Chapter 5

Lab 5.3

Objective: Behavior-driven development (BDD) on Dromedary using Chai JS and Cucumber

Using Chai for BDD

1. Overview of BDD using Chai JS and Mocha

   The previous use of Chai was in a TDD approach. The BDD is similar, but the syntax is typically more human-readable. One of the main benefits of Chai is the ability to use ordinary language, such as "**`describe`**", "**`should`**", and "**`expect`**".

2. Refactor previous tests to use "**`expect`**" instead of "**`assert`**"

   Using a text editor, open **`/dromedary/test/inmemorystorage`**, then find and comment out the following line:

   ```
   // assert.notEqual(backend.getCount(color), this.colorCounts[color]+2);
   ```

   Now, add the following line in its place:

   ```
   expect(backend.getCount(color)).to.not.equal(this.colorCounts[color]+2);
   ```

   This test verifies the same functionality in the software, but it is slightly more readable. Different frameworks have different syntax, but similar trends appear throughout all testing frameworks. You would use Chai JS with "**`expect`**" and "**`should`**", using the same **red/green/refactor** testing lifecycle.

Using Cucumber for BDD

Cucumber is a testing tool used in BDD to define features. It also acts as documentation for software. It works with Java, JavaScript, Ruby, selected frameworks, and more; and it helps teams create reliable software more efficiently.

1. Background on Cucumber

   Cucumber is a BDD tool that serves as both a testing framework and documentation for the project being worked on. One of the biggest benefits of this tool is that the **`.feature`** files are written in Gherkin -

plain-text English with some keywords thrown in - making it very easy for non-programmers, like business managers, to understand the software and make sure it matches the specification.

2. Install Cucumber using **npm**

   Install Cucumber as a development dependency. Use the `--save-dev` flag to add the development dependency in the `package.json` file. Run this inside of the `dromedary` directory:

   ```
   $ npm install --save-dev cucumber
   ```

3. Create the file structure for Cucumber

   Create a new directory for features:

   ```
   $ mkdir features
   $ cd features
   ```

   Next, create the `step_definitions` directory:

   ```
   $ mkdir step_definitions
   ```

   Try to run `cucumber-js` to make sure it runs:

   ```
   $ cd ..
   $ node_modules/.bin/cucumber-js
   ```

   Verify the correct output:

   ```
   0 scenarios
   0 steps
   0m00.000s
   ```

   If that has passed, then Cucumber is successfully installed. Otherwise, some troubleshooting may be required.

4. Write a feature using the Gherkin language

   Cucumber is BDD because each feature is written into a `.feature` file and then tested. These files are written in a Domain Specific Language (DSL) called “Gherkin” which is plain-text based. These features also act as documentation, because a non-programmer can typically read them.

   In order to keep this course focused on a test, the feature to be introduced is adding an additional color to the test.

Inside of **/dromedary/features/** use a text editor and open **add_fifth_color.feature**. Add the following:

```
Feature: User can choose from 5 colors
As a user
I want to be able to choose from 5 colors
So there is a better chance my favorite color is an option

Scenario: A fifth color is clicked
    Given there are 5 colors
    When I click the color
    Then the color updates the value
```

Now that the feature has been completed, rerun Cucumber (**node_modules/.bin/cucumber-js**). This will return the following outputs:

- The feature being tested and its description.
- Warnings about scenarios missing **step_definitions** and snippets to outline the code needed.
- A number of scenarios and steps that are undefined.

The programmer of the tests can use this information to make the proper amount of tests needed to cover all of the features/scenarios.

5. Create **step_definitions** outlines.

The Gherkin code in the **.feature** files needs to be accompanied by **step_definitions** in order to run actual tests. For JavaScript, these **step_definitions** are in the **.steps.js** files. Cucumber helps by providing the snippets of scenarios not covered.

Note: The file type for steps changes depending on the language used during development.

Open **/dromedary/features/step_definitions/add_fifth_color.steps.js** in a text editor.

First, add the following code:

```
'use strict';
var {defineSupportCode} = require('cucumber');
var InMemStor = require("../../lib/inMemoryStorage.js");
var colors = new InMemStor;
var assert = require('assert');
var chartData = colors.getChartData();
var color;

defineSupportCode(function({Given, Then, When}) {

Given(/^there are five colors$/, function () {
    // Test Code goes here
```

```
  });

  When(/^I click the color$/, function () {
    // Test Code goes here
  });

  Then(/^the color updates the value$/, function () {
    // Test Code goes here
  });
});
```

Note: You can copy the snippets provided by Cucumber from step 4 into the function. You can take the snippets and alter them to fit your needs. The code provided above is custom, but below you can find an example of a snippet:

```
this.Given('there are five colors', function (callback) {
// Write code here that turns the phrase above into concrete actions
callback(null, 'pending');
});
```

6. Write **`step_definitions`**

Open the file **`/dromedary/features/step_definitions/add_fifth_color.steps.js`** that was just created. In the “**`Given`**” function, add the following lines:

```
Given(/^there are five colors$/, function() {
var length = Object.keys(chartData.values).length; // Gets number of colors
in chartData
assert.equal(length, 5); // asserts number of colors
});
```

Now, run **`cucumber-js`** again, and notice that the test fails. The test will pass only if the fourth color has been added to **`/dromedary/lib/inMemoryStorage.js`** during in the previous steps.

The “Given” step checks the starting condition of the action.

The goal of this action is to make sure that incrementation is possible with the new color. Navigate back to **`/dromedary/features/step_definitions/add_fifth_color.steps.js`**. The “**`When`**” function will be used to capture the action. In the “**`When`**” function, add the following lines:

```
When(/^I click the color$/, function () {
color = 'blue'; // the color to be added
colors.incrementCount(color, function(err) {
  if (err) {
    assert.false();
  } else {
    assert.true();
  }
});
});
```

This function will verify that the new color can be incremented.

The “**Then**” function will contain the effect of the action which in this case is having an incremented count of the new color. In the “**Then**” function, add the following lines:

```
Then(/^the color updates the value$/, function () {
assert.equal(colors.getCount(color), 11);
});
```

This verifies that the incrementation worked properly. It assumes that the value was started at the seed number of “10”. If the starting value is not the same, then change the hard-coded “11” to the appropriate value, such that it is the initial seed value + 1.

7. Run the tests and change to code pass

   The final **add_color_steps.steps.js** should look as follows:

```
'use strict';
var {defineSupportCode} = require('cucumber');
var InMemStor = require("../../lib/inMemoryStorage.js");
var colors = new InMemStor;
var assert = require('assert');
var chartData = colors.getChartData();
var color;

defineSupportCode(function({Given, Then, When}) {

  Given(/^there are five colors$/, function () {
    var length = Object.keys(chartData.values).length;
    assert.equal(length, 5);
  });

  When(/^I click the color$/, function () {
    color = 'blue'; // Change to your color key
    colors.incrementCount(color, function(err) {
      if (err) {
        assert.false();
      } else {
        assert.true();
      }
    });
  });
  Then(/^the color updates the value$/, function () {
    assert.equal(colors.getCount(color), 11);
  });
});
```

Run **`cucumber-js`** again, and notice that the tests fail. Write the new color into **`/dromedary/lib/inMemoryStorage.js`**, and run the tests again; all of them should pass. That was adding a new color via BDD.

BDD Final Thoughts

Though this was a very simplistic example, it illustrates the way BDD can be completed with Cucumber. Cucumber is a great tool for writing efficient and reliable software. Remember to do this in a true TDD / BDD style with the **red/green/refactor** testing lifecycle.