Homework 3: BDD & 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.

Specifically, you will write Cucumber scenarios that test the happy paths of parts 1-3 of HW 2.

We've prepared the following repo, containing a "canonical" solution to HW2 against which to write your scenarios, and the necessary scaffolding for the first couple of scenarios. The repo is **`https://github.com/Genaina/HW3.git`** on GitHub.

Prior to starting this assignment, watch the two screencasts posted for this chapter:
youtu.be/-wgZXDmhRw4 (this is the demo I did in class)
youtu.be/wornoChkjfM

Cloning the Repository:

`git clone git://github.com/Genaina/HW3.git` (Get code onto your machine)
From the parent directory (not inside hw3_rottenpotatoes):

```
git init
git remote add origin
      https://github.com/yourGitHubpath.git
git add hw3_rottenpotatoes/
git commit -m "my first commit"
```

`git push origin master` (Push the code to your github private account)

This will set up your local git folder in the parent folder of hw3_rottenpotatoes, initialize the folder, tell git to start tracking the homework3 folder, and then commit (with commit message "my first commit" and push the code to that folder. When you go to your github page, you should see a folder named hw3_rottenpotatoes with all your code inside.
By creating the folder, you will be able to keep all your homeworks in the same private repository while keeping the code separate. In future assignments, you'll simply clone or fork the code into the parent directory and run `git add hw4_rottenpotatoes` and `git commit/push` to add it for tracking.

Remember to do `git commit` and `git push` when you finish each part successfully!

Note: if you get an error that origin already exists, make sure you are in the parent folder of hw3_rottenpotatoes (not inside hw3_rottenpotatoes!), then run git remote rm origin.

Getting Started:

After cloning,

```
cd hw3_rottenpotatoes
bundle install --without production
rake db:migrate
```

~~`rake db:seed`~~ ← Don't do this! The feature creates its own table of movies, so this should not be there. If you've already done this, see below to clean the DB.

`rake db:test:prepare` (This copies the database that you were using during development to the test environment- remember that these databases are kept separate!)

Cleaning the DB:

```
rake db:reset
rake db:migrate
rake db:test:prepare
```

Looking Around:

As you start, look at the new `features` folder. Your code for this assignment will go in there. First, look at the sort movie list feature and the filter movie list feature. Try running cucumber.

`cucumber` (to run all features)

or

`cucumber features/sort_movie_list.feature` (to run just 1).

Some step definitions are implemented for you:

`features/step_definitions/web_steps.rb`

Take a look at steps you can use from this file.

See the end of this document for errors that have been observed and how to get around them.

Part 1: Create a declarative scenario step for adding movies

As explained in Section ~~4.7~~ 5.7 of *Engineering Long-Lasting Software...*, 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 ~~4.7~~ 5.7, 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.

Part 2: Happy paths for filtering movies

a) 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 (and just as importantly, movies with unselected ratings do not appear).

b) 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:
When 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 4.7 in ESAAS.

c) 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 that will match steps of the form:
Then I should see all of the movies
in `movie_steps.rb`.
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.

Update: You are not required to implement the scenario for “no ratings selected.”

d) 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.

Part 3: Happy paths for sorting movies by title and by release date

a) 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.

b) Use the step definition you create in part (a) 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:

Make sure you have committed your final changes to git prior to submission. If you go to your GitHub repository your most recent commit should be visable.

To officially submit your assignment, go through aprender.unb.br. Attach features.tar.gz

Commonly seen errors (and how to get around them):

When running cucumber:

```
You have already activated activesupport 3.2.7, but your Gemfile
requires activesupport 3.1.0. Using bundle exec may solve this.
(Gem::LoadError)
```

Solution: Instead run

```
bundle exec cucumber
```

When running anything with rake:

```
undefined method `prerequisites' for nil:NilClass
```

Solution:

Edit Gemfile, line 15

```
gem 'rspec-rails', '2.6.1'
```

Then rerun `bundle install --without production`

In bundle install's output, check that rspec-rails is version 2.6.1

If you get different errors during this homework, please post them to piazza (preferably with your solution, if you have one), and I will add them to this list. Most errors occur because of updates or modifications that have been made to your system.

When pushing to git:

```
error: The requested URL returned error: 403 while accessing
https://github.com/saasbook/hw3_rottenpotatoes/info/refs
fatal: HTTP request failed
```

Solution:

Push from the parent directory where you ran your initial git commands. There's still some git garbage in the hw3_rottenpotatoes folder, so if you try to push from inside the folder, it will try to push back to saasbook (which you don't have push access to) instead of pushing to the repository that you set up.