44-563: Unit 11

Developing Web Applications and Services

Includes

- Project requirements
- Exam 2 Wed
- Client Introduction Friday

Exam 2
Wednesday

Exam 02

- 100 points online, 50 minutes (~ 50 questions)
- You may bring one 8-1/2 x 11 sheet of paper with **handwritten** notes (front and back).
- See Exam 02 **Review Guide**
- Similar to Exam 1 - all questions asked on computer.
- Recognize file extensions and code.
- Review slides.
- Know our standard tools - VS Code, Git
- Code examples may be taken from the class project.
- Security & testing (this week's content) not on exam 2, but may appear on the final

Project Requirements

Each section: Make a working app

1. **All costs** should be calculated correctly.
2. Use **Issue Tracker** to let your section know what each person is doing and the current status.

Requirements

- To reduce merge conflicts, we like to have **one and only one programming unit responsible for every code file**.
- In order to implement your responsible area, you will have ownership of one or more files.
- You must also have your "**about**" page where you will put a personal marketing page like you've developed since the beginning of class.
- Comments are a good way to start - they don't break the code.
- Complex files (e.g. the model, the controller) should have the unit that created them commit the initial code - any opening statements and provide comments letting other units know where to add their contributions.

M11 Fork the reference app

- Refer to the reference application here: https://bitbucket.org/professorcase/node-express-mvc-ejs-start/src
- Fork the repo. Get it running on your machine. Review the reference app. Explore the models, explore the controllers, explore the views. Find app.js. Locate the navigation functionality (the menu). Change things and explore how it all fits together.
- Fun incentive: there **may** be some examples drawn from the project or this reference app on Wednesday's exam

Continue your app

- Continue to create your files (files & folders are assigned to each unit - see the project-specific information in the project slides).
- Use comments to organize your thoughts & code
- More specific info on each units A04 requirements will be coming. (They will include building out your responsible files and delegating portions of complex files (e.g. the controller and views) to other units.

Pull code; add one set of comments; immediately push code.

*Never work on **stale** code (this makes merging more difficult).*

Get ready for client visit Friday

Agenda:

- Instructor will introduce the client
- The client make some opening remarks
- We will introduce each unit/pair to the client by responsible area. Unit 1 will start and turn it over to Unit 2 and so on... (a mock run-through during class today may be helpful).
- General questions and comments
- Working session: time for working on the app and the ability to come talk with your instructors, assistants, and the client to get additional information

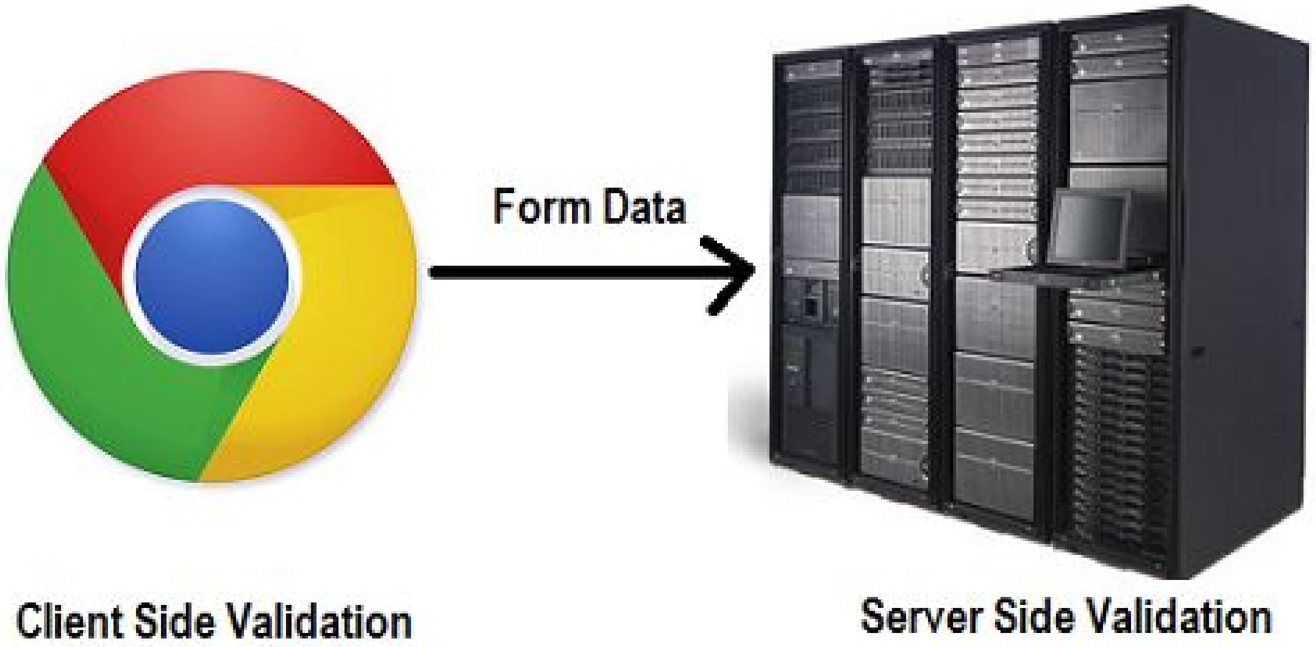
Security

Security

Should we address security on the **client** side - or the **server** side?

Security

Always address security on BOTH the client-side and server-side.

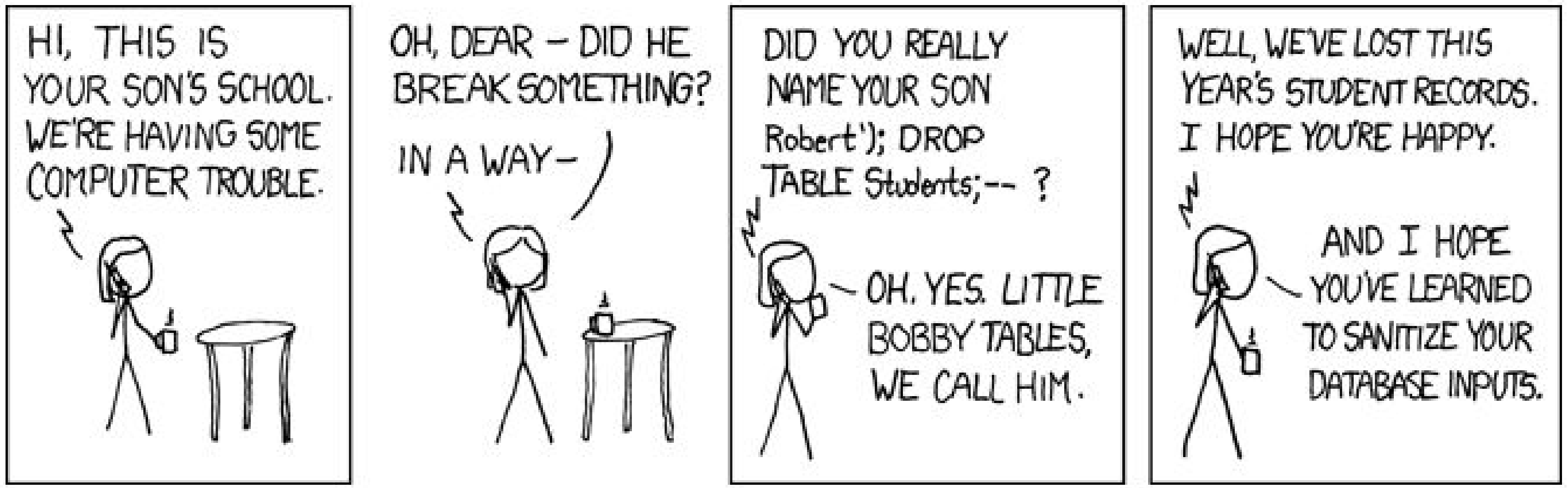


http://net-informations.com/faq/asp/validation.htm

Security

- **Critical** in all web apps. Client-side and especially server-side.
- **Validate/filter/sanitize all user inputs**. Guards against:
  - XSS (Cross site scripting - executable JS)
  - SQL Injection (read sensitive information with SQL)
  - Command injection (multiple commands in a URI)
- HTTP is clear text - secure with HTTPS
- Specify HTTP header settings
- Manage exceptions, errors, logging - don't expose unnecessary details
- Node.js Security Tips
- Nodejs Security Checklist

Don't simply concatenate user-supplied input :)


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Novice coder uses this in their code:

```
"SELECT * FROM Students WHERE firstname LIKE '" + user_input + "';"
```

User provides: *Robert'; DROP TABLE Students;*

When this gets executed now, it will actually be two commands:

```
"SELECT * FROM Students WHERE firstname LIKE 'Robert';
DROP TABLE Students;"
```

Testing

there’s *always* time for testing

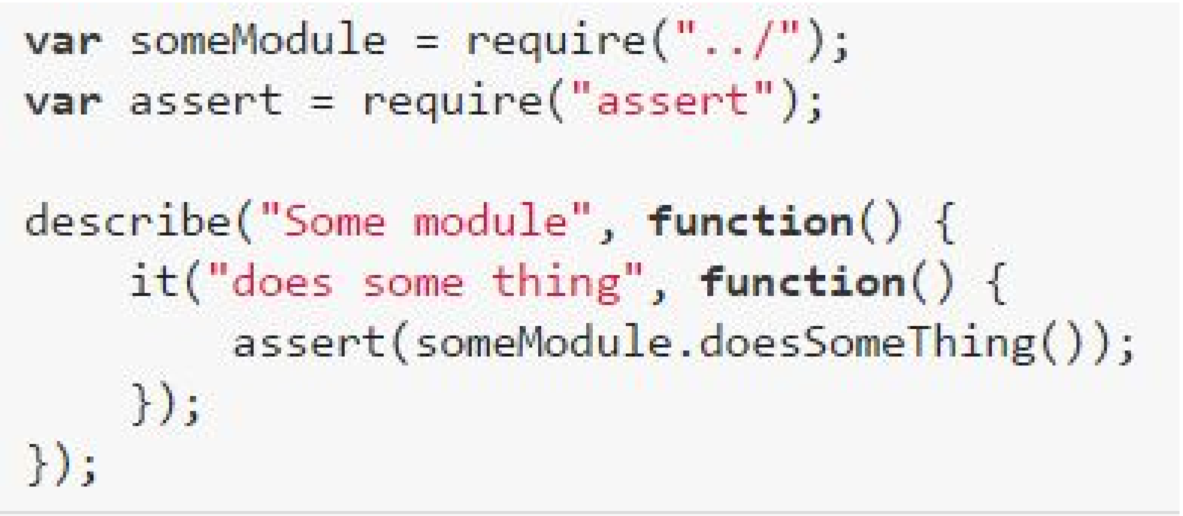
Testing

- Nearly every major project should include tests.
- By convention, tests go in **test** folder, with subfolders arranged like source (e.g. models, views, controllers).
- Popular choices for Node.js include:
  - **Mocha**
  - **Supertest**
  - **Chai**
  - should
  - assert

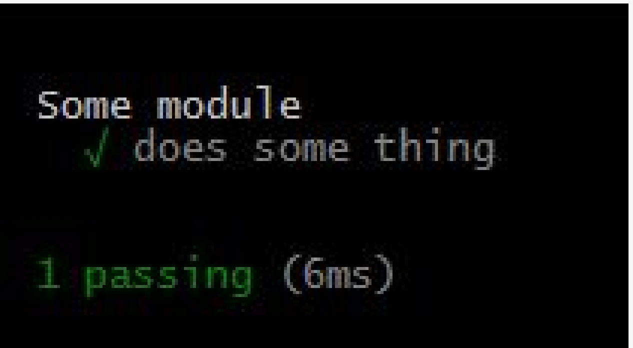
Mocha

- "Simple, flexible, fun"
- Test framework for Node.js
- https://mochajs.org/
- Handles async calls well

Mocha Test Framework



```
var someModule = require("../");
var assert = require("assert");

describe("Some module", function() {
    it("does some thing", function() {
        assert(someModule.doesSomeThing());
    });
});
```



```
Some module
  √ does some thing

1 passing (6ms)
```

1. **describe()** can be used to define test suites
2. A test suite contains test cases, defined by **it()**
3. it() takes a string and function.
4. The test case passes if no error is thrown within the function
5. Mocha can work with a number of assertion frameworks

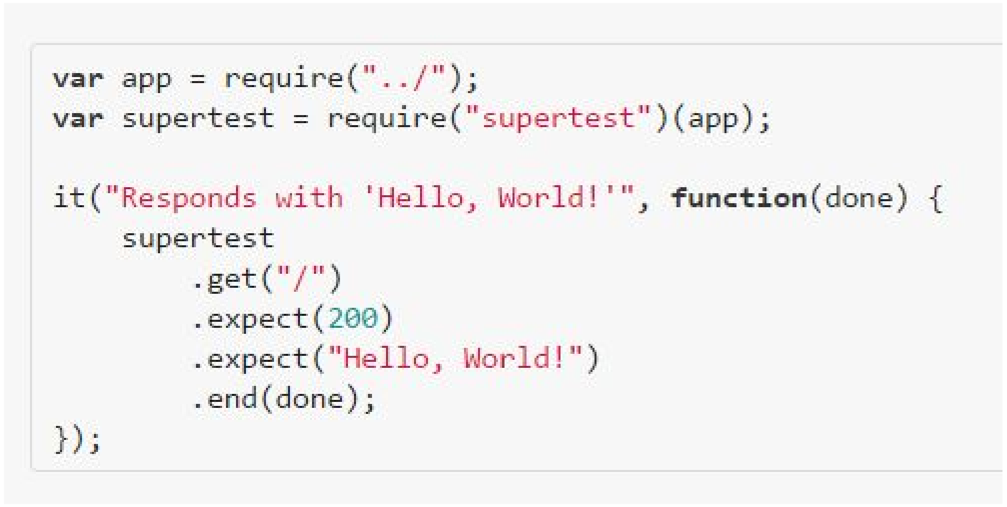
http://willi.am/node-mocha-supertest/#10

https://gist.github.com/samwize/8877226

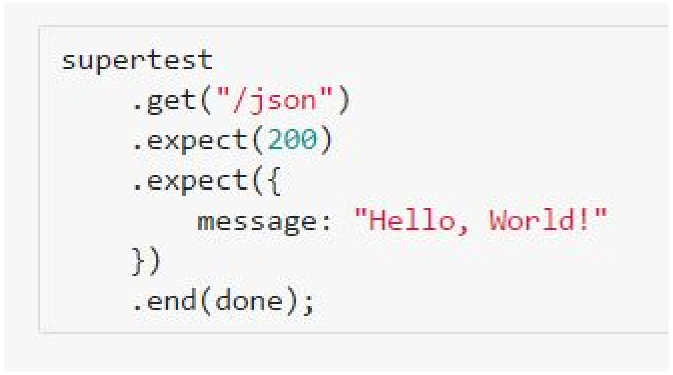
Supertest

- Support for HTTP tests
- Handles server setup and teardown
- Simplifies request & response validation
- Improves readability

The snippet at left shows Supertest being used in conjunction with Mocha.



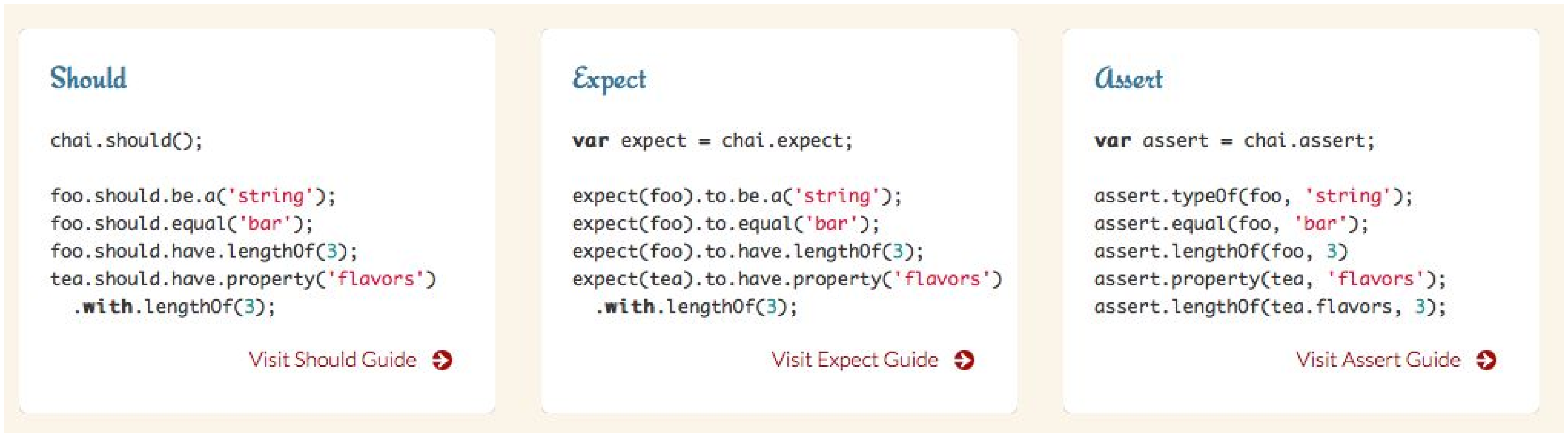
```
var app = require("../");
var supertest = require("supertest")(app);

it("Responds with 'Hello, World!'", function(done) {
    supertest
        .get("/")
        .expect(200)
        .expect("Hello, World!")
        .end(done);
});
```



```
supertest
    .get("/json")
    .expect(200)
    .expect({
        message: "Hello, World!"
    })
    .end(done);
```

Chai Assertion Library

- Behavior-driven development (BDD)
- Test-driven development (TDD)
- Has several alternative interfaces, pick your favorite!
- Should, Expect, Assert



Should

```
chai.should();

foo.should.be.a('string');
foo.should.equal('bar');
foo.should.have.lengthOf(3);
tea.should.have.property('flavors')
  .with.lengthOf(3);
```

Visit Should Guide

Expect

```
var expect = chai.expect;

expect(foo).to.be.a('string');
expect(foo).to.equal('bar');
expect(foo).to.have.lengthOf(3);
expect(tea).to.have.property('flavors')
  .with.lengthOf(3);
```

Visit Expect Guide

Assert

```
var assert = chai.assert;

assert.typeOf(foo, 'string');
assert.equal(foo, 'bar');
assert.lengthOf(foo, 3)
assert.property(tea, 'flavors');
assert.lengthOf(tea.flavors, 3);
```

Visit Assert Guide

Setting up

1. Install to development environment (just once - after that, it's in the package.json for the whole team):

- **npm install --save-dev mocha**
- **npm install --save-dev supertest**
- **npm install --save-dev chai**
- **npm install --save-dev should**
- Or add to package.json. [Done]

2. Add script in package.json [Done]

package.json

```
"dependencies": {
   "body-parser": "^1.13.2",
   "express": "^4.13.1".
  .....
 },

 "devDependencies": {
   "chai": "^3.5.0",
   "mocha": "^3.1.2",
   "should": "^11.1.1",
   "supertest": "^2.0.1"
 },

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

Run app & execute tests

Start app in the usual way:

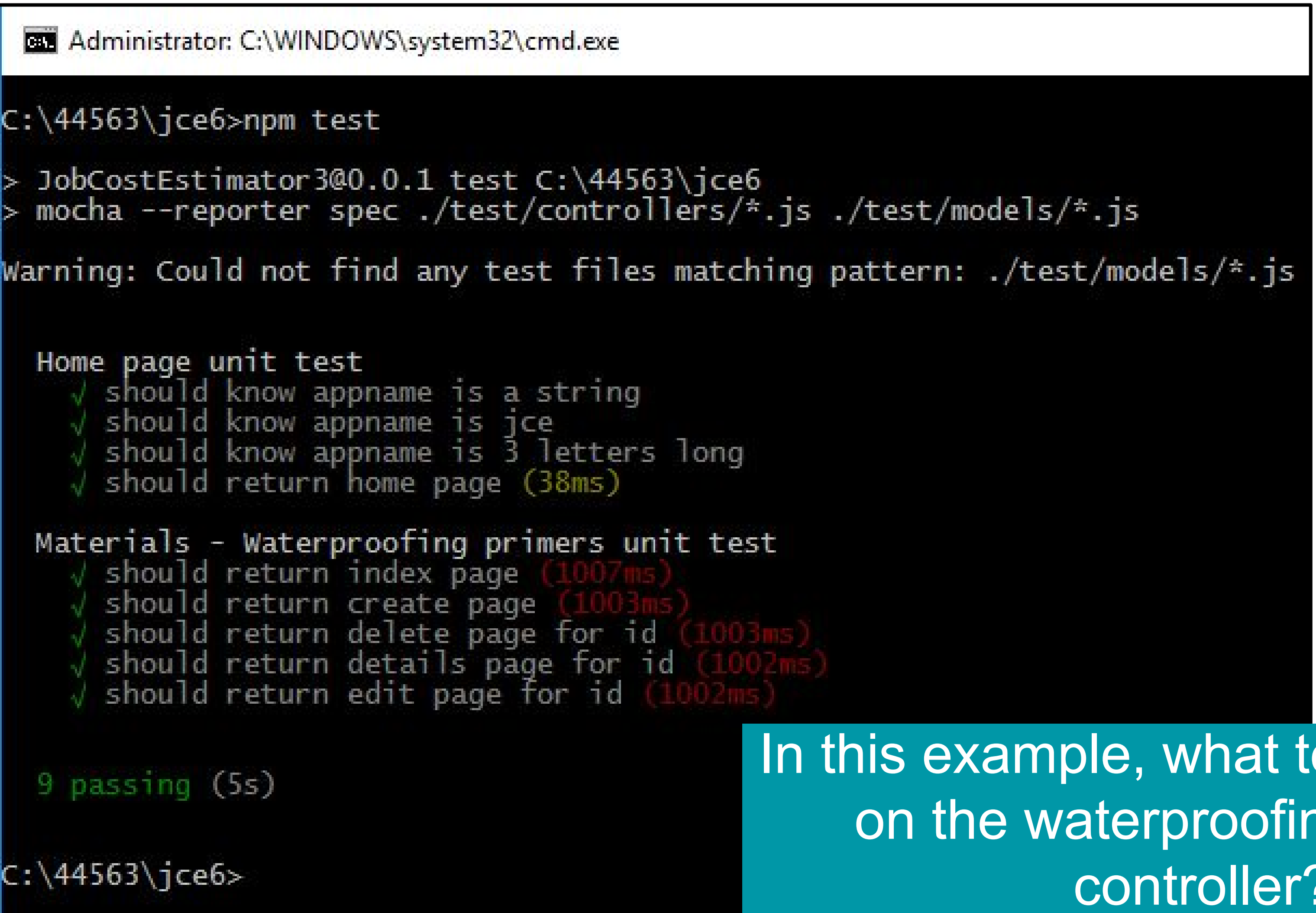
1. Refresh packages with **npm install**
2. Start app with **nodemon**.

Then, in a new command window:

3. Execute the test script with **npm test**

Setup is done when you pull the new code.
Just start up your app and execute the tests.

Testing window output:

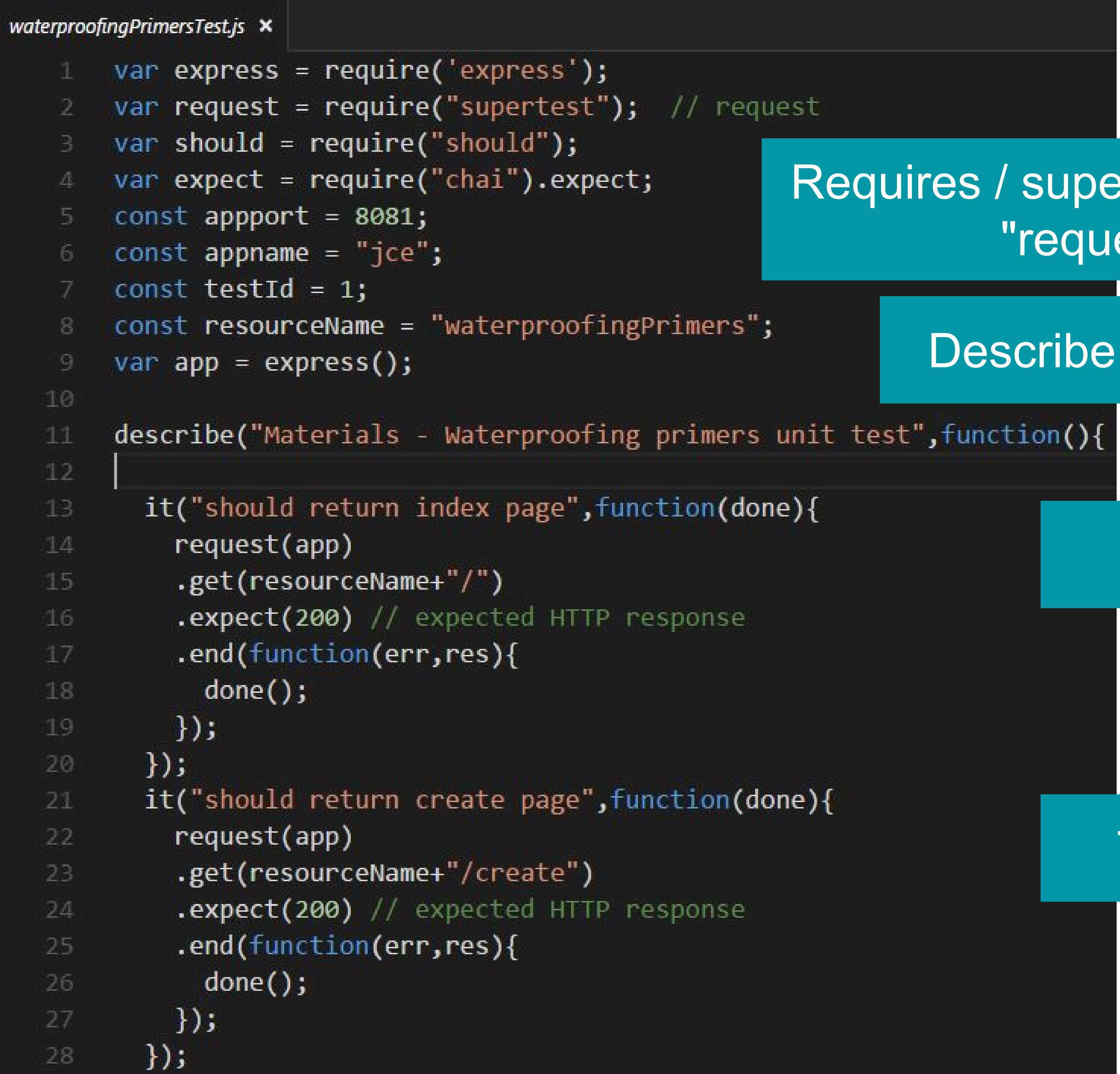


```
C:\44563\jce6>npm test

> JobCostEstimator3@0.0.1 test C:\44563\jce6
> mocha --reporter spec ./test/controllers/*.js ./test/models/*.js

Warning: Could not find any test files matching pattern: ./test/models/*.js


  Home page unit test
    √ should know appname is a string
    √ should know appname is jce
    √ should know appname is 3 letters long
    √ should return home page (38ms)

  Materials - Waterproofing primers unit test
    √ should return index page (1007ms)
    √ should return create page (1003ms)
    √ should return delete page for id (1003ms)
    √ should return details page for id (1002ms)
    √ should return edit page for id (1002ms)


  9 passing (5s)


C:\44563\jce6>
```

In this example, what tests are done on the waterproofingPrimers controller?

Create controller tests

1. Goto C:\44563, right-click on your cge folder / Open with code.
2. Open test/controller. Create your estimateTest.js.
3. Copy content from waterproofingPrimerTest.js.
4. What parts will you need to modify? (we'll look at the code next)



waterprooflingPrimersTest.js

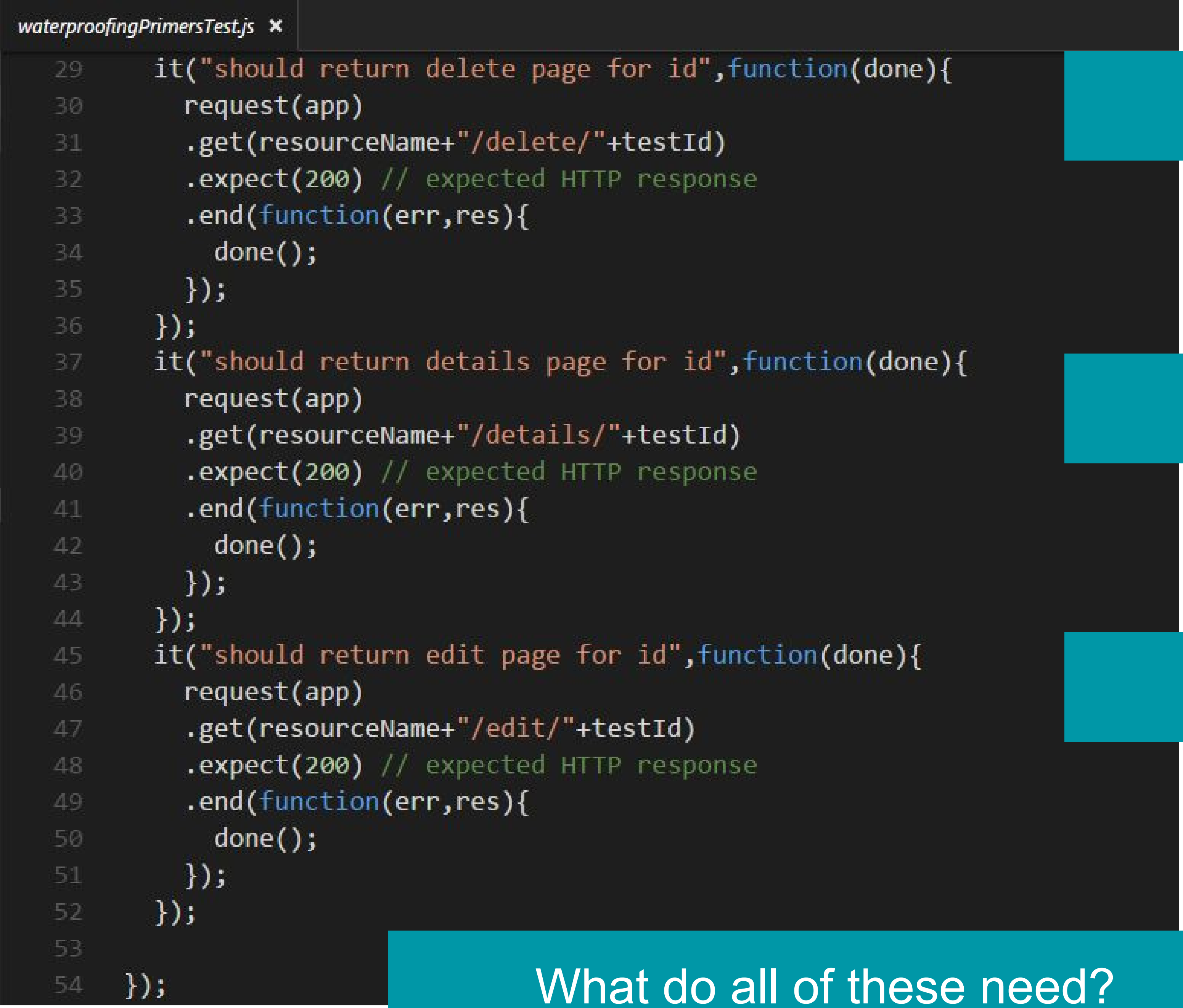
```
var express = require('express');
var request = require("supertest");  // request
var should = require("should");
var expect = require("chai").expect;
const appport = 8081;
const appname = "jce";
const testId = 1;
const resourceName = "waterproofingPrimers";
var app = express();

describe("Materials - Waterproofing primers unit test",function(){

  it("should return index page",function(done){
    request(app)
    .get(resourceName+"/")
    .expect(200) // expected HTTP response
    .end(function(err,res){
      done();
    });
  });
  it("should return create page",function(done){
    request(app)
    .get(resourceName+"/create")
    .expect(200) // expected HTTP response
    .end(function(err,res){
      done();
    });
  });
```

Requires / supertest is called "request"

Describe this unit test

Test index

Test create


waterproofingPrimersTest.js ×

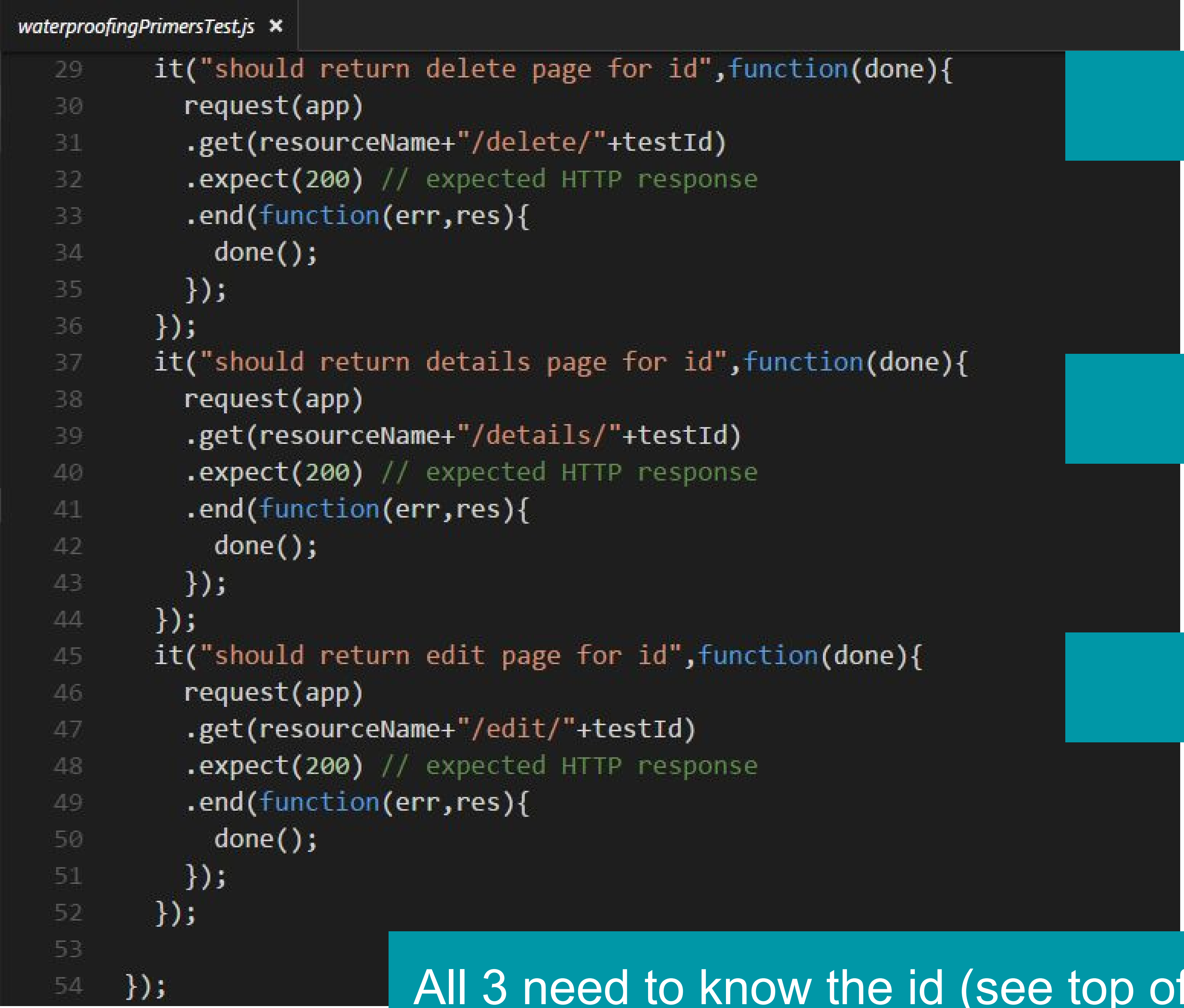
```
    it("should return delete page for id",function(done){
      request(app)
      .get(resourceName+"/delete/"+testId)
      .expect(200) // expected HTTP response
      .end(function(err,res){
        done();
      });
    });
    it("should return details page for id",function(done){
      request(app)
      .get(resourceName+"/details/"+testId)
      .expect(200) // expected HTTP response
      .end(function(err,res){
        done();
      });
    });
    it("should return edit page for id",function(done){
      request(app)
      .get(resourceName+"/edit/"+testId)
      .expect(200) // expected HTTP response
      .end(function(err,res){
        done();
      });
    });

  });
```

delete

details

edit

What do all of these need?



waterproofingPrimersTest.js ×

```
    it("should return delete page for id",function(done){
      request(app)
      .get(resourceName+"/delete/"+testId)
      .expect(200) // expected HTTP response
      .end(function(err,res){
        done();
      });
    });
    it("should return details page for id",function(done){
      request(app)
      .get(resourceName+"/details/"+testId)
      .expect(200) // expected HTTP response
      .end(function(err,res){
        done();
      });
    });
    it("should return edit page for id",function(done){
      request(app)
      .get(resourceName+"/edit/"+testId)
      .expect(200) // expected HTTP response
      .end(function(err,res){
        done();
      });
    });

  });
```

delete

details

edit

All 3 need to know the id (see top of the file).

Testing References

- http://willi.am/node-mocha-supertest/
- https://codeforgeek.com/2015/07/unit-testing-nodejs-application-using-mocha/
- https://thewayofcode.wordpress.com/2013/04/21/how-to-build-and-test-rest-api-with-nodejs-express-mocha/

Client Visit

Agenda

- Instructor will introduce the client
- The client make some opening remarks
- We will introduce each unit/pair to the client by responsible area. Unit 1 will start and turn it over to Unit 2 and so on...
- General questions and comments
- Working session: time for working on the app and the ability to come talk with your instructors, assistants, and the client to get additional information

C&G Coatings *specializes in roof restoration, waterproofing and floor coatings.*

They are a veteran-owned business, providing customers with quality service,
100% satisfaction,
and a cost effective solution to their
***roofing**,*
***flooring**,*
*and **waterproofing** needs.*

Our Client

Curt Carpenter

- C&G Coatings
- https://www.facebook.com/people/Cg-Coatings/100009692642964
- https://www.coatingsplusstore.com/
- Project objectives
- Contact information
    - Phone: 913-634-2588
    - curt.carpenter0@gmail.com
    - https://www.facebook.com/CG-Coatings-104818302930011/

Introduce 12 units & responsible areas:

1. root folder (.gitignore, README.md, LICENSE.md, package.json, app.js, .env.example)
2. /data/ (estimates.json, users.json)
3. /controllers/ (estimate.js, user.js, contact.js, about.js)
4. /models/   (estimate.js, user.js)
5. /routes/ (index.js)
6. /views (layout.ejs, index.ejs, header.ejs, footer.ejs) & views/estimate (create.ejs, delete.ejs,details.ejs, edit.ejs, index.ejs)
7. /utils/ (logger.js, seeder.js)
8. /test/controllers/ (estimateTest.js and userTest.js)
9. /test/models/ (estimateTest.js, userTest.js)
10. /config (ensureAuthenticated.js, passportConfig.js)
11. /views (about.ejs, 404.ejs) & /views/account (forgot.ejs, login.ejs, profile.ejs, reset.ejs, signup.ejs)
12. /public client-side assets and icons (css, images, js)

Our questions for the client / general discussion

Client: Questions, comments for the development team?

Working session

http://www.whittleseacommunityfutures.org.au/i/partnership.jpg