Angular testing

what we test in unit testing

.....

.

- Unit testing is the process of testing small, isolated pieces of code
- unit tests do not use external resources, such as the network or a database.
- Check only if particular class is present or not. (In most of the cases which we worked on)
- Jasmine : doesn't rely on other JavaScript frameworks, default test framework,
- Karma is a test runner for JavaScript

package.json

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

[
"test-headless": "ng test --watch=false --browsers=ChromeHeadless",
To run test : npm run test-headless
]

```
npm install karma-firefox-launcher --save-dev
```

karma.config.js

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

Before we start

TestBed : creates a dummy module to test a component .

SpyOn : it is a easy way to check a method is called or not .

SpYOn calls the original function using (.and.)

whenStable : it helps us to test promises by allowing us to wait until all the promises have completed .

tick() : simulates the passages of time until all the asynchronous task are executed .

(fakeAsync and tick function) ,(async and whenStable)

Example :

```
describe('component name', () => {
      beforeEach( () =>{ } )
                  it('should create', () => {  });
      afterEach ( () => { } )
});
```

```
describe('ViewComponent', () => {
 let component: ViewComponent;
 let fixture: ComponentFixture<ViewComponent>;
 let service: UserServiceService;
 let httpMock: HttpTestingController;

 beforeEach(async () => {
   await TestBed.configureTestingModule({
     declarations: [ViewComponent, DummyComponent],
     imports: [
       HttpClientTestingModule,
       RouterTestingModule.withRoutes(
```

```
      [
        { path: 'add', component: DummyComponent },
        { path: 'settings/ashif/edit/:item', component: DummyComponent }
      ]
       ),
       CommonModule,
       HttpClientModule
     ],
     providers:
       [
         UserServiceService,
         { provide: UserServiceService, useValue: userServicesMoc },
         { provide: UserServiceService, useClass: UserServiceServiceStub }
       ],
   })
     .compileComponents();
});

      beforeEach(() => {
        fixture = TestBed.createComponent(ViewComponent);
        component = fixture.componentInstance;
        dh = new DOMHelper(fixture);
        fixture.detectChanges();
        service = TestBed.inject(UserServiceService);
        service = TestBed.get(UserServiceService);
      });

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

});
```

LInk : 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
beForeEach() will execute for every test case and it setting up the initialization or the value or services or component
afterEach() nullify the value after execution, removing the initialization or value.

xdescribe or **xit** excludes the test

.....

simple component test

1 .
Branch : testing-5
FaqComponent

```
it('Download Pdf Version', () => {
 let button = fixture.debugElement.nativeElement.querySelector('.dd-btn');
   expect(button.textContent).toBe('Download Pdf Version');
 });


it('About Our Terms ', () => {
 const buttons = fixture.debugElement.queryAll(By.css('button'));
 const native: HTMLElement = buttons[0].nativeElement;
 expect(native.textContent).toBe(' About Our Terms ')
 })
```

2 .
HomeBannerComponent

```
it('should have a button PLAY NOW', async(() => {
let p = fixture.debugElement.nativeElement.querySelector('.theme-button-2');
expect(p.textContent).toBe('PLAY NOW');
 }));
```

3 .
LocalLotteriesComponent

```
it('should have a See All button ', () => {
   const buttons = fixture.debugElement.query(By.css('.theme-button'));
   const native: HTMLElement = buttons.nativeElement;
   expect(native.textContent).toBe('See All')
})
```

4 .
PopularDrawsComponent

```
imageBaseUrl: string;
gamesBaseUrl: string;

 it('should have an image url', () => {
   expect(component.imageBaseUrl).toBeTruthy();
 })

 it('should have a game base url', () => {
   expect(component.gamesBaseUrl).toBeTruthy();
 })

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/logo-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);
 })
```

5 .
HTTP client , routing and service problems

```
import { HttpClientTestingModule } from '@angular/common/http/testing';
import { RouterTestingModule } from '@angular/router/testing';
import { ReactiveFormsModule, FormsModule } from '@angular/forms';


imports: [
       HttpClientTestingModule,
       RouterTestingModule,
       ReactiveFormsModule,
       FormsModule,
       RouterTestingModule.withRoutes(
         [
{ path: 'add', component: DummyComponent },
{ path: 'settings/ashif/edit/:item', component: DummyComponent }
         ]
       ),

     ],
     providers:
       [
         UserServiceService,
         { provide: UserServiceService, useValue: userServicesMoc },
         { provide: UserServiceService, useClass: UserServiceServiceStub }
       ],
```

6 .
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');
});
```

```
it(' check every label ', () => {

   var lavels =  fixture.debugElement.queryAll(By.css('label'));
   var native : HTMLElement;

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

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

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

7 . branch testing-5
SupportingMembershipComponent

```
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);
 });
```

8 . client >> StudentComponent (loop testing with mock service and css)

```
<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');
 });
```

.....

button click

1 .

Path : client / `ReactiveFormTestComponent`

```
// using async
 it('should click Send button with async', async(() => {
   let buttonElement = fixture.debugElement.query(By.css('.send-button'));

   spyOn(component, 'sendData');
   //Trigger click event after spyOn
   buttonElement.triggerEventHandler('click', null);

   fixture.whenStable().then(() => {
     expect(component.sendData).toHaveBeenCalled();
   });
 }));

 // using fake async
 it('should click Send button with fakeAsync', fakeAsync(() => {
   let buttonElement = fixture.debugElement.query(By.css('.send-button'));

   spyOn(component, 'sendData');
   //Trigger click event after spyOn
   buttonElement.triggerEventHandler('click', null);

   tick();
   expect(component.sendData).toHaveBeenCalled();
 }));
```

.....

Pattern checking

1 . branch : testing-5

path : GameListItemsComponent

```
<span id="lottery_value">0.45</span>
```

spec.ts

```
 it(' >>>> lottery value  ', () => {
    const value = fixture.debugElement.query(By.css('#lottery_value'));
    const native: HTMLElement = value.nativeElement;
    console.log(native.textContent);
    var regex =/^\d+\.\d{0,2}$/;
    expect(native.textContent).toMatch(regex);
 })
```

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();
```

3 .

```
<span class="boy" style="width: 50%; font-size: 10px;"></span>

it('should have inline font style', () => {
const h1Element: HTMLElement = ixture.nativeElement.querySelector('.boy');
expect(h1Element.style.fontSize).toEqual('10px');
expect(h1Element.style.width).toEqual('50%');
});

<span
id="ashif-pattern">https://www.itsolutionstuff.com/post/angular-validation
-for-url-exampleexample.html
</span>

it('pattern checking  ', () => {
fixture.detectChanges();
const value = fixture.debugElement.query(By.css('#ashif-pattern'));
const native: HTMLElement = value.nativeElement;
const urlRegex
=/^[A-Za-z][A-Za-z\d.+-]*:\/*(?:\w+(?::\w+)?@)?[^\s/]+(?::\d+)?(?:\/[\w#!:
.?+=&%@\-/]*)?$/;
expect(native.innerHTML).toMatch(urlRegex);
})
```

.....

services testing

1 . services
Path : client / ViewComponent
Path : 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([]);

 }))
```

2 .
Path : client / RouterLinkTestComponent
Topic : **How To Unit Test Angular Component With Service.**
https://codehandbook.org/how-to-unit-test-angular-component-with-service/

3 .

Client / `UserServiceService`

```
beforeEach(()=>{
 service = TestBed.get(UserServiceService);
 httpMock = TestBed.get(HttpTestingController);
 })

afterEach(()=>{
   httpMock.verify();
 })
 it('getData() should http GET names', () => {

     const dummyPosts: User[] = [{
     userName: 'ashif',
     contact: 1521466521,
     password: 'password',
     city: 'uuu uuu',
     country: 'yyy yyyy',
     code: 'yyyy'
   },
   {
     userName: 'mokbul',
     contact: 1845041010,
     password: 'qqqq',
     city: 'barisal',
     country: 'bd',
     code: '4600'
   }];

   service.getAll().subscribe((res) => {
     console.log("res",res);
     expect(res).toEqual(dummyPosts);
   });

   const req = httpMock.expectOne('http://localhost:3006/user');
   expect(req.request.method).toEqual("GET");
   req.flush(dummyPosts);

 });
```

4 .
Path : client / UserServiceService
Post data unit test

```
 it('should add an user and return it HTTP POST', () => {
   const dummyPosts: User[] =[    {
     userName: 'aminul islam',
     contact: 125485698,
     password: 'asasas',
     city: 'kurigram',
     country: 'bd',
     code: '5600'
   }]

   service.create(dummyPosts).subscribe(
     data =>
     expect(data).toEqual(dummyPosts, 'should return the user')
   );
    const req = httpMock.expectOne('http://localhost:3006/user');
   expect(req.request.method).toEqual('POST');
   expect(req.request.body).toEqual(dummyPosts);

   const expectedResponse = new HttpResponse({ status: 201, statusText:
'Created', body: dummyPosts });
   req.event(expectedResponse);
 });
```

.....

Reactive Form

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

Spec.ts

```
it('form invalid when empty', () => {
   expect(component.supporterForm.valid).toBeFalsy();
 });

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

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');
 }); }) ));
```

2. Another routing procedure

Component : work / client / LoginComponent

app-routing.module.ts

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

```
import { routes } from '../app-routing.module'

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');
 }));
```

3 . navigateByUrl routing procedures

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

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

 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();
 }));
```

4 . navigateByUrl using id

Client : `LoginComponent`

https://semaphoreci.com/community/tutorials/testing-routes-in-angular-2

```
 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');
 }));

 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;
   expect(url).toEqual(['display/23']);
 }));
```

Finally coverage report .

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

Code-coverage reports every time you test,

Angular.json

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