**Unit testing in Angular**

Inhoud

[Introduction ( 1 - 2) 2](#_Toc46822645)

[AAA 2](#_Toc46822646)

[Testing principals 2](#_Toc46822647)

[Isolated Unit tests 2](#_Toc46822648)

[Mocking a service 2](#_Toc46822649)

[State based- and interaction tests 3](#_Toc46822650)

[Shallow Integration Tests 3](#_Toc46822651)

[Appendix 4](#_Toc46822652)

[Sources 4](#_Toc46822653)

# Introduction ( 1 - 2)

## AAA

Unit test should have the following structure:

* Arrange: all necessary preconditions and inputs
* Act: on the object or class under test
* **A**ssert: that the expected results have occurred.

This is anolog to given, when, then

## Testing principals

DRY (don’t repeat yourself) vs DAMP (repeat yourself if necessary)

Repeat if necessary applies to situations where, for instance the setup is very critical to the story. In that case the setup is not done in the beforeEach but in the it(‘shoul…’, () => {}

## Debugging tests in Karma

There are two features which can help you debugging tests in Karma:

* First one is to open the development console of Chrome and select the console tab. This might give extra info if tests are failing
* Karma and zone.js sometimes do not interact correct. To avoid this issue change the scrips > tests in package.json from: "test": "ng test" to: "test": "ng test --source-map=false"

# Isolated Unit tests

You can look at isolated test as if it’s not an angular class but just a piece of javascript. (in your mind) remove all the things like @Component, @Pipe, @Input etc.

## Mocking a service

Test delete method of the HeroComponent (depending on a service)

See: the Herocomponent.

* The HeroComponent has a heroes property. To test the delete function we have to populate heroes first.
* The HeroComponent is depending on the HeroService which is injected into the contructor.
* To be independent of the HeroService we have to Mock it. Jasmine provides craeateSpyObj()  
  which takes an array of methodnames of the HeroService we want to use.
* As the heroService.deleteHero(…) returns an observable, so should our mockHeroService. We can achieve that by returning an subject when calling deleteHero using the “of” method, like so:  
  mockHeroService.deleteHero.and.returnValue(of(true));
* The ‘should delete …..’ test is setup following the AAA principle.

The code for this test with some highlights:

import { Hero } from "../hero";

import { HeroesComponent } from "./heroes.component";

import { of } from "rxjs";

describe('HeroesComponent', () => {

let heroesComponent: HeroesComponent;

let HEROES;

let mockHeroService;

beforeEach(() => {

HEROES = [

{ id: 1, name: 'pietje', strength: 25 },

{ id: 2, name: 'jantje', strength: 75 },

{ id: 3, name: 'klaasje', strength: 10 }

]

mockHeroService = jasmine.createSpyObj(['addHero', 'getHeroes', 'deleteHero'])

heroesComponent = new HeroesComponent(mockHeroService);

});

it('should delete a hero', () => {

// **Arrange**: init HEROES: is already done by beforeEach

heroesComponent.heroes = HEROES;

// make the mockHeroService.delete retrurn a observable

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

// **Act**: delete a hero

heroesComponent.delete(HEROES[2]);

// **Assert**: HEROES contains two elements now

expect(heroesComponent.heroes.length).toBe(2);

expect(heroesComponent.heroes[0].id).toBe(1);

expect(heroesComponent.heroes[1].id).toBe(2);

})

})

## State based- and interaction tests

The above test checks if the state of the component has changed. It’s a **state based** test. What it doesn’t do is check if certain parts of the code were executed. That kind of tests is called **interaction** tests.

An example of an interaction test which checks if a method was (of the mocked service) was called:

it('shouild call heroService.deleteHero with correct hero', () => {

// Arrange

heroesComponent.heroes = HEROES;

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

// Act

heroesComponent.delete(HEROES[1]);

// Assert

// check that delleteHero was called

expect(mockHeroService.deleteHero).**toHaveBeenCalled()**;

// or even better... check that delleteHero was called with the correct parameter

expect(mockHeroService.deleteHero).**toHaveBeenCalledWith**(HEROES[1]);

})

# Shallow Integration Tests

# Appendix

## Sources

Code source: <https://github.com/joeeames/PSAngularUnitTestingCourse>

Course: <https://app.pluralsight.com/library/courses/unit-testing-angular/table-of-contents>