Now that you know about developing an angular application, in this section, let's learn about testing your applications using Jasmine

Angular application code can be unit tested using Jasmine framework.

While working on Angular applications using Angular CLI, you have seen that whenever you create a component or service or directive or a pipe, along with  \*.ts files, \*.spec.ts files also gets generated.

These files also called spec files consists of the unit tests written for the source file corresponding to which they are generated.

This section will help you explore how Jasmine testing framework is used by Angular for testing the application code.

In this section you will learn how to unit test an Angular:

1. Component
2. Pipe

**Note**: Each unit test is put in separate file and this file uses '**spec**' in the name to mark it as a spec file.

Example: If component is **app.component.ts** then its test spec would be **app.component.spec.ts**.

Now let's learn how the test an Angular component.

For testing angular components and its corresponding template angular provides a TestBed. **TestBed**helps in creating the required environment in order component being tested. Any component, pipe , service and route to be tested should be first configured in TestBed using **configureTestingModule**function.

After configuring all the components, you will compile them using **compileComponents**function.

Now that you have all the components compiled, in order to test a components first you need to create an instance of the component. For creating an instance you can use **createComponent**function, which will return you a **fixture**object using which you can create the instance of the component.

**fixture**is an object which provides access to components instance and the corresponding **DOM** associated with it.

In order to verify the component changes to HTML, **detectChanges**function fixture object is used.

Once you have the component instance, its attributes and methods can be tested by the combination of expect() and matchers from Jasmine framework.

Now let's learn how to test the component and its DOM with the help of a demo.

Highlights:

* Angular Component Testing
* Creating TestBed

**Demo Steps:**

1. Create a new Angular project named **AngularTest**with routing option using the below command. We will learn how to test routes later in this course.

1. ng new AngularTest --routing

2. Create a component called **message** using the below command.

1. ng generate component message

3. Copy the below code into **message.component.ts**file

1. import { Component, OnInit } from '@angular/core';
2. @Component({
3. selector: 'app-message',
4. templateUrl: './message.component.html',
5. styleUrls: ['./message.component.css']
6. })
7. export class MessageComponent implements OnInit {
8. title: string;
9. msg: string;
10. val: boolean;
11. constructor() {
12. this.title = "Thought for the day"
13. this.msg = "Success is achived through accurate efforts"
14. this.val = false;
15. }
16. ngOnInit() {
17. }
18. show() {
19. this.val = true;
20. return this.val;
21. }
22. hide(){
23. this.val = false;
24. return this.val;
25. }}

4. Copy the below code into **message.component.html**

1. <div>
2. <h3>{{title}}</h3>
3. </div>
4. <br/>
5. <div>
6. <button type="button" (click)="show()">Show</button>
7. <button type="button" (click)="hide()">Hide</button>
8. </div>
9. <br/>
10. <br/>
11. <div \*ngIf="val" id="msgText">
12. {{msg}}
13. </div>

5. Modify the code inside **AppComponent** class of **app.component.ts** with the code given below

1. export class AppComponent {
2. title = 'Angular Testing';
3. }

6. Modify the code in **app.component.html** with the code given below

1. <div style="text-align:center">
2. <h1>
3. Welcome to {{ title }}!
4. </h1>
5. <br/>
6. <br/>
7. </div>
8. <app-message></app-message>

7. Add the following code in **message.component.spec.ts** file

1. it('title should say Thought for the day',() => {
2. expect(component.title).toEqual('Thought for the day');
3. })
4. it('Should display message', ()=> {
5. expect(component.show()).toBeTruthy();
6. });
7. it('Should hide message', ()=> {
8. expect(component.hide()).toBeFalsy();
9. });
10. it('Message',() =>{
11. let a1= fixture.nativeElement.querySelectorAll('div')
12. let value = a1[1].innerHTML.trim;
13. expect(value).toEqual('Success is achived through accurate efforts'.trim)
14. })

**Note**: we are using trim to ignore the white spaces before and after the text.

8. Since we are using message component in the app component, the app component spec file needs to be updated. Add the MessageComponent in the declarations of the app.component.spec.ts as shown below:

1. describe('AppComponent', () => {
2. beforeEach(async(() => {
3. TestBed.configureTestingModule({
4. declarations: [
5. AppComponent,
6. MessageComponent
7. ],
8. }).compileComponents();
9. }));

9. Run the test cases by using the command **ng test**, and observe that all the four test cases created are being passed.

**Note: ng test** will run **all** the spec files. Therefore, remove the unnecessary test cases from app.component.spec.ts file. Else the test cases of app.component.spec.ts file will fail.

Highlights:

* Angular Pipe Testing
* Creating name pipe

Demosteps:

1. Create a new Pipe named **AlphaCase** with the below code using the below command.

1. ng generate pipe message/alphaCase

2. Add the below code to **alpha-case.pipe.ts** file

1. import { Pipe, PipeTransform } from '@angular/core';
2. @Pipe({
3. name: 'alphaCase'
4. })
5. export class AlphaCasePipe implements PipeTransform {
6. transform(value: any, args?: any): any {
7. if (args == 'U') {
8. return value.toUpperCase();
9. }
10. else if (args == 'L') {
11. return value.toLowerCase();
12. }
13. else if (args == 'T') {
14. var i = 0;
15. var titleCase = '';
16. value = value.toLowerCase();
17. var temp = value.split(' ');
18. for (var ele of temp) {
19. ele = ele.charAt(0).toUpperCase() + ele.slice(1);
20. titleCase=titleCase+' '+ele;
21. }
22. console.log(titleCase)
23. return titleCase;
24. }
25. else {
26. return value;
27. }
28. }
29. }

3. Modify the code in **message.component.html**file

1. <div \*ngIf="val" id="msgText">
2. {{msg | alphaCase : 'T'}}
3. </div>

4. Enter the below code in **alpha-case.pipe.spec.ts**file

1. import { AlphaCasePipe } from './alpha-case.pipe';
2. describe('AlphaCasePipe', () => {
3. let pipe:any;
4. beforeEach(() => {
5. pipe = new AlphaCasePipe();
6. });
7. it('create an instance', () => {
8. expect(pipe).toBeTruthy();
9. });
10. it('Transform to UpperCase',()=>{
11. let testInput = 'mY iNfY';
12. expect(pipe.transform(testInput,'U')).toMatch('MY INFY');
13. });
14. it('Transform to LowerCase',()=>{
15. let testInput = 'mY iNfY';
16. expect(pipe.transform(testInput,'L')).toMatch('my infy');
17. });
18. it('Transform to TitleCase',()=>{
19. let testInput = 'mY iNfY';
20. expect(pipe.transform(testInput,'T')).toMatch('My Infy');
21. })
22. });

5. Modify the code in **app.module.ts**file by importing the pipe and adding it in the declaration section as shown below.

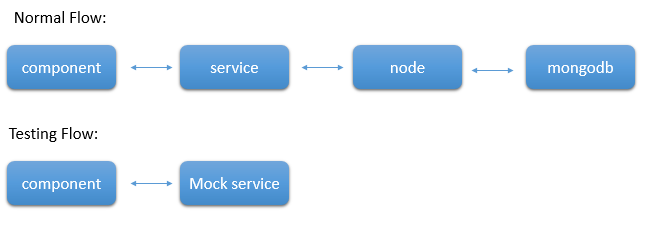
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { AppRoutingModule } from './app-routing.module';
4. import { AppComponent } from './app.component';
5. import { MessageComponent } from './message/message.component';
6. import { AlphaCasePipe } from './message/alpha-case.pipe';
7. @NgModule({
8. declarations: [
9. AppComponent,
10. MessageComponent,
11. AlphaCasePipe
12. ],
13. imports: [
14. BrowserModule,
15. AppRoutingModule
16. ],
17. providers: [],
18. bootstrap: [AppComponent]
19. })
20. export class AppModule { }

6. Run the test cases by using the command **ng test**, and observe that test cases created are being passed.

**Note:**In Angular testing, you have to mention the dependencies everywhere. For example, if the pipe is being used in a component, then the pipe should be declared in the declarations section of the components spec file. And if that component is used in app.component.html, the pipe should be added to the declarations of the app.component.spec.ts file as well. Else the test cases will fail.

If the component is using a service, then when you are testing the component, you need the service as well. If the service is connecting to a node backend, which is in turn connecting to a mongodb, then we need the entire setup just to test a component.

Mocking is the practice of replacing the actual service object with a dummy or mock service object. The mock service will be returning some hard coded dummy data instead of the actual data from the server.



Thus you don't need the entire set up to test the component alone.

To mock a service, we need to create a dummy service. Let's say that our original service (MessageSevice) has a getName() and it returns an observable of a string, which is returned from the back-end server. Then your mock service will look like:

**mock.service.ts:**

1. getName():Observable<string> {
2. return of("Hello World");
3. }

As you saw in Observables, ***of*** is used to convert a value into an Observable. In the component, where the service is being used, you need to update its spec file. For example, if the message component is using the service, then you need to update the providers of the **message.component.spec.ts** file as shown below:

1. providers:[{provide:MessageService,useClass:MockService}]

Thus during testing, MockSevice object will be injected instead of the MessageService object. Thus you can mock a service and test the component alone.

If your application uses routes, they should be included in the app.component.spec.ts. Else, the test cases will fail.

The below lines need to be included in the app.component.spec.ts:

1. import {APP\_BASE\_HREF} from '@angular/common';
3. describe('AppComponent', () => {
4. const routes:Routes=[
5. {path:'message',component:MessageComponent}
6. ]
7. beforeEach(async(() => {
8. TestBed.configureTestingModule({
9. declarations: [
10. AppComponent,
11. MessageComponent,
12. HelloPipe
13. ],
14. imports:[RouterModule.forRoot(routes)],
15. providers: [{provide: APP\_BASE\_HREF, useValue : '/' }]
16. }).compileComponents();

While running unit test cases for Angular application, you can even create code coverage reports using a tool called **Istanbul.**

Code coverage reports allows us to see any part of the code which is not properly tested by our unit test cases

To generate a code coverage report, run the following command

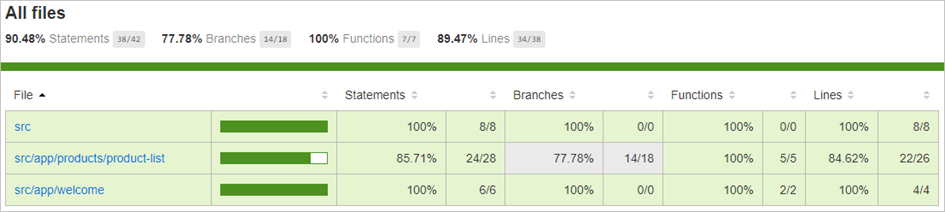
1. ng test --code-coverage

Once the tests execution are done, a new folder with name 'coverage' will get created in the project folder. Open coverage folder and run index.html page which will display the report with the source code and code coverage values.

Using code coverage percentages, you can estimate how much of our code is tested.

Following is a sample of code-coverage report

> ng test --no-watch --code-coverage



Now let's learn how to build and deploy the angular application.