**EXERCISE ANGULAR UNIT TEST**

1. Unit testing is the process of tests small, isolated pieces of code that do not use external resources, such as the network or a database. An Angular unit test is to uncover issues such as incorrect logic, misbehaving functions, etc. by isolating pieces of code. Angular unit testing enables you to test your app based on user behavior. Here we have a service, a component, and an async task to simulate data being fetched from the server.
2. This test script, which ends with .spec.ts, is always added. Let’s take a look at the initial test script file, which is the app.component.spec.ts
3. Karma 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. By default, Angular runs on Karma. Other test runners include Mocha and Jasmine. Karma provides tools that make it easier to call Jasmine tests while writing code in Angular.
4. The Angular testing package includes two utilities called TestBed and async. TestBed is the main Angular utility package.
5. Services often depend on other services that Angular injects into the constructor. In many cases, it is easy to create and inject these dependencies by adding providedIn: root to the injectable object, which makes it accessible by any component or service:

import { Injectable } from "@angular/core";

import { QuoteModel } from "../model/QuoteModel";

@Injectable({

providedIn: "root"

})

export class QuoteService {

public quoteList: QuoteModel[] = [];

private daysOfTheWeeks = ["Sun", "Mon", "Tue", "Wed", "Thurs", "Fri", "Sat"];

constructor() {}

addNewQuote(quote: String) {

const date = new Date();

const dayOfTheWeek = this.daysOfTheWeeks[date.getDate()];

const day = date.getDay();

const year = date.getFullYear();

this.quoteList.push(

new QuoteModel(quote, `${dayOfTheWeek} ${day}, ${year}`)

);

}

getQuote() {

return this.quoteList;

}

removeQuote(index:number) {

this.quoteList.splice(index, 1);

}

}

//Here are a few ways to test the QuoteService class:

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

import { QuoteService } from "./Quote.service";

describe("QuoteService", () => {

let service: QuoteService;

beforeEach(() => {

service = new QuoteService();

});

it("should create a post in an array", () => {

const qouteText = "This is my first post";

service.addNewQuote(qouteText);

expect(service.quoteList.length).toBeGreaterThanOrEqual(1);

});

it("should remove a created post from the array of posts", () => {

service.addNewQuote("This is my first post");

service.removeQuote(0);

expect(service.quoteList.length).toBeLessThan(1);

});

});

1. In the first block, beforeEach, an instance of QuoteService is created to ensure it is only created once and to avoid repetition in other blocks except for some exceptional cases:

it("should create a post in an array", () => {

const qouteText = "This is my first post";

service.addNewQuote(qouteText);

expect(service.quoteList.length).toBeGreaterThanOrEqual(1);

});

1. The first block tests if the post model QuoteModel(text, date) is created into an array by checking the length of the array. The length of the quoteList is expected to be 1:

it("should remove a created post from the array of posts", () => {

service.addNewQuote("This is my first post");

service.removeQuote(0);

expect(service.quoteList.length).toBeLessThan(1);

});

1. The second block creates a post in an array and removes it immediately by calling removeQuote in the service object. The length of the quoteList is expected to be 0.
2. The service is injected into the QuoteComponent to access its properties, which will be needed by the view:

import { Component, OnInit } from '@angular/core';

import { QuoteService } from '../service/Quote.service';

import { QuoteModel } from '../model/QuoteModel';

@Component({

selector: 'app-Quotes',

templateUrl: './Quotes.component.html',

styleUrls: ['./Quotes.component.css']

})

export class QuotesComponent implements OnInit {

public quoteList: QuoteModel[];

public quoteText: String ="";

constructor(private service: QuoteService) { }

ngOnInit() {

this.quoteList = this.service.getQuote();

}

createNewQuote() {

this.service.addNewQuote(this.quoteText);

this.quoteText ="";

}

removeQuote(index) {

this.service.removeQuote(index);

}

}

<div class="container-fluid">

<div class="row">

<div class="col-8 col-sm-8 mb-3 offset-2">

<div class="card">

<div class="card-header">

<h5>What Quote is on your mind ?</h5>

</div>

<div class="card-body">

<div role="form">

<div class="form-group col-8 offset-2">

<textarea #quote class="form-control" rows="3" cols="8" [(ngModel)]="quoteText" name="quoteText"></textarea>

</div>

<div class="form-group text-center">

<button class="btn btn-primary" (click)="createNewQuote()" [disabled]="quoteText == null">Create a new

quote</button>

</div>

</div>

</div>

</div>

</div>

</div>

<div class="row">

\*ngFor="let quote of quoteList; let i = index"

(click)="removeQuote(i)">

<div class="card-body">

<h6>{{ quote.text }}</h6>

</div>

<div class="card-footer text-muted">

<small>Created on {{ quote.timeCreated }}</small>

</div>

</div>

</div>

</div>

1. The first two blocks in the describe container run consecutively. In the first block, the FormsModule is imported into the configure test. This ensures the form’s related directives, such as ngModel, can be used.
2. The QuotesComponent is declared in the configTestMod similar to how the components are declared in ngModule residing in the appModule file. The second block creates a QuoteComponent and its instance, which will be used by the other blocks:

let component: QuotesComponent;

let fixture: ComponentFixture<QuotesComponent>;

beforeEach(() => {

TestBed.configureTestingModule({

imports: [FormsModule],

declarations: [QuotesComponent]

});

});

beforeEach(() => {

fixture = TestBed.createComponent(QuotesComponent);

component = fixture.debugElement.componentInstance;

});

1. This block tests if the instance of the component that is created is defined:

it("should create Quote component", () => {

expect(component).toBeTruthy();

});

1. The injected service handles the manipulation of all operations (add, remove, fetch). The quoteService variable holds the injected service (QuoteService). At this point, the component is yet to be rendered until the detectChanges method is called:

it("should use the quoteList from the service", () => {

const quoteService = fixture.debugElement.injector.get(QuoteService);

fixture.detectChanges();

expect(quoteService.getQuote()).toEqual(component.quoteList);

});

1. Now let’s test whether we can successfully create a post. The properties of the component can be accessed upon instantiation, so the rendered component detects the new changes when a value is passed into the quoteText model. The nativeElement object gives access to the HTML element rendered, which makes it easier to check if the quote added is part of the texts rendered:

it("should create a new post", () => {

component.quoteText = "I love this test";

fixture.detectChanges();

const compiled = fixture.debugElement.nativeElement;

expect(compiled.innerHTML).toContain("I love this test");

});

1. Apart from having access to the HTML contents, you can also get an element by its CSS property. When the quoteText model is empty or null, the button is expected to be disabled:

it("should disable the button when textArea is empty", () => {

fixture.detectChanges();

const button = fixture.debugElement.query(By.css("button"));

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

});

it("should enable button when textArea is not empty", () => {

component.quoteText = "I love this test";

fixture.detectChanges();

const button = fixture.debugElement.query(By.css("button"));

expect(button.nativeElement.disabled).toBeFalsy();

});

1. Just like the way we access an element with its CSS property, we can also access an element by its class name. Multiple classes can be accessed at the same time using By e.g By.css(‘.className.className’).
2. The button clicks are simulated by calling the triggerEventHandler . The event type must be specified which, in this case, is click. A quote displayed is expected to be deleted from the quoteList when clicked on:

it("should remove post upon card click", () => {

component.quoteText = "This is a fresh post";

fixture.detectChanges();

fixture.debugElement

.query(By.css(".row"))

.query(By.css(".card"))

.triggerEventHandler("click", null);

const compiled = fixture.debugElement.nativeElement;

expect(compiled.innerHTML).toContain("This is a fresh post");

});

1. Test an async operation in Angular: it’s inevitable that we will want to fetch data remotely. This operation is best treated as an asynchronous task.
2. fetchQoutesFromServer represents an async task that returns an array of quotes after two seconds:

fetchQuotesFromServer(): Promise<QuoteModel[]> {

return new Promise((resolve, reject) => {

setTimeout(() => {

resolve([new QuoteModel("I love unit testing", "Mon 4, 2018")]);

}, 2000);

});

}

1. spyOn objects simulate how fetchQuotesFromServer method works. It accepts two arguments: quoteService, which is injected into the component, and the fetchQuotesFromServer method.
2. fetchQuotesFromServer is expected to return a promise. spyOn chains the method using and with a fake promise call, which is returned using returnValue. Because we want to emulate how the fetchQuotesFromServer works, we need to pass a promise that will resolve with a list of quotes.
3. Just as we did before, we’ll call the detectChanges method to get the updated changes. whenStable allows access to results of all async tasks when they are done:

it("should fetch data asynchronously", async () => {

const fakedFetchedList = [

new QuoteModel("I love unit testing", "Mon 4, 2018")

];

const quoteService = fixture.debugElement.injector.get(QuoteService);

let spy = spyOn(quoteService, "fetchQuotesFromServer").and.returnValue(

Promise.resolve(fakedFetchedList)

);

fixture.detectChanges();

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

expect(component.fetchedList).toBe(fakedFetchedList);

});

});