Feed Reader Project

## Overview

The purpose of this project was to use [Jasmine 2.1](http://jasmine.github.io/2.1/introduction.html) to test an existing web-based application that uses RSS feeds. I used Jasmine to make sure that the application was working properly.

Click on the index.html page to open the application. The application will begin with a feed of four links to the most recent articles in the Udacity Blog. Click on the list/sandwich icon in the upper left hand corner to open up a menu that lists links to feeds of Udacity blog articles, CSS Tricks, HTML5 Rocks articles, and Linear Digressions links.

Each of these links will change the content of the displayed list of links in the main section of the index.html page to be from the corresponding clicked feed.

I used Jasmine to make sure that the feeds for each of the different topics contained at least one link. I also tested the menu on the left hand-side of the page to make sure that it was showing and hiding correctly. After that, I also tested that the feed was loading asynchronously.

## Tests

### Test that the allFeeds array exists.

After looking through the documents given to me, I determined that the “allFeeds” variable was an array containing the information for each of the four feeds (listed in the menu).

In the feedreader.js page, I tested that the “allFeeds” variable existed by writing a simple jasmine test that expected “allFeeds” “toBeDefined”.

I also wrote a test that determined if the “allFeeds” was full by determining that the length of the array was not 0, in other words the test showed, that there was at least one feed object present within the array of feeds.

I used the Jasmine Documentation for Jasmine 2.1 as a reference in writing these tests. You can find the documentation here:

<http://jasmine.github.io/2.1/introduction.html>

### Test that each feed’s name and url is defined and present.

These tests were similar to the test for the “allFeeds” array. I tested that the name and url variables were defined. I also tested that they were defined by confirming that the value was not null.

The main difference from the “allFeeds” test, was that these tests where placed within a for-loop. The loop tested for a name and url for each of the feeds within the “allFeeds” array.

These tests and the test for the “allFeeds” array were organized within the same test suite named “RSS Feeds”.

### Test that the menu is hidden on default.

I created another test suite with the name “menu”. I wanted to test that the menu was hidden when the page initially loaded. In order to do this, I had to figure out how the application was hiding and showing the menu on the left hand side.

Using the dev tools, I surmised that the menu was hidden when the body element had a class of “menu-hidden”. When the menu icon in the upper left was clicked, I noticed that the “menu-hidden” class was removed from the body class.

The “menu-hidden” class had a “translate 3d” value of (-12, 0, 0) and “menu” class had a “translate 3d” value of (0, 0, 0). Based upon the explanation from Web Platform, I understood this to mean that when the “menu-hidden” class was present, the menu shifted from it’s left hand corner starting at the x value of 0 to starting at the x value of -12. At this point, the menu is hidden. I referenced this link from Web Platform:

<https://docs.webplatform.org/wiki/css/functions/translate3d(>)

Within the menu test suite I created the variable “bodyClass” which was equal to “document.body.className”. I then wrote a test that expected the “bodyClass” to be “menu-hidden”. This test made sure that the body had a class of “menu-hidden” upon the first page load.

### Test that the menu changes visibly when the menu icon is clicked

Within the test suite named “menu”, I also wanted to test to make sure that the class “menu-hidden” disappeared and then reappeared when the menu-icon was clicked and then clicked again.

Using the dev tools, I knew that I would be working with the icon identified as “a.menu-icon-link” and a click function. I experimented with using a spy feature mentioned in the Jasmine documentation, but this resulted in quite a bit of frustration. After posting this problem to the Udacity forum, I was pointed towards the much simpler direction of triggering a click event.

I wrote two expect statements within the same test. The first statement, expected the class name of the body to not be “menu-hidden” and the second one, expected the class name to be “menu-hidden”. I then used a trigger method to test the click function on the menu-icon, before running each of the expect statements.

I actually had a bit of trouble with this as my test kept failing with the message that I was expecting “menu-hidden” not to be “menu-hidden”. I ended up splitting the variable “bodyClass”, which I had defined within the entire test suite (named “menu”) into two variables: body and className. I defined “body” as “document.body” and className as className.

I figured out that the one variable hadn’t worked because it was assigning the class name, “menu-hidden” to the body element within each of the test.