skillcrush

RAILS VS. SINATRA

cheatsheet

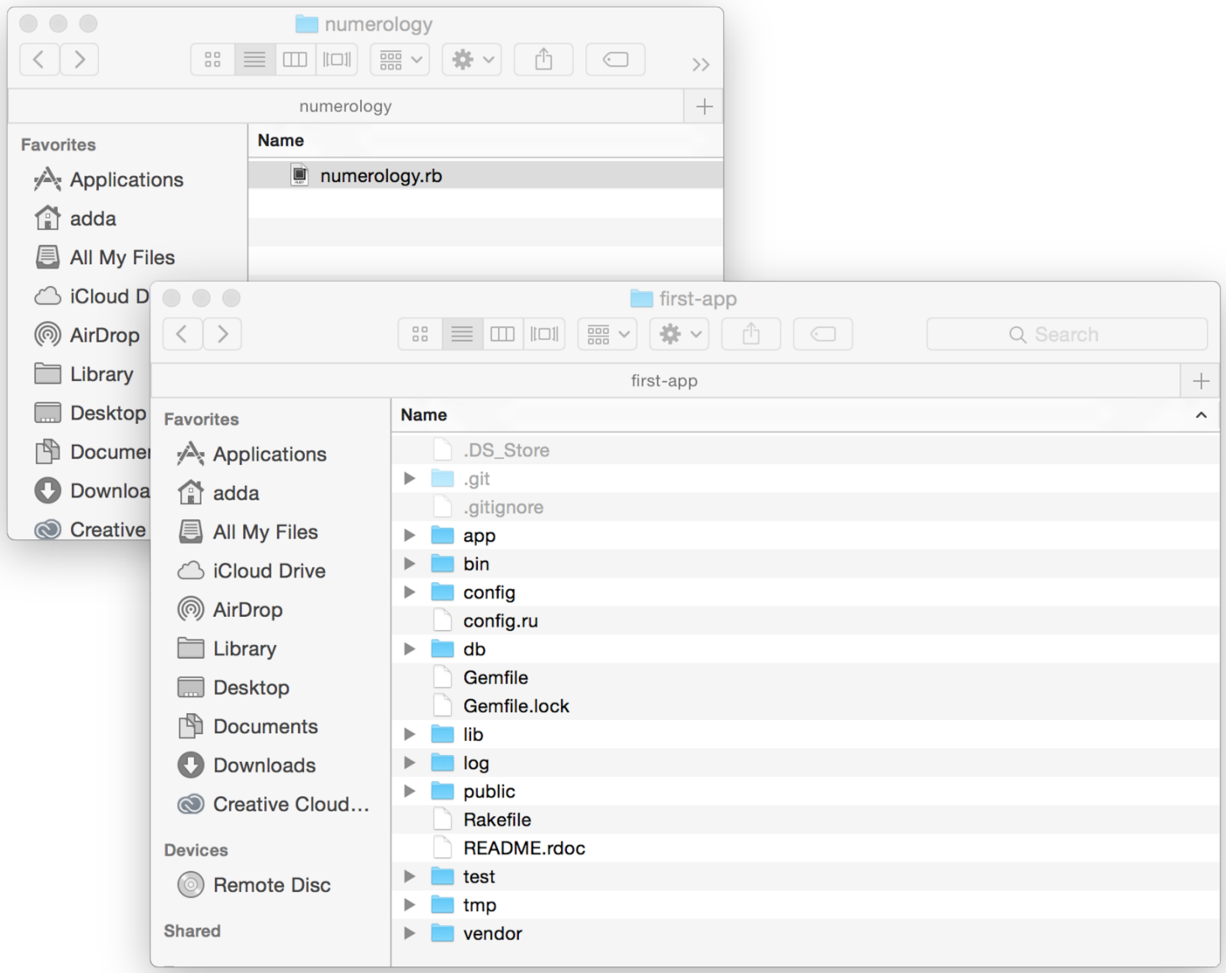
RAILS VS. SINATRA

For the past four weeks you have been working with the lovely Ruby web framework Sinatra. In that time you took Sinatra from one file to a whole, beautiful, best practice adherent web framework.

But now it’s time for Rails! In this cheatsheet we are going to cover some of the major differences between Sinatra and Rails so you can draw on your experience with Sinatra to understand Rails.

ONE FILES VS. MANY REQUIRED FILES & FOLDERS

The single biggest difference between Sinatra and Rails is really simple. You can run Sinatra from ONE file, while Rails always comes with a bunch of files:



numerology

Name
numerology.rb

first-app

Name
.DS_Store
.git
.gitignore
app
bin
config
config.ru
db
Gemfile
Gemfile.lock
lib
log
public
Rakefile
README.rdoc
test
tmp
vendor

And all of those files are there for an important reason, so don't even THINK about deleting any of them!!

FLEXIBLE VS. HIGHLY OPINIONATED

What this means in practice is that Sinatra is an incredibly FLEXIBLE framework that doesn't impose upon you any specific folder structure or organization system.

Conversely, Rails is considered a "highly opinionated" web framework, meaning that Rails has a certain set of conventions that it expects you to adhere to. And, honestly, you will be best served if you do as Rails says ;) The reason is that Rails is designed in accordance with what are widely considered industry best practices, which means that it makes it easy for you to do the right thing the right way.

Meaning better code! And better skills for you!

MVC VS. MVC

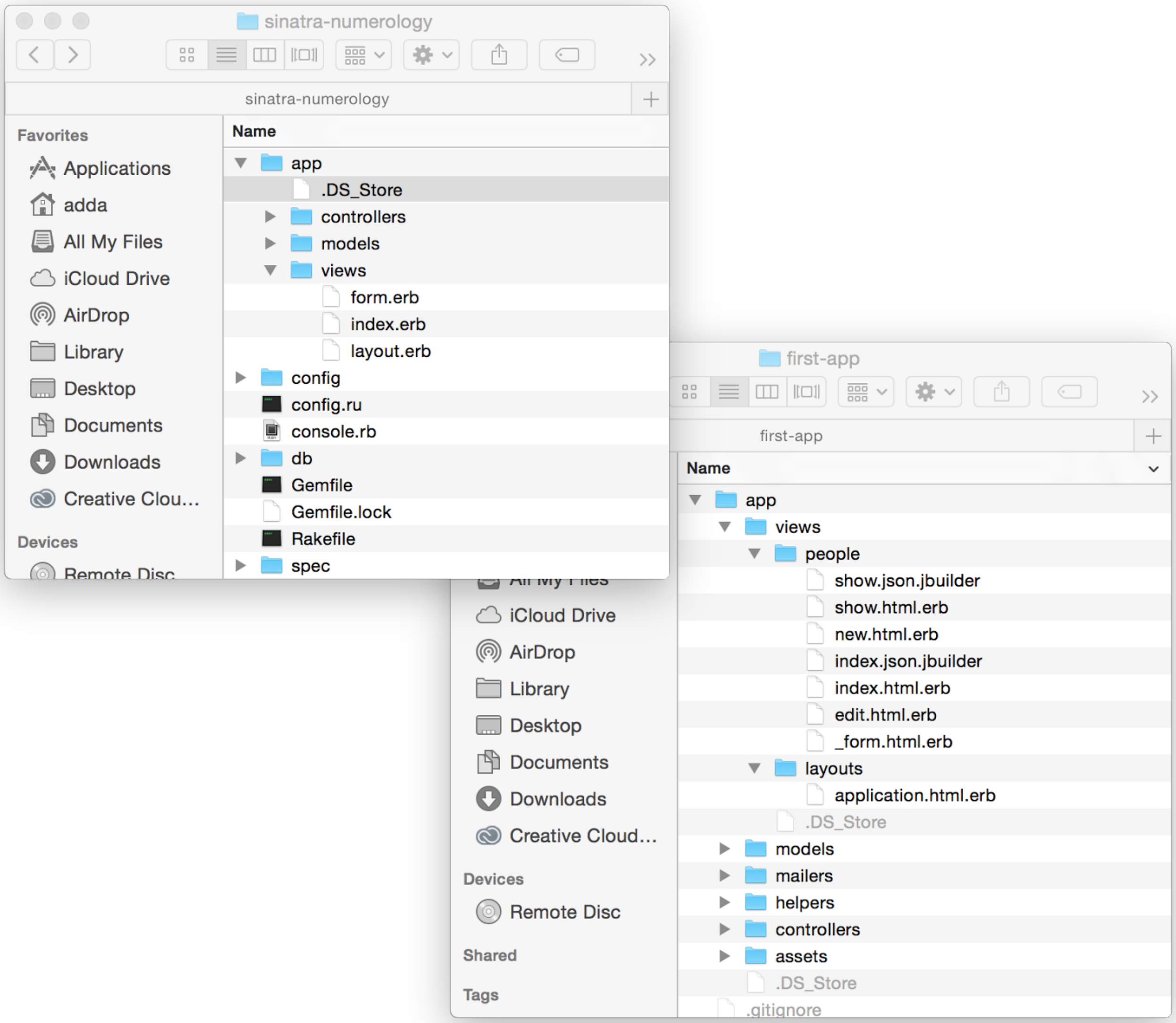
That said, both Sinatra and Rails use the MVC framework. The difference is that Sinatra doesn't force you to split your models, views, and controllers into separate files and folders unless you WANT to.

But as you know, doing this makes for cleaner, more organized code, and a happier YOU.

So in practice, you will usually use the MVC framework in both your Sinatra & Rails applications.

ERB TEMPLATES VS. ERB TEMPLATES

One thing that Sinatra and Rails share in common is that they both use the ERB templating engine. If you prefer you can also configure both Sinatra and Rails to use HAML, an another templating engine, instead. For both Sinatra & Rails, if you are going to go the templating route you need to put all your templates inside your views folder:



```
sinatra-numerology
▼ app
    .DS_Store
  ▶ controllers
  ▶ models
  ▼ views
      form.erb
      index.erb
      layout.erb
▶ config
  config.ru
  console.rb
▶ db
  Gemfile
  Gemfile.lock
  Rakefile
▶ spec
```

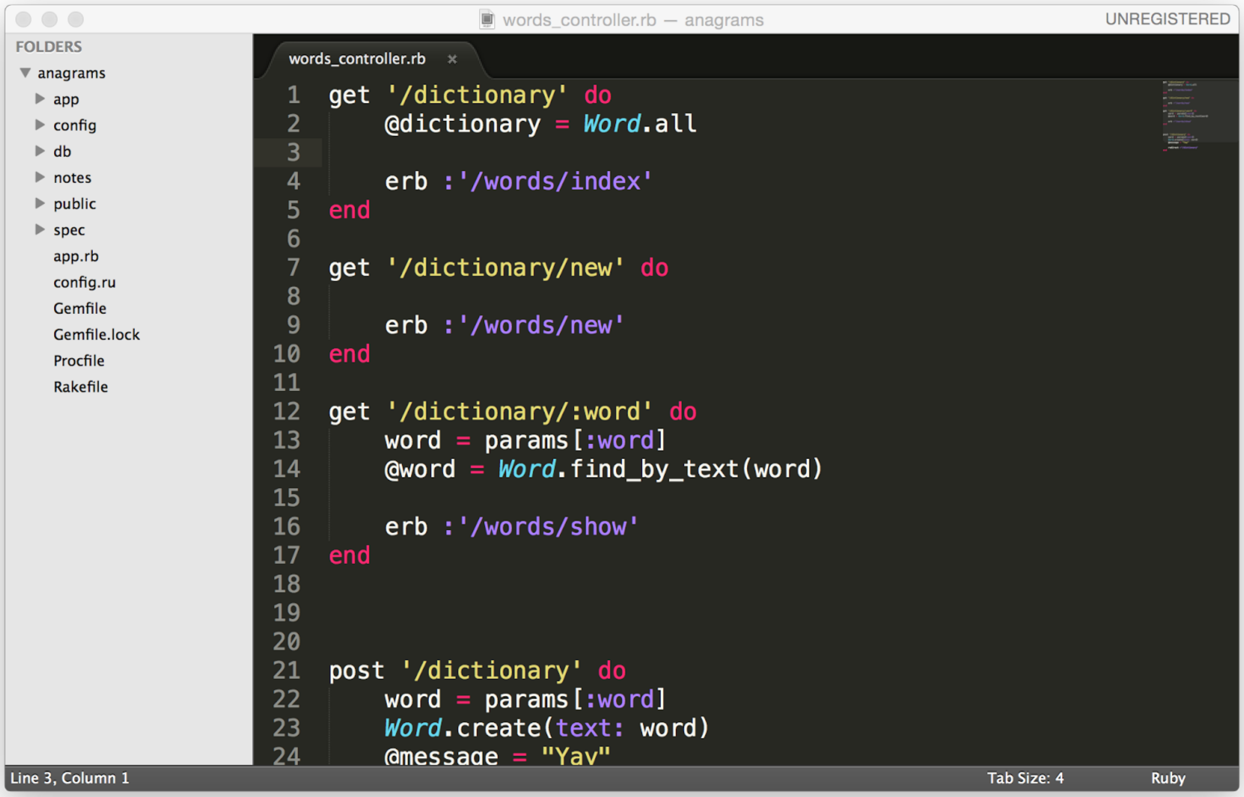
```
first-app
▼ app
  ▼ views
    ▼ people
        show.json.jbuilder
        show.html.erb
        new.html.erb
        index.json.jbuilder
        index.html.erb
        edit.html.erb
        _form.html.erb
    ▼ layouts
        application.html.erb
      .DS_Store
  ▶ models
  ▶ mailers
  ▶ helpers
  ▶ controllers
  ▶ assets
    .DS_Store
  .gitignore
```

In Rails the template names are slightly different in that they have a .html added to their file name. This is because Rails makes it possible to also generate templates in other formats, but don't worry about that for now!

skillcrush

CONTROLLERS VS. CONTROLLERS & ROUTES

From a daily coding perspective, one of the biggest differences between Rails and Sinatra is how they handle paths, controller actions, and routes.
In Sinatra, everything is handled in the controller:

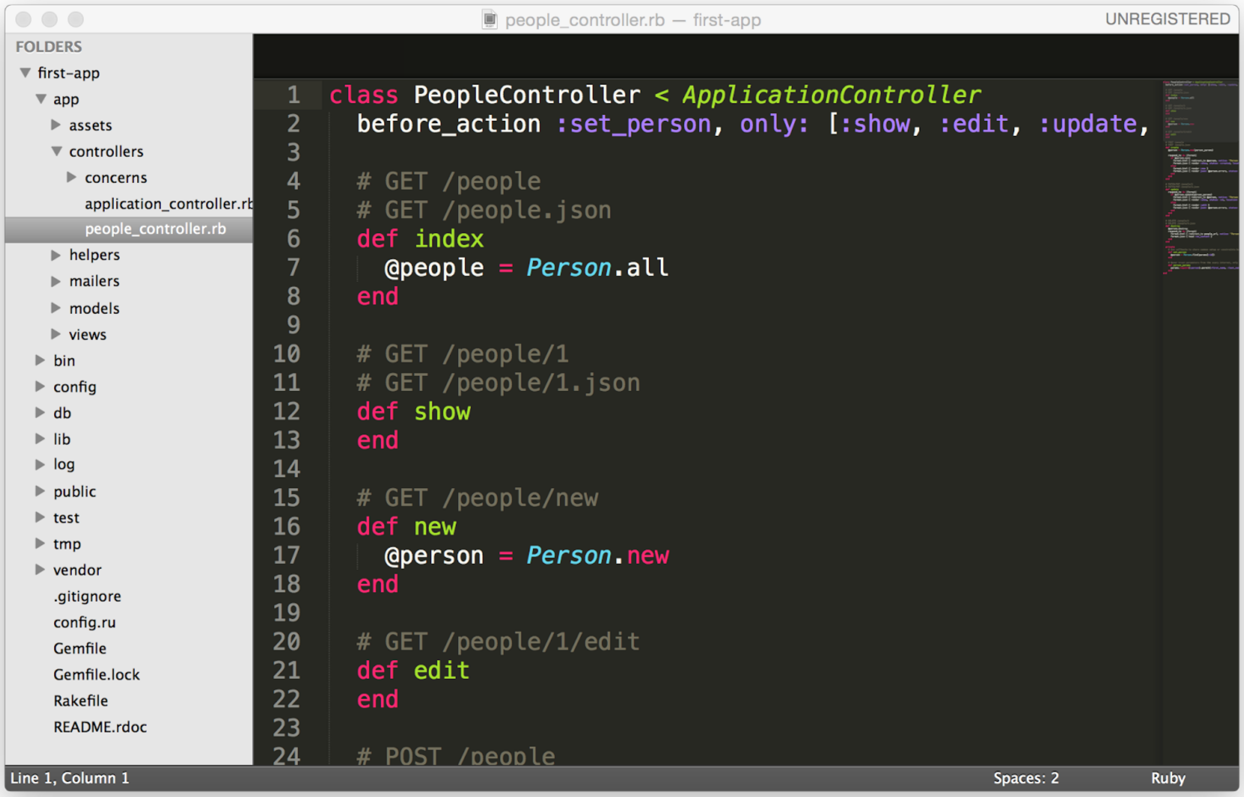


```ruby
get '/dictionary' do
    @dictionary = Word.all

    erb :'/words/index'
end

get '/dictionary/new' do

    erb :'/words/new'
end

get '/dictionary/:word' do
    word = params[:word]
    @word = Word.find_by_text(word)

    erb :'/words/show'
end



post '/dictionary' do
    word = params[:word]
    Word.create(text: word)
    @message = "Yay"
```

As you know from having written the controller in Sinatra, this is where you specify what your Sinatra app does with get, post, put, and delete requests and what happens at all your various paths.

Another important thing to note here is that there are no “default” Sinatra controller actions. If you want Sinatra to do something you have to tell it do it! Nothing is taken for granted. (Remember, Sinatra is very *flexible*.)

On the other hand, Rails splits up the work into two areas: the controller and routes. When you generate a Rails scaffold, Rails creates a controller for you with a whole bunch of default controller methods like so:

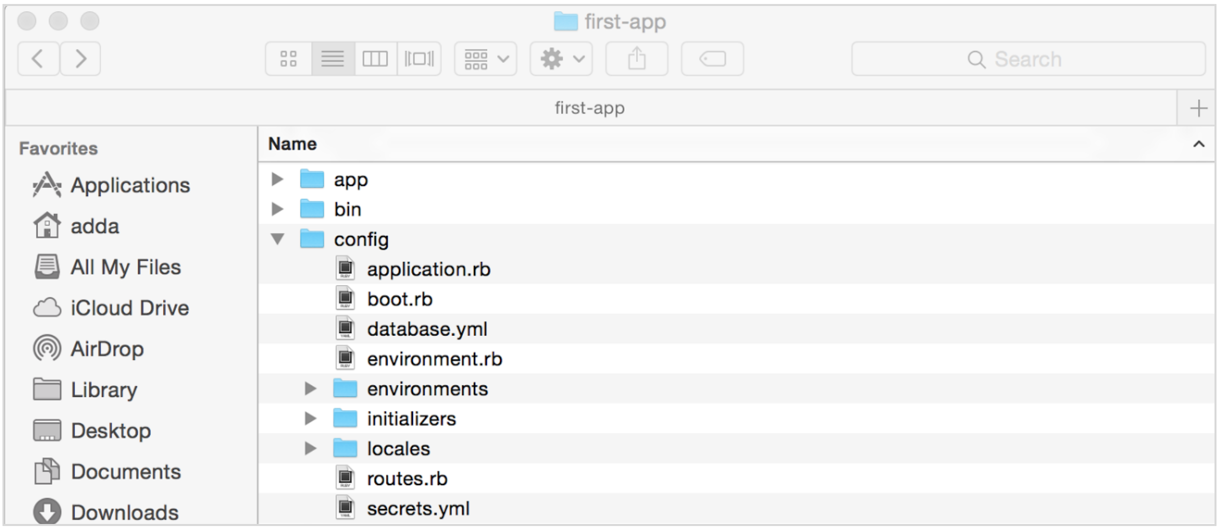


```ruby
class PeopleController < ApplicationController
  before_action :set_person, only: [:show, :edit, :update,

  # GET /people
  # GET /people.json
  def index
    @people = Person.all
  end

  # GET /people/1
  # GET /people/1.json
  def show
  end

  # GET /people/new
  def new
    @person = Person.new
  end

  # GET /people/1/edit
  def edit
  end

  # POST /people
```

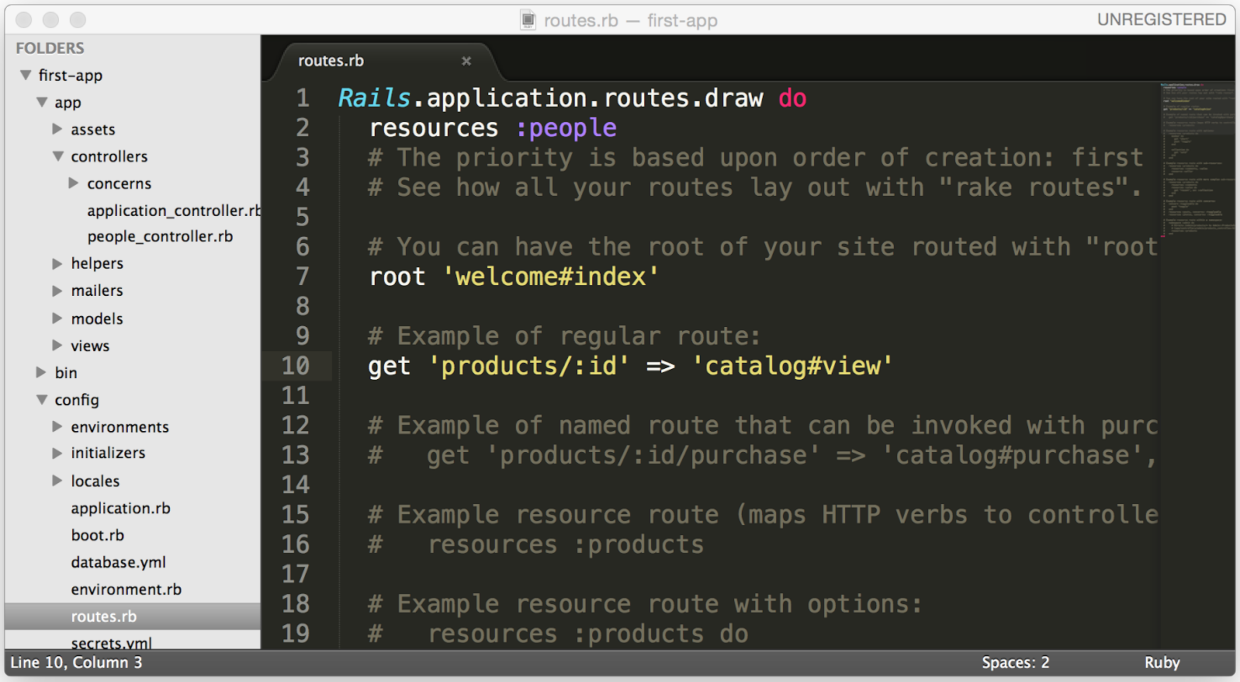
As you know, you didn't write any of these methods, Rails wrote them for you! And like all things Rails, they provide you a set of default settings that are in adherence with conventions and best practices.

But of course, it's perfectly normal (and reasonable) for you to want to edit some of your web app's paths. In order to do that, you need to edit your routes.rb file.

The routes.rb file is found inside your config folder:



Inside this file you will find some sample routes:



```ruby
Rails.application.routes.draw do
  resources :people
  # The priority is based upon order of creation: first
  # See how all your routes lay out with "rake routes".

  # You can have the root of your site routed with "root
  root 'welcome#index'

  # Example of regular route:
  get 'products/:id' => 'catalog#view'

  # Example of named route that can be invoked with purc
  #   get 'products/:id/purchase' => 'catalog#purchase',

  # Example resource route (maps HTTP verbs to controlle
  #   resources :products

  # Example resource route with options:
  #   resources :products do
```

In Rails routes work a lot like your controller actions did in Sinatra, but instead of grabbing data and business logic, you just point the new URL path to whatever controller method you want to use.