Mocha and Chai

What is NodeJS Unit testing?

- Test-driven development is a powerful tool for preventing bugs within your application.
- NodeJS Unit testing is the process of testing small and isolated pieces of code in your NodeJS application.
- This helps in improving the quality of the code and helps in finding bugs early on in the development life cycle.
- This also provides an added advantage to the users in the sense that they can add any new features without breaking any other part of their application.
- For your NodeJS applications, Mocha and Chai can be used together for Unit Testing.

Unit Testing with Mocha and Chai

- **Mocha** is a widely used JavaScript test framework running on NodeJS and browsers.
- It supports asynchronous testing running the tests serially, allowing for more flexible and accurate reporting.
- It is a highly customizable framework that supports different assertions and libraries.

Unit Testing with Mocha and Chai

- Chai is an assertion library that is mostly used alongside Mocha.
- It can be used both as a BDD / TDD assertion library for NodeJS and can be paired with any JavaScript testing framework.
- It has several interfaces that a developer can choose from and looks much like writing tests in English sentences.
- BDD provides an expressive and readable style of language via **Should & Expect**, whereas TDD provides a more Classical approach via **Assert**.

How to write Unit tests?

- There are two main methods to write Unit Tests as seen below:

  •**describe()** – It is a suite of Test scripts that calls a global function with two parameters: a string and a function.

  •**it()** – It is the smallest unit test case that is written to be executed.

    •it() calls a global function with two parameters i.e. a string and a function.

    •You can write multiple it() statements inside a describe() method.

How to write Unit tests?

- The third method used in a Unit Test is based on the developer's choice. Every **it()** statement has one of the below functions which take a value and expect a return in true form:

  •**expect()** – It is a BDD style library. Natural language assertions are chained together here. This is mainly used with non-descript topics such as booleans or numbers.

  •**should()** – It is a BDD style library. Natural language assertions are chained together in this case as well. However, it extends each object with a should property to start the chain.

  •**assert()** – It is a TDD style library. It provides additional tests and is browser compatible.

Installing Mocha and Chai

- Step 1: Create a new directory for your project file using the following command:

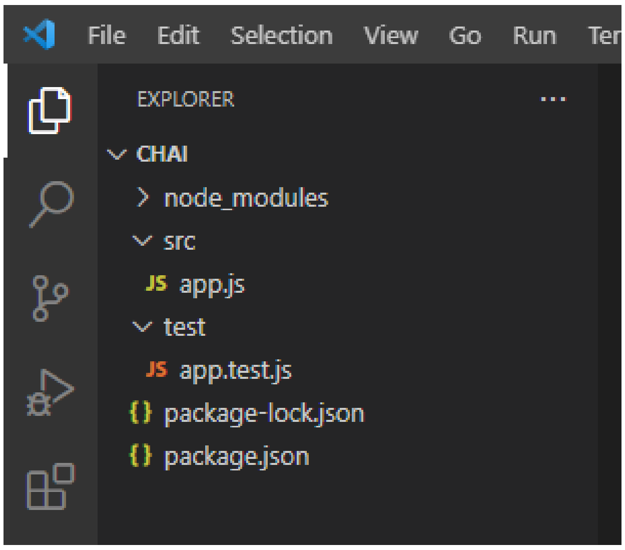
```
mkdir Chai
```

- Step 2: Go to the new directory and execute the below command to initialize a project with Default configurations:

```
cd Chai
npm init –y
```

Installing Mocha and Chai

- **Step 3:** The step 2 creates a **package.json** file. Open the project in VC IDE.



Installing Mocha and Chai

- **Step 4:** Create two folders named **src** and **test** respectively.
  - **src** stores the main file where the source code of the program is written
  - **test** folder stores test cases for unit testing.

Step 5: Create an **app.js** file under the **src** folder and

app.test.js file under the **test**.

Installing Mocha and Chai

- **Step 6:** Open the **package.json** file and change the "**scripts**" block to "**mocha**"

```
{
  "name": "chai",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "mocha"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type" : "module",
 }
```

Installing Mocha and Chai

- Step 7: In the terminal, type the following for installing mocha and chai:
  - For Global installation of Mocha:
    ```
    npm install mocha –g
    ```
  - For Project installation of Mocha:
    ```
    npm install mocha -- save-dev
    ```
  - For installation of Chai:
    ```
    npm install chai -- save-dev
    ```

Installing Mocha and Chai

- **Step 8:** The **package.json** file will look like this once both Chai and Mocha are installed:

```
{
  "name": "chai",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "mocha"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type" : "module",
  "dependencies": {
    "chai": "^4.3.6",
    "mocha": "^10.0.0",
    "save-dev": "^0.0.1-security"
  }
}
```

Creating a Simple NodeJS App

```
function simpleInterest(p , t, r ){
        let si = p * t * r / 100;
        return si;
}

function compoundInterest(p , t , r) {
        let ci = p * (1+r/100) ** t;
        return ci;
}

export{simpleInterest, compoundInterest}
```

NodeJS Unit Testing with Mocha and Chai

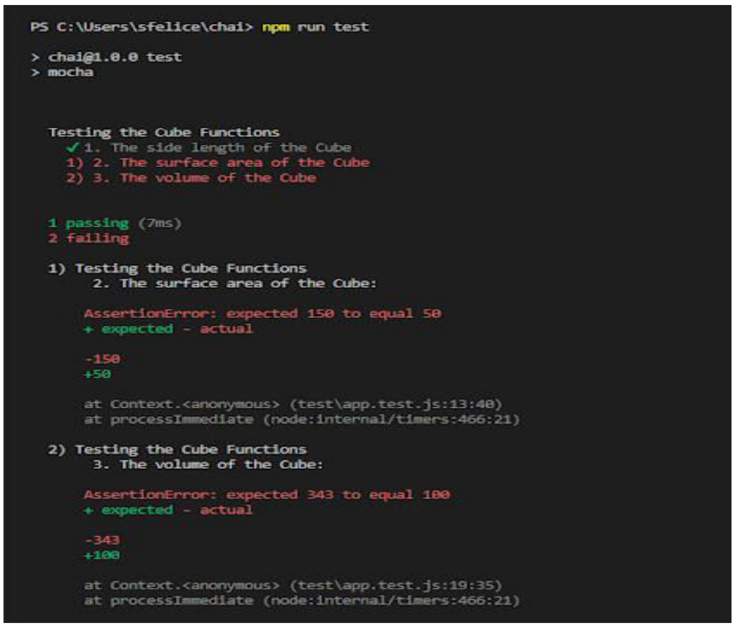
```
import { simpleInterest, compoundInterest } from '../src/app.js';
import { expect } from 'chai';


describe('Interest',()=>{
    it('Simple interest',()=>{
        expect(simpleInterest(1000,2,14.5)).equals(290);
    });

    it('Compound interest',()=>{
        expect(compoundInterest(1000,2,14.5)).equals(1311.025);
    })
})
```

Run the test

- Run the test using the below command
  npm run test



```
PS C:\Users\sfelice\chai> npm run test

> chai@1.0.0 test
> mocha



  Testing the Cube Functions
    ✓ 1. The side length of the Cube
    1) 2. The surface area of the Cube
    2) 3. The volume of the Cube


  1 passing (7ms)
  2 failing

  1) Testing the Cube Functions
       2. The surface area of the Cube:

      AssertionError: expected 150 to equal 50
      + expected - actual

      -150
      +50

      at Context.<anonymous> (test\app.test.js:13:40)
      at processImmediate (node:internal/timers:466:21)

  2) Testing the Cube Functions
       3. The volume of the Cube:

      AssertionError: expected 343 to equal 100
      + expected - actual

      -343
      +100

      at Context.<anonymous> (test\app.test.js:19:35)
      at processImmediate (node:internal/timers:466:21)
```