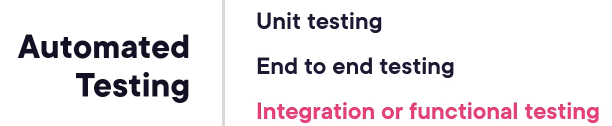
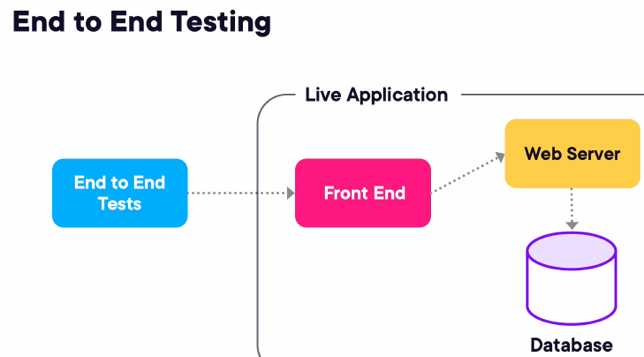
Angular Unit Testing Joe Eames

Intro



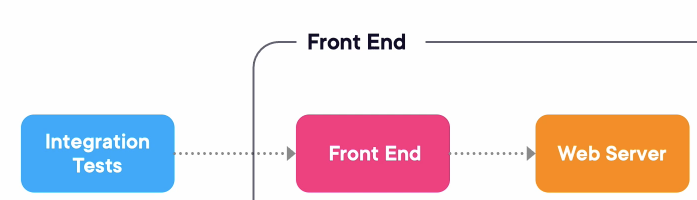


E2E

-tests our entire app

Unit

-single unit of code



Integration

-Somewhere in between

Mocking

-allows us to write our unit test against the code that we’re testing and not against other code

-don’t want to make actual HTTP calls

Types

Dummy

-simple objects to pass around and fill a place

Stub

-controllable behavior

-can decide to return a specific value

Spies

-keeps track of what methods (called, how many times, parameters)

True mocks

-object lets you tell it exactly what calls you expect to have against it then tells you whether or not those calls happened

Types of Unit Tests in Angular

Isolated

-single piece

-construct by using new keyword

Integration

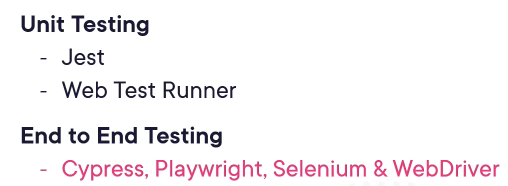
-uses testbed

Shallow: Just a Single component

Deep: Test the children of component

Tools

Karma, Jasmine are very popular

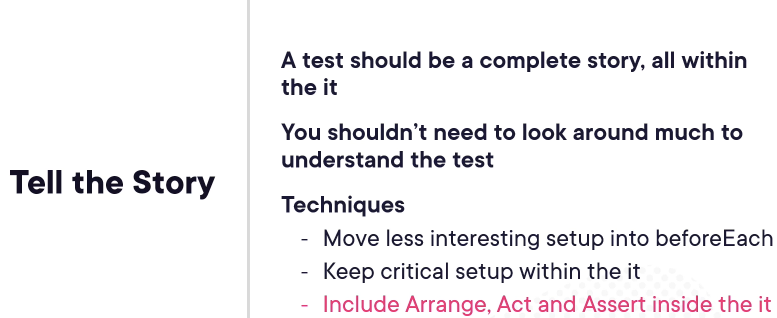




-to install into a directory of your choosing



-Follow DAMP, means we will repeat if necessary



Isolated Unit Tests

-Follow DAMP, means we will repeat if necessary

Mocking to Isolate Code

A problem that HeroesComponent had a dependency with HeroesService. Don’t really want to make an actual HeroesService because it will make HTTP call. So we will use spyObject

jasmine.createSpyObj([‘methodsName’])

-creates a mock object we can control

-can tell what methods, what to return, and what methods were called

mockHeroService.deleteHero.and.returnValue(of(true));

-Problem was this was returning an observable and subscribe was undefined?

Testing Interaction

ngOnInit

-we would have to call it ourselves because Angular frame is not running therefore is not automatically initializing the component

Shallow Integration Tests

Shallow Integration

-just testing single component, none of its children/directives

Testbed

-allows us to test component and template together

Testbed.configureTestBedModule

-creating another module that has only one component in it

let fixture: ComponentFixture<HeroComponent>;

fixture = Testbed.createComponent(HeroComponent);

-createComponent returns ComponentFixture which is a wrapper for a component

ComponentFixture

-object that represents a component and its DOM

fixture.componentInstance

-contains the component itself

fixture.detectChanges()

-

NO\_ERRORS\_SCHEMA

-Tells Angular don’t try to validate the template for the component or schema

-drawbacks are that it wont try to validate

Testing Rendered HTML

fixture.nativeElement

-gets a handle to the DOM element

-exposes the regular old browser’s DOM API



-probably should toEqual but is keeping our test from being unnecessarily brittle (hard but shatter easily)

-toContains is a little bit less brittle



-There will be an error because no change detection. So add detectChanges

NativeElement vs DebugElement

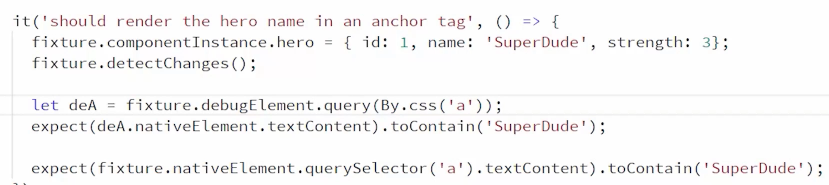
DE

-is more of a wrapper that has a different set of functionality that is similar to NE

-is a wrapper around the actual DOM node (similar to fixture)

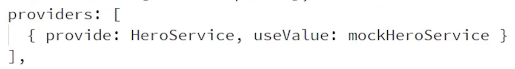
-can access the router link directive

-can know which component it belongs to (componentInstance)



Mocking an Injected Service

-we don’t to make real HTTP calls and don’t want testing two units at the same time



-Long handed way. When someone asks for HeroService, use this value instead

-for ngOnInit to fire, we need to run changeDectection. In Angular, changeDection causes lifecycle events to run

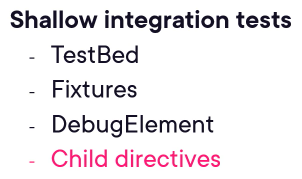


-this is what I did but this doesn’t follow integration tests

-want Angular to handle these things

Dealing with List of Elements

-Don’t always have to test template in integration tests



-To add child, add it to the Testbed module

Deep Integration Tests

Creating a Deep Intergration Test

-fixture.detectChanges() fires off ngOnInit, so we can have our component initialized. And add it fo beforEach

Finding Elements by Device <- REWATCH

-In Angular, a component is actually a subclass of a directive

Directive

-actually the parent class for both attribute directives and components

Integration Testing of Services

-Services doesn’t have HTML but it has HTTP

-can provide a special mock HTTP service

Integration Testing of Services

HTTP

-import HTTPClientTestingModule then import HTTPClientController

-

Testbed.inject(HTTPTestingController);

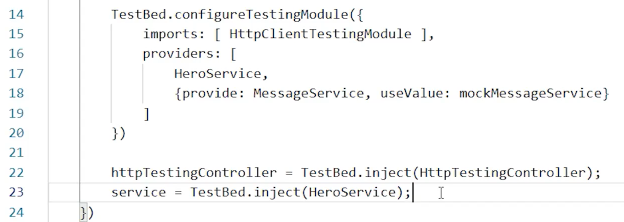
-acts dependency injection registry

-gets a handle to our service

HTTPClientTestingModule

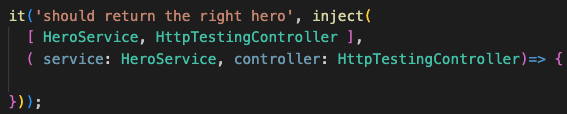
-will intercept any http calls

Method 1 of creating Heroservice



-Line23: tells the testing module to grab the HeroService that is created on Line17 and get a handle

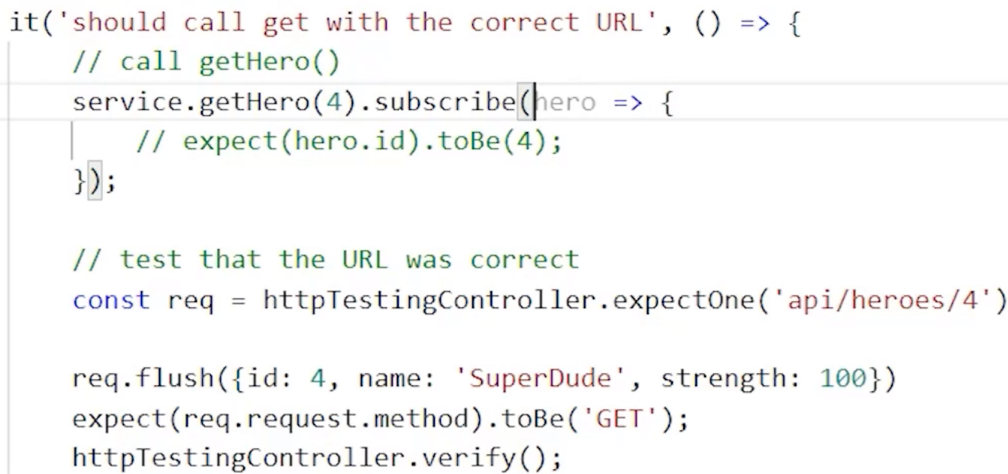
Method 2



-inject, takes two parameters

-Method 1 is a lot simpler

Implementing a Test with Mocked HTTP



expectOne()

-pass in what URL we expect

-this doesn’t send any results back to HTTP get request

flush

-decide what data to send back when call is made

verify

-verifies that it was the only requests that we expected

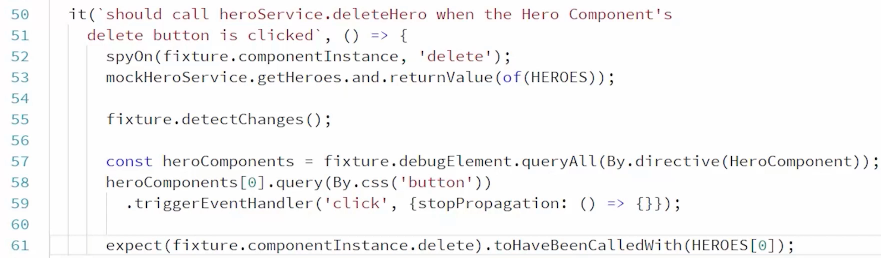
-extra calls would fail the test

Testing DOM interaction and routing components

Triggering Events on Elements



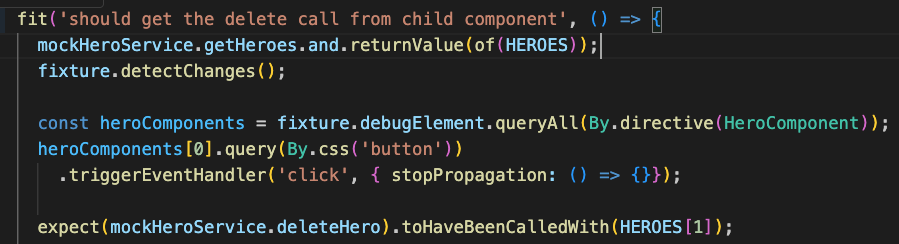
-Component is a subclass of a directive



-Need to handle $event.stopPropagation

-Line52

my way



Emitting Events from Children



Is underlying HTML or

-Subjective, one might worry about the bindings, might not have to worry about child component is working correctly (might be duplicating effort)

Raising Events on Child Directives

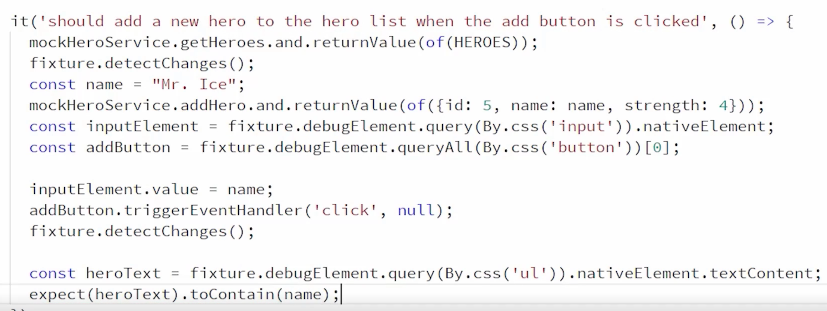


-telling debugElement to just trigger(raise) the delete event rather than telling the component to raise its delete event but testing less

Interacting with Input Boxes

.nativeElement

-grabs the underlying DOM elements



Testing With Activated Route

-we don’t want to test routing components works, assume Framework is working correctly

Asynchronous Tests and RxJS

Using fakeAsync Helper Function

-treat all of our asynchronous code if it were synchronous

tick(millis?: number);

-can tell our code to fast forward

-will call any code inside that timeframe

-Angular runs inside of zone.js, fakeAsync() runs in a special kind of zone that Zone.js creates

flush();

-look at zone and see if there’s any tasks waiting and fast forward the click until those tasks have been executed

Using the waitForfAsync Helper Function

-specific to promises

Misc:

-elements that have those decorator directives or attribute based directives that we are normally used to calling a directive