Homework 3 Saas 1 Ruby on Rails – Behavior-driven Design

In this homework, we created user stories to describe a feature of Saas app using the Cucumber tool to turn user stories into executable tests, and than ran those tests ran against Saas app. Specifically we wrote Cucumber scenarios that tested the happy paths of parts 1-3 of hw2 (described separately).

We first created a declarative scenario step for adding movies. The goal of Behavior-Driven Design (BDD) is to express behavioral tasks rather than low-level operations. The background step of all scenarios in this hw3 requires that the movies database contain movies. Note that it is NOT the goal of BDD to do this by writing scenarios that spell out every interaction required to add a movie, since adding new movies is not scenario test goal. The given steps of a user story specify the initial state of the system and it does not matter how the system got into that state.

Part 1: Creating a declarative scenario step for adding movies

For Part1, we create a step definition that matches the step "Given the following movies exist" in the Background section of both sort\_movie\_list.feature and filter\_movie\_list.feature. Note that later we will learn to DRY (Don't Repeat Yourself) out our code, such as the repeated section in the Background sections in the two feature files.

We then added our code to the movie\_steps.rb step definition file. We used Active Record to add movies directly to the database. The professors noted that bypassing the GUI associated with creating new movies is fine, since that is not what the scenario is testing. Success was when all Background steps for the scenarios in filter\_movie\_list.feature and movie\_list.feature were passing green.

Part 2: Happy Paths for Filtering Movies

In Part 2, we completed the scenario to restrict movies with 'PG' and 'R' ratings in the filter\_movie\_list.feature. We were allowed to use existing steps in the web\_steps.rb to check and uncheck the appropriate boxes, submit the form, and check whether the correct movies appear, ignoring the case where the movie has no ratings.

Some Hints Were:

--It's tedious to repeat steps such as:

When I check the 'PG' checkbox, And I check

the 'R' checkbox, etc.

--Instead 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. Another Hint was: This step definition can reuse existing steps in web\_steps.rb, as shown in the example in the text book for this class, the Ruby on Rails Saas book written by Professors Armando Fox and Dave Patterson.

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, we created step definitions in movie\_steps.rb that will match steps of the form:

--Then I should see all of the movies

A hint on this part was: Consider counting the number of rows in the table to implement these steps. If we had 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.

We used our new step definitions to complete the scenario all\_ratings\_selected. Success was 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

Since the scenarios in sort\_movie\_list.feature involve sorting, we needed the ability to have steps that test whether one movie appears before another in the output listing. We had to create a step definition that matched a step such as “Then I should see 'Aladdin' before 'Amelie'.”

Some hints were:

“page” is the Capybara method that returns what ever 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.

We used the step definition we created to complete the scenarios: we sorted the movies alphabetically and sorted the movies in increasing order of release date in sort\_movie\_list.feature.

Success was when all steps in all scenarios in both feature files were passing green.