PenPal Gladiators Technical Documentation

Version 1.0.1

May 7, 2015

Contents

1 A Word About Licensing

This app is licensed under GPLv3. Anyone is free to modify this app so long as they make the source code available under GPLv3 or later. Modifying this app and not making the source code available under GPLv3 is a violation of the license for this apps code and will result in legal action.

2 Setting Up Your Development Environment

2.1 Git Setup

Create an account on GitHuband run

```
    git clone https://github.com/Fong-/CS169-PenPal-Gladiators.git
```

to clone the repository. See the Git documentation if you're not familiar with Git.

2.1.1 PenPal Gladiators Git Conventions

All features and changes are performed on a branch. This is called branch per feature development. When a feature has been reviewed and is passing all tests (e.g. via CI), rebase the branch on the iteration branch (if applicable) or master branch (if not) and merge into the iteration branch or master by running:

```
git merge --no-ff [feature-branchname]
```

2.2 Heroku

Download the Heroku command line tools and add the Heroku remote:

```
heroku git:remote -a [projectname]
```

inside the project directory. This adds the Heroku repository to your local copy. Run

```
git push Heroku master
```

to deploy the master branch to Heroku

2.3 CodeClimate

CodeClimate is linked to GitHub. Sign up for a CodeClimate account if you would like it to run on your own repository. CodeClimate flags potential design issues like duplication and excessive complexity before they turn into maintainability problems.

2.4 TravisCI

TravisCI is an automated testing suite for our repo. Links to it can be found on our GitHub repo. Register for a TravisCI account and link your GitHub repository. While CodeClimate checks for the style of the code, TravisCI runs the app's test suite for correctness. Always look at the TravisCI results before merging anything to master, and always write tests for new features as they're implemented.

3 Customizing Content and Options

3.1 Overview

In order to add your own topics, survey questions, and response weights, you will need to first erase the old content, modify `db/seeds.rb`, and then reseed. We have provided a tool for you to erase old content without erasing user data which you can run in the project directory with

```
rails runner tools/reset_seeds.rb
```

or

```
heroku run rails runner reset_seeds.rb
```

if on Heroku. You can also reseed by running

```
rake db:seed
```

or

```
heroku run rake db:seed
```

if on Heroku.

3.2 Changing Topics

1. Open `db/seeds.rb` in a text editor of your choice
2. Find the line that starts with `topics = [`
3. You should see an indented set of lines in the format

   ```
   { :name => "topic name", :icon => "topic icon path"},
   ```

   Each line represents a topic. Edit, add, or remove as necessary.

3.3 Changing Questions of a Topic

1. Open `db/seeds.rb` in a text editor of your choice

2. Create a new question and response set using the following template:

```
name_of_question = Topic.find_by_name("Topic")
.survey_questions.create(:text => "Survey Question text", :
   index => Question Number)
responses = [
    {
    :index => Response Number,
    :text => "Response Text",
    :summary_text => "Summary Text"
    },
    ].map { |data| name_of_question.survey_responses.create
       data }
```

   Each question and response must be assigned an index (the questions and responses are 0-indexed). For example, if this was the first question, the question index would be 0. If it was the second, it would be 1.

3. To add the weights between responses, use this template for each pair of responses:

```
ResponseWeight.create :response1_id => responses[Response
   Number].id, :response2_id => responses[Response Number].id,
    :weight => [weight]
```

4 Customizing and Extending the Project

4.1 Database Models

These reside in `app/models`:

- `User` – stores the information for each user.
- `Topic` – represents a survey topic.
- `SurveyQuestion` – belongs to a survey topic, represents a question in the topic
- `SurveyResponse` – belongs to a survey question, represents a possible answer to a question
- `UserSurveyResponse` – represents a users response to a question, Belongs to a SurveyResposne as well as a User
- `ResponseWeight` – determines the score or ranking between two answers to the same question, a higher score will increase the likelihood of two users with those answers being matched.

- `Post` – represents a message in a conversation
- `Arena` – represents a match between two users
- `Conversation` – represents a conversation on a topic between two users, belongs to an Arena
- `Invite` – represents a match request from one user to another

Note: Nearly all models have their own controllers in `app/controllers`. The controllers publicly expose model functions.

5 Project Structure

Our project uses LESS, HTML, CoffeeScript and Angular.

5.1 File Locations

- Model (`.rb`) files are stored in `app/models`
- Controllers (`.rb`) files are stored in `app/controllers`
- Base level HTML pages are stored in `public/`. `homePage.html` is the main application view, `landingPage.html` is the landing page displayed to a first-time visitor, and `startPage.html` is the login and registration view
- Partial HTML fragments are stored in `app/assets/html`
- Front end CoffeeScript files are stored in `app/javascripts`
- Front end LESS files are stored in `app/stylesheets`
- Image resources for the topics and landing page are stored in `app/images`
- Fonts are stored in `vendor/fonts`
- Bootstrap CSS is stored in `vendor/stylesheets`
- Jasmine JS test code framework is stored in `vendor/javascripts`

5.2 Writing and Running Tests

Cucumber tests are stored in `features/`. JavaScript tests are stored in `spec/javascripts`. RSpec model tests are stored in `spec/models`. RSpec controller tests are stored in `spec/controllers`.

Run JavaScript tests with:

```
rake spec:javascript
```

Run RSpec tests with:

```
rspec
```

Run Cucumber tests with:

```
cucumber
```