TESTING WEB APIS

Frank Walsh

Web Application Development

AGENDA

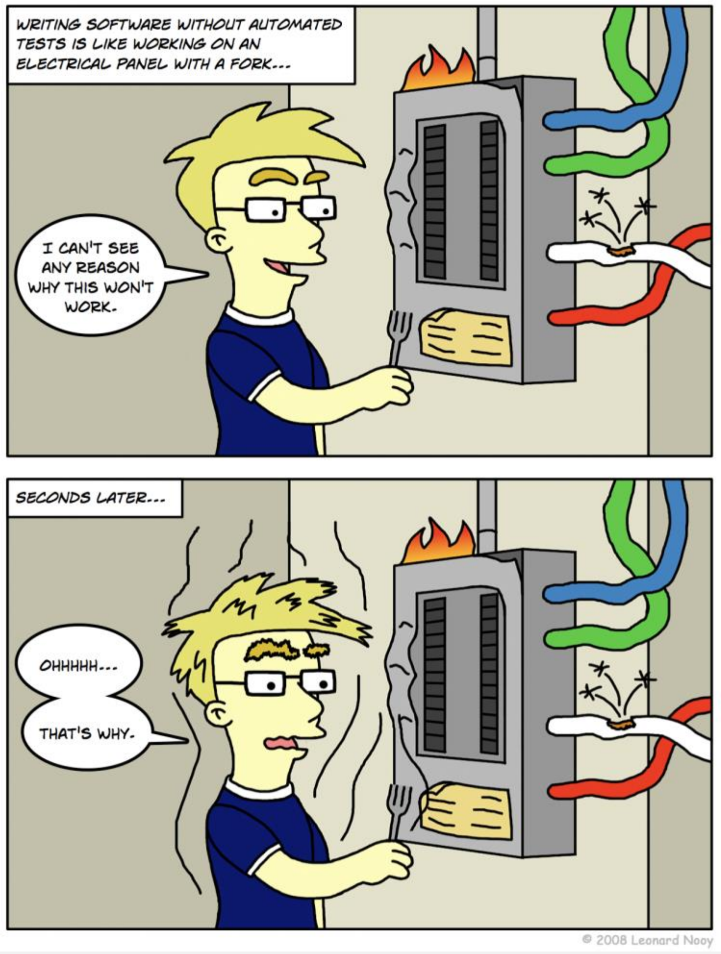
- Unit testing
- Mocha
- Should
- Sinon
- SuperTest

TESTING CATEGORIES

- **Static analysis/testing** helps to find out typos and basic syntax errors.
- **Unit testing** involves test one single unit at a time with isolation from other functionalities.
- **Integration testing** is where you will find out whether separate units of functionalities works with each other.
- **End-to-End** testing as the name suggests refers to testing the complete flow of the project from start to end.

UNIT TESTING

- Code written by developer that exercises a small, specific area of functionality.
- "Program testing can be used to show the presence of bugs, but never to show their absence!" – Dijkstra
- Up to now – Manual tests with Postman
  - Not structured
  - Not repeatable
  - Not easy
  - Not a unit test
- Usually unit testing is automated…

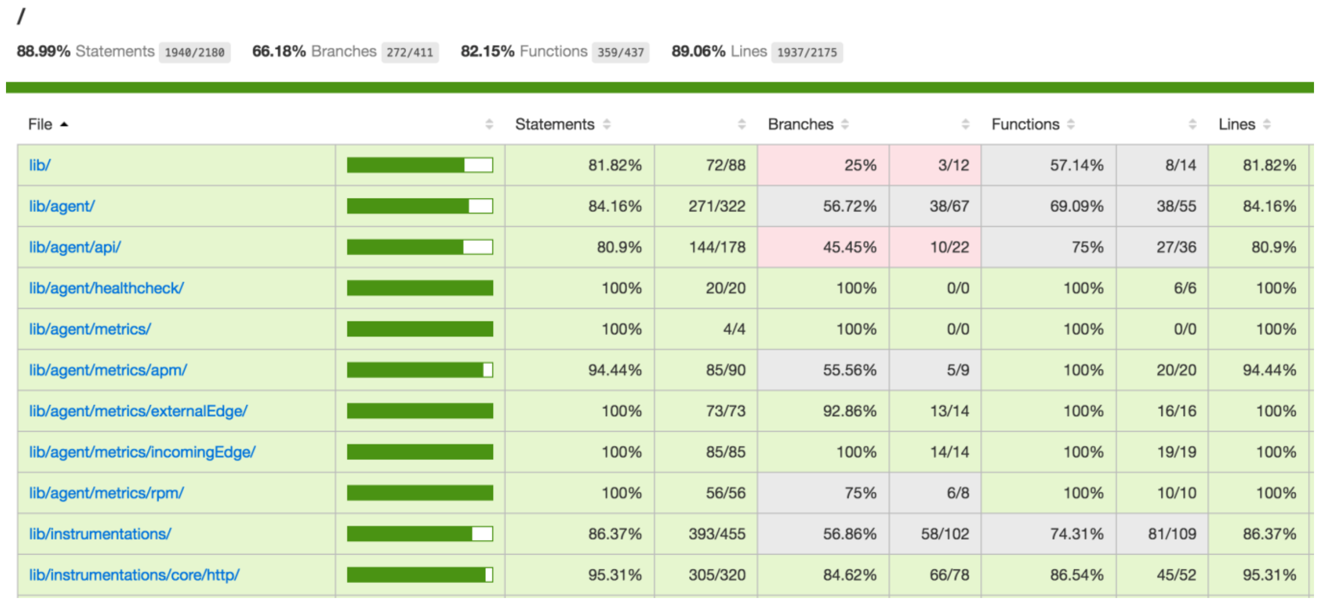

WRITING SOFTWARE WITHOUT AUTOMATED TESTS IS LIKE WORKING ON AN ELECTRICAL PANEL WITH A FORK...
I CAN'T SEE ANY REASON WHY THIS WON'T WORK.
SECONDS LATER...
OHHHHH...
THAT'S WHY.
© 2008 Leonard Nooy

UNIT TESTS

- Tests are specific pieces of code
- Tests are written by developers of the code, usually
  - Sometimes before the code is written
- Part of the code repository
  - They go where the code goes
- Use a framework
  - Junit, Jasmine, Mocha

UNIT TEST CONVENTION

- All objects and methods
- Aspire for 100% coverage
  - Although property getters/setters are sometimes omitted
- All tests should pass before commits to the repo?



/

88.99% Statements 1940/2180 **66.18%** Branches 272/411 **82.15%** Functions 359/437 **89.06%** Lines 1937/2175

File ▲		Statements		Branches		Functions		Lines
lib/		81.82%	72/88	25%	3/12	57.14%	8/14	81.82%
lib/agent/		84.16%	271/322	56.72%	38/67	69.09%	38/55	84.16%
lib/agent/api/		80.9%	144/178	45.45%	10/22	75%	27/36	80.9%
lib/agent/healthcheck/		100%	20/20	100%	0/0	100%	6/6	100%
lib/agent/metrics/		100%	4/4	100%	0/0	100%	0/0	100%
lib/agent/metrics/apm/		94.44%	85/90	55.56%	5/9	100%	20/20	94.44%
lib/agent/metrics/externalEdge/		100%	73/73	92.86%	13/14	100%	16/16	100%
lib/agent/metrics/incomingEdge/		100%	85/85	100%	14/14	100%	19/19	100%
lib/agent/metrics/rpm/		100%	56/56	75%	6/8	100%	10/10	100%
lib/instrumentations/		86.37%	393/455	56.86%	58/102	74.31%	81/109	86.37%
lib/instrumentations/core/http/		95.31%	305/320	84.62%	66/78	86.54%	45/52	95.31%

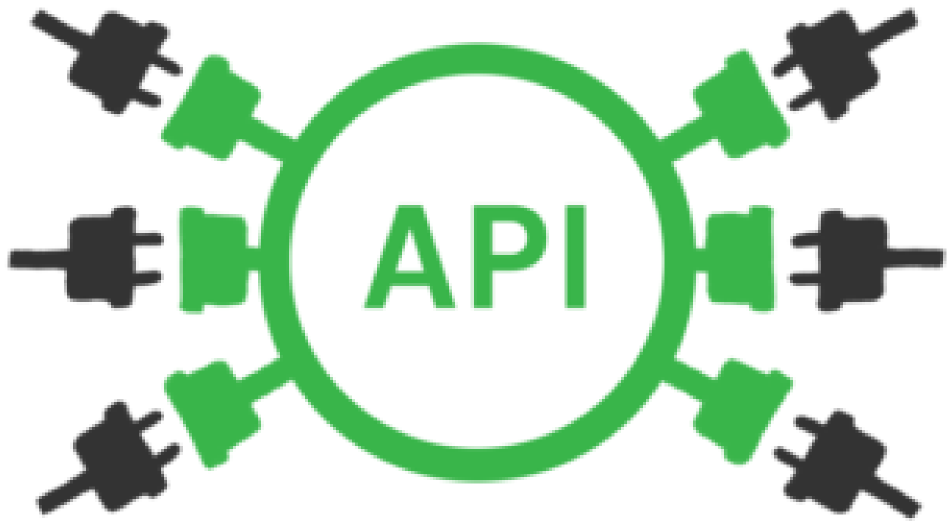
INTEGRATION TESTING

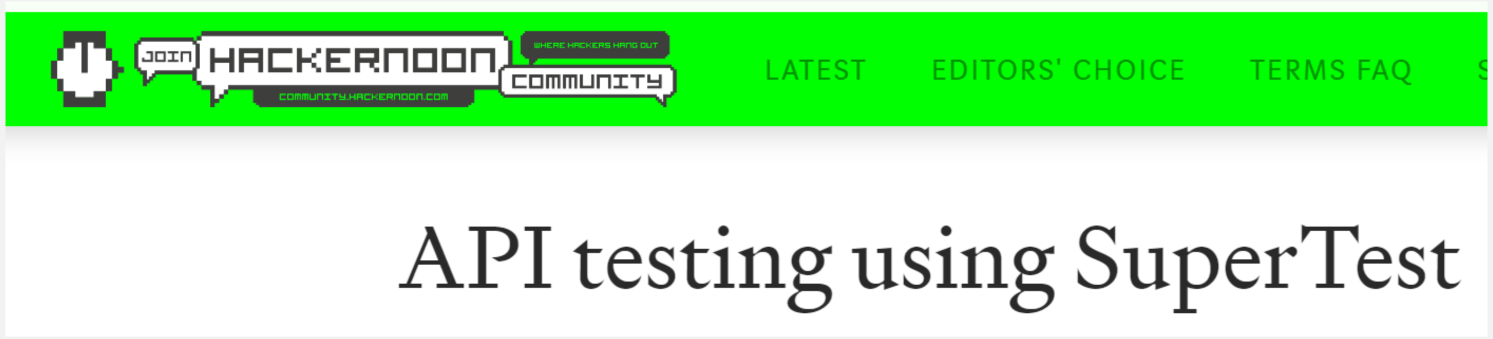
- **INTEGRATION TESTING combines** individual units in a test.
    - Test drivers and test stubs are used to assist in Integration Testing.
- Exposes faults in the interaction between integrated units
- Usually happens after unit testing.
- Both developers and independent testers perform integration testing

2 UNIT TESTS, 0 INTEGRATION TESTS
via reddit.com/r/programmerhumor

TESTING OUR API

- Is this integration or unit testing?
  - Integration testing, because you have to run a web server (locally)
  - Your Web API is an "Application boundary"
    - Requires HTTP to interact with it
  - And you've a DB/3rd party APIs going
  - So you're testing more than just your code…

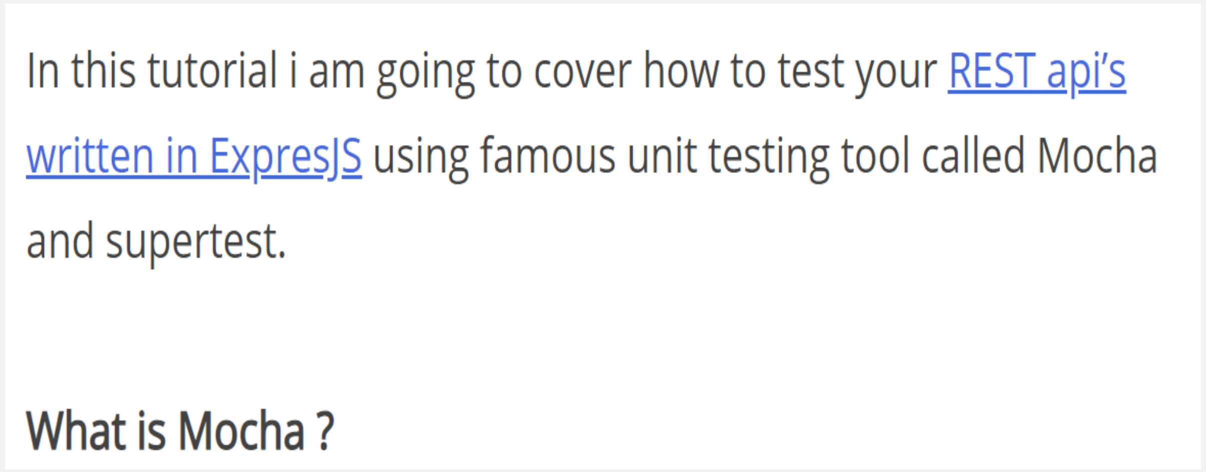




API testing using SuperTest

All You Need to Know About Integration Testing: SuperTest, Mocha, and Chai

Published Apr 14, 2017

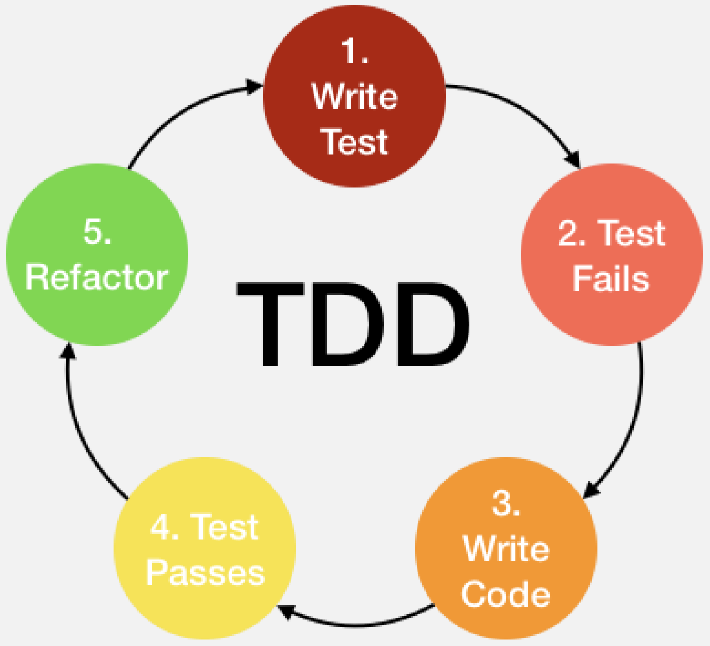


In this tutorial i am going to cover how to test your REST api's written in ExpresJS using famous unit testing tool called Mocha and supertest.

What is Mocha ?

ASIDE – TDD AND BDD

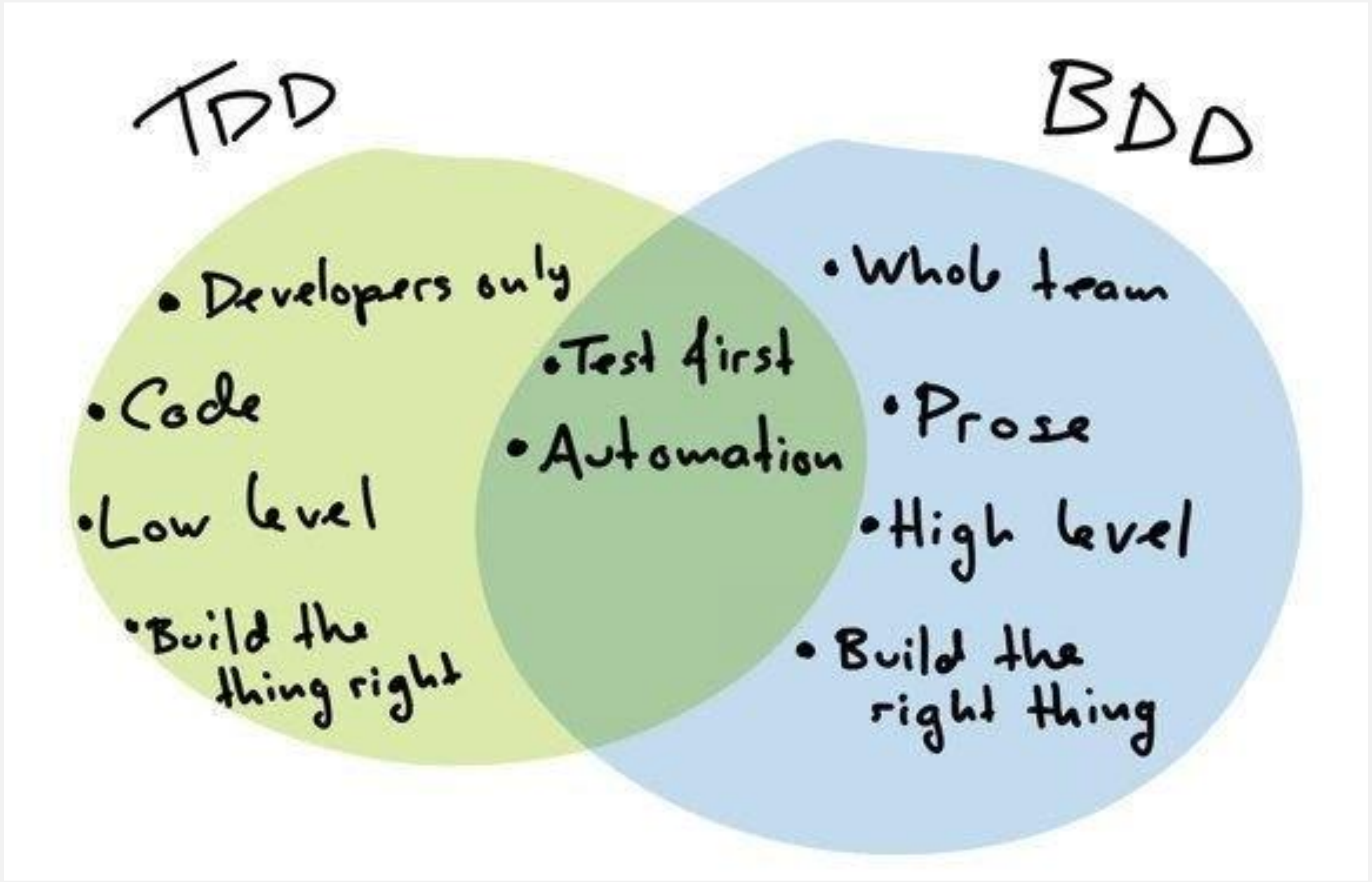
- **Test Driven Development**



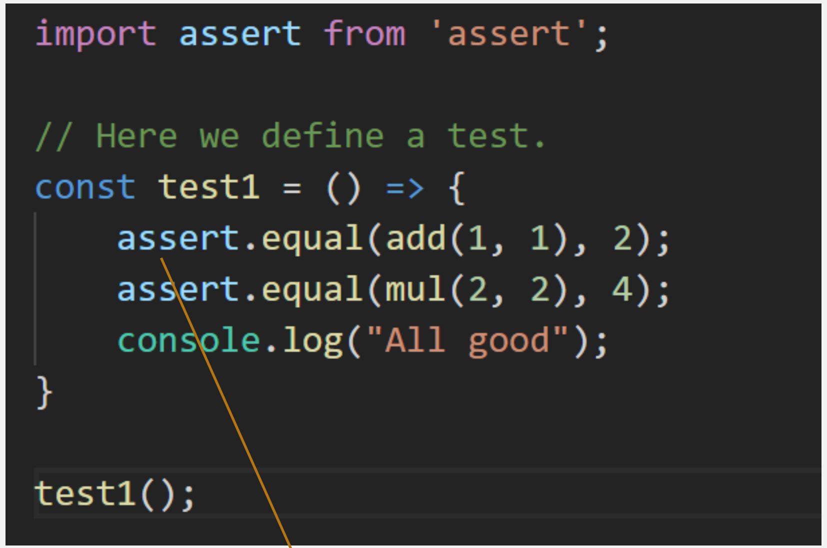
assertTheSame(user.name,'tj')

- **Behaviour Driven Development**
  - Specify desired behaviour of the unit
  - Based on requirements set by the business
  - Behavioural specification from business and developer

user.should.have.property('name', 'tj');


TDD
BDD
• Developers only
• Code
• Low level
• Build the thing right
• Test first
• Automation
• Whole team
• Prose
• High level
• Build the right thing

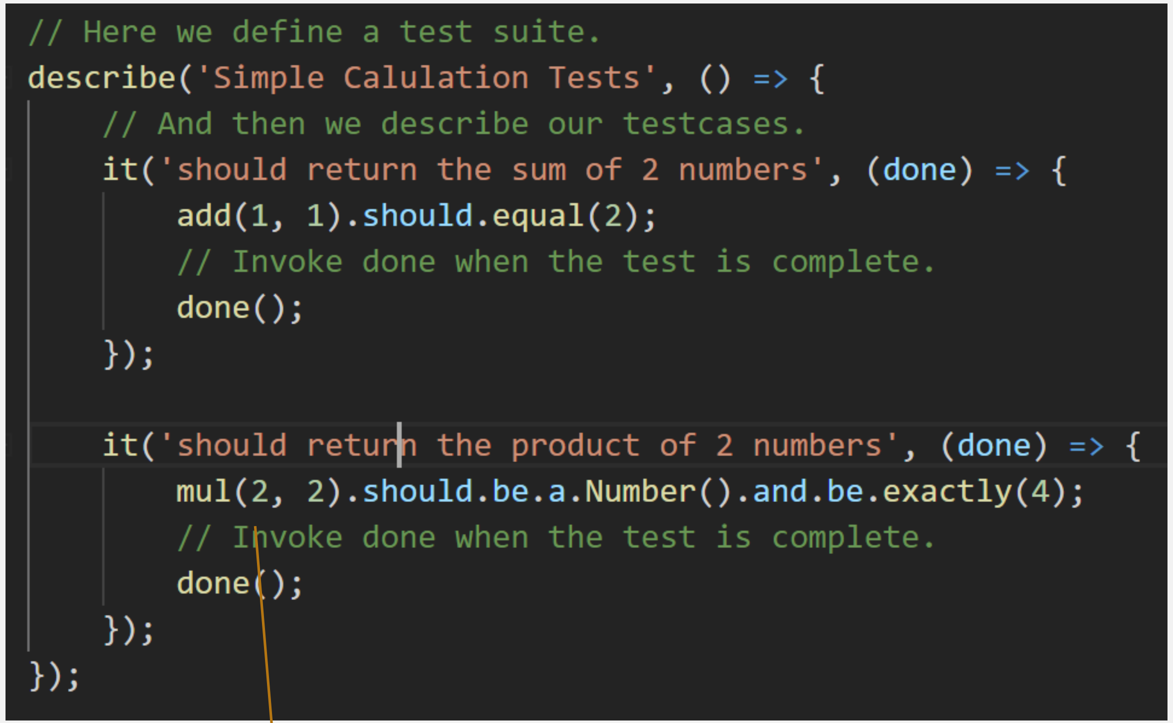
TDD-type test



```
import assert from 'assert';

// Here we define a test.
const test1 = () => {
    assert.equal(add(1, 1), 2);
    assert.equal(mul(2, 2), 4);
    console.log("All good");
}

test1();
```

"Assert add(1,1) equals 2"

BDD-type test



```
// Here we define a test suite.
describe('Simple Calulation Tests', () => {
    // And then we describe our testcases.
    it('should return the sum of 2 numbers', (done) => {
        add(1, 1).should.equal(2);
        // Invoke done when the test is complete.
        done();
    });

    it('should return the product of 2 numbers', (done) => {
        mul(2, 2).should.be.a.Number().and.be.exactly(4);
        // Invoke done when the test is complete.
        done();
    });
});
```

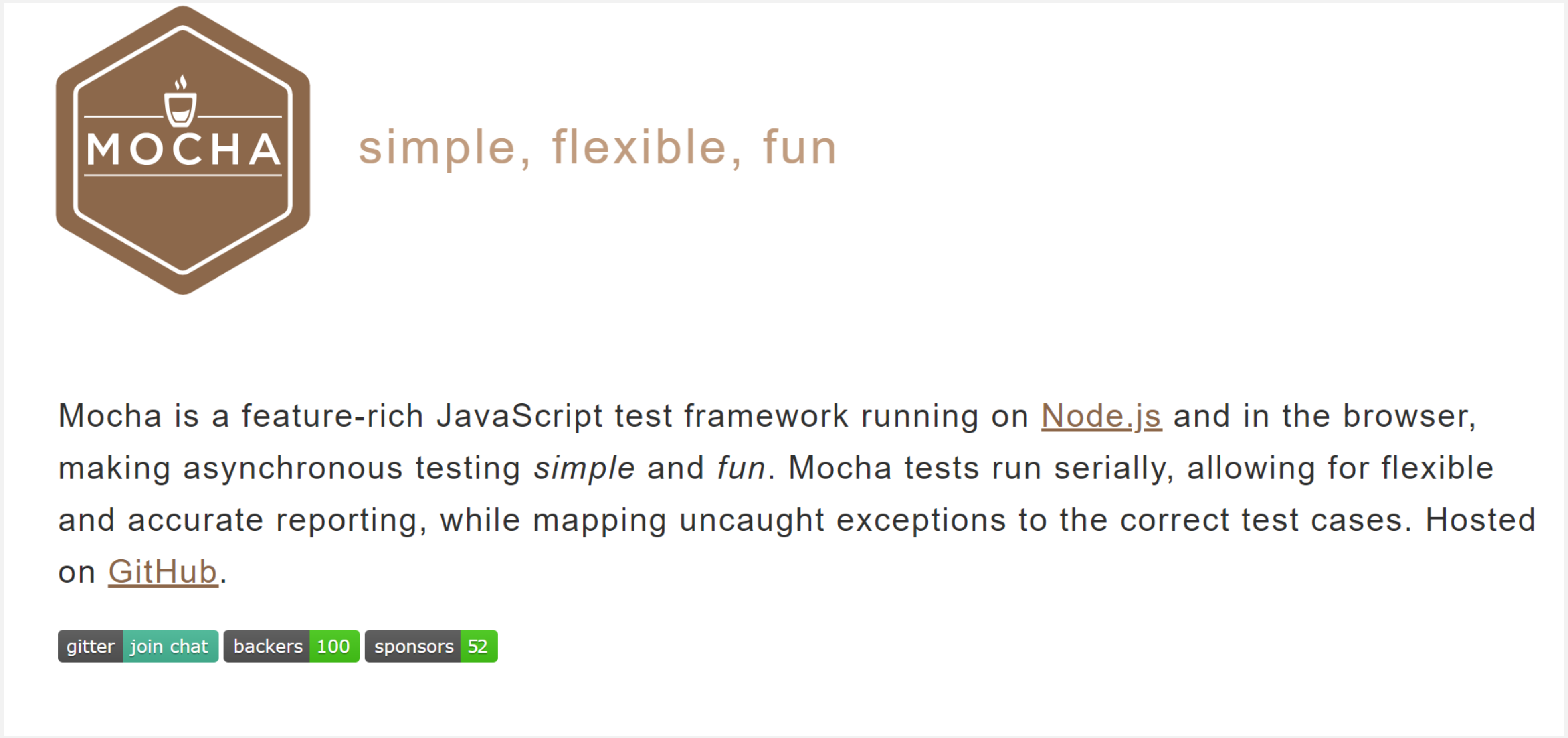
"2*2 should be a number and be exactly 4"

TESTING FRAMEWORKS, ASSERTIONS AND MOCKING

TESTING TOOLS

- **Test Frameworks**
  - Makes it easier to write tests
  - Provide hooks, test suites, test runners
  - Examples Junit, VS Team Test, PHP Unit, Mocha
- **Assertion Frameworks**
  - Perform checks and decisions
  - Examples: assert, chai.js, should.js
- **Mocking Frameworks**
  - Create mock dependencies, stubs, proxys
  - Sinon, Jmock, Mockito, Mockgoose!

Test Framework


MOCHA

simple, flexible, fun

Mocha is a feature-rich JavaScript test framework running on Node.js and in the browser, making asynchronous testing *simple* and *fun*. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Hosted on GitHub.

TEST FRAMEWORK

- Open Source framework for Javascript unit testing
  - Run in browser and server-side (e.g. node)
- Features
  - Expressive syntax
  - Can test Async code
  - Pluggable
    - Compatible with test runners such as Karma

mocha

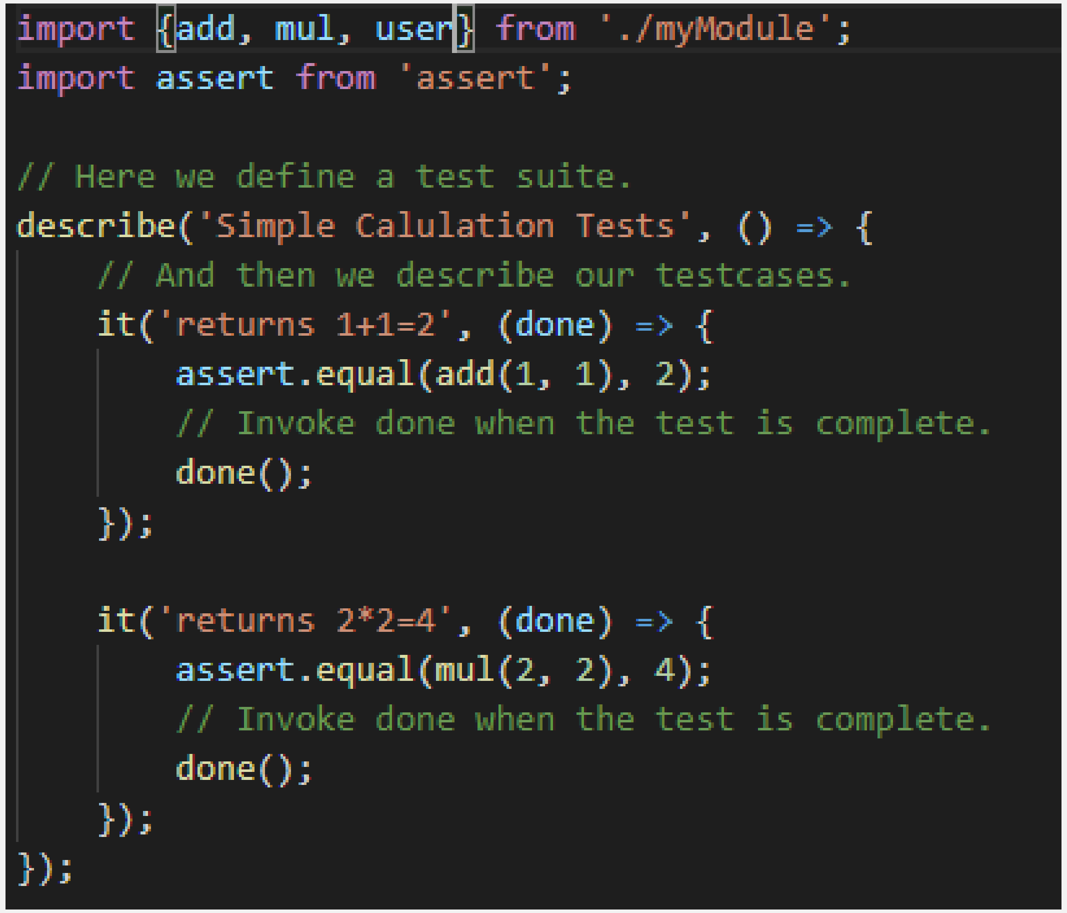
ASSERTION FRAMEWORK

- Can use the "assert" package
  - Node.js core package

1. Define a Test Suite
2. In the test suit, create test cases
3. Invoke "done()" when each test is complete.

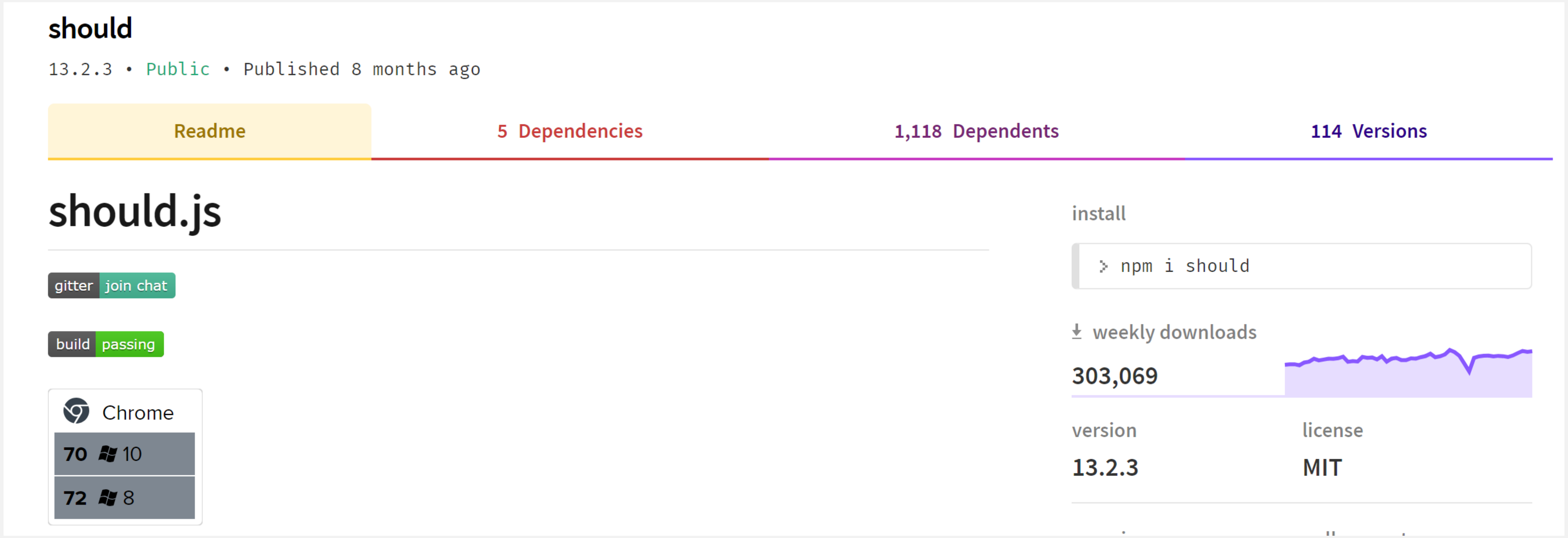
Not very expressive or high functioning

Better 3rd party options: chai, should…



```
import {add, mul, user} from './myModule';
import assert from 'assert';

// Here we define a test suite.
describe('Simple Calulation Tests', () => {
    // And then we describe our testcases.
    it('returns 1+1=2', (done) => {
        assert.equal(add(1, 1), 2);
        // Invoke done when the test is complete.
        done();
    });

    it('returns 2*2=4', (done) => {
        assert.equal(mul(2, 2), 4);
        // Invoke done when the test is complete.
        done();
    });
});
```

ASSERTION FRAMEWORKS



should

13.2.3 • Public • Published 8 months ago

Readme | 5 Dependencies | 1,118 Dependents | 114 Versions

should.js

install

```
npm i should
```

weekly downloads

303,069

version

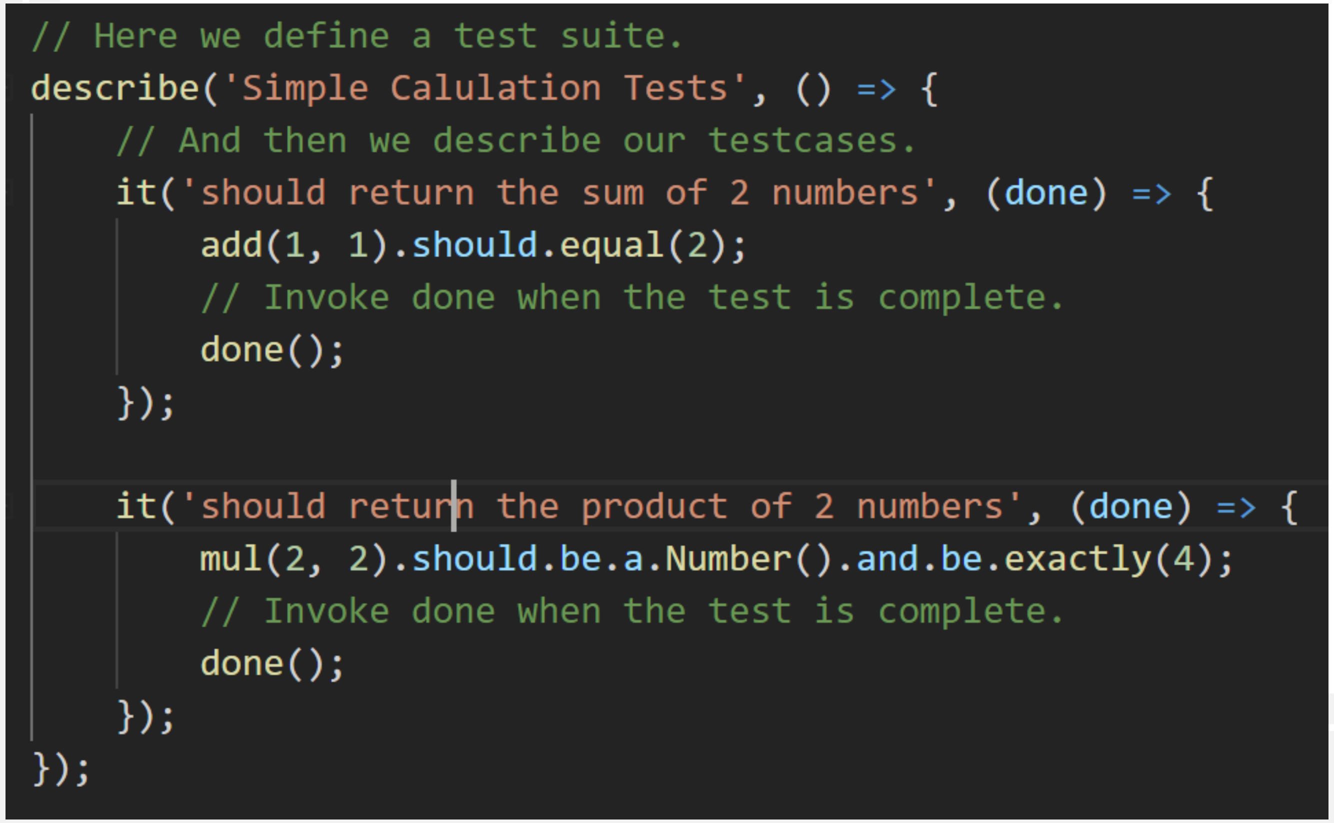
13.2.3

license

MIT

ASSERTION FRAMEWORK

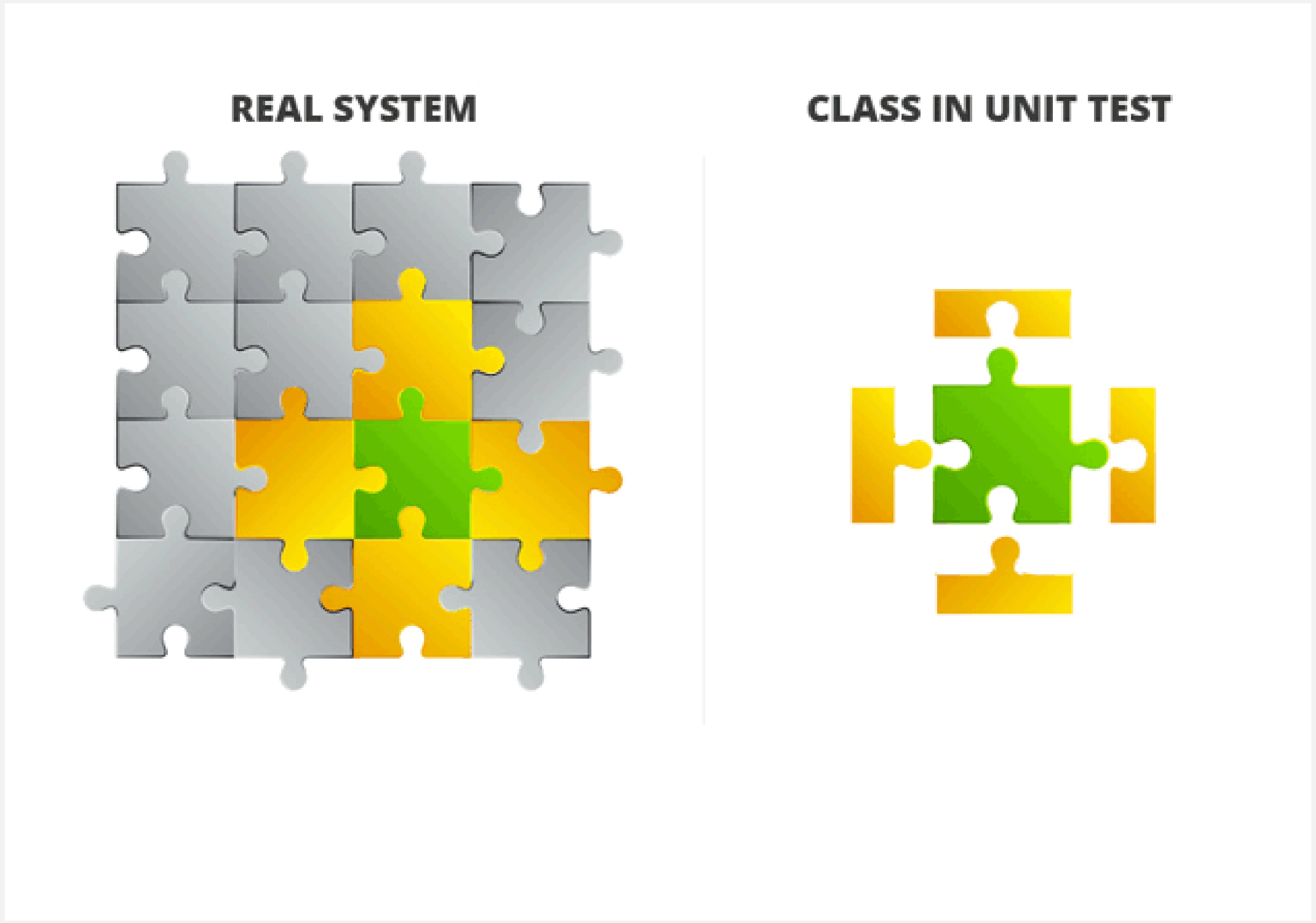
- Mocha allows you to use any assertion library you wish.
- should is an expressive, readable, framework-agnostic assertion library.
- Can use with Mocha to write cleaner, more BDD style tests
- Generates nice error messages (there's always error messages!)
- Works with Node and browsers
- Can use in asyc tests with Mocha



```javascript
// Here we define a test suite.
describe('Simple Calulation Tests', () => {
    // And then we describe our testcases.
    it('should return the sum of 2 numbers', (done) => {
        add(1, 1).should.equal(2);
        // Invoke done when the test is complete.
        done();
    });

    it('should return the product of 2 numbers', (done) => {
        mul(2, 2).should.be.a.Number().and.be.exactly(4);
        // Invoke done when the test is complete.
        done();
    });
});
```

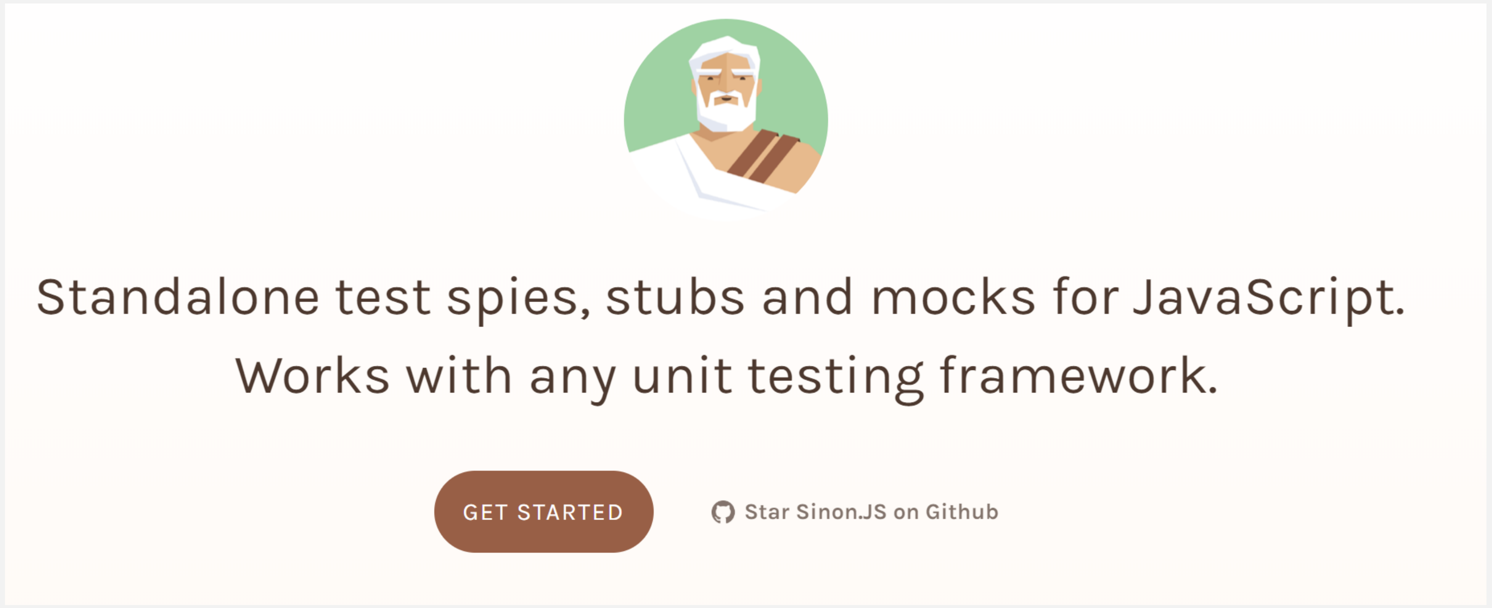
MOCKING FRAMEWORK

- What if your code has methods that use/integrate a DB?
- What if your code uses an API that's not ready
- Can use mocking and stubs to override/replace/mutate aspects of the code to allow you to test various scenarios in an isolated fashon
- Examples: Proxyquire, Sinon



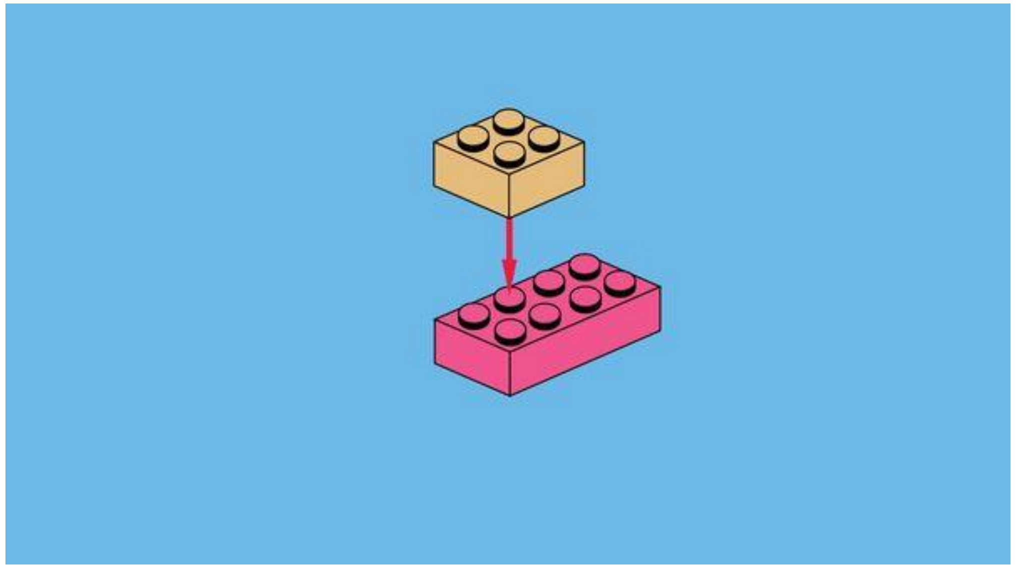
SINON MOCKING FRAMEWORK

- Can use sinon to create "stub" that can respond with fake data
- Allows isolation of target code.
- Why you might need this?
  - a js module may have public functions that need to be tested. Functions can make a call out to another service or to a database.
  - E.g. User Mongoose model…


Standalone test spies, stubs and mocks for JavaScript.
Works with any unit testing framework.

HOW IT WORKS…

- Provide description of test using **“describe”**
- Use **“it”** to define several test cases into it.
  - “it” provides a “done” function, used to indicate the end of test case.



GETTING MOCHA ETC.

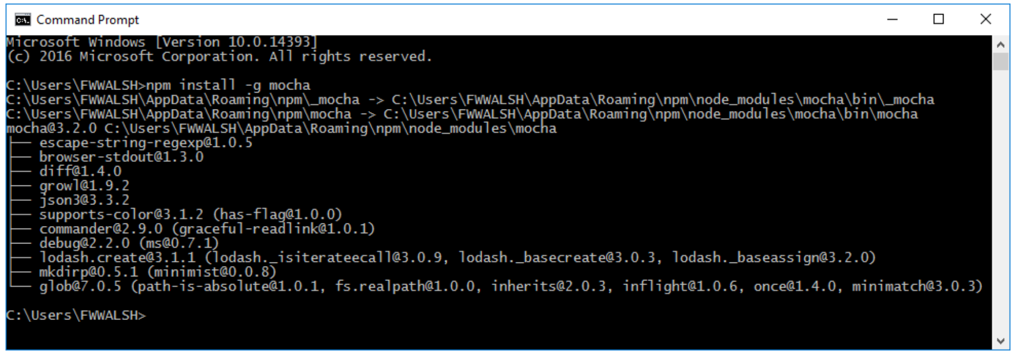
- Use NPM and install Mocha, Should and Supertest

npm install --save-dev mocha

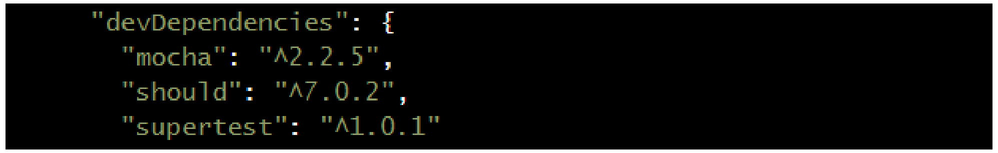
npm install --save-dev should

npm install --save-dev sinon sinon-test

npm install --save-dev supertest



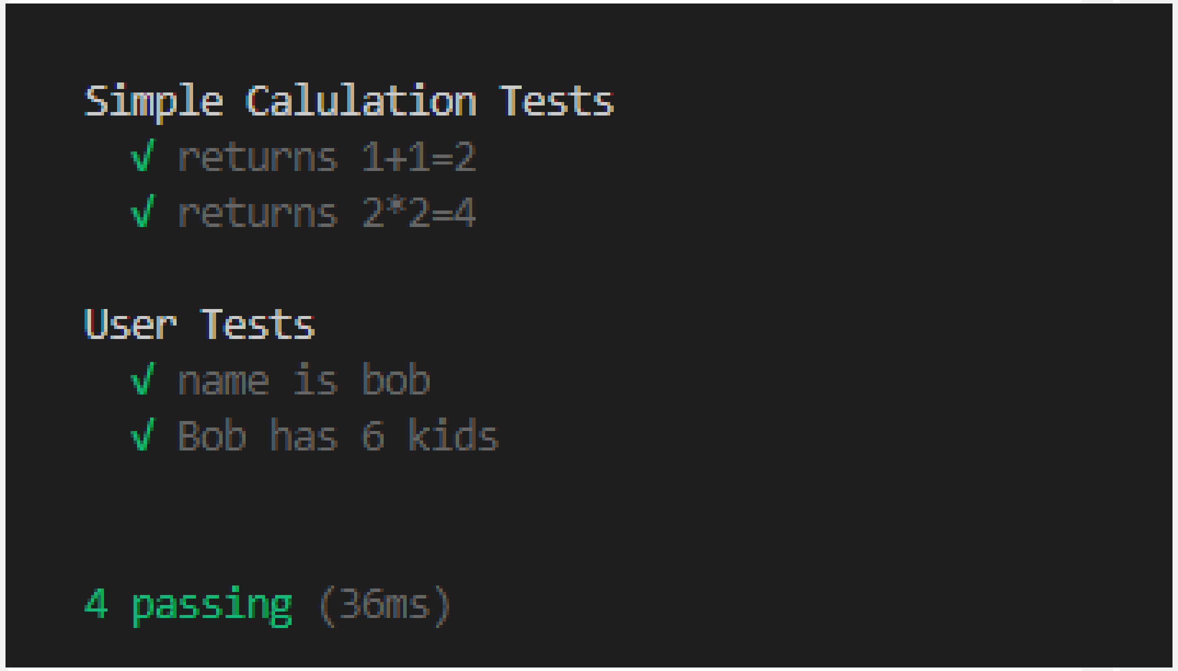
```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\FWWALSH>npm install -g mocha
C:\Users\FWWALSH\AppData\Roaming\npm\_mocha -> C:\Users\FWWALSH\AppData\Roaming\npm\node_modules\mocha\bin\_mocha
C:\Users\FWWALSH\AppData\Roaming\npm\mocha -> C:\Users\FWWALSH\AppData\Roaming\npm\node_modules\mocha\bin\mocha
mocha@3.2.0 C:\Users\FWWALSH\AppData\Roaming\npm\node_modules\mocha
├── escape-string-regexp@1.0.5
├── browser-stdout@1.3.0
├── diff@1.4.0
├── growl@1.9.2
├── json3@3.3.2
├── supports-color@3.1.2 (has-flag@1.0.0)
├── commander@2.9.0 (graceful-readlink@1.0.1)
├── debug@2.2.0 (ms@0.7.1)
├── lodash.create@3.1.1 (lodash._isiterateecall@3.0.9, lodash._basecreate@3.0.3, lodash._baseassign@3.2.0)
├── mkdirp@0.5.1 (minimist@0.0.8)
└── glob@7.0.5 (path-is-absolute@1.0.1, fs.realpath@1.0.0, inherits@2.0.3, inflight@1.0.6, once@1.4.0, minimatch@3.0.3)

C:\Users\FWWALSH>
```



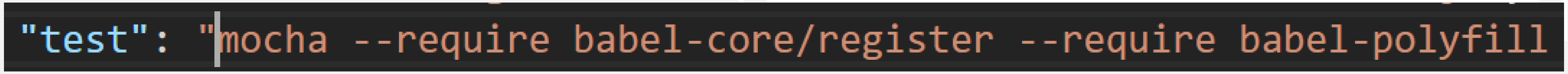
```
    "devDependencies": {
      "mocha": "^2.2.5",
      "should": "^7.0.2",
      "supertest": "^1.0.1"
```

RUNNING THE TEST MANUALLY

- From command prompt, type
  - npx mocha
- As we're ES6 and need to transpile, easier to create a script entry in **package.json**
- Can associate tests with node project by including new script property
- Set up a test script in package.json
  - Then use **npm run test** at the command line



```
Simple Calulation Tests
  √ returns 1+1=2
  √ returns 2*2=4

User Tests
  √ name is bob
  √ Bob has 6 kids


4 passing (36ms)
```



```
"test": "mocha --require babel-core/register --require babel-polyfill
```

EXAMPLE….

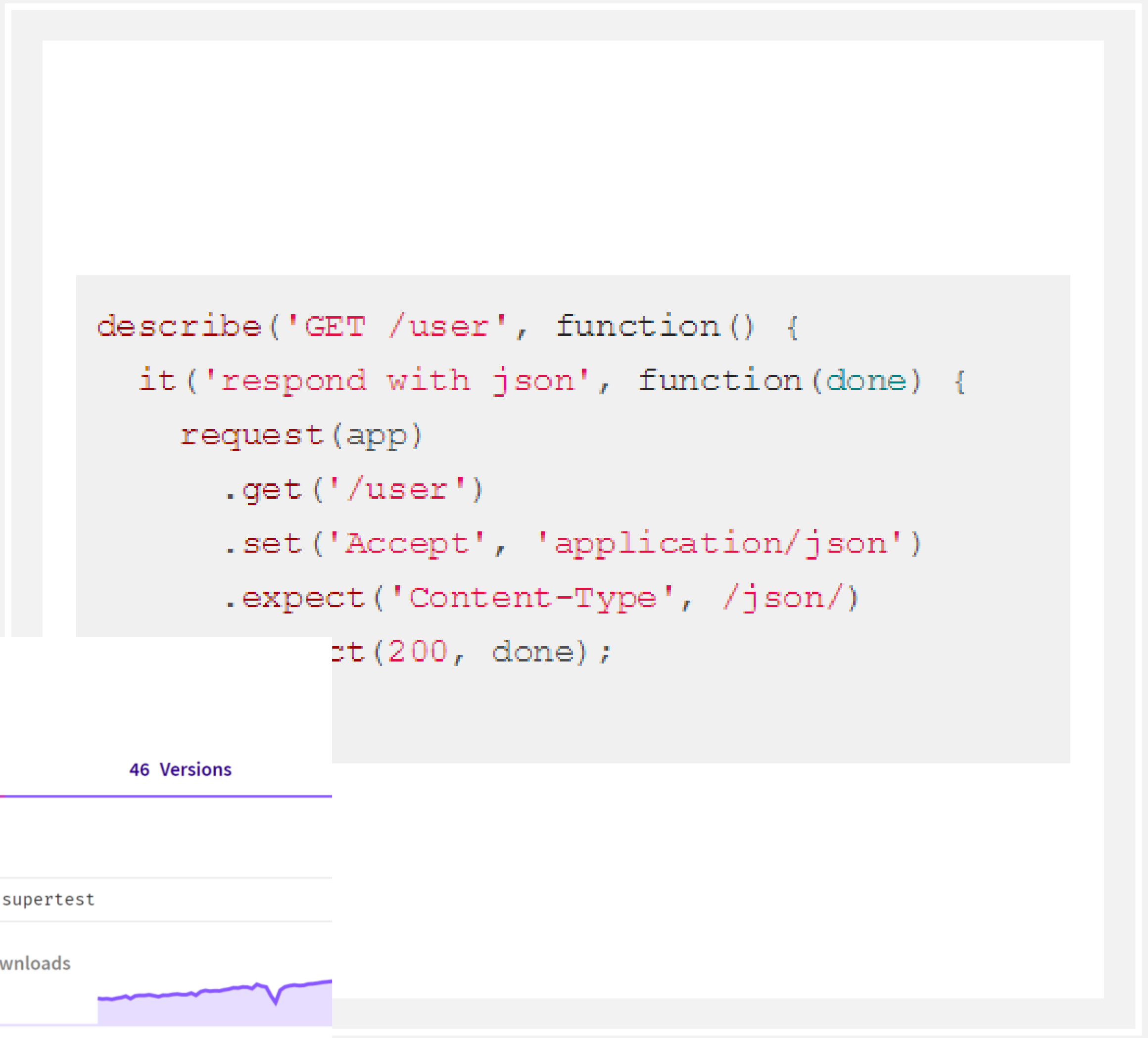
TESTING AN API (INTEGRATION TESTING)

API TESTS

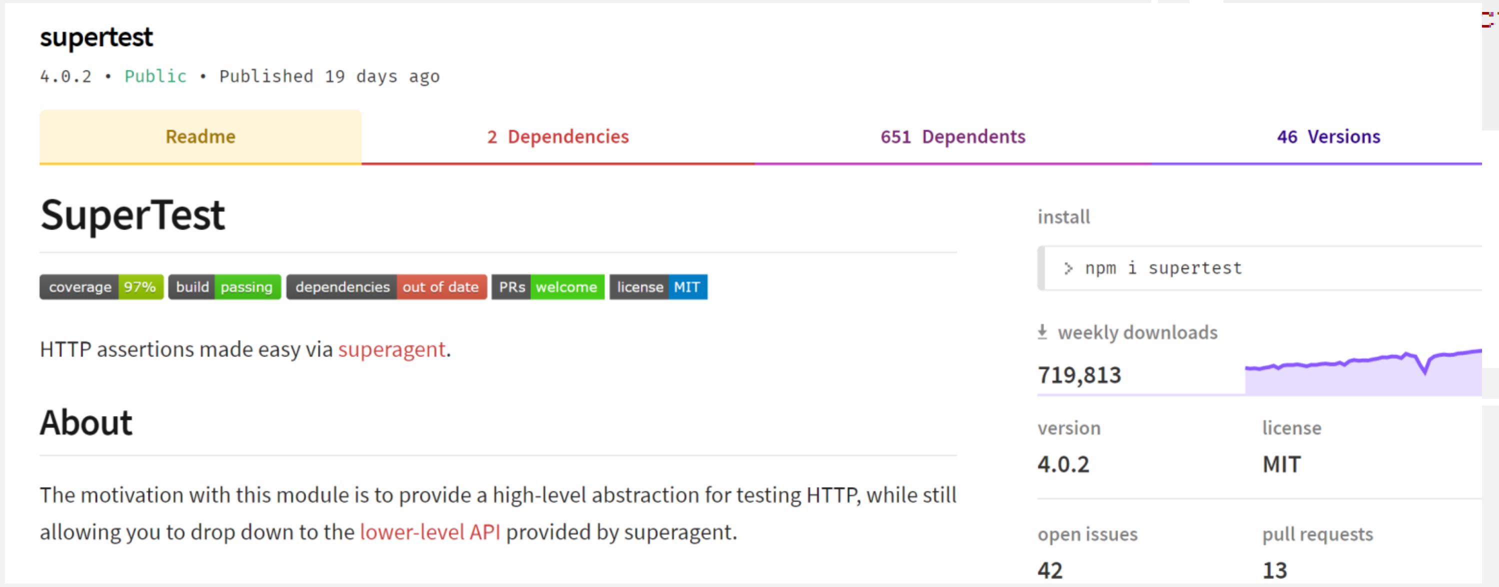
- Testing with postman not ideal
  - Cannot formally specify test suites
  - Cannot integrate into testing
- Fine for development cycle
- Need a more structured method of testing APIs
  - Regression (all routes should be checked before commit)
  - Use HTTP requests to test express app

TESTING OVER HTTP WITH **SUPERTEST**

- Provide a high-level abstraction for testing HTTP
- Works with any test framework
  - In our case, Mocha



```
describe('GET /user', function() {
  it('respond with json', function(done) {
    request(app)
      .get('/user')
      .set('Accept', 'application/json')
      .expect('Content-Type', /json/)
                ct(200, done);
```



supertest

4.0.2 • Public • Published 19 days ago

Readme | 2 Dependencies | 651 Dependents | 46 Versions

SuperTest

coverage 97% | build passing | dependencies out of date | PRs welcome | license MIT

HTTP assertions made easy via superagent.

About

The motivation with this module is to provide a high-level abstraction for testing HTTP, while still allowing you to drop down to the lower-level API provided by superagent.

install

```
> npm i supertest
```

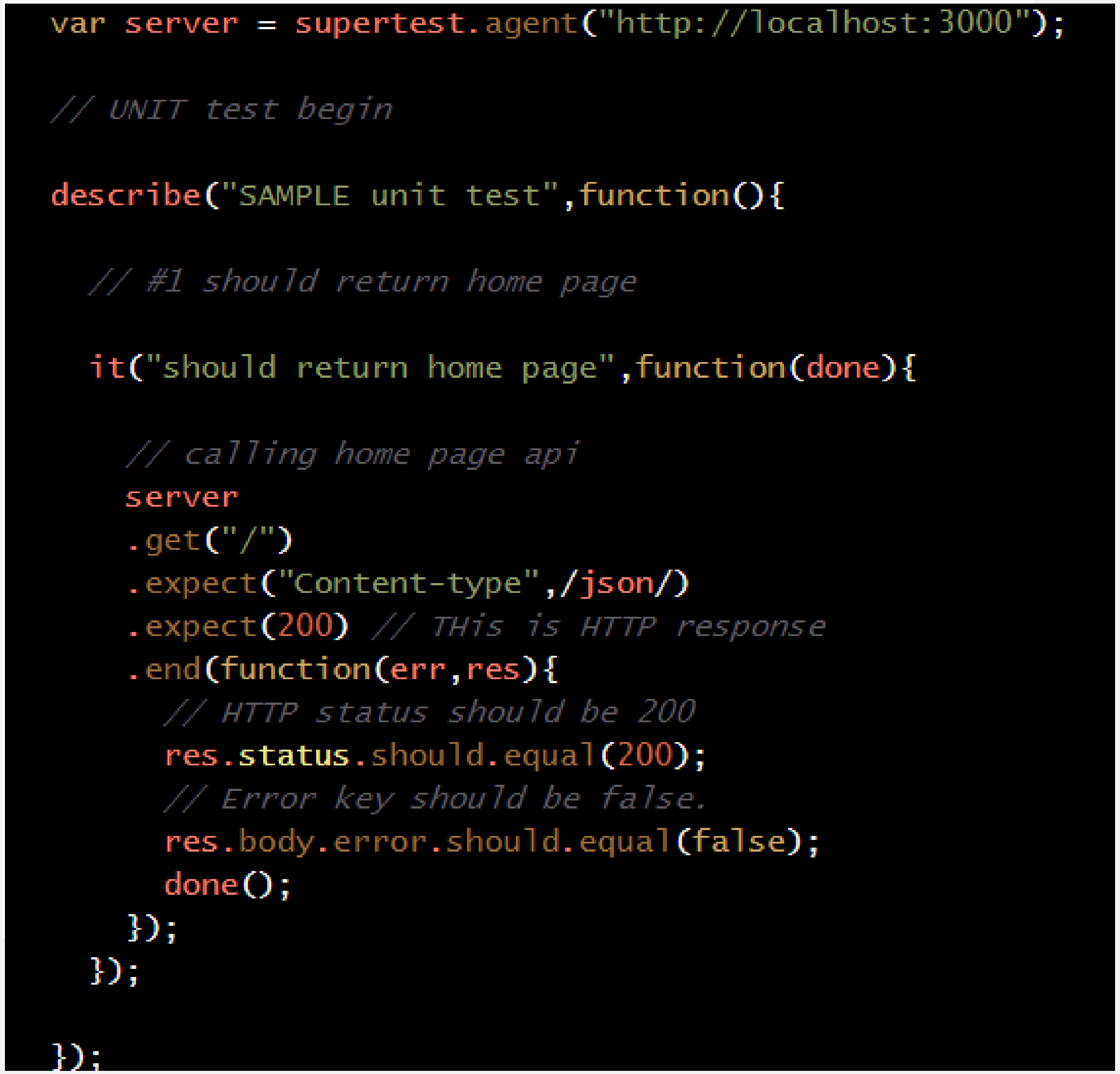
weekly downloads

719,813

version	license
4.0.2	MIT
open issues	pull requests
42	13

EXAMPLE – STATIC HOME PAGE TEST

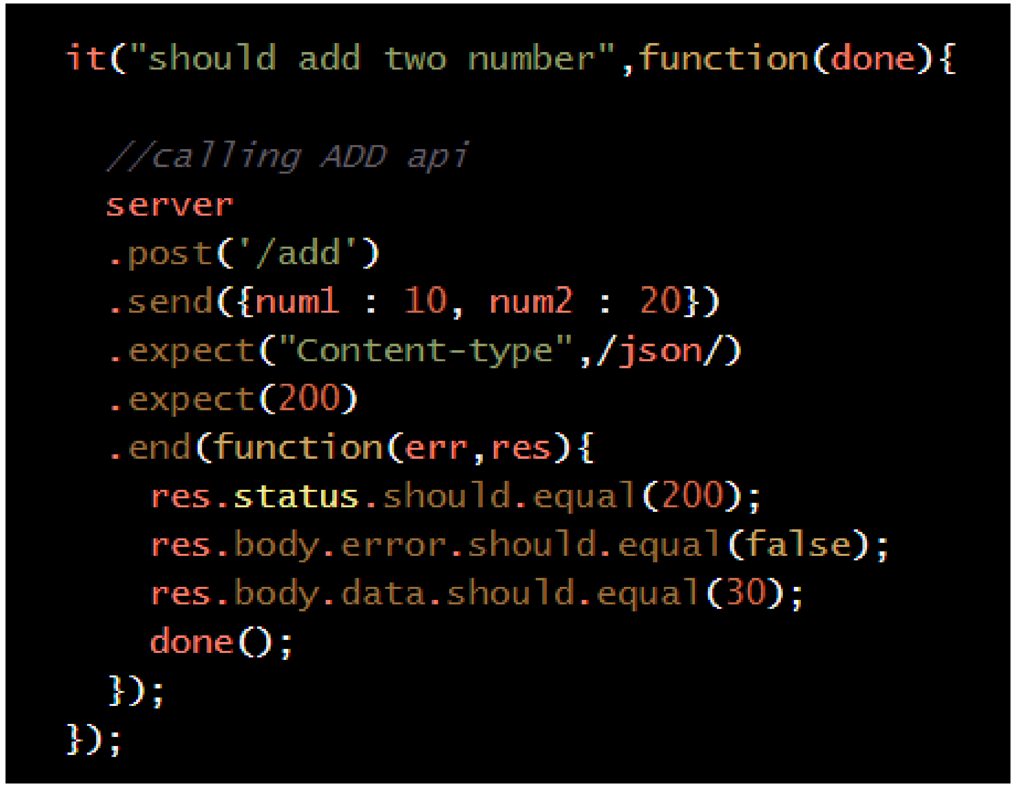
- Supertest.agent(…) returns server object constructed with test URL
- "describe" takes test name and test function
- "it" specifies the unit test that uses the server object to
  - Do a HTTP GET on the URL.
  - Define what's expected (e.g. content type, status
- Use "should" to check status of response object



```
var server = supertest.agent("http://localhost:3000");

// UNIT test begin

describe("SAMPLE unit test",function(){

  // #1 should return home page

  it("should return home page",function(done){

    // calling home page api
    server
    .get("/")
    .expect("Content-type",/json/)
    .expect(200) // THis is HTTP response
    .end(function(err,res){
      // HTTP status should be 200
      res.status.should.equal(200);
      // Error key should be false.
      res.body.error.should.equal(false);
      done();
    });
  });

});
```

TESTING A ROUTE

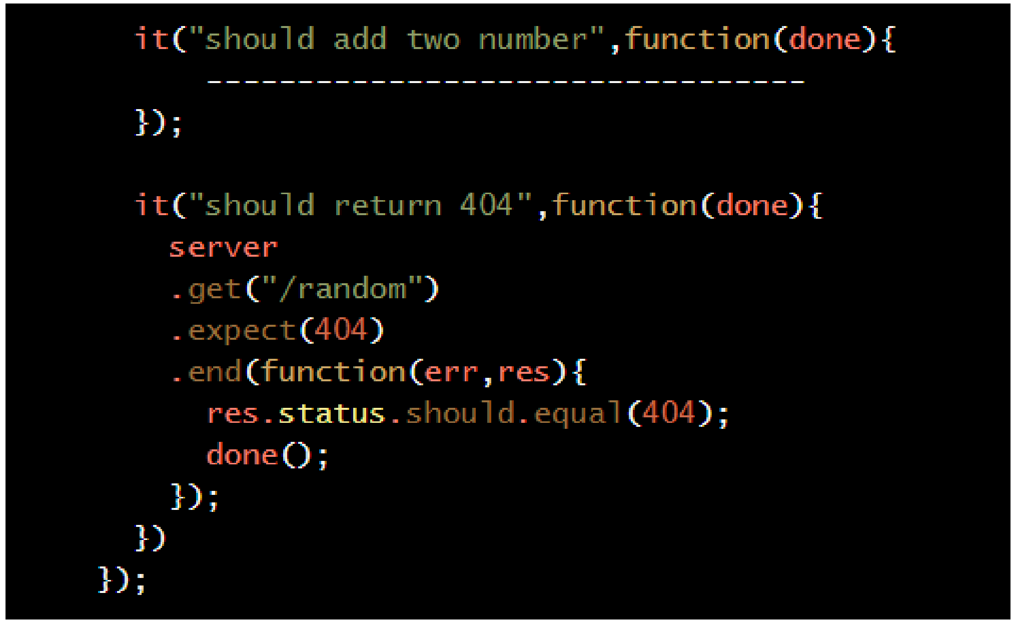
- '/add' route should add two numbers provided in HTTP body
  - Should return json response
  - Data item of body should equal sum of initial numbers
- "post" does a HTTP post on URL
- send inserts HTTP body
- Contents of reponse validated using should



```
it("should add two number",function(done){

  //calling ADD api
  server
  .post('/add')
  .send({num1 : 10, num2 : 20})
  .expect("Content-type",/json/)
  .expect(200)
  .end(function(err,res){
    res.status.should.equal(200);
    res.body.error.should.equal(false);
    res.body.data.should.equal(30);
    done();
  });
});
```

TESTING FAILURE

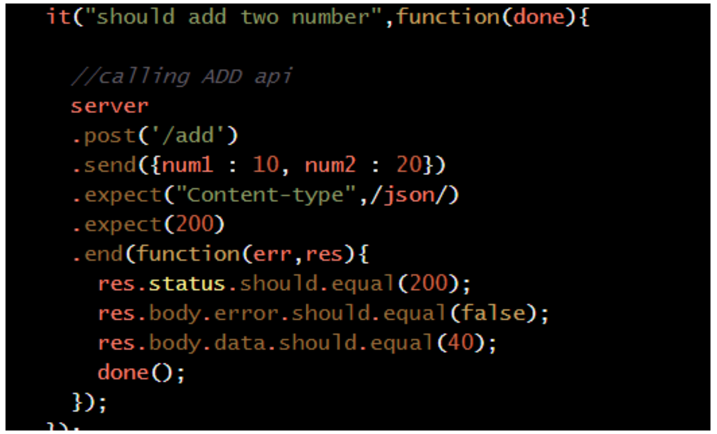
- Can test for non-existant/removed resources
  - E.g. after delete
- Check status of HTTP response is 404
- Check status of res object is also 404



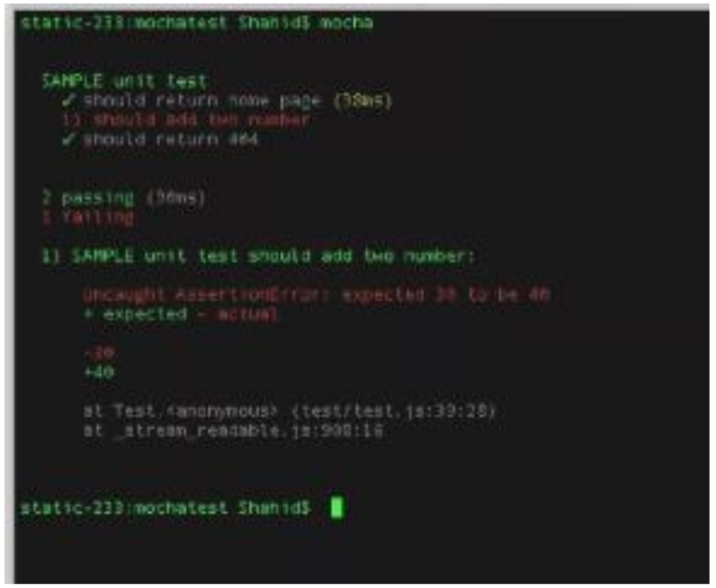
```
    it("should add two number",function(done){
      ----------------------------------
    });

    it("should return 404",function(done){
      server
      .get("/random")
      .expect(404)
      .end(function(err,res){
        res.status.should.equal(404);
        done();
      });
    })
  });
```

FAILING TEST

- Equal value of addition test is changed.
  - 40 (should be 30)
- Result is test failure
- Indicated clearly by test report.



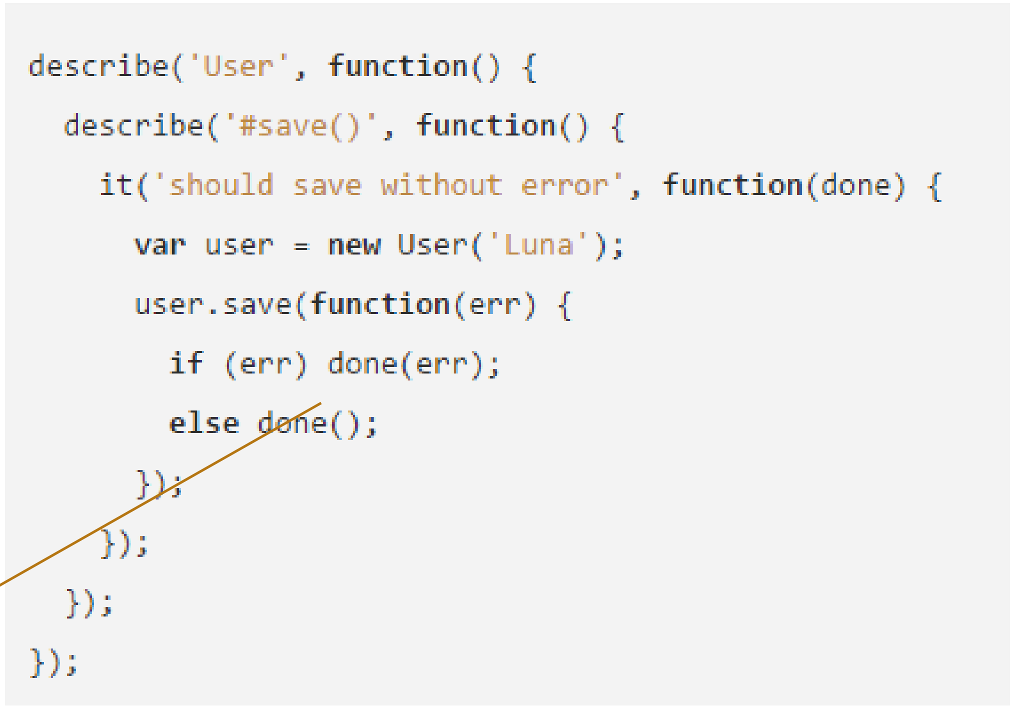
```
it("should add two number",function(done){

  //calling ADD api
  server
  .post('/add')
  .send({num1 : 10, num2 : 20})
  .expect("Content-type",/json/)
  .expect(200)
  .end(function(err,res){
    res.status.should.equal(200);
    res.body.error.should.equal(false);
    res.body.data.should.equal(40);
    done();
  });
```



```
static-233:mochatest Shahid$ mocha


  SAMPLE unit test
    ✓ should return home page (38ms)
    1) should add two number
    ✓ should return 404


  2 passing (56ms)
  1 failing

  1) SAMPLE unit test should add two number:

      Uncaught AssertionError: expected 30 to be 40
      + expected - actual

      -30
      +40

      at Test.<anonymous> (test/test.js:33:28)
      at _stream_readable.js:908:16


static-233:mochatest Shahid$ 
```

ASYNCHRONOUS CODE TEST ANATOMY

- Uses the callback pattern.
- The callback (usually named done) lets Mocha know when the test is complete
- Mocha waits for this function to be called before completing the test.



```
describe('User', function() {
  describe('#save()', function() {
    it('should save without error', function(done) {
      var user = new User('Luna');
      user.save(function(err) {
        if (err) done(err);
        else done();
      });
    });
  });
});
```

“done()” called after test is complete. In this case after user.save(..) returns

IMPROVEMENTS – MOCKING THE DB

- Unit testing should only concern the unit you're testing
  - Should be independent of servers/db dependencies
- Tests should just test the unit in question
- Unit under test may have dependencies on other (complex) units, e.g. database
- To isolate the behaviour of a unit, replace dependencies by "mocks" that simulate the behaviour
- DBs are impractical to incorporate into the unit test.
- In short, mocking is creating objects that simulate the behaviour of real objects.

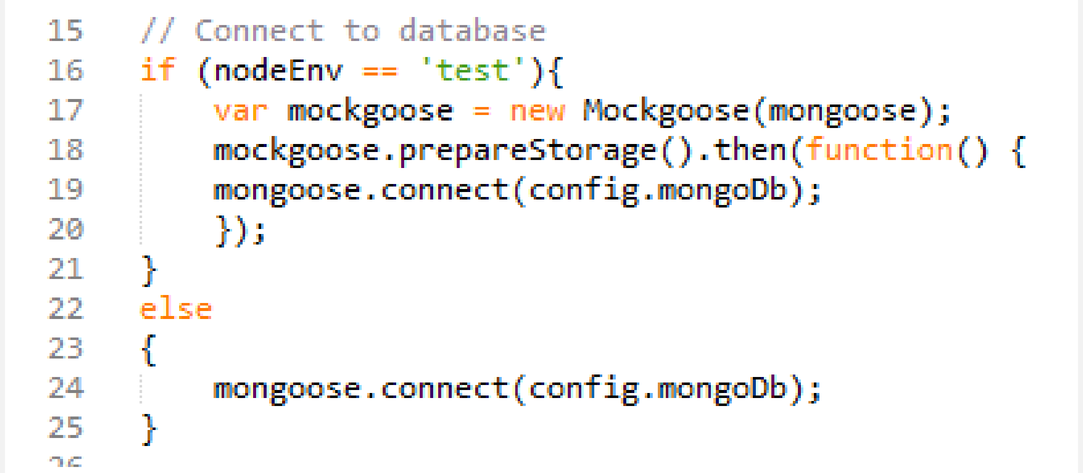
MOCKING MONGODB

- Several mocking frameworks out there
  - Mockery, PowerMockito
- We use Mongoose
  - How about “Mockgoose”?!
  - Turns out it exists!
- NPM install –save-dev Mockgoose

MOCKGOOSE

- Mockgoose spins up **mongod** when mongoose.connect call is made.
- Just uses memory store with no persistence.
- Can take a while on first test, after which it's fast
  - Tests may time out
  - You can increase mocha wait time

```
describe (…){
        this.timeout(10000);
```



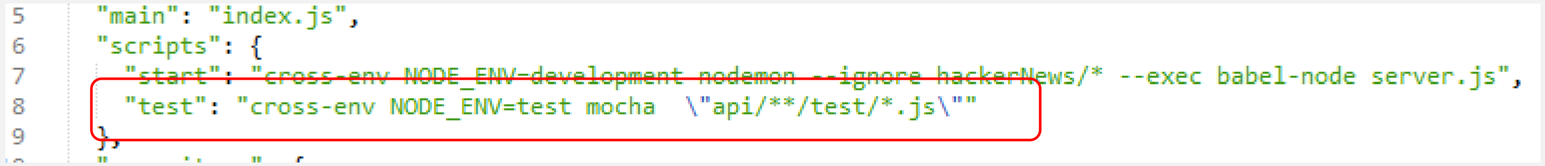
```
// Connect to database
if (nodeEnv == 'test'){
    var mockgoose = new Mockgoose(mongoose);
    mockgoose.prepareStorage().then(function() {
    mongoose.connect(config.mongoDb);
    });
}
else
{
    mongoose.connect(config.mongoDb);
}
```

RUNNING IN TEST ENVIRONMENT

- Notice in the last slide we only use Mockgoose in “test” envornment
- We need to set the NodeEnv environment variable as ‘test’ when we run out test script
  - Setting environment variables is differs across Operating Systems/platforms
- Cross-Env uses a single command to set env variables without worrying about the platform

npm install save-dev cross-env

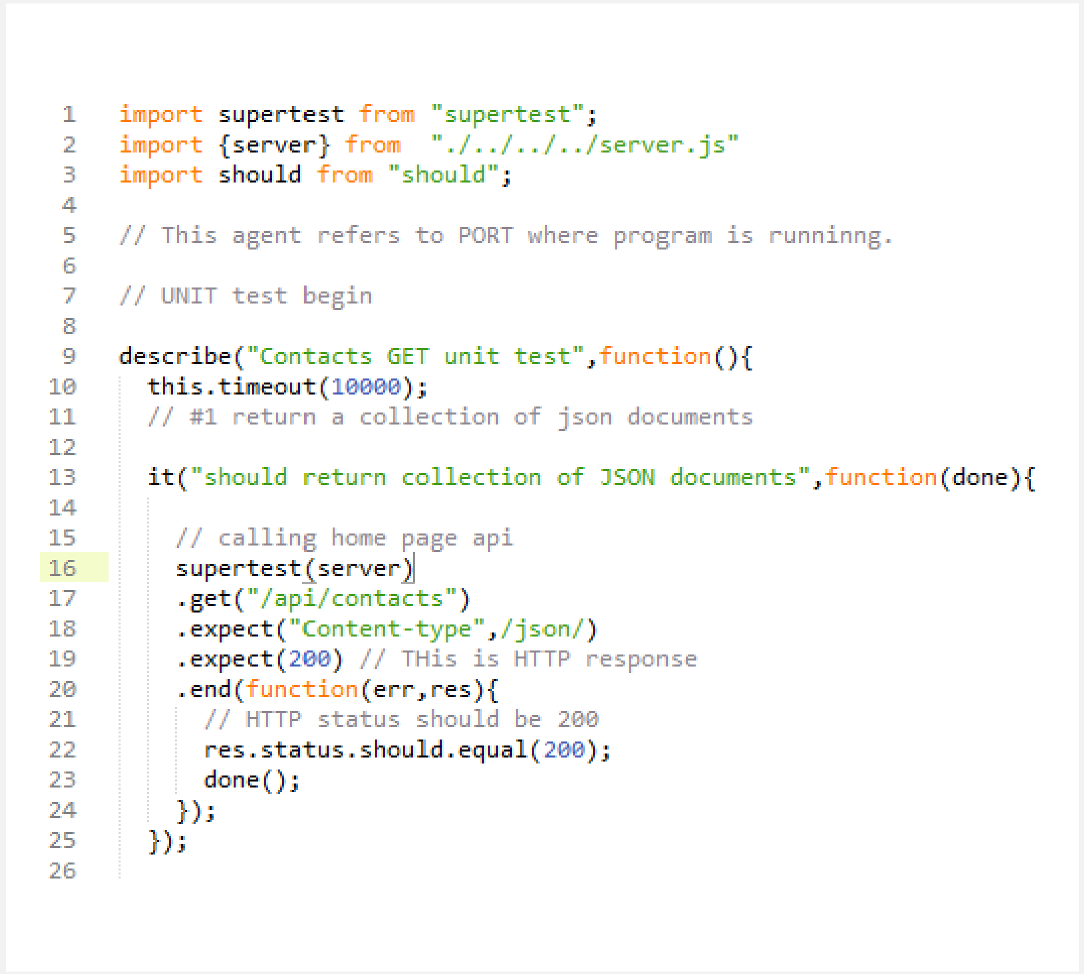
- Update the test script in **package.json** to set the correct environment(s)



```
"main": "index.js",
"scripts": {
  "start": "cross-env NODE_ENV=development nodemon --ignore hackerNews/* --exec babel-node server.js",
  "test": "cross-env NODE_ENV=test mocha  \"api/**/test/*.js\""
},
```

RUNNING SERVER AS PART OF TEST

- SuperTest allows you to create the Express API as part of the test
- You can pass instance of the server to SuperTest
  - if the server is not already listening for connections then SuperTest will bind to a port for you so there is no need to keep track of ports.
- So no need to start the server/bind to port in order to run the unit test.
- Very useful for automated testing.



```
import supertest from "supertest";
import {server} from  "./../../../server.js"
import should from "should";

// This agent refers to PORT where program is runninng.

// UNIT test begin

describe("Contacts GET unit test",function(){
  this.timeout(10000);
  // #1 return a collection of json documents

  it("should return collection of JSON documents",function(done){

    // calling home page api
    supertest(server)
    .get("/api/contacts")
    .expect("Content-type",/json/)
    .expect(200) // THis is HTTP response
    .end(function(err,res){
      // HTTP status should be 200
      res.status.should.equal(200);
      done();
    });
  });

```