

TESTING WEB APIS

Frank Walsh

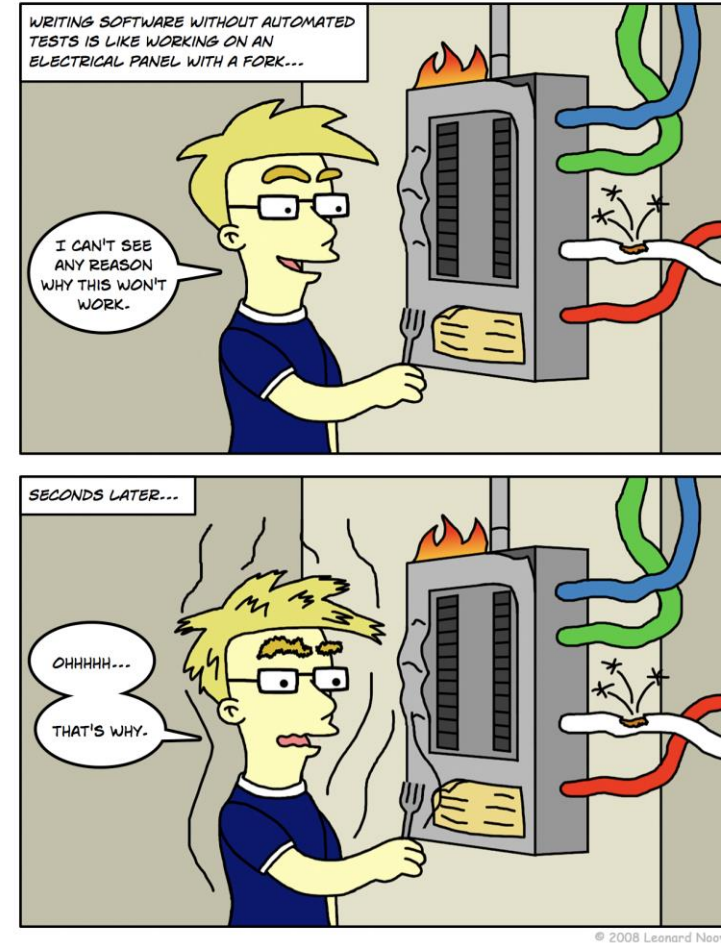
B.Sc.(hons) IT, Y4

AGENDA

- Unit testing
- Mocha
- Should
- SuperTest
- Asynchronous testing

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



UNIT TESTS FOR APIS

- Unit 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
TESTS



API Testing

UNIT TEST CONVENTION

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

/

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

MOCHA

- 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



ASSERTIONS WITH **SHOULD**

- 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 tests
 - Generates nice error messages (there's always error messages!)
 - Works with Node and browsers
 - Can use in async tests with Mocha

```
var should = require('should');

var user = {
  name: 'tj'
  , pets: ['tobi', 'loki', 'jane', 'bandit']
};

user.should.have.property('name', 'tj');
user.should.have.property('pets').with.lengthOf(4);
```

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/)   
      .expect(200, done);  
  });  
});
```

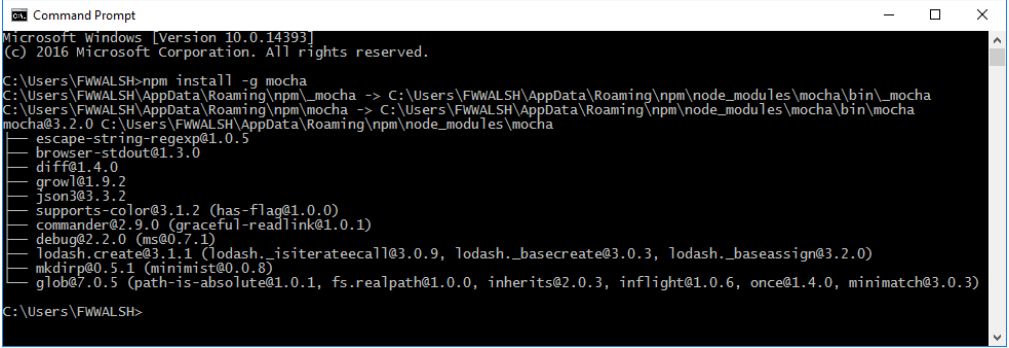

GETTING MOCHA ETC.

- Use NPM and install Mocha, Should and Supertest

npm install --save-dev mocha

npm install --save-dev should

npm install --save-dev supertest



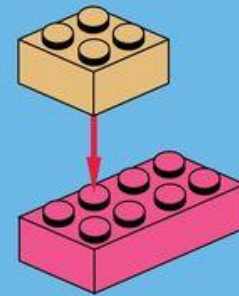
```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

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"
```

HOW IT WORKS...

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



EXAMPLE – 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(){

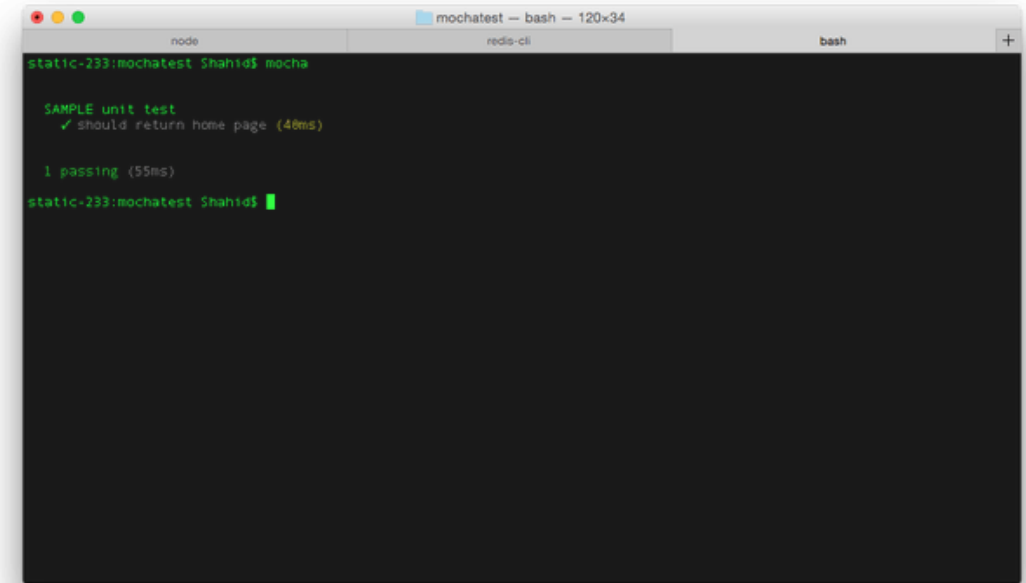
  // #1 should return home page

  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();
      });
  });
});
```

RUNNING THE TEST MANUALLY

- Assuming node API listening on port 3000
- From command prompt, type
 - Mocha
- Will run test scripts in the local folder

A terminal window titled 'mochatest -- bash -- 120x34' with tabs for 'node', 'redis-cli', and 'bash'. The terminal shows the command 'mocha' being executed, followed by test output: 'SAMPLE unit test', '✓ should return home page (40ms)', '1 passing (55ms)', and the prompt 'static-233:mochatest Shahid\$' with a green cursor.

```
static-233:mochatest Shahid$ mocha

SAMPLE unit test
✓ should return home page (40ms)

1 passing (55ms)
static-233:mochatest Shahid$
```

INCLUDE IN NODE PROJECT

- Can associate tests with node project by including new script property
- Set up a test script in package.json
- Run tests with **npm run test**

```
"scripts": {  
  "test": "mocha"  
}
```

```
$ npm test
```

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-existent/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:mocha:Shahid$ mocha  
  
SAMPLE unit test  
✓ should return home page (38ms)  
1) should add two number  
✓ should return 404  
  
2 passing (36ms)  
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:39:16)  
   at _stream_readable.js:990:16  
  
static-233:mocha: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.

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

```
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();  
      });  
    });  
  });  
});
```