GDP02 _ Group 05 : Workshop on Software Testing

Members: Nilantha Dambadeniya & Pavithar Devdas.
3/15/2019

Unit Testing JavaScript with Mocha Framework

In this documentation, we are going to describe how to use Mocha framework to do unit testing in JavaScript. Here we are using the TDD (Test-Driven Development) approach to explain the unit testing.

In this worksheet, we are going to explain how to develop a simple command line calculator app using Node.js and setup tests with Mocha testing framework.

Scope

The application should add, subtract, divide and multiply any two numbers

Technical Requirements

Node.js server environment and npm installed on the computer.

Setting up the environment

Setup the local environment for Node.js files and dependencies

1. Create a new directory called Calculator
2. In the command prompt, navigate to the directory and initialize a new project using "npm init" command
3. Set the package name, version, description and other common package details. [Important] When you reach the "test command", type "mocha" which is the testing framework we are going to use.
4. Next type "npm install --save-dev mocha" to install the mocha testing framework.

Test the mocha environment

1. Create a “test.js” file in the folder (use the command prompt or you can use any IDE or text editor)
2. Write the following test script into the test.js file to make sure the environment is working fine

```
const assert = require('assert');

it('should return true', () => {
  assert.equal(true, true);
});
```

3. Now, run the test on the command prompt using “npm test”

   If the test is passing, the test environment setup is completed.
4. Create another file named “operations.js” in the Calculator folder to hold all the operation functions.

Building the application

Since we are following the TDD approach, let’s first build the test case for adding two numbers.

1. Replace the script in the test.js file with the following script.
   ```
   const assert = require('assert');
   it('correctly calculates the sum of 1 and 3', () => {
     assert.equal(add(1, 3), 4);
   });
   ```
2. Run the script on command line using “npm test”

   The test should be failed since we didn’t create the add operation yet.
3. Add the following script to the “operations.js” file
   ```
   function add(a,b)
   {
           return a+b;
   }
   module.exports= {add}
   ```
4. Add the operations module to the “test.js” file and use the operations.add to assert the function. Then the test.js file would be like following
   ```
   const operations = require('./operations.js');
   const assert = require('assert');

   it('correctly calculates the sum of 1 and 3', () => {
   assert.equal(operations.add(1, 3), 4);
   });
   ```
5. Run the test on command line using “npm test”

   If it passes the test, we have implemented a function that satisfies the test case.
6. Now, let’s add all test cases to the test.js file. Add the following script to the “test.js” file

```
const assert = require('assert');
const operations = require("./operations.js");

it('correctly calculates the sum of 1 and 3', () => {
   assert.equal(operations.add(1, 3), 4);
});

it('correctly calculates the sum of -1 and -1', () => {
   assert.equal(operations.add(-1, -1), -2);
});

it('correctly calculates the difference of 33 and 3', () => {
   assert.equal(operations.subtract(33, 3), 30);
});

it('correctly calculates the product of 12 and 12', () => {
   assert.equal(operations.multiply(12, 12), 144);
});

it('correctly calculates the quotient of 10 and 2', () => {
   assert.equal(operations.divide(10, 2), 5);
});

it('indicates failure when a string is used instead of a number', () => {
  assert.equal(operations.validateNumbers('sammy', 5), false);
});

it('indicates failure when two strings is used instead of numbers', () => {
  assert.equal(operations.validateNumbers('sammy', 'sammy'), false);
});

it('successfully runs when two numbers are used', () => {
  assert.equal(operations.validateNumbers(5, 5), true);
});
```

7. We can run the test and see how the test get failed.
8. Add all the operations to operations.js file

```
function add(a, b) {
   return (+a) + (+b);
}

function subtract(a, b) {
   return (+a) - (+b);
}

function multiply(a, b) {
  return (+a) * (+b);
}

function divide(a, b) {
  return (+a) / (+b);
}
const validateNumbers = (x, y) => {
  if (isNaN(x) || isNaN(y)) {
    return false;
  }
  return true;
 }

module.exports = {
  add,
  subtract,
  multiply,
  divide,
  validateNumbers,
}
```

9. Run the test

   You can see all the 8 test cases are passed

Conclusion

In this practice, we created few JavaScript functions and Test Case scripts to test those functions. You can create the command line interface file to run the application on command line or use the operations in a different app.
We also tested the TDD approach to develop a basic app, and used the Mocha framework for testing the JavaScript code.