Structuring Sinatra Applications

Posted on November 29, 2014. Tagged with: sinatra, and ruby.

Sinatra is a cool little framework for building web tools in Ruby. I've been using it for years to build out little examples, tools like moviesapi, experimenting with REST and Hypermedia API design and now I'm using it to build a relatively complex web service that provides the core of a new project I'm part of.

But I've never written down how (and why) I structure projects in the way I do. Being such a tiny framework, Sinatra doesn't impose any pattern on you (both a blessing and a curse) and after several false starts (especially on the larger projects), I've found I've settled into a pattern which works very well.

Small Ones

The little applications are easy, it's just one file called `app.rb`. This is most often a 'classic' application, but sometimes it can be a 'modular' one instead. For example:

```
# app.rb
require 'sinatra'

get '/' do
  'Hello World!'
end
```

The directory structure will usually end up looking something like this:

```
.
├── Rakefile
├── app.rb
├── config.ru
└── spec
    ├── app_spec.rb
    └── spec_helper.rb
```

Directories like `models` or `lib` fit quite well into the top level like `spec` does. moviesapi provides a good working example of this.

Big Ones

The structure for small applications doesn't scale out very well because of the focus on one single file. Once I've reached (or expect to reach) the stage where I have several well grouped set of routes, I start to follow a Rails-like MVC pattern. This splits the routes into separate classes, almost exactly how Rails handles things. Some of this originally came from Sinatra: Up and Running which gives a nice overview.

(Sometimes, using Rails is still a better approach than replicating it's structure like this. I keep with Sinatra for machine facing projects like APIs once they get this big, anything else would be Rails.)

This structure looks a lot like this:

```
.
├── Rakefile
├── app
│   ├── controllers
│   │   ├── application_controller.rb
│   │   └── example_controller.rb
│   ├── helpers
│   │   └── application_helper.rb
│   └── views
│       ├── example.erb
│       ├── layout.erb
│       └── not_found.erb
├── config.ru
└── spec
    ├── controllers
    │   ├── application_controller_spec.rb
    │   └── example_controller_spec.rb
    ├── helpers
    │   └── application_helper_spec.rb
    └── spec_helper.rb
```

Here, `app.rb` has been broken out into a directory called `app` instead. Inside this are the `controllers`, `helpers`, `views` (and probably also `models`, too) which make up the application. In practice, each of the controllers inherit from `Sinatra::Base` and so we keep the simple route call feature of the original style and by inheriting from a common `application_controller` we can provide a few common configuration settings all around.

The `ApplicationController` subclass could be as simple as just: `class ApplicationController; end`, which should keep each of our controller subclasses easy

to reuse and test.

The `ApplicationController` and `ExampleController` look like this:

```ruby
# application_controller.rb
class ApplicationController < Sinatra::Base
  helpers ApplicationHelper

  # set folder for templates to ../views, but make the path absolute
  set :views, File.expand_path('../../views', __FILE__)

  # don't enable logging when running tests
  configure :production, :development do
    enable :logging
  end
end
```

```ruby
# example_controller.rb
class ExampleController < ApplicationController
  get '/' do
    'Example!'
  end
end
```

`config.ru` is used to bring everything up together and this looks like this:

```ruby
# config.ru
require 'sinatra/base'

# pull in the helpers and controllers
Dir.glob('./app/{helpers,controllers}/*.rb').each { |file| require file }

# map the controllers to routes
map('/example') { run ExampleController }
map('/') { run ApplicationController }
```

My biggest use of this structure has been on a recent project with a lot of routes that have been exposed. It's worked really well so far. That said, a lot of my projects are of the much smaller type — the kind of projects where Sinatra has always worked really well.