CAPÍTULO 15

Algumas Gems Importantes

"A minoria pode ter razão, a maioria está sempre errada"
— Mikhail Aleksandrovitch Bakunin

15.1 – ENGINES

A partir do *Rails 3.1* foi criado um conceito para facilitar a extensão de uma aplicação chamado *Rails Engine*. Uma *engine* é uma espécie de mini aplicação *Rails* que permite isolar funcionalidades que podem ser reaproveitadas em qualquer aplicação *Rails* "comum" de uma forma estruturada e **plugável**.

Um bom exemplo de *Rails engine* é a *gem kaminari* que facilita muito a criação de uma lista paginada para exibição de registros. Basta adicionar a dependência do `Gemfile`, rodar o comando `bundle install` e pronto! Toda a funcionalidade de paginação será fornecida sem a necessidade de nenhum tipo de configuração.

Primeiro vamos editar o arquivo `Gemfile` para adicionar a dependência:

```
gem "kaminari"
```

Agora para que os dados sejam paginados a partir de uma busca no banco de dados, basta adicionar a invocação do método `page` do *kaminari*. Podemos usar essa funcionalidade na listagem de restaurantes como no código a seguir:

```
@restaurantes = Restaurante.all.page params['page']
```

O método `page` funciona como um finder normal. Suporta todas as opções previamente vistas, como `:conditions`, `:order` e `:include`. Note também que passamos um parâmetro para o método `page` que é a **página** a partir da qual queremos listar os registros. Esse parâmetro será gerado pelo próprio `kaminari` através de um *helper method* que invocaremos na view, mais adiante você verá como isso funciona.

O número de items por página é padronizado em 25, mas você pode customizar de duas formas. Através do método per_page nas classes `ActiveRecord::Base`:

```
class Restaurante < ActiveRecord::Base
  paginates_per 10
  # ...
end
```

Ou invocando o método `per` e passando como parâmetro o número máximo de registros por página. O método `per` está disponível no objeto criado pelo `kaminari` portanto só pode ser invocado após uma chamada ao método `page`. O código seguinte exemplifica o seu uso:

```
@restaurantes = Restaurante.all.page(params['page']).per(10)
```

E como dito, veja como usar o *helper method* do *kaminari* para gerar os links de paginação já com o parâmetro `page` que usamos durante os exemplos de paginação:

```
1 # index.html.erb
2 <% @restaurantes.each do |restaurante| %>
3   <li><%= restaurante.nome %></li>
4 <% end %>
5 <%= paginate @restaurantes %>
```

Uma coisa bem interessante sobre o `kaminari` é sua facilidade de customização. Você pode alterar o texto padrão usado por ele para geração de links, assim como a aparência desses mesmos links. Veja o repositório oficial para maiores detalhes: https://github.com/amatsuda/kaminari.

15.2 – Exercícios: Paginação com Kaminari

1. Adicione o *kaminari* ao seu projeto, no arquivo `Gemfile`

```
gem 'kaminari'
```

Execute o seguinte comando no terminal para instalar a gem:

```
$ bundle install
```

2. Abra o arquivo **app/controllers/restaurantes_controller.rb**. Na action index, troque a linha:

```
@restaurantes = Restaurante.order("nome")
```

por

```
@restaurantes = Restaurante.order("nome").page(params['page']).per(3)
```

3. Abra o arquivo **app/views/restaurantes/index.html.erb**. Adicione a linha abaixo após o fechamento da *tag table* (`</table>`):

```
<%= paginate @restaurantes %>
```

4. Abra a listagem de restaurantes e verifique a paginação.

15.3 – File Uploads: Paperclip

Podemos fazer upload de arquivos sem a necessidade de plugins adicionais, utilizando algo como o código abaixo:

```
File.open("public/"+path, "nome") do |f|
  f.write(params[:upload]['picture_path'].read)
end
```

O código acima recebe o binário do arquivo e faz o upload para a pasta public. Porém, ao fazer um upload seria interessante fazer coisas como redimensionar a imagem, gerar thumbs, associar com models ActiveRecord, etc.

Um dos primeiros plugins rails voltados para isso foi o *attachment_fu*. Hoje em dia o plugin mais indicado é o **Paperclip**. O Paperclip tem como finalidade ser um plugin de fácil uso com o modelos Active Record. As configurações são simples e é possível validar tamanho do arquivo ou tornar sua presença obrigatória. O paperclip tem como pré-requisito o *ImageMagick*,

Economizando espaço em disco

É boa prática ao configurar os tamanhos das imagens que armazenaremos,

sobrescrever o tamanho do estilo "original", dessa maneira evitamos que o Paperclip salve imagens muito maiores do que utilizaremos em nossa aplicação.

```
has_attached_file :foto, styles: {
  medium: "300x300>",
  thumb: "100x100>",
  original: "800x600>"
}
```

No exemplo acima, a imagem "original" será salva com tamanho máximo de 800x600 pixels, isso evita que armazenemos desnecessariamente em nossa aplicação, por exemplo, imagens de 4000x3000 pixels, tamanho resultante de uma foto tirada em uma máquina fotográfica digital de 12MP.

15.4 - Exercícios: Possibilitando upload de imagens

1. Para instalar o paperclip, abra o arquivo Gemfile e adicione a gem:

```
gem 'paperclip'
```

E no terminal, rode :

```
bundle install
```

2. Adicione o **has_attached_file** do paperclip na classe Restaurante. Vamos configurar mais uma opção que daremos o nome de *styles*. Toda vez que a view chamar a foto do restaurante com essa opção, o Rails buscará pelo thumb.

```
class Restaurante < ActiveRecord::Base
  # .. validações e relacionamentos

  has_attached_file :foto, styles:
    { medium: "300x300>", thumb: "100x100>" }
end
```

3. Vamos habilitar um restaurante à receber um parâmetro **foto**. Para isto, abra o arquivo **app/controllers/restaurantes_controller.rb** e altere o método **restaurante_params**:

```
def restaurante_params
  params.require(:restaurante).permit(:nome, :endereco, :especialidade,
:foto)
end
```

4. a. Precisamos de uma migration que defina novas colunas para a foto do

restaurante na tabela de restaurantes. O paperclip define 4 colunas básicas para nome, conteúdo, tamanho do arquivo e data de update.O Paperclip nos disponibiliza um gerador para criar esse tipo de migration, para utilizá-lo execute no terminal:

```
rails generate paperclip Restaurante foto
```

b. Rode a migration no terminal com:

```
rake db:migrate
```

5. Abra a view **app/views/restaurantes/_form.html.erb** e altere o formulário. Seu form deve ficar como o abaixo:

```
<%= form_for @restaurantes, html: {multipart: true} do |f| %>
  <!--outros campos-->
  <%= f.file_field :foto %>
<% end %>
```

6. Abra a view **app/views/restaurantes/show.html.erb** e adicione:

```
<p>
  <b>Foto:</b>
    <%= image_tag @restaurante.foto.url(:thumb) %>
</p>
```

Repare que aqui chamamos o thumb, que foi configurado como um dos styles do model. Suba o server e insira um novo restaurante com foto.

15.5 - Nokogiri

Nokogiri é uma biblioteca poderosa para manipulação de xhtml. Bastante útil para capturar conteúdo da internet que não tenha sido criado pensando em integração e não oferece formatos mais adequados para serem consumidos por outros sistemas, como json ou xml.

```
gem install nokogiri
```

`open-uri` é uma biblioteca que usaremos para fazer requisições http:

```
doc = Nokogiri::HTML(open('https://twitter.com/caelum'))
```

Analisando o html gerado pelo twitter, vemos que os tweets estão sempre dentro de elementos com a classe `"tweet"`. Além disso, dentro de cada `tweet`, a única parte que nos interessa é o conteúdo dos subitems de classe `"js-tweet-text"`, onde encontraremos as mensagens.

Podemos procurar estes itens com o Nokogiri, usando seletores *CSS*. Expressões *XPath* também poderiam ser usadas:

```
doc.css ".tweet .js-tweet-text"
```

Para imprimir cada um dos itens de uma maneira mais interessante:

```
items = doc.css ".tweet .js-tweet-text"
items.each do |item|
  puts item.content
end
```


15.6 – Exercícios: Extraindo dados do Twitter

1. Vamos fazer um leitor de tweets de um determinado usuário

   a. Crie um arquivo chamado **"twitter_reader.rb"**

   b. Adicione ás seguintes linhas:

```
require 'rubygems'
require 'open-uri'
require 'nokogiri'

doc = Nokogiri::HTML(open('https://twitter.com/caelum'))
items = doc.css ".content"
items.each do |item|
  autor = item.css(".fullname").first.content
  tweet = item.css(".js-tweet-text").first.content

  puts autor
  puts tweet
  puts
end
```

c. Teste usando o comando `ruby twitter_reader.rb`

CAPÍTULO ANTERIOR:

Ajax com Rails

PRÓXIMO CAPÍTULO:

Apêndice: Testes

Casa do Código

Twitter