Assignment for Module #3: Recipe Finder

The overall goal of this assignment is to implement a Rails application using model, view, and controller classes.

- the model class will access information
- the view class will display information and accept commands from the user
- the controller class will implement actions through internal service logic and the delegation to model and view classes.

The functional goal is to provide web page access to recipe information served by http://food2fork.com/api through JSON and images. Documentation for the API can be found at http://food2fork.com/about/api.

Functional Requirements

You are tasked with creating a Rails app that will display an index page with the recipes returned by a search.

- the user will supply a keyword to search for recipes
- the Rails app will pass that query to http://food2fork.com/api and accept the results
- the Rails app will build a web page that will display the results and accept the next keyword search
- the web page displayed will provide HTML links to other websites, where user will be able to find more details about the recipe.

You should already have the `Recipe` class from an earlier assignment. (Remem- ber that, unlike in that assignment, you will not need to require HTTParty gem in your code, because loading HTTParty gem should be the Bundler's job.)

You are also tasked with deploying your solution to Heroku — to be accessed by friends, family, other students, co-workers, and prospective employers.

Getting Started

1. Create a new Rails application called **recipefinder**. Use the `rails` command to do it.

2. Download and extract the starter set of boostrap files.

- replace the generated `Gemfile` with the `Gemfile` from the bootstrap fileset
- run the `bundle` command to resolve new gems

```
|-- Gemfile
|-- README.md
|-- .rspec (important hidden file)
`-- spec
    |-- recipes_app_spec.rb
    `-- spec_helper.rb
```

3. Install the following gems used by the RSpec unit tests. You may have some of these already installed. The last gem is used for headless web page testing.

```
$ gem install rspec
$ gem install rspec-its
$ gem install capybara
$ gem install poltergeist
```

4. Make sure that PhantomJS is installed on your computer and that you have the path to the executable on your PATH environment variable. You can check it by running the command `phantomjs -v`. This binary is used by the poltergeist gem to implement a headless unit test. You can interact with your Rails app directly using a browser without this library. It is only needed by the RSpec tests. **PhantomJS** installation was covered in **Module 1**. In case you need more information, the download URLs are below. Linux users will need to use version 1.9.8 or build from source. All other platforms can easily use 2.0.0.

- phantomjs downloads: http://phantomjs.org/download.html

- bitbucket: https://bitbucket.org/ariya/phantomjs/downloads

5. Run the RSpec test(s) to receive feedback. They must be run from the root of your application. All tests will (obviously) fail until you complete the solution.

```
Finished in 1.69 seconds (files took 0.41211 seconds to load)
8 examples, 8 failures

Failed examples:
rspec ./spec/recipes_app_spec.rb:6 # Recipes App displays 'Kahlúa-Spiked' when request parameter 'search' is mocha
rspec ./spec/recipes_app_spec.rb:11 # Recipes App utilizes the FOOD2FORK_SERVER_AND_PORT environment variable
rspec ./spec/recipes_app_spec.rb:16 # Recipes App utilizes the FOOD2FORK_KEY environment variable
rspec ./spec/recipes_app_spec.rb:24 # Recipes App visit root displays chocolate (default)
rspec ./spec/recipes_app_spec.rb:28 # Recipes App visit root displays 'Powered By Food2Fork.com'
rspec ./spec/recipes_app_spec.rb:32 # Recipes App visit root displays table element that has a row with 3 columns
rspec ./spec/recipes_app_spec.rb:36 # Recipes App visit root column 1 should have the thumbnail inside img tag inside a link tag
rspec ./spec/recipes_app_spec.rb:40 # Recipes App visit root title should be inside a second column inside a link tagink tag
```

6. Implement your Rails app solution and use the RSpec tests to help verify your completed Rails app solution.

7. Submit your Rails app solution for grading.

8. (Optional) Post your Rails app solution to Heroku.

Technical Requirements

1. Create a new Rails app called **recipefinder**. Use the `Gemfile` provided in the boostrap files. Do not change the `Gemfile` from what is provided or your submitted solution may not be able to be processed by the grader (i.e., do not add any additional gems or change gem versions).

2. Generate `RecipesController` (`recipes_controller.rb`) that will have an index action

3. The `RecipesController` `index` action should

- check if a request parameter `search` was passed in.
- use the search term as the keyword if supplied, and use a default value of **chocolate** if not supplied

4. Create a model, `Recipe` (`recipe.rb`) that will contain a `for` class method.

5. The Recipe `for` class method should

- accept a `keyword` as argument
- use the `keyword` to query the Food2Fork API for a result
- add the HTTP query parameter key (your developer key) to each out-going URL request to http://food2fork.com/api using HTTParty `default_params`
- obtain the key value from an environment variable FOOD2FORK_KEY
- obtain the url (and/or port) value from an environment variable FOOD2FORK_SERVER_AND_PORT

You will use the http://food2fork.com/api `host` and `port#` (default=:80) during development and Heroku deployment. *That is why you do not need to assign FOOD2FORK_SERVER_AND_PORT on your system.* **However**, your assignment will be graded off-line and should get its `host` and `port#` from the FOOD2FORK_SERVER_AND_PORT environment variable. Your assignment must use the defined value if present and default to the real value otherwise.

```
class Recipe
...
key_value = ENV['FOOD2FORK_KEY']
hostport = ENV['FOOD2FORK_SERVER_AND_PORT'] || 'www.food2fork.com'
base_uri "http://#{hostport}/api"
...
```

6. **Foods2Fork** requires attribution when using their API. Place the following somewhere in your application layout file (`application.html.erb`) to be displayed alongside the recipes.

```
<p>Powered By Food2Fork.com</p>
```

7. The view should

- list each recipe as a row in an HTML table (`<table>`)
- each row (`<tr>`) should have 3 columns (`<td>`) where
    - column 1 should contain the **thumbnail** of the recipe
    - column 2 should contain the **title** and
    - column 3 should contain the **social rank** of the recipe.

You are not required to create an HTML form for the search term. You may specify the search keyword using just the URL with the following syntax in the browser.

```
http://localhost:3000/recipes/index?search=swiss
```

8. Add anchor tags to your image and title. You should be able to click on either the title or the thumbnail and go straight to the actual recipe (out there on the web). Look at image_tag Rails helper for help with defining an `img` tag and use this helper as the first argument to `link_to` helper.

9. Inside the image tag, specify `width` and `height` of 100 for your images.

10. Sanitize recipe titles displayed.

Rails automatically escapes HTML in strings to avoid XSS attacks. Because of this, some of your titles will look wrong.

For example, try searching for **mocha** and look at your titles.

To get around this issue, Rails has a `sanitize` (or `raw`) helper that will help you display HTML characters properly.

11. Make the `RecipesController` `index` action the default (**root**) page for your application.

Instead of having to go to http://localhost:3000/recipes/index to get to your recipes, you want this page to be the default (root). You should therefore be able to go to http://localhost:3000/?search=apple pie , for example, and see your results.

12. (optional – ungraded) Deploy your app to Heroku at recipefinder-X.herokuapp.com where X is any available number from 1 to 10000000.

In order to do this you will have to define the FOOD2FORK KEY with your key to use the Food2Fork api on Heroku. Instructions for doing that can be found at the following link:

https://devcenter.heroku.com/articles/config-vars#example

Self Grading/Feedback

Some unit tests have been provided in the bootstrap files. They are examples of the tests that the grader will be running against your submitted solution.

Remember that the `rspec` command must be run from the root of your application.

```
$ rspec
...

Recipes App

displays 'Kahlúa-Spiked' when request parameter 'search' is mocha
utilizes the FOOD2FORK_SERVER_AND_PORT environment variable
utilizes the FOOD2FORK_KEY environment variable
visit root
  displays chocolate (default)
  displays 'Powered By Food2Fork.com'
  displays table element that has a row with 3 columns
  column 1 should have the thumbnail inside img tag inside a link tag
  title should be inside a second column inside a link tag

Finished in 2.73 seconds (files took 0.54954 seconds to load)
8 examples, 0 failures
```

The tests assume your server is running on `localhost:3000`. If it is not the case with your development environment, please adjust the source code in `recipes_app_spec.rb`

```
ruby Capybara.app host = "http://localhost:3000"
```

Submission

Submit a `.zip` archive (other archive forms not currently supported) with your solution root directory as the top-level (e.g., your `Gemfile` and sibling files must be in the root of the archive and not in a sub-folder.

The grader will replace the `spec` files with fresh copies and will perform a test with different query terms.

``` |-- app | |-- assets | |-- controllers | |-- helpers | |-- mailers | |-- models | |-- views |-- bin |-- config |-- config.ru |-- db |-- Gemfile |-- Gemfile.lock |-- lib |-- log |-- public |-- Rakefile |--
```

README.rdoc |-- test `-- vendor

Last update: 2016-01-31