Assignment 4
CSCI 3308 F13

WORKING WITH CUCUMBER

In this homework you will create user stories to describe a feature of a SaaS app, use the Cucumber tool to turn those stories into executable acceptance tests, and run the tests against your SaaS app.

SETUP

Specifically, you will write Cucumber scenarios that test the happy paths of parts 1-3 of Assignment 3. The GitHub repo

http://github.com/williammortl/hw3_rottenpotatoes

contains a "canonical" solution to Assignment 3 against which you will write your scenarios. It also contains the necessary scaffolding for the first couple of scenarios. We suggest that you go to github, login, then proceed to the URL given above. Then, fork this repository into your account and then use git to pull the source from your new repository.

Note that you'll want to immediately run:

```
cd hw3_rottenpotatoes
bundle install
```

In order to pull in some new libraries specified in the Gemfile, and then you'll want to run:

```
bundle exec rake db:migrate
```

which will get your database set up and will ensure that the libraries (gems) installed by bundle will be used, rather than anything else that has previously been installed. You may also find it helpful to run the following test database preparation command before running any tests:

```
bundle exec rake db:test:prepare
```

In general you will want to prefix all commands with 'bundle exec' in order to ensure you are using the correct libraries (gems). See http://bundler.io/ for more information on the bundle command. We would also recommend that you do a

```
git commit
```

as you get each part working. As an optional additional help, git allows you to associate tags -- symbolic names -- with particular commits. For example, immediately after doing a commit, you could say

```
git tag hw4-part1b
```

and thereafter you could use

```
git diff hw4-part1b
```

to see differences since that commit, rather than remembering its commit ID. Note that after creating a tag in your local repo, you need to say

```
git push origin –tags
```

to push the tags to a remote. (Tags are ignored by deployment remotes such as Heroku, so there's no point in pushing tags there.)

You may also find the following help video useful:

https://www.youtube.com/watch?feature=player_embedded&list=PLjbL0BCR04Q1XvAs8gN1Ie30jjnRbqCOZ&v=GpWkS8bfaKg

CREATE A DECLARATIVE SCENARIO STEP FOR ADDING MOVIES

If you have access to the SaaS textbook and have never used Cucumber before we strongly recommend that you work step by step through the code in Chapter 7 of the Engineering Software as a Service textbook. We also have a series of free screencasts that walk through the code in that chapter, which you can view even if you don't have the textbook.

As explained in Section 7.5 of the textbook, the goal of BDD is to express behavioral tasks rather than low-level operations.

The background step of all the scenarios in this homework requires that the movies database contain some movies. Analogous to the explanation in Section 7.5, it would go against the goal of BDD to do this by writing scenarios that spell out every interaction required to add a new movie, since adding new movies is not what these scenarios are

about.

Recall that the Given steps of a user story specify the initial state of the system; it doesn't matter how the system got into that state. For Part 1, therefore, you will create a step definition that will match the step, "Given the following movies exist" in the Background section of both `sort_movie_list.feature` and `filter_movie_list.feature`. (Later in the course, we will show how to DRY out the repeated Background sections in the two feature files.)

Add your code in the `movie_steps.rb` step definition file. You can just use ActiveRecord calls to directly add movies to the database; it's OK to bypass the GUI associated with creating new movies, since that's not what these scenarios are testing.

Success is when all Background steps for the scenarios in `filter_movie_list.feature` and `sort_movie_list.feature` are passing green.

HAPPY PATHS FOR FILTERING MOVIES

Complete the scenario restrict to movies with 'PG' or 'R' ratings in `filter_movie_list.feature`. You can use existing step definitions in `web_steps.rb` to check and uncheck the appropriate boxes, submit the form, and check whether the correct movies appear (you may ignore the case where the movie has no ratings).

Since it's tedious to repeat steps such as

```
When I check the 'PG' checkbox, And I check
the 'R' checkbox, etc.
```

create a step definition to match a step such as:

```
Given I check the following ratings: G, PG, R
```

This single step definition should only check the specified boxes, and leave the other boxes as they were. *Hint*: This step definition can reuse existing steps in `web_steps.rb`, as shown in the example in Section 7.5 in ESaaS.
For the scenario `all ratings selected`, it would be tedious to use "And I should see" to name every single movie. That would detract from the goal of BDD to convey the behavioral intent of the user story. To fix this, create step definitions in `movie_steps.rb` that will match steps of the form
`Then I should see all of the movies`

Hint: Consider counting the number of rows in the table to implement these steps. If you have computed rows as the number of table rows, you can use the assertion

```
rows.should == value
```

to fail the test in case the values don't match.

Use your new step definitions to complete the scenario `all ratings selected`. *Success* is when all scenarios in `filter_movie_list.feature` pass with all steps green.

HAPPY PATHS FOR SORTING MOVIES BY TITLE AND BY RELEASE DATE

Since the scenarios in `sort_movie_list.feature` involve sorting, you will need the ability to have steps that test whether one movie appears before another in the output listing. Create a step definition that matches a step such as "Then I should see 'Aladdin' before 'Amelie'."

Hints

- `page` is the Capybara method that returns whatever came back from the app server. `page.body` is the page's HTML body as one giant string.
- A regular expression could capture whether one string appears before another in a larger string, though that's not the only possible strategy.

Use the step definition you just created to complete the scenarios: sort movies alphabetically and sort movies in increasing order of release date in `sort_movie_list.feature`.

Success is all steps of all scenarios in both feature files passing green.

SUBMISSION

To submit your assignment, please submit a .tar.gz compressed archive file containing just your features directory. The command for doing this in a UNIX environment is:

```
tar czf features.tar.gz features/
```

Make sure that the features directory is contained in the archive. For example, unarchiving your submission should create a directory named `features/` in the current working directory. It should not extract all your features directly into the current working directory.

Upload this archive to moodle.