de segurança, devemos armazenar isso de forma encriptada, para que não seja possível que qualquer pessoa com acesso ao banco de dados consiga ler essas informações.

Com o modelo gerado e migrações executadas, vamos alterar o nosso modelo para encriptar o *password* toda vez que ele for setado. Abra o modelo `Company` em *app/models/company.rb* e faça as seguintes alterações:

```
require "bcrypt"

class Company < ActiveRecord::Base
  attr_accessible :email, :name, :password

  def password=(new_password)
    @password = new_password
    self.encrypted_password = BCrypt::Password.create(@password)
```

```
  end

  def password
    @password
  end
end
```

Nós definimos um método chamado `password=` que recebe o novo valor do campo `password` como argumento. Este valor é armazenado na variável de instância `@password`. Em seguida, nós setamos o campo `encrypted_password` com o resultado de `BCrypt::Password.create`. O **BCrypt** é uma biblioteca de encriptação segura (http://pt.wikipedia.org/wiki/Bcrypt), que foi carregada no começo do arquivo com `require 'bcrypt'`, e que vamos utilizar durante o curso para encriptar informações.

Após a definição do método `password=`, criamos um método `password` que simplesmente retorna o valor da variável de instância `@password` setada anteriormente.

Finalmente, observe que mudamos a linha do `attr_accessible`, substituindo o valor `:encrypted_password` por `:password`. O motivo de tal mudança é porque precisamos que o *encrypted_password* seja interno ao modelo *Company*, como se fosse um campo *privado*, e portanto não deve ser acessível. Todas as mudanças neste campo devem acontecer através do atributo *password*, que por sua vez poderá ser setado pelo usuário ao criar ou atualizar os dados de uma empresa.

Para garantir que os nossos métodos funcionam como esperado, vamos começar um console do Rails. Porém, como nós estamos utilizando *BCrypt*, nós precisamos primeiro adicioná-lo como dependência da aplicação. Para isso, abra o arquivo *Gemfile* e adicione:

```
gem 'bcrypt-ruby', '~> 3.0'
```

Com isso, nós adicionamos a gem *Bcrypt* como dependência na aplicação. O segundo parâmetro diz explicitamente que queremos o *BCrypt* na versão 3.0 em diante. Vamos instalar essa nova dependência:

```
$ bundle install
```

Nota: Perceba que não passamos a opção `--without production` para o comando `bundle install`. O **Bundler** salva essa informação a primeira vez que executamos o comando com este argumento, assim não precisamos utilizar ele toda vez que rodamos `bundle install`.

E agora sim podemos iniciar um console do Rails:

```
$ rails c
```

Dentro desse console, podemos instanciar uma nova empresa e verificar que os atributos *password* e *encrypted_password* estão vazios:

```
>> company = Company.new
=> #<Company id: nil, name: nil, email: nil, encrypted_password: nil, created_at: nil, updated_
>> company.password
=> nil
```

```
>> company.encrypted_password
=> nil
```

Porém, se setarmos o atributo *password*, ele modificará ambos valores:

```
>> company.password = "123456"
=> "123456"
>> company.password
=> "123456"
>> company.encrypted_password
=> "$2a$10$3A0Cztna2OGJpgI9elLj9eUYJwjUCquQAGH6rZVwxPWjlIOEVNv52"
```

Observe como o atributo *encrypted_password* não armazena o *password*, e sim sua versão encriptada. Cada máquina terá uma versão encriptada diferente, logo você não receberá o mesmo valor que acima.

Por último, vamos identificar que as modificações que fizemos em `attr_accessible` funcionam como esperado. Ainda no console, digite:

```
>> Company.new(encrypted_password: "123456")
ActiveModel::MassAssignmentSecurity::Error: Can't mass-assign protected attributes: encrypted_pa
```

Veja que ao tentarmos criar uma nova empresa setando o atributo *encrypted_password*, acontece uma exceção, já que este campo não é acessível. Porém, setar um *password* deve ser possível:

```
>> Company.new(password: "123456")
=> #<Company id: nil, name: nil, email: nil, encrypted_password: "$2a$10$LQ1FMw2QxPbHWO841fJQDe(
```

Observe que setar um *password* não apenas funcionou, mas como também já setou o valor do campo *encrypted_password* para o valor encriptado. Isso aconteceu porque, para os modelos do Rails, `Company.new(password: "123456")` é similar a escrevermos:

```
company = Company.new
company.password = "123456"
company
```

Com o modelo criado, precisamos agora gerar a *rota*, o *controller* e a *view* para cadastrar novas empresas.

Rotas, controllers, views e formulários

Agora nós vamos criar um novo *controller* chamado `CompaniesController` que terá duas *actions*: *new* e *create*. Para gerar tais rotas, vamos utilizar o método *resources*, o mesmo utilizado pelo *scaffold*, mas vamos configurar apenas as rotas que precisamos para as actions *new* e *create*. Logo, abra o arquivo *config/routes.rb* e adicione:

```
resources :companies, only: [:new, :create]
```

Podemos ver as novas rotas geradas com `rake routes`:

```
   companies POST   /companies(.:format)            companies#create
new_company GET    /companies/new(.:format)        companies#new
```

Ao acessarmos */companies/new* através do navegador, ele deverá acessar o controller e action *companies#new* e renderizar um formulário com os campos para criação de empresas. Ao preencher e enviar o formulário, nós enviaremos uma requisição **POST** para a URL */companies* que será recebida por *companies#create* e deverá criar uma nova empresa no nosso banco de dados através do modelo `Company`.

Agora, vamos criar um arquivo chamado *app/controllers/companies_controller.rb* com o seguinte conteúdo:

```
class CompaniesController < ApplicationController
  def new
    @company = Company.new
  end

  def create
    @company = Company.new(params[:company])

    if @company.save
      redirect_to root_path, notice: "Company was successfully created."
    else
      render action: "new"
    end
  end
end
```

Não há nada de especial nas duas *actions* do nosso *controller*. A *action new* simplesmente seta a variável de instância `@company` que vai ser utilizada na view, enquanto a *action create* tenta criar uma nova empresa através de `@company.save`, recebendo os parâmetros do formulário. Caso a empresa seja criada com sucesso, nós redirecionamos para a página inicial da aplicação, caso contrário, renderizamos a *action new* novamente com o formulário e as mensagens de erro.

Por padrão, a *action new* vai renderizar a *view app/views/companies/new.html.erb*. Como essa *view* não existe, vamos criá-la com o seguinte conteúdo:

```
<h1>New company</h1>

<%= render 'form' %>

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

Seguindo o padrão do *scaffold* gerado pelo Rails, a nossa *view* de criação de empresas simplesmente renderiza a *partial* de formulários. Logo, vamos criar essa *partial* em *app/views/companies/_form.html.erb* com o seguinte conteúdo:

```
<%= form_for(@company, html: { class: 'form-horizontal' }) do |f| %>
  <% if @company.errors.any? %>
    <div class="alert alert-error">
      <h3><%= pluralize(@company.errors.count, "error") %> prohibited this company from being s
```

```
      <ul>
      <% @company.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
      </ul>
    </div>
  <% end %>

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

Novamente, o conteúdo é bastante parecido com a *partial* de *form* gerada pelo *scaffold*, apenas ajustamos o layout conforme o *Twitter Bootstrap*. A principal diferença foi a adição do campo *password_confirmation* além dos campos *name*, *email* e *password*. É bastante comum solicitar aos usuários que informem suas senhas duas vezes para garantir que não há erros de digitação, e é para isso que adicionamos o *password_confirmation*. Este atributo é automaticamente gerenciado pelas validações do Rails, apenas precisamos adicionar as validações ao modelo `Company`. Volte para o arquivo do modelo e adicione as seguintes validações:

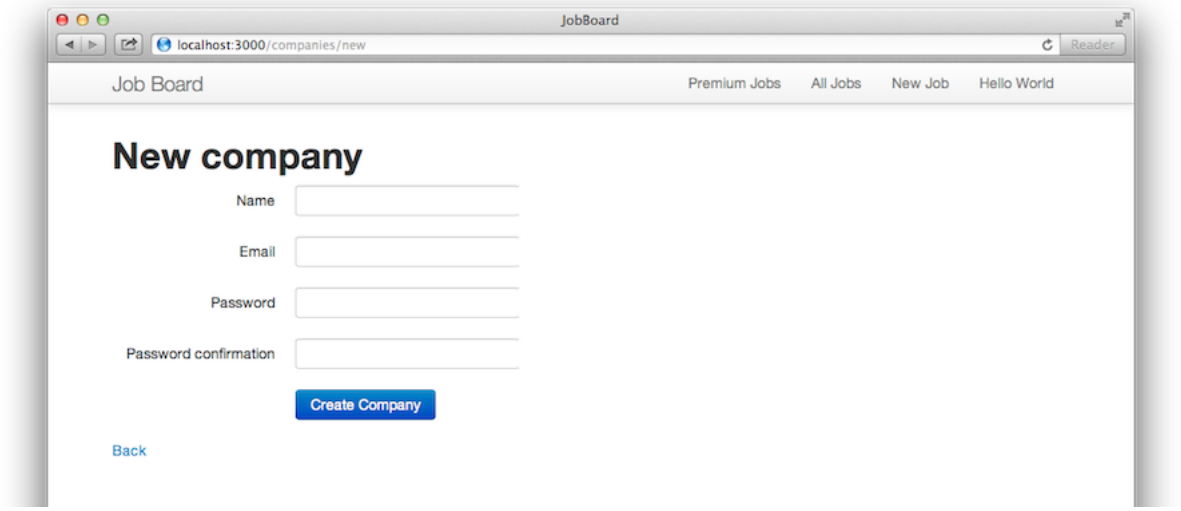
```
validates_presence_of :email, :name, :password
validates_uniqueness_of :email
validates_length_of :password, minimum: 6
validates_confirmation_of :password
```

Além de validarmos a presença dos campos *name*, *email* e *password*, nós também validamos que o *email* é único, ou seja, não permite duplicados, que o *password* tem ao menos 6 dígitos e é igual à sua confirmação.

Também precisamos adicionar o novo atributo `password_confirmation` à lista de atributos acessíveis, altere como abaixo:

```
attr_accessible :email, :name, :password, :password_confirmation
```

Para garantir que todo o código funciona como esperado, vamos iniciar um novo servidor do Rails com `rails s`. Caso você já tenha um servidor rodando, terá que reiniciá-lo de qualquer forma, já que alteramos o *Gemfile* para adicionar o *BCrypt* como dependência. Após iniciar o servidor, acesse *http://localhost:3000/companies/new (http://localhost:3000/companies/new)* para ver o formulário:



Ao preencher o formulário corretamente, somos redirecionados para a página inicial, o que indica que tudo ocorreu bem. Volte ao console do Rails e execute `Company.first` para verificar que a empresa foi criada com sucesso! Fique à vontade para testar as validações no formulário também!

Com isso encerramos o primeiro capítulo! Agora que empresas podem se cadastrar na nossa aplicação, no próximo capítulo adicionaremos o mecanismo necessário para que elas possam efetuar login utilizando o email e senha cadastrados. Não perca!

Para saber mais

-
- O método `validates_presence_of` usado em validações só aceita um atributo se ele estiver presente. Mas o que quer dizer que um atributo está presente? No Rails, um atributo é considerado presente (`present?`) quando ele não for vazio (`blank?`). Para mais informações, verifique a documentação destes métodos na documentação Rails:
    -
    - `blank?` (http://api.rubyonrails.org/classes/Object.html#method-i-blank-3F)
    -
    - `present?` (http://api.rubyonrails.org/classes/Object.html#method-i-present-3F)
-
- Veja mais sobre o `bcrypt-ruby` no github (https://github.com/codahale/bcrypt-ruby).