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 `www.recipepuppy.com` through JSON and images. Documentation for the API can be found at http://www.recipepuppy.com/about/api/.

Functional Requirements

You are tasked with creating a Rails app that will display a recipe index page based on a search keyword entered.

- the user will supply a keyword to search for
- the Rails app will pass that query to `www.recipepuppy.com` and accept the results
- the Rails app will build a web page display of the results and accept the next keyword search
- the web page displayed will provide HTML links to more detailed recipe information from other web sites.

You should already have the `Recipe` class from the an earlier assignment. (Remember, that unlike in that assignment – you will not need to `require` HTTParty gem in your code, since 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 using the `rails` command called `recipefinder`.
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 two gems are used for headless web page testing.

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

4. Make sure phantomJS is installed and in your bin PATH on your system ($ phantomjs –version). This binary is used by the `poltergeist` gem to implement a headless unit test for the Web interface. You can interact with your Rails app directly using a browser without this library. It is only needed by the rspec tests to provide you feedback for example criteria the grader will be looking for later when submitted. 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 can be run from their original location. If you copy/move them, be sure to include the important `.rspec` hidden file. All tests will (obviously) fail until you complete the specified solution.

```
Finished in 11.31 seconds (files took 0.3853 seconds to load)
5 examples, 5 failures

Failed examples:

rspec ./spec/recipes_app_spec.rb:6 # Recipes App displays Café Mocha when request parameter
rspec ./spec/recipes_app_spec.rb:14 # Recipes App visit root displays chocolate (default)
rspec ./spec/recipes_app_spec.rb:18 # Recipes App visit root displays table element that has
rspec ./spec/recipes_app_spec.rb:22 # Recipes App visit root column 1 should have the thumbn
rspec ./spec/recipes_app_spec.rb:26 # Recipes App visit root title should be inside a second
```

6. Implement your Rails app solution and use the rpsec tests to help verify your completed Rails app solution.
7. (optional) Post your Rails app solution to Heroku.
8. Submit your Rails app solution for grading.

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

- take a keyword to query
- query the Recipe Puppy API for a result.
- add the (undocumented) HTTP query parameter `onlyImages=1` to each outgoing URL request to `www.recipepuppy.com` using HTTParty `default_params`. This will produce results with images and will be consistent with what the off-line grader expects your application to be provided.

You will use the `www.recipepuppy.com` host and port# (default=:80) during development and Heroku deployment. However, your assignment will be graded off-line and should get its host and port# from the `RECIPEPUPPY_HOSTPORT` environment variable. Your assignment must use the defined value if present and default to the real value otherwise.

```
class Recipe
    ...
    hostport = ENV['RECIPEPUPPY_HOSTPORT'].nil? ? 'www.recipepuppy.com' : ENV['RECIPEPUPPY_H
    base_uri "http://#{hostport}/api"
    ...
```

6. Create your view that 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 ingredients.

You are not required to create a 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
```

7. Add `href` 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 (http://api.rubyonrails.org/classes/ActionView/Helpers/AssetTagHelper.html#method-i-image_tag) and use this helper as the first argument to `link_to` helper.
8. Inside the `image_tag` specify `width` and `height` of 100 for your images.
9. Sanitize recipe titles displayed. Rails automatically escapes HTML in your strings (to avoid XSS attacks http://en.wikipedia.org/wiki/Cross-site_scripting). 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 (http://api.rubyonrails.org/classes/ActionView/Helpers/SanitizeHelper.html#method-i-sanitize) that will help you display HTML characters properly
10. 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%20pie` for example and see your results.
11. (optional – ungraded) Deploy your app to Heroku at recipefinderX.herokuapp.com where `X` is any available number from 1 to 10000000

Self Grading/Feedback

Some unit tests have been provided in the bootstrap files and provide examples of some tests the grader will be evaluating for when you submit your solution. They can be run from any location but be sure to copy the hidden `.rspec` file if you move them.

```
$ rspec
...
Recipes App
  displays Café Mocha when request parameter 'search' is mocha
  visit root
    displays chocolate (default)
    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)
5 examples, 0 failures
```

The tests assume your server is running on localhost:3000. Please adjust the source code in `recipes_app_spec.rb` if that is not the case with your development environment.

```
Capybara.app_host = "http://localhost:3000"
```

Submission

Submit an .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
```