

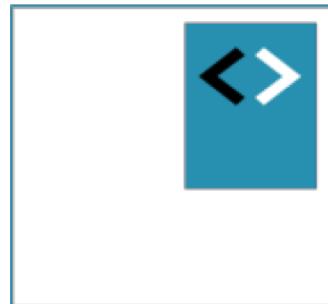


Angular Fundamentals

Module 8 – Testing introduction



A CANON COMPANY



Peter Kassenaar
info@kassenaar.com

Good introductory article

The screenshot shows a DZone article page. At the top, there's a navigation bar with links for REF CARDZ, GUIDES, ZONES, and various technology categories like AGILE, BIG DATA, CLOUD, DATABASE, DEVOPS, INTEGRATION, IOT, JAVA, MOBILE, PERFORMANCE, SECURITY, and WEB DEV. There's also a sign-in button, a search icon, and a 'Let's be friends' section with social media links for RSS, Twitter, Facebook, Google+, and LinkedIn.

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>

Tooling



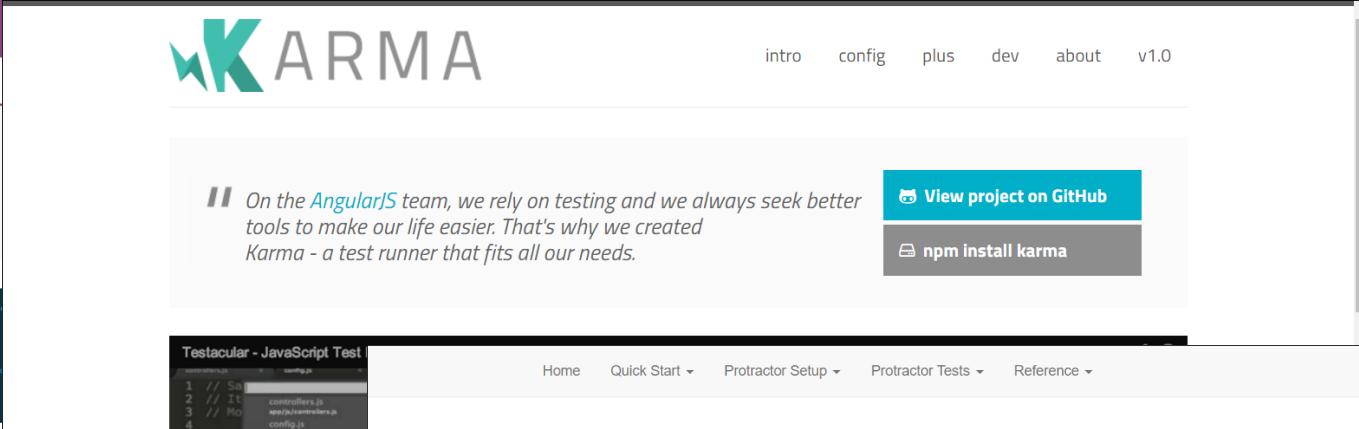
Jasmine:

Write Unit Tests & Specs

Jasmine
Behavior-Driven Javascript

FAST
Low overhead, no external dependencies.

```
describe("A suite is just a function", function() {  
  it("and so is a spec", function() {  
    expect(true).toBe(true);  
  });  
});
```



Karma:

Test Runner

KARMA

On the [AngularJS](#) team, we rely on testing and we always seek better tools to make our life easier. That's why we created Karma - a test runner that fits all our needs.

[View project on GitHub](#)

[npm install karma](#)

```
module.exports = function(config) {  
  config.set({  
    // ...  
    files: [  
      'app/**/*.js',  
      'test/unit/*.js'  
    ],  
    // list of files to exclude  
    exclude: [],  
    // use dots reporter,  
    // possible values: 'dots',  
    reporter: 'progress'  
  });  
};
```



Protractor:

End to end testing

Protractor

end to end testing for AngularJS

[View on GitHub](#)

[Follow @ProtractorTest](#)

Protractor is an end-to-end test framework for Angular and AngularJS applications. Protractor runs tests against your application running in a real browser, interacting with it as a user would.

Test Like a User
Protractor is built on top of WebDriverJS, which uses native events and browser-specific drivers to interact with your application as a user would.

For Angular Apps
Protractor supports Angular-specific locator strategies, which allows you to test Angular-specific elements without any setup effort on your part.

Automatic Waiting
You no longer need to add waits and sleeps to your test. Protractor can automatically execute the next step in your test the moment the webpage finishes pending tasks, so you don't have to worry about waiting for your test and webpage to sync.

Setup

<https://jasmine.github.io/>

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

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

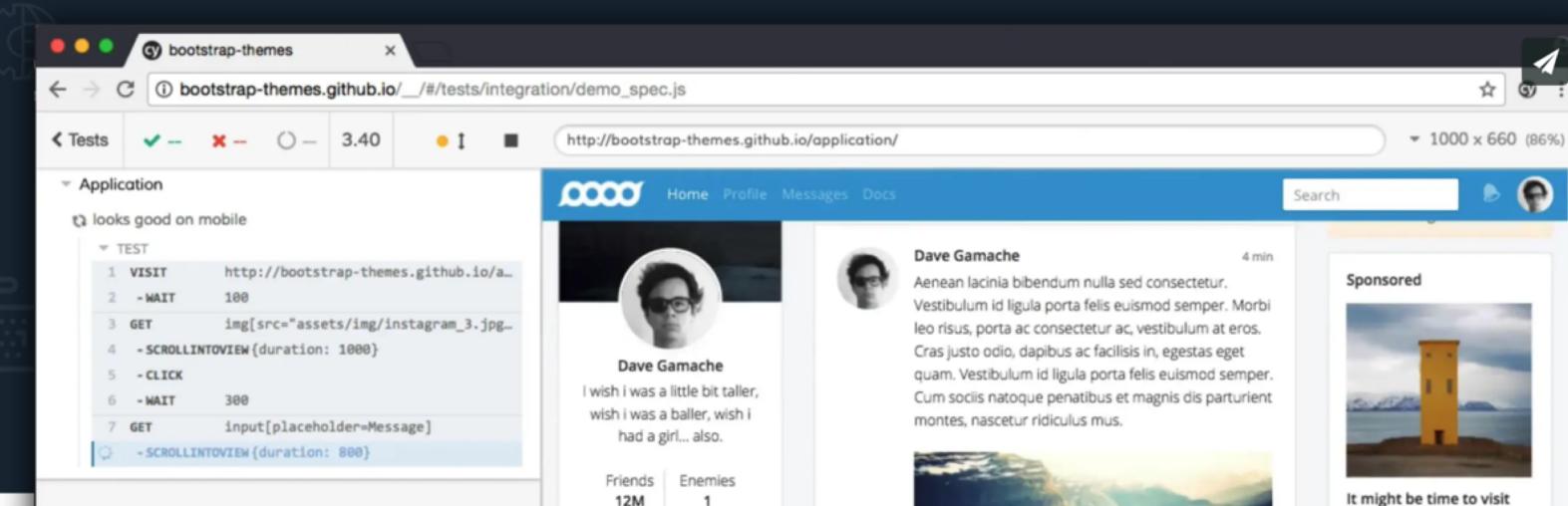
An alternative to E2E-testing - Cypress

The web has evolved.
Finally, testing has too.

Fast, easy and reliable testing for anything that runs in a browser.

\$ npm install cypress or [Download Now](#)

Install Cypress for Mac, Linux, or Windows, then [get started](#).



A screenshot of a web browser window titled "bootstrap-themes". The address bar shows "bootstrap-themes.github.io/_/#/tests/integration/demo_spec.js". The browser interface includes a navigation bar with back, forward, and search buttons, and a status bar indicating "1000 x 660 (86%)". On the left, a sidebar displays a test script for "looks good on mobile" with steps like "VISIT http://bootstrap-themes.github.io/a...". The main content area shows a social media feed with a post by "Dave Gamache" and a sponsored post about a yellow building.

<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!

The screenshot shows a GitHub repository page for 'PeterKassenaar / ng-testing'. The repository is described as a 'Sample Angular-cli project with Jasmine/Karma unit tests'. It has 9 commits, 1 branch, 0 releases, and 1 contributor. The latest commit was made on 31 Aug. The repository contains files like .angular-cli.json, .editorconfig, .gitignore, README.md, karma.conf.js, package.json, protractor.conf.js, and tsconfig.json.

PeterKassenaar / ng-testing

Sample Angular-cli project with Jasmine/Karma unit tests

9 commits · 1 branch · 0 releases · 1 contributor

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

File	Description	Time
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

<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

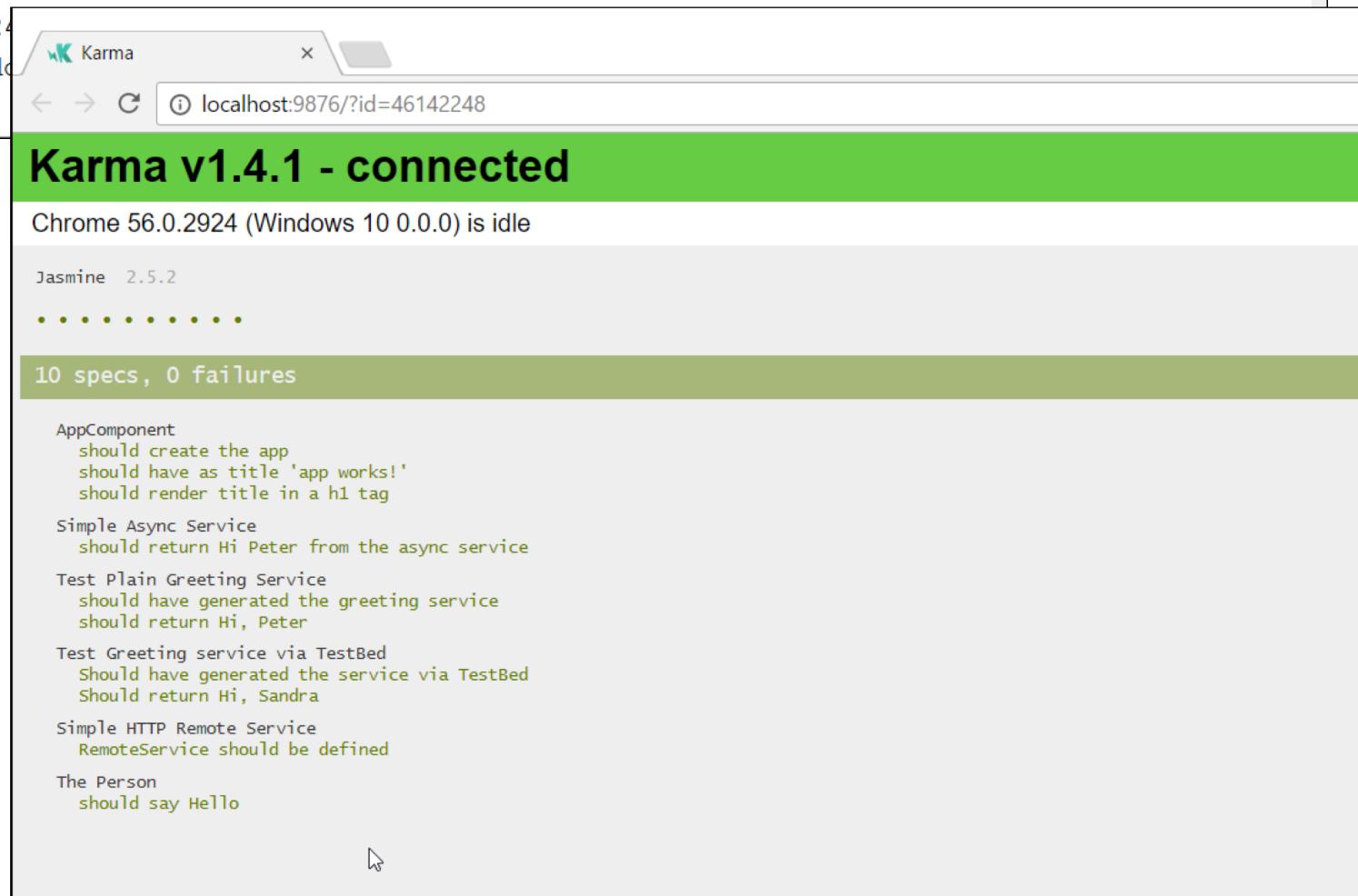
- Angular-cli: all dependencies already installed and configured.
 - Command: `ng test`
- Manual project? Install & setup karma and jasmine yourself
 - Create and adapt `karma.conf.js`
 - `karma --init`
 - Install and setup jasmine reporters
 - **We're not doing that here.**
 - If you really want to,
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 30x3NIH4ohFYk
MAuAAAA with id 4614224
Chrome 56.0.2924 (Windows 10 0.0.0)
```



If anything goes wrong w/ default settings...

Hard to read output:

```
+ :21) [ProxyZone]
x     at ProxyZoneSpec.onInvoke (webpack:///~/zone.js/dist/proxy.js:79:0 <- src/test.ts:67226:3
9) [ProxyZone]
    at Zone.run (webpack:///~/zone.js/dist/zone.js:126:0 <- src/polyfills.ts:3081:43) [<root>
=> ProxyZone]
    at Object.<anonymous> (webpack:///~/zone.js/dist/jasmine-patch.js:104:0 <- src/test.ts:66
Chrome 56.0.2924 (Windows 10 0.0.0): Executed 10 of 10 (1 FAILED) (0.516 secs / 0.478 secs)
```

Karma v1.4.1 - connected

Chrome 56.0.2924 (Windows 10 0.0.0) is idle

Jasmine 2.5.2

• • • • • x

10 specs, 1 failure

Spec List | Failures

The Person should say Hello

Expected 'Hi, Peter' to be 'Hi, Sandra'.

```
Error: Expected 'Hi, Peter' to be 'Hi, Sandra'.
  at stack (http://localhost:9876/base/node_modules/jasmine-core/lib/jasmine-core/jasmine.js?916005cc407925f4764668d61d04888d59258f5d:1640:17) [ProxyZone]
  at buildExpectationResult (http://localhost:9876/base/node_modules/jasmine-core/lib/jasmine-core/jasmine.js?916005cc407925f4764668d61d04888d59258f5d:1610:14) [ProxyZone]
  at Spec.expectationResultFactory (http://localhost:9876/base/node_modules/jasmine-core/lib/jasmine-core/jasmine.js?916005cc407925f4764668d61d04888d59258f5d:655:18) [ProxyZone]
  at Spec.addExpectationResult (http://localhost:9876/base/node_modules/jasmine-core/lib/jasmine-core/jasmine.js?916005cc407925f4764668d61d04888d59258f5d:342:34) [ProxyZone]
  at Expectation.addExpectationResult (http://localhost:9876/base/node_modules/jasmine-core/lib/jasmine-core/jasmine.js?916005cc407925f4764668d61d04888d59258f5d:599:21) [ProxyZone]
  at Expectation.toBe (http://localhost:9876/base/node_modules/jasmine-core/lib/jasmine-core/jasmine.js?916005cc407925f4764668d61d04888d59258f5d:1564:12) [ProxyZone]
  at Object.it (http://localhost:9876/base/src/test.ts?4cc86c17deb77ce26c34c56b304abe7083100bae:51800:21) [ProxyZone]
  at ProxyZoneSpec.onInvoke (http://localhost:9876/base/src/test.ts?4cc86c17deb77ce26c34c56b304abe7083100bae:67226:39) [ProxyZone]
  at Zone.run (http://localhost:9876/base/src/polyfills.ts?3741ae39b5a5909025117982541bc4fac58b679c:3081:43) [<root> => ProxyZone]
  at Object.<anonymous> (http://localhost:9876/base/src/test.ts?4cc86c17deb77ce26c34c56b304abe7083100bae:66926:34) [<root>]
  at attemptSync (http://localhost:9876/base/node_modules/jasmine-core/lib/jasmine-core/jasmine.js?916005cc407925f4764668d61d04888d59258f5d:1950:24) [<root>]
  at ZoneQueueRunner.QueueRunner.run (http://localhost:9876/base/node_modules/jasmine-core/lib/jasmine-core/jasmine.js?916005cc407925f4764668d61d04888d59258f5d:1938:9) [<root>]
```

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

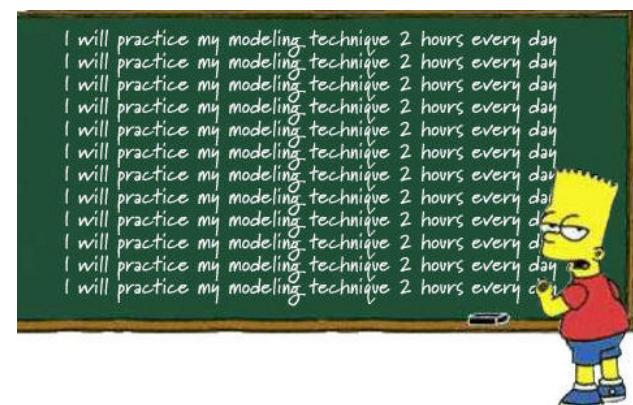
Running the tests

- 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 functionality 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.”*

The screenshot shows the Angular API documentation for the `TestBed` class. The top navigation bar includes links for FEATURS, DOCS, RESOURCES, EVENTS, and BLOG, along with a search bar. The left sidebar has sections for GETTING STARTED, TUTORIAL, FUNDAMENTALS, TECHNIQUES, and API, with `stable (v4.3.4)` selected. The main content area shows `TestBed` as a CLASS. It provides details about the npm package (@angular/core), the module (import { TestBed } from '@angular/core/testing';), and the source file (core/testing/src/test_bed.ts). A description box states: "Configures and initializes environment for unit testing and provides methods for creating components and services in unit tests." Below this is an "Overview" section containing a code snippet for the `TestBed` class:

```
1. class TestBed implements Injector {
2.   static initTestEnvironment(ngModule: Type<any>|Type<any>[], platform: PlatformRef, aotSummaries?: () => any[]): TestBed
3.   static resetTestEnvironment()
4.   static resetTestingModule(): typeof TestBed
5.   static configureCompiler(config: {providers?: any[]; useJit?: boolean;}): typeof TestBed
6.   static configureTestingModule(moduleDef: TestModuleMetadata): typeof TestBed
7.   static compileComponents(): Promise<any>
8.   static overrideModule(ngModule: Type<any>, override: MetadataOverride<NgModule>): typeof TestBed
9.   static overrideComponent(component: Type<any>, override: MetadataOverride<Component>): typeof TestBed
10.  static overrideDirective(directive: Type<any>, override: MetadataOverride<Directive>): 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, XXXX');
    })
});
```



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

The screenshot shows a Google search result for the query "async vs fakeasync". The snippet from a Stack Overflow question discusses the differences between the two methods. A red arrow points from the text "fakeAsync() makes your async code synchronous" to the line of code where fakeAsync is used.

```
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>

The screenshot shows the Angular API documentation for the `async` function. The page has a blue header with the Angular logo and navigation links for FEATURES, DOCS, RESOURCES, EVENTS, and BLOG. A search bar is on the right. The left sidebar has sections for GETTING STARTED, TUTORIAL, FUNDAMENTALS, TECHNIQUES, and API, with `stable (v4.3.4)` highlighted under API. The main content area shows `async` as a FUNCTION. It includes an `npm Package` entry for `@angular/core`, a `Module` entry with the import statement `import { async } from '@angular/core/testing';`, and a `Source` entry pointing to `core/testing/src/async.ts`. Below this is the function signature `function async(fn: Function): (done: any) => any;`. The **Description** section states: "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." An example code block shows a Jest test using the `async` function:

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

<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 `XHRAccend`, 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

The screenshot shows the Angular documentation website with a blue header bar. The header includes the Angular logo, navigation links for FEATURES, DOCS, RESOURCES, EVENTS, and BLOG, and a search bar. On the left, there's a sidebar with sections for GETTING STARTED, TUTORIAL, FUNDAMENTALS, TECHNIQUES, and API, with 'stable (v4.3.4)' highlighted under API. The main content area is titled 'MockBackend' and shows the 'CLASS' tab selected. It contains information about the npm package (@angular/http), the module import statement (import { MockBackend } from '@angular/http/testing';), and the source code location (http/testing/src/mock_backend.ts). Below this is an 'Overview' section with a code snippet for the MockBackend class:

```
class MockBackend implements ConnectionBackend {  
  connections: any  
  connectionsArray: MockConnection[]  
  pendingConnections: any  
  verifyNoPendingRequests()  
  resolveAllConnections()  
  createConnection(req: Request): MockConnection  
}
```

Under the 'Description' section, it says: "A mock backend for testing the `Http` service."

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

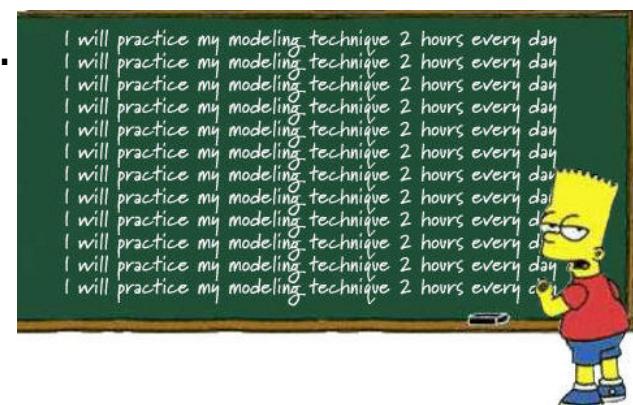
<https://angular.io/api/http/testing/MockConnection>

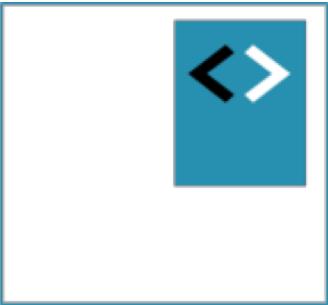
Summary

- 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.configureTestingModuleTestingModule()`
- 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-befor each>

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*, your 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()

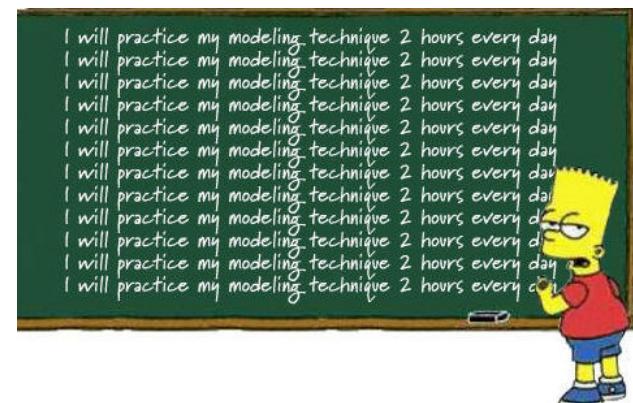
```
spyOn(Object, 'method')
```

```
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 whether the component is correctly created
- Test if this.cars[] is constructed after initialisation. Expect the length of the array to be 2.
- Test whether 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{}

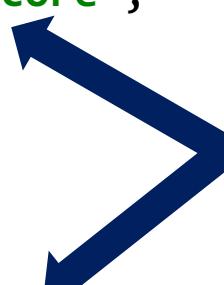
...
```

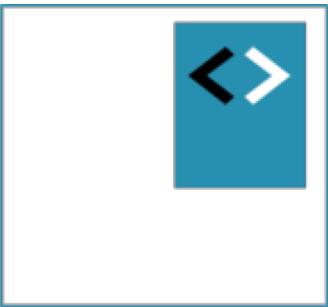
#3 Using NO_ERRORS_SCHEMA

- Provide a schema to the testing module, ignoring all errors and always continue 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



Protractor

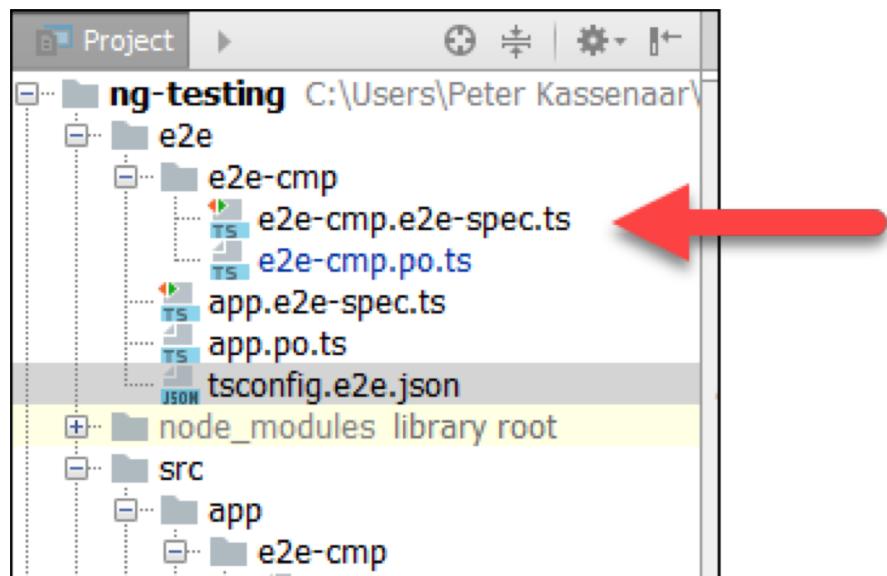
Developed by Team Angular – wrapper around selenium

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

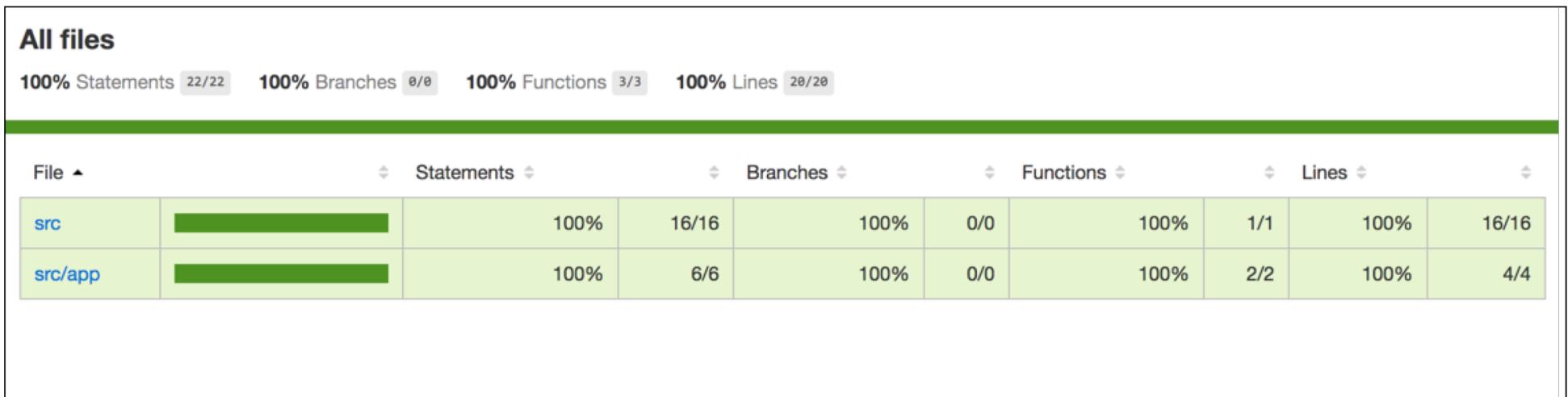
Executing e2e tests

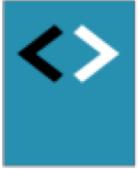
- Generic command: `ng e2e`
- Examples: `\e2e-cmp`



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()
 - .componentInstance
 - .debugElement
 - .nativeElement
 - By (helper class)
 - NO_ERROR_SCHEMA
 - Mocking strategies





More info

Elsewhere on the interwebz...

Testing documentation

The screenshot shows the Angular documentation website for the 'Testing' guide. The top navigation bar includes links for FEATURES, DOCS, RESOURCES, EVENTS, and BLOG, along with a search bar. The main content area has a heading 'Testing' and a sub-section 'Live examples'. A list of test examples is provided, each with a link to download. To the right, a sidebar is highlighted with a red border, containing a tree view of testing topics under the 'Testing' category.

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.

● Testing

- Live examples
- Introduction to Angular Testing
- Tools and technologies
- Setup
- Isolated unit tests vs. the Angular testing utilities
- The first karma test
- Run with karma
- Test debugging
- Try the live example
- Test a component
- TestBed*
- createComponent*
- ComponentFixture, DebugElement, and query(By.css)*
- The tests
- detectChanges: Angular change detection within a test*
- Try the live example
- Automatic change detection
- Test a component with an external template

<https://angular.io/guide/testing>

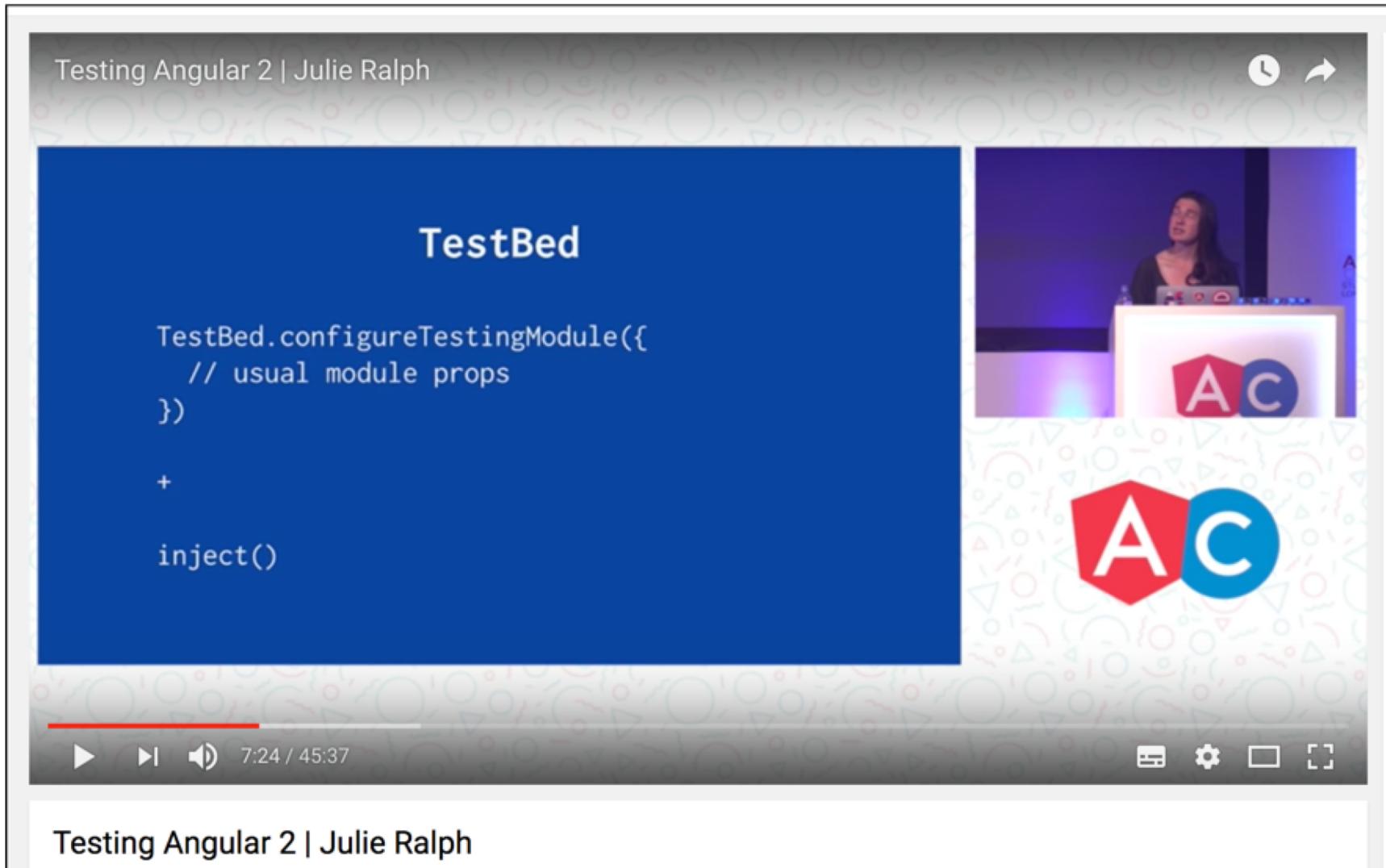
Repo – Angular Testing recipes

The screenshot shows the GitHub repository page for 'juristr/angular-testing-recipes'. The repository has 43 commits, 1 branch, 0 releases, and 1 contributor. The latest commit was made on Jan 13. The repository contains files like e2e, src, .gitignore, .travis.yml, README.md, angular-cli.json, karma.conf.js, package.json, and protractor.conf.js. The commits are all from the master branch and are less than a month old.

File	Commit Message	Time Ago
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

The screenshot shows a DZone article page. At the top, there's a navigation bar with links for REF CARDZ, GUIDES, ZONES, and various technology categories like AGILE, BIG DATA, CLOUD, DATABASE, DEVOPS, INTEGRATION, IOT, JAVA, MOBILE, PERFORMANCE, SECURITY, and WEB DEV. To the right of the navigation is a sign-in button, a user icon, and a search icon. Below the navigation, the main title of the article is displayed: "Testing With Angular 2: Some Recipes (Talk and Slides)". A brief summary follows: "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." Below the summary, the author's profile picture and name ("by Juri Strumpflohner MVB"), the publication date ("Jan. 16, 17 · Web Dev Zone"), and social sharing links for Like (-2), Comment (0), Save, and Tweet are shown. To the right, there are icons for RSS, Twitter, Facebook, Google+, and LinkedIn. Further down, a call-to-action button says "JOIN FOR FREE". A note at the bottom mentions a partnership with Qlik. The text "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." is visible, along with a "Subscribe" button.

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

Gerard Sans on testing

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

The screenshot shows a website layout for 'CODE CRAFT'. At the top right, there is a navigation bar with links for 'HOME', 'BLOG', 'COURSES ▾', and 'ABOUT'. On the far left, a sidebar contains a list of Angular topics: 'Quickstart', 'ES6 JavaScript & TypeScript', 'Angular CLI', 'Components', 'Built-in Directives', 'Custom Directives', 'Reactive Programming with RxJS', 'Pipes', 'Forms', 'Dependency Injection & Providers', 'HTTP', 'Routing', 'Unit Testing' (which is currently selected), 'Overview', 'Jasmine & Karma', 'Testing Classes & Pipes', 'Testing with Mocks & Spies', 'Angular Test Bed', 'Testing Change Detection', 'Testing Asynchronous Code', and 'Testing Dependency Injection'. The main content area has a breadcrumb navigation path: 'Angular 4 / Unit Testing / Testing Routing'. The main title 'Testing Routing' is displayed prominently. Below the title, a dark video thumbnail features a blurred view of a terminal window with code. Overlaid on the thumbnail is the text 'Tired of reading? Watch the videos instead'. At the bottom of the thumbnail, there is a call-to-action button labeled 'Watch NOW!'. Below the thumbnail, a smaller text link says 'Click to find out more info and purchase the associated video course.' At the very bottom of the page, there are navigation links for 'WRAPPING UP ▾' and '< TESTING HTTP'.

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