**Routing I: Basics**

**The Purpose of the Rails Router**

The Rails router recognizes URLs and chooses a controller method to which the request is dispatched for processing. The router is the part of Rails which receives a GET request for /patients/17 and realizes that PatientsController#show should be called for Patient #17. Note that the router matches on both HTTP method and path.

The router defines the structure of your application's API. The router defines what the valid paths are and decides what code to run to construct the response.

**Resource Routing: the Rails Default**

Say that we have a Photo model, and we would like to begin building a PhotosController to display photos, create new ones, edit existing ones, delete old ones...

**Resource routing** will generate a mapping from a set of conventional url paths to a set of conventional controller actions. Let's create our first resource routing like so:

# config/routes.rb

FlickrClone::Application.routes.draw do

resources :photos

end

This single line in config/routes.rb will generate a map of the following requests for URLs to a set of controller actions in the PhotosController.

| **HTTP Verb** | **Path** | **action** | **used for** |
| --- | --- | --- | --- |
| GET | /photos | index | display a list of all photos |
| GET | /photos/new | new | return an HTML form for creating a new photo |
| POST | /photos | create | upload and create a new photo |
| GET | /photos/:id | show | display a specific photo |
| GET | /photos/:id/edit | edit | return an HTML form for editing a photo |
| PATCH or PUT | /photos/:id | update | update a specific photo |
| DELETE | /photos/:id | destroy | delete a specific photo |

The areas in the path that start with a : (i.e., :id) are named dynamic segments; :id can match any string not containing a /. GET /photos/5 and GET /photos/203 both map to the same controller action (show). The controller will be able to access the value of :id, which will be either 5 or 203, respectively.

It is typical that :id be the primary key of the model to show/edit/update/destroy.

Your routes are now set up: you can begin writing your controller actions to implement these actions!

**Follow the conventions**

Controllers are **always** named in the plural: PhotosController, UsersController, etc. When defining a plural resource (resources), use the plural name of the model/controller (:photos).

We will later see that you can also declare singular resources, but don't worry about it yet. Even with singular resources, we will continue to name our controllers in the plural.

**What does RESTful mean?**

We will talk later about what REST means, and what it means to define RESTful routes. However, at present, please note that the structure of the URLs and methods all specify creating/reading/updating/destroying a **resource**, which is a Photo.

The REST philosophy is that even more complicated actions, like "liking a photo", should be thought of in terms of **CRUD** (create/read/update/destroy) actions on resources. For instance, instead of creating a custom, non-RESTful controller action to "like" a photo, we might create a new resource, a Like object, which we could either create/destroy in the normal way.

This part doesn't need to make a lot of sense right now; you kind of have to live the experience before you can buy into the philosophy. But keep in the back of your mind the idea that in REST, URLs refer to either collections of resources or single instances of resources. The different HTTP methods specify the limited number of things you can do to a collection/instance (create, read, update, destroy).

**Paths and Routing Helpers**

Specifying a resource route will also create a number of **routing helper methods** that your controllers and views can use to build URLs. In the case of resources :photos:

| **method** | **url** |
| --- | --- |
| photos\_url | http://www.example-site.com/photos |
| new\_photo\_url | http://www.example-site.com/photos/new |
| photo\_url(@photo) | http://www.example-site.com/photos/#{@photo.id} |
| edit\_photo\_url(@photo) | http://www.example-site.com/photos/#{@photo.id}/edit |

Always prefer the routing helpers to building your own URLs through string interpolation. The routing helpers are less error prone and tedious. They also are more semantically clear, and more easily changed. If you build URLs by hand in Rails, **you're doing it wrong**.

Because the router looks at the HTTP verb when routing a request for a path, four URLs map to seven actions. Many methods that take a URL will also accept a :method option to specify the option. For instance, to create an HTML button that will destroy a photo, we write

<form action="<%= photo\_url(@photo) %>" method="POST">

<input type="hidden" value="delete" name="\_method" />

<input type="submit" value="Delete photo" />

</form>

<%= button\_to "Delete photo", photo\_url(@photo), method: :delete %> <!-- equivalent to the above -->

Finally, note that you can embed query-string options into the url-helpers easily:

photos\_url(recent: true) == http://www.example-site.com/photos?recent=true

On the streets, you will see a \_path version of these helpers; the \_path version just gives you the path component, not the full URL (/photo, /photo/:id/edit). Make life easy and never use \_path; just be consistent and use \_url all the time.

| **HTTP Verb** | **Controller#action** | **Path** | **URL** |
| --- | --- | --- | --- |
| GET | Photos#index | /photos | http://www.example-site.com/photos |
| GET | Photos#edit | /photos/1/edit | http://www.example-site.com/photos/1/edit |

**Inspecting and Testing Routes**

To get a complete list of the available routes in your application, execute the rails routes command in your terminal. This will list all of your routes, in the same order that they appear in routes.rb. For each route, you'll see:

* The route name; you can tack \_url after this to get the routing helper,
* The HTTP verb used,
* The URL pattern to match,
* The controller#action to route to

For example, here's a small section of the rails routes output for a RESTful route:

users GET /users(.:format) users#index

POST /users(.:format) users#create

new\_user GET /users/new(.:format) users#new

edit\_user GET /users/:id/edit(.:format) users#edit

Notice that the create route does not repeat the URL name users. Rails does this to reduce redundancy, since the create route has the same URL structure as the index action.

For instance, notice that there is no create\_user URL name. That's because the create action corresponds to a POST request to /users. /users already has a URL helper: users\_url. We do not have different URL helpers for index and create actions because they are both activated by different request methods (GET/POST) to the same URL (/users).

**TIP**: You'll find that the output from rails routes is much more readable if you widen your terminal window until the output lines don't wrap. Enlarge your terminal appropriately.

**Using**root

You can specify the controller action that Rails should run for GET / by using the root method:

root to: 'posts#index'

This invokes the PostsController's index method when the root URL is requested.

**Additional Resources**

* [Rails Guide on Routing](http://guides.rubyonrails.org/routing.html)