Angular Unit Testing

What is unit testing ?

- It is also called component testing .
- Testing small, isolated pieces of code .
- Check only if particular class is present or not (Improves design).
- Do not use external resources, such as the network or a database .
- Unit Tests isolate a section of code and verify its correctness .
- A unit may be an individual function, method , module, or object.
- Unit tests help to fix bugs early in the development cycle and save costs.
- Good unit tests serve as project documentation

When we need to do unit testing ?

- Coding / Development Phase .

Who do Unit testing ?

- Developers .
- Sometimes QA engineers .

What shall we do in angular unit testing ?

- Write test case .
- Reactive form test , Field validation , pattern matching .
- Are specific css classes present in a .html file or not ?
- Checks a button with definite value is present or not and a method is called or not .
- Checks the array , objects , methods .
- Test the service .
- Test the routing which present the .html or .ts files .
- A form can have multiple labels and test the labels are matching with specific value or not .
- Some hard coded or static data checking . (Ex. error in base / image url can not be changed)
- End of the day , we can find the documentation of each component which helps a new developer to understand about the component and how it interacts with the project .

Karma : It is a JavaScript test runner that runs the unit test snippet in Angular. Karma also ensures the result of the test is printed out either in the console or in the file log. We define the browser in which we find the test result . Karma provides tools that make it easier to call Jasmine tests while writing code in Angular.It uses a configuration file in order to set the startup file, the reporters, the testing framework, the browser among other things.

Jasmine : Jasmine is a JavaScript testing framework that supports a software development practice called Behaviour-Driven Development, or BDD . It's a specific flavour of Test-Driven Development (TDD).Jasmine, and BDD in general, attempts to describe tests in a human readable format so that non-technical people can understand what is being tested. However even if you are technical reading tests in BDD format makes it a lot easier to understand what's going on.It has a bunch of functionalities to allow us the write different kinds of tests.

Configuration

package.json

```
"test": "ng test",
"karma-chrome-launcher": "~3.1.0",
```

karma.config.js

```
require('karma-chrome-launcher'),
browsers: ['Chrome'],
```

To run the firefox browser : `npm install karma-firefox-launcher --save-dev`

```
require('karma-firefox-launcher'),
browsers: [Firefox],
```

test.ts

```
const context = require.context('./', true, /\.spec\.ts$/);
```

Key terms

`describe :` The describe(string, function) function defines what we call a Test Suite, contains different blocks (it, beforeEach, xit, etc.).

beforeAll : This function is called once, before all the specs in a test suite (describe function) are run.

afterAll : This function is called once after all the specs in a test suite are finished.

beforeEach : This function is called before each test specification (it function) is run.

afterEach : This function is called *after* **each** test specification is run.

it : The it(string, function) function defines an individual Test Spec.

expect() : The expect(actual) expression is what we call an Expectation.

Some Jasmine keywords

Jasmine matcher function :

Link - 1 : http://jasmine.github.io/edge/introduction.html#section-Included_Matchers

LInk - 2 : https://jasmine.github.io/api/2.6/matchers.html

toBe === string , number
toEqual == array , object (used for deep matching)
toMatch : digits in string , or a substring in a text
toContain also find the substring in a text

xdescribe or **xit** excludes the test

Some angular testing module

```
import { HttpClientTestingModule } from '@angular/common/http/testing';
import { RouterTestingModule } from '@angular/router/testing';
import { HttpTestingController } from '@angular/common/http/testing';
import { TestBed, async, fakeAsync, tick, ComponentFixture } from '@angular/core/testing';

import { By } from '@angular/platform-browser';
import { ReactiveFormsModule, FormsModule } from '@angular/forms';
import { Location, CommonModule } from '@angular/common';
import { Router } from '@angular/router';
```

`TestBed :` `TestBed` is the primary api for writing unit tests for Angular applications and libraries. It creates dummy module to test a specific component .It creates an Angular testing module — a @NgModule class — that you configure with the configureTestingModule method to produce the module environment for the class you want to test. `TestBed.configureTestingModule` in your test suite's `beforeEach` block and give it an object with similar values as a regular **NgModule** for declarations, providers and imports. You can then chain a call to `compileComponents` to tell Angular to compile the declared components.

`compileComponents :` The `compileComponents` object is called to compile your component's resources like the template, styles etc.

The **`fixture.debugElement.componentInstance`** creates an instance of the class (`Component Name`).

`fixture.debugElement.nativeElement :` access to its child element (html tags , css class)

detectChanges : The detectChanges triggers a change detection cycle for the component.

```
describe('AddUserComponent', () => {

      beforeEach(async () => {
        await TestBed.configureTestingModule({
         })
        .compileComponents();
      });

      beforeEach(() => {
        });

      it('should create', () => {
        });

});
```

Structure of spec.ts file

```
import { HttpClientModule } from '@angular/common/http';
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { AddUserComponent } from './add-user.component';

describe('AddUserComponent', () => {
 let component: AddUserComponent;
 let fixture: ComponentFixture<AddUserComponent>;

 beforeEach(async () => {
   await TestBed.configureTestingModule({
     declarations: [ AddUserComponent ],
     imports: [
       HttpClientModule
     ]
   })
   .compileComponents();
 });

 beforeEach(() => {
   fixture = TestBed.createComponent(AddUserComponent);
   component = fixture.componentInstance;
   fixture.detectChanges();
 });

 it('should create', () => {
   expect(component).toBeTruthy();
 });
});
```

```
 it('should be a heading tag ', () => {
   let h2 = fixture.debugElement.nativeElement.querySelector('h2');
   expect(h2.textContent).toBe('Register Today!');
 });
```

Path : LocalLotteriesComponent

```
<p class="theme">abcd</p>
```

```
it('should have a 50/50 Draws heading ', () => {
   const headings = fixture.debugElement.query(By.css(p)); // query(By.css(.theme));
   const native: HTMLElement = headings.nativeElement;
   expect(native.textContent).toBe('abcd')
 })
```

```
<h2>Local Lotteries, 50/50 Draws, Prize Draws & Bingo!</h2>
<h2>Local Lotteries</h2>
<h2>50/50 Draws</h2>
```

```
it('should have a 50/50 Draws heading ', () => {
   const headings = fixture.debugElement.queryAll(By.css('h2'));
   const native: HTMLElement = headings[2].nativeElement;
   expect(native.textContent).toBe('50/50 Draws')
 })
```

simple component test

Check only if particular class is present or not ?

```
JoinSupporterComponent ( its a registration form )

it(' registration form class existence  ', () => {
   let formdesign =  fixture.debugElement.query(By.css('.request-form'));
   expect(formdesign).toBeTruthy()
 });

it('check the theme-button class ', () => {
let submitButton = fixture.debugElement.query(By.css('.theme-button'));
let native : HTMLElement = submitButton.nativeElement;
expect(native.textContent).toBe('SUBMIT');
});
```

Image Error Handling

```
PopularDrawsComponent :

errorImage : string = "https://smartlottoassets.s3-eu-west-1.amazonaws.com/logo-2.png" ;

errorHandler(event: any) {
  event.target.src = this.errorImage;
}

it('should have an image not found errorHandler() method' , () => {
component.errorHandler;
expect(component.errorImage).toEqual('https://smartlottoassets.s3-eu-west-1.amazonaws.com/l
ogo-2.png')
expect(component.errorHandler).toBeTruthy();
const urlRegex
=/^[A-Za-z][A-Za-z\d.+-]*:\/*(?:\w+(?::\w+)?@)?[^\s/]+(?::\d+)?(?:\/[\w#!:.?+=&%@\-/]*)?$/;
expect(component.errorImage).toMatch(urlRegex);
 })
```

HTTP client , routing and service test

```
imports: [
       HttpClientTestingModule,
       RouterTestingModule,
       ReactiveFormsModule,
       FormsModule,
     ],
     providers:
       [
         UserServiceService,
         { provide: UserServiceService, useClass: UserServiceServiceStub }
       ],
```

Reactive Form labels

```
Are all the labels sequentially arranged ?
it(' check every label ', () => {
   var levels =  fixture.debugElement.queryAll(By.css('label'));
   var native : HTMLElement;

   native  = levels[0].nativeElement;
   expect(native.textContent).toBe('Contact Name');

   native = levels[1].nativeElement;
   expect(native.textContent).toBe('Phone Number');

   native = levels[2].nativeElement;
   expect(native.textContent).toBe('Email');
  });
```

Inline Css testing , Mock service , and *ngIf

Project : client >> StudentComponent

Is the *ngIf working ?

```
<div class="success-message" *ngIf="submitted">hello</div>
```

```
it('check the successful message ' , () => {
expect(fixture.debugElement.query(By.css('.success-message'))).toBeNull();
   component.submitted = true;
   fixture.detectChanges();
   let message = fixture.debugElement.query(By.css('.success-message'));
   let native : HTMLElement =  message.nativeElement;
   expect(native.textContent).toBe('hello');
 });
```

test the loop data ?

```
getUsers(): Stu[] {
  return [
    {  name: 'user1',  surname: 'usersurname1',  },
    {  name: 'user2',  surname: 'usersurname2'   },
    {  name: 'user3',  surname: 'usersurname3'   } ]; }
```

```
<div *ngFor='let item of users; let i = index'>
 <p class="user-list"> {{i}} {{item.name}} - {{item.surname}}</p>
</div>
```

```
it(`using class inside loop`, async(() => {
  fixture.detectChanges();
  let users = fixture.debugElement.queryAll(By.css('.user-list'));
  let native: HTMLElement = users[1].nativeElement
  expect(native.innerText).toContain('user2');
}));

it(`should have Stu[1].name is user2` , async(() => {
  expect(component.users[1].name).toBe('user2');
}));
```

Reactive form pattern testing

2 .
from a email field validation
Path : JoinSupporterComponent

```
email: new FormControl('', [ Validators.required , Validators.pattern("[^ @]*@[^ @]*")]),
```

spec.ts

```
   email.setValue("test");
   errors = email.errors || {};
   expect(errors['pattern']).toBeTruthy();

   email.setValue("test@gmail.com");
   errors = email.errors || {};
   expect(errors['pattern']).toBeFalsy();
```

Button click testing

1 . Path : `TrialRequestBlockComponent`

```
<button type="submit" class="theme-button">Start your 30 day Free Trial </button>
```

click the button, and check that the method was called ?

```
it('should call the method saveRegInfo() while clicking button', async(() => {
  spyOn(component, 'saveRegInfo');
  let button = fixture.debugElement.nativeElement.querySelector('.theme-button');
  button.click();
  fixture.whenStable().then(() => {
    expect(component.saveRegInfo).toHaveBeenCalled();
  });
}));
```

async test as the button click contains asynchronous event handling, and need to wait for the event to process by calling fixture.whenStable()

```
it('should call the method with fakeAsync', fakeAsync(() => {
  spyOn(component, 'saveRegInfo');
  let button = fixture.debugElement.nativeElement.querySelector('.theme-button');
  button.click();
  tick();
  expect(component.saveRegInfo).toHaveBeenCalled();
}));
```

The fakeAsync is the Angular testing API that wraps a test function in a fake asynchronous test zone.

The tick() simulates the asynchronous passage of time. Wait all asynchronous code to be finished and then continue .

Reactive Form testing

1 . client : ReactiveFormTestComponent

```
<button class="theme-button-2" [disabled]="loginForm.invalid" type="submit">Log In</button>
```

```
 private createDetailForm() {
   this.loginForm = this.fb.group(
     {
       email: new FormControl('', [Validators.required]),
       password: new FormControl('', Validators.required)
     }
   );
 }
```

Spec.ts

```
it('submitting a form ', () => {
   expect(component.loginForm.invalid).toBeTruthy();
   let btn = fixture.debugElement.query(By.css('.theme-button-2'))
   let native: HTMLElement = btn.nativeElement;
   expect(native.innerHTML).toBe('Log In');

   expect(btn.nativeElement.disabled).toBeTruthy();

   component.loginForm.controls['email'].setValue('sadad');
   component.loginForm.controls['password'].setValue('01521466521');
   fixture.detectChanges();
   expect(btn.nativeElement.disabled).toBeFalsy();
   component.onSubmit();
   fixture.detectChanges();
   expect(component.loginForm.valid).toBeTruthy();
 });
```

2 .
JoinSupporterComponent

```
it('contact_name field validity', () => {
   let name = component.supporterForm.controls['contact_name'];
   expect(name.valid).toBeFalsy();
   expect(name.pristine).toBeTruthy();
   let errors: any = {};
   errors = name.errors || {};
   expect(errors['required']).toBeTruthy();
   name.setValue('sds');
   expect(name.errors).toBeNull();
   expect(name.valid).toBeTruthy();
 });

it('submitting a form ', () => {
 expect(component.supporterForm.invalid).toBeTruthy();
 let btn = fixture.debugElement.query(By.css('.theme-button'))
 let native :HTMLElement = btn.nativeElement;
 expect(native.innerHTML).toBe('SUBMIT');
 expect(btn.nativeElement.disabled).toBeTruthy();
 component.supporterForm.controls.contact_name.setValue('sadad');
 fixture.detectChanges();
 expect(component.supporterForm.valid).toBeTruthy();
 expect(btn.nativeElement.disabled).toBeFalsy();
 component.saveSupporterInfo();
 fixture.detectChanges();
  });
```

Routing : part - 1

1.

NavbarComponent

```
<a class="route-1" [routerLink]="['/']">
<a class="active" [routerLink]="['/features/home']" >Home</a>
<a class="route-3" [routerLink]="['/features/membership']">Membership</a>
```

```
@Component({
 template: ''
})
class DummyComponent {
}

RouterTestingModule.withRoutes([
          { path: '', component: DummyComponent },
          { path: 'features/home', component: DummyComponent },
          { path: 'features/membership', component: DummyComponent },
        ])
```

```
it('should go to url  " / " ' ,
async(inject([Router, Location], (router: Router, location: Location) => {
      fixture.detectChanges();
      fixture.debugElement.query(By.css('.route-1')).nativeElement.click();
      fixture.whenStable().then(() => {
      expect(location.path()).toEqual('/');
}); }) ));

it('should go to url  " /features/home " ' ,
async(inject([Router, Location], (router: Router, location: Location) => {
      fixture.detectChanges();
      fixture.debugElement.query(By.css('.active')).nativeElement.click();
      fixture.whenStable().then(() => {
      expect(location.path()).toEqual('/features/home');
}); }) ));

it('should go to url  " /features/membership " ',
      async(inject([Router, Location], (router: Router, location: Location) => {
      fixture.detectChanges();
      fixture.debugElement.query(By.css('.route-3')).nativeElement.click();
      fixture.whenStable().then(() => {
      expect(location.path()).toEqual('/features/membership');
 }); }) ));
```

Routing : part - 2

2. Component : work / client / LoginComponent

app-routing.module.ts

```
{   path: 'display',   component: ViewComponent },
```

```
let location: Location;
let router: Router;
RouterTestingModule.withRoutes(routes)

//inside before each
router = TestBed.get(Router);
location = TestBed.get(Location);
router.initialNavigation();

 it('navigate to "/display" redirects you to /display', fakeAsync(() => {
   router.navigate(['display']);
   tick();
   expect(location.path()).toBe('/display');
 }));
```

Routing : part - 3

3 . navigateByUrl routing procedures Client : LoginComponent

```
<button class="list-group-item" type="button" (click)="go(1)">
Navigate</button>

 go() {
      this.router.navigateByUrl('display');
 }

 it('should test component with Activated Route' , fakeAsync(() => {
   fixture.detectChanges();
   let liElement = fixture.debugElement.query(By.css('.list-group-item'));
   liElement.nativeElement.click();
   tick();
   expect(location.path()).toContain('/display');
   fixture.detectChanges();
 }));
```

```
 hello() {
   this.router.navigateByUrl('vai');
 }


 it('should call hello method Router.navigateByUrl("vai/") ', inject([Router],
(router: Router) => {
   const spy = spyOn(router, 'navigateByUrl');
   component.hello();
   const url = spy.calls.first().args[0];
   expect(url).toBe('vai');
 }));
```

4 . navigateByUrl using id

```
 goto(supporter_id) {
   this.router.navigateByUrl('display/'+supporter_id);
 }

it('should call Router.navigateByUrl("display/:id") ' , inject([Router], (router:
Router) => {
   const spy = spyOn(router, 'navigateByUrl');
   component.goto(23);
   const url = spy.calls.first().args[0];
   expect(url).toBe('display/23');
 }));
```

Services testing

1 . services
Path : client / `ViewComponent`

```
it('does test promise',
inject([UserServiceService], async (myService: UserServiceService) => {
const result = await myService.getAll();
     expect(result).not.toBeNull();
 }))

it('show a services how it works', fakeAsync(() => {
let serve = fixture.debugElement.injector.get(UserServiceService);
let stub = spyOn(serve, 'getAll').and.callFake(() => { return of([]);  })
component.viewData();
tick();
expect(component.userValue).toEqual([]);
 }))
```

Testing the Get , Post , Delete method with dummy data

Client / `UserServiceService`

```
service.getAll().subscribe((res) => {
     expect(res).toEqual(dummyPosts);
   });
```

This Verify the observable when it resolves, its result matches test data.

```
const req = httpMock.expectOne('http://localhost:3006/user');
```

Verify the matched URL get called in the GET API else it throws errors.

```
expect(req.request.method).toEqual("GET");
```

Verify that the request called is GET HTTP method only.

```
req.flush(dummyPosts);
```

Ensures the correct data was returned using Subscribe callback.

```
httpMock.verify();
```

Ensures that all request are fulfilled and there are no outstanding requests.

coverage report

ng test --no-watch --code-coverage

Code-coverage reports every time you test,

angular.json

```
"test": {
            "options"  :  {
              "codeCoverage" :  true
            }
        }
```

It can check inline css properties but can not check external properties .

```
<div class="sky" style="background-color: rgb(180, 168, 192);">
</div>

it('should create', () => {
    let heads = fixture.debugElement.query(By.css('.sky'));
    let native: HTMLElement = heads.nativeElement;
    expect(native.style.backgroundColor).toBe('rgb(180, 168,
192)');
});
```

SupportingMembershipComponent

Did this hard coded object have all data which matched the required pattern ?

```
it('slide Configuration ', () => {
   let slideConfigDummy = {
     "slidesToShow": 3,
     "slidesToScroll": 1,
     "autoplay": true,
     "centerMode": true,
     "vertical": false,
     "dots": false,
     "infinite": true,
     autoplaySpeed: 2000,
     draggable: false,
     swipe: false,
     touchMove: false,
     focusOnSelect: false,
     accessibility: false,
     responsive:
    [
      {  breakpoint: 1199, settings: { slidesToShow: 3 } },
      {  breakpoint: 991,  settings: { slidesToShow: 2, "centerMode": false,     "dots": true, } },
      {  breakpoint: 767, settings: { slidesToShow: 1 } }
    ]
   };
   expect(component.slideConfig).toEqual(slideConfigDummy);
   expect(component.slideConfig.responsive[2].breakpoint).toBe(767);
 });
```