Extra Notes - NodeJS - Testing - Mocha

- Dr Nick Hayward

A brief outline of testing Node.js apps with Mocha.

Contents

Intro We can use testing within our Node.js apps to check and verify that logic and functionality is actually working as expected.

For example, abstract and test a function to correctly fetch a user id, name &c. from a database.

Mocha testing We can start by creating a simple app, which can then be used to demonstrate testing options for Node.js.

A basic app directory, `node-testing` , and a sub-directory for some test custom modules, e.g. `utilities` .

In a file, `utilities.js` , we can add a sample function, which we'll then use for basic testing.

```
// basic function for testing
module.exports.adder = (a, b) => a + b;
```

Mocha is framework we can use to configure our test suite for such Node.js apps. Further information,

- Mocha JS Testing

Install Mocha,

```
npm install mocha --save-dev
```

The flag `--save-dev` will not be installed for production apps, Heroku &c.

Then we can create a test file for Mocha, e.g. `/utilities/utilities.test.js` . Mocha will look in a directory for files with this type of filename pattern, i.e. with `..test.js` .

Test case for Mocha Then we can add a test case for Mocha. e.g.

```
const utils = require('./utilities');

// behaviour driven development (bdd) - describe what you want the test to do...
it('should add some numbers', () => {
  var total = utils.adder(22, 11);
});
```

`it()` is a function provided by Mocha. The first parameter is a description, which should use `it` as part of the test description - known as **behaviour driven development**.

Then, the second parameter is a function for the testing itself. We can create a simple variable to help run and test the function `adder()` in the utilities.js file.

In the `package.json` file, we need to modify the `test` script to run Mocha and any available test files, e.g.

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

We're now calling Mocha, which checks every directory for any file that has a matching ending of `.test.js`. i.e. our test files for a node.js app.

We can run our initial test with Mocha, which is executed from the command specified in `package.json`, e.g.

```
npm test
```

Mocha will run the test we created in the `/utilities/utilities.test.js` file for the required `/utilities/utilities.js`, and output some comments for the test to the command line, including whether the test has passed, how long the test took to run, and the `it()` test description.

Add error checking to intial test cases In addition to simply checking whether a function runs correctly, we can also add some useful error checking and reporting to a test, e.g.

```
it('should add some numbers', () => {
  var total = utils.adder(22, 11);
  // error check and report
  if (total !== 33) {
    throw new Error(`result should have been 33, and not ${total}`);
  }
});
```

We can check this error handling and reporting by slightly modifying the original `adder()` function to deliberately create an error with the add calculation, perhaps adding more than the expected two numbers.

If we then run Mocha for this test, we'll get a series of debug messages, and the returned error message. Very useful for debugging problems part of a Node.js app.

Auto re-run tests for changes We can use `nodemon` to help monitor and re-run `npm test`, e.g.

```
nodemon --exec "npm test"
```

We can also update our app's `package.json` file to include a custom script, which will make it less verbose to run this monitored test script, e.g.

```
...
"scripts": {
  "test": "mocha **/*.test.js",
  "test-mon": "nodemon --exec \"npm test\""
}
...
```

This means we can now run monitored tests using the `npm run test-mon` command. This allows us to run a custom script.

n.b. we use double quotes for the `"npm test"` command to make sure this will work on most OSs, including Windows.

References

- Mocha JS Testing