Chapter 5

Lab 5.2

Objective: Test-driven Development (TDD) on Dromedary using Chai JS and Mocha

According to Chai Assertion Library,

> "***Chai*** *is a BDD / TDD assertion library for node, and the browser that can be delightfully paired with any JavaScript testing framework*".

Overview:

1. Run current tests to see the code coverage.
2. Add a trivial test to the existing code base.
3. Use TDD and add a new color option to voting.

Run Current Tests to See the Code Coverage

1. To see where testing currently stands, run the tests with either command:

   ```
   $ npm test
   ```

   OR

   ```
   $ gulp test
   ```

   You should see that there are 14 tests passing. Find `.incrementCount()`, and note that there is one test that it runs.

Add a Trivial Test to the Existing Code to See the Syntax

This is not TDD, nor does it increase the code coverage. The test that has been added just looks at different possible outcomes of the same functionality. In this situation, the test covers a case that should not occur.

1. There is a `inMemoryStorage.js` file that is located in `/dromedary/test/`. Open it with a text editor of your choice, and take a look:

   ```
   $ vi test/inMemoryStorage.js
   ```


2. Create an *assert variable* at the beginning of the file, but after the **`requires`**:

```
$ var assert = ("chai").assert;
```

3. Next, create another unit test to verify that colors are only incremented by "1" when **`incrementCount()`** is called. Add the code below into the file. This will go into the section that describes **`incrementCount()`**:

```
it("doesn't add two to the count per increment", function(){
var color;
for (color in this.colorCounts) {
  assert.notEqual(backend.getCount(color), this.colorCounts[color]+2);;
}
});
```

4. Exit the editor and run the tests with the command that you previously used. 15 tests should pass, and you should see that **`.incrementCount()`** has now two different tests in there.

A note about automated testing with JavaScript, **gulp**, and **npm**:
This repository is already set up with the testing automation ready to use. Open **`gulpfile.js`** and find the test **`gulp task`**. It is line 29. If you search for "Execute unit tests", you will also find the section. Once there, notice the way it is hooked up. If you add new files in the application itself, then you can simply add another test file and it will be tested. **`npm test`** runs **`gulp test`**. Leveraging this in a testing system like Jenkins CI or Travis CI will allow the team working on this to run tests on every commit.

Add a New Feature Using TDD

The additional feature includes adding the 4th color to the chart. First, write/alter the tests, and then, run the tests. After the tests fail, add the code necessary to pass. In this case, the code is **`JSON`**. When this is done, there should be a new color added to the chart.

1. Open **`/dromedary/test/inMemoryStorage.js`** in a text editor. Locate **`expectedNumberOfItems`** and change the value to "4". This variable is used in the testing to make sure that the proper number of colors is set up for the chart.

2. Run the tests again. There should be two failures. One test will fail in **`getForChartJs()`** and another in **`getAllCounts()`**.

3. Provide the code to add a color to the pie chart and test.

4. Open **`/dromedary/lib/inMemoryStorage.js`**. The **`chartData`** variable is a **`JSON`** list of colors. Create another color using the previous colors as an example.

5. Run the tests again and all tests should pass.

6. Run the application again and notice that the new color exists.

