

TDD in JavaScript

Section 3: Browser Based Testing

3.1 Creating Test UI, Enabling Browser Testing and Browserify

We can move Mocha and Chai TDD into the web browser. The benefit of this is to allow someone to visit a page and run those test right inside the browser and see what is going on without having to worry about running anything via a command line.

In our projects within the tests directory we can create a new index.html page and use a css stylesheet to style this test page. The mocha.css file can be found in the mocha module folder (node_modules/mocha/mocha.css) copied and pasted into the test directory. This will help style up the visual of the test within the browser.

The index.html will actually run the test in the web browser. Below is the example index.html code for our test web page.

```
tests > index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <link rel="stylesheet" href="mocha.css">
7      <title>Browser Based Testing (TDD)</title>
8    </head>
9    <body>
10     <div id="mocha"></div>
11     <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.0.0-alpha1/jquery.min.js"></script>
12     <script src="https://cdnjs.cloudflare.com/ajax/libs/mocha/2.3.4/mocha.min.js"></script>
13     <script src="https://cdnjs.cloudflare.com/ajax/libs/chai/3.4.1/chai.min.js"></script>
14     <script src="https://cdnjs.cloudflare.com/ajax/libs/sinon.js/1.15.4/sinon.min.js"></script>
15
16     <script src="calculator-test.js" defer></script>
17
18     <h1>Mocha Unit Tests</h1>
19     <script>
20       mocha.setup('bdd');
21       window.expect = chai.expect;
22       window.onload = function() {
23         mocha.run();
24       };
25     </script>
26   </body>
27 </html>
```

This imports a few script libraries such as jquery, mocha, chai and sinon minified js files from the web as well as our calculator-test.js file. The defer attribute will only load this JavaScript file once everything else has loaded on the page before it can execute.

At the bottom of the index.html file we would have a script tag to run a JavaScript code. This script will define the setup of the test runners/ The mocha.setup('bdd') will setup the mocha behavioural driven development environment which will give us reference to some

things we were already using such as `describe()`, `it()`, `before()`, `after()`, etc. in order to be available in the browser. Using the `chai.expect` (i.e. `chai` assertion library). Finally, on the `windows` object we would use the `onload` method to run an anonymous function which will call the `mocha` test runner using the `mocha.run()` method.

This will complete the `index.html` test page which we can view in the browser. We now need to setup a web server to run this page on a web server. One thing to note is that we would also see an error loading the `calculator-test.js` because JavaScript does not understand the `require` keyword which is a Node.js keyword and not part of the JavaScript engine inside the browser. We would have to use something else to get around that error.

To create a `http` server we would need to install two `npm` packages which are `http-server` and `concurrently` using the following commands within our project directory path in the terminal:

```
$ npm install http-server concurrently --save-dev
```

`Concurrently` will allow us to run different terminal commands concurrently as the package name suggests.

Once both packages have installed we would need to make some changes to our `package.json` file which should have the two installed modules added to the `devDependencies` property. The changes would need to be applied to the `script` property to add a new script using commands from the newly installed packages.

The `start` script will allow us to run `node.js` server while the `myTest` and `open` scripts will use the `concurrently` to run two commands concurrently i.e. the first is to setup a web server on `localhost` and the port specified while the second will open `URL` path in the default browser.

```
{
  "scripts": {
    "test": "./node_modules/mocha/bin/mocha",
    "start": "npm run",
    "myTest": "concurrently \"http-server -a localhost -p 3000\" \"open http://localhost:3000/tests/index\"",
    "open": "concurrently \"http-server -a localhost -p 3000\" \"open http://localhost:3000/build/index\"",
  },
}
```

We can now run the following terminal command to startup our web server on `http://localhost:3000` serving up our `index.html` file depending on the script that is run:

```
$ npm start open $ npm start myTest
```

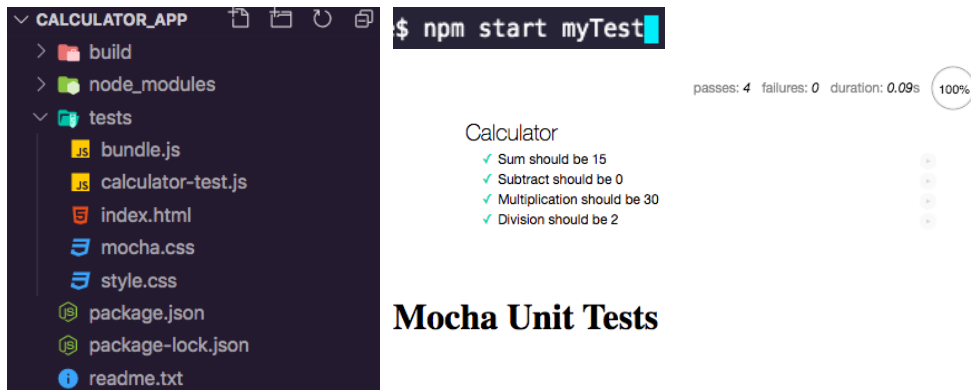
Now that we have our web server up and running we need to install one more package which would allow us to convert our Node.js code into a JavaScript file that we can use in our `index.html` file (i.e. in the browser) which is ES5 compatible i.e. browser's JavaScript Engine compatible. This will remove the error we was having with the `require` keyword. This package is called `Browserify` which we need to install globally using the `sudo` command:

```
$ sudo npm install -g browserify
```

Once this has been installed we would run the following command:

```
$ browserify calculator-test.js -o bundle.js
```

This will create a new bundle.js file that is browser compatible i.e. converts our node.js code into ES5 JavaScript that is compatible with browsers. We can now update our index.html file to reference this bundle.js file to remove the require error we had with the calculator-test.js file.



Running the web server to serve the index.html file from our tests directory should now display our TDD Mocha and Chai tests in the Browser UI as seen in the above example.

Important Note: You may run into some issues whereby you close Visual Studio code but the server is still running in the background so that the next time you come into VS code and run the script you would end up running into some errors that the port is in use. To handle this situation we can run the following commands:

This command will provide information to the PID number that is using the port. We can use this PID number to terminate it and free up the port. In the second command replace <PID> with the number from the previous step to kill the process.

```
$ sudo lsof -i :3000
```

```
$ kill -QUIT <PID>
```

END