Ruby on Rails kickoff - Text directions and references

Rails overview:

<https://guides.rubyonrails.org/>

<https://rubyonrails.org/>

<https://weblog.rubyonrails.org/>

Creator of Rails: David Heinemeier Hansson

<https://dhh.dk/>

Ruby meetups:

<https://www.meetup.com/topics/ruby/>

Online code repository hosting services:

<https://github.com/>

<https://bitbucket.org/>

Devise gem GitHub repository:

<https://github.com/heartcombo/devise>

Gems for Ruby and Ruby on Rails:

<https://rubygems.org/>

Model, View, Controller and Rails App Structure - Text references

So Rails follows a MVC pattern of web applications and mvc creates a separation of presentation layer (what the user of the application sees in the browser/mobile device) and the business-logic or backend (invisible layer).

MVC - Model, View, Controller

MODELS :- models are the resources which are in our web application like :- users , post , stock , article, etc we know we have a database or tables in database , like for users we have users data base , hence we have models associated with these databases such that users model have users table and these models helps to communicate with databases.

VIEWS:- view makes the front-end of our application that will be seen by our users like homepage , port-folio page , etc and we will use ruby codes in the teplates so ruby has embedded code for these templates

Hence , instead of .html file , we have to save file with .html.erb

CONTROLLERS :- controllers are invisible to the users bcz they are in backend , like for users we have users\_controller to control the user activity in our application and so on.

General flow of Rails application:

-> Request made at browser

-> Request received at router of rails application

-> Request routed to appropriate action in a controller

-> Controller#action either renders a view template or communicates with model

-> Model communicates with database

-> Model sends back information to controller

-> Controller renders view

Root route, controller and more MVC - Text directions and references

To set a root route, navigate to config/routes.rb file and enter in the following code ->

root 'pages#home'

In order to preview the application, you will need to start the rails server. You can start the server by navigating to the terminal and typing in the following command from your application directory followed by enter/return:

rails server

You can also type rails s for short. Once the server starts you can open up a browser window and type in localhost:3000 to preview the app. If you are using AWS cloud9, please use the videos in section 14 where previewing the applications are covered (after creating new rails apps using rails 5 and 6) since the preview link differs (it won't be localhost:3000).

Back to the app, in order for the code to work there will need to be a pages controller and a home action within it.

Create a pages controller by navigating to the terminal/command line and typing in the following command from your application directory:

rails generate controller pages

There should now be a pages\_controller.rb file under app/controllers folder.

Open up this file and add the home action to it with a rendering of 'Hello World!' like below:

1. def home
2. render html: 'Hello World!'
3. end

Once you save this file and reload the browser preview, Hello World! should be displayed in the browser.

In order to do this the conventional way using views remove the code within the home action and leave it as an empty method/action like below:

1. def home
2. end

This will expect a home.html.erb template under the pages folder under views. This pages folder under the views folder was created when the pages controller was generated.

Within this app/views/pages folder, create a new file called home.html.erb and add in the following code inside the file:

Hello World!

Once you save this file and preview it from the browser, it should still say Hello World! as expected.

Congratulations, you have made your Rails application say 'Hello World!'

STRUCTURE OF RAILS

| **File/Folder** | **Purpose** |
| --- | --- |
| app/ | Contains the controllers, models, views, helpers, mailers, channels, jobs, and assets for your application. You'll focus on this folder for the remainder of this guide. |
| bin/ | Contains the rails script that starts your app and can contain other scripts you use to set up, update, deploy, or run your application. |
| config/ | Contains configuration for your application's routes, database, and more. This is covered in more detail in [Configuring Rails Applications](https://guides.rubyonrails.org/configuring.html). |
| config.ru | Rack configuration for Rack-based servers used to start the application. For more information about Rack, see the [Rack website](https://rack.github.io/). |
| db/ | Contains your current database schema, as well as the database migrations. |
| Gemfile Gemfile.lock | These files allow you to specify what gem dependencies are needed for your Rails application. These files are used by the Bundler gem. For more information about Bundler, see the [Bundler website](https://bundler.io/). |
| lib/ | Extended modules for your application. |
| log/ | Application log files. |
| package.json | This file allows you to specify what npm dependencies are needed for your Rails application. This file is used by Yarn. For more information about Yarn, see the [Yarn website](https://yarnpkg.com/lang/en/). |
| public/ | Contains static files and compiled assets. When your app is running, this directory will be exposed as-is. |
| Rakefile | This file locates and loads tasks that can be run from the command line. The task definitions are defined throughout the components of Rails. Rather than changing Rakefile, you should add your own tasks by adding files to the lib/tasks directory of your application. |
| README.md | This is a brief instruction manual for your application. You should edit this file to tell others what your application does, how to set it up, and so on. |
| storage/ | Active Storage files for Disk Service. This is covered in [Active Storage Overview](https://guides.rubyonrails.org/active_storage_overview.html). |
| test/ | Unit tests, fixtures, and other test apparatus. These are covered in [Testing Rails Applications](https://guides.rubyonrails.org/testing.html). |
| tmp/ | Temporary files (like cache and pid files). |
| vendor/ | A place for all third-party code. In a typical Rails application this includes vendored gems. |
| .gitignore | This file tells git which files (or patterns) it should ignore. See [GitHub - Ignoring files](https://help.github.com/articles/ignoring-files) for more info about ignoring files. |
| .ruby-version | This file contains the default Ruby version. |

GENERATE A VIEW

1. define a route that points to a controller#action
2. have an approximately named controller , for example : if dealing with layouts or static pages of the application, a name could be pages\_controller
3. have an approximately named action, for example : if dealing with a homepage, the action/method could be named home
4. if done this way, under views, rails will expect a pages folder (named for the pages controller) and a [home.html.erb](http://home.html.erb) template (named for the home action)

Add About page and homework assignment - Text reference and code

To add an About page to the application, first add the following route to the config/routes.rb folder:

get 'about', to: 'pages#about'

You can then add the about action to the pages controller like below:

1. def about
2. end

Now you will need a view template, so add an about.html.erb file under the app/views/pages folder and fill in some text to display within it:

<h1>This is the About page</h1>

Once you save the file, you'll be able to preview this page by appending /about to your homepage. In your local machine it would be localhost:3000/about

Homework:

1. Create an alpha-blog application.

2. Setup tracking with Git and make a commit of your code with the message "Initial commit". Set up a GitHub repository for the app.

3. Create a home page view (as the root route) and an about page like the test app.

4. Make sure you are able to start the server and preview the application. Take snapshots of the two pages (home and about) and post them to the Q & A.

5. Make a commit of your code and push to your GitHub repository for the application.

Production Deploy - Text directions, references and code

You can sign-up for a heroku account at: <https://www.heroku.com/>

Preparation for production deployment:

- Remove sqlite3 gem from top of your Gemfile to within group :development, :test do block

Mine looks like below:

1. group :development, :test do
2. gem 'sqlite3', '~> 1.4'
3. gem 'byebug', platforms: [:mri, :mingw, :x64\_mingw]
4. end

- Create a group production ->

1. group :production do
2. gem 'pg'
3. end

- Save Gemfile

- Run bundle install --without production to update Gemfile.lock file

- Commit your changes to git repo ->

git add -A

git commit -m "Make app production ready"

You can check if you already have Heroku CLI installed by going to your terminal and typing in:

heroku --version

or simply heroku

You can get the Heroku CLI from here: <https://devcenter.heroku.com/articles/heroku-cli>

Check Heroku installation:

heroku --version

heroku # for list of common heroku commands

Once installed, login to your Heroku account from your application directory:

heroku login

To create a new production version of your app hosted in Heroku, use the following command:

heroku create

To push your application code to Heroku and deploy your app, use the command below, but **make sure all your code changes are committed by checking git status first** (following the steps in the beginning of this text lecture) ->

git push heroku master

To change the name of your application ->

heroku rename newnameofyourapp

replace newnameofyourapp above with the name you'd like to give your app

Your app will then be accessible from the following browser URL ->

https://newnameofyourapp.herokuapp.com

CRUD and scaffold generators - Text directions, references and code

Query language to communicate with database: SQL (Structured Query Language)

CRUD actions:

C - Create

R - Read

U - Update

D - Delete

Scaffold generator command to create an article model (with two attributes), articles controller, views for articles and migration file to create articles table:

rails generate scaffold Article title:string description:text

Command to see routes presented in a viewer-friendly way:

rails routes --expanded

The line resources :articles in the config/routes.rb file provides the following routes:

- index of articles (GET request)

- new article (GET)

- create article (POST)

- edit article (GET)

- update article (PUT and PATCH)

- show article (GET)

- delete article (DELETE)

From UI perspective ->

- index lists all the articles in the articles table in the database of the app

- new article deals with the form to enter in new article details

- create handles the submission of the items in the new article form

- edit article deals with the form to enter edited information for an existing article

- update article deals with the submission of the edit article form

- show article displays an individual article based on selection

- delete article deletes an article from the articles table

In preparation for the next section, learn and practice SQL here: <https://www.w3schools.com/sql/>