13 Angular Unit Tests(27)

https://ide.c9.io/laczor/angular2

Session_27_Unit_Testing(Testing) (jasmine folder might be deleted)
369. Analyzing testing setup
270 Running test in the CLI
271 Adding a new Component + fitting tests
Check Service
Check DOM element
Check Async 1 (With instant testing)
Check Async 2 (With waiting all of the async task to be executed)
Check Async 3 (With faking as Async and force stop with tick())
375 Isolated Pipe Testing
376. More informatin about testing

369. Analyzing testing setup

- It is basically, creating a virtual component, then accessing it's elements, properties and check against a condition, if it pass it will give back the result.

app.component.spect,ts

```
/* tslint:disable:no-unused-variable */


// Testing package

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

import { AppComponent } from './app.component';

//We describe what we want ot test

//Which has an anonymus function which will be executed instantly

describe('App: CompleteGuideFinalWebpack', () => {

//We are setting up the settings which should be executed prior each testing


  beforeEach(() => {

    TestBed.configureTestingModule({

      declarations: [

        AppComponent

      ],
    });
  });

// it(name,async function)
// we are creating an app.component for the testing
// debugElement is a property, which enables us to access some elements, like
// componentInstance which is the application itself
//Expect (selector).condition()
  it('should create the app', async(() => {
    let fixture = TestBed.createComponent(AppComponent);
    let app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();
  }));

//Access the title, a component's variable
  it(`should have as title 'app works!'`, async(() => {
    let fixture = TestBed.createComponent(AppComponent);
    let app = fixture.debugElement.componentInstance;
    expect(app.title).toEqual('app works!');
  }));
//Access a DOM element
  it('should render title in a h1 tag', async(() => {
    let fixture = TestBed.createComponent(AppComponent);
    fixture.detectChanges();
    let compiled = fixture.debugElement.nativeElement;
    expect(compiled.querySelector('h1').textContent).toContain('app works!');
  }));
});
```

270 Running test in the CLI

in the terminal run

```
ng test
```

- Will run the testing, and will provide failed description

271 Adding a new Component + fitting tests

app.component.spec.ts

```
/* tslint:disable:no-unused-variable */


import { TestBed, async,

 fakeAsync,

 tick } from '@angular/core/testing';

import { UserComponent } from './user.component';

import { UserService } from "./user.service";

import { DataService } from "../shared/data.service";


//Create a basic enviroment, passing the component

describe('Component: User', () => {

  beforeEach(() => {

    TestBed.configureTestingModule({

      declarations: [UserComponent]

    });

  });

//Component should be created

  it('should create the app', () => {

    let fixture = TestBed.createComponent(UserComponent);

    let app = fixture.debugElement.componentInstance;

    expect(app).toBeTruthy();

  });


//1.Create a component for the test

//2.To get an instance of  "UserService"the service with the injector.get()

//3.Wait for changes
```

```
//4.Check the service
  it('should use the user name from the service', () => {
    let fixture = TestBed.createComponent(UserComponent);
    let app = fixture.debugElement.componentInstance;
    let userService = fixture.debugElement.injector.get(UserService);
    fixture.detectChanges();
    expect(userService.user.name).toEqual(app.user.name);
  });

//1.Create a component for the test
//2.Modify the component's state
//3.Detect changes
//4.Get an instance created component's DOM
//5.Checks the Dom element's context with the component's variable
  it('should display the user name if user is logged in', () => {
    let fixture = TestBed.createComponent(UserComponent);
    let app = fixture.debugElement.componentInstance;
    app.isLoggedIn = true;
    fixture.detectChanges();
    let compiled = fixture.debugElement.nativeElement;
    expect(compiled.querySelector('p').textContent).toContain(app.user.name);
  });

  it('shouldn\'t display the user name if user is not logged in', () => {
    let fixture = TestBed.createComponent(UserComponent);
    let app = fixture.debugElement.componentInstance;
    fixture.detectChanges();
    let compiled = fixture.debugElement.nativeElement;
    expect(compiled.querySelector('p').textContent).not.toContain(app.user.name);
  });

//1.Create a component for the test
```

```
//2. Get an instance of the DataService

//3. Use the spyOn('service','function') and return a value

//3.1 the function got executed but what is returned is the our defined promise of (Promise.res
olve('Data'))

//4.Listen for changes

//5.It is returning the value immediately, not waiting for the async function to finish


  it('shouldn\'t fetch data successfully if not called asynchronously', () => {

    let fixture = TestBed.createComponent(UserComponent);

    let app = fixture.debugElement.componentInstance;

    let dataService = fixture.debugElement.injector.get(DataService);

    let spy = spyOn(dataService, 'getDetails')

      .and.returnValue(Promise.resolve('Data'));

    fixture.detectChanges();

    expect(app.data).toBe(undefined);

  });



//5.Waiting for the async functions to be completed

//6.TO react when all of the async functions has been finished -->

//fixture.whenStable()

  it('should fetch data successfully if called asynchronously', async(() => {

    let fixture = TestBed.createComponent(UserComponent);

    let app = fixture.debugElement.componentInstance;

    let dataService = fixture.debugElement.injector.get(DataService);

    let spy = spyOn(dataService, 'getDetails')

      .and.returnValue(Promise.resolve('Data'));

    fixture.detectChanges();

    fixture.whenStable().then(() => {

      expect(app.data).toBe('Data');

    });

  }));
```

```
//5. It will fake the async functions as if they were really initated

//6. tick()  --> In a fake async enviroment we force to finish all of the async functions

//7.(Note that we are returning immediately a promise! not waiting for the function)

  it('should fetch data successfully if called asynchronously', fakeAsync(() => {

    let fixture = TestBed.createComponent(UserComponent);

    let app = fixture.debugElement.componentInstance;

    let dataService = fixture.debugElement.injector.get(DataService);

    let spy = spyOn(dataService, 'getDetails')

      .and.returnValue(Promise.resolve('Data'));

    fixture.detectChanges();

    tick();

    expect(app.data).toBe('Data');


  }));

});
```

Check Service

```
//1.Create a component for the test

//2.To get an instance of  "UserService"the service with the injector.get()

//3.Wait for changes

//4.Check the service

  it('should use the user name from the service', () => {

    let fixture = TestBed.createComponent(UserComponent);

    let app = fixture.debugElement.componentInstance;

    let userService = fixture.debugElement.injector.get(UserService);

    fixture.detectChanges();

    expect(userService.user.name).toEqual(app.user.name);

  });
```

Check DOM element

```
  //1.Create a component for the test

//2.Modify the component's state

//3.Detect changes

//4.Get an instance created component's DOM

//5.Checks the Dom element's context with the component's variable

  it('should display the user name if user is logged in', () => {

    let fixture = TestBed.createComponent(UserComponent);

    let app = fixture.debugElement.componentInstance;

    app.isLoggedIn = true;

    fixture.detectChanges();

    let compiled = fixture.debugElement.nativeElement;

    expect(compiled.querySelector('p').textContent).toContain(app.user.name);

  });
```

Check Async 1 (With instant testing)

- We return immediatly a value

```
  //1.Create a component for the test

//2. Get an instance of the DataService

//3. Use the spyOn('service','function') and return a value

//3.1 the function got executed but what is returned is the our defined promise of (Promise.res
olve('Data'))

//4.Listen for changes

//5.It is returning the value immediately, not waiting for the async function to finish


  it('shouldn\'t fetch data successfully if not called asynchronously', () => {

    let fixture = TestBed.createComponent(UserComponent);

    let app = fixture.debugElement.componentInstance;

    let dataService = fixture.debugElement.injector.get(DataService);
    let spy = spyOn(dataService, 'getDetails')

      .and.returnValue(Promise.resolve('Data'));

    fixture.detectChanges();

    expect(app.data).toBe(undefined);

  });
```

Check Async 2 (With waiting all of the async task to be executed)

```
//5.Waiting for the async functions to be completed

//6.TO react when all of the async functions has been finished -->

//fixture.whenStable()

  it('should fetch data successfully if called asynchronously', async(() => {

    let fixture = TestBed.createComponent(UserComponent);

    let app = fixture.debugElement.componentInstance;

    let dataService = fixture.debugElement.injector.get(DataService);

    let spy = spyOn(dataService, 'getDetails')

      .and.returnValue(Promise.resolve('Data'));

    fixture.detectChanges();

    fixture.whenStable().then(() => {

      expect(app.data).toBe('Data');

    });

  }));
```

Check Async 3 (With faking as Async and force stop with tick())

```
  /5. It will fake the async functions as if they were really initated

//6. tick()  --> In a fake async enviroment we force to finish all of the async functions

//7.(Note that we are returning immediately a promise! not waiting for the function)

  it('should fetch data successfully if called asynchronously', fakeAsync(() => {
    let fixture = TestBed.createComponent(UserComponent);

    let app = fixture.debugElement.componentInstance;

    let dataService = fixture.debugElement.injector.get(DataService);

    let spy = spyOn(dataService, 'getDetails')

      .and.returnValue(Promise.resolve('Data'));

    fixture.detectChanges();

    tick();

    expect(app.data).toBe('Data');

  }));

});
```

375 Isolated Pipe Testing

1. Creating a pipe

```
import { Pipe } from "@angular/core";


@Pipe({

  name: 'reverse'

})

export class ReversePipe {

  transform(value: string) {

    return value.split("").reverse().join("");

  }

}
```

2. Write the seperate testing, since basically this is a seperate function, no need to involve angular
- Import the only dependency **ReversePipe**
- Testing : **describe('name',function)**

```
/* tslint:disable:no-unused-variable */

//0. Import the pipe

//1. create anonymus functions

//2. Create an instance of a new Reversepipe
```

```
//3. Pass a variable and checks the output

import { ReversePipe } from "./reverse.pipe";

describe('Pipe: ReversePipe', () => {

  it('should reverse the inputs', () => {

    let reversePipe = new ReversePipe();

    expect(reversePipe.transform('hello')).toEqual('olleh');

  });


});
```

376. More informatin about testing

This Module only provides a brief and basic Introduction to Angular 2 Unit Tests and the Angular 2 Testing Suite. This Course isn't focused on Testing.

If you want to dive deeper, the official Docs actually are a great place to start. There you'll also find a Non-CLI Setup!

Official Docs: https://angular.io/docs/ts/latest/guide/testing.html

I can also recommend the following Article: https://semaphoreci.com/community/tutorials/testing-components-in-angular-2-with-jasmine

For more Information on how to run Tests with the CLI have a look at their official Docs:

=> Unit Tests: https://github.com/angular/angular-cli#running-unit-tests

=> E2E Tests: https://github.com/angular/angular-cli#running-end-to-end-tests