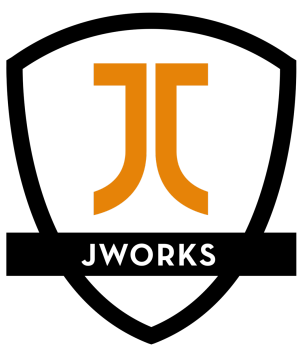
ANGULAR TESTING

WORKSHOP 2017



POWERED BY ORDINA

RYAN DE GRUYTER

Mobile Developer @ KBC
@ryandegruyter
https://github.com/ryandegruyter

OVERVIEW

- Introduction to testing
- Testing tools
- Unit testing
- Isolated testing
- Async tests
- Testing Angular Services
- Angular Injector
- Testing Observables
- TDD vs TAD
- Testing Angular Components
- Using Angular TestBed
- Shallow component testing
- Deep Component Testing
- Additional Testing Strategies
- Writing good tests
- Change Detection Strategies
- DOM Interaction Strategies

INTRODUCTION TO TESTING

Unit testing

Integration testing

E2E Testing

BENEFITS

Documentation for your code

Less bugs

Better design of your API's

Safer refactoring

SAMPLE APPLICATION

https://github.com/ryandegruyter/angular-testing-workshop/

KARMA

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

Test runner (in browser)

watch files and rerun tests

Report test results

Alternative: JEST

WALLABYJS

IDE Test runner

Get instant feedback

Not for free

TESTING LIBRARY

Jasmine Framework

Assertions and mocking tools

BDD syntax (WHEN ... IT SHOULD ...)

DESCRIBE YOUR CODE

- describe('when the user logs in', () => {})
- beforeEach(() => {})
- it('should validate', () => {});

ASSERT USING MATCHERS

expect(actual).toEqual(expected)

- toEqual(...)
- not.toEqual(...)
- toBeTruthy()
- toContain(...)

EXCLUSIONS / FORCING

x (exclude), f (focus)

- fdescribe()
- xdescribe()
- xit()
- fit()

TASK 1: WRITE JASMINE TESTS

Write 2 tests for the String.slice method

- it('should substring up to the end index', () => {...})
- it('should continue to the end of the object when end is not specified', () => {...})

UNIT TESTS

Only test the unit itself without any of its dependencies

USE TEST DOUBLES

Mocks: fake implementations

Stubs: provide canned answers

Spies: record info

JASMINE.SPY

stub and spy

```
const mockUserService = jasmine.createSpyObj('userService', ['getUser']);
mockUserService.getUser.and.returnValue({id: 1, name: 'Ryan'});
getUser(1)
expect(mockUserService.getUser).toHaveBeenCalledWith(1);
```

SPY ON A PARTICULAR METHOD

```
class User {
    getUser(id: number): void {
        ...
    }
}
```

```
const user = new User();
spyOn(user, 'getUser');
user.getUser.and.returnValue(...);
```

TASK 2: CREATE TESTS WITH SPIES

Write a test for the Hero Service

- Create an instance of the service
- Spy on the getHeroes method
- Let the getHeroes spy return a stub list
- Bonus: Instead of creating an instance use a jasmine.createSpyObj

TESTING IN ANGULAR: STRATEGIES

Testing a class: Isolated (unit test)

Testing services: use Angular Injector or TestBed

Testing components: use TestBed

- Shallow
- Deep / Integrated

ISOLATED TESTING

Manually instantiate (or use the Angular Injector ;-)

Mock out all dependencies

Pure unit test

TASK 3: WRITE AN ISOLATED TEST

Manually instantiate the class under test, but mock out any dependencies

write the following test:

```
describe('when getting a list of heroes', () => {
    it('should make a GET request to /heroes', () => {
                    ...
```

ASYNC OPERATIONS IN TESTS

Jasmine done parameter in an **it** clause

```
it('should do an async operation', (done) => {
    let value;
    setTimeout(() => {
        value = 5;
        expect(value).toEqual(5);
        done();
    }, 2000);
    expect(value).toBeUndefined();
});
```

ANGULAR COMES WITH ASYNC TESTING TOOLS

async()

fakeAsync() + tick()

ASYNC()

Test automatically completes when all async calls are done

```
it('should do an async operation', async(() => {
    let value;
    setTimeout(() => {
        value = 5;
        expect(value).toEqual(5);
    }, 2000);
    expect(value).toBeUndefined();
}));
```

FAKEASYNC() + TICK()

```
it('should do an async operation', async(() => {
    let value;
    setTimeout(() => {
        value = 5;
    }, 200);
    tick(200);
    expect(value).toEqual(5);
}));
```

All async calls are captured in a list that can be flushed synchronously.

TASK 4: TEST ASYNC METHODS

Fix the tests

- the first 2 skip the assertion
- the last one fails

BREAK

TESTING ANGULAR SERVICES

Create an injector to configure dependencies

Mock out dependencies

Use mocks that come with Angular (MockBackend, MockConnection)

ANGULAR INJECTOR

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

let injector: ReflectiveInjector;

beforeEach(() => {
    injector = ReflectiveInjector.resolveAndCreate([ ...providers ]);
}
```

TESTING AN HTTP SERVICE

```
beforeEach(() => {
    injector = ReflectiveInjector.resolveAndCreate([
        {provide: ConnectionBackend, useClass: MockBackend},
        {provide: RequestOptions, useClass: BaseRequestOptions},
        Http,
        MyService
    ]);
}
```

- Get the service
- Get the mock backend
- Make a mock response

TASK 5: TEST HTTP SERVICE

Create a Test suite for the Post Service

- Assert a Get request is made
- Assert the correct url is being called (from the environment)

TESTING OBSERVABLES

Subscribe to the observable

Trigger the method under test

Check the emitted values in the subscribe block

MOCKING OBSERVABLES

Use observableOf()

TASK 6: TEST OBSERVABLES

Continue with the Post Service test suite

- subscribe to getPost and assert the emitted value
- do the same for getAll

TEST DRIVEN DEVELOPMENT

TDD VS TAD

TDD: Write tests before implementation code

TAD: Write tests after code implementation

TDD

Red, Green, Refactor

Don't write code unless there is a failing test first

Design your API's (client perspective)

Code that is difficult to test is a code smell

More upfront thinking

Better coverage

No extra code

LUNCH

TESTING ANGULAR COMPONENTS

ANGULAR COMPILER

Unit of compilation are NgModule

Specifies:

- Templates to compile (components, directives, pipes)
- Other NgModules
- Templates to export
- Providers
- Components to be bootstrapped

ANGULAR COMPONENTS TESTING STRATEGIE

Use the Angular TestBed

Shallow or Deep?

TESTBED

Tool for testing components

ComponentFixture

DebugElement

CONFIGURE A TESTING MODULE

TestBed Configures a temporary testing module

```
beforeEach(() => {
    TestBed.configureTestingModule({
        declarations: [ HeroComponent ],
        imports: [ ... ],
        providers: [ ... ]
    });
}
```

CREATING THE COMPONENT

TestBed creates the component in a componentfixture

fixture = TestBed.createComponent(component)

component = fixture.componentInstance

FIXTURE.DETECTCHANGES()

calls the component lifecycle methods.

COMPONENT FIXTURE

- componentInstance - the instance of the component created by TestBed
- debugElement - provides insight into the component and its DOM element
- nativeElement - the native DOM element at the root of the component
- detectChanges() - trigger a change detection cycle for the component
- whenStable() - returns a promise that resolves when the fixture is stable

DEBUGELEMENT

Get information on the components DOM representation

parent / children - the immediate parent or children of this DebugElement

- query(predicate) - search for one descendant that matches
- queryAll(predicate) - search for many descendants that match
- injector - this component's injector
- listeners - this callback handlers for this component's events and @Outputs
- triggerEventHandler(listener) - trigger an event or @Output

SHALLOW COMPONENT TEST

Create a component with TestBed

But mock out or ignore other angular components

DETECTING CHANGES

- fixture.detectChanges()
- fixture.autoDetectChanges()
- fixture.whenStable(() => {})

Ignoring other elements

```
beforeEach(() => {
    TestBed.configureTestingModule({
        ...,
        schemas: [NO_ERRORS_SCHEMA],
        ...
    });
}
```

QUERYING THE DOM

Use DebugElement

- debugElement.query(By.css(selector))
- debugElement.query(By.directive(ComponentClass))

SHALLOW TESTING A COMPONENT

- Setup test module
- Create fixture
- Test component instance
- Test DOM

demo

TASK 7: SHALLOW TEST A COMPONENT

Write the following tests for the comment component

```
describe( description: 'when there is no comment', specDefinitions: () => {
    it( expectation: 'should not render comment', assertion: () => { ... });
});

describe( description: 'when there is a comment', specDefinitions: () => {
    it( expectation: 'should render comment', assertion: () => { ... });
    it( expectation: 'should render name', assertion: () => { ... });
    it( expectation: 'should render email', assertion: () => { ... });
    it( expectation: 'should render body', assertion: () => { ... });
});
```

CHANGE DETECTION STRATEGIES

- async() + detectChanges()
- async() + autoDetectChanges()
- fakeAsync() + tick()

DOM INTERACTION

DebugElement API

```
it(`should change the hero's name (via nativeElement API)`, () => {
    const ngModel = fixture.debugElement.query(By.directive(NgModel));
    ngModel.triggerEventHandler('ngModelChange', 'Mr. Nice');
    fixture.detectChanges();
    expect(getHeadingText(fixture)).toContain('Mr. Nice');
});
```

TASK 8: ADD TEST WITH DOM INTERACTION

Implement these 2 tests

```
xture: ComponentFixture<TestComponent>
describe( description: 'Dom Interaction', specDefinitions: () => {
    let fixture: ComponentFixture<TestComponent>;
    let comp: TestComponent;

    beforeEach( action: () => {
        TestBed.configureTestingModule( moduleDef: {
            imports: [FormsModule],
            declarations: [TestComponent]
        });
        fixture = TestBed.createComponent(TestComponent);
        comp = fixture.componentInstance;
        fixture.detectChanges();
    });
    it( expectation: 'should set default title', assertion: () => { ... });
    it( expectation: 'when model changes it should render a new title', fakeAsync( fn: () => { ... }));
});
```

BREAK

DEEP COMPONENT TEST

Create a component with TestBed

Test with nested components

- Check that the child components are rendered correctly
- Child is receiving the correct inputs
- The parent handles output correctly

CONFIGURE THE TESTBED WITH COMPONENTS AND PROVIDERS

```
TestBed.configureTestingModule({
    imports: [IonicModule]
    declarations: [
        CardListComponent,
        CardComponent
    ],
    providers: [
        { provide: FeedService, useValue: mockFeedService},
        { provide: Router, useValue: mockRouter },
    ]
});
```

CHECK NESTED COMPONENT

Query the child component

```
const cardList = fixture.debugElement.queryAll(By.directive(CardComponent));
```

Check @Inputs

```
expect(cardList[0].componentInstance.title).toEqual(mockList[0].title);
```

Trigger @Output bindings

```
cardList[0].triggerEventHandler('delete', null);
```

TASK 9: ADD DEEP COMPONENT TESTS FOR COMPONENT LIST

TIPS WHEN WRITING TESTS

AAA

Arrange all necessary preconditions and inputs

Act on the object or method under test

Assert that the expected results have occured

DRY

Don't Repeat Yourself

Removes code duplication

DAMP

Descriptive and meaningful phrases

Promotes readability of code

Try to find a good balance between DRY & DAMP

minize logic in tests (what will test the tests?)

TELL THE STORY

A test should be a complete story, all inside the it block

You shouldn't need to look around much to understand the test

- Remove less interesting stuff in the beforeEach
- Keep critical setup within the it

EXAMPLE DAMP VS DRY

THANKS FOR WATCHING!

Now kick some ass!