**KARMA INTEGRATION TESTING**

Try to generate the .spec file using the terminal to avoid writing the initial code always.

It is not possible to create instance of the component as we created it in Unit Test. We need to ask angular to create an instance of the component for us. For this we need to import the TestBed class.

Then in the beforeEach(), we need to create a dynamic Module and put our component in that Module. This can be done by using TestBed.configureTestingModule() method.

How is it declared?

BeforeEach(()=>{

let component : Component Name;

let fixture :Component Fixture<Component Name>;

TestBed.configuretestingModule({

declarations: [Component Name on Which Integration Testing is performed]

});

//will create a dynamic module

TestBed.createComponent(Component Name) will create a component and return type is a fixture. The fixture wraps the component instance. With that fixture, we can get access to both the component and it’s template.

Fixture = TestBed.createComponent(Component Name);

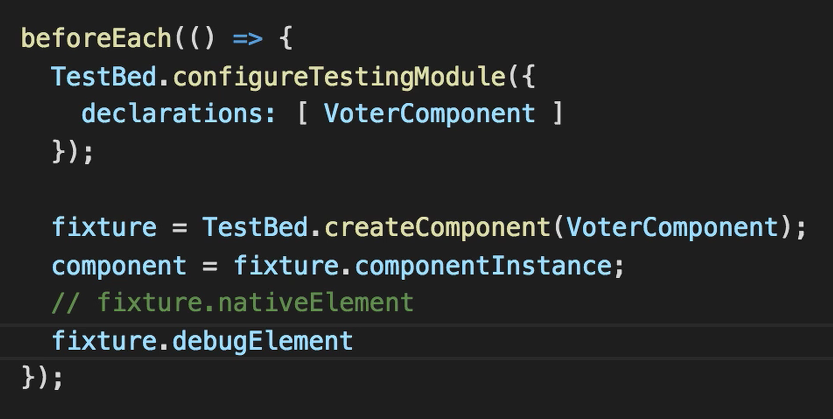
Component = fixture.componentInstance(); gives access to component

Fixture.nativeElement returns an HTML element which is the root Dom of this component.

It also has another method called fixture.debugElement that is a wrapper over the native element.

We can also run change detection using the fixture element.

});\



|  |  |
| --- | --- |
| TestBed.configureTestingModule | Used to create a dynamic Module which will be used to access the whole component (component + template) |
| TestBed.createComponent(Component) | Will return a fixture that acts as a wrapper over the component. This fixture is used to access both the component and its template. |
| Fixture.componentInstance() | Gives access to the component. |
| Fixture.nativeElement;  Fixture.debugElement; | Gives access to the template. |
| Fixture.detectChanges; | Detect CD in template. |

We don’t have to write this code all the time. If we create the component using the terminal (**ng generate component componentName**), this will be pre-written in the spec file.

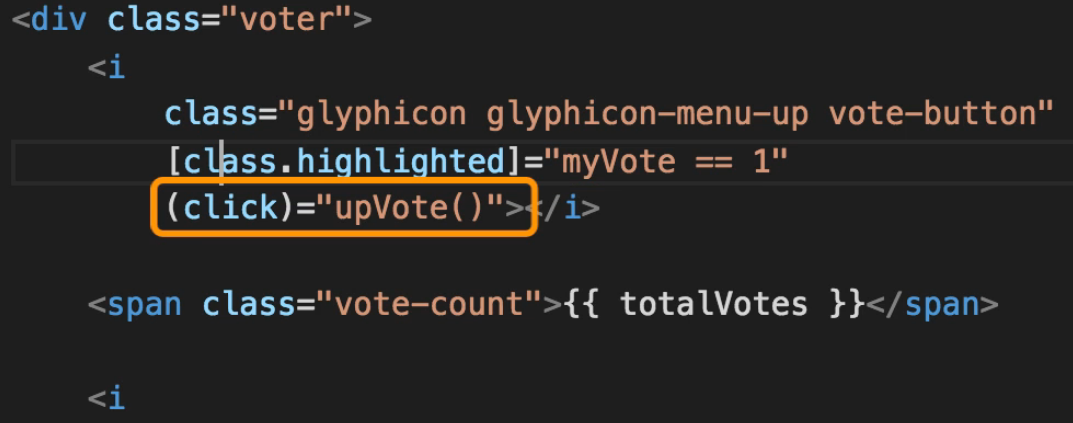
The automatically generated code looks something like below. It has two BeforeEach () blocks. We can keep the two BeforeEach () blocks or merge it into one block without the async() function.



IT on Bindings:

* Data Bindings/ Property Bindings
* Class Binding
* Style Binding
* Event Bindings

Sample .html file



* 1. **IT on Property Binding**

In IT on property binding, we must check if the correct value of a property is displayed on the template based on the value retrieved from the component. (.ts file)

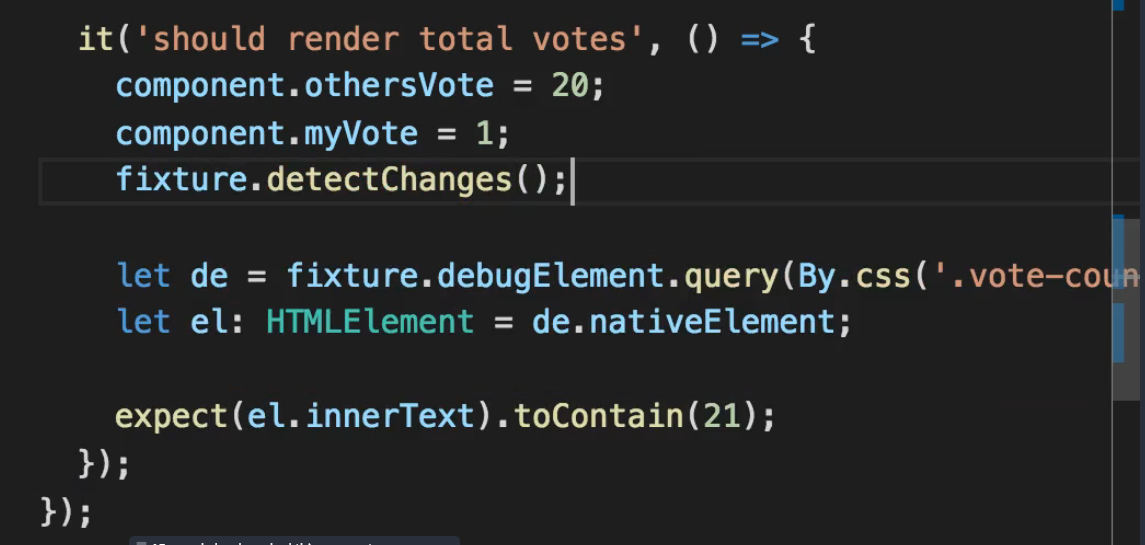
First, we must query the DOM and get reference to that element. For example,

**<span class=” vote-count”> {{**totalVotes**}} </span>**

So, we must get reference of that element by using the class “vote-count” by using the debugElement.query command (which gives the first occurrence of that element) and then check its inner HTML text by using the native element command.

And of course, this must be tested every time a CD occurs. So, use

**Fixture.detectChanges()**



Fixture.debugElement(0 helps us to get access to the template. Then, in the template we check the occurrence of a CSS class “vote-count”. When the “vote-count” class is found, check the inner html text and according to the condition it should be 21.

* 1. **IT on Class Bindings**

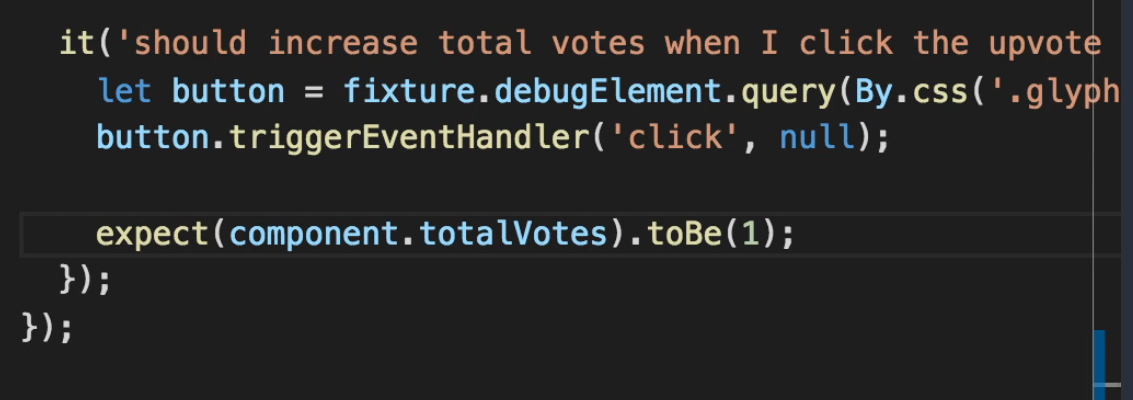
**[class. highlighted] = “myVote == 1”, apply highlighted class when myVote becomes 1.**



Here we do not check the inner html text by using the native element because we must check if the “highlighted” class is being applied on upvoting. So, query the predicate and get the first occurrence of the “glyphicon-menu-up” class and see if the highlighted class is being applied.

* 1. **IT on Event Binding**

**Call the upVote () method when I click on the button. And test is done to see if upVote () is increasing the totalVotes.**



**Note:** Write unit tests and integration tests in separate files.