**JEST TESTING @ 14 June 2019**

npm -I -y

npm install jest

create index.js in root

create index.test.js in root

Testing – Assertions

Add to index.js

function add(a, b) {

}

module.exports = {

add

}

Add to index.test.js

const {

add

} = require('./index')

const assert = require('assert')

the assertion follows this convention

assert.equal(actualvalue , expectedvalue)

add to bottom index.test.js

assert.equal(add(1, 5), 6)

console.log("All tests successul")

Run the test by going “node index.test.js”

Test driven development – run the first few tests with the simplest parameters you can possibly write

Get the tests to cater for the exception

Add tests to index.test.js

const {

add

} = require('./index')

const assert = require('assert')

// test numeric input

console.log("Test 1: add (2,5) is equal to 7.")

assert.equal(add(1, 5), 6)

console.log("Test 1 successful.")

//test string parameters

// this only works when the excption fails

// explicitly testing does it throw an exception when I pass in a string

console.log("Test 2: add ('fsddsf', 5) throws an exception.")

assert.throws(() => {

add('fsddsf', 5)

}) // when run test without the throw exception, this will cause an error message "Missing expected Exception"

console.log("Test 2 successful.")

Writing individual tests is not helpful. Use JEST framework.

No need to require Jest in the codebase. You run in the terminal line.

Reinstall jest globally npm install -g jest

* Automatically picks up the test file because of the file naming convention

Group tests together by wrapping all the test with a describe statement

describe("add()", () => {

test("Test 1: add (2,5) is equal to 7.", () => {

assert.equal(add(1, 5), 6)

})

test("Test 2: add ('fsddsf', 5) throws an exception.", () => {

assert.throws(() => {

add('fsddsf', 5)

})

})})

Npx could make testing faster with compilation code.

Jest comes with own assertion libraries.

IN index.test.jsComment out assert

// const assert = require('assert')

Changes the assertions in index.test.js to

describe("add()", () => {

test("Test 1: add (2,5) is equal to 7.", () => {

expect(add(1, 5)).toBe(6)

})

//test string parameters

// this only works when the exception fails

// explicitly testing does it throw an exception when I pass in a string

test("Test 2: add ('fsddsf', 5) throws an exception.", () => {

expect(() => {

add('fsddsf', 5)

}).toThrow()

})

})

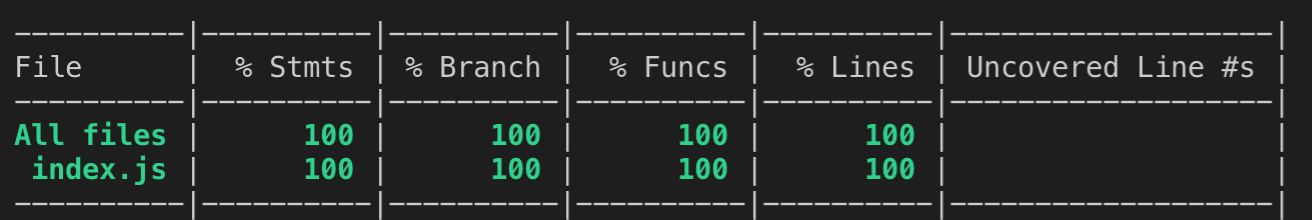
**Code Coverage**

How much the codebase is covered by automated testing

If in development, code coverage will not be in 100% because some functionalities in beta or in complete suite.

Generally the code coverage should be above 90%. 100% is ideal.

Run Jest –coverage in terminal to tell you what the code coverage is



Add a subtract function to index.js

function subtract(a, b) {

if (typeof a != 'number' || typeof b != 'number') {

throw "Numbers only"

} else {

return a - b

}

}

Add to subtract at the bottom of index.js

module.exports = {

add,

subtract

}

Add to top of index.test.js

const {

subtract

} = require('./index')

Note: Having achieved 100% code coverage does not mean that you have written all the tests that could be written (you have tested all the corner cases)

Code coverage helps you to see whether there is any area of the code you haven’t tested yet.

Check individual statements, branches to ensure everything is tested

Jest originally written for React

**Book-starter tests**

Create jest.config.js at root

Paste in the following

module.exports = {

testEnvironment: 'node'

}

Create a folder for our tests > test

Create file in test older author\_controller.test.js

In Author\_controller.test.js, add the test

const AuthorController = require('../controllers/author\_controller')

describe("AuthorController", () => {

describe("index()", () => {

test("calls res.render", async () => {

const res = {

render: jest.fn()

}

await AuthorController.index(null, res)

expect(res.render).toBeCalled()

})

})

})

Need to add mongoose database connection for the testing not to timeout

const AuthorController = require('../controllers/author\_controller')

const mongoose = require('mongoose')

describe("AuthorController", () => {

describe("index()", () => {

test("calls res.render", async () => {

mongoose.connect("mongodb://localhost/test\_books\_r\_us", {

useNewUrlParser: true

})

const res = {

}

await AuthorController.index(null, res)

expect(res.render).toBeCalled()

mongoose.connection.close()

})

})

})

Run jest –detectOpenHandles (for async functionalities)

Note: previously the test were timing out because there wasn’t a connection to mongoDB when running the index method, which draws out books and authors from the database to display. The connection to the database needs to be closed at the end or the testing will also stall.

Open database connection at the start of the test suite and close it at the end of the testing suite.

We can manage the database connection outside for the test to make it less repetitive nad redundant (repeated open and close connections for every test suite). So to do that, we add it to the top of the file (under the require statements) in author\_controller.test.js.

beforeEach() is referred to a Setup

afterEach() is referred to a teardown

// SEtup

beforeEach(() => {

mongoose.connect("mongodb://localhost/test\_books\_r\_us", {

useNewUrlParser: true

})

})

// Tear down

afterEach(() => {

mongoose.connection.close()

})

Get rid of the database connections inside the testing suite (describe statements)

To further streamline, modify the **beforeEach** to **beforeAll** and **afterEach** to **afterAll**

This tells jest to connect to database once before any tests are run and then disconnect database once after all tests have been run.

// Set up

beforeAll(() => {

mongoose.connect("mongodb://localhost/test\_books\_r\_us", {

useNewUrlParser: true

})

})

// Tear down

afterAll(() => {

mongoose.connection.close()

})

Add model to author\_controller\_test

const AuthorModel = require('../models/author\_model')

update the testing suite as per follows (This doesn’t work yet)

describe("AuthorController", () => {

describe("index()", () => {

test("calls res.render", async () => {

const res = {

render: jest.fn() // mocking

}

AuthorModel.find = jest.fn().mockResolvedValue([])

await AuthorController.index(null, res)

expect(res.render).toBeCalled()

expect(AuthorModel.find).toBeCalled()

})

})

})

**Integration test**

Testing the app as a whole with all the components integrated together

npm i supertest

test creating a new author make sure we get the correct status code back.

In test folder, create integration folder

In integration folder, create “create\_author.test.js”

In create\_author.js, add

* Require mongoose
* Tear down, set up
* Require supertest, the app itself

const supertest = require('supertest')

const app = require('../../app')

const mongoose = require('mongoose')

beforeAll(() => {

mongoose.connect("mongodb://localhost/test\_books\_r\_us", {

useNewUrlParser: true

})

})

// Tear down

afterAll(() => {

mongoose.connection.close()

})

On app.js, at the very bottom, add

module.export = app

in app.js, comment out the app.listen

// app.listen(port, () => console.log(`Server is listening on port ${port}`))

Add this test to app.js

describe("The user creates a new author", () => {

test("POST /authors with valid req body", async () => {

const response = await supertest(app).post("/authors").send({

name: "Garrett",

bio: "my bio",

gender: "male",

}).expect(200)

})

})

Add

app.use(upload.single('image'))// for testing