Routing

Introduction

The router is the doorman of your application. When an HTTP request arrives from the user's browser, it needs to know which controller action (method) should be run. Should we display the "new user" webpage? Should we edit an existing user with whatever data got sent along?

The Router is basically just a matching service. It looks at the HTTP verb (GET, POST, PUT, DELETE) and the URL that it being requested and matches it with the appropriate controller action to run. It's a pretty simple function but an essential one. If it can't find a route that matches the request, your application will throw an error.

The other handy thing that goes on when a request enters your application is that Rails grabs all the parameters that came with it and makes them available for you in a special hash called `params` that you can later use in your controller. That's good for things like form submissions so that you later can use that form data to create or modify objects.

If you open the routes file in your Rails app (located in `config/routes.rb`), you'll see a huge blob of comments that do a pretty good job of explaining how it works, so you're never in much danger of losing your way.

Lots of training courses and tutorials kind of gloss over routes, and they seem quite easy in hindsight, but I remember learning Rails and getting hung up on what exactly is going on. Luckily, typing `$ rake routes` into the command line will give you an output of all the routes that are available to your application. In this section we'll go into what's actually happening with this file.

Points to Ponder

Look through these now and then use them to test yourself after doing the assignment

- What is the "Root" route?
- What are the seven RESTful routes for a resource?
- Which RESTful routes share the same URL but use different verbs?
- How do you specify an ID or other variable in a route?
- How can you easily write all seven RESTful routes in Rails?
- What is the Rails helper method that creates the HTML for links?

Root

The most important (and simplest) route in your file is the root url... where should users be deposited when they land on `http://supercutekittenphotos.com`? Just tell Rails which controller and action to map that route to, and it is so:

```
    root :to => "kittens#index"  #kittens controller, index action (method)
```

Remember, when we say "action" we really mean "the method inside the controller that is called that", e.g. the `index` action is just the `index` method that's defined in the KittensController

RESTful Routes

If you recall our earlier discussion about REST, there are basically seven main types of actions that you can (and should) do to a "resource", or an object like a blog post or user... something with its own database model. From that discussion, they are:

1. GET all the posts (aka **"index"** the posts)
2. GET just one specific post (aka **"show"** that post)

3.GET the page that lets you create a new post (aka view the **"new"**post page)

4.POST the data you just filled out for a new post back to the server so it can create that post (aka **"create"** the post)

5.GET the page that lets you edit an existing post (aka view the **"edit"**post page)

6.PUT the data you just filled out to edit the post back to the server so it can actually perform the update (aka **"update"** the post)

7.DELETE one specific post by sending a delete request to the server (aka **"destroy"** the post)

The highlighted words correspond to standard Rails controller actions!

Each of these represents a "RESTful" route, and so it makes sense that you'll need a way to write these in your Router file so the requests they represent are actually routed to the proper action of your controller (in this case, the "Posts" controller). One way to write them out would be the long way:

```
get "/posts" => "posts#index"
get "/posts/:id" => "posts#show"
get "/posts/new" => "posts#new"
post "/posts" => "posts#create"  # usually a submitted form
get "/posts/:id/edit" => "posts#edit"
put "/posts/:id" => "posts#update" # usually a submitted form
delete "/posts/:id" => "posts#destroy"
```

Each of these routes is basically a Ruby method that matches that particular URL and HTTP verb with the correct controller action. Two things to notice:

1.The first key thing to notice is that several of those routes submit to the SAME URL... they just use different HTTP verbs, so Rails can send them to a different controller action. That trips up a lot of beginners.

2.The other thing to notice is that the "id" field is prepended by a colon... that just tells Rails "Look for anything here and save it as the ID in the params hash". It lets you submit a GET request for the first post and the fifth post to the same route, just a different ID:

```
/posts/1  # going to the #show action of the Posts controller
/posts/5  # also going to the #show action of PostsController
```

You will be able to access that ID directly from the controller by tapping into the params hash where it got stored.

The Rails Way to Write Restful Routes

Rails knows you want to use those seven actions all the time... so they came up with a handy helper method which lets you do in one line what we just wrote in seven lines in our resources file:

```
# in config/routes.rb
  ...
  resources :posts
  ...
```

That's it. That is a Ruby method which basically just outputs those seven routes we talked about before. No magic. You see it a whole lot, now you know what it does.

Rake Routes and Route Helpers

With that above line in my routes file, what do my routes look like? If you type `$ rake routes` on the command line, it'll output all the routes your application knows, which look like:

```
    edit_post  GET  /posts/:id/edit(.:format)  posts#edit
```

You can see the incoming HTTP verb and URL in the middle columns, then the controller action they map to on the right, which should all be quite familiar because you just wrote it in the routes file. The `(.:format)` just means that it's okay but not required to specify a file extension like `.doc` at

the end of the route... it will just get saved in the `params` hash for later anyway. But what's on the leftmost column? That's the "name" of the route.

There are a lot of situations where you want to be able to retrieve the URL for a particular route, like when you want to show navigation links on your webpage (do NOT hard code the URLS, because you'll be out of luck when you decide to change the URLs and have to manually go in and change them yourself). Rails gives you a helper method that lets you create links called `link_to`, but you'll need to supply it with the text that you want to show and the URL to link it to.

```
    link_to "Edit this post", edit_post_path(3) # don't hardcode 3!
```

We're jumping a little bit ahead, but in this case, the second argument is supposed to be a path or a URL, so we use the path helper method to generate that. `edit_post_path(3)` will generate the path `/posts/3/edit`.

Rails automatically generates helper methods for you which correspond to the names of all your routes. These methods end with `_path` and `_url`. `path`, as in `edit_post_path(3)`, will generate just the path portion of the URL, which is sufficient for most applications. `url` will generate the full URL.

Any routes which require you to specify an ID or other parameters will need you to supply those to the helper methods as well (like we did above for edit). You can also put in a query string by adding an additional parameter:

```
    post_path(3, :referral_link => "/some/path/or/something")
```

Now the `:referral_link` parameter would be available in your `params` hash in your controller in addition to the normal set of parameters.

Routes go to Controller Actions!

Just to drive home that routes correspond directly to controller actions, a very simple sample controller which would fulfill the above routes generated by `resources :posts` might look like:

```
# in app/controllers/posts
class PostsController < ApplicationController

  def index
    # very simple code to grab all posts so they can be
    # displayed in the Index view (index.html.erb)
  end

  def show
    # very simple code to grab the proper Post so it can be
    # displayed in the Show view (show.html.erb)
  end

  def new
    # very simple code to create an empty post and send the user
    # to the New view for it (new.html.erb), which will have a
    # form for creating the post
  end

  def create
    # code to create a new post based on the parameters that
    # were submitted with the form (and are now available in the
    # params hash)
  end

  def edit
    # very simple code to find the post we want and send the
    # user to the Edit view for it(edit.html.erb), which has a
    # form for editing the post
  end

  def update
    # code to figure out which post we're trying to update, then
    # actually update the attributes of that post.  Once that's
    # done, redirect us to somewhere like the Show page for that
    # post
  end
```

```
    def destroy
      # very simple code to find the post we're referring to and
      # destroy it.  Once that's done, redirect us to somewhere fun.
    end

  end
```

I Don't Want All Seven Routes!

Sometimes you just don't want all seven of the RESTful routes that `resources` provides. Easy, either specify just the ones you want using `only` or just the ones you DON'T want using `except`:

```
  resources :posts, :only => [:index, :show]
  resources :users, :except => [:index]
```

Non-RESTful Routes

Of course, you don't have to do everything the RESTful way. You probably should, but there are times that you want to make up your own route and map it to your own controller action. Just follow the examples we gave at the top for RESTful routes:

```
  get '/somepath' => 'somecontroller#someaction'
```

... of course, the `config/routes.rb` comments should be helpful to you here as well.

Assignment

You should have a good sense of what's going on in the routes file by now but probably also have plenty of questions. The Rails Guides to the rescue!

1.Read the Rails Guides chapter on Routing, sections 1-2.5, 3.1-3.4, and 4.6.

Additional Resources

This section contains helpful links to other content. It isn't required, so consider it supplemental for if you need to dive deeper into something

•CodeSchool's Rails 4 Zombie Outlaws - Level 1 is free and goes into routes.

•CodeSchool's Surviving APIs with Rails - Level 1 is free and gets into REST, Routes, Constraints, and Namespaces.