**Revise:**

**Day 1.**

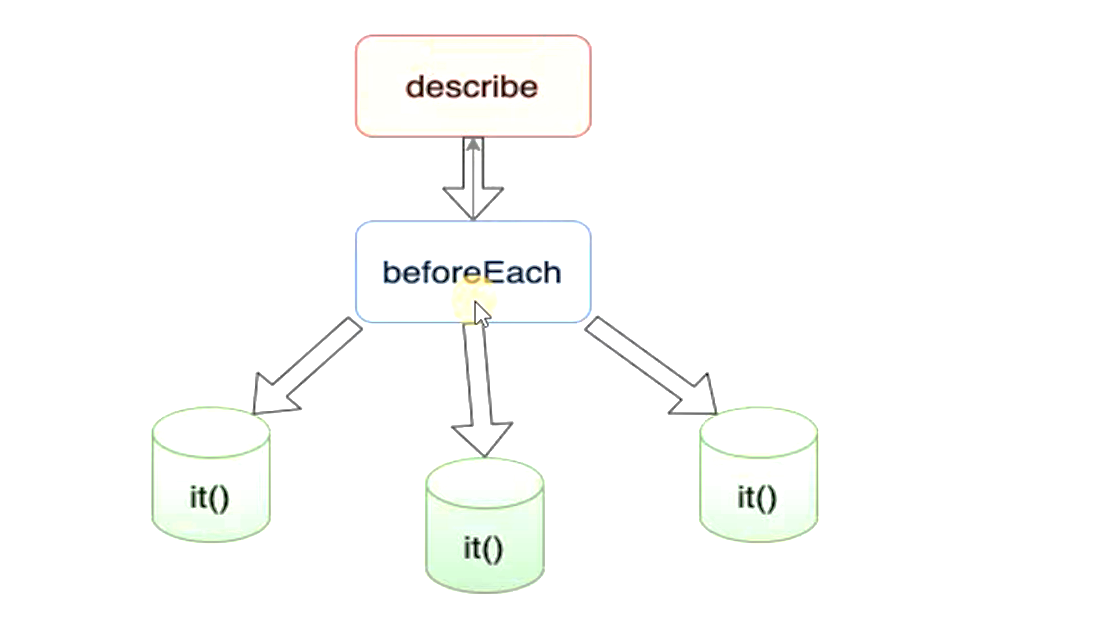
**What is Jasmine?**

**Ans:** It is an open-source Java script **Testing Library/ framework** to test application.

**What is Karma?**

**Ans:** It is a testing automation tool. Karma is open-source tool.

**Workflow of Unit test:**



We have to write test cases in respective **spec.ts** file.

* In order to write jasmine async test cases we can use promise.

For writing unit test cases we need

TestBed. If you will open spec.ts file then we will have to import.

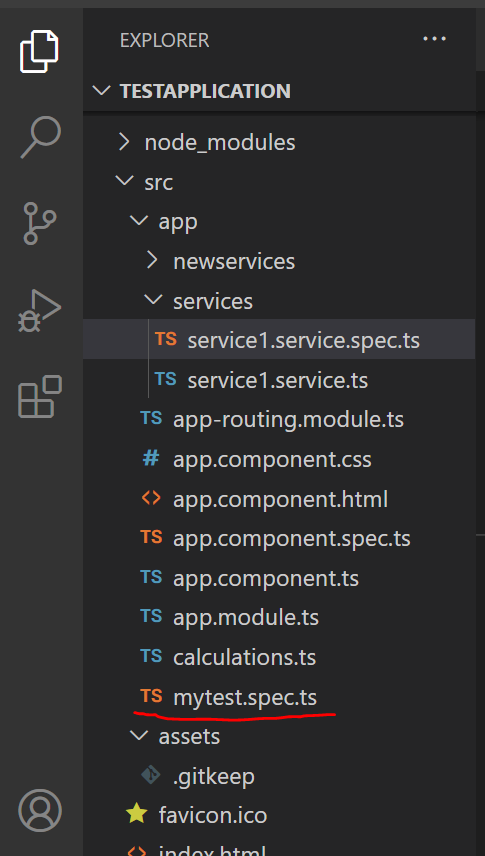
To exclude any of test we have to use **x** before it.

**import { TestBed } from '@angular/core/testing';**

We can create a new file also create it as testing file.

**Step 1.** Select App folder and add new file. Name of file must be followed by spec.ts.

**Example**: name of file. Mytest.spec.ts



**Step 2:** Import

**import { TestBed } from '@angular/core/testing';**

**and add describe method.**

import { TestBed } from '@angular/core/testing';

import { RouterTestingModule } from '@angular/router/testing';

import { AppComponent } from './app.component';

import { addTwoNumber, divisionOfTwo, multiplication,substraction} from './calculations';

describe('My Custom Suite',()=>{

  beforeEach(()=>{

    console.log("Before Each called");

  })

  it('My Test',()=>{

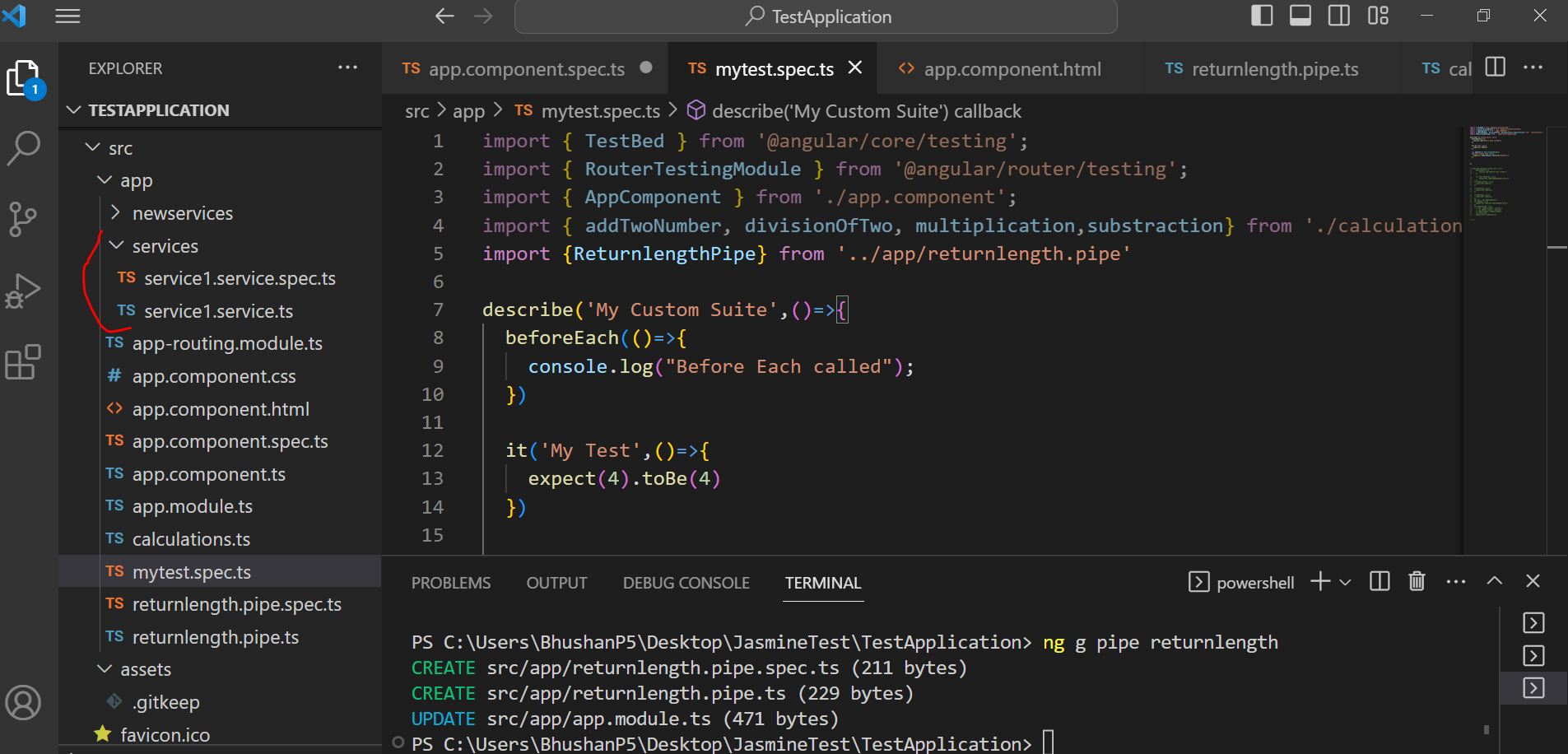
    expect(4).toBe(4)

  })

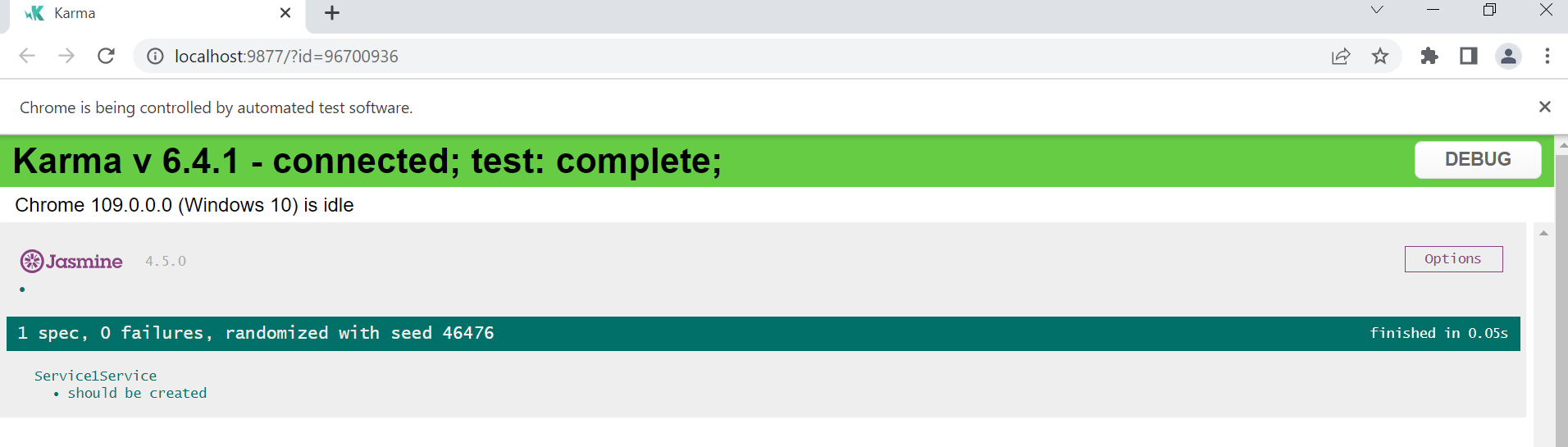
})

**To Run test tests in only selected folder.**

**Step 1.** Create new folder as service and add new service inside service folder.



ng test --include src/service



**Add a Custom Pipe and write a test case for testing it.**

**Step 1. Add a pipe as returnLength and modify transformMethod.**

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

  name: 'returnlength'

})

export class ReturnlengthPipe implements PipeTransform {

  transform(value: string): number {

    return value.length;

  }

}

**Step 2. Write test case in respective file.**

import { ReturnlengthPipe } from './returnlength.pipe';

describe('ReturnlengthPipe', () => {

  it('create an instance', () => {

    const pipe = new ReturnlengthPipe();

    expect(pipe).toBeTruthy();

  });

  let myPipe=new ReturnlengthPipe();

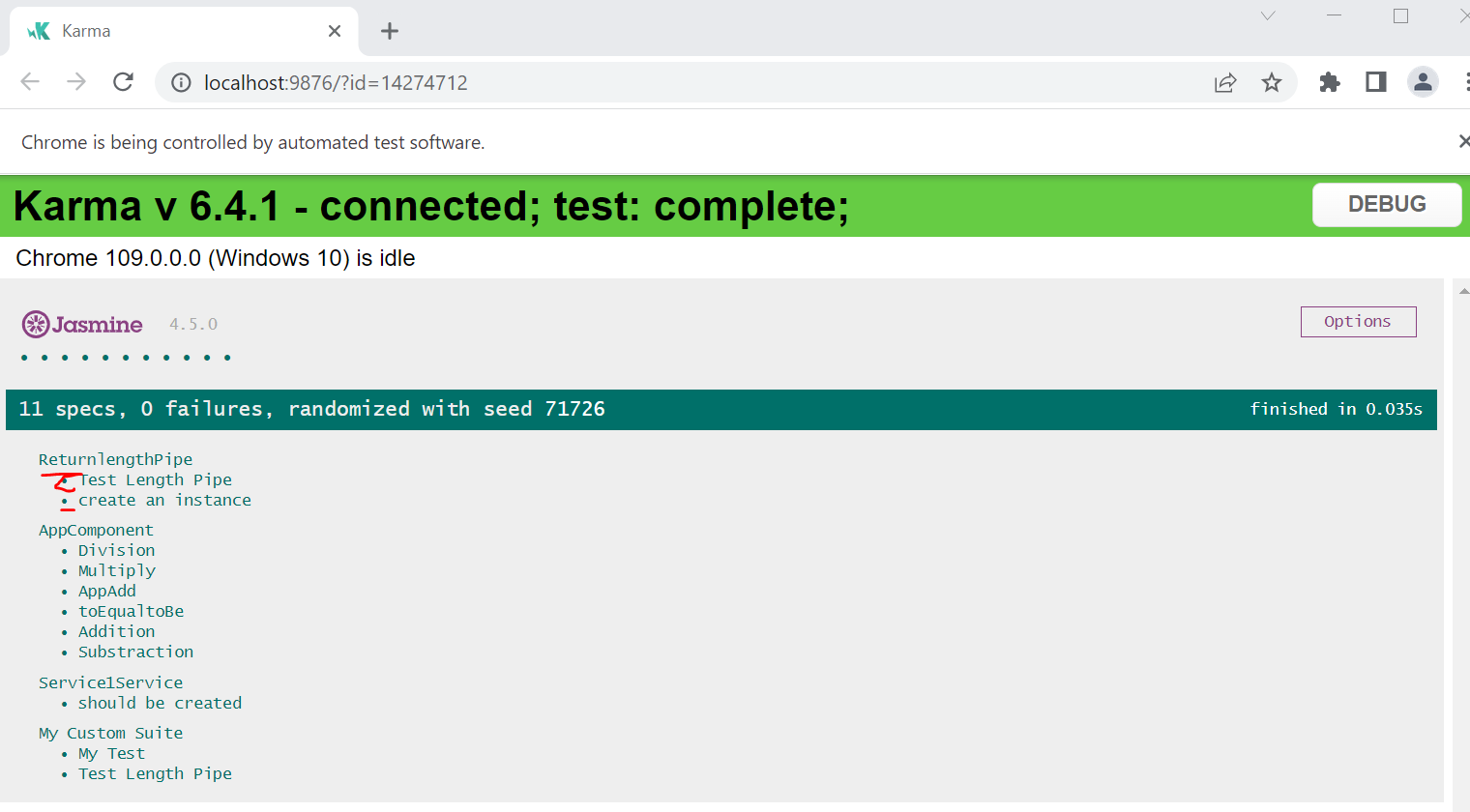
  it("Test Length Pipe",()=>{

    expect(7).toBe(myPipe.transform('Bhushan'))

  })

});

**Step 3: run tests again.**



**Important Note:**

1. Karma run tests on multiple browsers simultaneously
2. We can run multiple test at same time.
3. We can have multiple assertions in a test.
   1. **Ex:**

let myPipe=new ReturnlengthPipe();

it("Test Length Pipe",()=>{

expect(7).toBe(myPipe.transform('Bhushan'))

expect(3).toBe(myPipe.transform('Tom'))

})

1. Will given below test will pass or fail?

it('should do something….', (done: any) =>

 {    expect(true).toBe(false);});

1. Consider

let isValidUser:boolean =true

Which of the following Jasmine matchers can be used assert that the value of "isPrime" is not true?

🡪

* toNotEqual(true)
* toMatch(false)
* toNotMatch(true)
* toEquals(false)
* not.toBe(true)

it('should do something….', (done: any) =>

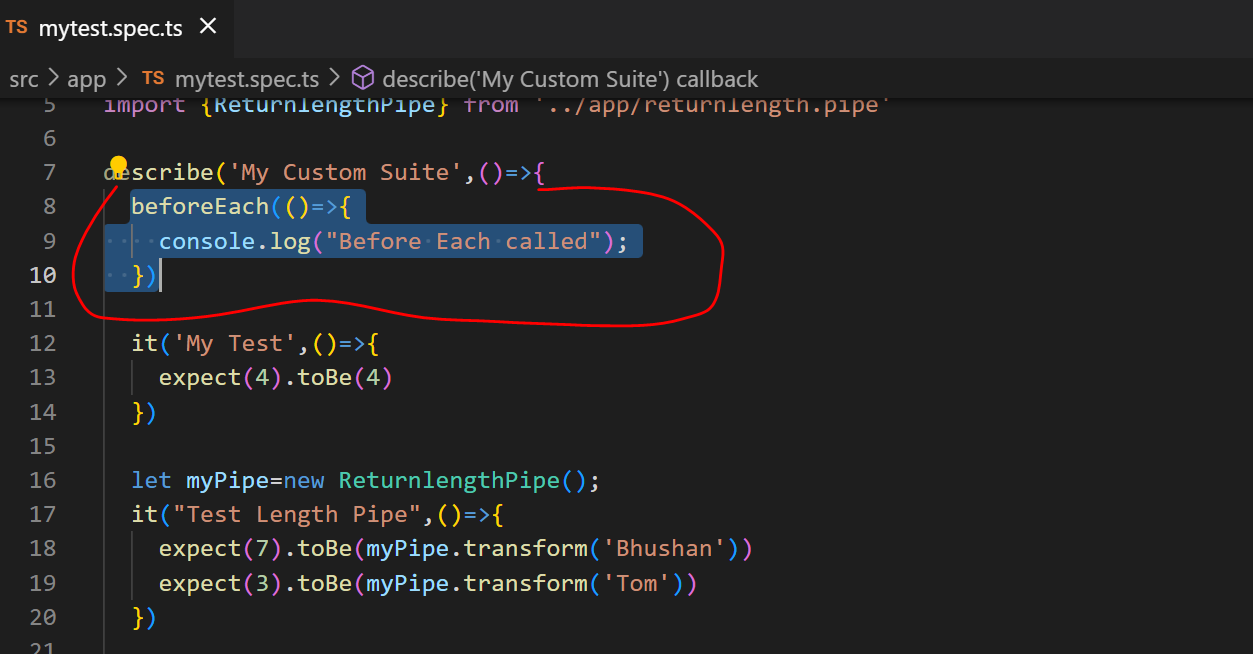
 {    expect(true).toBe(false);});

let isValidUser:boolean=true;

expect(isValidUser).not.toBe(true)

}

So, far we have seen Before Each Method



So similarly we have beforeAll, afterEach and afterAll

Methods.

**Can you guess what will be use of these methods?**

So, sometimes we have to create object of any class then we can create in beforeEach or BeforeAll so, similarly we can use **afterAll method as teardown** method to nullify object.

**TestBed and Component Fixture.**

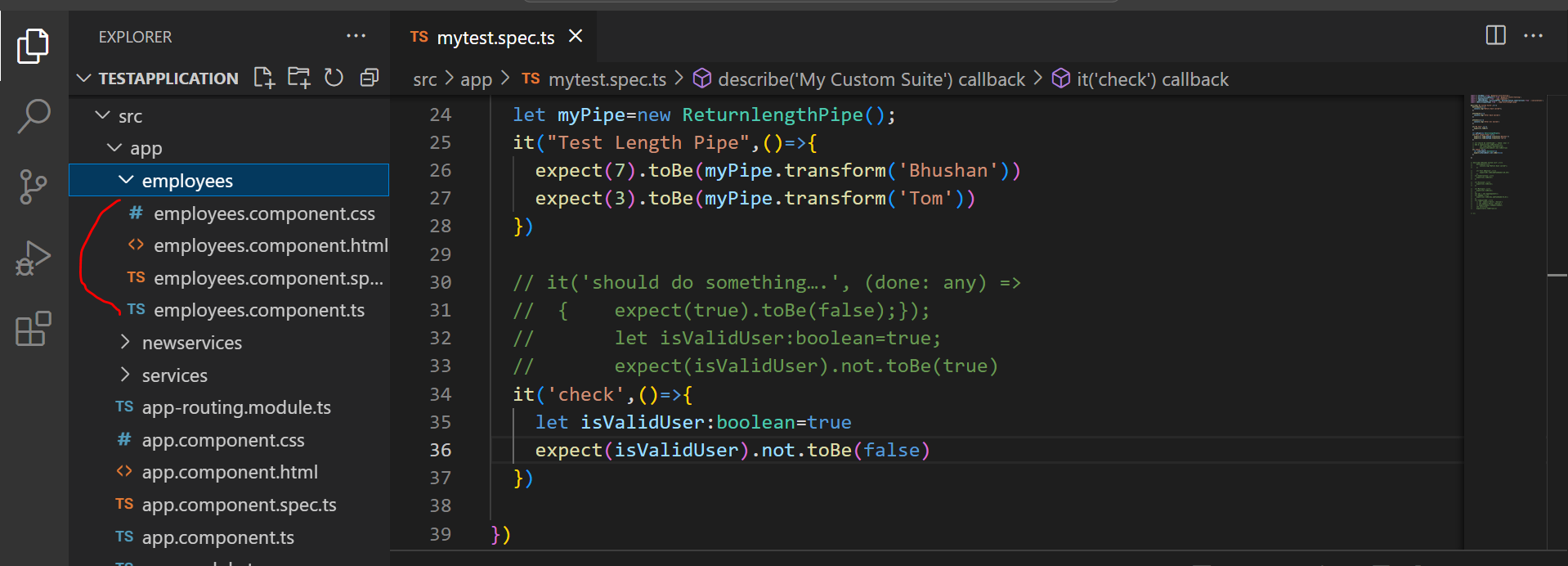
Angular TestBed allows us to do easily test behaviour that depends on Angular framework.

When to use TestBed?

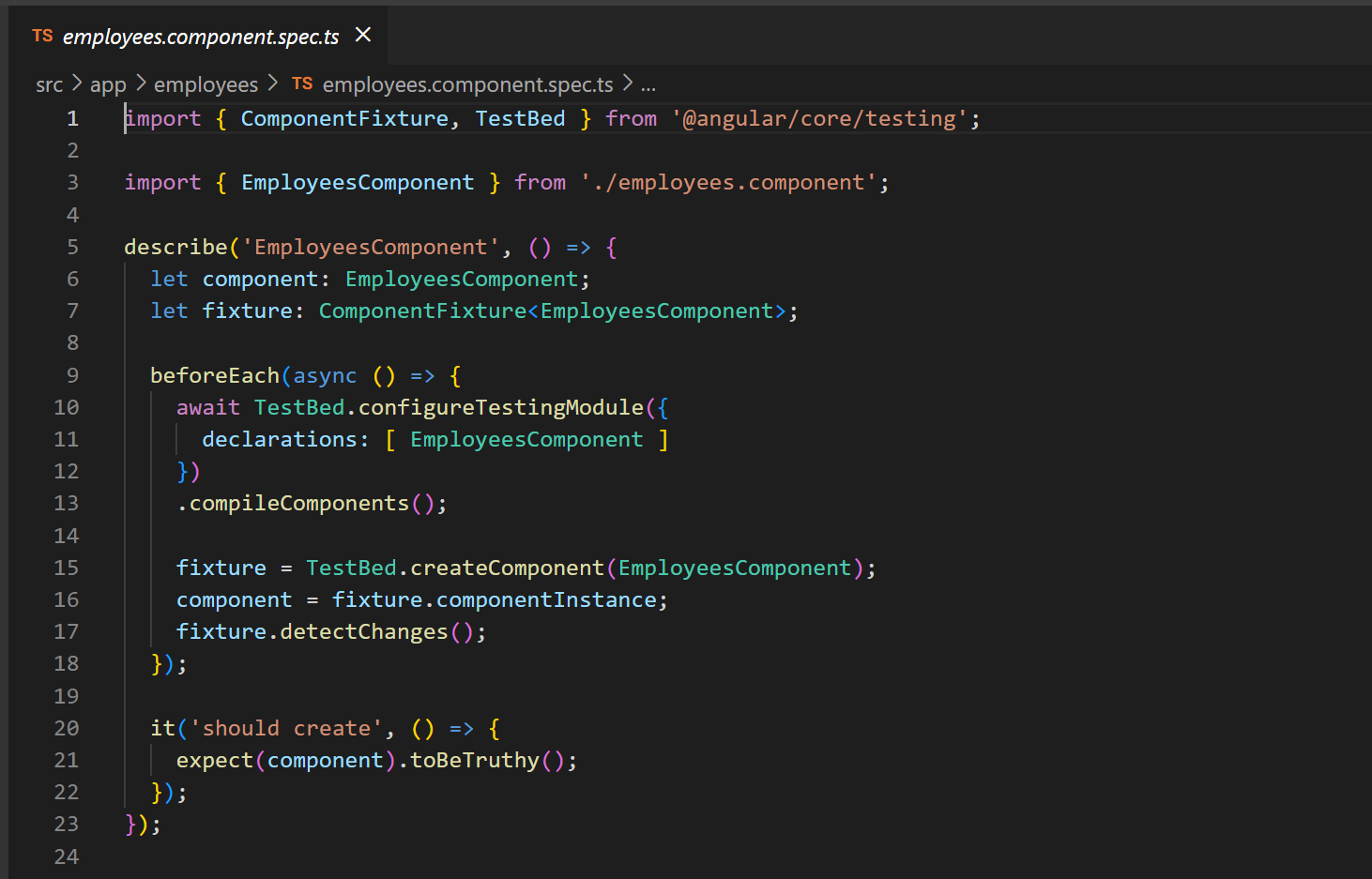
* Using this we can test interaction of directives or components with it’s template.
* It allows to test change detection.
* It allows us to test DI framework.
* It allows us to test user interaction with click events or input fields.
* In simple words, using test bed we can test HTML elements, css.

In order to understand it well, we will create new component.

ng g c employee



Open employee.component.spec.ts



So, using TestBed and Fixture we can access members of components. Like template, css.

If there will be some services we have to test then we can add array of providers, imports.

import { ComponentFixture, TestBed } from '@angular/core/testing';

import { EmployeesComponent } from './employees.component';

describe('EmployeesComponent', () => {

  let component: EmployeesComponent;

  let fixture: ComponentFixture<EmployeesComponent>;

  beforeEach(async () => {

    await TestBed.configureTestingModule({

      declarations: [ EmployeesComponent ],

      providers:[],

      imports:[]

    })

    .compileComponents();

    fixture = TestBed.createComponent(EmployeesComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});

is created when a component instance using the TestBed class it returns **ComponentFixture<T>**

In above code we can see

let fixture: ComponentFixture<EmployeesComponent>;

**SpyOn 🡪**

* It is a Jasmine method. Jasmine spy helps us to mock the execution of angular methods.
* Using this we can check whether any method called or not.
* SpyOn also helps us to return dummy values.
* SpyOn can call original function using .and.callThrough().

Let’s demonstrate spyOn method.

**Step 1**. Create two methods in newly created component.

  getFullName(fname:string, lname:string){

    return fname+''+lname

  }

  logFullName(){

    let fullName:string=this.getFullName('Bhushan','Kumar');

    let fullName2:string=this.getFullName('Rakesh','Kumar');

    console.log(fullName)

  }

**Step 2. Now, write a test case to spyOn on method.**

it('Spy On To Check Method called or Not',()=>{

    spyOn(component,'getFullName')

    component.logFullName();

    // To check whether called or not

    expect(component.getFullName).toHaveBeenCalled();

    // To test whether called 2 times or not

    expect(component.getFullName).toHaveBeenCalledTimes(2);

  })

**Note:**

In angular, Angular Service, Angular component class, custom pipe class can be mocked.

Given below are wrong ways to run test. We have already seen how to execute test.

* ng test giventest.spec.ts
* ng test --test mytest.spec.ts
* ng test --file mytest.spec.ts

**Change Detection:**

We have seen what is change detection. Change detection means on doing changes in angular it will be reflected in view/template.

Let’s see how we can utilise it in angular test case.

**Step 1**. We will use same component that is employee. Open **employee.component.html** and write a simple html code

<h1>Student Result</h1>

<button (click)="getResult()">Get Result</button>

<h1>{{getResult()}}</h1>

**Step 2.** Open **employee.component.ts.** and add a simple function.

getResult(){

    if(true){

      return "Pass"

    }

    else{

      return "Fail"

    }

  }

Now we will have to write a test to check when such changes will be detected.

**Step 3.**

1. To write test case for changes detection we have to get html element in spec.ts file.

To achieve this follow given below steps:

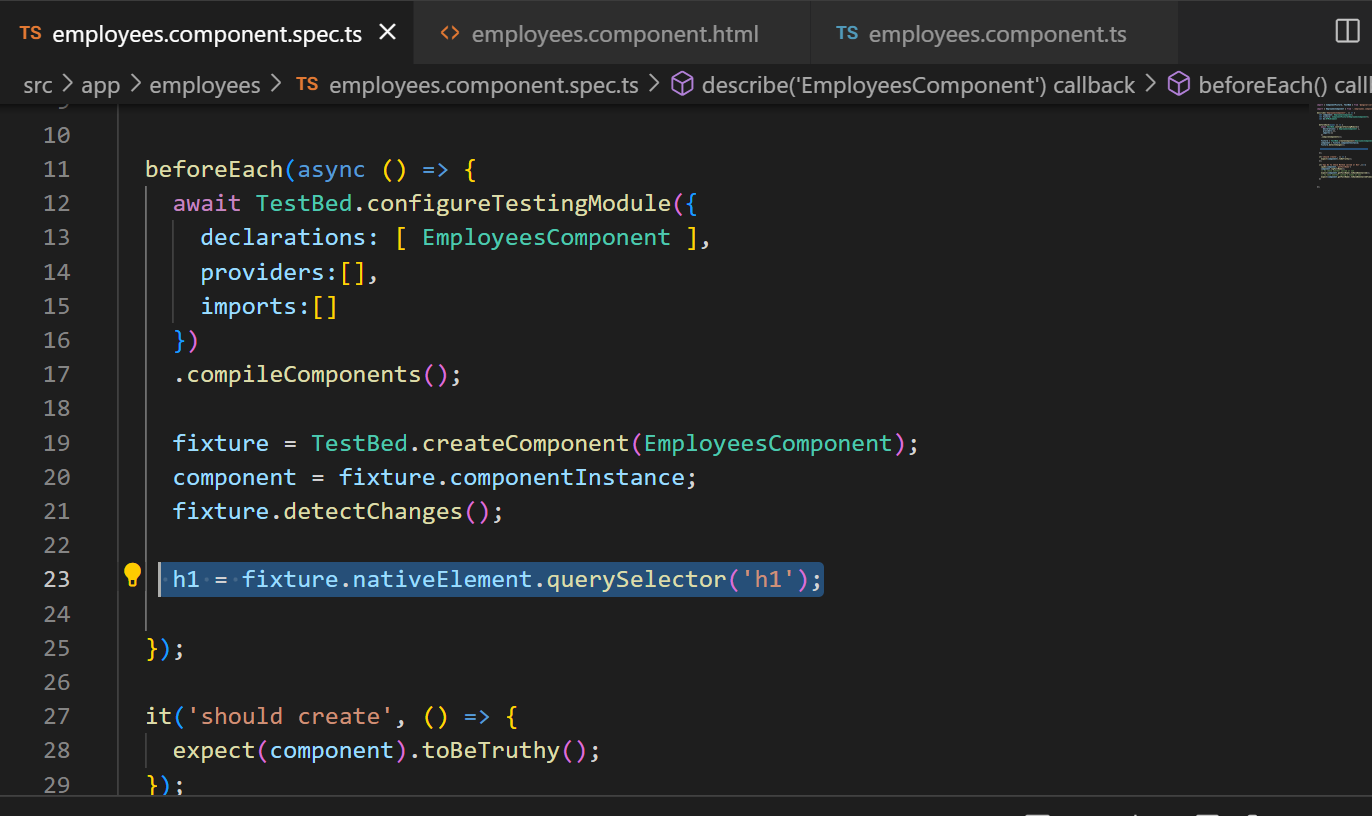
let h1:HTMLElement

add above code add it in describe.



Now, in before each create html element.

 h1 = fixture.nativeElement.querySelector('h1');



Now, create a test as given below:

  it('Test Change Detection',()=>{

    component.getResult();

    expect(h1.textContent).toBe(component.getResult())

  })

To check whether on click will it detect changes or not. At this point our test will be failed because it won’t detect changes.

it('Test Change Detection',()=>{

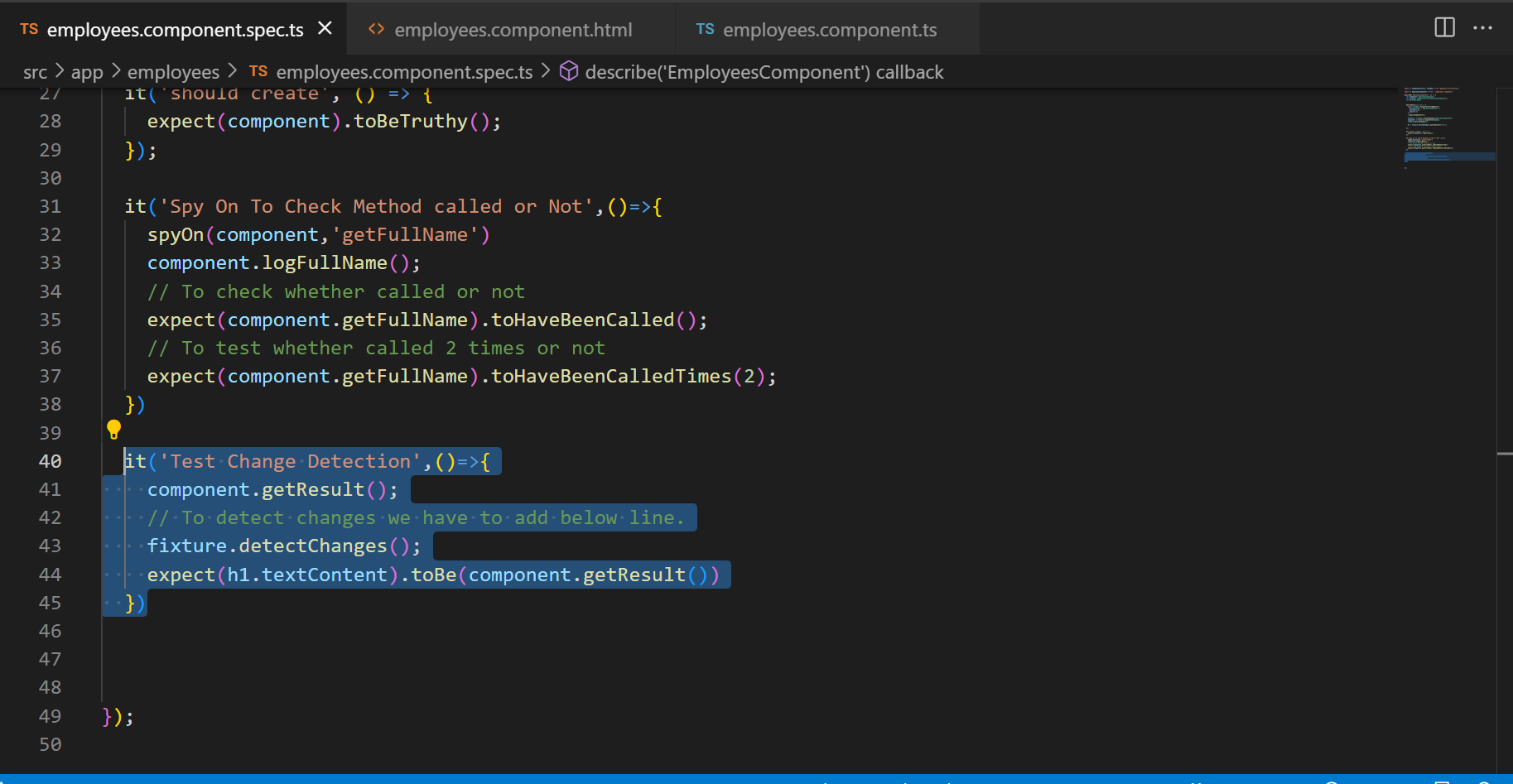
    component.getResult();

    // To detect changes we have to add below line.

    fixture.detectChanges();

    expect(h1.textContent).toBe(component.getResult())

  })



**Debug Element and DOM event.**

I hope you are familiar with html element and events in html.

Let’s understand what is debug element and dom event.

* **Debug Element:** It is angular class that contain all kind of references and method relevant to investigate element as well as component.

**How to get element:**

fixture.debugElement.query(By.css(‘h1’))

Instead of creating HTML element tree, Angular creates DebugElement tree that wrap native element.

* **nativeElement** is a property which unwrap the DebugElement and return platform specific element object.

It returns reference to DOM element

Visit this link for Code coverage 🡪 [Angular - Find out how much code you're testing](https://angular.io/guide/testing-code-coverage)