Angular Advanced Testing – introduction

Peter Kassenaar
info@kassenaar.com

Official documentation

v20

Ctrl K

Docs

Tutorials

Playground

Reference

Testing

- Overview
- Code coverage
- Testing services
- Basics of testing components
- Component testing scenarios
- Testing attribute directives
- Testing pipes
- Debugging tests
- Testing utility APIs
- Experimental unit testing integration
- Component harnesses overview
- Using component harnesses in tests
- Creating harnesses for your components
- Adding harness support for additional testing environments

In-depth Guides > Testing

Testing

Testing your Angular application helps you check that your application is working as you expect.

On this page

Set up testing

The Angular CLI downloads and installs everything you need to test an Angular application with Jasmine testing framework.

The project you create with the CLI is immediately ready to test. Just run the `ng test` CLI command:

```
$ ng test
```

The `ng test` command builds the application in *watch mode*, and launches the Karma test runner

https://angular.dev/guide/testing

Generic repo – not in `../examples` !

This repository Search Pull requests Issues Marketplace Explore

PeterKassenaar / ng-testing

Unwatch 1 Star 1 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Sample Angular-cli project with Jasmine/Karma unit tests

Edit

Add topics

9 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

PeterKassenaar Added links and comment toe README.md Latest commit fe5ef09 on 31 Aug

e2e	First commit of code, with sample tests.	8 months ago
src	some small changes	2 months ago
.angular-cli.json	First commit of code, with sample tests.	8 months ago
.editorconfig	First commit of code, with sample tests.	8 months ago
.gitignore	Added Firefox launcher	4 months ago
README.md	Added links and comment toe README.md	2 months ago
karma.conf.js	Adapted for ChromeHeadless testing	2 months ago
package.json	Adapted for ChromeHeadless testing	2 months ago
protractor.conf.js	First commit of code, with sample tests.	8 months ago
tsconfig.json	some small changes	2 months ago

github.com/PeterKassenaar/ng-testing

Tooling

Jasmine:

Write Unit Tests & Specs

Karma:

Test Runner

Protractor:

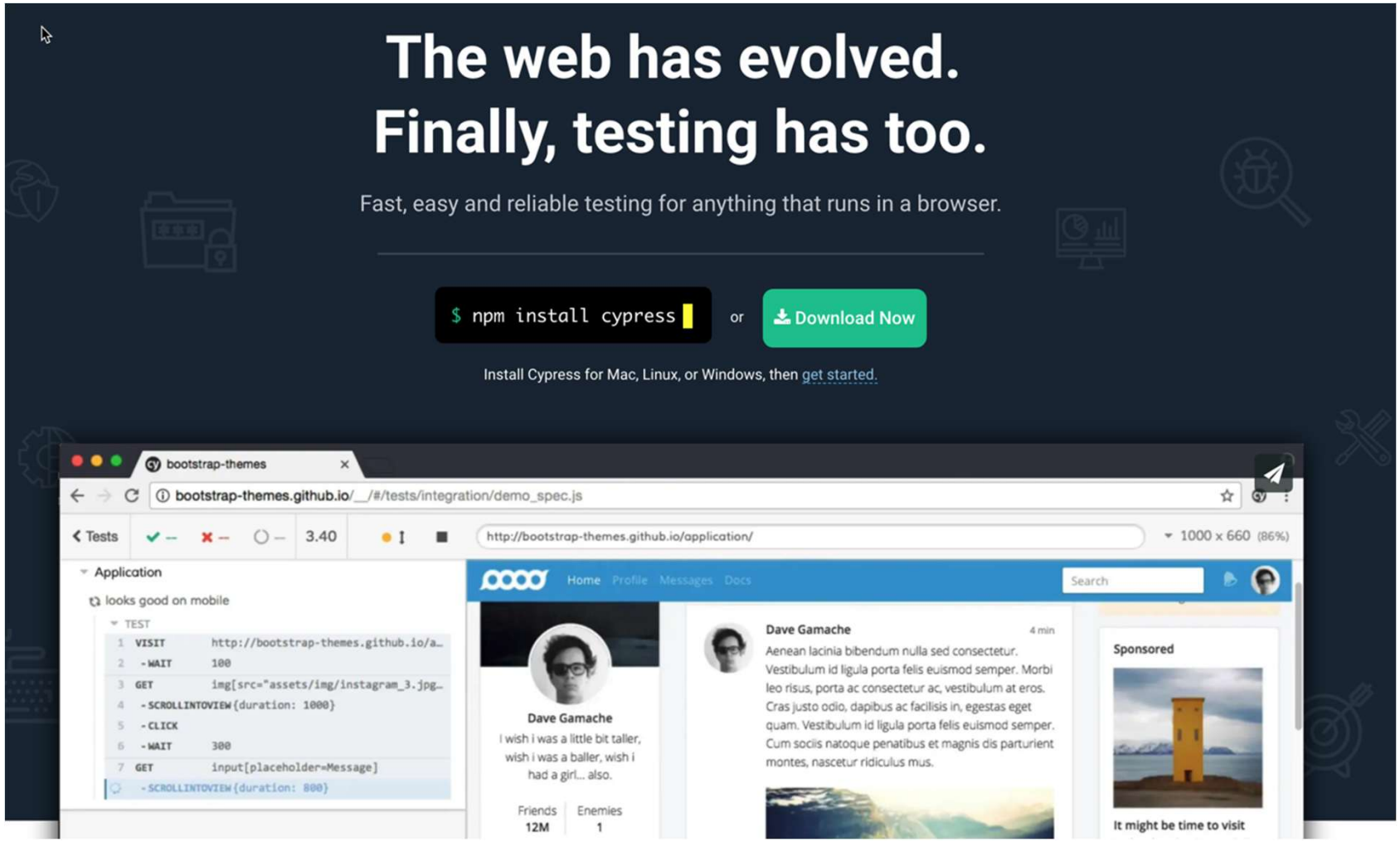
End to end testing

https://jasmine.github.io/

https://karma-runner.github.io/1.0/index.html

http://www.protractortest.org/#/

An alternative to E2E-testing - Cypress



https://www.cypress.io/

General Testing

General Jasmine syntax for testing classes and models

General testing pattern/syntax

Just a simple class and its usage:
import {Person} **from "./person.model"**;

```
// Person.model.ts
export class Person {
   constructor(public name?: string,
            public email?: string) {
   }

   sayHello(): string {
      return `Hi, ${this.name}`;
   }
}
```

General unit test pattern

```
// Generic testing pattern

// 1. Describe block for every test suite
describe('The Person', () => {

   // 2. Variables used by this test suite
   let aPerson;

   // 3. Setup block, run before every individual test
   beforeEach(() => {
      aPerson = new Person('Peter');
   });

   // 4. Clean up after every individual test
   afterEach(() => {
      aPerson = null;
   });

   // 5. Perform each test in an it()-block
   it('should say Hello', () => {
      let msg = aPerson.sayHello();
      expect(msg).toBe('Hi, Peter');
   });

   // 6. More it()-blocks…
});
```

So a `*.spec.ts` file typically contains:

One or more `describe()` blocks

One or more `beforeEach()` blocks

One or (typically) more `it()` blocks, using `expect()` statements and Jasmine *matchers*

We're using `@angular/cli` here

- Angular-cli: all dependencies already installed and configured.
  - Command: `ng test`

- Manual project? Install & setup karma and jasmine yourself
  - Create and adapt `karma.conf.js`
  - `karma --init`
  - Install and setup jasmine reporters
  - *We're not doing that here*.
  - See documentation

```
…
},
"devDependencies": {
   …
  "@types/jasmine": "2.5.38",
  "@types/node": "~6.0.60",
  "codelyzer": "~2.0.0",
  "jasmine-core": "~2.5.2",
  "jasmine-spec-reporter": "~3.2.0",
  "karma": "~1.4.1",
  "karma-chrome-launcher": "~2.0.0",
  "karma-cli": "~1.0.1",
  "karma-jasmine": "~1.1.0",
  "karma-jasmine-html-reporter": "^0.2.2",
  "karma-coverage-istanbul-reporter": "^0.2.0",
  "protractor": "~5.1.0",
   …
}
…
```

Browser like…

```
C:\Users\Peter Kassenaar\Desktop\ng-testing>ng test
10 03 2017 16:50:40.157:WARN [karma]: No captured browser, open http://localhost:9876/
10 03 2017 16:50:40.170:INFO [karma]: Karma v1.4.1 server started at http://0.0.0.0:9876/
10 03 2017 16:50:40.171:INFO [launcher]: Launching browser Chrome with unlimited concurrency
10 03 2017 16:50:40.310:INFO [launcher]: Starting browser Chrome
10 03 2017 16:50:42.220:INFO [Chrome 56.0.2924 (Windows 10 0.0.0)]: Connected on socket 3Ox3NIH4ohFYk
MAuAAAA with id 46142248
Chrome 56.0.2924 (Windows 10 0.0.0): Executed 10 of 10 SUCCESS (0.461 secs / 0.439 secs)
```

Karma

localhost:9876/?id=46142248

Karma v1.4.1 - connected

Chrome 56.0.2924 (Windows 10 0.0.0) is idle

Jasmine 2.5.2

• • • • • • • • • •

10 specs, 0 failures

AppComponent
  should create the app
  should have as title 'app works!'
  should render title in a h1 tag

Simple Async Service
  should return Hi Peter from the async service

Test Plain Greeting Service
  should have generated the greeting service
  should return Hi, Peter

Test Greeting service via TestBed
  Should have generated the service via TestBed
  Should return Hi, Sandra

Simple HTTP Remote Service
  RemoteService should be defined

The Person
  should say Hello

On Angular-specific terms

- TestBed
- inject
- async
- fakeAsync
- ComponentFixture
- DebugElement
- configureTestingModule

Basic component test included

```
1   import { TestBed, async } from '@angular/core/testing';
2   import { AppComponent } from './app.component';
3
4   describe('AppComponent', () => {
5     beforeEach(async(() => {
6       TestBed.configureTestingModule({
7         declarations: [
8           AppComponent
9         ],
10      }).compileComponents();
11    }));
12
13    it('should create the app', () => {
14      const fixture = TestBed.createComponent(AppComponent);
15      const app = fixture.debugElement.componentInstance;
16      expect(app).toBeTruthy();
17    });
18
19    it(`should have as title 'oceTestApp'`, () =>{
20      const fixture = TestBed.createComponent(AppComponent);
21      const app = fixture.debugElement.componentInstance;
22      expect(app.title).toEqual('oceTestApp');
23    });
24
25    it('should render title in a h1 tag', () => {
```

Breaking the .spec file down:

- `TestBed` – the Angular Testing implementation of a Module
- `.configureTestingModule()` – configure only the parts of the module you want to test
- `.compileComponents()` – we're testing a component here. Not necessary for services, models, etc.
- `fixture` – is of type `ComponentFixture`, acts as a wrapper around the actual component
- `debugElement.nativeElement` - access to the vDOM of the actual component instance
- `detectChanges` – run change detection manually

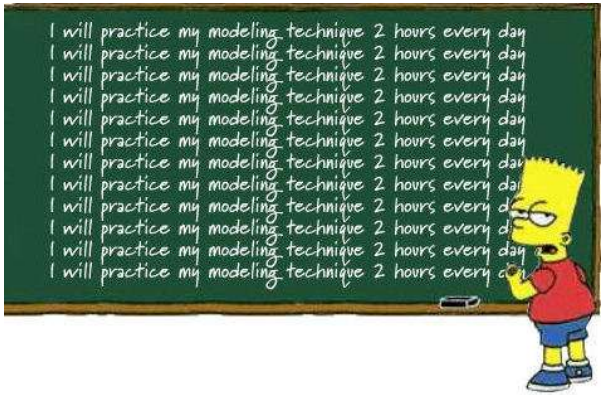
Running the tests

- `ng test`
  - Run all the tests in the .spec-files in the project
  - You can keep this running in the background!
    - It might slow down development
- `fit` - run this test only
- `xit` - run all tests except this one

Workshop

- Create a new, empty Angular Project
- Run `ng test` – identify what tests are run
- Try to make some simple changes to the tests. See if they still run

- Add a new component and run its tests
- Add an array of Cities and functionality for adding and deleting a city
- Write tests for the new component.
  - Add a test for the `counter` property
  - Create new functions (`decrement`, `reset`) and write tests for them…
- https://github.com/PeterKassenaar/ng-testing
- Example: `../spy/spy.component.spec.ts`



Testing Services

Testing a single service

One of the more easier concepts to test. So let's start there.

```
// greeting.service.ts
import {Injectable} from '@angular/core';

@Injectable()
export class GreetingService {

   constructor() {
   }

   greet(name: string): string {
      return `Hi, ${name}`;
   }
}
```

```
// greeting.service.spec.ts
import {GreetingService}from './greeting.service';
describe('Test Plain Greeting Service', () => {
    let greetingService;
    beforeEach(() => {
        greetingService = new GreetingService();
    });

    it('should have generated the greeting service', () => {
        expect(greetingService).toBeTruthy()
    });

    it('should return Hi, Peter', () => {
        let msg = greetingService.greet('Peter');
        expect(msg).toEqual('Hi, Peter');
    });
});
```

Output

```
Terminal
START:
  Test Plain Greeting Service
    √ should have generated the greeting service
    √ should return Hi, Peter
  The Person
    √ should say Hello
```

But what about DI?

- Most of the time we don't have a simple , single service
- We use it in the context of an `ngModule()`

```
…
import {GreetingService} from './shared/services/01-greeting.service';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [GreetingService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Using TestBed

- In testing we have the same concept of modules, using `TestBed`
- TestBed has a method `.configureTestingModule()` to mimic an `ngModule`.
- You only specify the stuff you need!
  - No need to build the complete module, importing all dependencies

```
let service;
beforeEach(() => {
    TestBed.configureTestingModule({
        providers: [GreetingService]
    });
    service = TestBed.get(GreetingService);
});
```

Dependencies

Get instance of the service via TestBed.get()

“TestBed is the primary api for writing unit tests for Angular applications and libraries.”

ANGULAR FEATURES DOCS RESOURCES EVENTS BLOG Search

GETTING STARTED
TUTORIAL
FUNDAMENTALS
TECHNIQUES
API

stable (v4.3.4)

TestBed CLASS

npm Package	@angular/core
Module	`import { TestBed } from '@angular/core/testing';`
Source	core/testing/src/test_bed.ts

Configures and initializes environment for unit testing and provides methods for creating components and services in unit tests.

Overview

```
class TestBed implements Injector {
  static initTestEnvironment(ngModule: Type<any>|Type<any>[], platform: PlatformRef, aotSummaries?: () => any[]): TestBed
  static resetTestEnvironment()
  static resetTestingModule(): typeof TestBed
  static configureCompiler(config: {providers?: any[]; useJit?: boolean;}): typeof TestBed
  static configureTestingModule(moduleDef: TestModuleMetadata): typeof TestBed
  static compileComponents(): Promise<any>
  static overrideModule(ngModule: Type<any>, override: MetadataOverride<NgModule>): typeof TestBed
  static overrideComponent(component: Type<any>, override: MetadataOverride<Component>): typeof TestBed
```

https://angular.io/api/core/testing/TestBed

Async behavior

- If the services uses async calls, for instance Promises or Observables
- The spec-file will always returns true, because the `expect`-statement is not run in the `.then()`-clause
- So this wil NOT work:

```
it('should return Hi Peter from the async service', ()=>{
   service.greetAsync('Peter')
      .then(result =>{
         expect(result).toEqual('Hi, Petertest');
      })
});
```

Test will incorrectly pass

Using Angular `async` and `fakeAsync`

- Solution: import and use Angular `async()` or `fakeAsync()` construct and wrap the expect statement in this function
- Jasmine will wait for the `async()` function to return and *then* perform the test

Use async(). Test will run OK

```
it('should return Hi Peter from the async service', async(()=>{
     service.greetAsync('Peter’)
        .then((result)=>{
           expect(result).toEqual('Hi, Peter');
     })
 }))
```

Async vs. fakeAsync

- Mostly interchangeable
- `fakeAsync` offers more configuration/control, but is used less often
- Use `fakeAsync` if you need manual control of zones `tick()` function

Google

```
it('should work with fakeAsync', fakeAsync(() => {
  let value;
  service.simpleAsync().then((result) => {
    value = result;
  });
  expect(value).not.toBeDefined();

  tick(50);
  expect(value).not.toBeDefined();

  tick(50);
  expect(value).toBeDefined();
}));
```

"fakeAsync() makes your async code synchronous"

https://stackoverflow.com/questions/42971537/what-is-the-difference-between-fakeasync-and-async-in-angular2-testing

Async documentation

ANGULAR FEATURES DOCS RESOURCES EVENTS BLOG Search

GETTING STARTED
TUTORIAL
FUNDAMENTALS
TECHNIQUES
API

stable (v4.3.4)

async FUNCTION

npm Package	@angular/core
Module	`import { async } from '@angular/core/testing';`
Source	core/testing/src/async.ts

```
function async(fn: Function): (done: any) => any;
```

Description

Wraps a test function in an asynchronous test zone. The test will automatically complete when all asynchronous calls within this zone are done. Can be used to wrap an `inject` call.

Example:

```
it('...', async(inject([AClass], (object) => {
  object.doSomething.then(() => {
    expect(...);
  })
});
```

https://angular.io/api/core/testing/async

Mocking backend

Testing asynchrounous XHR-calls

Like any external dependency, the HTTP backend needs to be mocked so your tests can simulate interaction with a remote server. The @angular/common/http/testing library makes setting up such mocking straightforward.

https://angular.io/guide/http#testing-http-requests

Testing XHR calls

Setup (dummy) remote service to fetch some data over HTTP

```
constructor(private http: HttpClient) {
}
// Get fake people
public getPeople(): Observable<Person[]> {
   return this.http
      .get('someEndPoint/somePeople.json')
      .map(result => result.json());
}
```

Using `HttpClientTestingModule` and `HttpTestingController`

- Don't perform the tests to a 'real' API with `HttpClientModule`
  - Import `HttpClientTestingModule` in your `.spec`-file
- Mock your backend by giving the test fake data

"A test expects that certain requests have or have not been made, performs assertions against those requests, and finally provide responses by "flushing" each expected request."

Import the correct modules

```
import {HttpClientTestingModule,
        HttpTestingController} from '@angular/common/http/testing';
```

Then, add the `HttpClientTestingModule` to the TestBed and write the tests

```
// 2. Setup in the beforeEach() block
beforeEach(() => {
  TestBed.configureTestingModule({
    imports: [HttpClientTestingModule],
    providers: [RemoteService]
  });
  // 2a. Assign variables.
  injector = getTestBed();
  remoteService = injector.get(RemoteService);
  httpMock = injector.get(HttpTestingController);
  // 2b. Our mocked response
  mockUsers = [{
    name: 'Peter',
    email: 'info@kassenaar.com'
  }, {
    name: 'Sandra',
    email: 'sandra@kassenaar.nl'
  }]
});
```

HttpClientTestingModule

Mocked (fake) data

Now requests made in the tests will hit the testing backend instead of the normal backend.

Write test for mocked backend

```
it('should return an Observable<User[]>', () => {
  // Make an HTTP GET request
  remoteService.getPeople()
    .subscribe(users => {
      expect(users.length).toBe(2);
      expect(users).toEqual(mockUsers);
    });
  // verify and flush our request
  const req = httpMock.expectOne(remoteService.url);
  expect(req.request.method).toBe("GET"); // just to be safe. Not mandatory.

  // Only after a flush() the .subscribe expecations are evaluated and available!
  req.flush(mockUsers);

  // Finally, verify if there are no outstanding requests.
 // httpMock.verify()
});
```

Subscriber. Results are only available after a Flush

Provide the subscriber with our (fake) data by calling .flush()

Documentation on HttpTestingModule

ANGULAR FEATURES DOCS RESOURCES EVENTS BLOG

STARTED
ENTALS
cture
nents & Templates
ables & RxJS
apping
ules
ency Injection
ent
& Navigation
ons
UES
DEPLOYMENT
File Structure
ace Configuration
pendencies

Testing HTTP requests

Like any external dependency, the HTTP backend needs to be mocked so your tests can simulate interaction with a remote server. The `@angular/common/http/testing` library makes setting up such mocking straightforward.

Mocking philosophy

Angular's HTTP testing library is designed for a pattern of testing wherein the app executes code and makes requests first.

Then a test expects that certain requests have or have not been made, performs assertions against those requests, and finally provide responses by "flushing" each expected request.

At the end, tests may verify that the app has made no unexpected requests.

> You can run these sample tests / download example in a live coding environment.
>
> The tests described in this guide are in `src/testing/http-client.spec.ts`. There are also tests of an application data service that call `HttpClient` in `src/app/heroes/heroes.service.spec.ts`.

Setup

To begin testing calls to `HttpClient`, import the `HttpClientTestingModule` and the mocking controller, `HttpTestingController`, along with the other symbols your tests require.

app/testing/http-client.spec.ts (imports)

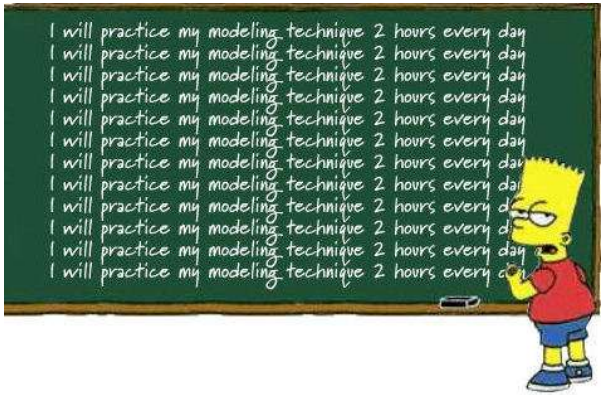
https://angular.io/guide/http#testing-http-requests

Summary

- What have we learned?
  - Jasmine Syntax: `describe()`, `beforeEach()`, `it()`, `expect()`
  - `TestBed.configureTestingModule()`
  - `TestBed.get()`
  - Testing Classes
  - Testing Services
- Mocking backend
  - `HttpClientTestingModule`, `HttpTestingController`
  - flush()

Workshop

- Study the example `../ng-testing`.
- Study `shared/model/10-car.model.ts` and create a test suite for it
- Study `10-car.service.ts` and create a test suite for it, using `TestBed.configureTestingModule()`
- Study `11-car.remote.service.ts` and create a test suite for it, using `HttpClientTestingModule`
  - The service is fetching Cars from a dummy backend.
  - Also write a test for the second method, fetching cars from a specific year.



Testing components

Building blocks of every application

Testing components

- Less often tested than services, routes, etc.
- Test View components only for their `@Input()` en `@Output()'s`
- Test Smart components as simple as possible

- New concepts:
  - `.compileComponents()`
  - `Fixture`
  - `.detectChanges()`
  - `componentInstance`
  - `DebugElement`
  - `NativeElement`

Simple Component testing

- Generate a simple component, using `{{ … }}` data binding for instance
- Create a TestBed, with instance of the component
  - Now using the `declarations` array
- Compile the component using `.compileComponents()`.

```
beforeEach(async(() => {
      TestBed.configureTestingModule({
         declarations: [CityComponent]
      })
      .compileComponents();
   })
);
```

"Do not configure the TestBed after calling compileComponents. Make compileComponents the last step before calling TestBed.createComponent to instantiate the component-under-test."

https://angular.io/guide/testing#the-first-asynchronous-beforeeach

Component Instance

- Component is now compiled for testing purposes (and added to the TestBed), but is not instantiated yet
- Use variables `component` and `fixture` for that.

```
let component: CityComponent;
let fixture: ComponentFixture<CityComponent>;

beforeEach(async(() => {
      …
      fixture   = TestBed.createComponent(CityComponent);
      component = fixture.componentInstance;
      fixture.detectChanges();
   })
);
```

"The fixture provides access to the component instance itself and to the DebugElement, which is a handle on the component's DOM element."

Complete test

```
import {async, ComponentFixture, TestBed} from '@angular/core/testing';

import {CityComponent} from './city.component';

describe('CityComponent', () => {
   let component: CityComponent;
   let fixture: ComponentFixture<CityComponent>;

   beforeEach(async(() => {
         TestBed.configureTestingModule({
            declarations: [CityComponent]
         })
            .compileComponents();

         fixture   = TestBed.createComponent(CityComponent);
         component = fixture.componentInstance;
         fixture.detectChanges();
      })
   );

   it('should be created', () => {
      expect(component).toBeTruthy();
   });
});
```

Testing a method on a component

Suppose the component has a `setCity(name)` method like so:

```
setCity(name: string) {
   this.city = name;
}
```

Usage in a test

```
it('city should have the name Amsterdam', ()=>{
   component.setCity('Amsterdam');
   expect(component.city).toEqual('Amsterdam');
});
```

Accessing DOM-elements

Handy Helper library:`By`

- Provides querying the DOM like jQuery and `.querySelector[All]`

```
import {By} from '@angular/platform-browser';
```

Use debugElement and NativeElement.

Don’t forget to trigger change detection on the fixture!

```
it('should have rendered Amsterdam on the page', ()=>{
   const de = fixture.debugElement.query(By.css('h1'));
   const element = de.nativeElement;
   component.setCity('Amsterdam');
   fixture.detectChanges();
   expect(element.textContent).toContain('Amsterdam');
});
```

Testing @Input() and @Output

Testing component attributes and events, using Jasmine spies

Testing @Input() parameters

- An `Input()`-parameter is just a property on the class.
- So treat it as such…

```
import {Component, Input} from '@angular/core';

@Component({
   selector   : 'app-input',
   templateUrl: './input.component.html',
   styles     : []
})
export class InputComponent {
   @Input() msg: string;
}
```

```
With the template just: "{{ msg }}"
```

Writing the @Input test

```
import {async, ComponentFixture, TestBed} from '@angular/core/testing';

import {InputComponent} from './input.component';

describe('InputComponent', () => {
   let component: InputComponent;
   let fixture: ComponentFixture<InputComponent>;

   beforeEach(async(() => {
     …
   }));

   it('should have the message defined', ()=>{
      expect(fixture.debugElement.nativeElement.innerHTML).toEqual('');

      // now let's set the message
      component.msg = 'Hi, there';

      fixture.detectChanges();

      expect(fixture.debugElement.nativeElement.innerHTML).toEqual('Hi, there');

   })
});
```

Different ways to access the underlying DOM

- Access DOM-element via `debugElement` and `By`:
  - `fixture.debugElement.query(By.css('h1'));`
  - Recommended. DOM is abstracted by Angular. Works also in Non-DOM environments (like server-apps)
- Access native element directly:
  - Not recommended, but certainly possible (if you *know*, you app will never run outside a browser environment)
  - `fixture.nativeElement.querySelector('h1');`

Testing @Output() events

- Simple class, outputting a `msg` event, submitting a message.

```
import {Component, EventEmitter, Output} from '@angular/core';

@Component({
   selector   : 'app-output',
   templateUrl: './output.component.html'
})
export class OutputComponent {

   @Output() msg: EventEmitter<string> = new EventEmitter<string>();

   sendMsg(msg: string) {
      this.msg.emit(msg);
   }
}
```

```
 <p>
  <button (click)="sendMsg('Hi, there')">Send Message</button>
</p>
```

Different strategies for testing events

1st strategy:
simply subscribe to the event and call the method that fires the event

```
it('should fire the msg event', ()=>{
   // 1st strategy : subscribe to the msg event
   component.msg.subscribe(msg =>{
      expect(msg).toBe('Hi, there');
   });
   // emit the actual event with a value
   component.sendMsg('Hi, there');
})
```

Creating a Jasmine Spy

- Spies are Jasmine ‘watchers’
- You can query the spy and expect that
  - it has been called,
  - It has not been called
  - It has been called with a specific value,
  - It has been called a specific number of times,
  - …
- Documentation: https://jasmine.github.io/api/2.7/global.html#spyOn
- Cheat sheet: https://daveceddia.com/jasmine-2-spy-cheat-sheet/
- Tutorial: http://www.htmlgoodies.com/html5/javascript/spy-on-javascript-methods-using-the-jasmine-testing-framework.html

Jasmine spyOn()

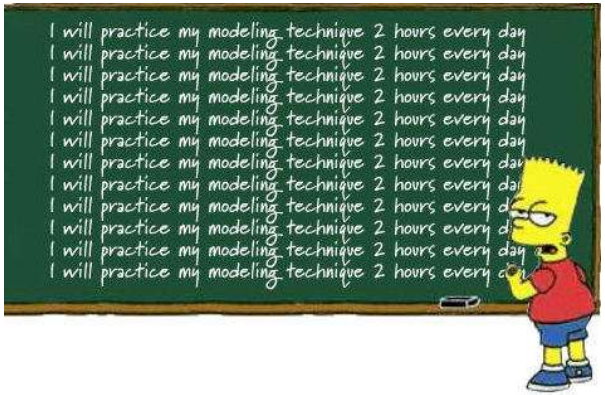
spyOn(Object, ‘method’)

```
it('should spy on the msg event', ()=>{
   // 2nd strategy : create a Jasmine Spy, watch the 'emit' event.
   spyOn(component.msg, 'emit');
   const button = fixture.debugElement.nativeElement.querySelector('button');
   button.click();

   expect(component.msg.emit).toHaveBeenCalledWith('Hi, there');
})
```

Workshop

- Study `car.component.ts` and create a test suite for it
- Test wether the component is correctly created
- Test if `this.cars[]` is constructed after initialisation. Expect the length of the array to be `2`.
- Test wether the `@Output()` event is called when clicked on a car
- Create an `@Input` property for the component yourself and test it



Mocking components

Strategies for minimizing the dependency chain of components

Multiple components

- What if an a component has multiple, nested components?
- Different possible strategies
  - Include all your components
  - Override your components at test time
  - Ignore errors and continue, using `NO_ERRORS_SCHEMA`

```
<!--card.component.html|ts|spec.ts-->
<card-header>
   Some Title
</card-header>
<card-content>
   Some content
</card-content>
<card-footer>
   Copyright (C) - 2017
</card-footer>
```

#1 Include all components

- Import all components and reference them in `declarations : […]` section
- *Pro* – complete test coverage, compile the component as it would run in the live app
- *Con* – more overhead, slower, tests for nested components are possibly elsewhere, overkill

```
import {CardComponent, CardContent, CardFooter, CardHeader} from './card.component';

describe('CardComponent', () => {
   …
   beforeEach(async(() => {
      TestBed.configureTestingModule({
         declarations: [CardComponent, CardHeader, CardContent, CardFooter],
      })
         .compileComponents();
   }));
   …
});
```

#2 – Override components at test time

- Simply provide empty components to the testsuite
- *Pro* – Component would compile as at run time
- *Con* – duplicate code, not a nice view, more overhead.

```
@Component({
   selector: 'card-header',
   template:''
})
export class CardHeaderMock{}

@Component({
   selector: 'card-content',
   template:''
})
export class CardContentMock{}
…
```

#3 Using NO_ERRORS_SCHEMA

- Provide a schema to the testing module, ignoring all errors and always continu the test
- *Pro* – flexible setup, no dependencies, faster compiling
- *Con* – you might not catch other errors

```
import {NO_ERRORS_SCHEMA} from '@angular/core';

describe('CardComponent', () => {
   …
   beforeEach(async(() => {
      TestBed.configureTestingModule({
         declarations: [CardComponent],
         schemas     : [NO_ERRORS_SCHEMA] // ignore alle errors, just go on.
      })
      .compileComponents();
      …
   }));
   …
});
```

End-2-End testing

Testing complete scenario’s

What is End 2 End testing

- Test complete processes, not single units
- Test in real environments, with the whole application
- Test real live situations
- Know if most important (tested) features work with the application
- Does *not* test edge cases
- Does *not* necessarily improve code quality

E2e test Tooling

Test Automation

Selenium

Automate browsers to perform scenario's

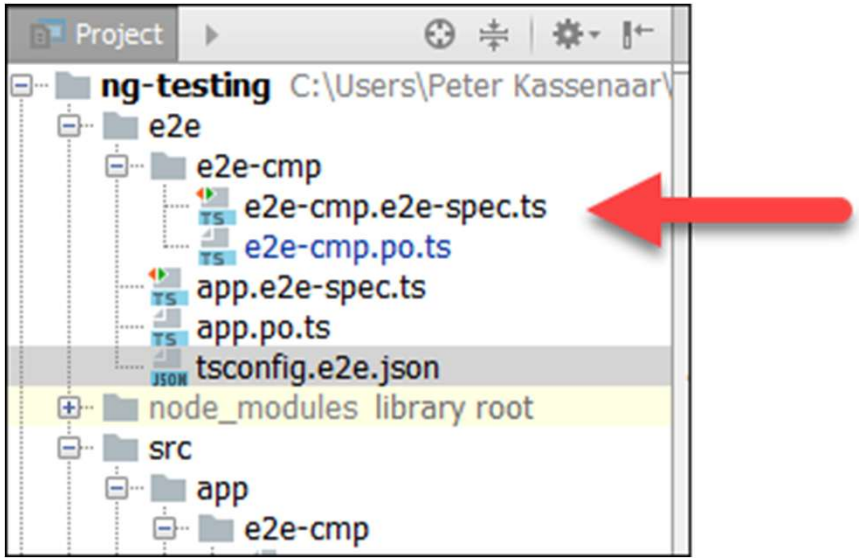
Developed by Team Angular – wrapper around selenium

Parts of e2e tests

- Folder: `/e2e`
- Page Objects – describe the scenario in a `*.po.ts`-file
  - Export a class with a `navigateTo()` function,
  - Plus functions for all the elements you want to retrieve/test
- Test files – write tests as usual
  - They load the Page Object class
  - Write a `beforeEach()` block to instantiate the Page Object
  - Write tests to invoke and test the elements on the page

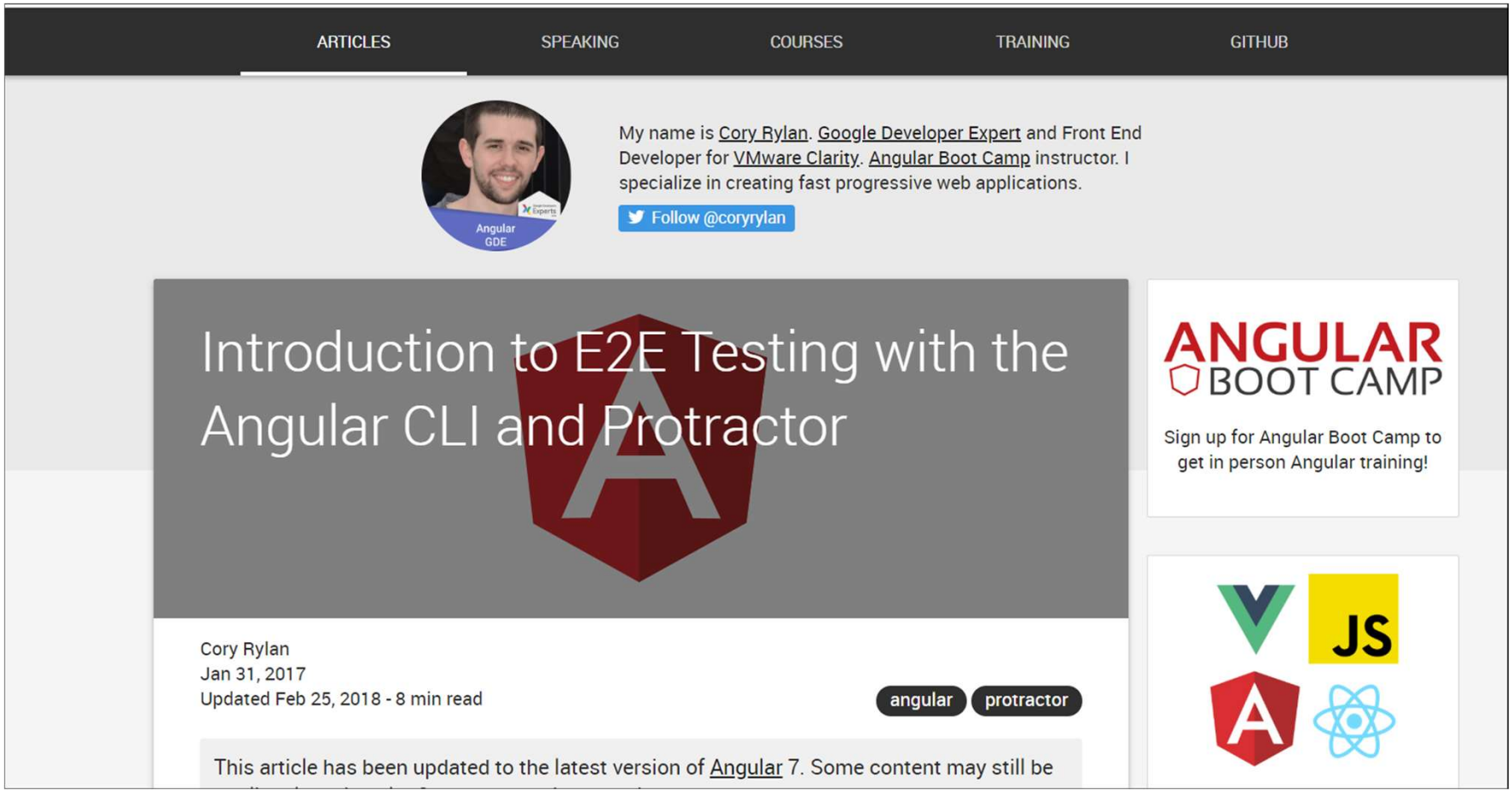
Executing e2e tests

- Generic command: `ng e2e`
- Examples: `\e2e-cmp`



More information on e2e testing

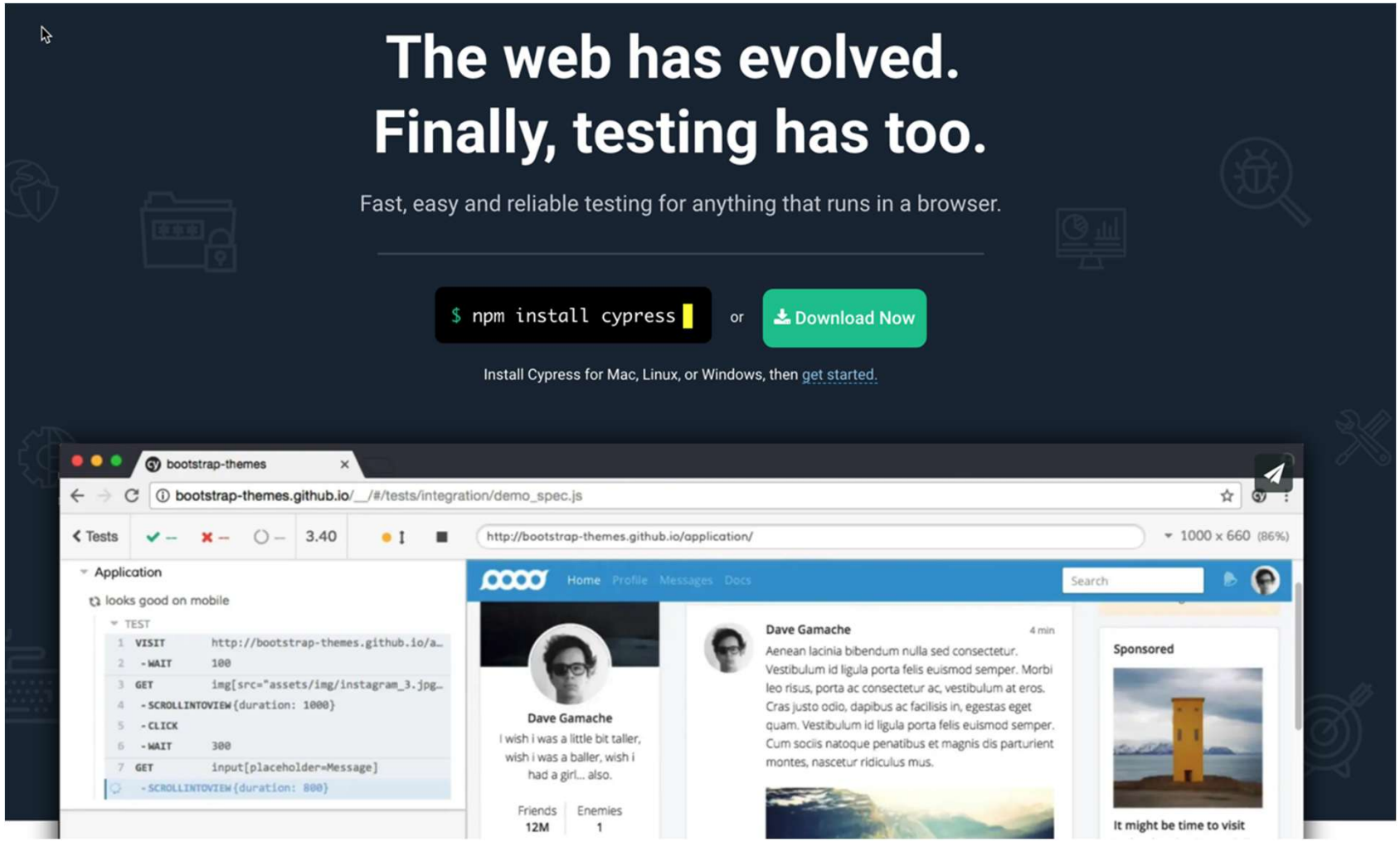
- https://coryrylan.com/blog/introduction-to-e2e-testing-with-the-angular-cli-and-protractor



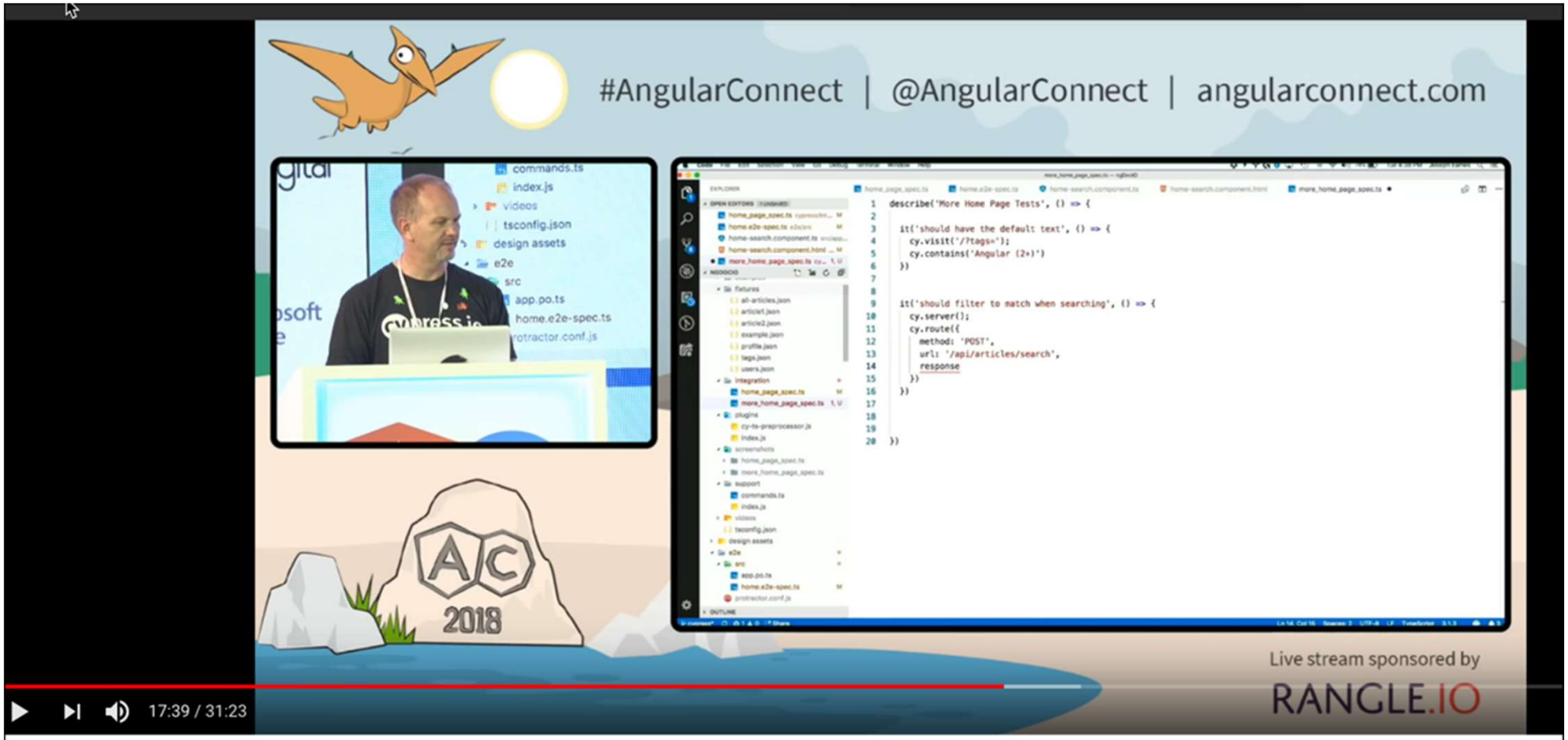
Testing with Cypress

The Future of testing?

An alternative to E2E-testing - Cypress



https://www.cypress.io/



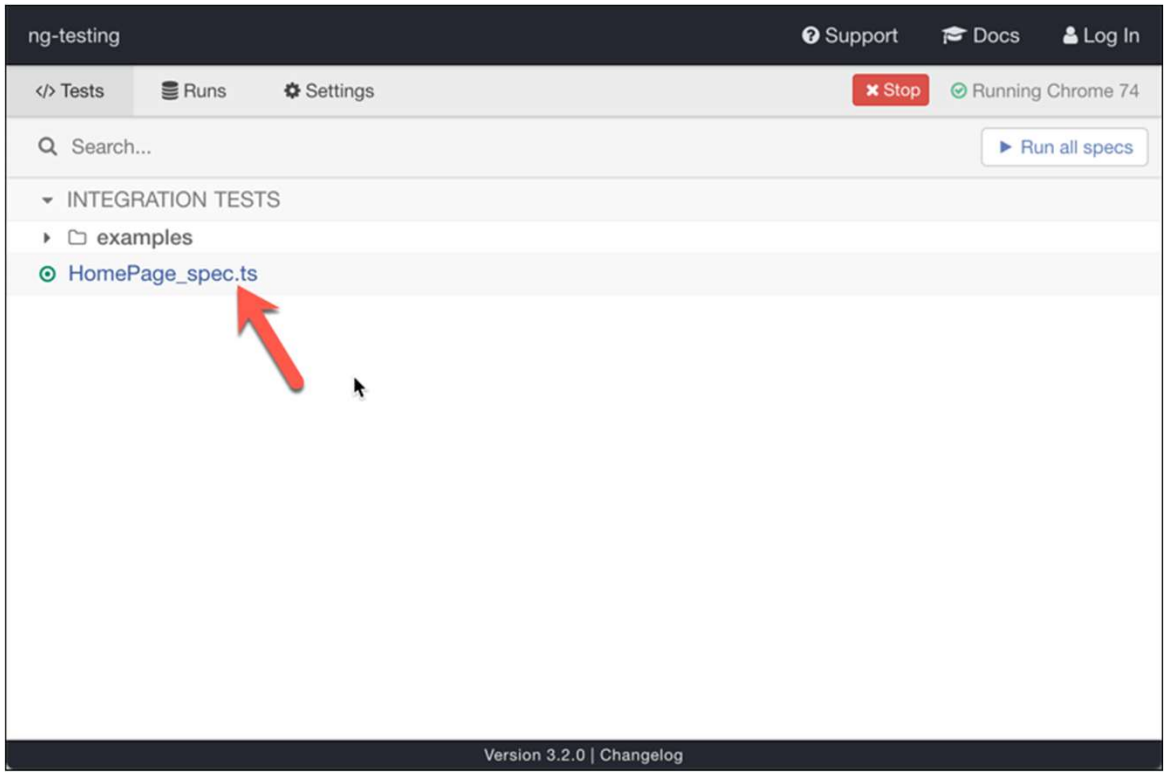
Testing Angular with Cypress.io | Joe Eames | AngularConnect 2018

Volgende

https://www.youtube.com/watch?v=eZyD-8qglWY

In testing repo

- A small test for the homepage
  - Start project on `localhost:4200`
  - `npm run cypress:open`



ng-testing

localhost:4200/__/#/tests/integration/HomePage_spec.ts

Tests ✔ 6 ✖ -- ◯ -- 2.54

http://localhost:4200/

Cypress testing the home page

- should visit the home page
- should have the title Welcome to app

Testing the e2e component

- should have the title Example e2e testing component
- should have a default of 1 point
- should add points when clicked 3 times
- should reset when clicked

Welcome to app!!

Here are some links to help you start:

- Tour of Heroes
- CLI Documentation
- Angular blog

Workshop 1

Check the code

Project

- ng-testing ~/Desktop/ng-testing
  - cypress
    - fixtures
    - integration
      - examples
      - HomePage_spec.ts
    - plugins
    - screenshots
    - support
  - e2e
  - node_modules library root
  - src
  - .angulardoc.json
  - .editorconfig
  - .gitignore
  - angular.json
  - cypress.json
  - karma.conf.js
  - package.json
  - package-lock.json
  - protractor.conf.js

cypress.json | README.md | HomePage_spec.ts | package.json

```
1  // HomePage_spec.ts - some cypress tests for our homepage
2  // documentation: https://docs.cypress.io/guides/tooling/typescript-s
3
4  describe('Cypress testing the home page', () => {
5    it('should visit the home page', () => {
6      cy.visit('http://localhost:4200');
7    });
8
9    it('should have the title Welcome to app', () => {
10     cy.visit('http://localhost:4200');
11     cy.get('h1').contains('Welcome to app')
12   });
13
14   describe('Testing the e2e component', () => {
15     it('should have the title Example e2e testing component', () => {
```

Code Coverage

- "How much of my code is covered by tests?"
- 100% would be ideal, but many teams are OK with 80-90%.
  - To be discussed
- Built in in CLI as a flag:
  - `ng test --code-coverage`

All files

100% Statements 22/22 **100%** Branches 0/0 **100%** Functions 3/3 **100%** Lines 20/20

File	Statements		Branches		Functions		Lines	
src	100%	16/16	100%	0/0	100%	1/1	100%	16/16
src/app	100%	6/6	100%	0/0	100%	2/2	100%	4/4

Summary

What have we learned

Summary

- We learned about:
  - ComponentFixture
  - .compileComponents()
  - .detectChanges()
  - .componentIntstance
  - .debugElement
  - .nativeElement
  - By (helper class)
  - NO_ERROR_SCHEMA
  - Mocking strategies
  - End to End testing with Cypress or Protractor

More info

Elsewhere on the interwebz…

Testing documentation

ANGULAR FEATURES DOCS RESOURCES EVENTS BLOG Search

Testing

This guide offers tips and techniques for testing Angular applications. Though this page includes some general testing principles and techniques, the focus is on testing applications written with Angular.

Live examples

This guide presents tests of a sample application that is much like the *Tour of Heroes* tutorial. The sample application and all tests in this guide are available as live examples for inspection, experiment, and download:

- A spec to verify the test environment / download example .
- The first component spec with inline template / download example .
- A component spec with external template / download example .
- The QuickStart seed's AppComponent spec / download example .
- The sample application to be tested / download example .
- All specs that test the sample application / download example .
- A grab bag of additional specs / download example .Back to top

Introduction to Angular Testing

This page guides you through writing tests to explore and confirm the behavior of the application. Testing does the following:

1. Guards against changes that break existing code ("regressions").
2. Clarifies what the code does both when used as intended and when faced with deviant conditions.
3. Reveals mistakes in design and implementation. Tests shine a harsh light on the code from many angles. When a part of the application seems hard to test, the root cause is often a design flaw, something to cure now rather than later when it becomes expensive to fix.

https://angular.io/guide/testing

Repo – Angular Testing recipes

This repository Search Pull requests Issues Gist

juristr / angular-testing-recipes

Watch 3 Star 18 Fork 1

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs

Simple testing patterns for Angular version 2+

angular jasmine karma testing

43 commits 1 branch 0 releases 1 contributor

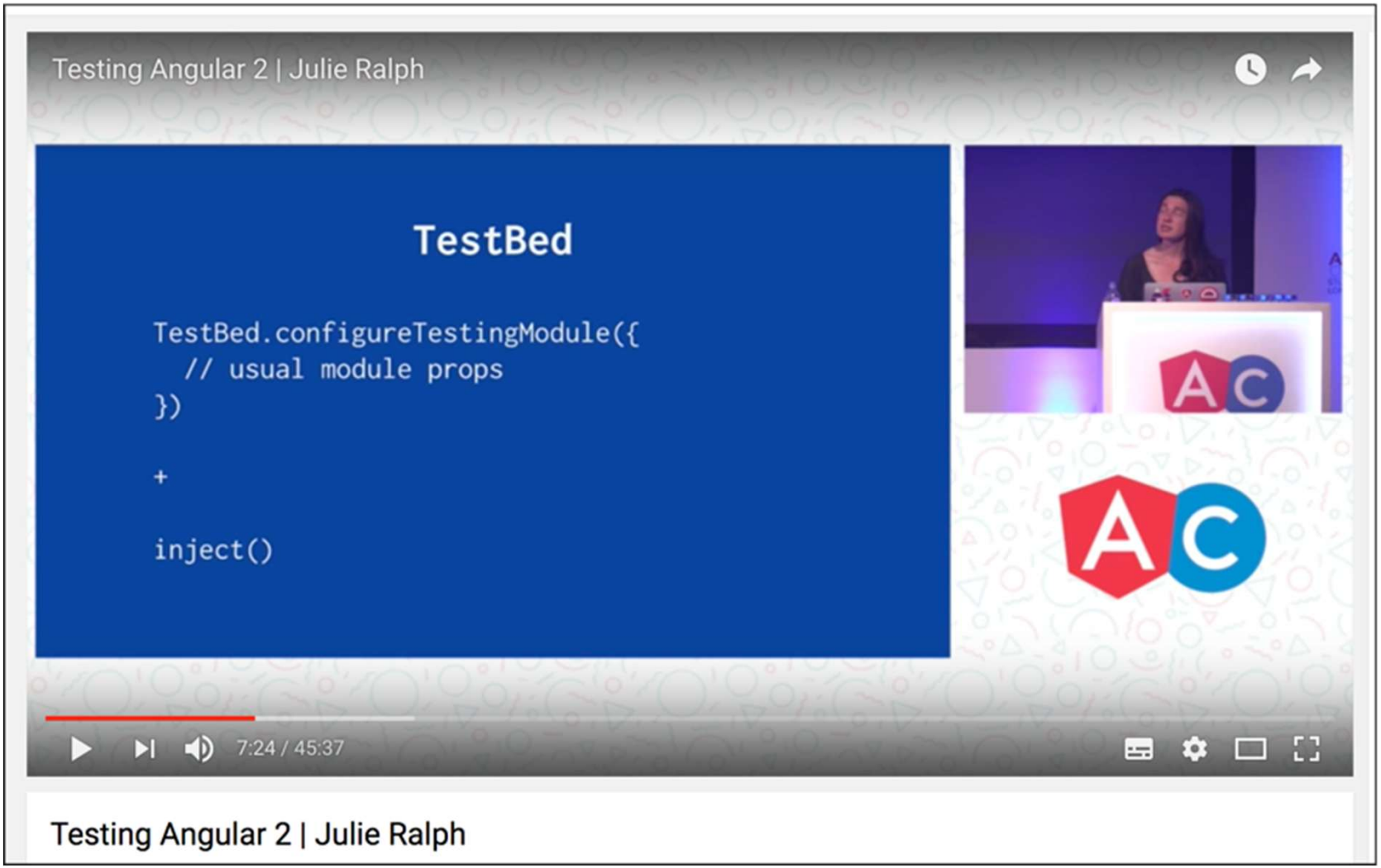
Branch: master New pull request Create new file Upload files Find file Clone or download

juristr committed on GitHub docs: adjust intro Latest commit a77caab on Jan 13

e2e	chore: base setup with CLI	a month ago
src	fix: missing parentheses	a month ago
.gitignore	chore: add .vscode to gitignore	a month ago
.travis.yml	chore: add travis config file	a month ago
README.md	docs: adjust intro	a month ago
angular-cli.json	chore: base setup with CLI	a month ago
karma.conf.js	feat: add karma mocha reporter	a month ago
package.json	chore(packages): upgrade TypeScript reference	a month ago
protractor.conf.js	chore: base setup with CLI	a month ago

https://github.com/juristr/angular-testing-recipes

Videos on testing



https://www.youtube.com/watch?v=f493Xf0F2yU

Good introductory article + video

DZone / Web Dev Zone

Over a million developers have joined DZone. Sign In / Join

REFCARDZ GUIDES ZONES | AGILE BIG DATA CLOUD DATABASE DEVOPS INTEGRATION IOT JAVA MOBILE PERFORMANCE SECURITY WEB DEV

Let's be friends:

Testing With Angular 2: Some Recipes (Talk and Slides)

Juri Stumpflohner reflects on his recent talk about diving deeper into testing Angular 2 apps. He also links to a dedicated code repository on GitHub with the purpose of collecting testing recipes for various scenarios one might encounter while testing Angular applications.

by Juri Strumpflohner MVB · Jan. 16, 17 · Web Dev Zone

Like (-2) Comment (0) Save Tweet 4,327 Views

Join the DZone community and get the full member experience. JOIN FOR FREE

Start coding today to experience the powerful engine that drives data application's development, brought to you in partnership with Qlik.

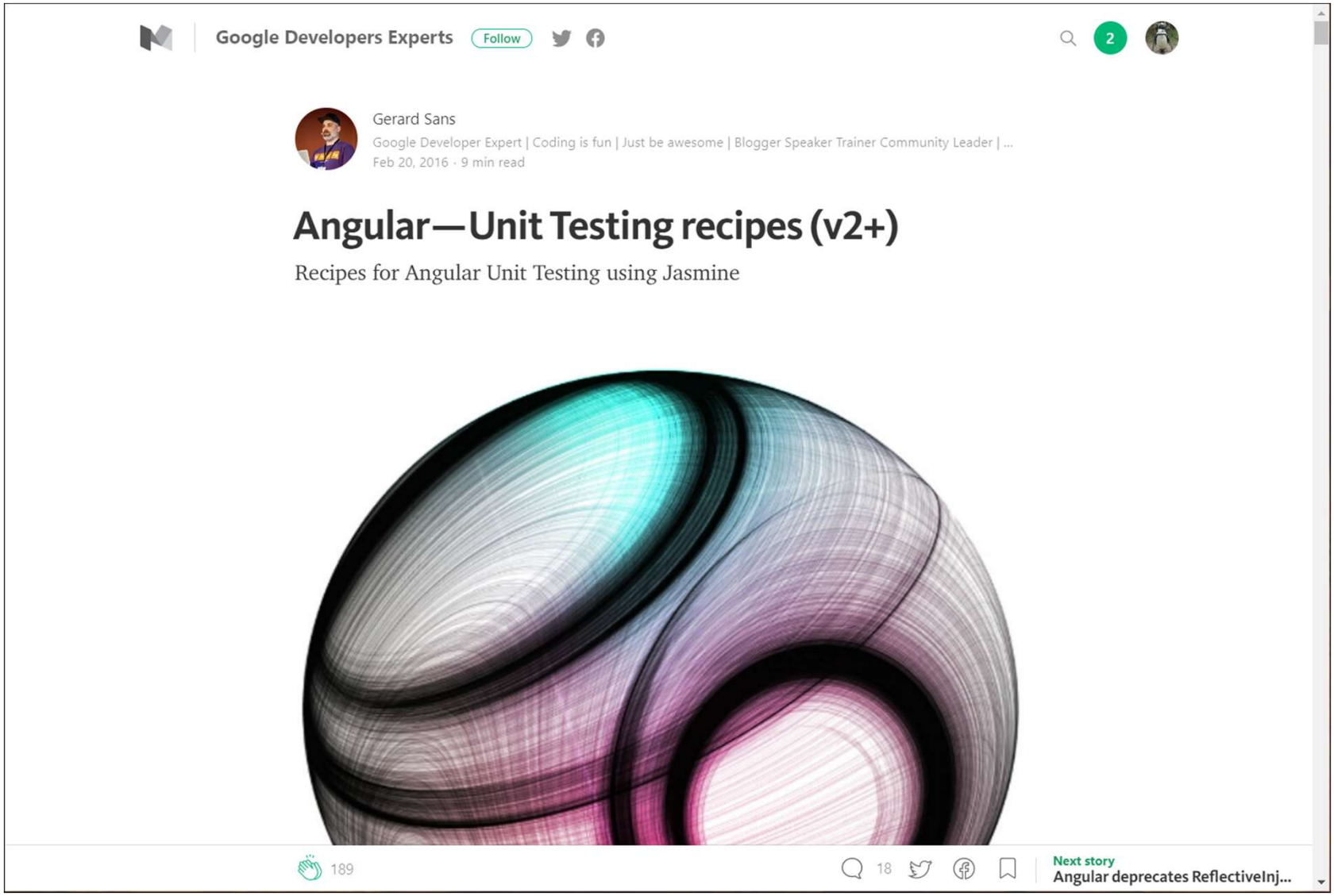
I recently wanted to dive deeper into testing Angular applications, in specific on how to write proper unit tests for some common scenarios you might encounter.

Dave, the organizer of the Angular Hamburg Meetup group, asked me whether I'd be interested in

Subscribe

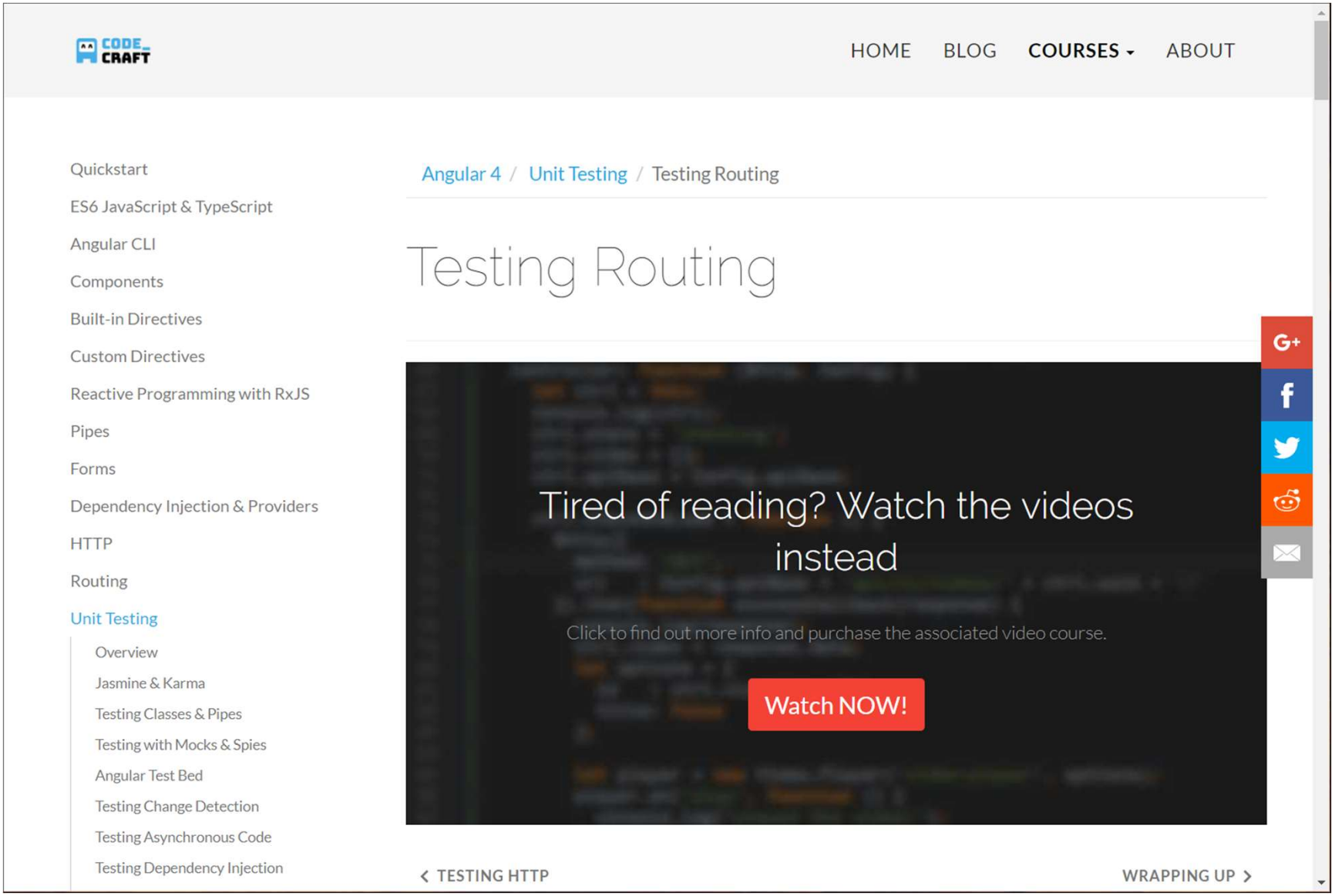
https://dzone.com/articles/talk-testing-with-angular-some-recipes

Gerard Sans on testing



https://medium.com/google-developer-experts/angular-2-unit-testing-with-jasmine-defe20421584

Testing Routing



https://codecraft.tv/courses/angular/unit-testing/routing/