1)

Unit Testing: -

Unit Testing is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development of an application by the developers.

Functional Testing: -

Functional Testing is a type of software testing that validates the software system against the functional requirements/specifications. The purpose of Functional tests is to test each function of the software application, by providing appropriate input, verifying the output against the Functional requirements.

2)

**1. Angular automated testing**

describe('TestSuitName', () => {

// suite of tests here

it('should do some stuff', () => {

// this is the body of the test

});

});

**2. Components**

describe('HeaderComponent', () => {

let component: HeaderComponent;

let element: HTMLElement;

let fixture: ComponentFixture<HeaderComponent>;

// \* We use beforeEach so our tests are run in isolation

beforeEach(() => {

TestBed.configureTestingModule({

// \* here we configure our testing module with all the declarations,

// \* imports, and providers necessary to this component

imports: [CommonModule],

providers: [],

declarations: [HeaderComponent],

}).compileComponents();

fixture = TestBed.createComponent(HeaderComponent);

component = fixture.componentInstance; // The component instantiation

element = fixture.nativeElement; // The HTML reference

});

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

expect(component).toBeTruthy();

});

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

// \* arrange

const title = 'Hey there, i hope you are enjoying this article';

const titleElement = element.querySelector('.header-title');

// \* act

component.title = title;

fixture.detectChanges();

// \* assert

expect(titleElement.textContent).toContain(title);

});

});

**3. Pipes**

describe('TroncaturePipe', () => {

it('create an instance', () => {

const pipe = new TroncaturePipe(); // \* pipe instantiation

expect(pipe).toBeTruthy();

});

it('truncate a string if its too long (>20)', () => {

// \* arrange

const pipe = new TroncaturePipe();

// \* act

const ret = pipe.transform('123456789123456789456666123');

// \* asser

expect(ret.length).toBe(20);

});

});

**4. Directives**

// \* Host component:

@Component({

template: `<div [appPadding]="2">Test</div>`,

})

class HostComponent {}

@NgModule({

declarations: [HostComponent, PaddingDirective],

exports: [HostComponent],

})

class HostModule {}

// \* Test suite:

describe('PaddingDirective', () => {

let component: HostComponent;

let element: HTMLElement;

let fixture: ComponentFixture<HostComponent>;

beforeEach(() => {

TestBed.configureTestingModule({

imports: [CommonModule, HostModule], // \* we import the host module

}).compileComponents();

fixture = TestBed.createComponent(HostComponent);

component = fixture.componentInstance;

element = fixture.nativeElement;

fixture.detectChanges(); // \* so the directive gets appilied

});

it('should create a host instance', () => {

expect(component).toBeTruthy();

});

it('should add padding', () => {

// \* arrange

const el = element.querySelector('div');

// \* assert

expect(el.style.padding).toBe('2rem'); // \* we check if the directive worked correctly

});});

**5. Services**

describe('LocalService', () => {

let service: LocalService;

beforeEach(() => {

TestBed.configureTestingModule({

providers: [LocalService],

});

service = TestBed.get(LocalService); // \* inject service instance

});

it('should be created', () => {

expect(service).toBeTruthy();

});

it('should set the local', () => {

// \* act

service.setLocal('fr');

// \* assert

expect(service.getLocal()).toBe('fr');

});

});

**6. Dealing with HTTP requests**

class FakeApiService {

// Implement the methods you want to overload here

getData() {

return of({ items: [] }); // \* mocks the return of the real method

}

}

//...

TestBed.configureTestingModule({

imports: [],

declarations: [myComponent],

providers: [

{

provide: RealApiService,

useClass: FakeApiService,

}, ], });

**7. Dealing with the Angular router using spies**

it('should do something', () => {

// arrange

const service = TestBed.get(dataService);

const spyOnMethod = spyOn(service, 'saveData').and.callThrough();

// act

component.onSave();

// assert

expect(spyOnMethod).toHaveBeenCalled();

});

**8. Dealing with asynchronous code**

it('should do something async', async () => {

// \* arrange

const ob = { id: 1 };

component.selected = ob;

// \* act

const selected = await component.getSelectedAsync(); // get the promise value

// \* assert

expect(selected.id).toBe(ob.id);

});

**9. Dealing with observables**

it('should do something async', (done) => {

// \* arrange

const ob = { id: 1 };

component.selected = ob;

// \* act

const selected$ = component.getSelectedObs(); // get an Observable

// \* assert

selected$.subscribe(selected => {

expect(selected.id).toBe(ob.id);

done(); // let Jasmine know that you are done testing

});

});

**10. Dealing with timeouts**

const TIMEOUT\_DELAY = 250;

//...

it('should do something async', (done) => {

// \* arrange

const ob = { id: 1 };

// \* act

component.setSelectedAfterATimeout(ob);

// \* assert

setTimeout(() => {

expect(component.selected.id).toBe(ob.id);

done(); // let Jasmine know that you are done testing

}, TIMEOUT\_DELAY);

});