Large Scale JavaScript on Client and Server

Module 4: Testable JavaScript

Shawn Wildermuth
Wilder Minds LLC
@shawnwildermuth

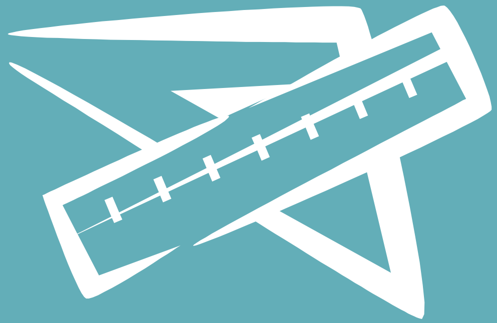
pluralsight
hardcore dev and IT training

Agenda

- **Testable JavaScript**
  - Why is Testing JavaScript Hard?
  - What is Unit Testing
  - Using Jasmine to Write Unit Tests
  - Using GruntJS to Execute Tests
  - Using GruntJS to Automate Executing Tests

M
S

```
// Delete Button
$("#trip-list").on("click", ".delete-button", function () {
  var item = $(this).parent().parent();
  var key = item.attr("data-id");
  var name = destinations.cache[key].city;
  item.remove();
  trip.stops.splice(trip.stops.indexOf(key), 1);
});

// Save Button
$("#saveList").on("click", function () {
  trip.name = $("#tripName").val();
  if (trip.name.length > 0) {
    $.post("/api/user/" + userid + "/trips", trip)
      .then(function (r) {
        alert("saved");
        window.location = "/";
      }, handleError);
  }
});

// Add to List Button
$destList.on("click", ".add-button", function () {
  var item = $(this).parent();
  var cityKey = item.attr("data-id");
  if (!_.contains(trip.stops, cityKey)) {
    var dest = destinations.cache[cityKey];
    var stop = cityTemplate({ key: cityKey, dest: dest});
    $("#trip-list").append(stop);
    trip.stops.push(cityKey);
  }
});
```



Ad-hoc JavaScript fails to separate concerns and is simply impossible to test except for UI-centric testing.

Why is Testing This So Hard?

```
// Save Button
$("#saveList").on("click", function () {
  trip.name = $("#tripName").val();
  if (trip.name.length > 0) {
    $.post("/api/user/" + userid + "/trips", trip)
      .then(function (r) {
        alert("saved");
        window.location = "/";
      }, handleError);
  }
});
```

Why is Testing This So Hard?

```
// Save Button
$("#saveList").on("click", function () {
  trip.name = $("#tripName").val();
  if (trip.name.length > 0) {
    $.post("/api/user/" + userid + "/trips", trip)
      .then(function (r) {
        alert("saved");
        window.location = "/";
      }, handleError);
  }
});
```

Anonymous Function

Why is Testing This So Hard?

```
// Save Button
$("#saveList").on("click", function () {
  trip.name = $("#tripName").val();
  if (trip.name.length > 0) {
    $.post("/api/user/" + userid + "/trips", trip)
      .then(function (r) {
        alert("saved");
        window.location = "/";
      }, handleError);
  }
});
```

Anonymous Function

Why is Testing This So Hard?

Side effect (global or closure)

```
// Save Button
$("#saveList").on("click", function () {
  trip.name = $("#tripName").val();
  if (trip.name.length > 0) {
    $.post("/api/user/" + userid + "/trips", trip)
      .then(function (r) {
        alert("saved");
        window.location = "/";
      }, handleError);
  }
});
```

Why is Testing This So Hard?

Business Rule

```
// Save Button
$("#saveList").on("click", function () {
  trip.name = $("#tripName").val();
  if (trip.name.length > 0) {
    $.post("/api/user/" + userid + "/trips", trip)
      .then(function (r) {
        alert("saved");
        window.location = "/";
      }, handleError);
  }
});
```

Why is Testing This So Hard?

```
// Save Button
$("#saveList").on("click", function () {
  trip.name = $("#tripName").val();
  if (trip.name.length > 0) {
    $.post("/api/user/" + userid + "/trips", trip)
      .then(function (r) {
        alert("saved");
        window.location = "/";
      }, handleError);
  }
});
```

Navigation mixed in

TESTING IN WEB APPLICATIONS

UI TESTS

UNIT TESTS

INTEGRATION TESTS

u·nit test·ing

/yoo-nit tɛst-ɪŋ/

Noun

1. A method by which individual units of source code together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for use.

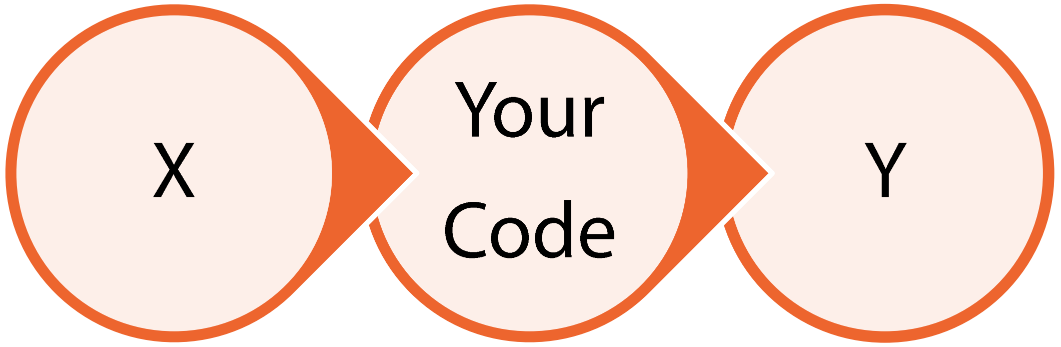
“…we write the first horrible implementation that works. And then you go back and make it better. Tests let you do that much more safely. If your first version had tests to prove it worked, the same tests should pass when you write your second version.”

— Rebecca Murphey

WHAT IS A UNIT TEST?

Atomic test that determines expected behavior

- **For specific input, what is the expected output**



Testability in JavaScript is relative to the amount of effort you put into creating maintainable and scalable code. Good separation and modularization of your code makes it more testable.

WRITING UNIT TESTS

For JavaScript, you have a couple of options for tests:

- Jasmine
- QUnit
- Mocha
- Etc.

Doesn't matter which one…
Just make sure you're testing your code!

Jasmine

- **A behavior-driven development framework for testing JavaScript code**
  - http://pivotal.github.io/jasmine/
  - Supported in many test runners
    - We'll see how it works with GruntJS

An Example Using Jasmine

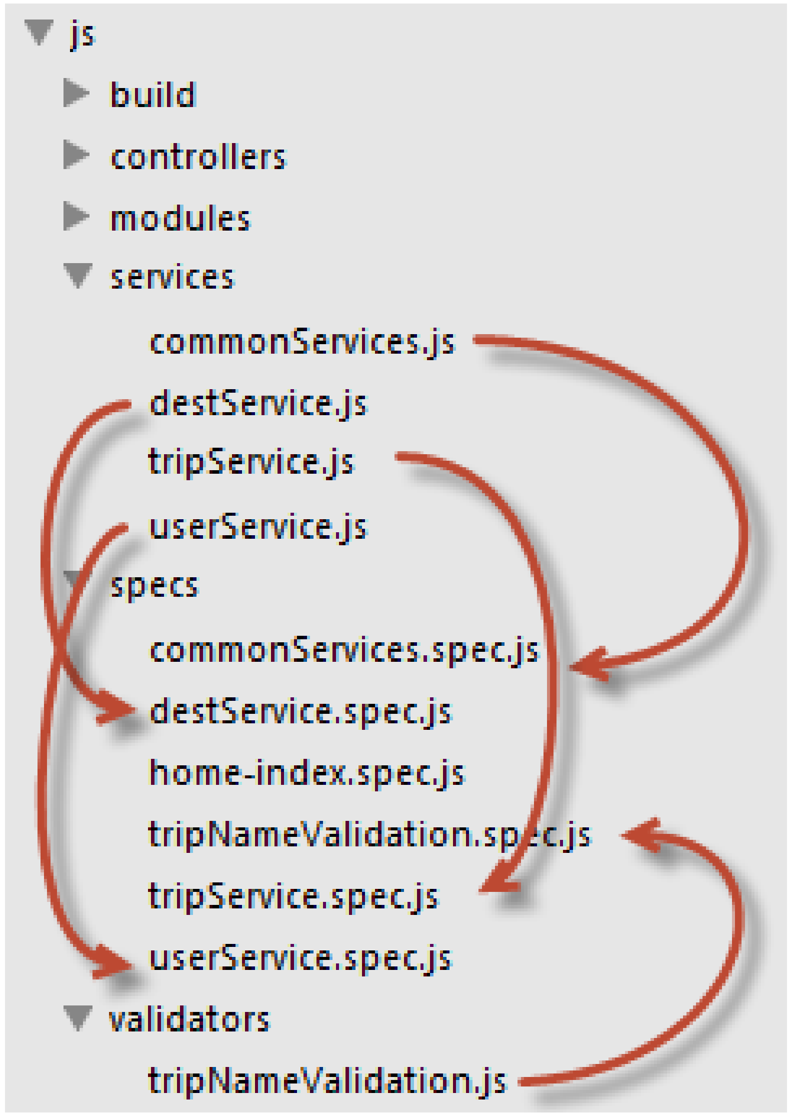
Suite of Tests

Spec

Expectation

```
describe("a test", function() {

  it("what you are testing", function() {

    expect(true).toBeTruthy();

  });

});
```

Typically Suite Per JavaScript File*



```
▼ js
  ▶ build
  ▶ controllers
  ▶ modules
  ▼ services
      commonServices.js
      destService.js
      tripService.js
      userService.js
  ▼ specs
      commonServices.spec.js
      destService.spec.js
      home-index.spec.js
      tripNameValidation.spec.js
      tripService.spec.js
      userService.spec.js
  ▼ validators
      tripNameValidation.js
```

* assumes you're writing one file per unit of work

Lots of Tests Per Suite

```
describe("tripNameValidation", function() {

  it("is valid", function () {
    expect(worldTripper.tripNameValidation).toBeDefined();
  });

  it("verify - just numbers", function () {
    expect(worldTripper.tripNameValidation.verify("12345678")).toBeTruthy();
  });

  it("verify - too long", function () {
    expect(worldTripper.tripNameValidation.verify("12345678901234567890"))
      .not.toBeTruthy();
  });
});
```

Supports Nested Suites

```
describe("app test", function() {

  describe("module test", function() {

    it("module should be defined", function() {
      var mod = angular.module("home-index");
      expect(mod).toBeDefined();
    });

  });
});
```

Setup and Teardown

```
describe("app test", function() {

  var mod = null;
  beforeEach(function () { mod = new Module(); });
  afterEach(function () { mod = null; });

  it("module should be defined", function() {
    expect(mod).toBeDefined();
  });

});
```

Dependencies and Mocks

Mocked Dependency

```
describe("app test", function() {

  var mod, ds;
  beforeEach(function () {
    ds = { // Mock the dependency
      saveData: function (obj) { return true;}
    };
    mod = new SomeModule(ds);
  });

  it("module should be defined", function() {
    expect(mod.save({})).toBeTruthy();
  });
});
```

Required Dependency

Spying

```
describe("app test", function() {

  var mod, ds;
  beforeEach(function () {
    ds = { // Mock the dependency
      saveData: function (obj) { return true;}
    };
    mod = new SomeModule(ds);
    spyOn(ds, "saveData").andReturn(true);
  });

  it("module should be defined", function() {
    expect(mod.save({})).toBeTruthy();
    expect(ds.saveData).toHaveBeenCalled();
  });
});
```

See if dependency is ever called

Test expectation

Testing Framework Code

- **App Frameworks usually include support for unit testing**
  - E.g. AngularJS
    - Supports mocking of objects and calls
    - Specific Support for mocking up REST/HTTP Calls
    - See my course for more details

GruntJS Can Run Jasmine Tests (et al.)

```
> npm install grunt-contrib-jasmine --save-dev
```

```
grunt.initConfig({
  jasmine: {
    src: [ 'public/js/home-index.js',
           'public/js/services/*.js',
           'public/js/validators/*.js' ],
    options: {
      vendor: ["public/js/vendor/angular.js", ],
      specs: 'public/js/specs/*.js'
    }
  },
...
grunt.loadNpmTasks('grunt-contrib-jasmine');
```

Your code

Code to load first

Specs (tests)

```
> grunt jasmine
```

demonstration

GruntJS and Jasmine

TEST AUTOMATION

Not just for running tests during 'build' phase

- You want to simplify your writing of tests
- Re-execution of tests on source/unit change is common method
- Audible to see if your tests are failing

Automating Jasmine with GruntJS

```
> npm install grunt-contrib-watch --save-dev
```

```
grunt.initConfig({
  watch: {
    scripts: {
      files: ['public/js/**/*.js'],
      tasks: ['jasmine']
    },
  },
...
grunt.loadNpmTasks('grunt-contrib-jasmine');
```

If any of these files change

Run these tasks

```
> grunt watch
```

demonstration

GruntJS and Watch

Summary

- **Testable JavaScript**
  - Event-driven, nested JavaScript is hard to test
  - Too many mixed concerns are a problem to testing
  - Maintainable and Scalable JavaScript should be easy to test
  - Unit testing is critical, but don’t get bogged down with dogma of process
  - Jasmine and GruntJS are a great combination to simplify this
  - Automating running unit tests can speed up your overall testing