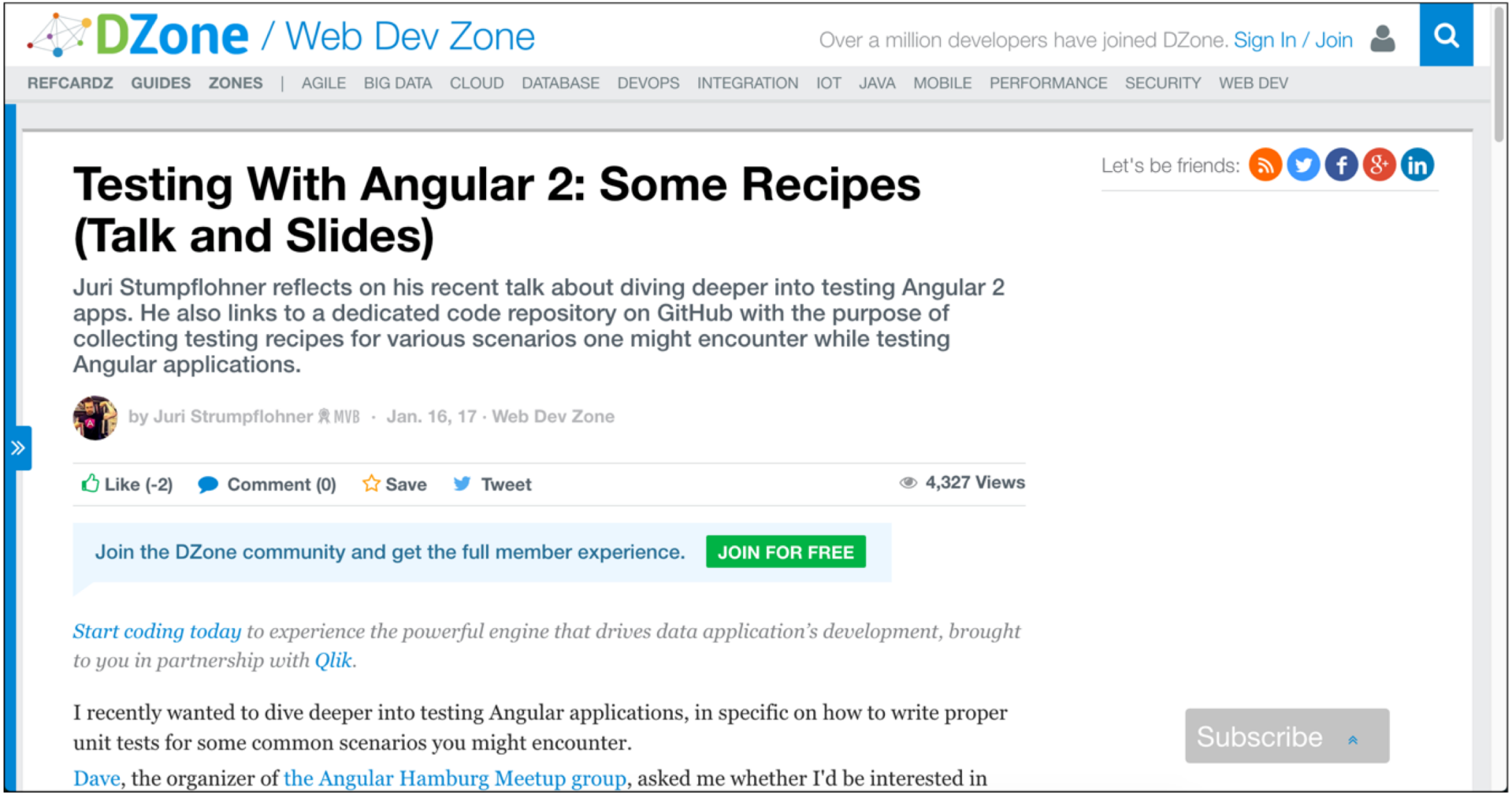
Angular Fundamentals
Module 8 – Testing introduction

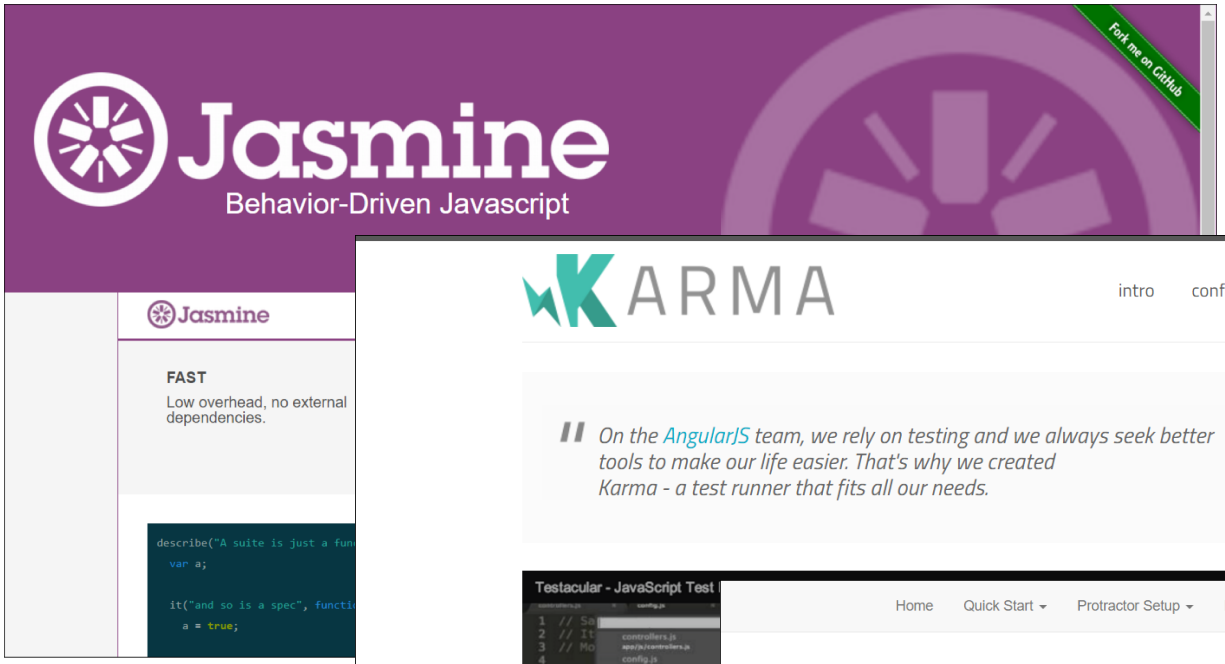
Peter Kassenaar
info@kassenaar.com

Good introductory article



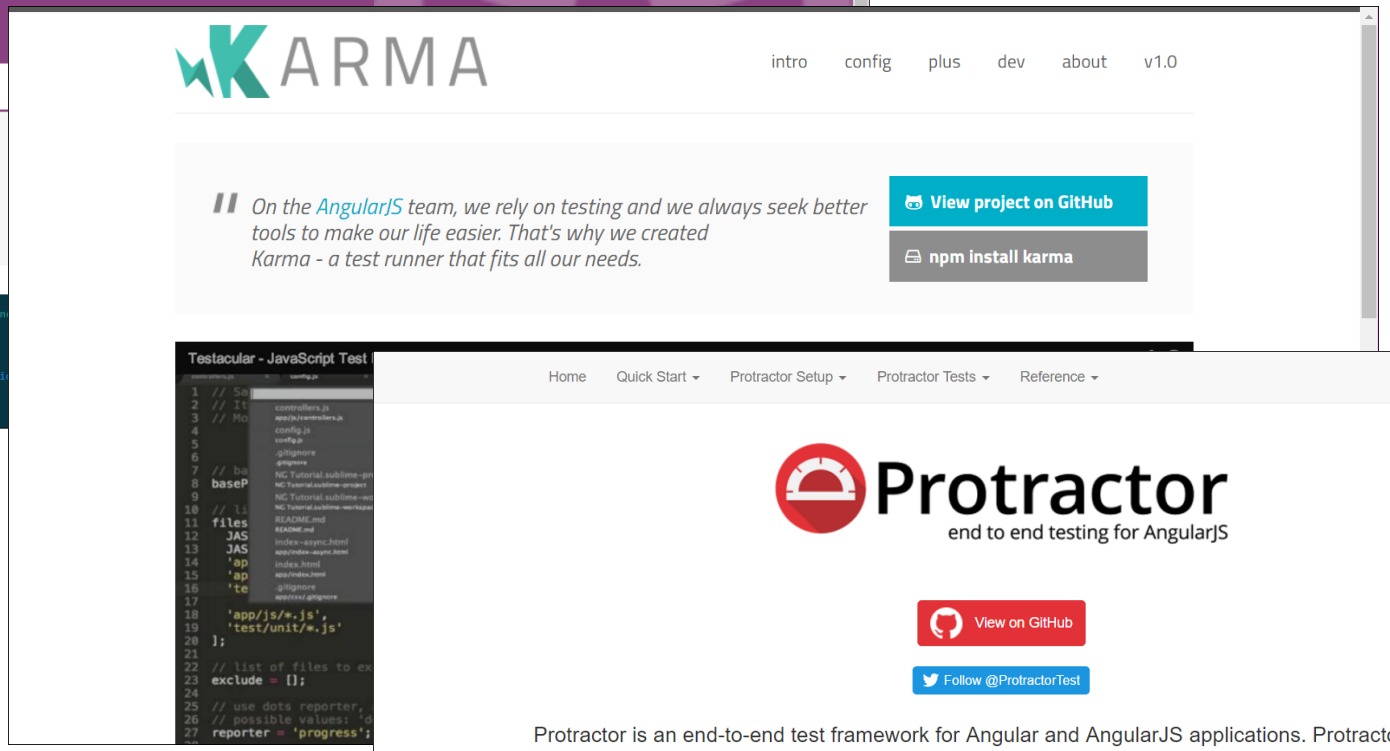
https://dzone.com/articles/talk-testing-with-angular-some-recipes

Tooling



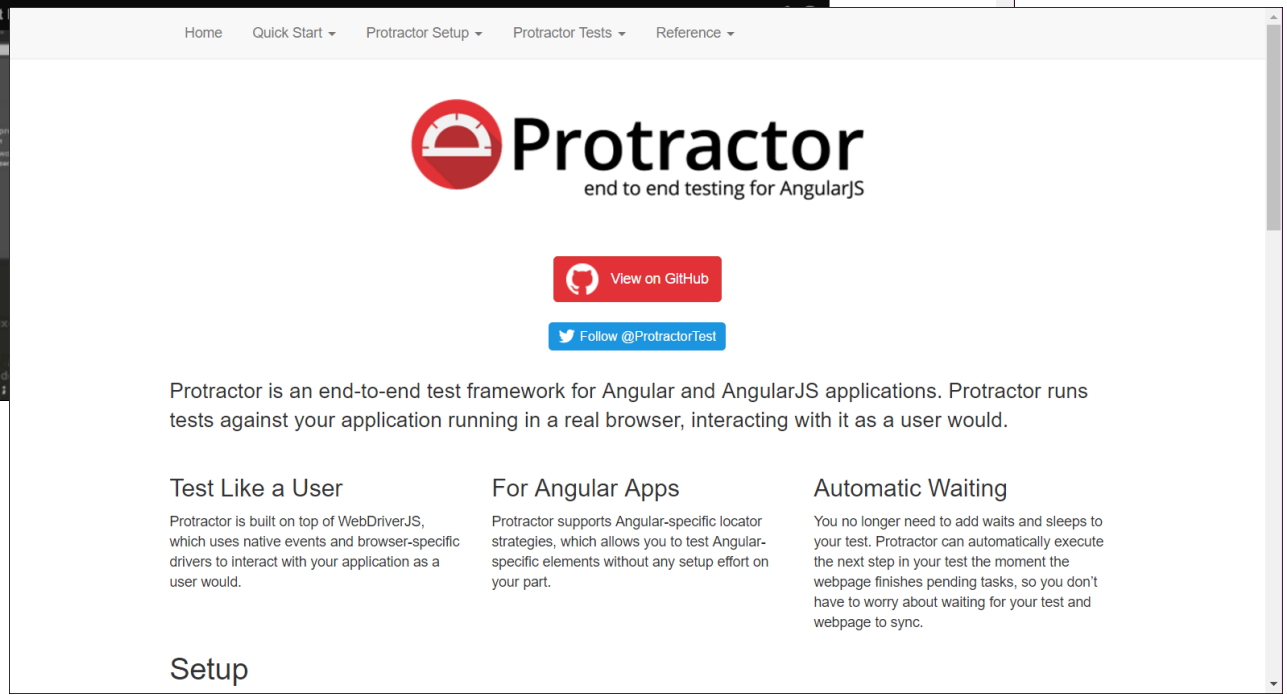
Jasmine:

Write Unit Tests & Specs



Karma:

Test Runner



Protractor:

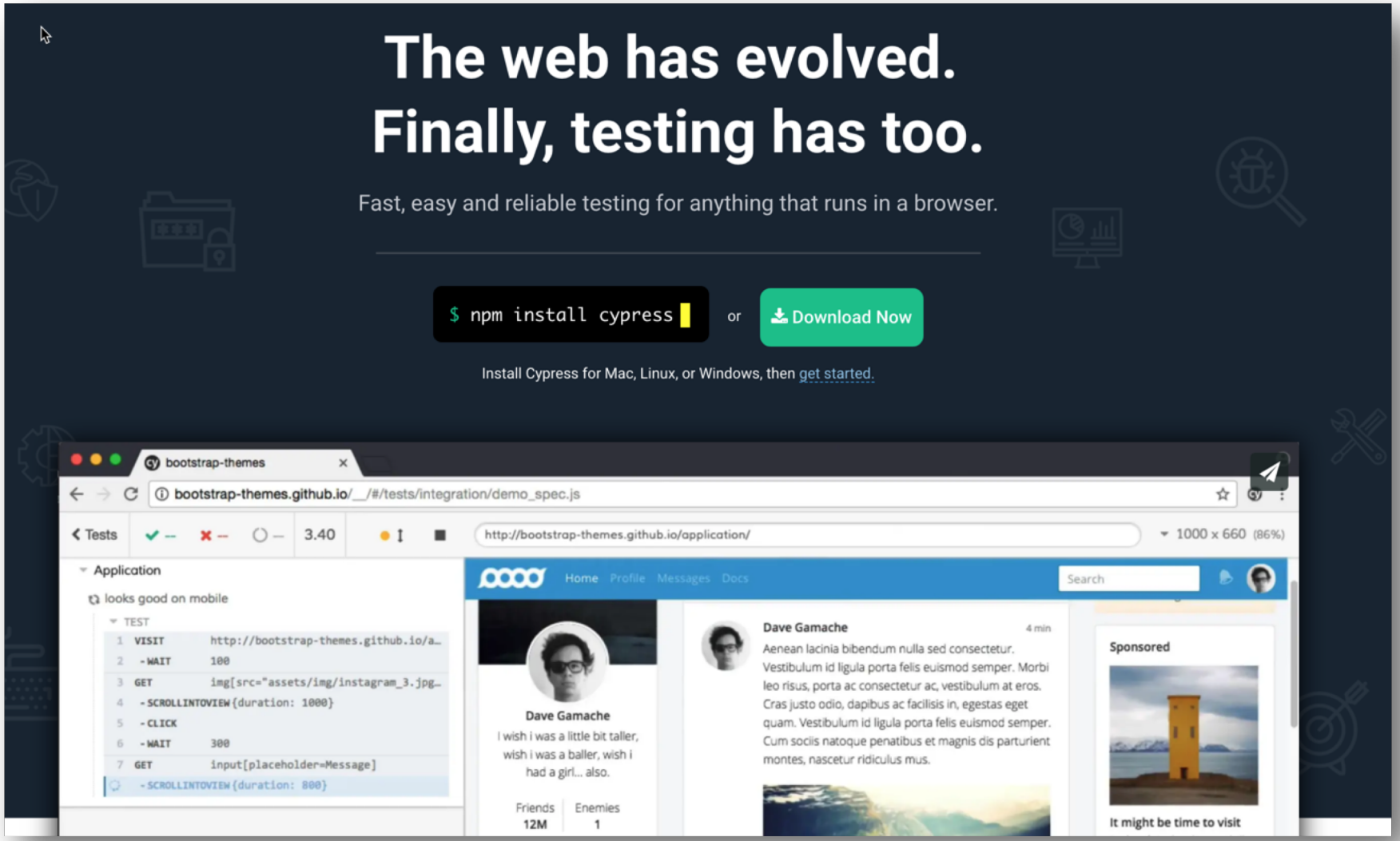
End to end testing

https://jasmine.github.io/

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

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

An alternative to E2E-testing - Cypress



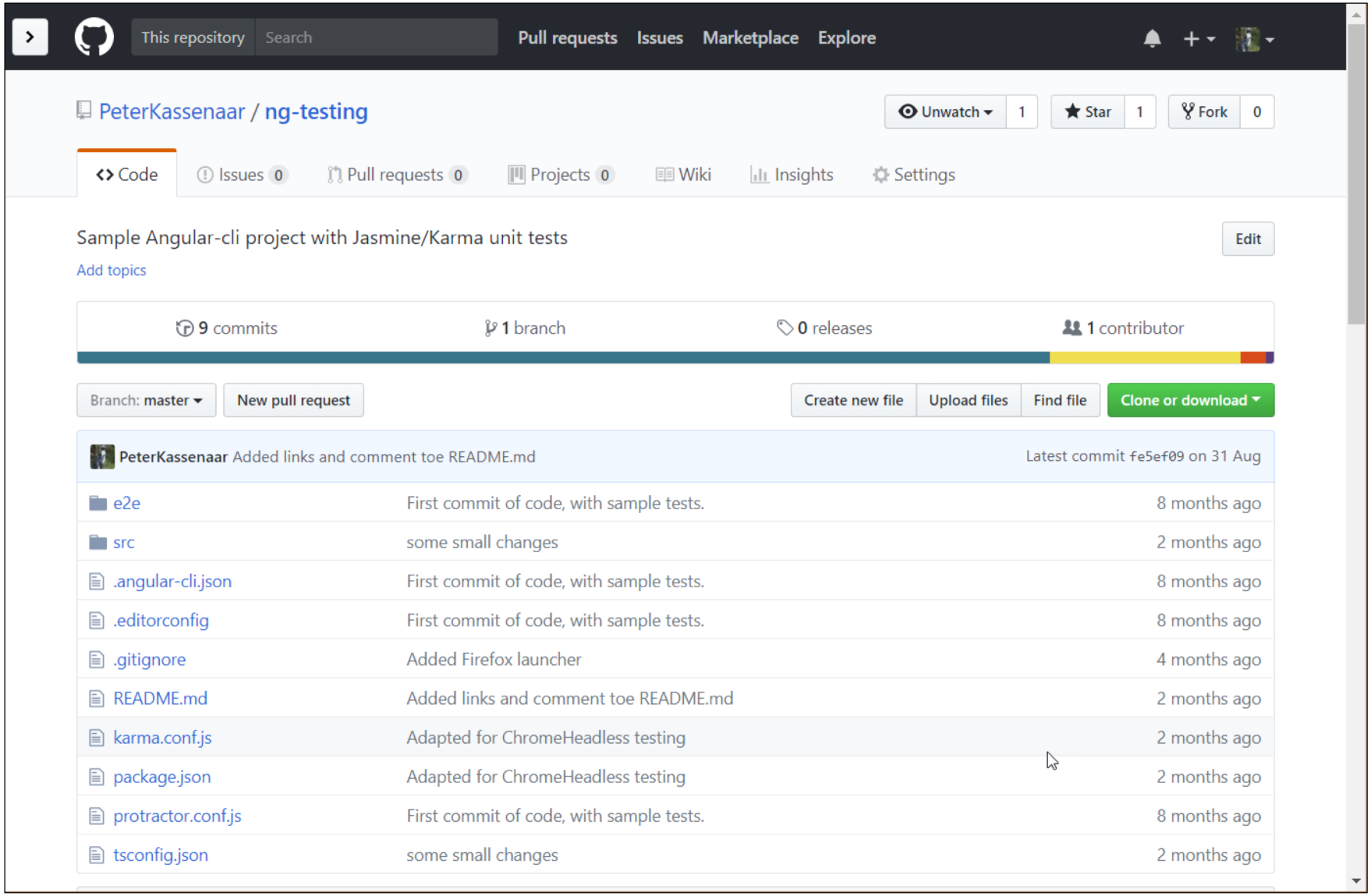
https://www.cypress.io/

```
// Karma configuration file, see link for more information
// https://karma-runner.github.io/0.13/config/configuration-file.html

module.exports = function (config) {
  config.set({
    …
    reporters: ['progress', 'kjhtml'],
    port: 9876,
    colors: true,
    logLevel: config.LOG_INFO,
    autoWatch: true,
    browsers: ['ChromeHeadless'],
    singleRun: false
  });
};
```

General Testing

General Jasmine syntax for testing classes and models

Generic repo – not in `/examples`!



https://github.com/PeterKassenaar/ng-testing

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 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`

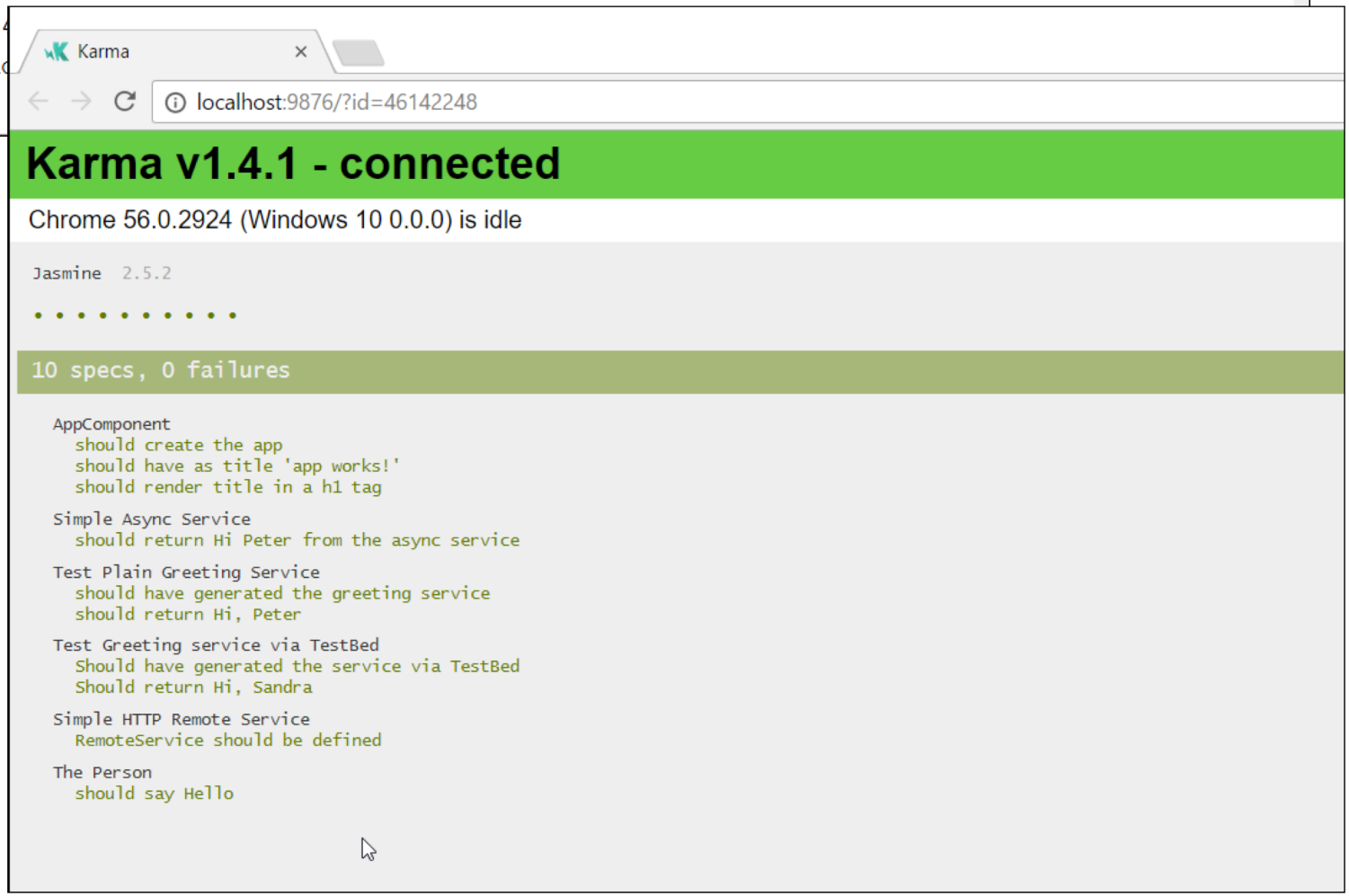
- Manual project? Install & setup karma and jasmine yourself
  - Create and adapt `karma.conf.js`
  - `karma --init`
  - Install and setup jasmine reporters
  - **We're not doing that here.**
  - If you really want to,

    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",
   …
}
…
```

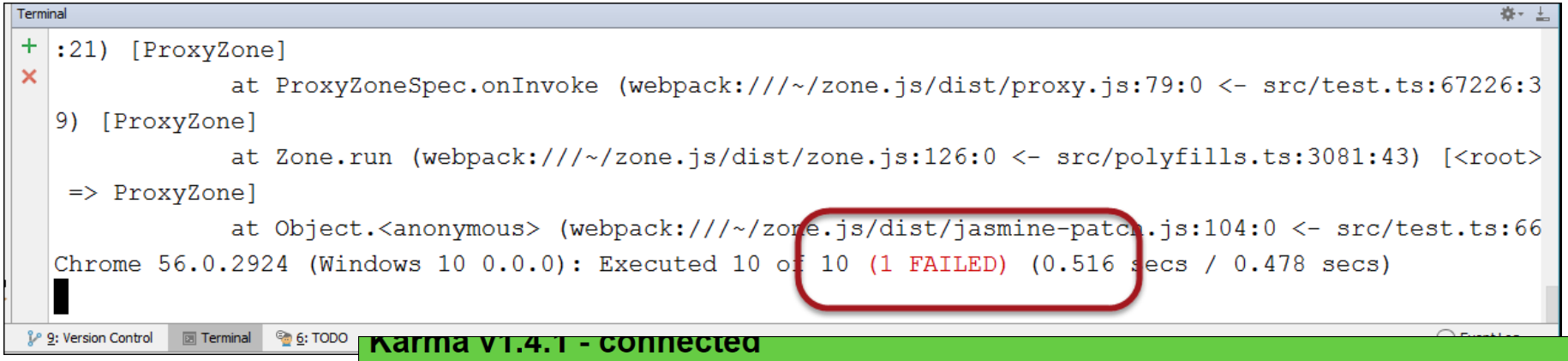
Browser like...

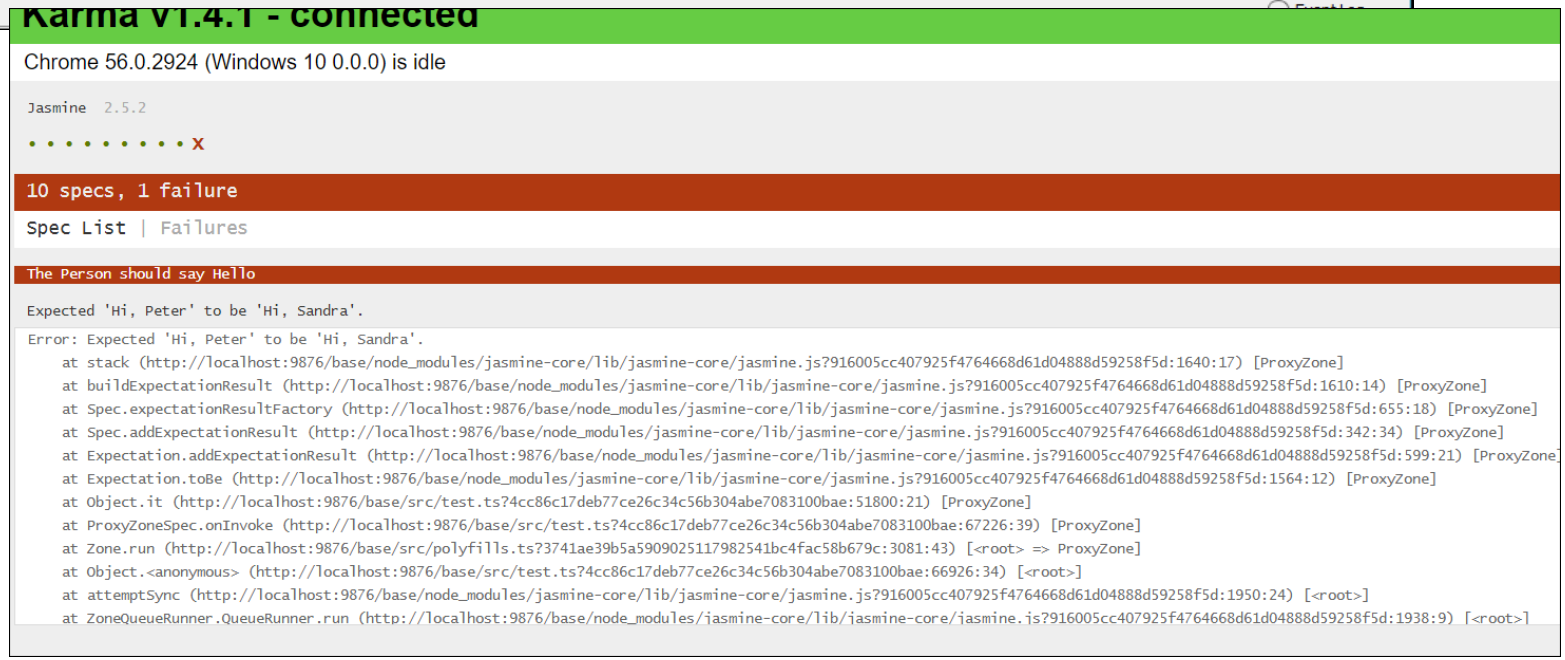
```
C:\Users\Peter Kassenaar\Desktop\ng-testing>ng test
10 03 2017 16:50:40.157:WARN [karma]: No captured browser, open http://localhost:9876/
10 03 2017 16:50:40.170:INFO [karma]: Karma v1.4.1 server started at http://0.0.0.0:9876/
10 03 2017 16:50:40.171:INFO [launcher]: Launching browser Chrome with 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 4614224
Chrome 56.0.2924 (Windo
```



If anything goes wrong w/ default settings...

Hard to read output:





Solution: install custom reporters

On to Angular-specific terms

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

Basic component test included

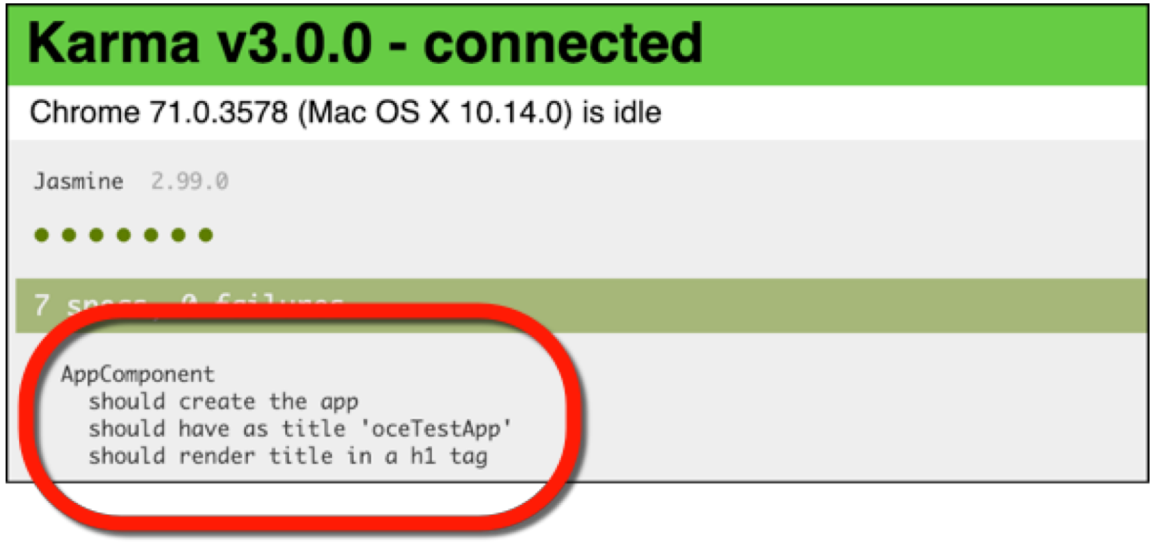
```
1  import { TestBed, async } from '@angular/core/testing';
2  import { AppComponent } from './app.component';
3
4  describe('AppComponent', () => {
5    beforeEach(async(() => {
6      TestBed.configureTestingModule({
7        declarations: [
8          AppComponent
9        ],
10     }).compileComponents();
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 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 actual component instance
- `detectChanges` – run change detection manually

Running the tests

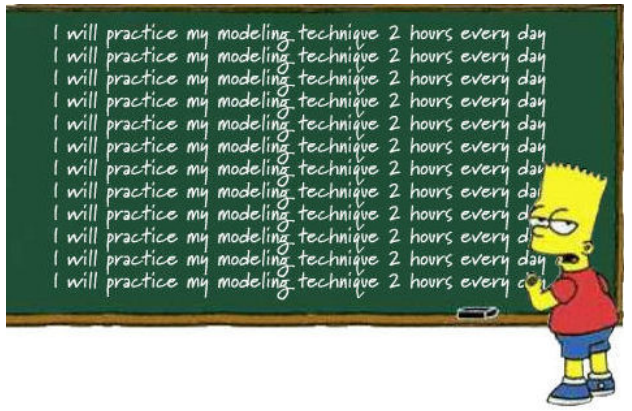
- `ng test`
  - Run all the tests in the .spec-files in the project
  - You can keep this running in the background!
    - It might slow down development
- `fit` - run this test only
- `xit` - run all tests except this one



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 funcionality for adding and deleting a city
- Write tests for the new component.



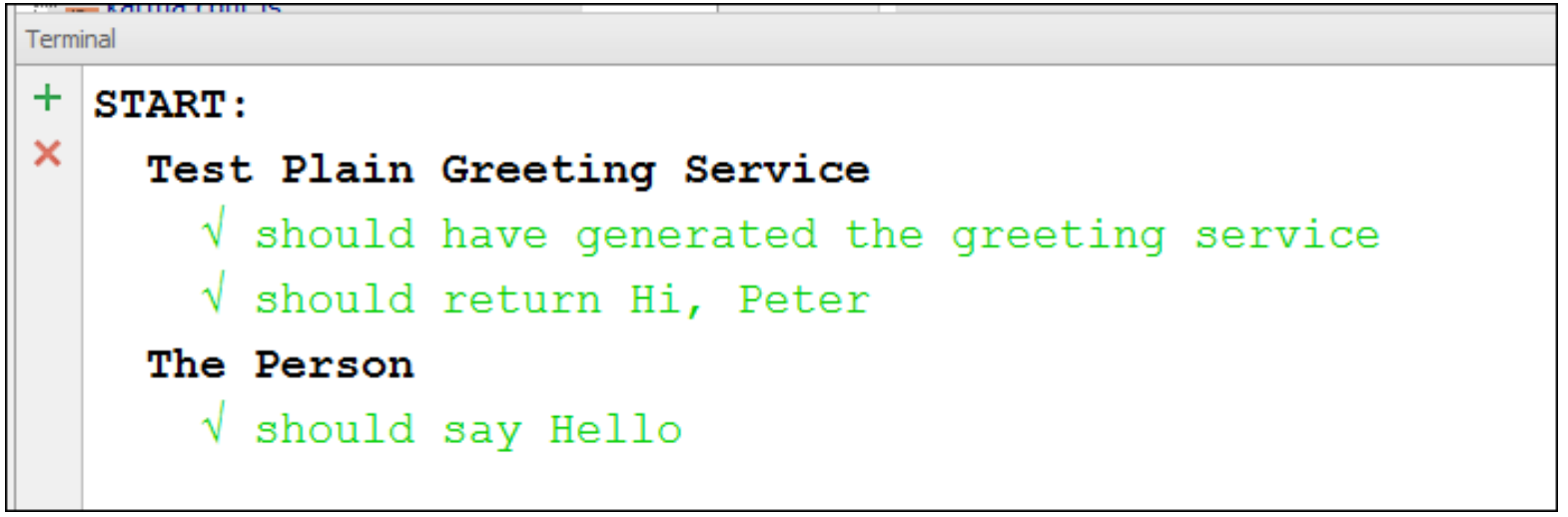
Testing a single service

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

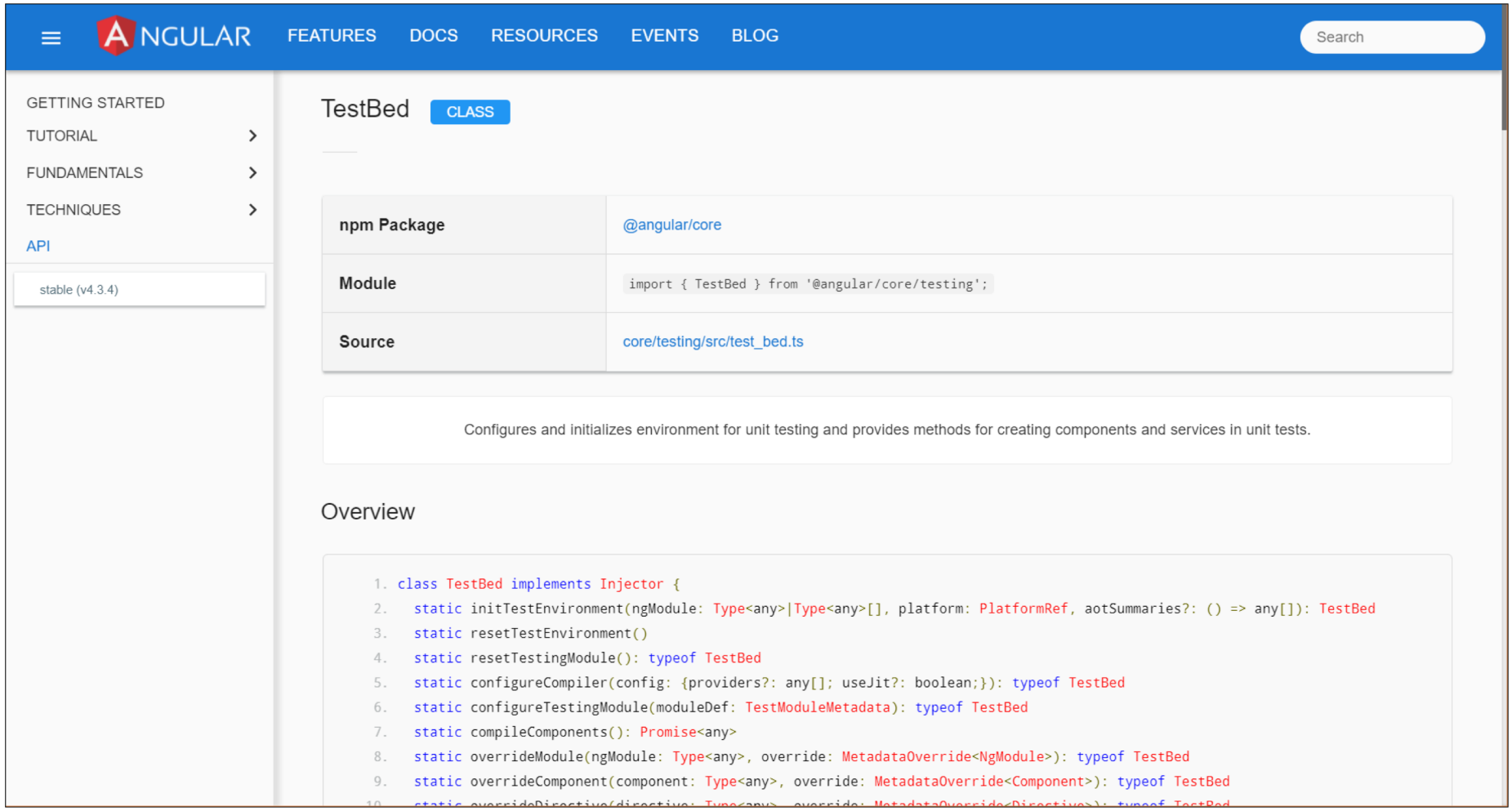
- In testing we have the same concept of modules, using `TestBed`
- TestBed has a method `.configureTestingModule()` to mimic an `ngModule`.
- You only specify the stuff you need!
  - No need to build the complete module, importing all dependencies

```
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.”



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

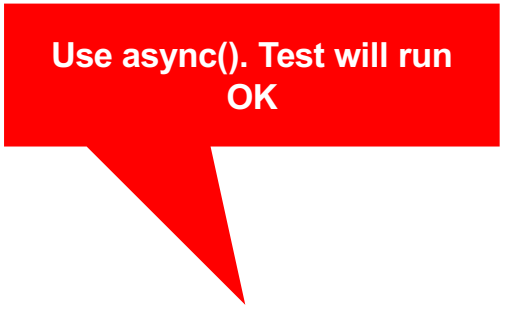
Async behavior

- 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, XXXX');
      })
});
```

Using Angular `async` and `fakeAsync`

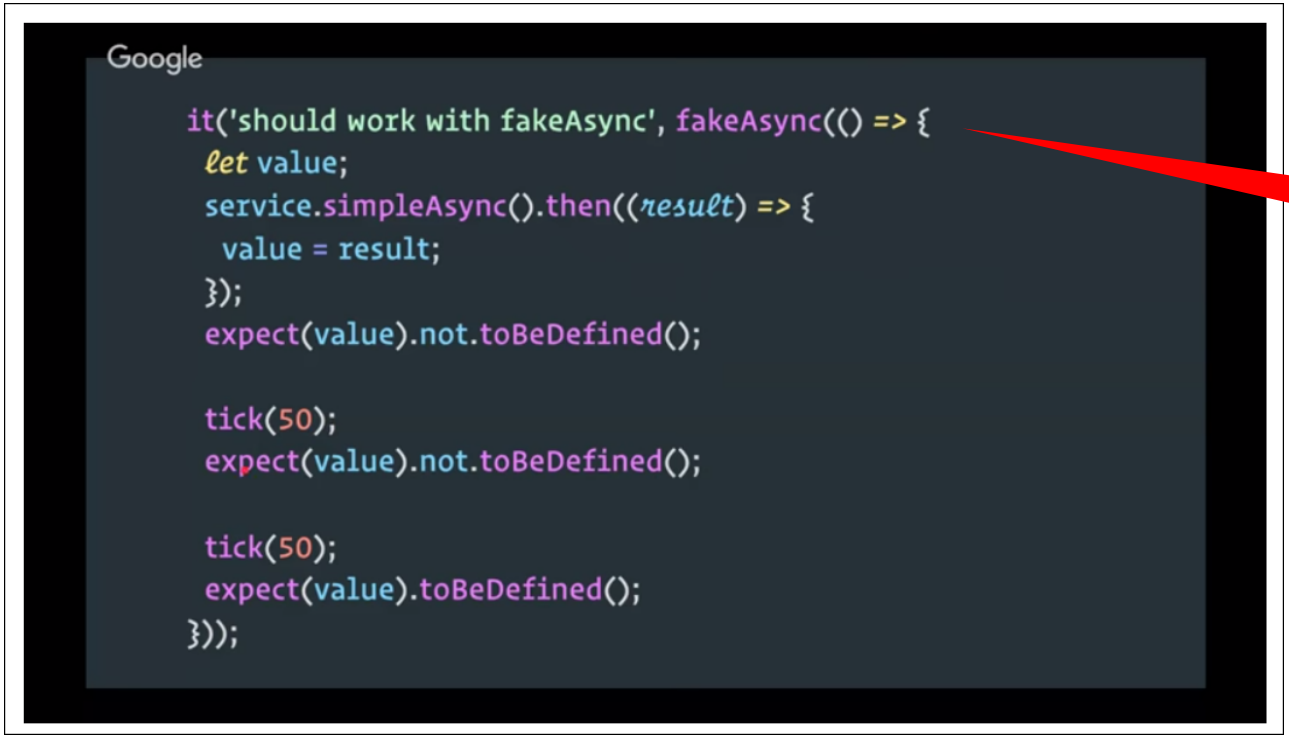
- Solution: import and use Angular `async()` or `fakeAsync()` construct and wrap the expect statement in this function
- Jasmine will wait for the `async()` function to return and *then* perform the test




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

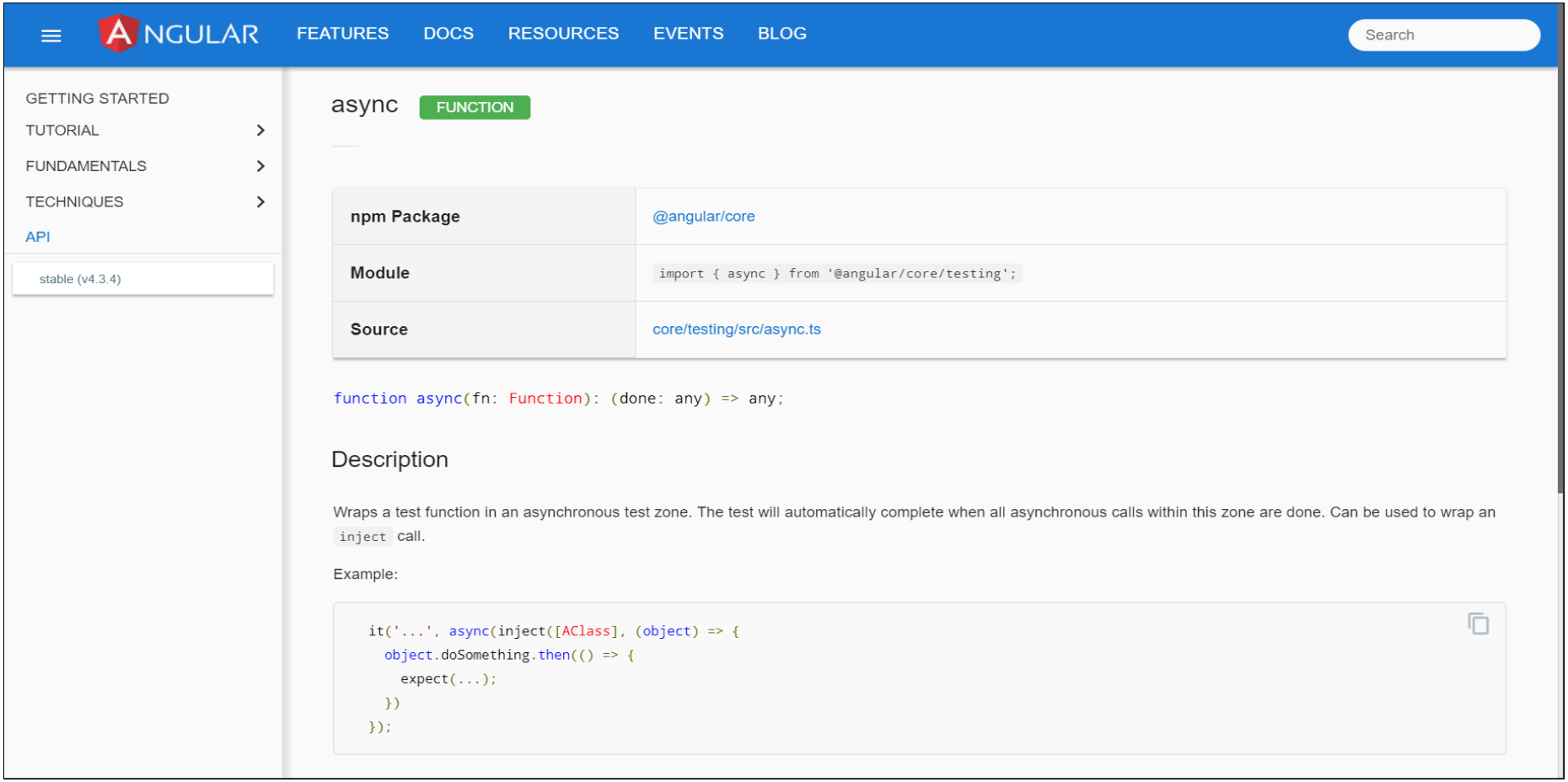
Async vs. fakeAsync

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



"fakeAsync() makes your async code synchronous"

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



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

Mocking backend

Testing asynchrounous XHR-calls

Testing XHR calls

- Setup remote service to fetch some data over http

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

Testing this will fail – attempt to call backend

```
beforeEach(() => {
   TestBed.configureTestingModule({
      imports  : [HttpModule],
      providers: [RemoteService]
   });
   remoteService = TestBed.get(RemoteService);
});
```

```
it('Should return people.json', async(() => {
   let people;
   remoteService.getPeople().subscribe((result) => {
      people = result
   });
   expect(people).toBeDefined();
}));
```

Solution: create Mock Backend

- Creating a `mockBackend` and `mockResponse`
- Provide the `mockBackend`
- Override the `Http`-specific Angular stuff

"When the application asks for XHRBackend, give him the class MockBackend"

Import mockBackend, mockConnection

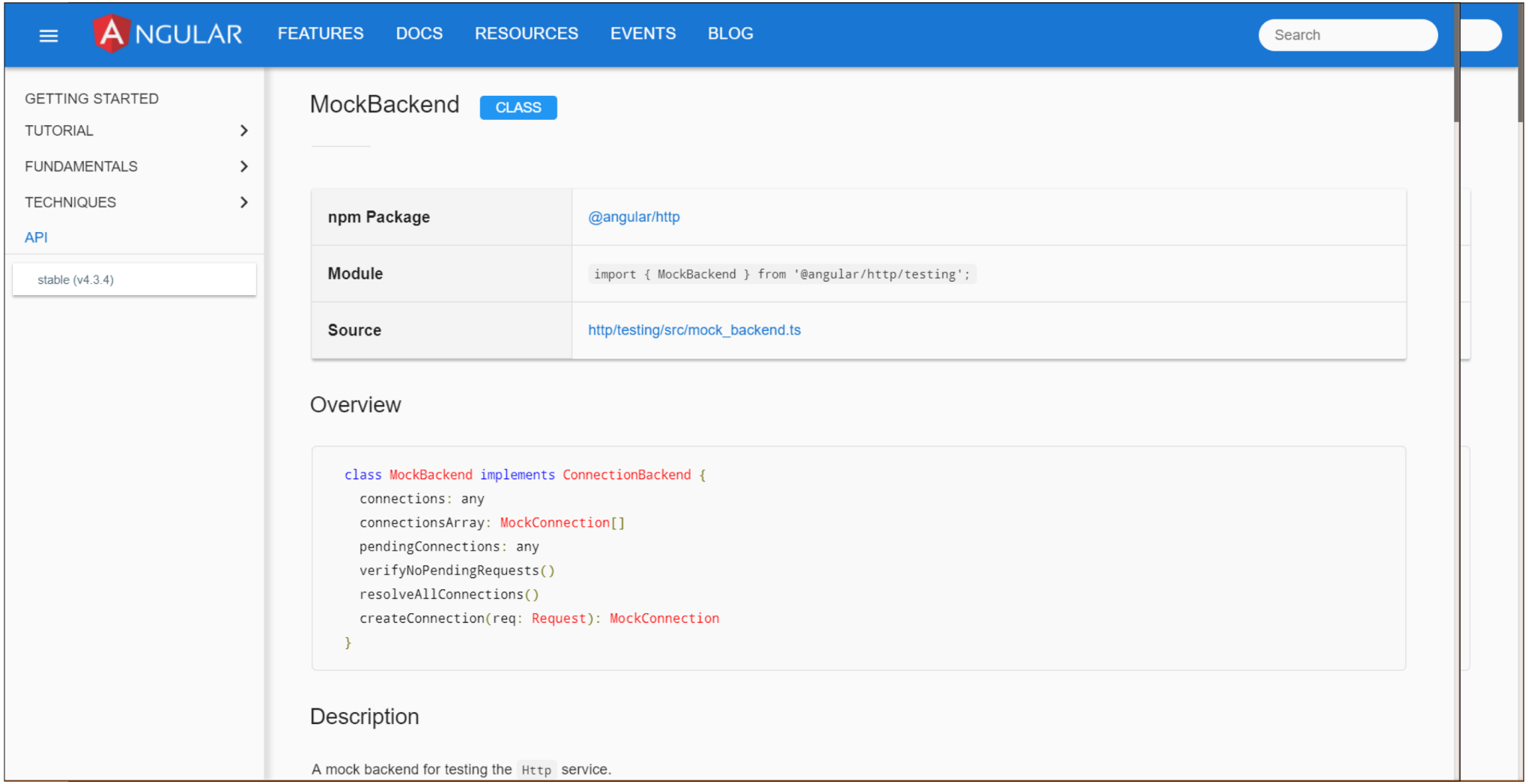
```
let remoteService, mockBackend, mockResponse;
beforeEach(() => {
   TestBed.configureTestingModule({
      imports  : [HttpModule],
      providers: [
         RemoteService,
         MockBackend,
         {provide: XHRBackend, useClass: MockBackend}
      ]
   });
   remoteService = TestBed.get(RemoteService);
   mockBackend   = TestBed.get(MockBackend);
   mockResponse  = new ResponseOptions([{
      name : 'Peter',
      email: 'info@kassenaar.com'
   }, {
      name : 'Sandra',
      email: 'sandra@kassenaar.nl'
   }])
});
```

Write test for mocked backend

- `mockBackend` has a property `connections`
- Provide this a `connection`, of type `MockConnection`
- Mock the response, using the data defined in your test

```
it('Should return people.json', async(() => {
   // Create mock backend connection
   mockBackend.connections.subscribe((conn: MockConnection) => {
      conn.mockRespond(new Response(mockResponse))
   });
   remoteService.getPeople().subscribe((result) => {
      expect(result).toBeDefined();
   });
}));
```

Documentation on MockBackend and - Connection



https://angular.io/api/http/testing/MockBackend

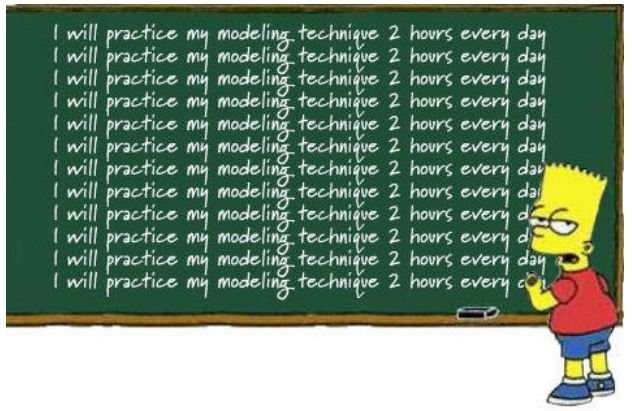
https://angular.io/api/http/testing/MockConnection

Summary

- What have we learned?
  - Jasmine Syntax: `describe()`, `beforeEach()`, `it()`, `expect()`
  - `TestBed.configureTestingModule()`
  - `TestBed.get()`
  - Testing Classes
  - Testing Services
- Mocking backend
  - `MockBackend`, `MockConnection`, `MockResponse`
  - `MockRespond`

Workshop

- Study the example `../400-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 `MockBackend` and `MockResponse`.
  - 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

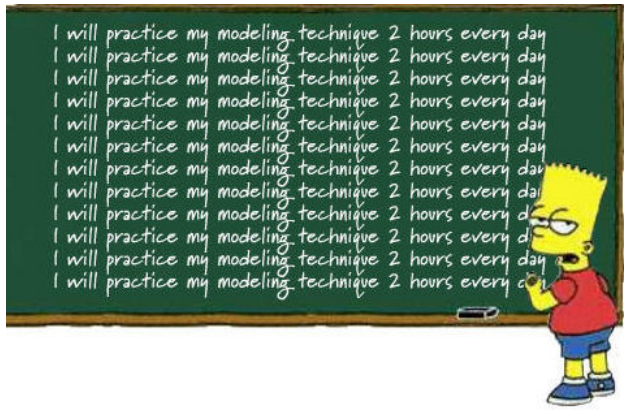
Jasmine spyOn()

`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

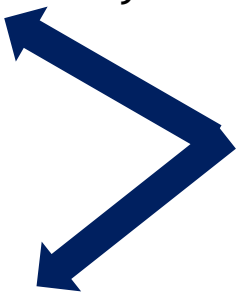
- Simply provide empty components to the testsuite
- *Pro* – Component would compile as at run time
- *Con* – duplicate code, not a nice view, more overhead.

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

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

#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

Automate browsers to perform scenario's

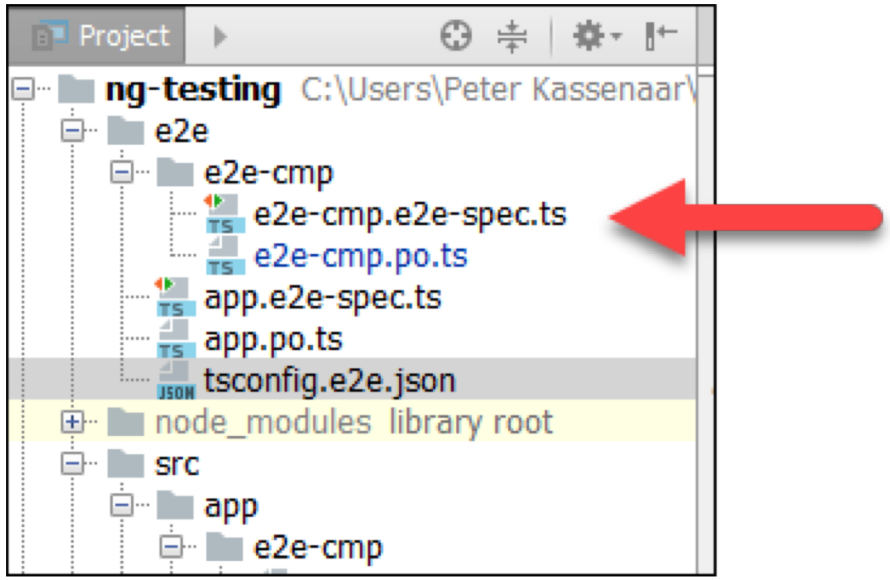
Developed by Team Angular – wrapper around selenium

Parts of e2e tests

- Folder: `/e2e`
- Page Objects – describe the page 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

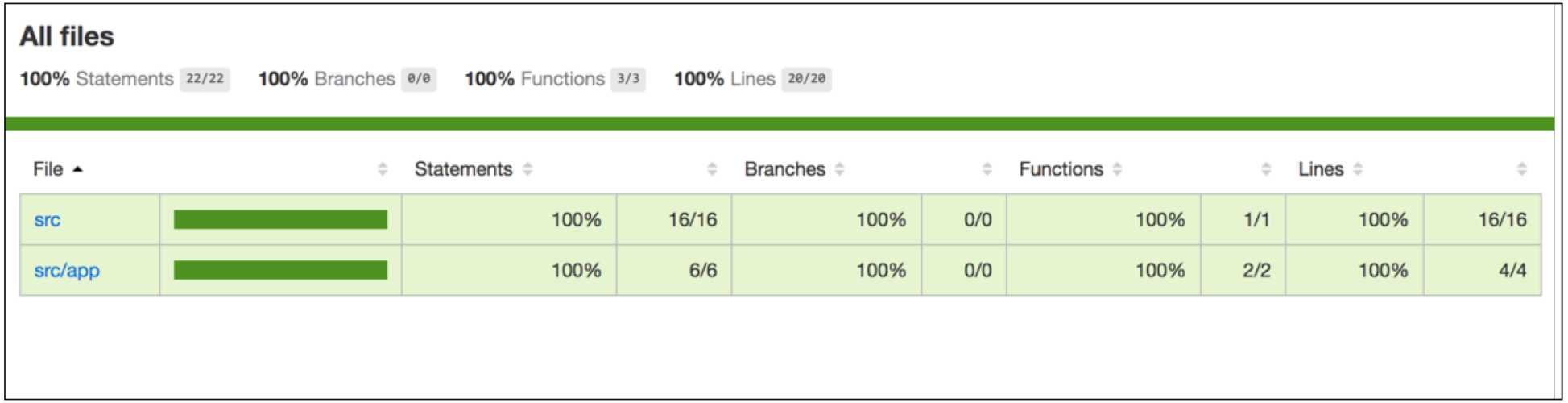
Executing e2e tests

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



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`



Summary

What have we learned

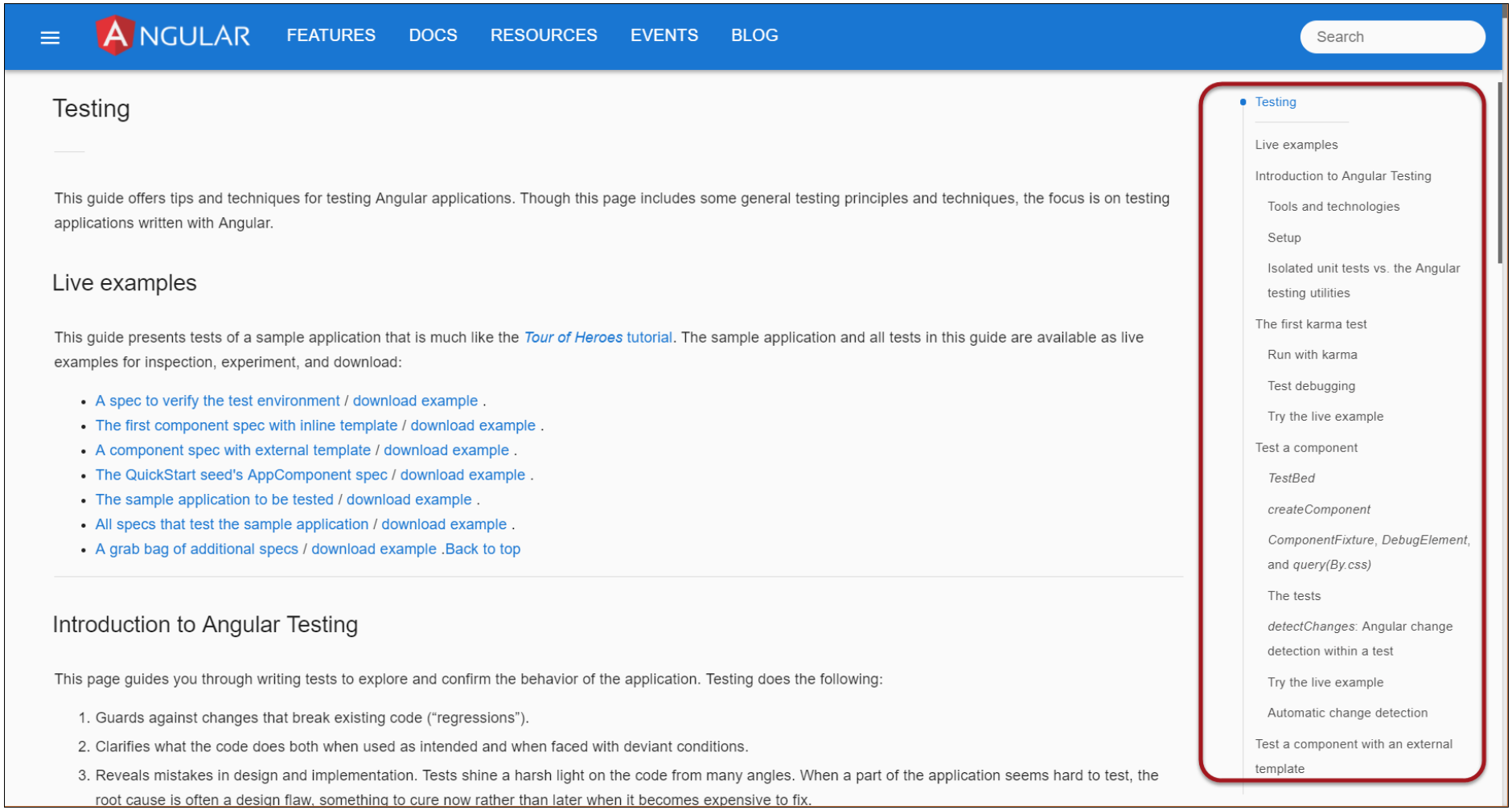
Summary

- We learned about:
  - ComponentFixture
  - .compileComponents()
  - .detectChanges()
  - .componentIntstance
  - .debugElement
  - .nativeElement
  - By (helper class)
  - NO_ERROR_SCHEMA
  - Mocking strategies

More info

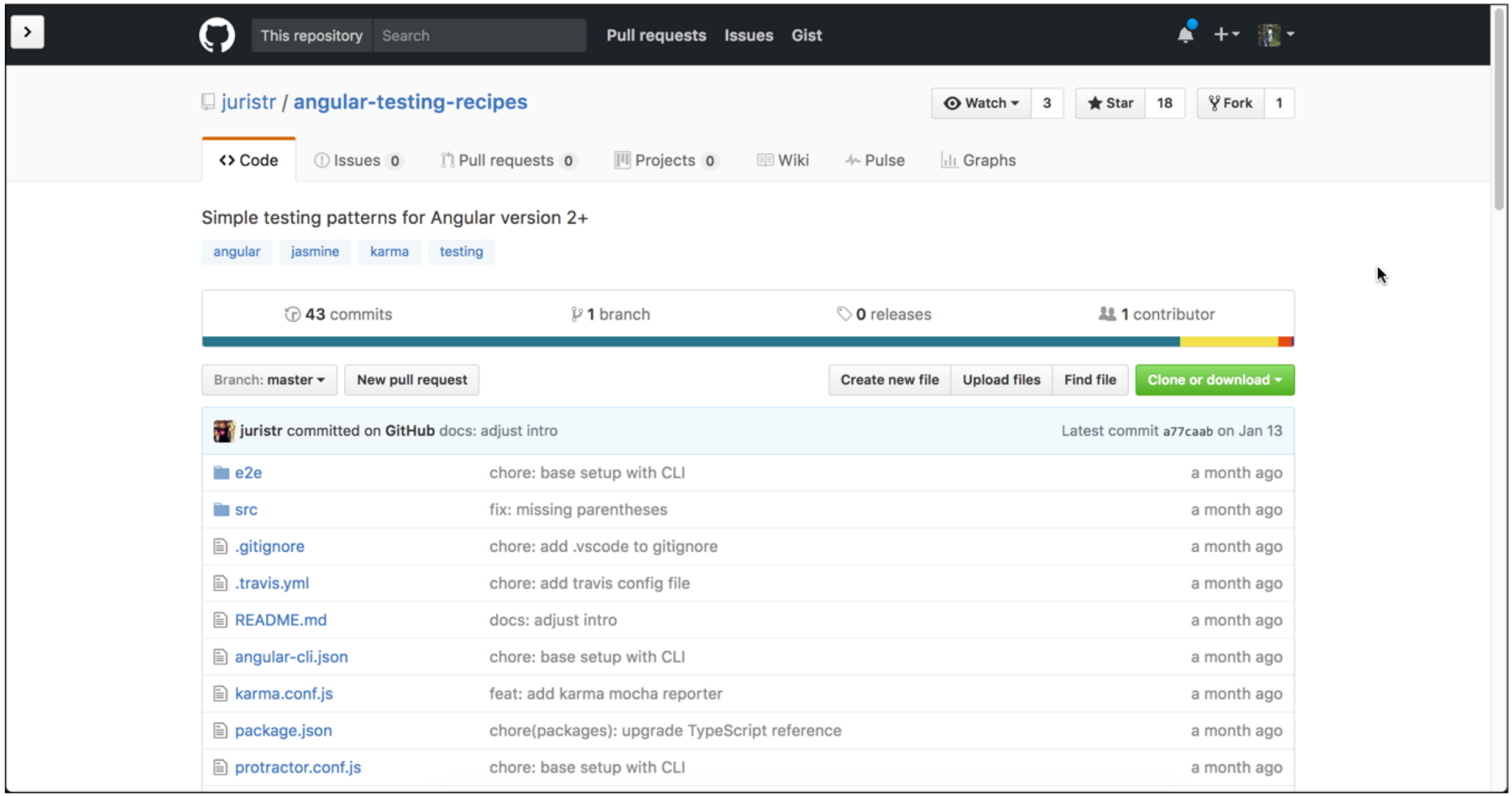
Elsewhere on the interwebz…

Testing documentation



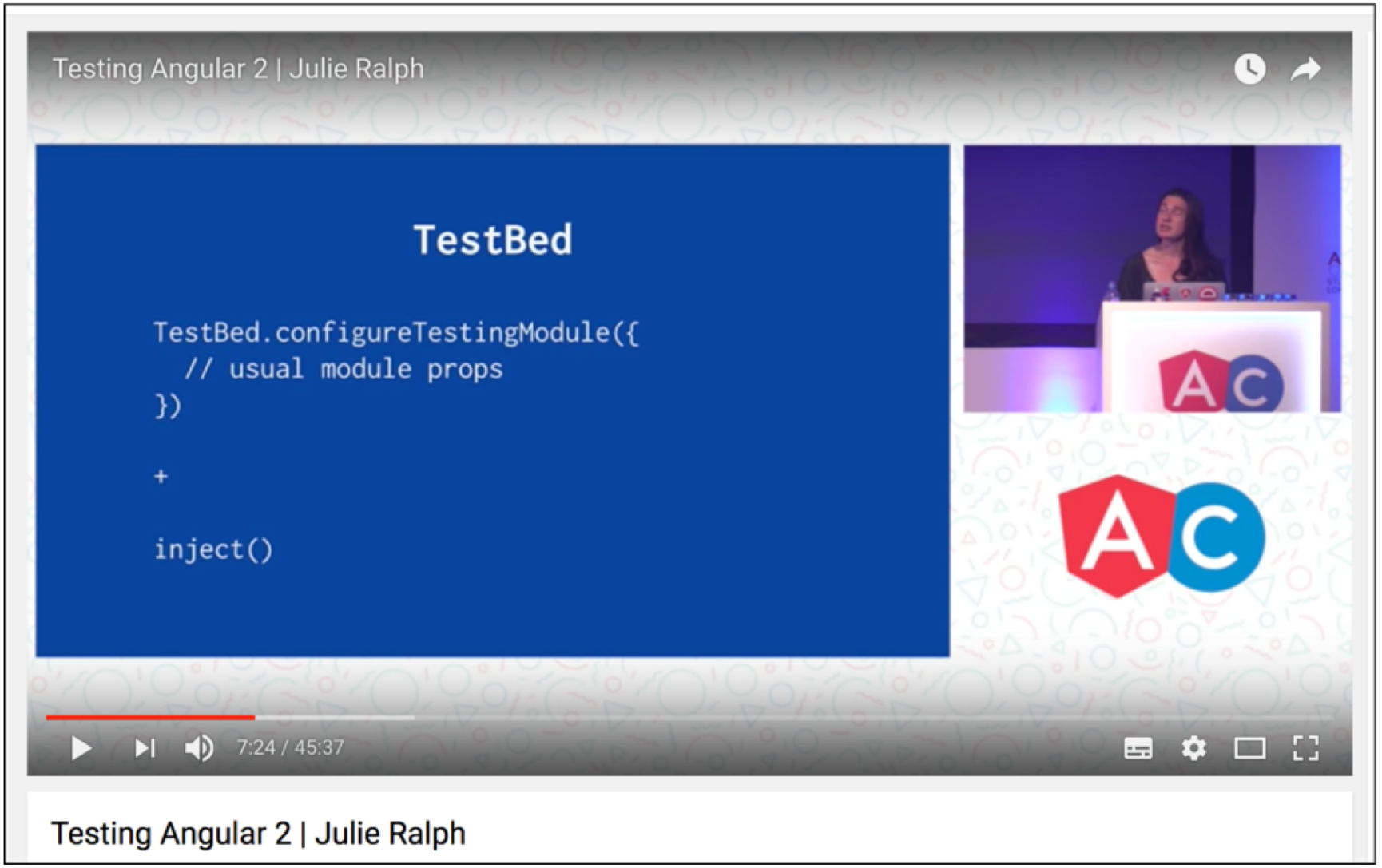
https://angular.io/guide/testing

Repo – Angular Testing recipes



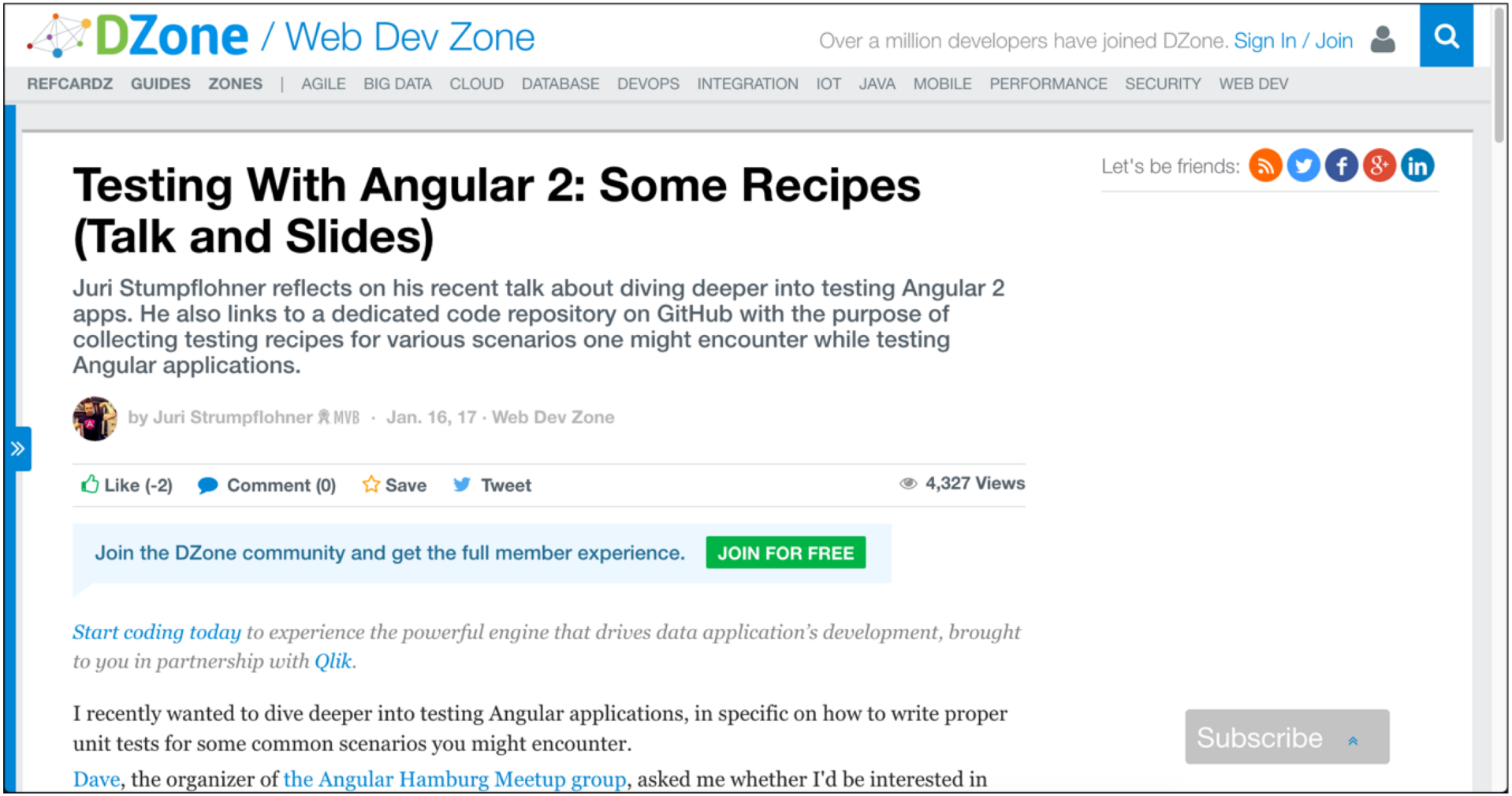
https://github.com/juristr/angular-testing-recipes

Videos on testing



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

Good introductory article + video



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

Gerard Sans on testing

Google Developers Experts Follow

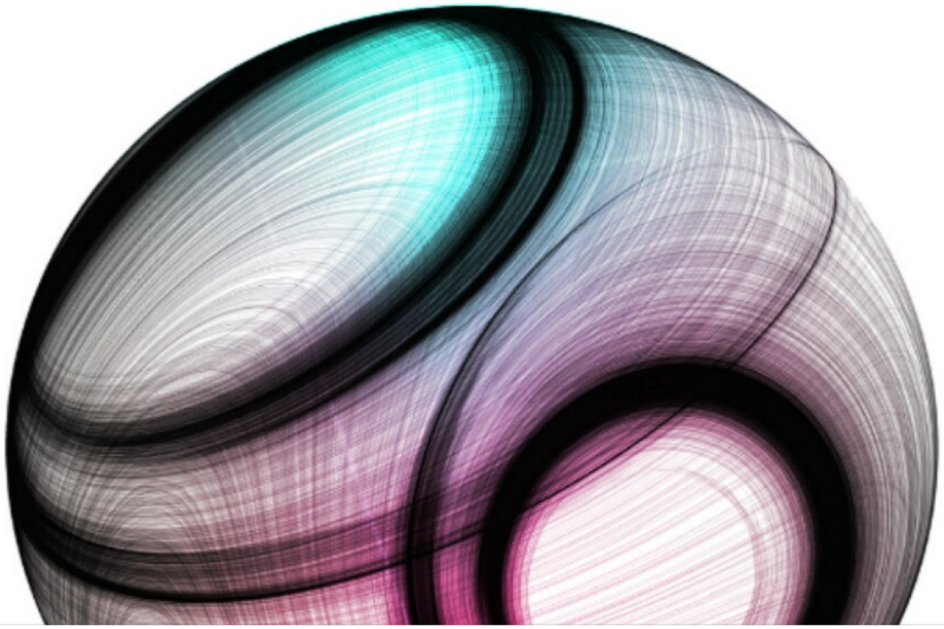
Gerard Sans

Google Developer Expert | Coding is fun | Just be awesome | Blogger Speaker Trainer Community Leader | ...

Feb 20, 2016 · 9 min read

Angular—Unit Testing recipes (v2+)

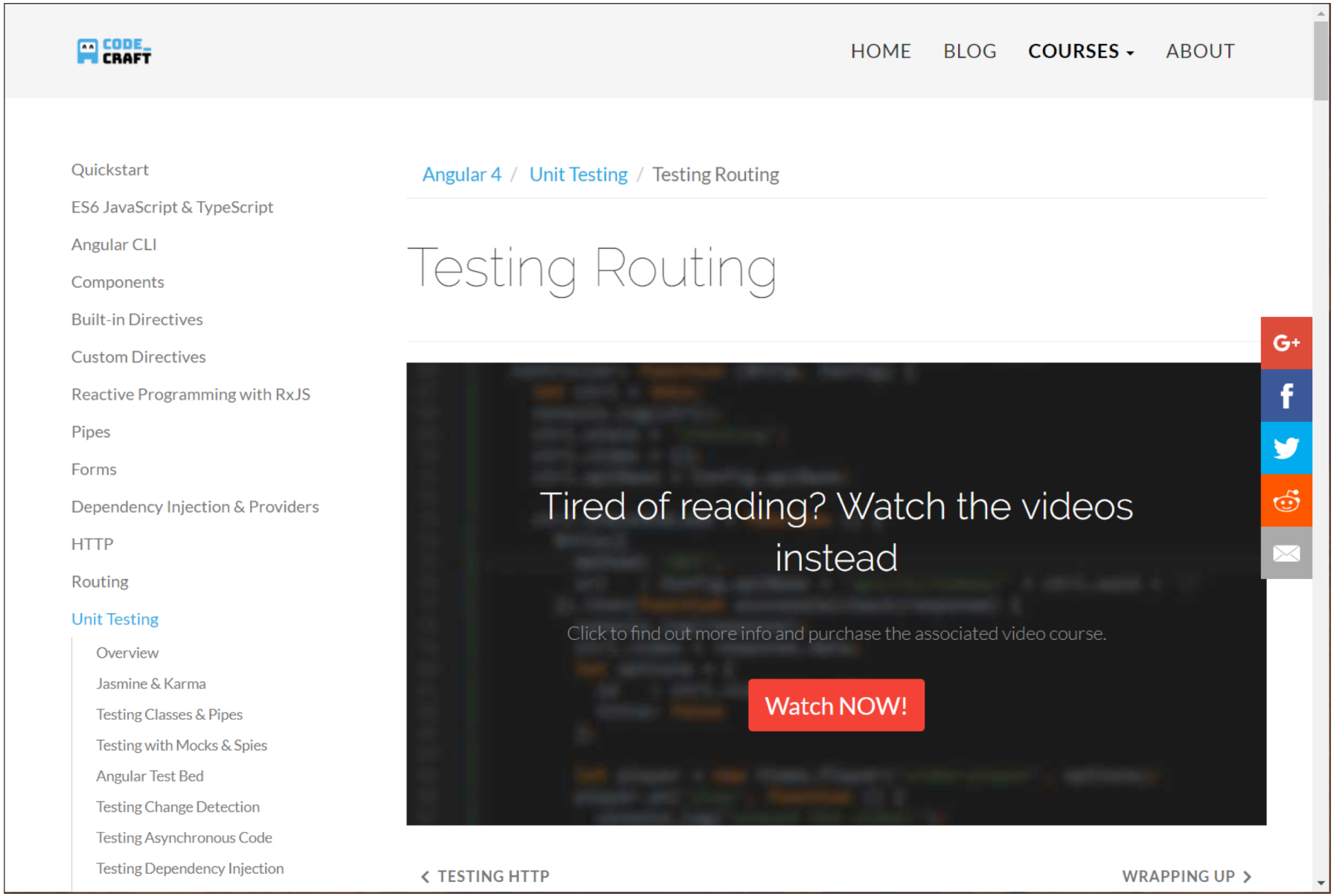
Recipes for Angular Unit Testing using Jasmine



189 18 Next story Angular deprecates ReflectiveInj...

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

Testing Routing



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