03

Criando um job board em Ruby on Rails

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)

Criando uma nova aplicação

O Rails é um framework para desenvolver aplicações web. Para conhecermos melhor como utilizá-lo, vamos criar um "Classificado de Empregos", onde empresas podem cadastrar novas vagas de emprego, e candidatos podem interagir com as vagas. Tendo isso em mente, nosso primeiro objetivo será criar esse aplicativo web e acessá-lo através do nosso navegador. Vamos utilizar um comando que o Rails disponibiliza para criar uma nova aplicação, que vamos chamar de **Job Board**. Execute o comando abaixo em seu terminal:

```
$ rails new job_board
```

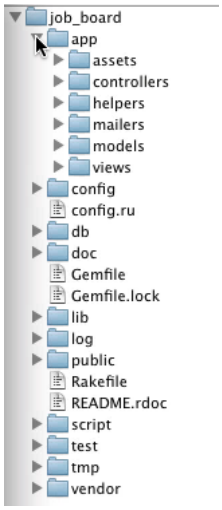
Nota: Todos os comandos identificados com `$` deverão ser executados em seu terminal / linha de comando (digite-os sem o `$`).

O resultado deste comando é bastante verboso, pois o Rails nos explica tudo que está acontecendo: um novo diretório chamado `job_board` é criado, e dentro dele o Rails monta todo o esqueleto de nossa aplicação:

```
$ rails new job_board
      create
      create  README.rdoc
      create  Rakefile
      create  config.ru
      create  .gitignore
      create  Gemfile
      create  app
      create  app/assets/images/rails.png
      create  app/assets/javascripts/application.js
      create  app/assets/stylesheets/application.css
      create  app/controllers/application_controller.rb
      create  app/helpers/application_helper.rb
      create  app/mailers
      create  app/models
      create  app/views/layouts/application.html.erb
      create  app/mailers/.gitkeep
      create  app/models/.gitkeep
      create  config
      create  config/routes.rb
      create  config/application.rb
      create  config/environment.rb
      create  config/environments
      create  config/environments/development.rb
      create  config/environments/production.rb
      create  config/environments/test.rb
```

```
      create  config/initializers
      create  config/initializers/backtrace_silencers.rb
      create  config/initializers/inflections.rb
      create  config/initializers/mime_types.rb
      create  config/initializers/secret_token.rb
      create  config/initializers/session_store.rb
      create  config/initializers/wrap_parameters.rb
      create  config/locales
      create  config/locales/en.yml
      create  config/boot.rb
      create  config/database.yml
      create  db
      create  db/seeds.rb
      create  doc
      create  doc/README_FOR_APP
      create  lib
      create  lib/tasks
      create  lib/tasks/.gitkeep
      create  lib/assets
      create  lib/assets/.gitkeep
      create  log
      create  log/.gitkeep
      create  public
      create  public/404.html
      create  public/422.html
      create  public/500.html
      create  public/favicon.ico
      create  public/index.html
      create  public/robots.txt
      create  script
      create  script/rails
      create  test/fixtures
      create  test/fixtures/.gitkeep
      create  test/functional
      create  test/functional/.gitkeep
      create  test/integration
      create  test/integration/.gitkeep
      create  test/unit
      create  test/unit/.gitkeep
      create  test/performance/browsing_test.rb
      create  test/test_helper.rb
      create  tmp/cache
      create  tmp/cache/assets
      create  vendor/assets/javascripts
      create  vendor/assets/javascripts/.gitkeep
      create  vendor/assets/stylesheets
      create  vendor/assets/stylesheets/.gitkeep
      create  vendor/plugins
      create  vendor/plugins/.gitkeep
         run  bundle install
Fetching gem metadata from https://rubygems.org/.........
...
Your bundle is complete! It was installed into vendor/bundle
```

A estrutura da aplicação é organizada como a imagem abaixo:



São vários arquivos que vamos conhecendo aos poucos durante o curso, o mais importante nesse momento é lembrarmos que o diretório `app` é onde fica o código principal da nossa aplicação.

Isto conclui a primeira parte do nosso objetivo: criar uma aplicação web. Mas, agora precisamos encontrar uma maneira de acessarmos esta aplicação através do navegador, como podemos fazer isto? Felizmente, o Ruby já possui um mini servidor web chamado `WEBrick`, pronto para utilizarmos exatamente para este fim, e o Rails disponibiliza um comando para rodar a aplicação usando este servidor web por padrão. Isto é tudo que precisamos para acessar a aplicação. Para levantar o servidor, navegue para o diretório que acabamos de criar:

```
$ cd job_board
```

E execute o comando:

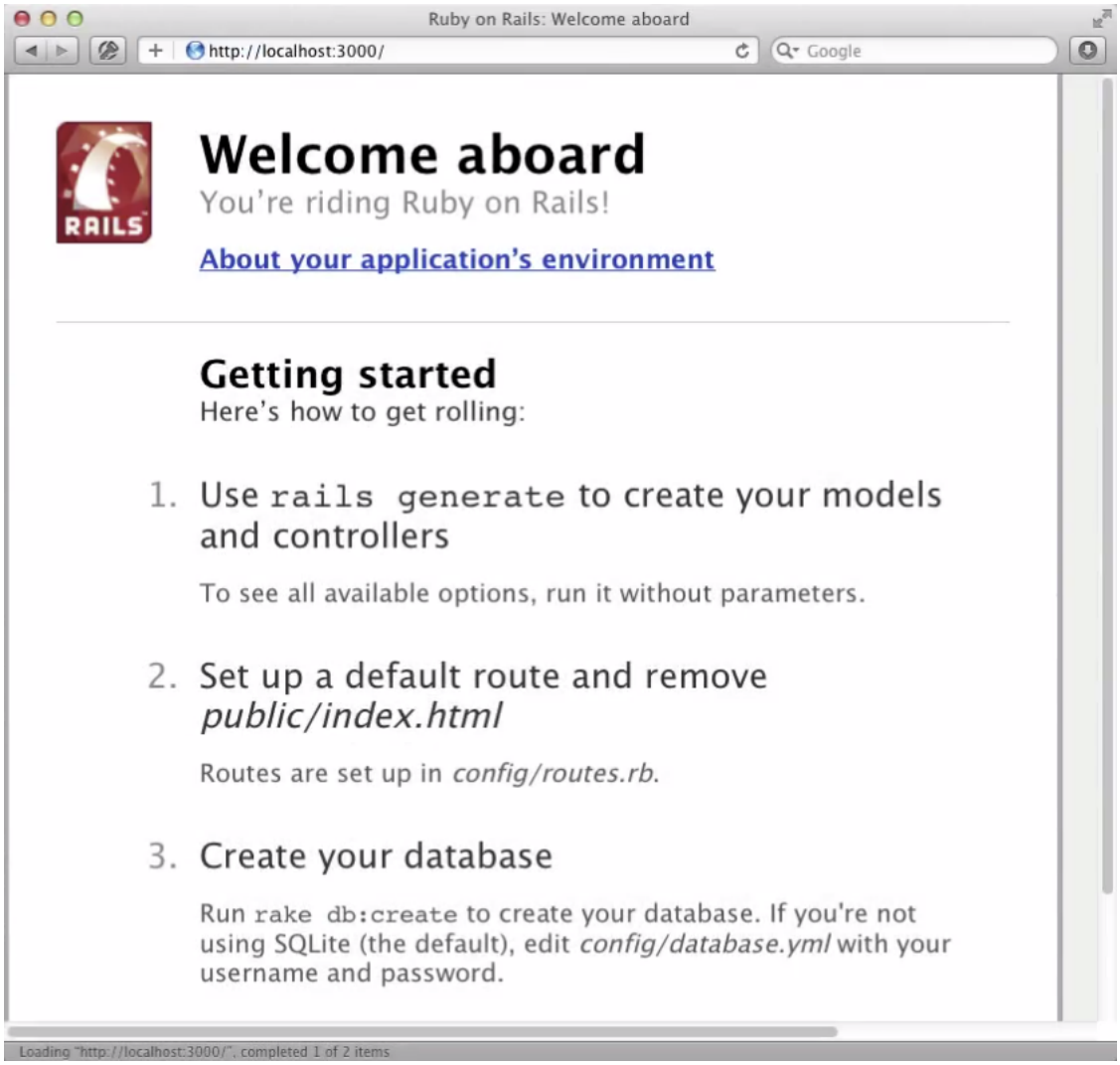
```
$ rails server
```

Ao executar este comando no terminal, você verá o seguinte:

```
$ rails server
=> Booting WEBrick
=> Rails 3.2.6 application starting in development on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
[2012-07-05 10:57:25] INFO  WEBrick 1.3.1
[2012-07-05 10:57:25] INFO  ruby 1.9.3 (2012-04-20) [x86_64-darwin11.4.0]
[2012-07-05 10:57:25] INFO  WEBrick::HTTPServer#start: pid=41829 port=3000
```

Na primeira linha vemos que o Rails está realmente utilizando o servidor `WEBrick`. Logo abaixo o Rails nos informa que ele está usando o ambiente de **desenvolvimento** (*development*), e que a aplicação está acessível em nossa máquina (*localhost* ou *0.0.0.0*), na porta **3000**. Ele também nos informa que se for necessário parar o servidor, basta pressionarmos `Ctrl + C`.

Agora que o servidor está rodando, vamos acessar a aplicação no navegador, através da url (http://localhost:3000)http://localhost:3000 (http://localhost:3000), e devemos visualizar uma página informativa do Rails, indicando que nossa aplicação está rodando sem problemas, como abaixo:



Esta página também nos mostra os próximos passos para continuarmos o desenvolvimento, que veremos adiante.

Criando o primeiro Hello World

Nosso objetivo agora é exibir a mensagem "Hello World" acessando uma página através da url (http://localhost:3000/hello/world)http://localhost:3000/hello/world (http://localhost:3000/hello/world).

Para que isso seja possível, precisamos escrever algum código que entenda que a url */hello/world* exista em nossa aplicação, e devolva o conteúdo da página, "Hello World". Ao invés de escrevermos todo este código manualmente, podemos pedir ao Rails que gere automaticamente o que precisamos para começar - estes geradores de código são chamados de *generators*.

Logo, vamos pedir ao Rails que gere um controlador (*controller*) chamado *hello* com uma ação (*action*) *world*, que vai responder pela url */hello/world* e devolver o conteúdo que definirmos. Abra um novo terminal, navegue até o diretório da aplicação `job_board` , e execute o seguinte:

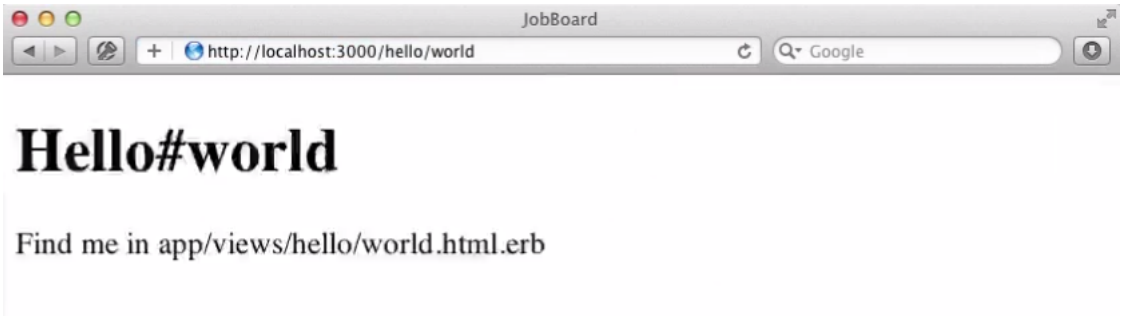
```
$ rails generate controller hello world
```

O Rails nos mostrará todos os arquivos que ele gera automaticamente:

```
$ rails generate controller hello world
      create  app/controllers/hello_controller.rb
       route  get "hello/world"
```

```
invoke  erb
create    app/views/hello
create    app/views/hello/world.html.erb
invoke  test_unit
create    test/functional/hello_controller_test.rb
invoke  helper
create    app/helpers/hello_helper.rb
invoke    test_unit
create      test/unit/helpers/hello_helper_test.rb
invoke  assets
invoke    coffee
create      app/assets/javascripts/hello.js.coffee
invoke    scss
create      app/assets/stylesheets/hello.css.scss
```

Com isso, já conseguimos alcançar uma boa parte do nosso objetivo: podemos agora acessar a url (http://localhost:3000/hello/world)http://localhost:3000/hello/world (http://localhost:3000/hello/world) em nosso navegador, e veremos o seguinte:



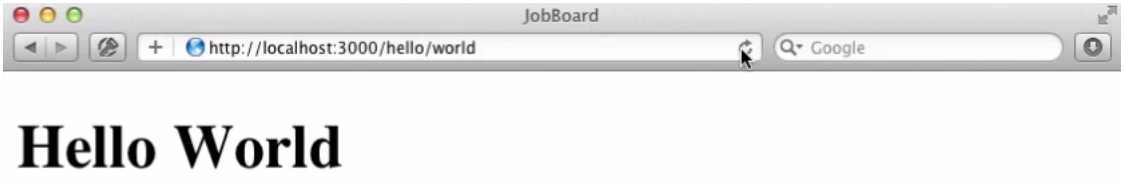
Quase lá! Nosso próximo passo é alterar o que está sendo exibido nesta página, para mostrarmos o "Hello World". Um dos arquivos criados pelo *generator* do Rails é o *app/views/hello/world.html.erb*, que contém o conteúdo da página que acabamos de visualizar no navegador. Abra esse arquivo em seu editor, e você verá o HTML que é exibido naquela página:

```
<h1>Hello#world</h1>
<p>Find me in app/views/hello/world.html.erb</p>
```

Vamos alterar esse conteúdo para o seguinte:

```
<h1>Hello World</h1>
```

Salve e arquivo e volte para o navegador. Atualize a página que estávamos vendo, (http://localhost:3000/hello/world)http://localhost:3000/hello/world (http://localhost:3000/hello/world), e agora veremos nosso novo HTML:



Sucesso, completamos nosso objetivo! Note que em nenhum momento foi necessário reiniciar o servidor para visualizar as mudanças no navegador: o Rails recarrega automaticamente estas mudanças sem precisar parar e iniciar o servidor novamente, basta atualizar a página! Isto ocorre quando estamos utilizando o ambiente *development* que citamos mais acima, para facilitar nosso processo de desenvolvimento. Existe também o ambiente de produção (*production*), onde esse processo de recarregar os arquivos automaticamente não acontece, fazendo com que esse ambiente seja mais rápido e adequado para ser utilizado quando nossa aplicação for ao ar.

Criando a primeira estrutura de empregos

Como o objetivo de nossa aplicação gira em torno de empregos, ou *jobs* como vamos nos referir a partir de agora, nosso próximo passo será desenvolver o código necessário que nos permita criar, exibir, alterar e remover *jobs* (create, read, update, e delete, também conhecido como *CRUD*).

Novamente, ao invés de escrevermos todo o código manualmente, vamos pedir ao Rails que gere uma estrutura de *CRUD* pronta para nós, através de um *generator* chamado *scaffold*. Mas, antes de executarmos os comandos necessários, precisamos saber quais atributos um *job* precisa ter: o mínimo necessário seriam um título com poucos caracteres, e uma descrição mais detalhada. Então vamos começar com isto: gerar uma estrutura de *CRUD* (*scaffold*) para o *job*, com os campos título (*title*) e descrição (*description*). Na linha de comando, execute:

```
$ rails generate scaffold job title:string description:text
```

Isto nos mostra tudo que o *scaffold* gera para nós automaticamente:

```
$ rails generate scaffold job title:string description:text
      invoke  active_record
      create    db/migrate/20120705163028_create_jobs.rb
      create    app/models/job.rb
      invoke    test_unit
      create      test/unit/job_test.rb
      create      test/fixtures/jobs.yml
      invoke  resource_route
       route    resources :jobs
      invoke  scaffold_controller
      create    app/controllers/jobs_controller.rb
      invoke    erb
      create      app/views/jobs
      create      app/views/jobs/index.html.erb
      create      app/views/jobs/edit.html.erb
      create      app/views/jobs/show.html.erb
      create      app/views/jobs/new.html.erb
      create      app/views/jobs/_form.html.erb
      invoke    test_unit
      create      test/functional/jobs_controller_test.rb
      invoke    helper
      create      app/helpers/jobs_helper.rb
      invoke      test_unit
      create        test/unit/helpers/jobs_helper_test.rb
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/jobs.js.coffee
      invoke    scss
      create      app/assets/stylesheets/jobs.css.scss
```

```
invoke  scss
create    app/assets/stylesheets/scaffolds.css.scss
```

Quando falamos em um *CRUD* de *jobs*, precisamos pensar que será necessário armazenar estes *jobs* em algum local, como um banco de dados (*database*, muitas vezes apenas chamado de *db*). Agora nos perguntamos: preciso criar isso manualmente para a minha aplicação? A resposta é: não. O Rails possui uma série de **tarefas** (ou *tasks*) prontas para nos auxiliar nesse processo, e para executar estas tarefas ele traz consigo um comando chamado `rake`. Vamos então utilizar o `rake` para criar nosso *database*, no terminal:

```
$ rake db:create
```

Nota: Por padrão o Rails utiliza o banco de dados SQLite, que já está configurado em nosso ambiente.

Nota: se você ver uma mensagem como *db/development.sqlite3 already exists*, não se preocupe, isto significa que o banco de dados já existe, e foi gerado quando acessamos a aplicação a primeira vez - isto acontece quando utilizamos o SQLite apenas. Veremos mais detalhes sobre banco de dados posteriormente.

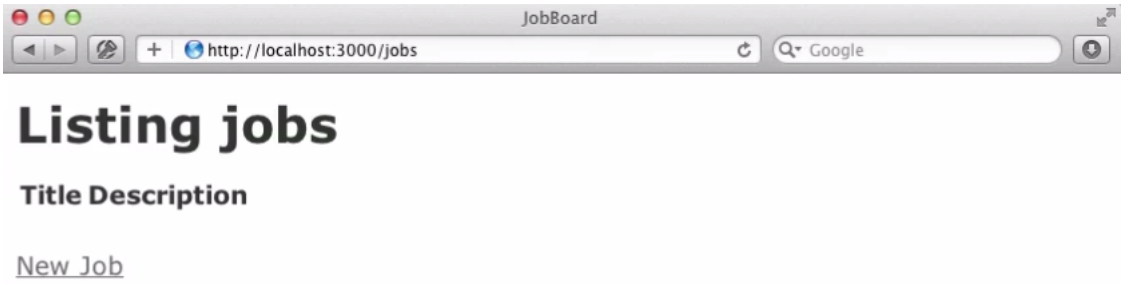
Nenhuma mensagem será exibida no terminal, o que confirma que a criação do nosso banco de dados funcionou. O banco de dados serve como um local de armazenamento, onde cada item deve ser armazenado em uma tabela. No nosso caso, vamos armazenar os nossos *jobs* em uma tabela chamada **jobs** - note que mantemos uma convenção de nomenclatura. Felizmente, o *scaffold* gera instruções para adicionar a tabela *jobs* ao banco de dados, só precisamos executar estas instruções com o comando:

```
$ rake db:migrate
```

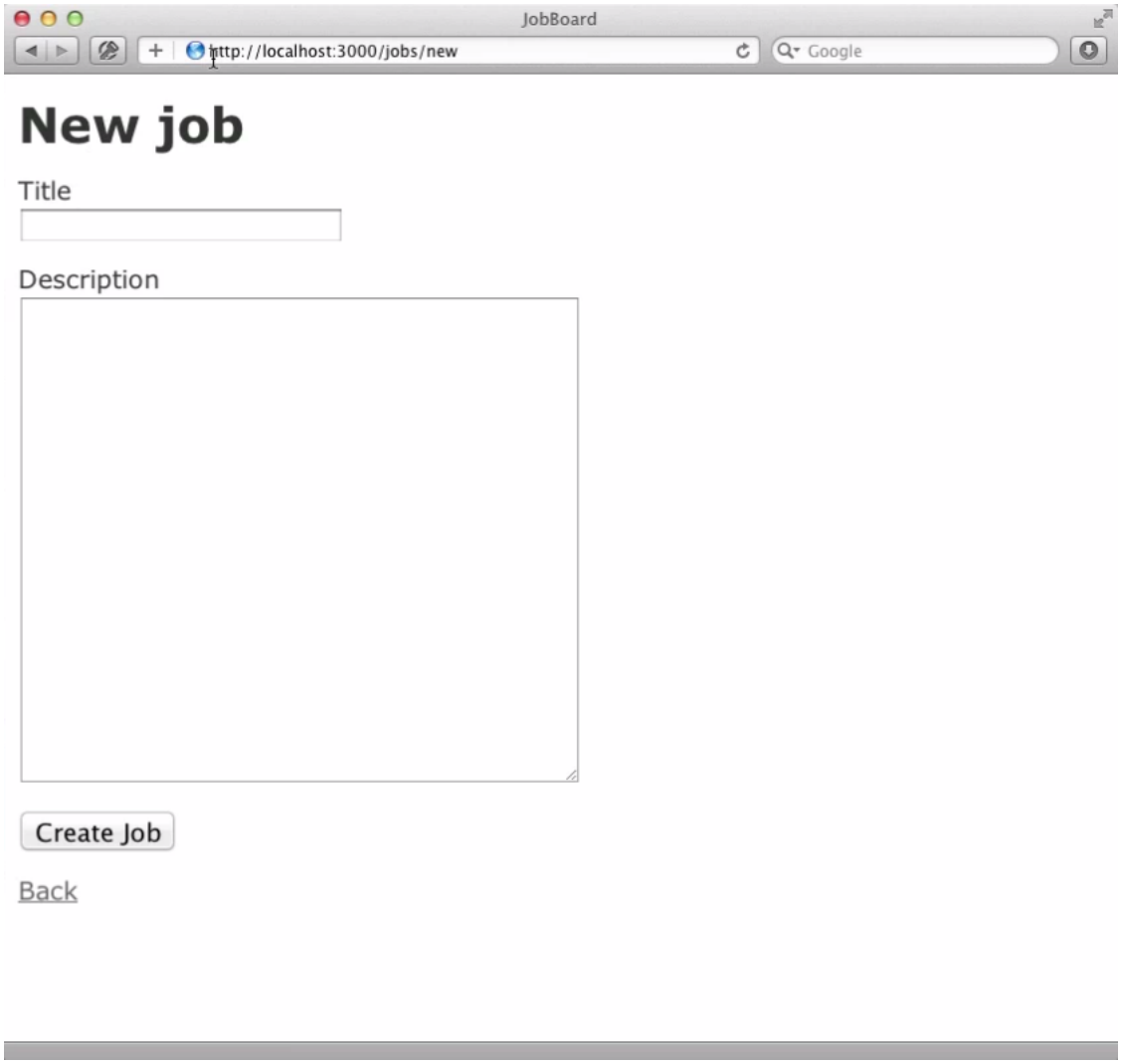
A saída do terminal nos indica que nossa tabela *jobs* foi criada com sucesso:

```
$ rake db:migrate
==  CreateJobs: migrating ======================================================
-- create_table(:jobs)
   -> 0.0022s
==  CreateJobs: migrated (0.0032s) =============================================
```

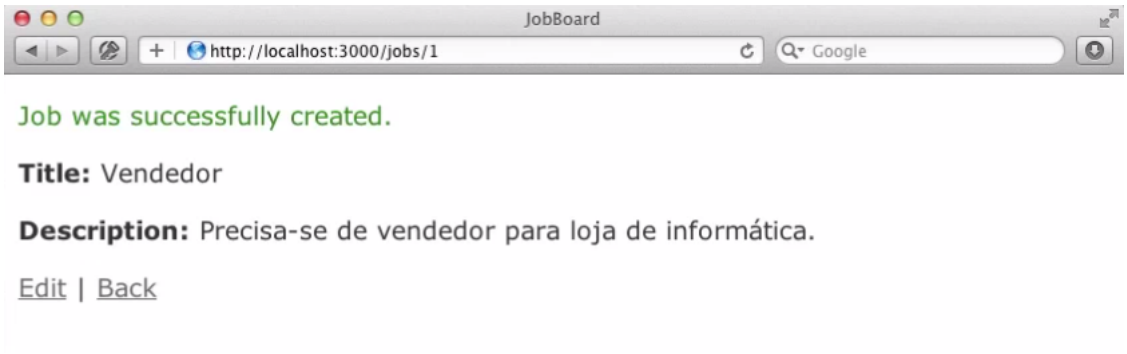
Com o *scaffold* de *job* gerado e nosso *database* pronto, vamos agora voltar para o navegador e visualizar a listagem de jobs. Basta acessar (http://localhost:3000/jobs)http://localhost:3000/jobs (http://localhost:3000/jobs), e você verá uma página como abaixo:



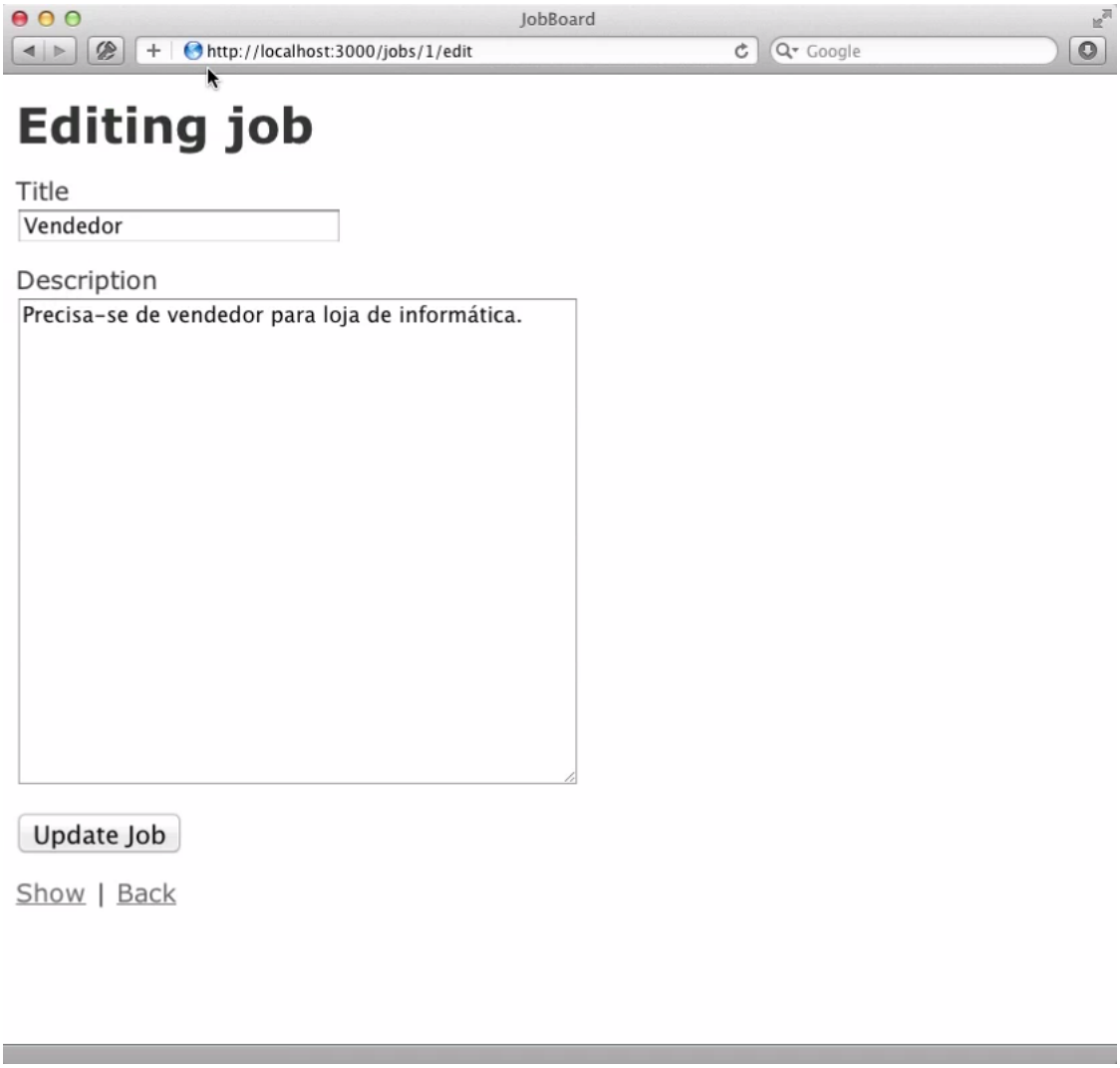
Através desta listagem de *jobs*, vamos agora criar um novo *job*. Clique no link "New Job", indo para uma tela que contém os campos que definimos para o *job*, *title* e *description*:



Olhando para a url no navegador, vemos que ela agora é (http://localhost:3000/jobs/new)http://localhost:3000/jobs/new (http://localhost:3000/jobs/new). Vamos cadastrar um novo *job* no banco de dados: preencha o campo "Title" com "Vendedor", e o campo "Description" com "Precisa-se de vendedor para loja de informática". Finalmente clique em "Create Job". Ao fazer isto, estas informações serão salvas pelo Rails dentro do banco de dados, na tabela *jobs*, e você verá uma tela que exibirá as informações do *job* que você acabou de preencher, já vindas do banco de dados, como esta:

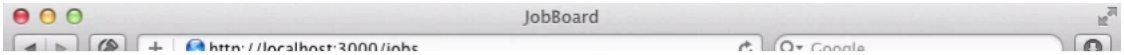


Perceba que agora estamos na url (http://localhost:3000/jobs/1)http://localhost:3000/jobs/1 (http://localhost:3000/jobs/1), a tela de exibição do *job*. A partir desta tela temos duas ações principais, através dos links na parte de baixo: "Edit", para voltar ao formulário com os dados do *job* e editá-los; e "Back", que nos levará para a mesma listagem de *jobs* que vimos antes. Vamos primeiro usar o "Edit", e visualizaremos novamente o formulário, mas agora já com os dados daquele *job* em específico preenchidos, para que possamos alterá-los:

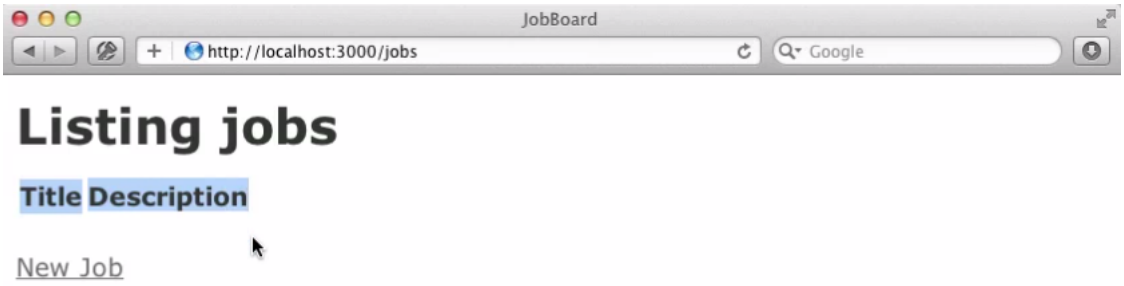


Verifique novamente a url no navegador: agora estamos em (http://localhost:3000/jobs/1/edit)http://localhost:3000/jobs/1/edit (http://localhost:3000/jobs/1/edit). Esta url nos diz que estamos não mais na página para criar um novo *job*, mas na página de edição do *job* **1**, mesmo elas parecendo semelhantes. Vamos alterar alguma informação do nosso *job*, e salvá-lo novamente. Modifique o campo "Description" para "Precisa-se de vendedor com bons conhecimentos para loja de informática.", e clique em "Update Job". Isso atualizará nosso *job* **1** no banco de dados, e nos levará de volta para a mesma tela de exibição que vimos antes, agora com os dados modificados:

Vamos agora visualizar a listagem de *jobs* novamente, clique no link "Back", e você verá que nosso *job* está aparecendo onde antes havia uma lista vazia:



Ótimo! Nosso próximo passo é testar como remover um *job* do banco de dados, e vamos fazer isso a partir desta mesma listagem, através do link "Destroy" à direita. Clique nele, você deve receber uma mensagem de confirmação com o texto "Are you sure?". Clique em OK, e você deverá ver a lista de *jobs*, agora vazia novamente pois o *job* foi removido do banco de dados:



Nota: como esta é uma ação destrutiva, é sempre uma boa prática pedir a confirmação ao usuário como a mensagem que acabamos de ver.

E com isso concluímos nosso *CRUD* de *jobs*. Fique à vontade para navegar, criar novos *jobs*, alterá-los, e excluí-los do banco de dados, para fixar bem o funcionamento do *CRUD*.

Ótimo, mais um objetivo alcançado! O *scaffold* gerou automaticamente toda a estrutura que precisávamos para criar, exibir, alterar, e remover *jobs* sem escrevermos uma linha de código sequer! E novamente, não tivemos que reiniciar o servidor em nenhum momento, pois o Rails recarregou as mudanças automaticamente, sucesso!

Só tem um pequeno problema: se navegarmos até a página para criar um novo *job*, e clicarmos em "Create Job" sem preencher nenhum dos campos, um *job* em branco será salvo no banco de dados! Não queremos isto, pois um *job* em branco não pode ser considerado válido, certo? Sendo assim, nosso próximo objetivo é bloquear a entrada dos campos em branco para o *job*.

Adicionando validações

Como vimos, um *job* com os campos *title* ou *description* em branco não pode ser considerado válido. Para criarmos esse bloqueio e não permitirmos que o usuário entre com dados em branco, vamos sujar as nossas mãos e escrever um pouco de código.

Lembra que comentamos que a maior parte do código de nossa aplicação fica no diretório `app` ? Pois é lá que vamos fazer as alterações necessárias. Para a nossa estrutura de *CRUD* funcionar, é necessário que algo descreva para o Rails os *jobs* que estão salvos no banco de dados. O *scaffold* gerou automaticamente um modelo chamado *Job* em

app/models/job.rb, que faz exatamente isto: descreve aquela nossa tabela *jobs* com os campos *title* e *description* do banco de dados, para que o Rails entenda como e aonde gravar e buscar estas informações, e ele faz isso através de um de seus componentes chamado `Active Record`, sobre o qual vamos discutir e aprender mais durante o curso.

Bom, vamos nos familiarizar com esse modelo. Abra o arquivo *app/models/job.rb* em um editor:

```
class Job < ActiveRecord::Base
  attr_accessible :description, :title
end
```

Agora, voltando ao nosso problema inicial, precisamos garantir que nossos *jobs* não terão dados em branco, ou seja, que os atributos *title* e *description* estarão sempre presentes, então vamos adicionar uma linha de validação ao model:

```
class Job < ActiveRecord::Base
  attr_accessible :description, :title
  validates_presence_of :description, :title
end
```

Acrescentamos apenas a linha que contém `validates_presence_of`. Essa linha sozinha vai garantir para nós que os atributos não estejam em branco. Com essa validação no lugar, volte para o navegador e faça um teste: tente criar um novo *job* como fizemos anteriormente, sem preencher nenhum dos campos. Ao clicar em "Create Job", o que acontece? Você deve estar visualizando uma tela como esta:

Agora que adicionamos a validação, quando tentamos salvar um *job* sem algum dos atributos preenchidos, o Rails verificará que aquela validação não passa, e não permitirá que o *job* seja gravado no banco de dados, retornando para a tela do formulário e exibindo algumas mensagens de erro que descrevem o que aconteceu. Basicamente as mensagens dizem que os campos *title* e *description* não podem estar em branco. Você pode tentar preencher apenas o *title* e enviar o formulário novamente, que o Rails rodará as validações da mesma forma e retornará para o formulário exibindo apenas o erro relacionado ao *description*. E se você preencher todos os campos e enviar novamente, o *job* será salvo normalmente.

Com apenas uma linha de código, conseguimos garantir que dados inválidos não sejam salvos no nosso banco de dados. Ótimo, não acha? Isso fecha o nosso primeiro capítulo, vamos a uma pequena revisão, e vejo você no próximo!

Para saber mais

-
- Se você quiser checar mais informações sobre o seu ambiente local, como versões do Ruby e do Rails, você pode acessar a página de informações que vimos neste capítulo, e clicar no link *About your application's environment* (Sobre o ambiente de sua aplicação).
-
- Nesta mesma página, existe uma série de links a direita que apontam para a documentação do Rails e do Ruby, que vale a pena explorar:
  -
  - **Rails Guides**: uma série de guias sobre os componentes do Rails.
  -
  - **Rails API**: documentação interna de todas as funcionalidades do Rails.

-
- **Ruby core** e **Ruby standard library**: documentação do Ruby.