CAPÍTULO 8

Rotas

8.1 – ROUTES.RB

Veja abaixo como criar uma nova rota na nossa aplicação através do arquivo **config/routes.rb**:

```
VotaPrato::Application.routes.draw do
  match 'inicio', controller: 'restaurantes', action: 'index', via: 'get'
end
```

`match` cria uma nova rota e tem 2 formas de ser utilizado. Na nova maneira, ele recebe dois parâmetros. O primeiro deve ser um hash, onde a chave do primeiro elemento é a rota que queremos mapear, e o valor para essa chave deve ser uma string com o nome do controlador e da action separado por um carácter '#'. O segundo parâmetro será um hash de opções, sendo obrigatório declarar a que métodos http a rota responderá, como no exemplo abaixo:

```
match 'inicio' => 'restaurantes#index', via: 'get'
```

Nesse exemplo o método `match` recebeu um hash onde a primeira chave é 'inicio' (rota a ser mapeada) e o valor para essa chave é 'restaurantes#index', ou seja, quando acessar `localhost:3000/inicio` o Rails vai executar a action `index` no controller `restaurantes`.

A outra abordagem é usar o método match recebendo dois parâmetros:

- url
- hash com o conjunto de parâmetros de requisição a serem preenchidos

No exemplo acima, para a url "localhost:3000/inicio" o método index do controlador de restaurantes (`RestaurantesController`) é chamado.

Qualquer parâmetro de requisição pode ser preenchido por uma rota. Tais parâmetros podem ser recuperados posteriormente através do hash de parâmetros da requisição, `params['nome']`.

Os parâmetros `:controller` e `:action` são especiais, representam o controlador e a action a serem executados.

Uma das características mais interessantes do rails é que as urls das rotas podem ser usadas para capturar alguns dos parâmetros:

```
match 'categorias/:nome', controller: 'categorias', action: 'show', via:
'get'
```

Neste caso, o parâmetro de requisição `params['nome']` é extraído da própria url!

> **Rotas padrão**
>
> Antigamente o Rails criava uma rota padrão. Hoje em dia ela continua no arquivo *config/routes.rb*, mas vem comentada por padrão:
>
> **match ':controller(/:action(/:id(.:format)))'**
>
> Esta rota padrão que nos permitiu usar o formato de url que temos usado até agora. O nome do controlador e a action a ser executada são retirados da própria url chamada.

Você pode definir quantas rotas forem necessárias. A ordem define a prioridade: rotas definidas no início tem mais prioridade que as do fim.

8.2 - Pretty URLs

A funcionalidade de roteamento embutida no Rails é bastante poderosa, podendo até substituir mod_rewrite em muitos casos. As rotas permitem uma grande flexibilidade para criação de urls que se beneficiem de técnicas de *Search Engine Optimization* (SEO).

Um exemplo interessante seria para um sistema de blog, que permitisse a exibição de posts para determinado ano:

```
match 'blog/:ano' => 'posts#list'
```

Ou ainda para um mês específico:

```
match 'blog/:ano/:mes' => 'posts#list'
```

Os parâmetros capturados pela url podem ter ainda valores *default*:

```
match 'blog(/:ano)' => 'posts#list', :ano => 2011
```

Para o último exemplo, a url 'http://localhost:3000/blog' faria com que a action `list` do controlador `PostsController` fosse chamada, com o `params['ano']` sendo 2011.

8.3 - Named Routes

Cada uma das rotas pode ter um nome único:

```
match 'blog/:ano' => 'posts#list', :as => 'posts'
```

O funcionamento é o mesmo de antes, com a diferença que usando o ':as' demos um nome à rota.

Para cada uma das *Named Routes* são criados automaticamente dois helpers, disponíveis tanto nos controladores quanto nas views:

- posts_path => '/blog/:ano'
- posts_url => 'http://localhost:3000/blog/:ano'

A convenção para o nome dos helpers é sempre `nome_da_rota_path` e `nome_da_rota_url`.

Você pode ainda ver o roteamento para cada uma das urls disponíveis em uma aplicação rails com a ajuda de uma task do rake:

```
$ rake routes
```

8.4 - REST - RESOURCES

REST é um modelo arquitetural para sistemas distribuídos. A ideia básica é que existe um conjunto fixo de operações permitidas (*verbs*) e as diversas aplicações se comunicam aplicando este conjunto fixo de operações em recursos (*nouns*) existentes, podendo ainda pedir diversas representações destes recursos.

A sigla REST vem de *Representational State Transfer* e surgiu da tese de doutorado de Roy Fielding, descrevendo as ideias que levaram a criação do protocolo HTTP. A web é o maior exemplo de uso de uma arquitetura REST, onde os verbos são as operações disponíveis no protocolo (GET, POST, DELETE, PUT, HEADER, ...), os recursos são identificados pelas URLs e as representações podem ser definidas através de *Mime Types.*

Ao desenhar aplicações REST, pensamos nos recursos a serem disponibilizados pela aplicação e em seus formatos, ao invés de pensar nas operações.

Desde o Rails 1.2, o estilo de desenvolvimento REST para aplicações web é encorajado pelo framework, que possui diversas facilidades para a adoção deste estilo arquitetural.

As operações disponíveis para cada um dos recursos são:

- **GET**: retorna uma representação do recurso
- **POST**: criação de um novo recurso
- **PUT**: altera o recurso
- **DELETE**: remove o recurso

Os quatro verbos do protocolo HTTP são comumente associados às operações de CRUD em sistemas *Restful*. Há uma grande discussão dos motivos pelos quais usamos *POST* para criação (*INSERT*) e *PUT* para alteração (*UPDATE*). A razão principal é que o protocolo HTTP especifica que a operação PUT deve ser *idempotente*, já POST não.

> **Idempotência**
>
> Operações idempotentes são operações que podem ser chamadas uma ou mais vezes, sem diferenças no resultado final. Idempotência é uma propriedade das operações.

A principal forma de suporte no Rails a estes padrões é através de rotas que seguem as convenções da arquitetura REST. Ao mapear um recurso no routes.rb, o Rails cria automaticamente as rotas adequadas no controlador para tratar as operações disponíveis no recurso (GET, POST, PUT e DELETE).

```
# routes.rb
resources :restaurantes
```

Ao mapear o recurso `:restaurantes`, o rails automaticamente cria as seguintes rotas:

- GET /restaurantes`:controller => 'restaurantes', :action => 'index'`
- POST /restaurantes`:controller => 'restaurantes', :action => 'create'`
- GET /restaurantes/new`:controller => 'restaurantes', :action => 'new'`
- GET /restaurantes/:id`:controller => 'restaurantes', :action => 'show'`
- PUT /restaurantes/:id`:controller => 'restaurantes', :action => 'update'`
- DELETE /restaurantes/:id`:controller => 'restaurantes', :action => 'destroy'`
- GET /restaurantes/:id/edit`:controller => 'restaurantes', :action => 'edit'`

Como é possível perceber através das rotas, todo recurso mapeado implica em sete métodos no controlador associado. São as famosas sete actions REST dos controladores rails.

Além disso, para cada rota criada, são criados os helpers associados, já que as rotas são na verdade *Named Routes*.

```
restaurantes_path          # => "/restaurantes"
new_restaurante_path       # => "/restaurantes/new"
edit_restaurante_path(3)   # => "/restaurantes/3/edit"
```

Rails vem com um `generator` pronto para a criação de novos recursos. O controlador (com as sete actions), o modelo, a migration, os esqueleto dos testes

(unitário, funcional e fixtures) e a rota podem ser automaticamente criados.

```
$ rails generate resource comentario
```

O gerador de scaffolds do Rails 2.0 em diante, também usa o modelo REST:

```
$ rails generate scaffold comentario conteudo:text author:string
```

Na geração do scaffold são produzidos os mesmos artefatos de antes, com a adição das views e de um layout padrão.

Não deixe de verificar as rotas criadas e seus nomes (*Named Routes*):

```
$ rake routes
```

8.5 – Actions extras em Resources

As sete actions disponíveis para cada resource costumam ser suficientes na maioria dos casos. Antes de colocar alguma action extra nos seus resources, exercite a possibilidade de criar um novo resource para tal.

Quando necessário, você pode incluir algumas actions extras para os resources:

```
resources :comentarios do
  member do
    post :desabilita
  end
# url: /comentarios/:id/desabilita
# named_route: desabilita_comentario_path
end
```

`:member` define actions que atuam sobre um recurso específico: */comentarios/1/desabilita*. Dentro do bloco member usar dessa forma `method :action`, onde `method` pode ser `get`, `post`, `put`, `delete` ou `any`.

Outro bloco que pode ser usado dentro de um resource é o `collection` que serve para definir actions extras que atuem sobre o conjunto inteiro de resources. Definirá rotas do tipo */comentarios/action*.

```
resources :comentarios do
  collection do
    get :feed
  end
# url: /comentarios/feed
# named_route: feed_comentarios_path
end
```

Para todas as actions extras, são criadas *Named Routes* adequadas. Use o `rake routes` como referência para conferir os nomes dados às rotas criadas.

8.6 - Para Saber Mais - Nested Resources

Quando há relacionamentos entre resources, podemos aninhar a definição das rotas, que o rails cria automaticamente as urls adequadas.

No nosso exemplo, :restaurante has_many :qualificacoes, portanto:

```
# routes.rb
resources :restaurantes do
  resources :qualificacoes
end
```

A rota acima automaticamente cria as rotas para qualificações específicas de um restaurante:

- GET /restaurante/:restaurante_id/qualificacoes:controller => 'qualificacoes', :action => 'index'
- GET /restaurante/:restaurante_id/qualificacoes/:id:controller => 'qualificacoes', :action => 'show'
- GET /restaurante/:restaurante_id/qualificacoes/new:controller => 'qualificacoes', :action => 'new'
- ...

As sete rotas comuns são criadas para o recurso `:qualificacao`, mas agora as rotas de `:qualificacoes` são sempre específicas a um `:restaurante` (todos os métodos recebem o `params['restaurante_id']`).

8.7 - Exercício: Rotas

1. Abra o arquivo **config/routes.rb** e crie a seguinte rota:

```
VotaPrato::Application.routes.draw do
  match 'ola' => 'ola_mundo#index', via: 'get'
end
```

2. Definimos uma rota em nossa aplicação quando queremos disponibilizar algum recurso. Já temos nossa rota definida, temos agora que criar esse recurso. Para isso, criaremos um novo controller. Não se preocupe, falaremos mais sobre rotas no próximo capítulo.

   a. Gere o controller usando o seguinte comando:

   ```
   $ rails generate controller OlaMundo
   ```

   b. Abra o arquivo **app/controllers/ola_mundo_controller.rb**

   c. Dentro da classe, crie o metodo index:

   ```
   def index
     render text: "Olá mundo"
   end
   ```

   O metodo *render* deverá apresentar o texto na tela.

3. Inicie o servidor do rails usando o comando **rails server** e acesse a url **http://localhost:3000/ola**

Verifique o que foi apresentado na tela

CAPÍTULO ANTERIOR:

Active Record

PRÓXIMO CAPÍTULO:

Controllers e Views

Blog Caelum

Cursos Online

Facebook

Newsletter

Casa do Código

Twitter