**CREATE A NEW RAILS PROJECT**

3.1 Installing rails

1. Verify you have a current version of Ruby installed:

$ ruby –v

1. Verify that SQLite3 is correctly installed and in your PATH:

$ sqlite3 --version

The program should report its version.

1. To install Rails, use gem install, the command provided by RubyGems:

$ gem install rails

1. To verify you have everything installed correctly, you should be able to run the following:

$ rails --version

3.2 Creating the Blog Application

1. Use the new application generator to create everything necessary to start working on a new task:

$ rails new blog

🡪 This will create a Rails application called Blog in a blog directory, and install the gem dependencies that are already mentioned in Gemfile using bundle install.

To see all command line options that the Rails application builder accepts, run: rails new –h

1. Switch to the blog application folder:

$ cd blog

**4) Hello, Rails!**

* 1. Starting up the Web Server

1. To see the Rails app, start a web server on your dev machine. Run the following in the blog directory:

$ bin/rails server

🡪 This will fire up Puma, a web server distributed with Rails by default. To see the app in action, go to <http://localhost:3000> in the browser

4.2 Get Rails to say “Hello”

1. To do this, you need to create a *controller* and a *view*
   1. The *controller* receives specific requests for the app – this is where information is collected (then, is displayed by the view).
   2. *Routing* decides which controller receives which requests
   3. There is often more than one route to each controller, and different routes can be served by different *actions*.
   4. Each action’s purpose is to collect info and provide it to a view, which displays this info in human readable format.
      1. By default, view templates are written in a language called eRuby (embedded Ruby) which is processed by the request cycle in Rails before being sent to the user.
2. To create a new controller, run the “controller” generator and tell it you want a controller called “Welcome” with an action called “index”:

$ bin/rails generate controller Welcome index

🡪 Rails will create several files and a route for you.

create app/controllers/welcome\_controller.rb

route get ‘welcome/index’

invoke erb

create app/views/welcome

create app/views/welcome/index.html.erb

invoke test\_unit

create test/controllers/welcome\_controller\_test.rb

invoke helper

create app/helpers/welcome\_helper.rb

invoke test\_unit

invoke assets

invoke coffee

create app/assets/javascripts/welcome.coffee

invoke scss

create app/assets/stylesheets/welcome.scss

1. Open the app/views/welcome/index.html.erb file in your text editor. Delete all of the existing code in the file, and replace it with:

<h1>Hello, Rails!</h1>

4.3 Setting the App Home Page

Now that we’ve made the controller and view, we need to tell Rails when we want “Hello, Rails!” to show up; in our case, we want it to show up when we navigate to the root URL of the site, <http://localhost:3000> - right now, “Welcome Aboard” is occupying that spot (I think this is ‘Yay! You’re on Rails!’ 🡪 is that right?)

1. To tell Rails where the home page is located:

* Open the file config/routes.rb in the text editor (Sublime, in my case) – the config sub-folder is a direct sub-folder of ‘blog’
  + This is the application’s routing file, which holds entries in a special DSL (domain-specific language) that tells Rails how to connect incoming requests to controllers and actions.
* Edit the file by adding the line of code root ‘welcome#index’ after get ‘wecome/index’ so that it looks like:

Rails.application.routes.draw do

get 'welcome/index'

root ‘welcome#index’

# For details on the DSL available within this file, see http://guides.rubyonrails.org/routing.html

end

* root ‘welcome#index’ tells Rails to map requests to the root of the application, to the welcome controller’s index action
* get ‘welcome/index’ tells Rails to map requests to <http://localhost:3000/welcome/index> to the welcome controller’s index action (this was created when I ran the controller action earlier (bin/rails generate controller Welcome index)
* NOW, the web server should be running the “Hello, Rails!” message I put in to app/views/welcome/index.html.erb, indicating that this new route is going to WelcomeController’s index action and is rendering the view correctly.
* **Q: WHAT WAS THE ROOT PREVIOUSLY? WHERE WAS THE ORIGINAL ‘YAY RAILS ETC.’ MESSAGE COMING FROM?**

**5) Getting Up and Running**

Now that the controller, an action and a view have been created, I’m going to create a new **resource** in the Blog application.

* A **resource** is a term used for a collection of similar objects, such as articles, people or animals.
  + You can create, read, update and destroy items for a resource 🡪 these operations are referred to as *CRUD* operations
* Rails provides a *resources* method which can be used to declare a standard *REST resource*. You need to add the *article resource* to the config/routes.rb so the file will look as follows:

Rails.application.routes.draw do

get 'welcome/index'

resources :articles

root 'welcome#index'

# For details on the DSL available within this file, see http://guides.rubyonrails.org/routing.html

end

* When we run bin/rails routes, we’ll see that it has defined routes for all the standard RESTful actions. 🡪 Rails can infer the singular form article and make meaningful use of the distinction.

Jessicas-MacBook-Pro:blog jessicafisher$ bin/rails routes

       Prefix Verb   URI Pattern                  Controller#Action

welcome\_index GET    /welcome/index(.:format)     welcome#index

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

              POST   /articles(.:format)          articles#create

  new\_article GET    /articles/new(.:format)      articles#new

 edit\_article GET    /articles/:id/edit(.:format) articles#edit

      article GET    /articles/:id(.:format)      articles#show

              PATCH  /articles/:id(.:format)      articles#update

              PUT    /articles/:id(.:format)      articles#update

              DELETE /articles/:id(.:format)      articles#destroy

         root GET    /                            welcome#index

**5.1 Laying down the groundwork** – adding the ability to create new articles in your app and be able to view them – the “C” and “R” from CRUD: create and read.

1. Pick a place within the app to create a new article: /articles/new would be great; with the route already defined, requests can now be made to /articles/new in the app.
2. If we navigate to <http://localhost:3000/articles/new> we will see a routing error.
   1. Error occurs because the route needs to have a controller defined to serve the request. The error reads:  **uninitialized constant ArticlesController**
   2. So we need to a create a controller called ArticlesController. Do this by running the command:

$ bin/rails generate controller Articles

Now, within a new controller, which can be found in app/controllers/articles\_controller.rb, will be generated – it’s fairly empty.

class ArticlesController < ApplicationController

end

🡪 A controller is just a class that’s defined to inherit from ApplicationController.

🡪 Inside this class we’ll define methods that will become the actions for this controller; these actions will perform CRUD operations on the articles within our system.

There are public, private and protected methods in Ruby, but only public methods can be actions for controllers.

1. Now, at localhost:3000/articles/new, we’ll get a new error:

Unknown action

The action 'new' could not be found for ArticlesController

🡪 indicates that Rails cannot find the new action inside the ArticlesController that was generated 🡪 since when controllers are generated in Rails they are empty by default, unless we tell them the desired actions during the generation process

1. To manually define an action inside a ctonroller, you need to define a new method inside the controller.

🡪 Open app/controllers/articles\_controller.rb and inside the ArticlesController class, define the new method so that the controller now looks like:

class ArticlesController < ApplicationController

def new

end

end

a. A new error will pop up:

**ActionController::UnknownFormat in ArticlesController#new**

**ArticlesController#new is missing a template for this request format and variant. request.formats: ["text/html"] request.variant: [] NOTE! For XHR/Ajax or API requests, this action would normally respond with 204 No Content: an empty white screen. Since you're loading it in a web browser, we assume that you expected to actually render a template, not… nothing, so we're showing an error to be extra-clear. If you expect 204 No Content, carry on. That's what you'll get from an XHR or API request. Give it a shot.**

This error message pops up because Rails expects plain actions like this one to have views associated with them to display their information. With no view available, Rails will raise an exception.

Let’s decode the error message.

🡪 Part 1: indicates which template is missing: it’s the articles/new template.

* + Rails will first look for this template
  + If it’s not found, it will attempt to load a template called application/new
  + It looks for a template because ArticlesController inherits from ApplicationController

🡪 Part 2: contains request.formats - which specifies the format of template to be served in response.

* + It’s set to text/html as we requested this page via browser, so Rails is looking for an HTML template
  + Request.variant specifies what kind of physical devices would be served by the response and helps Rails determine which template to use in response; it’s empty here because no info has been provided

🡪 SO. The simplest template that would work in this case would be one located at app/views/articles/new.html.erb

* + The extension of the file name: the first extension is the *format* of the template, and the second extension is the *handler* that will be used to render the template
  + Rails is attempting to find a template called articles/new within app/views for the app. The format for this template can only be html and the default handler for HTML is erb.
    - \*\* Rails uses other handlers for other formats.
      * Builder handler is used to build XML templates
      * Coffee handler uses CoffeeScript to build JavaScript templates
    - Since I want to create a new HTML form, I’m going to use the ERB language which is designed to embed Ruby in HTML.