**CS 1699 - DELIVERABLE 3: Web Testing with BDD**

**for imgur.com**

Mike Byrne & Jordan McAleer

mjb187@pitt.edu

jmm291@pitt.edu

https://github.com/mjb187/CS1699-Deliverable3

CS 1699: Software Testing

Bill Laboon

March 3rd, 2015

**DECISION TO TEST IMGUR.COM**

We decided to test the general functionality of the popular image-hosting and image-sharing site, imgur.com (pronounced “image-er” according to the site itself). Specifically, we decided to test the site’s ability to prompt unregistered users to either login or signup, to accept keyboard shortcuts, to allow users a suite of commenting options, [x], [x], and [x]. We had initially wanted to also test functionality on the mobile version of the site, but since we could not force m.imgur.com to load in the desktop browser, we elected to omit testing this functionality.

We decided on imgur as the basis for testing for a few reasons. First off, it has a user account system that allows for increased site interactivity when utilized (and thus giving us a good basis to use when formulating test cases). Secondly, imgur is large site, making it easy to section off our tests and not cause a disruption by uploading test images and test comments. Additionally, its functionality is similar to that of a messageboard; if we were testing Amazon (a site where the main functionality is to make purchases and leave useful product feedback), it would feel odd to test everything but the ability to purchase things. It would also make me very uneasy to trust something like Selenium to actually follow through with purchases, lest it accidentally purchase ten things that I do not want on accident. For this reason, we deemed imgur a “safe” choice for testing. Finally, imgur is a widely used and popular site, so we thought it would be informative to put ourselves in the mindset of the developers of such a high-traffic site and come up with tests as if we were first designing the site.

**DESCRIPTION OF ISSUES**

As with before, the first issue we faced was simply deciding what to test. With such a big site like imgur, there are many features that come into play, as well as the two complementary states of “logged in” and “not logged in” changing the features that are available to the user. The features we eventually decided on were chosen due to their relatively simple ability to test (such as searching for a login prompt on the page or checking that a keypress triggered the appropriate event) and their level of discretion (such as avoiding flooding the image stream with xkcd comics and cute kitten pictures while running tests). However, in choosing tests in this way, we acknowledge that we have skipped over a great deal of the site’s functionality. In a professional setting, it would be wise to be more thorough.

The next issue to tackle was the issue of the Selenium API itself. Navigating to pages and selecting elements or text from the page is simple enough, but there are more nuances that appeared through actually attempting to test the scenarios and user stories we decided on. For example, one of the biggest hurdles was the fact that Selenium does not seem to be able to select elements within an iframe (which imgur uses to create login and signup prompts). To get around this, we had to test the source URL for the iframe itself. Figuring out how to log into the site without being able to select the individual textarea elements was almost impossible until we discovered a static login page for imgur. Other issues like this popped up while creating tests, but they were eventually overcome thanks to the robust nature of Selenium – it even includes the ability to spoof touchscreen interaction on a browser, which is neat. Selenium is quite finicky at times though, as one of us encountered the phenomenon where Firefox would crash and the program would display a fatal error whenever driver.quit() was called.

One of the biggest issues we faced was imgur itself. The site is, obviously, image-heavy, resulting in long test execution times. Some elements are added or removed via JavaScript, making them almost impossible to select with Selenium without it being a significant hassle. JavaScript in particular made us rethink most of our user stories because they were simply not testable without incurring a major headache on our part. Additionally, the fast-moving, high-userbase nature of the site actually caused one of our simplest tests to consistently fail; in a real professional environment, we would be testing on a development or review server as opposed to the production server to mitigate these sorts of issues.

Another issue we faced (that is again similar to the last deliverable) was deciding exactly how to test the scenarios that we came up with. We eventually agreed that a scenario should be analogous to a single unit test, but the issue of how to create our assertions was a common source of confusion throughout the entire project. For example, take the test s1\_keyboardShortcuts.test\_favorite(); there are a few different ways to test this: going to the user account’s “favorites” page and verifying the favorited image appears there, checking that the upvote counter incremented with the press of the favorite button, or checking that the favorite button now displays a visual indication of the favoriting action. For navigating to the user account’s list of favorites, we thought this might be relying on too many other dependencies (for instance, this might be controlled by a database somewhere, meaning there are functions to add to or fetch from the database, etc.), so we decided against testing it in that way. For incrementing the upvote counter, we didn’t think this was a natural thing to do since upvoting on its own also increments the upvote counter and favoriting seems to be a separate action. Thus, we decided to test the visual indication of the action, which amounts to checking the CSS classes applied to the HTML element; this information was gathered via investigating the webpage using Chrome’s Developer Tools. We were not sure if this was appropriate, however, as it seemed as though we were almost testing at a level below feature-level testing in this way. So we had to address the question of “what makes up the ‘favorite’ feature?” This is something that we encountered often, because were a human is able to almost instantly verify a feature, a program has a much more difficult time doing this. Eventually, for the favoriting example, we decided that the complete test of the favoriting feature was to check if it is indeed added to the user account’s favorites gallery.

An important takeaway from this deliverable is the importance of being able to recognize a feature as a whole, as mentioned in the above paragraph. If we were to continue testing this project, it would be very beneficial for us to track down a list of imgur features from a design document or something similar. This would give us the complete set of features that need testing, which would greatly simplify the task of creating scenarios and user stories. It would also ensure a more “complete” test of the site. If we were to move forward without this list, it’s very possible that we would be overtesting or needlessly testing parts of the site that are not necessarily features per se.

Additionally, I would guess that as we attempted to move forward in testing imgur, the tests would become more difficult to automate with Selenium. As mentioned before, we have already run into the issue where Selenium is unable to select individual HTML elements within HTML tags in iframes, and I would expect that kind of issue to be common on a modern site full of JavaScript functions and modal popup boxes. I would also recommend that the project branch out to different web browsers and mobile devices (via actually installing a Selenium plugin on a mobile browser instead of attempting to access the mobile site on a desktop computer) in order to identify cross-platform issues. I expect that in particular to be a difficult feat, but if the goal is to properly and thoroughly feature test a website, then it is something that should be considered.

Finally, if we were to continue this project, I would expect our user account to be banned for spamming.

**FAILED TESTS**

Note that all tests have the possibility to fail when executed due unseen factors such as the webpage taking longer to load than the WebDriver’s implicit wait time or Selenium choosing to disregard that implicit wait time. We attempted to minimize this sort of thing in our code, but every so often a glitch like this would cause a random test known to pass to fail.

***s1\_keyboardShortcuts.java***

*test\_minus()*

This test fails consistently for the most baffling of reasons. It seems that the image point counters update in real time. Every single time we try to check that the counter has decreased due to a downvote, it appears that at least two people have upvoted the “random” image that imgur has served us, cancelling out the downvote that we have cast. While we can verify visually that the downvote occurs, it’s much harder to due programmatically due to this real-time update phenomenon. I suppose there’s just an abundance of positivity on imgur?

***s2\_login.java***

*test\_uploadButton()*

This test fails (or rather, errors) because imgur allows users to upload images to their site without having to create a user account first. This image is given a page and a permalink (most likely to be removed after a few days) so that images can be easily shared in this way. However, these images cannot be “published” to the site and appear in feeds and streams without a user account. If we had tested for the ability to publish rather than strictly upload an image, this test would have passed. We decided to only test the ability to upload because we assumed that was the “base” functionality of the site.

***s3\_comments()***

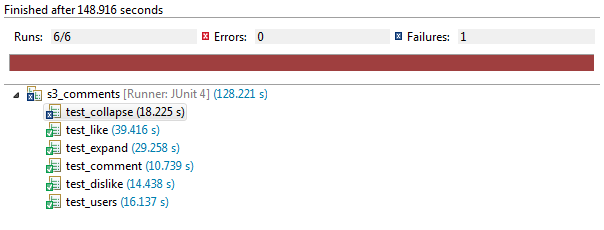
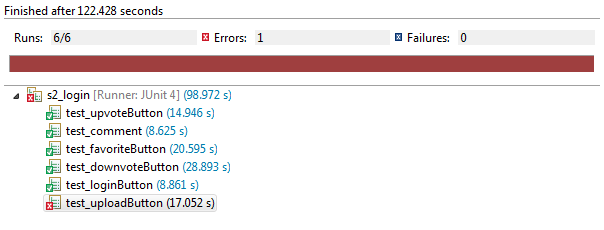
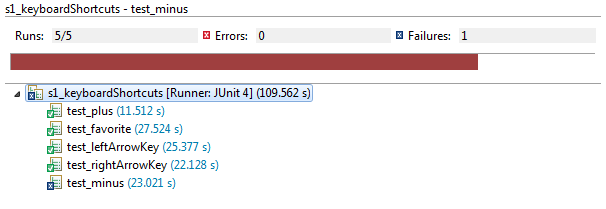
*test\_collapse()*

Without looking to deeply at the site’s code, I assume that the reason that the expand test passes and the collapse test fails is because the expand function actually adds comment elements into the HTML whereas the collapse function simply hides these newly added elements. Selenium finds these elements despite them being hidden, resulting in the two lists being of the same length before and after collapsing. This is another example of something simple for humans to verify that is difficult for a computer to check.

**SPECIAL INSTRUCTIONS**

There are no special instructions for running these tests.

**EXECUTED TESTS**

****