CMPS 356 Enterprise Application Development - Spring 2019

Lab 6 –JavaScript Asynchronous Programming

Objective

The objective of this lab is to practice JavaScript asynchronous programming using **Callbacks**, **Promises** and **async/await**.

Overview

This lab has three parts:

- **Part A:** Practice JavaScript Unit testing using Mocha and Chai (1h).
- **Part B:** Practice Callbacks, Promises and Async-Await (1h 50mins).

Part A – Unit Testing Using Mocha and Chai

1. Sync cmps356-content repo to get the Lab files.
2. Copy ***Lab6-AsyncJS*** folder from cmps356-content repo to your repository.
3. Open ***Lab6-AsyncJS\UnitConverter*** in Webstorm. You should see a JavaScript file named *UnitConverter.js*. In this exercise, you will create a spec file to unit test the function of the *UnitConverter* class.
4. First, create the package.json file using `npm init .` This file is used to define dependencies by listing the npm packages used by the app.

   Refresh your project to see the **package.json** file.
5. Install mocha and chai using *node package manager* (npm):

   ```
   npm install mocha -D
   npm install chai -D
   ```

   This will add 2 dev dependencies to package.json file.

   ```
   "devDependencies": {
     "chai": "^4.1.2",
     "mocha": "^5.0.4"
   }
   ```

6. Create a JavaScript file named **UnitConverter.spec.js**
7. Import an instance the *UnitConverter* class to be tested and the *chai expect* package.

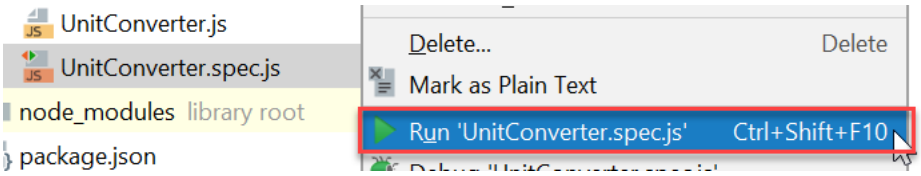
   ```
   const unitConverter = require('./UnitConverter');
   const expect = require("chai").expect;
   ```

8. Write 2 test cases for each method of **UnitConverter** class.

You may start with the following inputs and expected results. Then use search for "google unit converter" to compute the expected results for more input values.

Method	Input	Expected Result
kgToOunce	1	35.274
kgToPound	2	4.4092
meterToInch	1	39.3701
meterToFoot	2	6.5617

9. Run the unit tests as you develop them using WebStorm:



Also run the unit tests from the command line using: `npm test`

But first, you should have the following in package.json file.

```
"scripts": {
  "test": "mocha **/*.spec.js"
},
```

Part B – Practice Callbacks, Promises, and Async-Await

In this exercise, we will collaboratively build the solution of each step after you attempting it by yourself. Then the instructor will demonstrate and explain the model solution. At the end of this exercise, you should be able to use **callbacks**, **promises** and **async/await** to read/write files. This will enable you to complete the remaining tasks of the Lab.

First, open ***Lab5-AsyncJS\CountryExplorer*** in WebStorm. You should see a folder named **data** containing three files: ***country.json*** and ***country-literacy.json*** and ***world-universities.json***

1. Create a JavaScript file named countryRepo-cb.js and implement the following functions. Make sure that you test as you go. You may write the test function in the same file to keep it simple (no need to use mocha).
   a. **getCountries**: reads all the countries from the ***country.json*** file using **fs** package and return an array of countries. **[test it]**

      Note: ***It's critical that you do not move to the next method without testing the previous one. By testing your code frequently as soon as you write them, will help you avoid accumulated errors and allow you to debug your code faster.***

   b. **getCountry(countryName):** return the matching country details. This function should reuse getCountries function. [**test it**].

   c. **getCapitalCity(countryName)** : return the capital city of the matching country

d. **getCountryLiteracy(*countryName*):** returns a country literacy details from country-literacy-rate.json. [**test it**].

e. Now, change the **getCountry** function done in (**b**) to call **getCountryLiteracy** function then return a country object with an extra attribute **literacyRate** [**test it**].

$$LiteracyRate = \frac{maleLiteracy + femaleLiteracy}{2}$$

In the previous exercise, you have seen the **callbacks JavaScript pattern** as an excellent way of writing code that is non-blocking. However, this approach can cause a ***callback hell*** as shown in **getCountry(countryName)** function. We solve this issue by using **Promises and Async-Await**

CALLBACK

```
getData(a ⇒ {
    getMoreData(a, b ⇒ {
        getMoreData(b, c ⇒ {
            getMoreData(c, d ⇒ {
                getMoreData(d, e ⇒ {
                    console.log(e);
                });
            });
        });
    });
});
```

PROMISES

```
getData()
.then(a ⇒ getMoreData(a))
.then(b ⇒ getMoreData(b))
.then(c ⇒ getMoreData(c))
.then(d ⇒ getMoreData(d))
.then(e ⇒ console.log(e));
```

1. To use promises, first install the **fs-extra** package using npm. fs-extra provides promises based API to read/write files:

   ```
   npm install fs-extra
   ```

2. Create a JavaScript file named country-promise.js and re-write the functions you did in countryRepo-cb.js using promises instead of callbacks. Use **`fs-extra`** instead of **`fs`**), i.e., use *const fs = require('fs-extra');* to be able to read from files using promises. Make sure that you test as you go. You may write the test function in the same file to keep it simple.

PROMISES

```
getData()
.then(a ⇒ getMoreData(a))
.then(b ⇒ getMoreData(b))
.then(c ⇒ getMoreData(c))
.then(d ⇒ getMoreData(d))
.then(e ⇒ console.log(e));
```

ASYNC-AWAIT

```
const a = await getData();
const b = await getMoreData(a);
const c = await getMoreData(b);
const d = await getMoreData(c);
const e = await getMoreData(d);
console.log(e);
```

1. Create a JavaScript file named country-async.js. Create a class named CountryRepo and re-write the functions using async-await instead of promises. CountryRepo should have the methods shown in the table below. Make sure that you test as you go. You can create an app.js file to test CountryRepo class (no need to use mocha).

getCountry(countryName)	Returns country object with a literacyRate attribute computed as the sum of femaleLiteracyRate and maleLiteracyRate retrieved from country-literacy-rate.json file.
getNeighboursLiteracy(countryName)	Returns neighboring countries and their literacy rates. The return list of countries should be sorted based on the literacy rate.
getRegionLiteracy(region)	Returns all the countries in the region (e.g., Asia). Sort the list based on the femaleLiteracyRate.
getTop3LowLiteracy	Get the top 3 countries with the lowest literacy rate.
getTop3HighLiteracy	Get the top 3 countries with the highest literacy rate.

Important Notes: the country objects returned by all methods (except the first one) should have the following structure: countryCode, country, femaleLiteracyRate, maleLiteracyRate, literacyRate (which is the average of femaleLiteracyRate and maleLiteracyRate).

Part C –Async-Await – Book Donation

Deadline : 1 hour Before Next Lab

1. Open the project named **BooksDonation**.
2. Inside the project you will find a folder called **data** that contains **catalog.books.json**
3. Create a class named **BooksRepo** and write the following methods shown in the table below using async-await
4. Create a test file called **BooksRepo.spec.js** that tests all the methods inside the **BooksRepo.js**. Make sure that you write all the tests using **Mocha** and **Chai**.
5. Finally create an app.js file to test **BooksRepo** class (no need to use mocha here).

<table>
<tr><td>getBook(bookName)</td><td>Returns the book object if it exists, otherwise null</td></tr>
<tr><td>getBooksWIthPageCountMoreThanX(pageCount)</td><td>Returns all the books that have a pageCount more than the given pageCount. Eg. If the user calls the function with pageCount=1000. It should return all the books that have pagecount >1000</td></tr>
<tr><td>getAuthorBooks(author)</td><td>Returns all the books authored by that specific author.

Note: some books have more than one author. You should consider those too and return them as well.</td></tr>
<tr><td>getAuthorsBookCount()</td><td>Returns a map that contains author name and a count of books their authored .

Eg.
<table><tr><th>Author Name</th><th>Book Count</th></tr><tr><td>James</td><td>2</td></tr><tr><td>Ali</td><td>4</td></tr></table></td></tr>
<tr><td>getBooksbyCatagory(category)</td><td>Returns all the books of that category.

E.g. category = Java => should return all the books that are related to java</td></tr>
</table>

You should use async/await (or promises), and other JavaScript features such as arrow functions, array functions (.map, .reduce, .filter, .splice, .sort…), spread operator, object literals, and classes wherever required.

After you complete the lab, fill in the ***Lab6-TestingDoc-Grading-Sheet.docx*** and save it inside ***Lab6-AsyncJS*** folder. Sync your repository to push your work to Github.