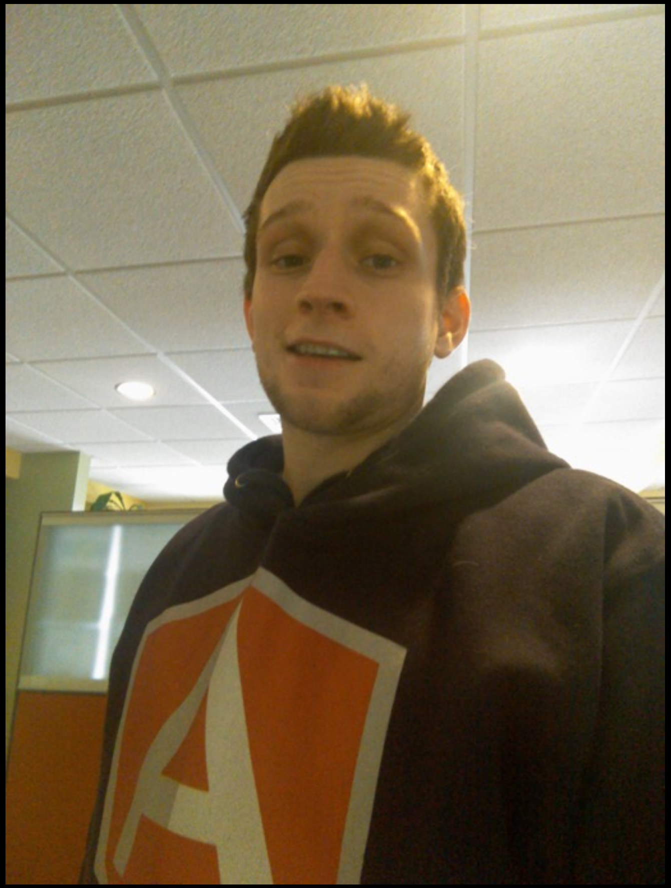
Describe's Full of It's

Jim Lynch

June 2016

Hi, I'm Jim Lynch



Front-End Engineer
at Altered Image

WebStorm
Ambassador

Programming Tweeter
@webWhizJim

Slides available here:
http://www.slideshare.net/JimLynch22/describes-full-of-its

Who is This Talk For?

- AngularJS developers.
- Front-End developers.
- Anyone interested in unit testing.

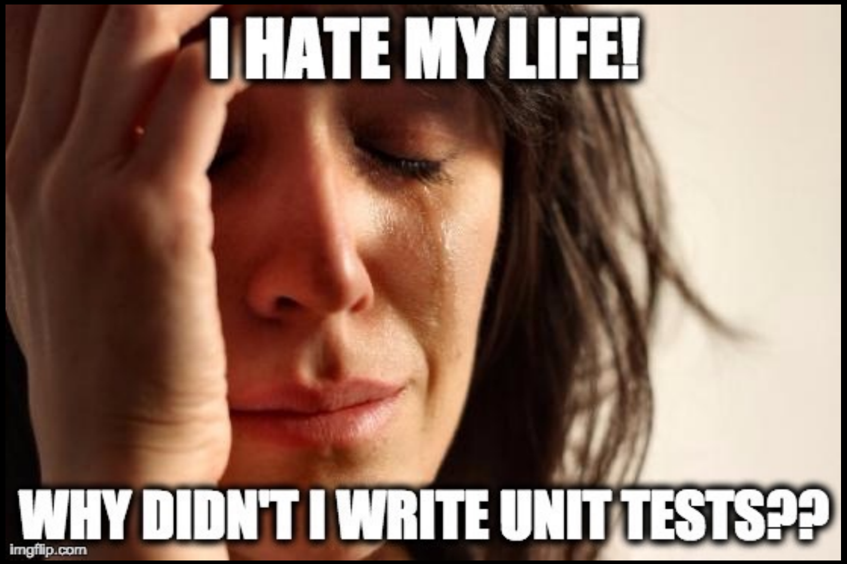
Why Test?

- To prevent regression (recurring bugs).
- So you don’t have to keep testing manually.
- To catch bugs before end users see them.
- To remove fear from refactoring.
- To document what your code should do.
- To gain a sense of confidence that you can ***never*** have without tests!

If You Don’t Test…

→

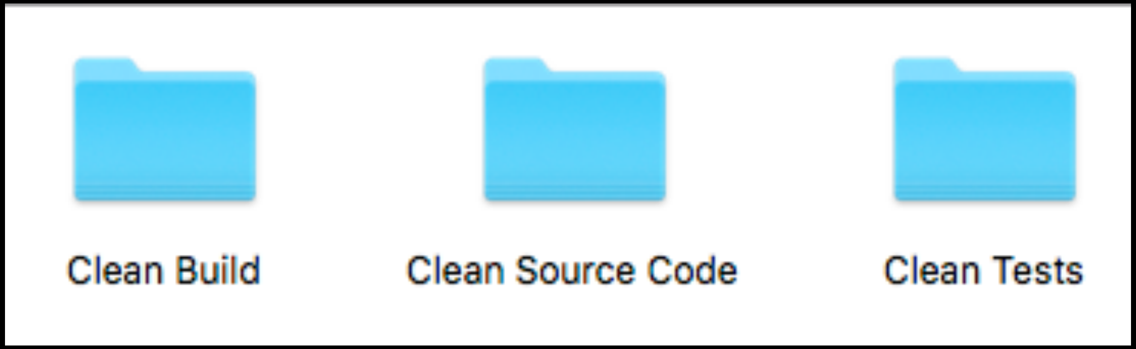
→



DON’T IGNORE TESTING!

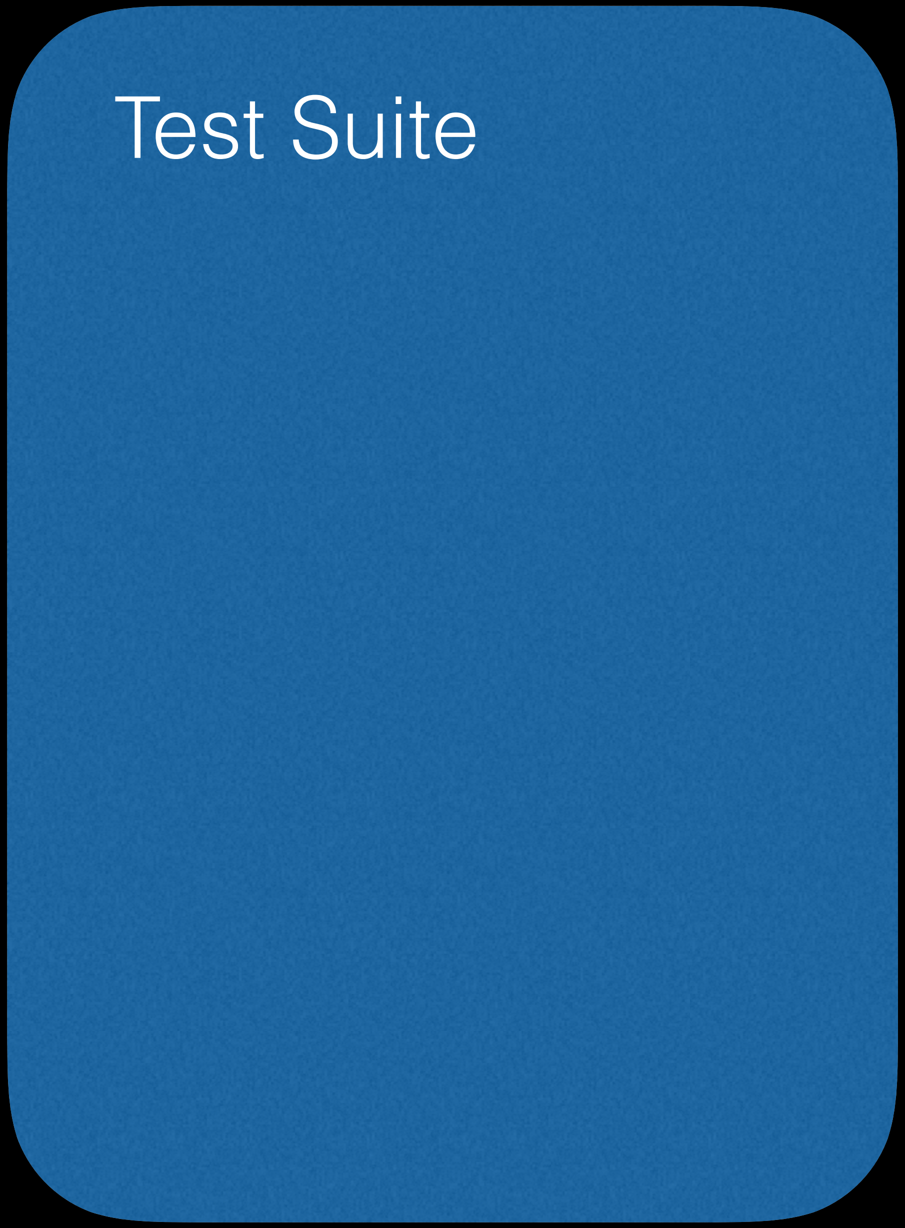
Artifacts

- When the project is over, what will you leave behind?
- "A programmer's deliverables should be clean code and clean tests"
  - Pete Heard
- True, but also a clean production build (the dist/ directory)

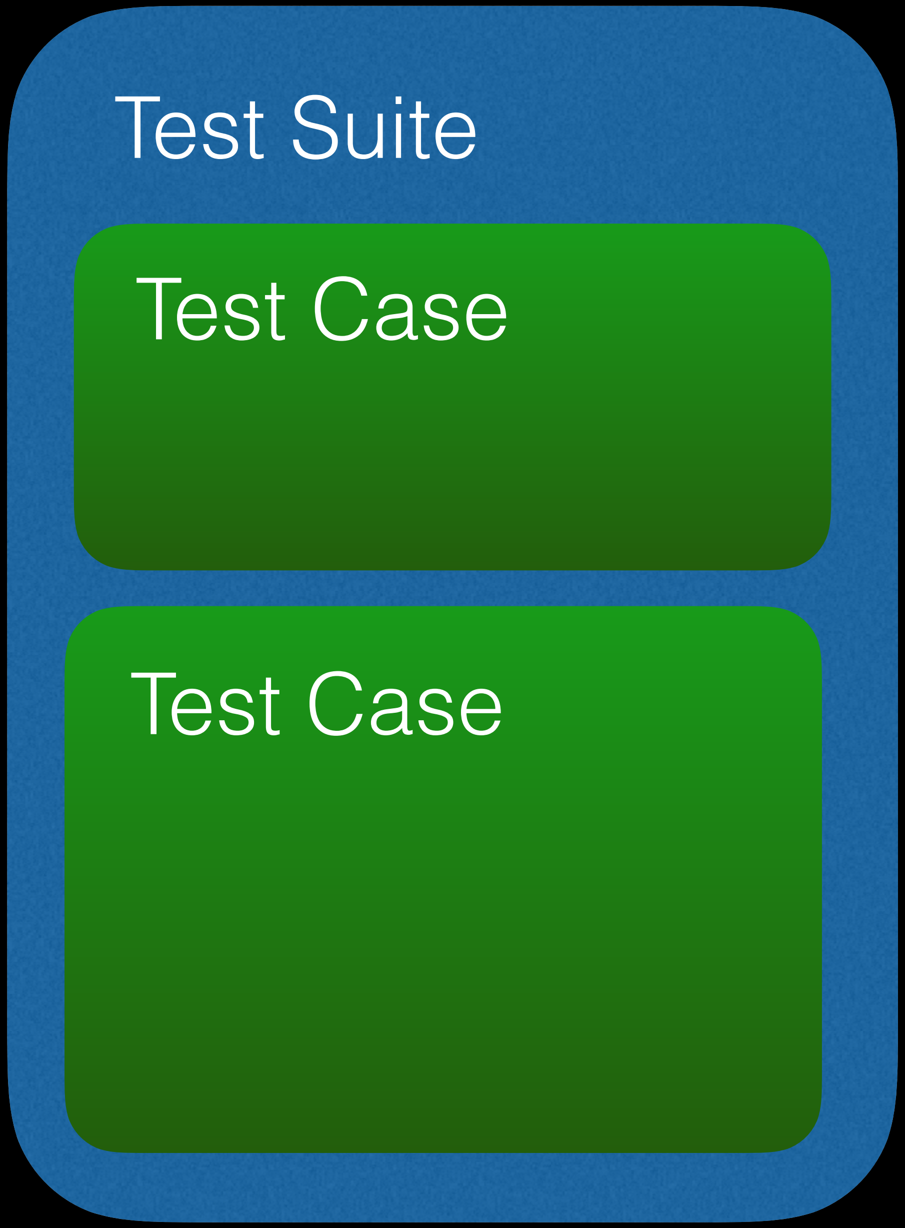


Anatomy of a Test Suite

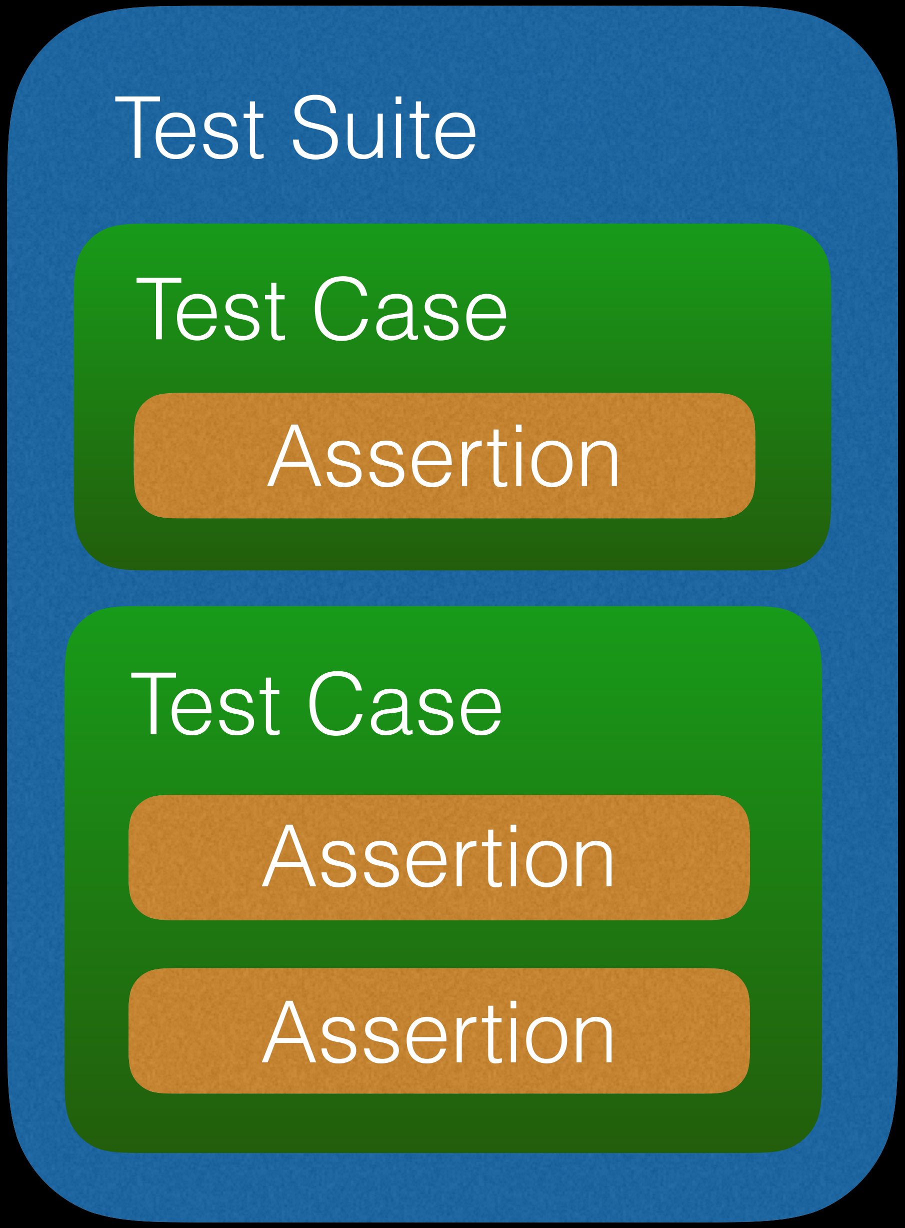
Anatomy of a Test Suite



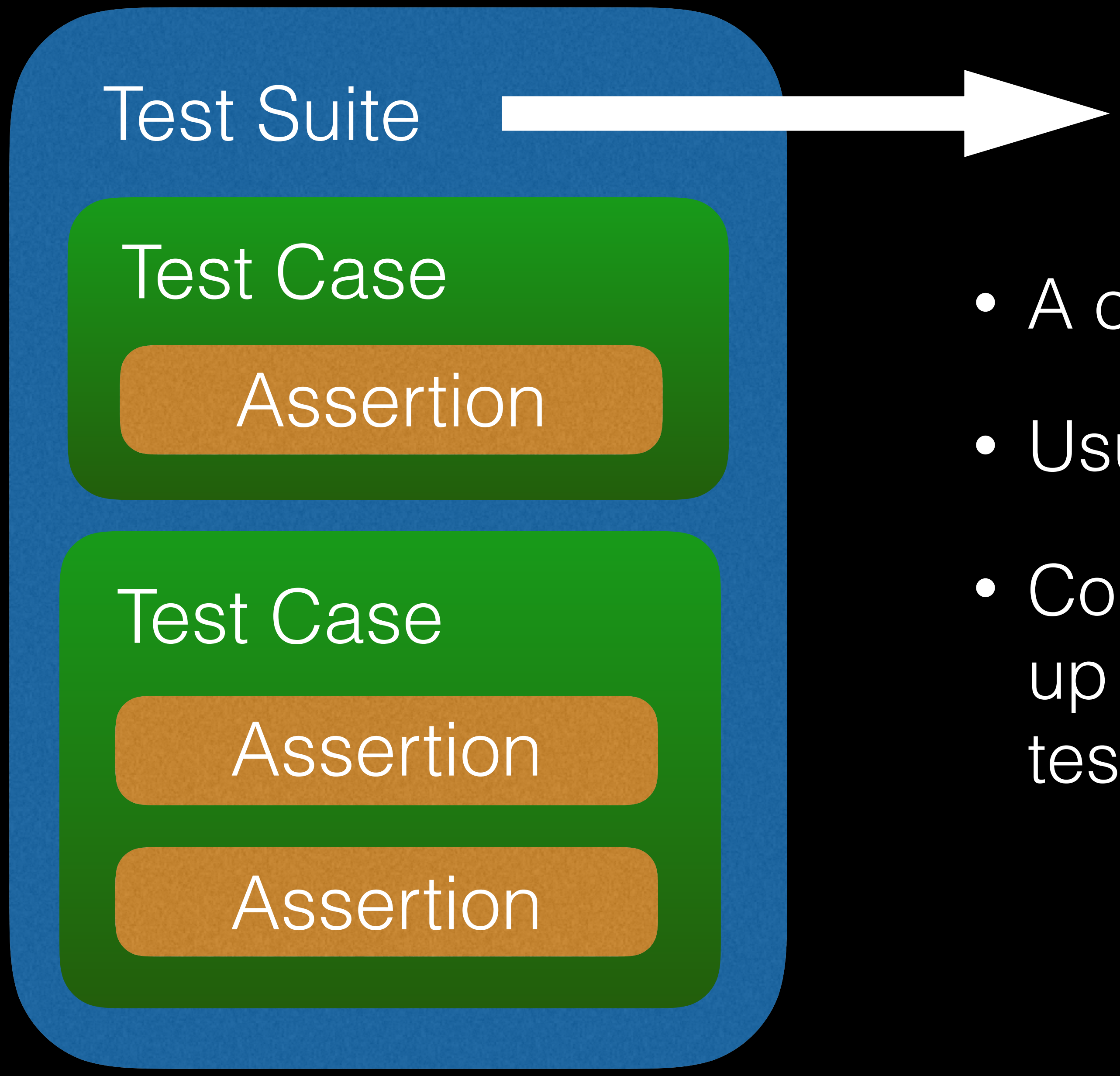
Anatomy of a Test Suite



Anatomy of a Test Suite



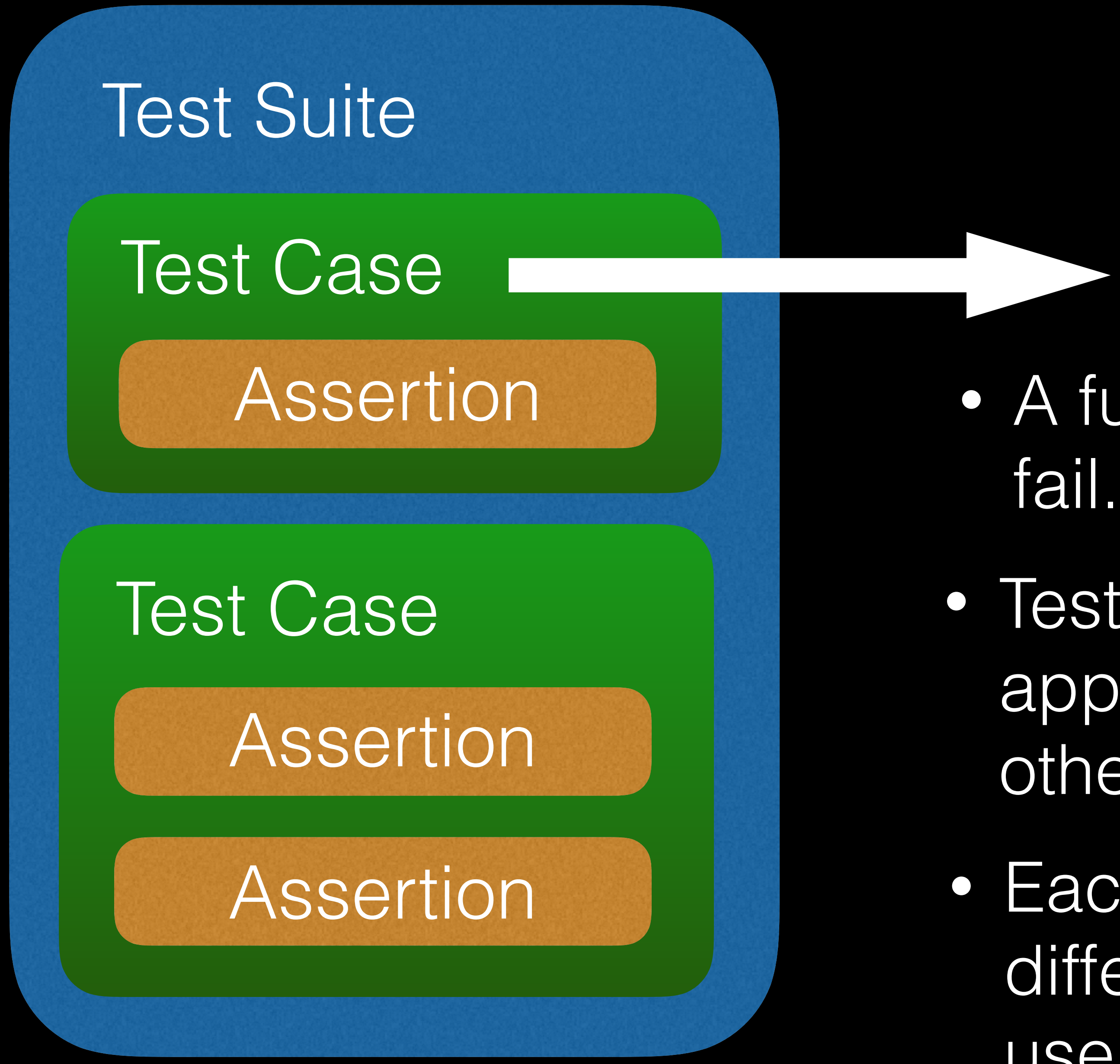
Anatomy of a Test Suite



Test Suite

- A collection of independent tests.
- Usually exists as it’s own file.
- Contains methods for setting up for and tearing down unit tests.

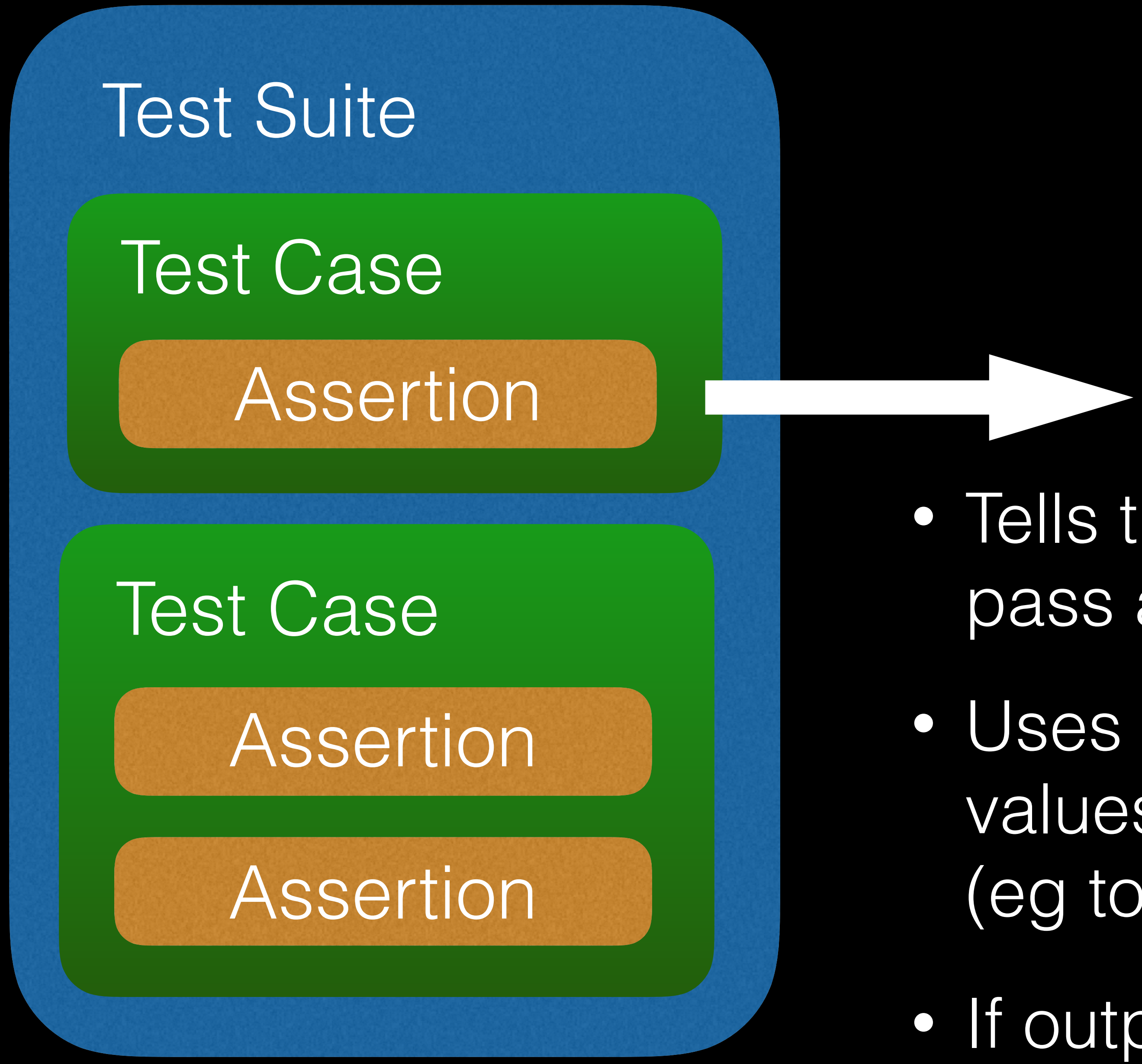
Anatomy of a Test Suite



Test Case

- A function that can either pass or fail.
- Tests a single “piece” of your application independent of the other code.
- Each case should test a different “situation” from the user’s perspective (BDD).

Anatomy of a Test Suite



Assertion

- Tells the test case when it should pass and when it should fail.
- Uses a matcher API for comparing values
  (eg toEqual)
- If output values for SUT (system under test) are as expected then behavior of SUT is as expected.

Unit testing in JavaScript

uses a peculiar syntax…

If you are ever feeling lost, just remember that a test suite in JavaScript is this:

A **describe**

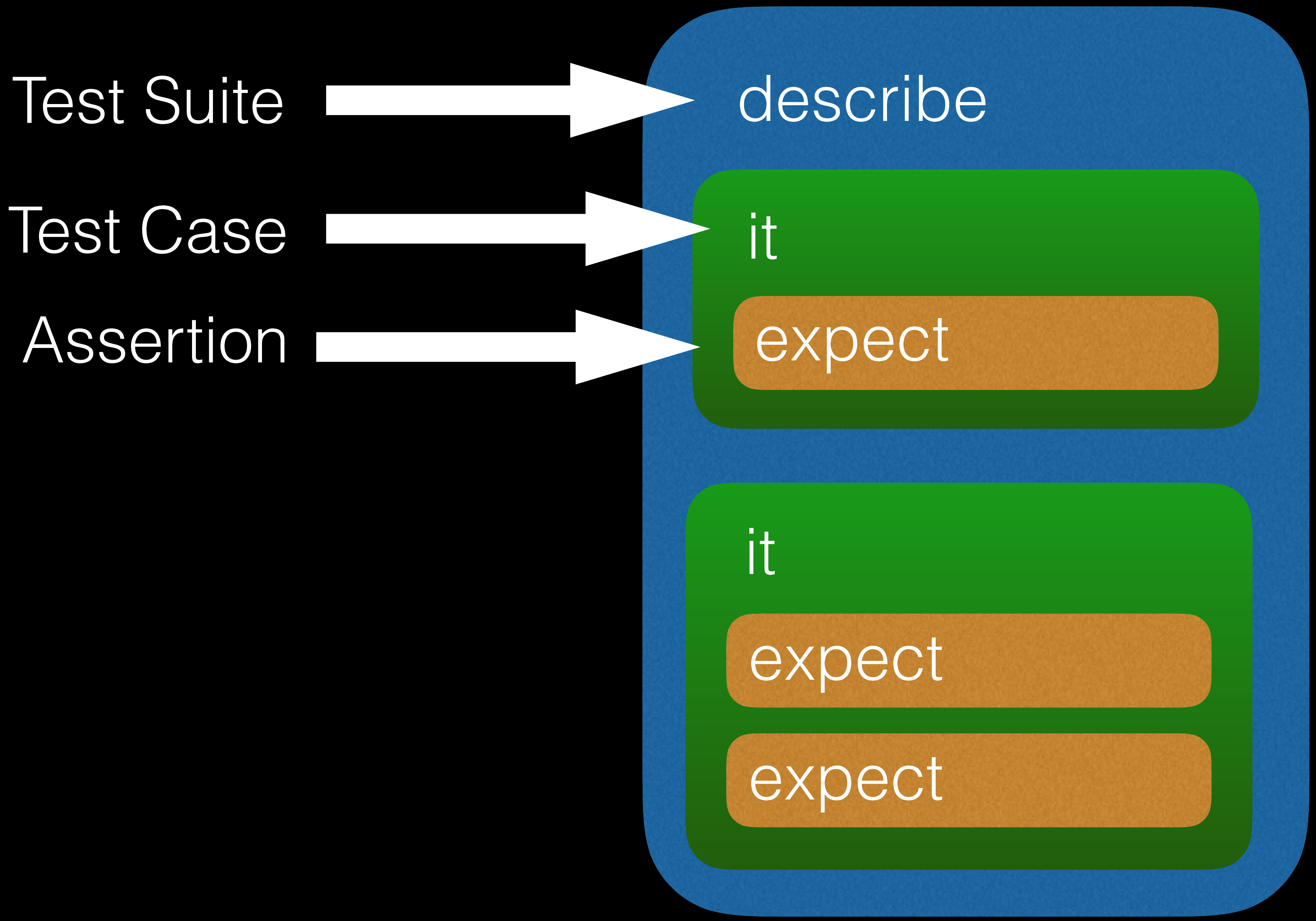
full of

it’s

with some

beforeEach’s

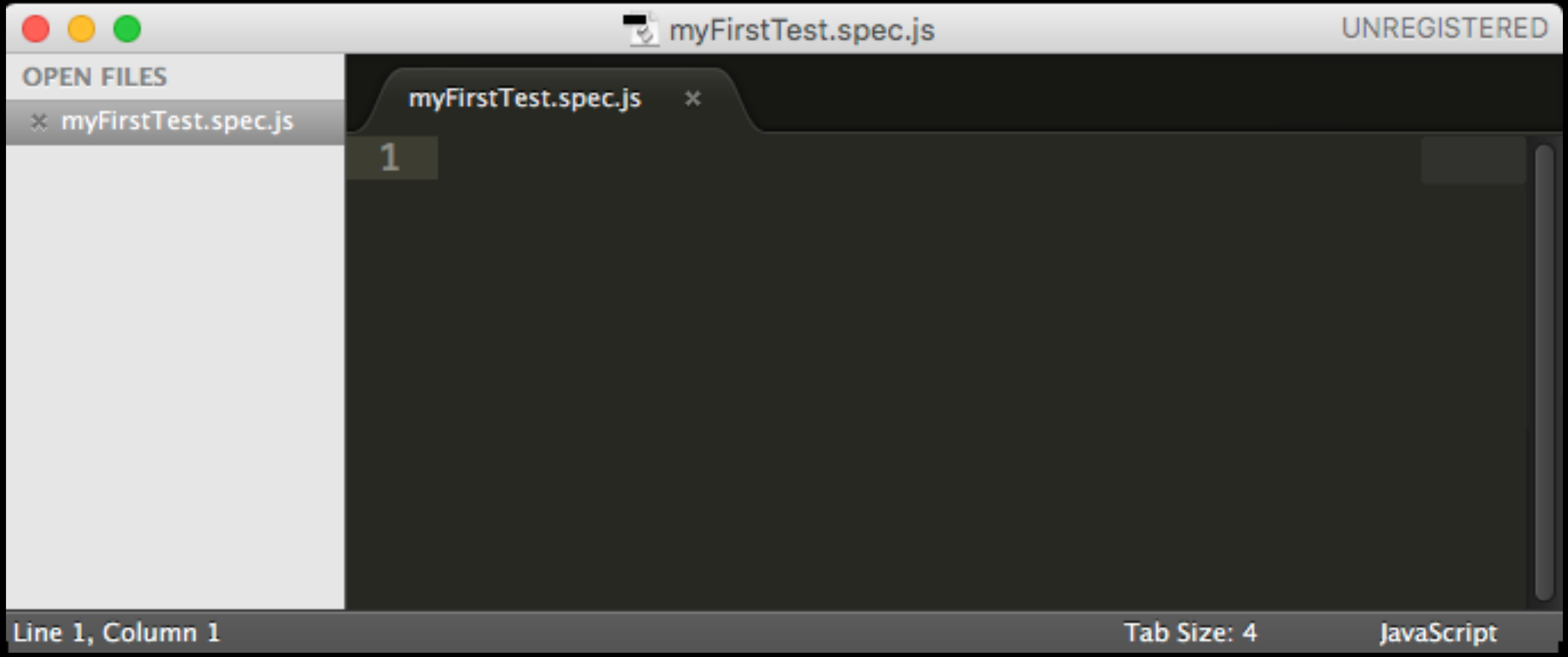
Test Suite Anatomy for JS Testing Frameworks



Building Your First Test Suite

- A test suite is simply a Javascript file.
- Karma will automatically consider *.spec.js files to be test suites.

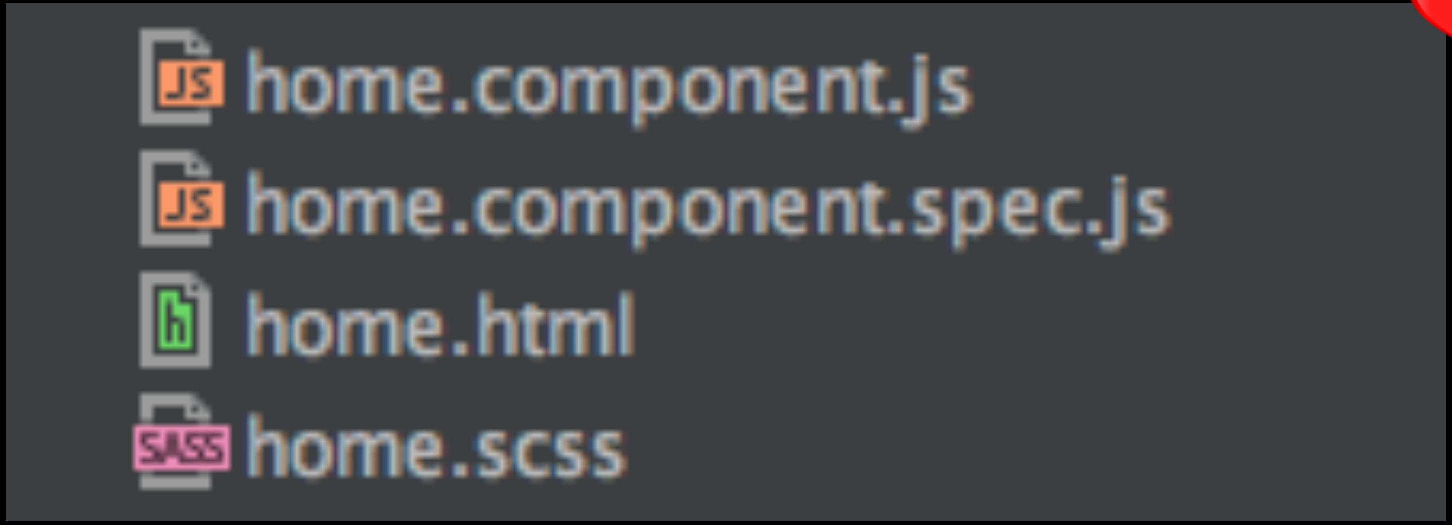
Step 1) Create an empty *.spec.js file.



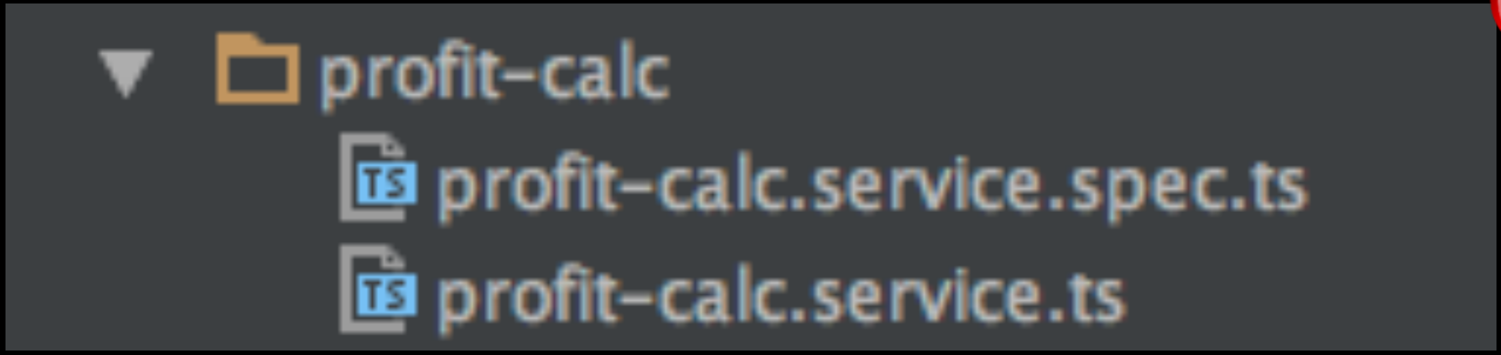
Keep the Tests Close By

- For every .js file, make a .spec.js file right next to it.
- Test file should have exact same name
  (other than the .spec part).
- Gulp scripts should recognize tests throughout entire project directory.
- Having a root level “tests” folder is an old-school practice and not recommended.

Keep the Family Together



```
home.component.js
home.component.spec.js
home.html
home.scss
```



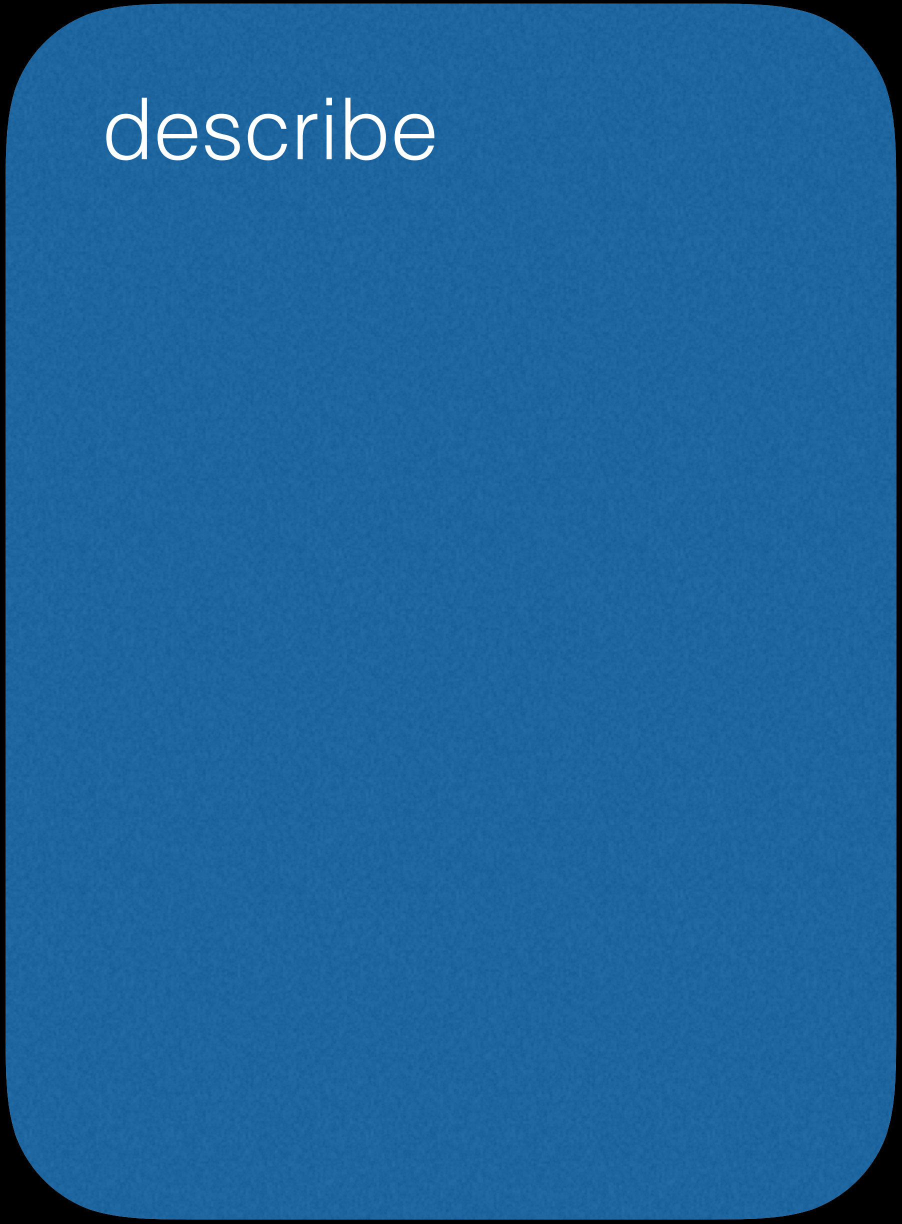
```
▼ profit-calc
    profit-calc.service.spec.ts
    profit-calc.service.ts
```

Two Lovebirds

Let’s Look At Some Code…

Building Your First Test Suite

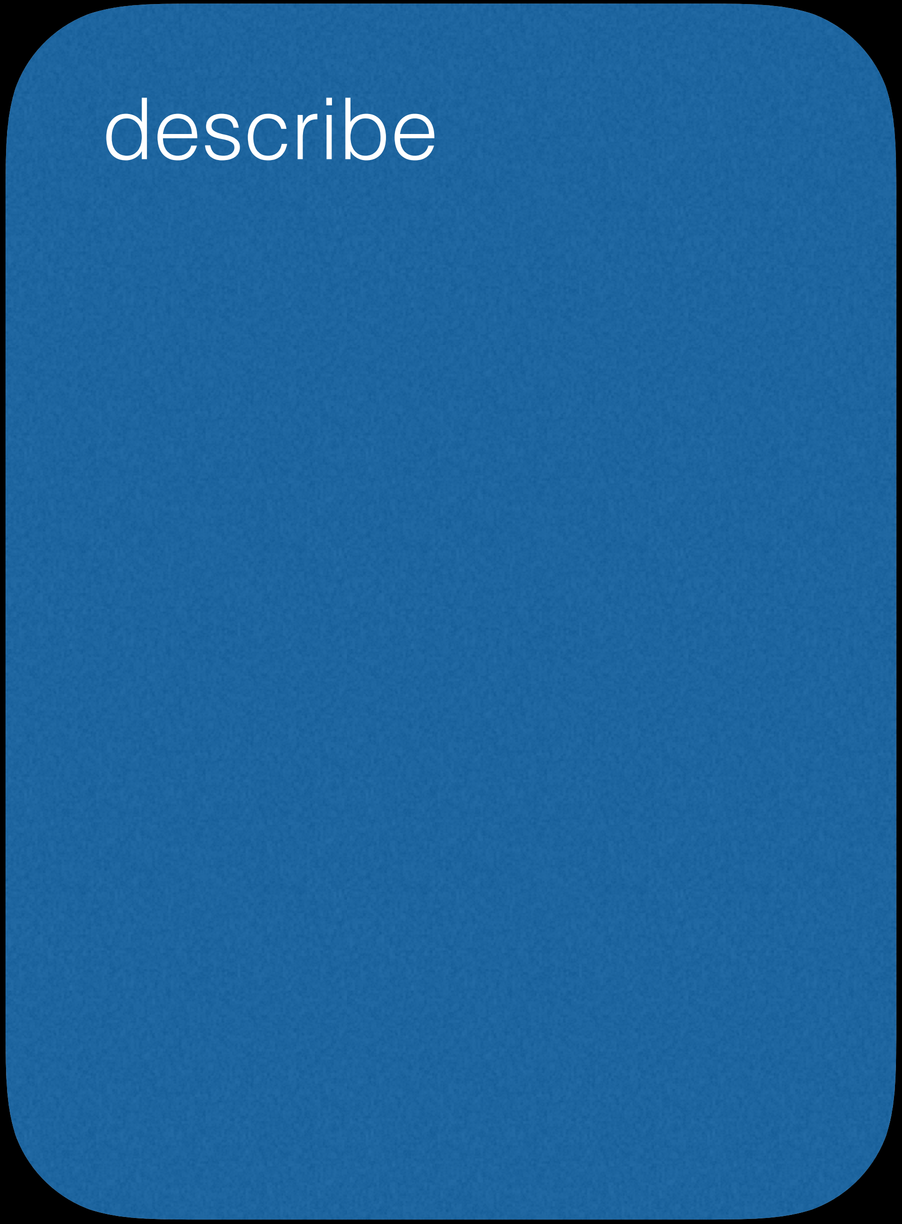
Adding a describe.



```
describe('MyController', function() {

})
```

Building Your First Test Suite

Adding a describe.

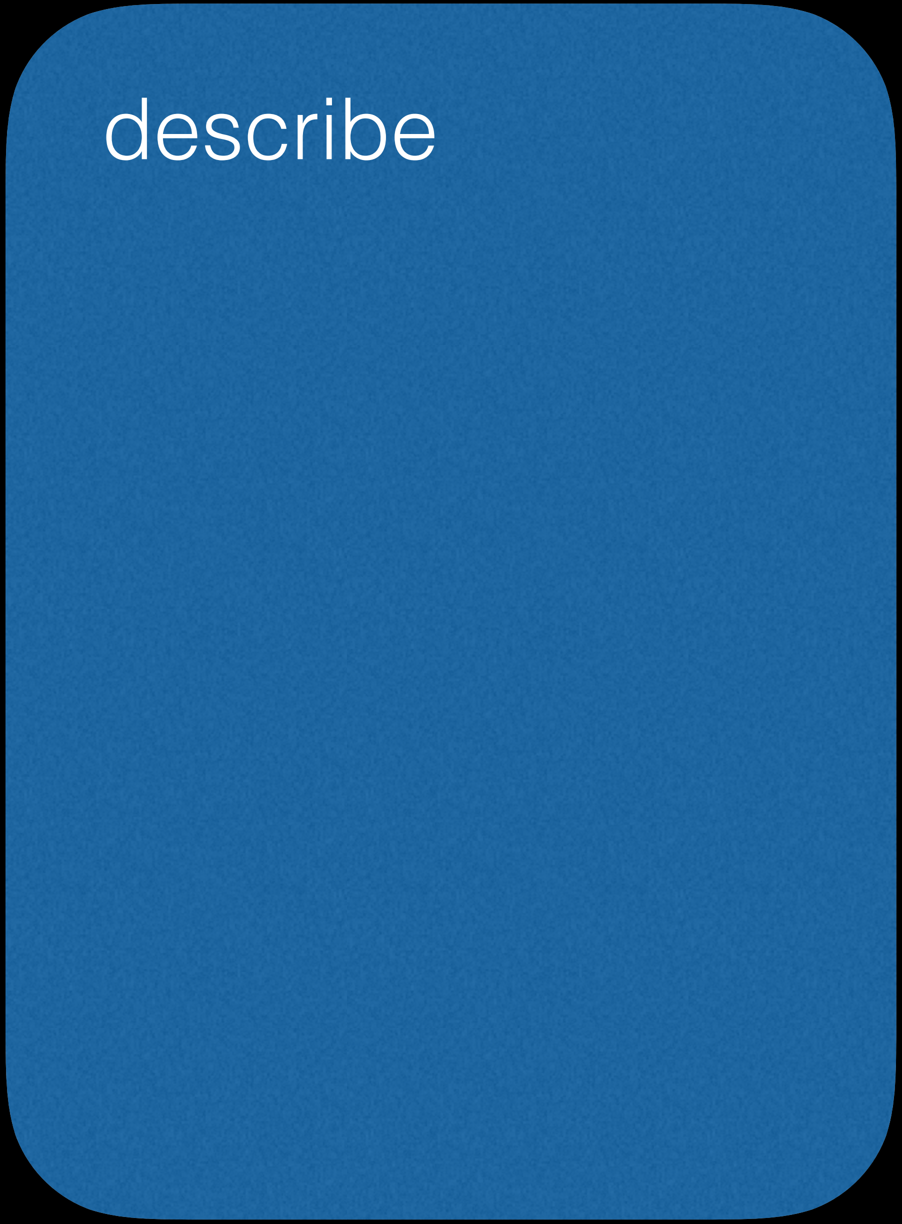


```
describe('MyController', function() {

})
```

A name for your test suite (can be anything, but it should *describe* what you are testing!).

Building Your First Test Suite

Adding a describe.

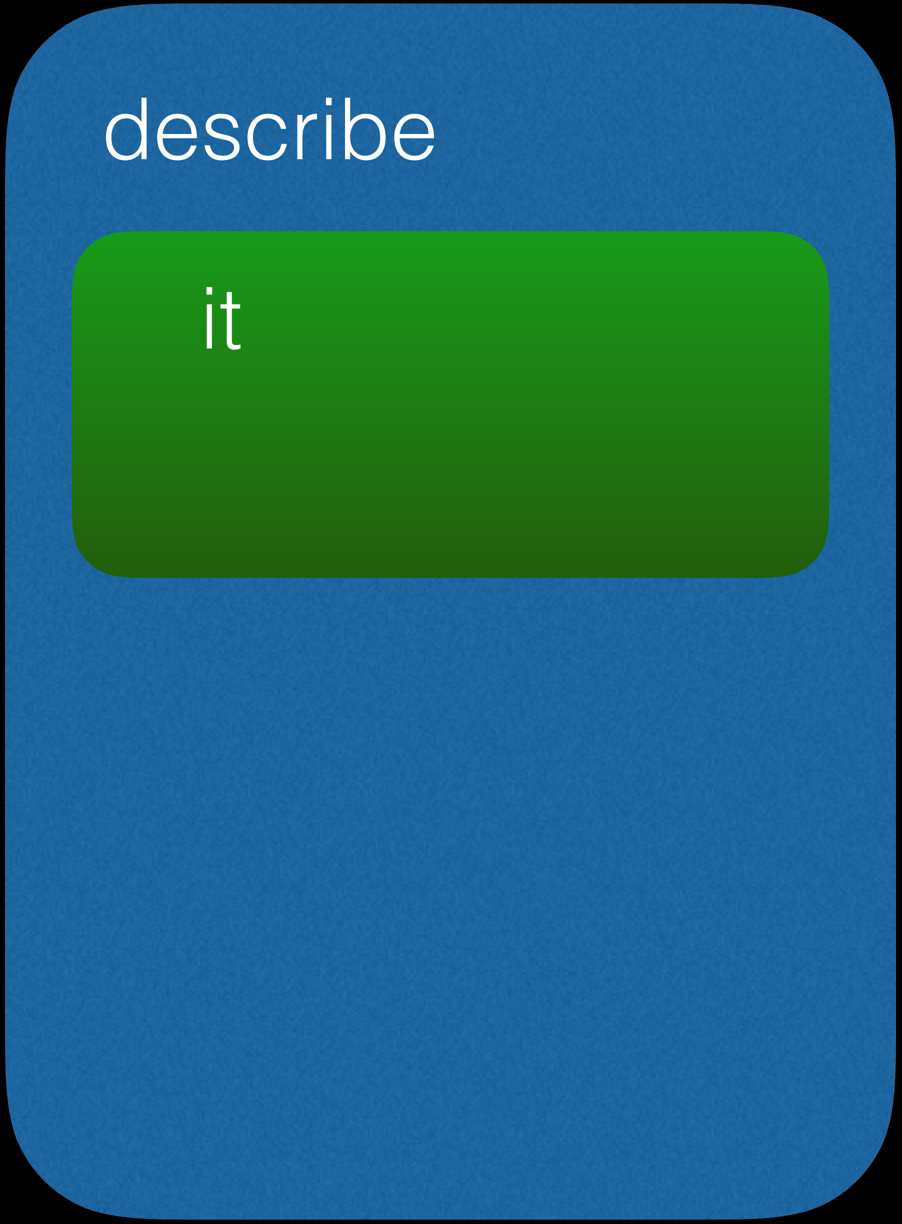


```
describe('MyController', function() {

})
```

A function that takes no arguments. This creates the "wrapper" around your test cases.

Building Your First Test Suite

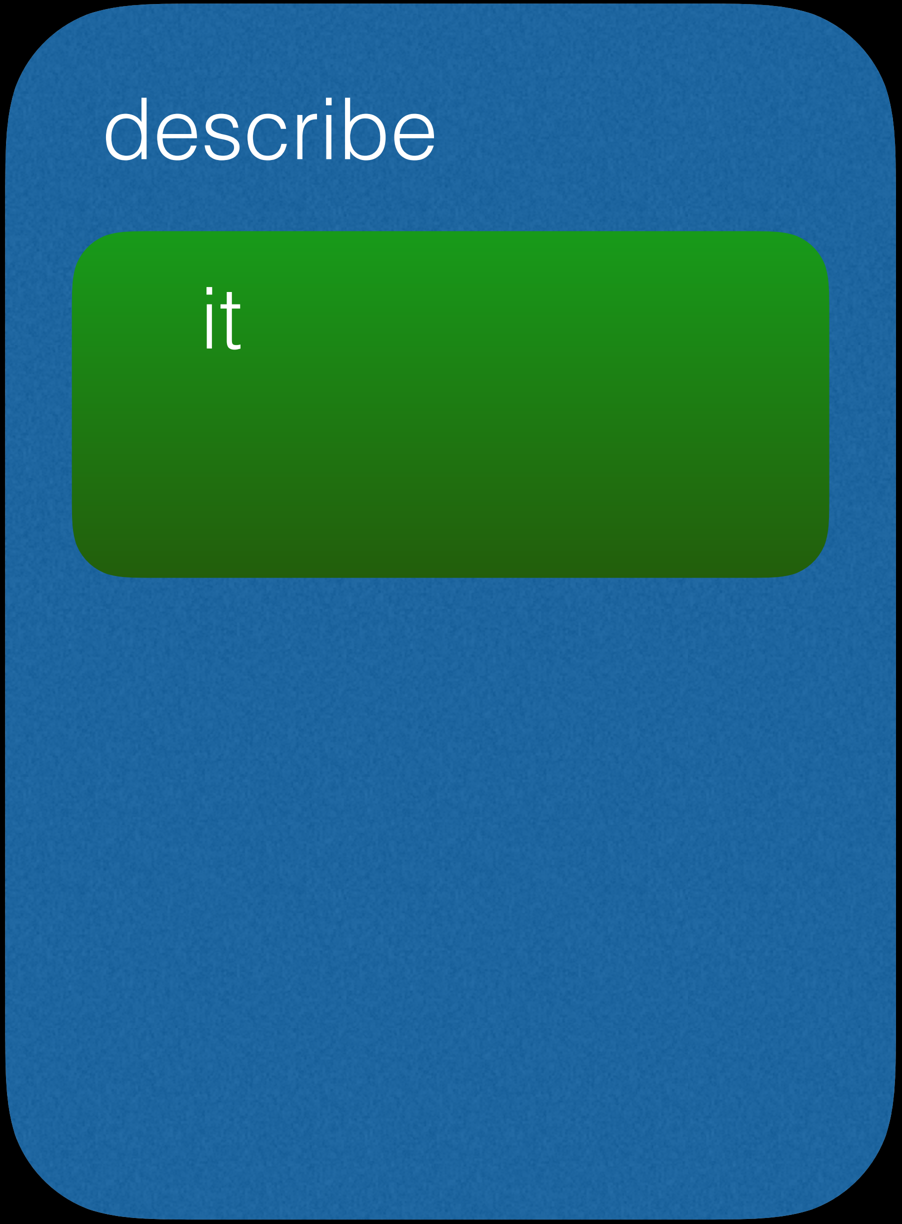
Adding an it.



```
describe('MyController', function() {
    it('Should do something…', function() {

    });

})
```

Building Your First Test Suite

Adding an it.

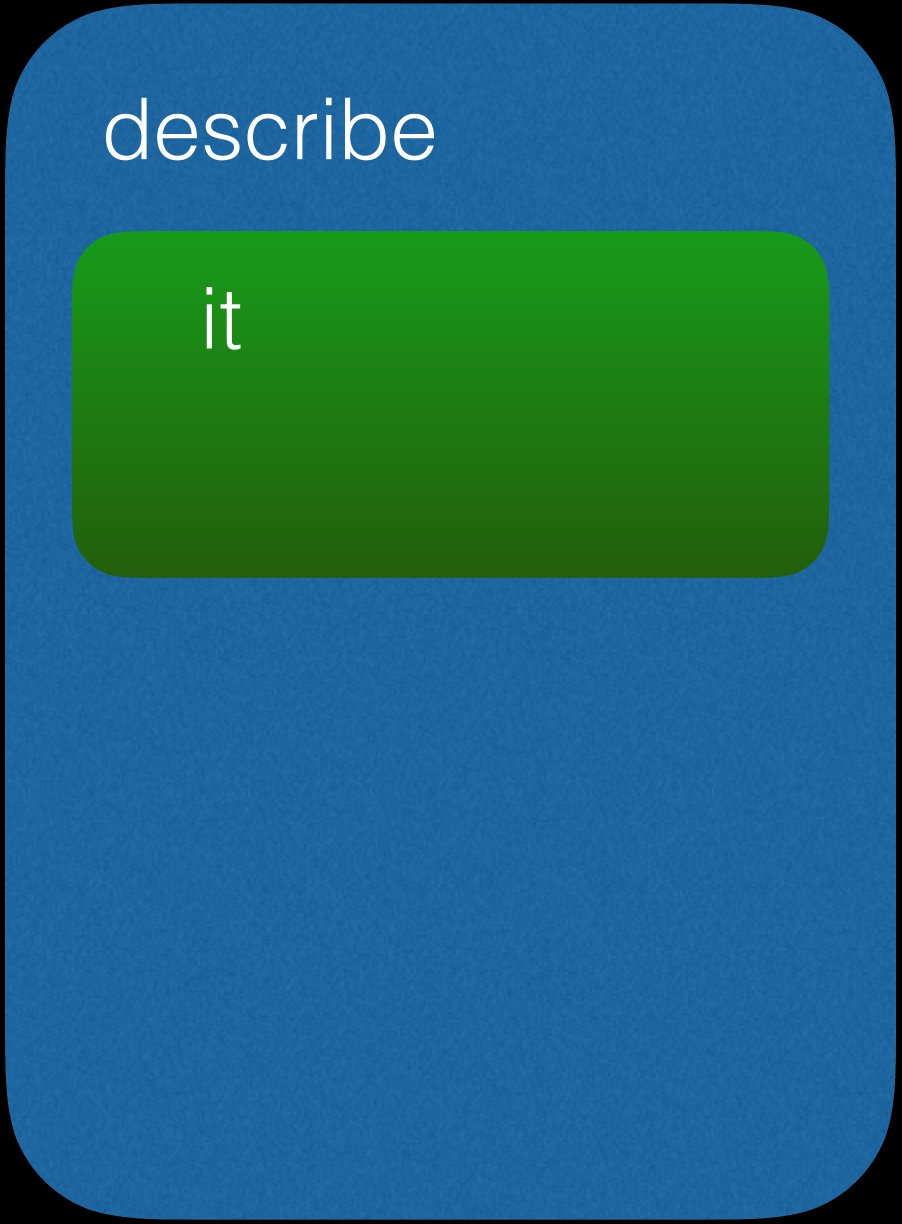


```
describe('MyController', function() {
    it('Should do something…', function() {

    });

})
```

Some text that describes the purpose of this test case. Can be anything but usually begins with the word *should.*

Building Your First Test Suite

Adding an it.

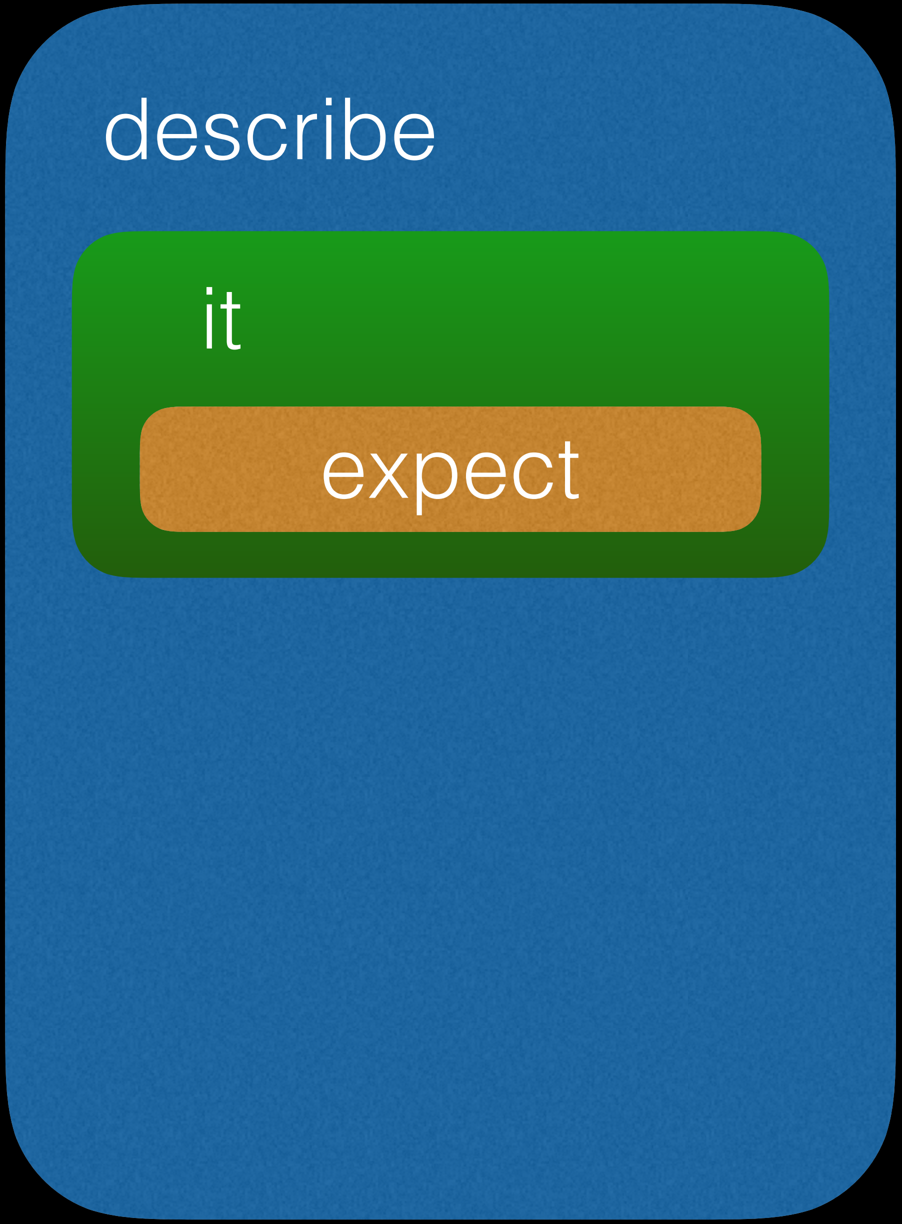


```
describe('MyController', function() {
    it('Should do something…', function() {

    });

})
```

💡 A function that takes no arguments. The code for this test case is contained in this function.

Building Your First Test Suite

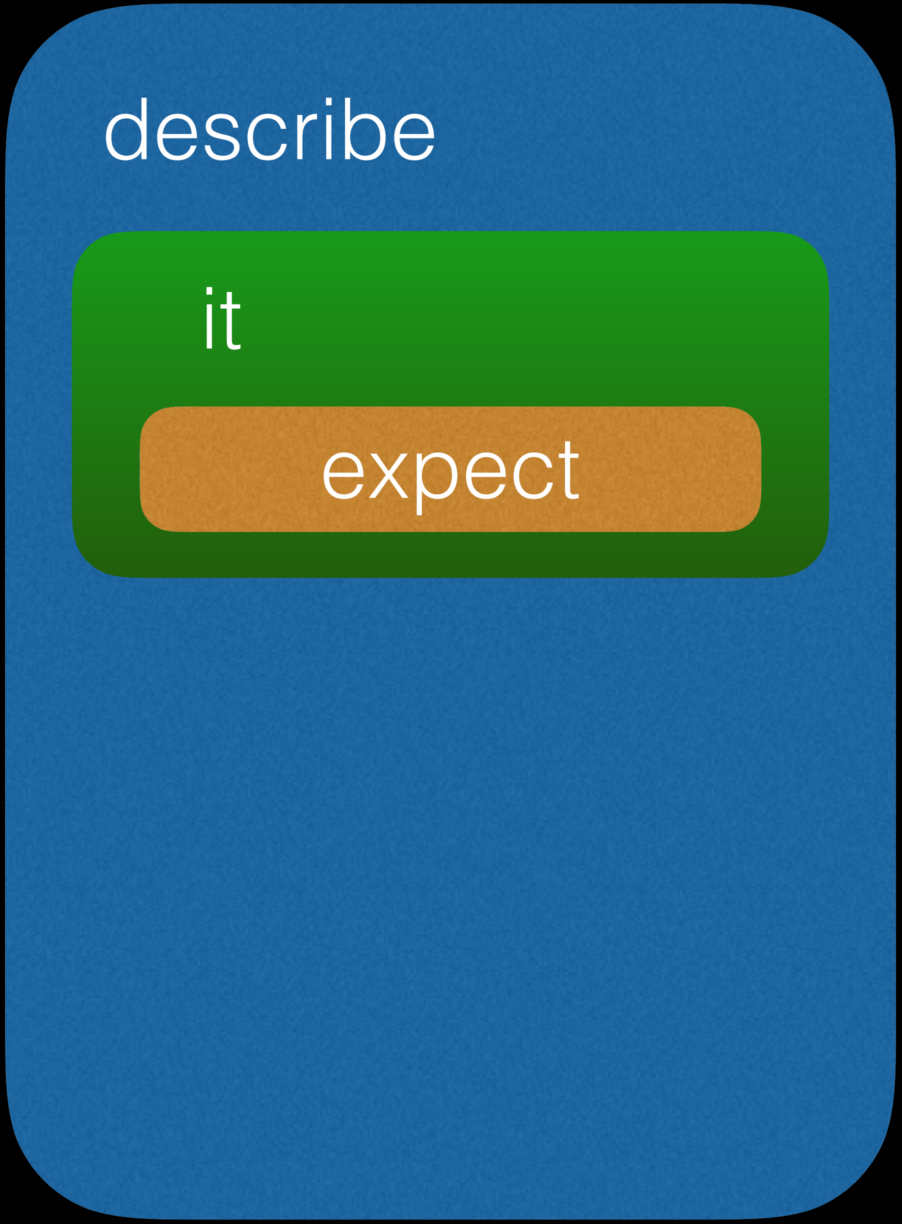
Adding an assertion.



```
describe('MyController', function() {
    it('Should do something…', function() {
        expect(true).toEqual(true);
    });
})
```

Building Your First Test Suite

Adding an assertion.

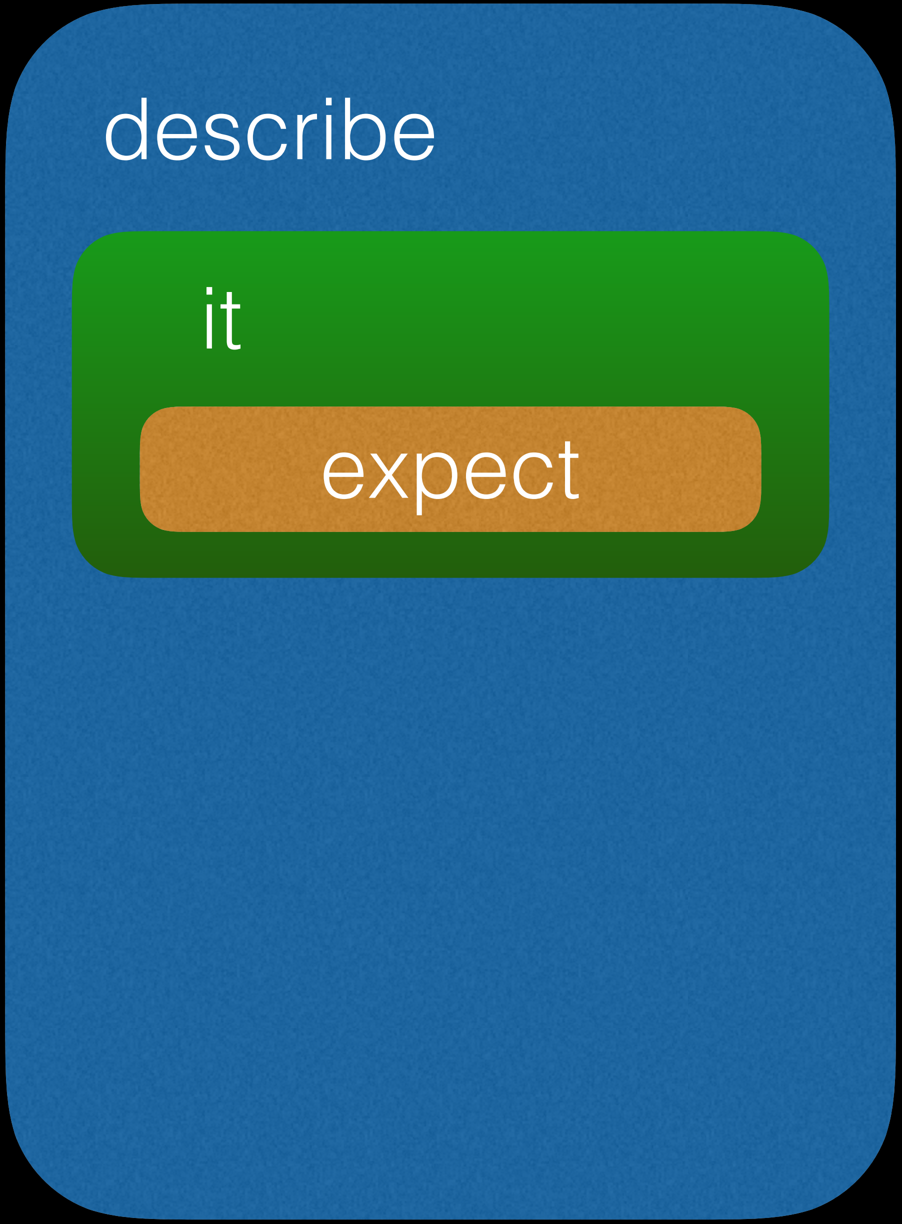


```
describe('MyController', function() {
    it('Should do something…', function() {
        expect(true).toEqual(true);
    });
})
```

The expect keyword let's the test case know that we want to do an assertion here.

Building Your First Test Suite

Adding an assertion.

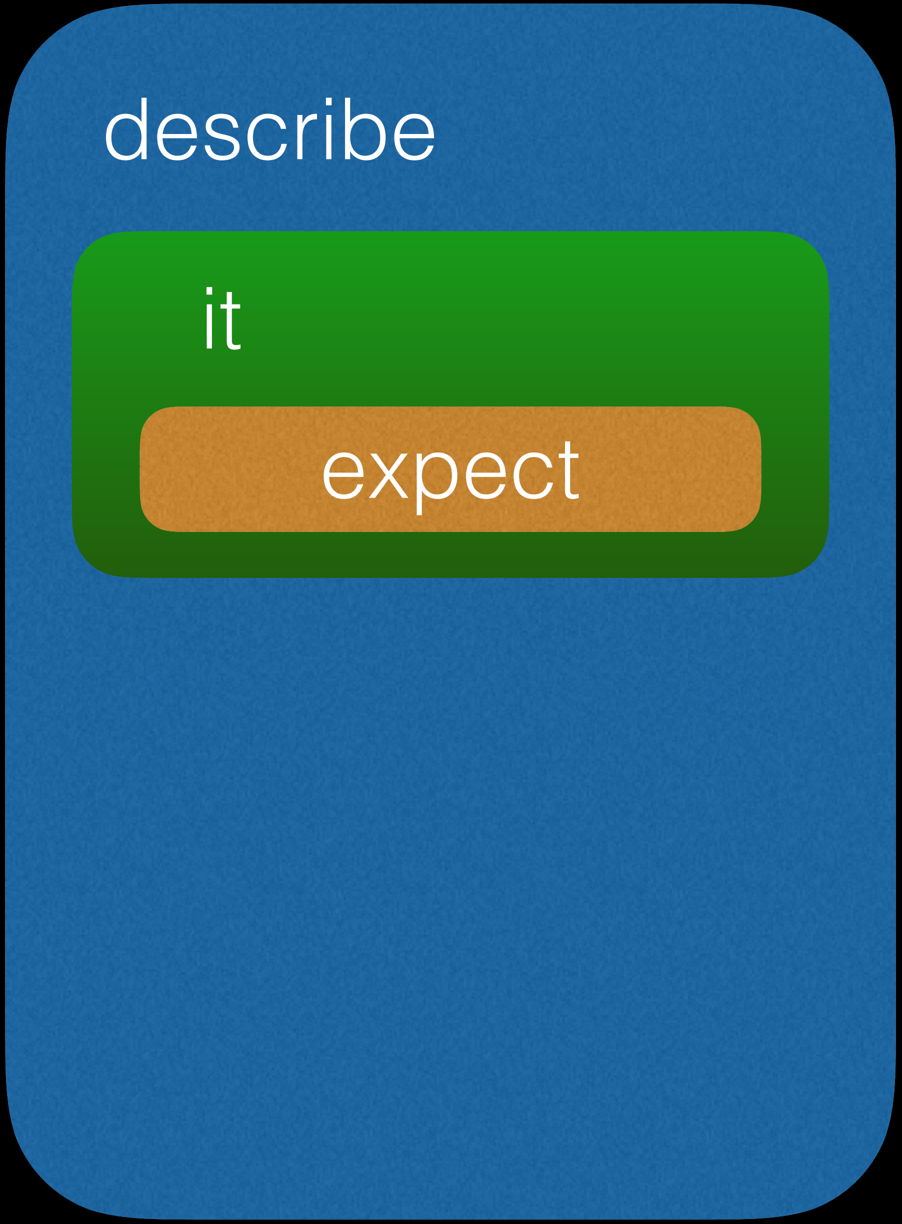


```
describe('MyController', function() {
    it('Should do something…', function() {
        expect(true).toEqual(true);
    });
})
```

The expect method takes one argument, the variable whose value you wish to check.

Building Your First Test Suite

Adding an assertion.

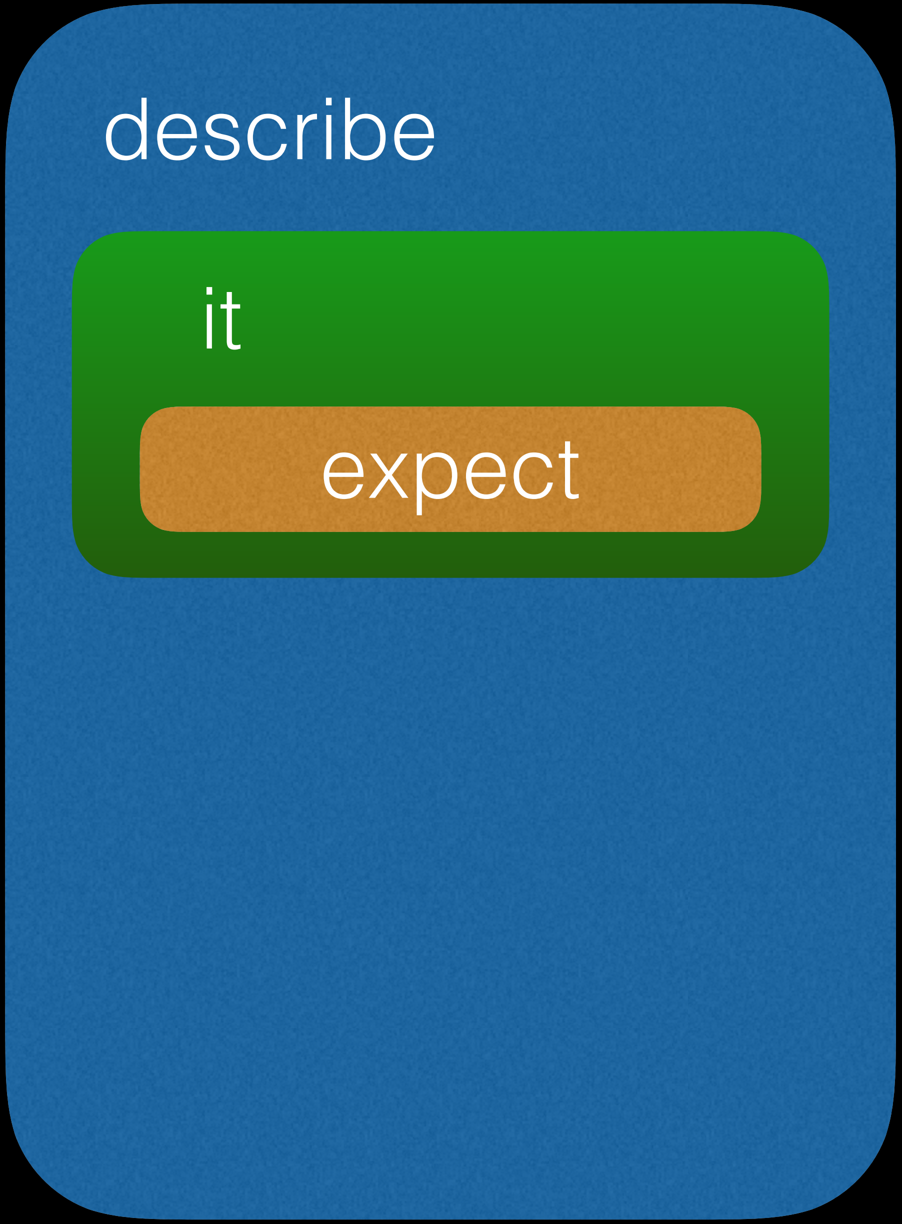


```
describe('MyController', function() {
    it('Should do something…', function() {
        expect(true).toEqual(true);
    });
})
```

Depending on how you wish to compare the two values, a *matcher* method is chained onto the end of the expect.

Building Your First Test Suite

Adding an assertion.



```
describe('MyController', function() {
    it('Should do something…', function() {
        expect(true).toEqual(true);
    });
})
```

The matcher method takes one argument. The expected value for the variable being passed into the expect method.

Building Your First Test Suite

You did it!

```
describe('MyController', function() {
    it('Should do something…', function() {
        expect(true).toEqual(true);
    });
})
```

Ahhh, so a test suite is really just…

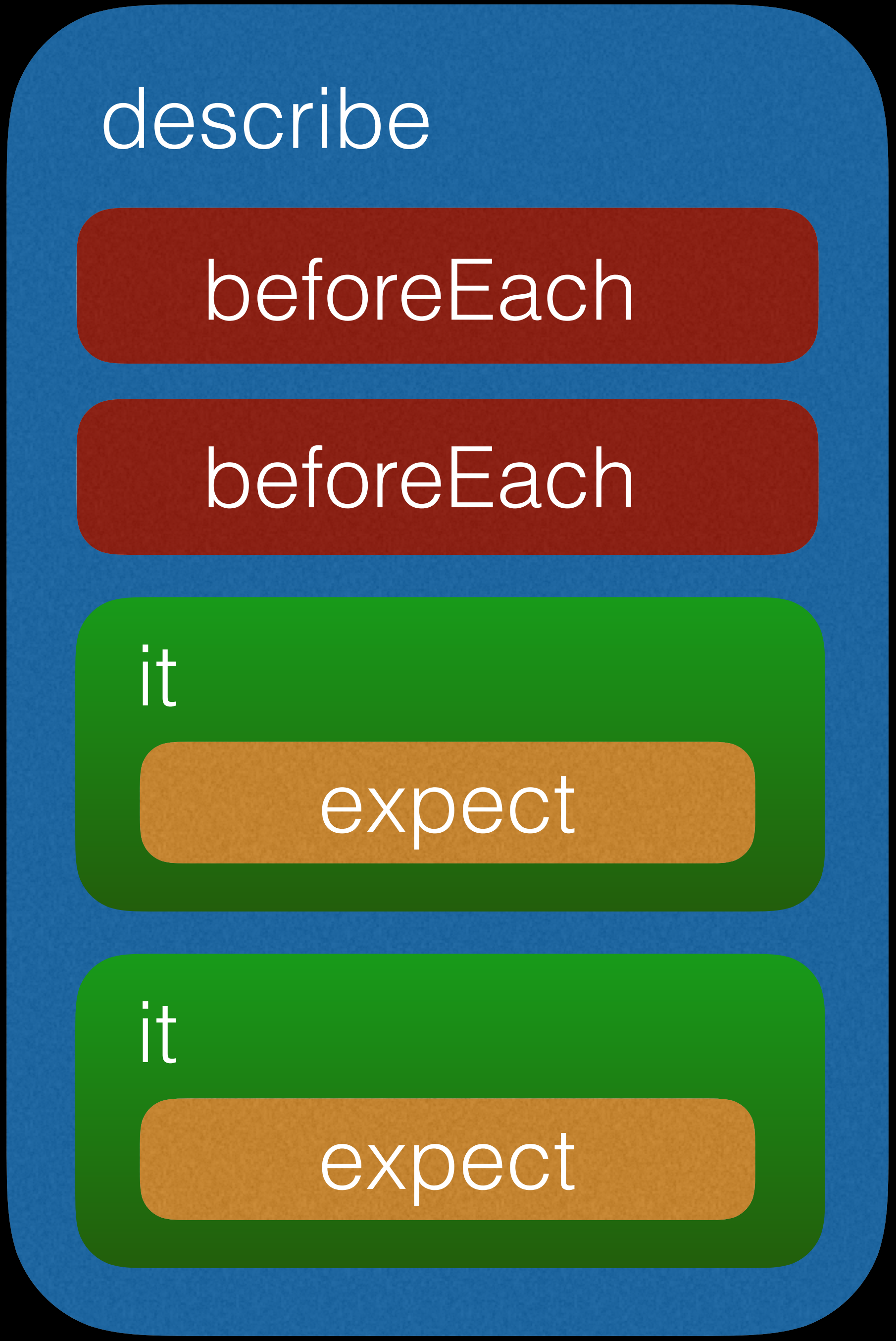
A **describe**

full of

it’s!

…with some beforeEach’s.

beforeEach



- Goes inside the describe but outside of the it’s.
- Gives you access to your module, controllers, services, etc. through DI.

beforeEach

```
describe(‘MyController’, function() {
  beforeEach(module(‘YOUR_MODULE’));
  it(‘Should do something…’, function() {
      expect(true).toEqual(true);
  });
})
```

beforeEach

```
beforeEach(module('YOUR_MODULE'));
```

Keyword that runs argument
before every *it.*

beforeEach

```
beforeEach(module('YOUR_MODULE'));
```

Allows you to load in your model so that you have access to it's controllers, services, filters, etc.

beforeEach

```
beforeEach(module(‘YOUR_MODULE’));
```

Replace this with the name of your project’s module.

Injecting a Controller with beforeEach

```
describe(‘MyController’, function() {

    beforeEach(module(‘YOUR_MODULE’));

    beforeEach(inject(function(_$controller_) {
        $controller = _$controller_;
    }));

    it(‘Should do something…’, function() {

        var myController = $controller('MyController', {})
        expect(true).toEqual(true);
    });

})
```

Injecting a Controller with beforeEach

A method from the angular-mocks.js file that allows you to inject services into your unit tests.

↓

```
beforeEach(inject(function(_$controller_) {
    $controller = _$controller_;
}));
```

Injecting a Controller with beforeEach

Angular knows to "unwrap", the underscores, find corresponding provider, and give you a reference to the service.

```
beforeEach(inject(function(_$controller_) {
    $controller = _$controller_;
}));
```

Q. But what's the deal with those underscores on either side?

Injecting a Controller with beforeEach

Angular knows to “unwrap”, the underscores, find corresponding provider, and give you a reference to the service.

```
beforeEach(inject(function(_$controller_) {
    $controller = _$controller_;
}));
```

Suppose you didn’t use the underscores. You want to set a variable named $controller available inside of your “it’s” equal to the function’s argument, but the function argument *must* be named $controller in order to be injected properly. Doing this is not possible in JavaScript (outer variable is overshadowed) so the Angular team implemented the underscore notation to work around the issue.

Injecting a Controller with beforeEach

```
beforeEach(inject(function(_$controller_) {
    $controller = _$controller_;
}));
```

You can then use this global reference anywhere in the test suite to instantiate controllers.

Using the Injected Controller

This var has all of the properties and methods you defined for the specified controller.

$\downarrow$

```
var myController = $controller('MyController', {})
```

Using the Injected Controller

💡 This is the global $controller variable that was set in the beforeEach.

↓

```
var myController = $controller('MyController', {})
```

Using the Injected Controller

Replace this with the name of the controller you want to instantiate.

```
var myController = $controller('MyController', {})
```

Using the Injected Controller

💡 Pass in any arguments to your controller with this object.

↓

```
var myController = $controller('MyController', {})
```

The Complete Suite

A good start to a nice looking test suite:

```
describe(‘MyController’, function() {

    beforeEach(module(‘YOUR_MODULE’));

    beforeEach(inject(function(_$controller_) {
        $controller = _$controller_;
    }));

    it(‘Should do something…’, function() {

        var myController = $controller('MyController', {})
        expect(true).toEqual(true);
    });

})
```

Q) Okay, so how do I run these test suites?

A) Karma

Fun Facts About Karma

- A command line test runner built to be *fast.*
- Runs tests on all browsers (even PhantomJS).
- Integrates with practically all CI tools.
- Worked with non-Angular projects as well.

How Does Karma Work?

- It’s installed from npm: *npm install karma*
- The karma.conf.js file allows you to configure it to run with your desired settings.
- It automatically see *.spec.js files in your project folder as test suites.
- It integrates nicely with Gulp and Grunt (gulp test) or runs on its own (karma start).

How do I add Karma to My Project?

Adding Karma to Your Project

Easy Way

Use a yeoman generator to scaffold a project that already has karma set up for you (such as the Gulp-Angular yeoman generator or the Angular 2 CLI).

Hard(er) Way

Install and configure it manually.

karma-runner.github.io

And then you're ready to start testing!

Workflow

Gulp Serve

Browsersync

Chrome Dev Tools

See Your App Running

Logs / Debugging

Gulp Test:Auto

Runs Unit Tests on File Changes

No need to test manually

Reminds you to write more tests

Workflow

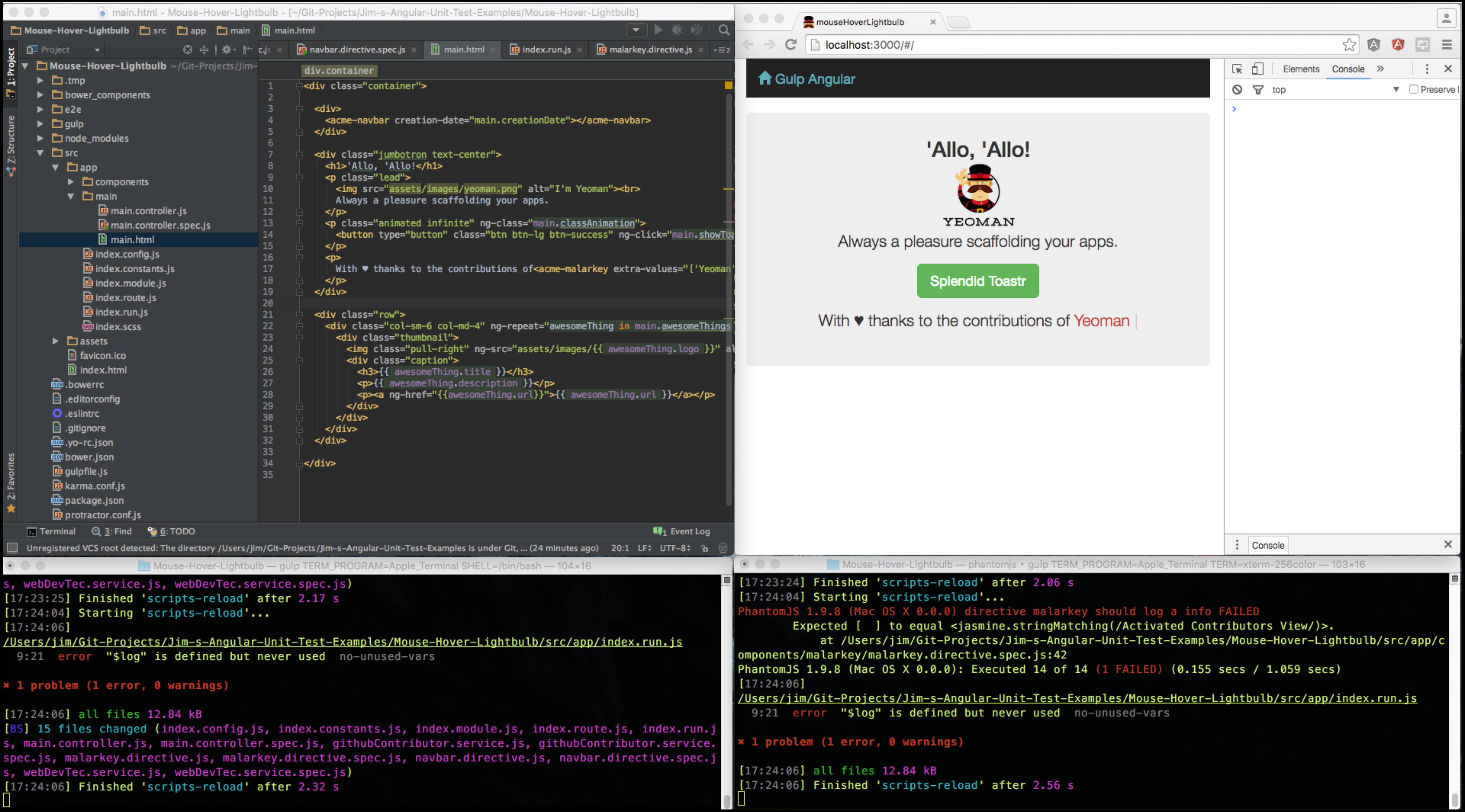
Gulp Serve or **Gulp Test:Auto**

???

Workflow

Gulp Serve & **Gulp Test:Auto**

“Dual Shell Development”



```
<div class="container">

  <div>
    <acme-navbar creation-date="main.creationDate"></acme-navbar>
  </div>

  <div class="jumbotron text-center">
    <h1>'Allo, 'Allo!</h1>
    <p class="lead">
      <img src="assets/images/yeoman.png" alt="I'm Yeoman"><br>
      Always a pleasure scaffolding your apps.
    </p>
    <p class="animated infinite" ng-class="main.classAnimation">
      <button type="button" class="btn btn-lg btn-success" ng-click="main.showToa
    </p>
    <p>
      With ♥ thanks to the contributions of<acme-malarkey extra-values="['Yeoman'
    </p>
  </div>

  <div class="row">
    <div class="col-sm-6 col-md-4" ng-repeat="awesomeThing in main.awesomeThings"
      <div class="thumbnail">
        <img class="pull-right" ng-src="assets/images/{{ awesomeThing.logo }}" al
        <div class="caption">
          <h3>{{ awesomeThing.title }}</h3>
          <p>{{ awesomeThing.description }}</p>
          <p><a ng-href="{{awesomeThing.url}}">{{ awesomeThing.url }}</a></p>
        </div>
      </div>
    </div>
  </div>

</div>
```

```
Unregistered VCS root detected: The directory /Users/jim/Git-Projects/Jim-s-Angular-Unit-Test-Examples is under Git, ... (24 minutes ago)   20:1  LF÷  UTF-8÷
```

```
Mouse-Hover-Lightbulb — gulp TERM_PROGRAM=Apple_Terminal SHELL=/bin/bash — 104×16
s, webDevTec.service.js, webDevTec.service.spec.js)
[17:23:25] Finished 'scripts-reload' after 2.17 s
[17:24:04] Starting 'scripts-reload'...
[17:24:06]
/Users/jim/Git-Projects/Jim-s-Angular-Unit-Test-Examples/Mouse-Hover-Lightbulb/src/app/index.run.js
  9:21  error  "$log" is defined but never used  no-unused-vars

× 1 problem (1 error, 0 warnings)

[17:24:06] all files 12.84 kB
[BS] 15 files changed (index.config.js, index.constants.js, index.module.js, index.route.js, index.run.j
s, main.controller.js, main.controller.spec.js, githubContributor.service.js, githubContributor.service.
spec.js, malarkey.directive.js, malarkey.directive.spec.js, navbar.directive.js, navbar.directive.spec.j
s, webDevTec.service.js, webDevTec.service.spec.js)
[17:24:06] Finished 'scripts-reload' after 2.32 s
```

```
Mouse-Hover-Lightbulb — phantomjs • gulp TERM_PROGRAM=Apple_Terminal TERM=xterm-256color — 103×16
[17:23:24] Finished 'scripts-reload' after 2.06 s
[17:24:04] Starting 'scripts-reload'...
PhantomJS 1.9.8 (Mac OS X 0.0.0) directive malarkey should log a info FAILED
        Expected [  ] to equal <jasmine.stringMatching(/Activated Contributors View/)>.
            at /Users/jim/Git-Projects/Jim-s-Angular-Unit-Test-Examples/Mouse-Hover-Lightbulb/src/app/c
omponents/malarkey/malarkey.directive.spec.js:42
PhantomJS 1.9.8 (Mac OS X 0.0.0): Executed 14 of 14 (1 FAILED) (0.155 secs / 1.059 secs)
[17:24:06]
/Users/jim/Git-Projects/Jim-s-Angular-Unit-Test-Examples/Mouse-Hover-Lightbulb/src/app/index.run.js
  9:21  error  "$log" is defined but never used  no-unused-vars

× 1 problem (1 error, 0 warnings)

[17:24:06] all files 12.84 kB
[17:24:06] Finished 'scripts-reload' after 2.56 s
```

Gulp serve in one command shell and gulp test:auto in the other.

So What's Different for Unit Testing Angular 2?

Not Much!

The CLI Replaces Gulp Tasks

https://cli.angular.io/

	Angular 1		Angular 2
New project:	`yo gulp-angular`	→	`ng new / ng init`
Run locally:	`gulp serve`	→	`ng serve`
Run prod build:	`gulp serve:dist`	→	`ng serve --build`
Run unit tests	`gulp test:auto`	→	`ng test -w`
Run e2e tests	`gulp protractor`	→	`ng e2e`
Create prod build	`gulp build`	→	`ng build`

In Angular 2 You Can Still Think of a Unit Test as…

A **describe**

full of

it’s

with some

beforeEach’s

A **describe**

full of

it’s

with some

beforeEach’s

(with a bunch of “import” statements)

A **describe**

full of

it’s

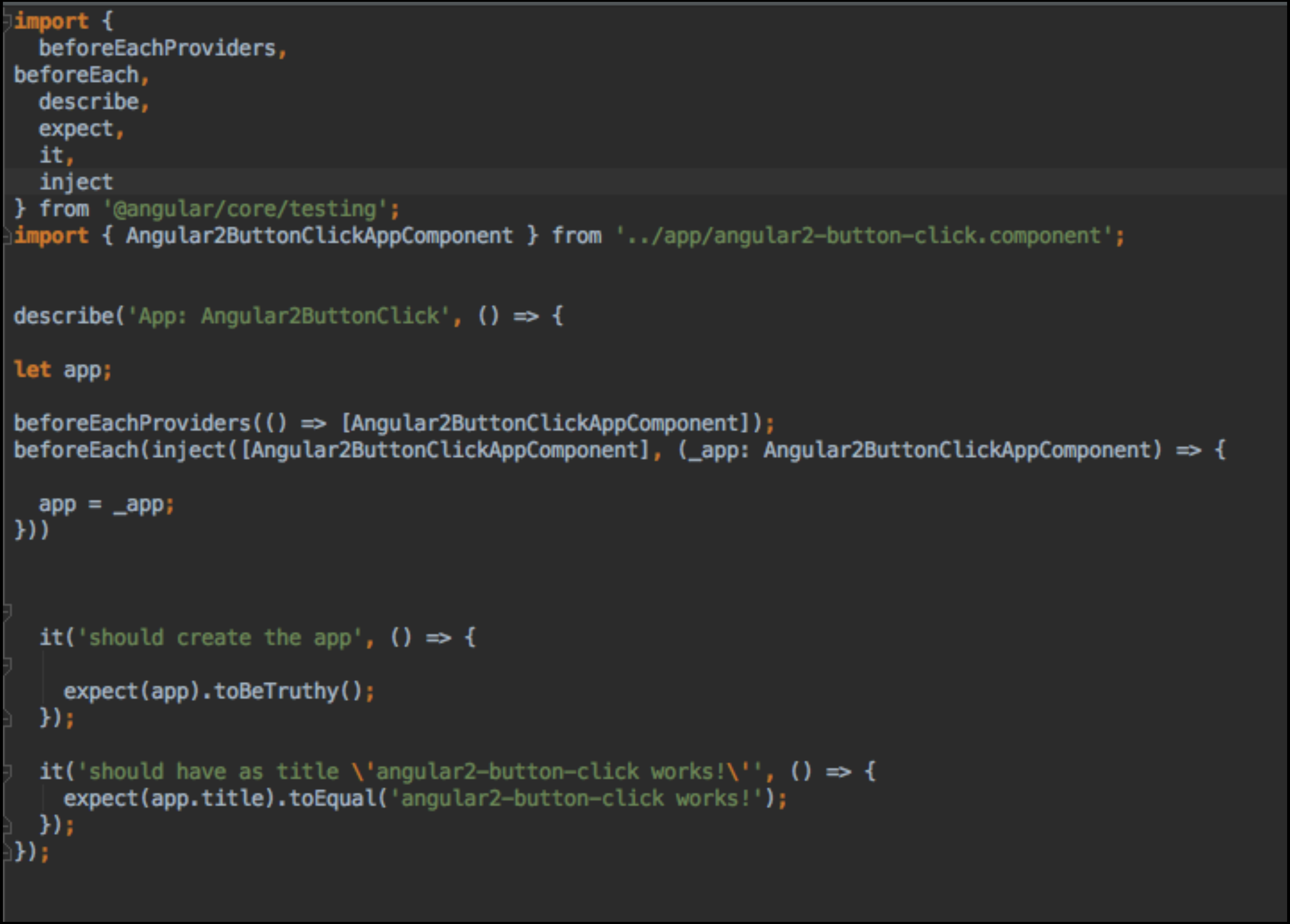
with some

beforeEach’s

(with a bunch of “import” statements)

(And also “beforeEachProviders”)

Example of an Angular 2 Unit Test



```
import {
  beforeEachProviders,
beforeEach,
  describe,
  expect,
  it,
  inject
} from '@angular/core/testing';
import { Angular2ButtonClickAppComponent } from '../app/angular2-button-click.component';


describe('App: Angular2ButtonClick', () => {

let app;

beforeEachProviders(() => [Angular2ButtonClickAppComponent]);
beforeEach(inject([Angular2ButtonClickAppComponent], (_app: Angular2ButtonClickAppComponent) => {

  app = _app;
}))



  it('should create the app', () => {

    expect(app).toBeTruthy();
  });

  it('should have as title \'angular2-button-click works!\'', () => {
    expect(app.title).toEqual('angular2-button-click works!');
  });
});
```

Thanks!

twitter.com/webWhizJim

github.com/JimTheMan

jim@ng-nj.com

www.wisdomofjim.com

The official "Describe's Full of It's" T-Shirt! Available now!

www.teepublic.com/t-shirt/468156-describes-full-of-its-white-text

Slides available here:
http://www.slideshare.net/JimLynch22/describes-full-of-its