

SENG 299: SOFTWARE ARCHITECTURE & DESIGN
LAB 7: UNIT TESTING
Student: William Grosset (V00820930)
Lab Instructor: Ali Dehghan
Due: July 10th, 2016

Purpose

The purpose of this lab is to get students familiar with unit testing.

Overview

Here is a described list of concepts that were explained in lab:

1. **Unit testing:** Unit testing allows programmers to check units of code. This can be very useful in situations when code needs to be integrated into a larger code base. Unit testing allows programmers to isolate the functionality and test what is necessary. This process can be very flexible and inexpensive, allowing programmers to “self-check” their own work. Repeating this process of testing allows reassurance of your own design and improves maintainability. As demonstrated in lab with Mocha and Blanket, unit tests can give instant visual feedback and help redesign when necessary. In software engineering, it is vital to distribute quality programs to customers, which requires complete, proper testing of a system.
2. **Mocha:** Mocha is a test framework that runs on Node.js and in the browser. This framework allows flexible and accurate reporting, which makes asynchronous testing simple. Mocha can be easily used to test all control flows of the code and assist in redesigning functionality easily. There are many different libraries to use, such as: `should.js`, `express.js`, `chai`, etc. For example, the Chai library has many predefined functions for testing different objects, strings, booleans, etc. As demonstrated in lab, our program uses the `chai` assertion library and utilizes the `assert` function which allows us to write our own test expressions.
3. **Blanket:** The Blanket tool allows code coverage measurement of unit tests. `Blanket.js` allows customizability to users, or can be run as is. Using Blanket requires configuration of the “`package.json`” file, specifying the main file that needs to be tested. Blanket works through the process of: loading the source files, parsing the code, and connecting to the “hook” in the test program to output coverage details.¹ As demonstrated in lab, Blanket can be integrated with Mocha in a command line to test and provide complete code coverage.

Lab Exercise

The goal of the lab exercise is broken up into three parts. Firstly, students were asked to write at least two *JavaScript* functions of code to be tested. Secondly, students are then required to write corresponding unit tests for each function that was previously written (using *Mocha*). Lastly, as a bonus, students were asked to integrate *Blanket.js* with the *Mocha* framework to measure their code coverage.

PART 1:

Initially, I wrote a simple function to test if a value is even or odd. This function only has 2 possible answers (with real integers). As shown in Figure 1.0, there is a simple **if** and **else** statement that cover each case.

```

3 ▼ function oddOrEven(value) {
4 ▼     if (value%2==0) {
5         return (value + " is even.");
6     }
7 ▼     else {
8         return (value + " is odd.");
9     }
10 }

```

Figure 1.0. oddOrEven() Function

I then wrote a function that takes a string and reverses it (including empty strings). As shown in Figure 1.1, there is a simple **for loop** that starts from the end of the string, which appends each character in the string array to a new string (e.g var flip).

```

12 ▼ function stringFlip(string) {
13     var flip = "";
14 ▼     for (var i = string.length; i >= 0; i--) {
15         flip += string.charAt(i);
16     }
17     return flip;
18 }

```

Figure 1.1. stringFlip() Function

PART 2:

After writing functions to test, I then proceeded to install and use the *Mocha* test framework (`npm install mocha`). I wrote test units in the same JavaScript file as my original functions. This required the assertion library, which can be used for test cases that determine whether or not the output of the function is as expected.

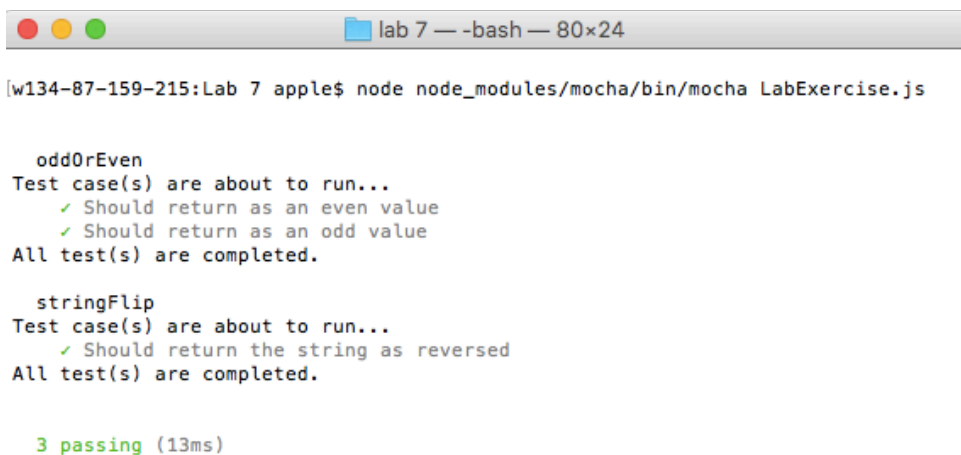
Next, I wrote a **describe** function for each previous function that specifies the function I am testing (see Figure 2.0). Inside this function, I created two integer values to test with, which test both cases in the `oddOrEven()` function. In the other function, I specified a string value to test with to flip in the `stringFlip()` function. Using the `assert()` method, the first parameter I write my own test statement, with the second parameter is the error message that is thrown if the first parameter statement is false. In both **describe** functions, I used the `before()` and `after()` function to have the code tested in the desired block of my JS file. As shown in Figure 2.1, the command: `mocha LabExercise.js` lets us see if all cases are properly, which is represented by a green checkmark.

```

20 describe('oddOrEven', function() {
21   var testNum1 = 60;
22   var testNum2 = 61;
23
24   before(function() {
25     console.log('Test case(s) are about to run...');
26   })
27
28   it('Should return as an even value', function() {
29     assert(oddOrEven(testNum1)==(testNum1+" is even."), "The value is not even.")
30   })
31
32   it('Should return as an odd value', function() {
33     assert(oddOrEven(testNum2)==(testNum2+" is odd."), "The value is not odd.")
34   })
35
36   after(function() {
37     console.log('All test(s) are completed.')
38   })
39 })

```

Figure 2.0. Mocha describe() Function



```

lab 7 — -bash — 80x24

[w134-87-159-215:Lab 7 apple$ node node_modules/mocha/bin/mocha LabExercise.js ]

  oddOrEven
  Test case(s) are about to run...
    ✓ Should return as an even value
    ✓ Should return as an odd value
  All test(s) are completed.

  stringFlip
  Test case(s) are about to run...
    ✓ Should return the string as reversed
  All test(s) are completed.

  3 passing (13ms)

```

Figure 2.1. Unit Test Passes (Mocha on Terminal Window)

Both of control statements (if, loop) in each function are important to test. As a programmer, certain cases can be missed and not tested by accident, which can cause major issues in particular situations (especially in web servers, embedded systems, etc.).

PART 3:

Initially, I also had to install *Blanket.js* (`npm install blanket`). Running the previous tests allowed me to do checks on particular cases. However, utilizing the *Blanket* library with *Mocha* allows me to check for complete coverage. I specified the name of the main file to be tested (`LabExercise.js`) and excluded the `node_modules` directory from coverage in the “package.json” file. I then ran *Mocha* with *Blanket* and saved the output in an HTML file. As shown in Figure 3.0, the coverage reports shows 100% for all functions.

Coverage

100% coverage 29 SLOC

/Users/apple/Documents/BracketsCode/SENG299/lab
7/test/LabExercise.js

100% coverage 29 SLOC

```
1 1 var assert = require('assert');
2
3 1 function oddOrEven(value) {
4 2     if (value%2==0) {
5 1         return (value + " is even.");
6
7     }
8     else {
9 1         return (value + " is odd.");
10
11     }
12 }
13
14 1 function stringFlip(string) {
15     var flip = "";
```

Figure 3.0. Blanket.js Tool

Constraints

Initially, I had a Mac administrator issue that was occurring when I tried to install node packages. After researching online, the proper way to navigate and install without restriction (while in terminal) is to use the `sudo` command as necessary. This tool allows non-root users to perform tasks that would only be allowed by the root user account.

Also, I had issues with running the *Blanket.js* on the ELW lab computers. This issue was due to an older version of NodeJS being used that cannot be upgraded on the lab machine. To fix this, I upgraded my NodeJS version on my laptop and was able to utilize the *Blanket* library properly.

Conclusion

In this lab, I got a first hands-on experience with proper unit testing. This lab has taught me about the importance and proper usage for code coverage in all applications of programming. I love being to get experience with different languages, frameworks, and methodologies in labs, as it improves my overall coding and problem solving skills. I look forward to using the *Mocha* framework with *Blanket* for testing functions in the Go project.

References (MLA)

1. "Alex-Seville/Blanket." *GitHub*. Alex Seville, 2012. Web.