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 3-5 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 **`saasbook/hw3_rottenpotatoes`** on GitHub. We suggest you first fork that repo on GitHub (by visiting the github repository page at https://github.com/saasbook/hw3_rottenpotatoes and clicking the "Fork" button at the top), then clone from your own fork:
`git clone git@github.com:`*YourGitHubAccount***`/hw3_rottenpotatoes.git`**

Please make sure you make your forked repository **private** before continuing. You can set this with the Admin link at the top of your fork's page. We 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 hw3-part1b`** , and thereafter you could use **`git diff hw3-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.)

Part 1: Create a declarative scenario step for adding movies

As explained in Section 4.7 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, 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:
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 4.7 in ELLS.

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.

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.

To submit, follow the instructions in the homework (by clicking “View Instructions” in the Homework 3 box).