46)Modal Component and structural directives- Section Introduction

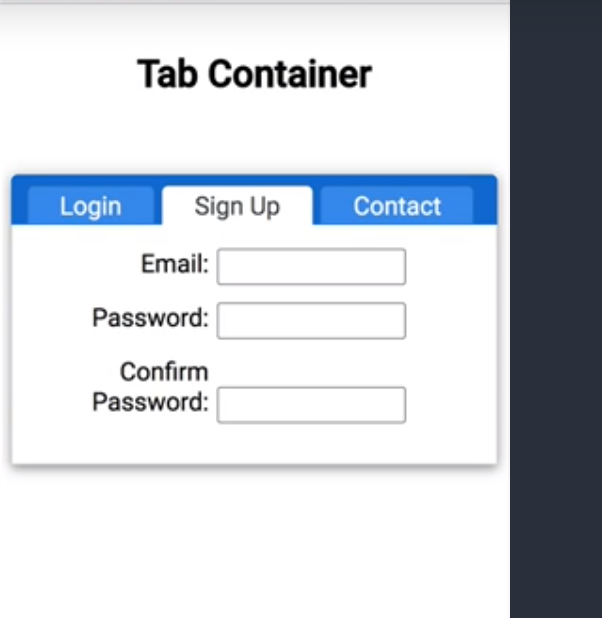
In this section we are going to build reusable modal component. we will use our au-input component here. we are going to create a authentication modal for signup. We will talk about structural directives , we will build our own structural directive and we are going to show multiple directives of a module can be combined togather and can interact with each in a very transparent way by using module specific service. Lets switch branch now. We need to go to remote branch **au-modal.** So run this command-

**git checkout –b au-modal origin/au-modal**

go to folder au-modal

35)Angular application Demo- A tab container with configurable look and feel

We will be making this-



All look and feel of component is going to be customizable. There will be couple of default styles and default look and feel for example tab buttons here but we can override it like this.



Here we define template that specifies how buttons should look. Ten we pass this template as input to our component. now tis new template will be used instead of default template. We will cover-

Ng-content

Ng-container

contentChildern

ngTemplateOutlet

template refrences

36) Switching To a New Branch - Reviewing the Tab Container Starting Point

Switch to new branch-

**git checkout -b au-tab-panel origin/au-tab-panel**

now we have static content. html-

<h2>

Tab Panel

</h2>

<div class="container">

<div class="tab-panel">

<ul class="tab-panel-buttons">

<li class="selected">Login</li>

<li>Sign Up</li>

<li>Contact</li>

</ul>

<div class="tab">

<form>

<div class="form-field">

<label>Email:</label><input>

</div>

<div class="form-field">

<label>Password:</label><input>

</div>

</form>

</div>

<div class="tab">

<form>

<div class="form-field">

<label>Email:</label><input>

</div>

<div class="form-field">

<label>Password:</label><input>

</div>

<div class="form-field">

<label>Confirm Password:</label><input>

</div>

</form>

</div>

<div class="tab">

<form>

<div class="form-field">

<label>Email:</label><input>

</div>

<div class="form-field">

<label>Message:</label>

<textarea></textarea>

</div>

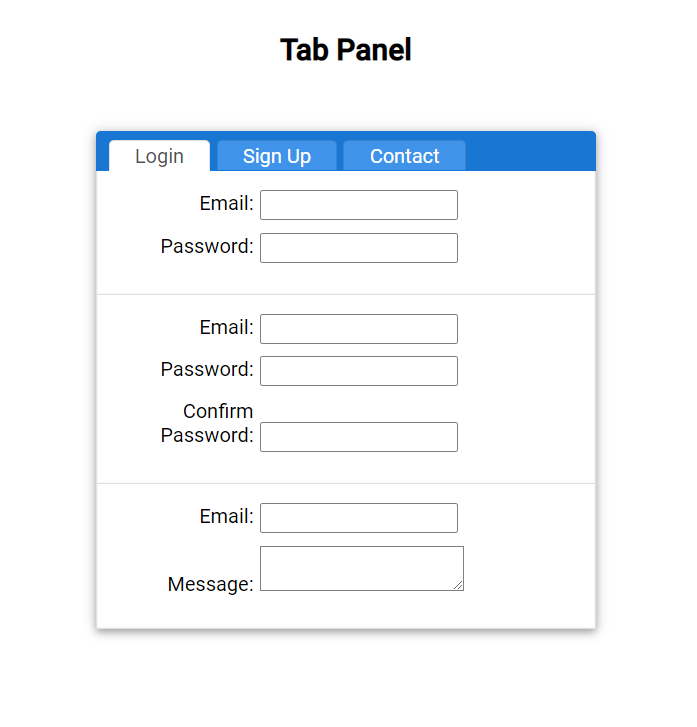
</form>

</div>

</div>

</div>

This is what we see in browser-



We do not have anything in ts. In view we have all inputs shown for all tabs. We want to show inputs depending upon which tab is selected.

37)Angular Component styling-Commonly Needed SaaS Features

Here we will see most common features of saas and we will see how those features really help us to build more maintainable styles of our component. we have 3sccss files,- app.compoennt, theme.scss and tab-panel.component.scss.

Theme.scss-

$primary: rgb(25, 118, 210);

$secondary: rgba(0, 0, 0, 0.117647);

$primary-tint-1: #3F94E9;

$text:rgba(26, 35, 38, 0.8);

$box-shadow: 0 1px 16px 0 rgba(0, 0, 0, 0.2), 0 2px 8px 0 rgba(0, 0, 0, 0.14), 0 4px 8px -1px rgba(0, 0, 0, 0.12);

$tab-border: 1px solid $secondary;

Here we have some variables. We can see that scss variable can not only be colors, we can specify a combined set of properties, for example to define border. Then w ecan use it in multiple places in our styles.

We can see it in action in

Tab-panel.component.scss-

@import "./theme";

.tab-panel {

box-shadow: $box-shadow;

.tab-panel-buttons {

list-style: none;

text-align: left;

margin-bottom: 0;

padding: 7px 0 0 10px;

background: $primary;

border-top-left-radius: 4px;

border-top-right-radius: 4px;

li {

display: inline-block;

color: white;

background: $primary-tint-1;

padding: 2px 20px;

cursor: pointer;

border-top-left-radius: 4px;

border-top-right-radius: 4px;

border-top: $tab-border;

border-left: $tab-border;

border-right: $tab-border;

margin-right: 5px;

&.selected {

background: white;

color:$text;

border-bottom: 1px solid white;

}

}

}

.tab {

border-left: $tab-border;

border-right: $tab-border;

border-bottom: $tab-border;

padding: 15px 25px;

}

}

We can see here we import theme.scss file and we use variable declared in theme.scss file. as w e can see , we can use nested styles in scss.

38)Implementing the Tab Panel Component-Initial version Up and running

Here we will define api of our component. we will have a outer component. we will call it au-tab-panel. inside of it ,we will be having series of tab components. So we are going to be needing 2 components that are going to be closely tied togather, so they will be tightly coupled.

We will have au-tab-panel component and within it we will have series of tabs and those will be inside tab container. For each tab we will have a title property. Lets see how we do it.

We will use content projection, in both of our components. We also made a minor change in scss, in au-tab.component.scss we have imported all variable s and defined style for tab class.

app.component.html-

<h2>

Tab Panel

</h2>

<div class="container">

<au-tab-panel>

<au-tab title="Login">

<form>

<div class="form-field">

<label>Email:</label><input>

</div>

<div class="form-field">

<label>Password:</label><input>

</div>

</form>

</au-tab>

<au-tab title="Sign Up">

<form>

<div class="form-field">

<label>Email:</label><input>

</div>

<div class="form-field">

<label>Password:</label><input>

</div>

<div class="form-field">

<label>Confirm Password:</label><input>

</div>

</form>

</au-tab>

<au-tab title="Contract">

<form>

<div class="form-field">

<label>Email:</label><input>

</div>

<div class="form-field">

<label>Message:</label>

<textarea></textarea>

</div>

</form>

</au-tab>

</au-tab-panel>

</div>

Au-tab-panel.component.html-

<div class="tab-panel">

<ul class="tab-panel-buttons">

<li class="selected">Login</li>

<li>Sign Up</li>

<li>Contact</li>

</ul>

<ng-content></ng-content>

</div>

Au-tab.component.html-

<div class="tab">

<ng-content></ng-content>

</div>

Au-tab.component.scss-

@import '../theme.scss';

.tab {

border-left: $tab-border;

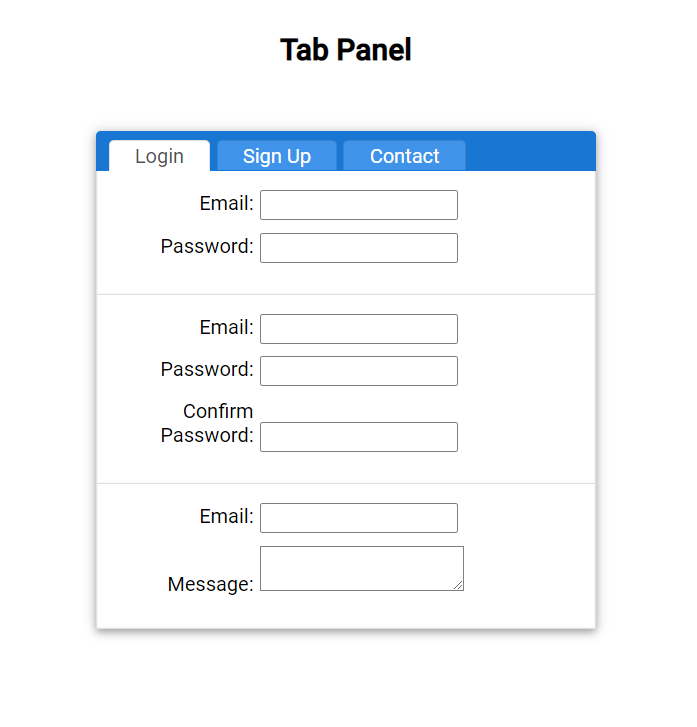
border-right: $tab-border;

border-bottom: $tab-border;

padding: 15px 25px;

}

If we run our code, we can see our component is displayed correctly.



And with this, what we are going to do next is we are going to optimize and simplify a lot html and css of our component. this will be good example of how the angular template features and angular components styling fetaures really make it simple to build very simple html and css using agular template mechanism and view encapsulation solition fetaues.

40) Content projection with @ContentChildern and The AfterContentInit Lifecycle Hook

App.compoent.html-

<au-tab-panel>

<au-tab title="Login">

<form>

<div class="form-field">

<label>Email:</label><input>

</div>

<div class="form-field">

<label>Password:</label><input>

</div>

</form>

</au-tab>

<au-tab title="Sign Up">

<form>

<div class="form-field">

<label>Email:</label><input>

</div>

<div class="form-field">

<label>Password:</label><input>

</div>

<div class="form-field">

<label>Confirm Password:</label><input>

</div>

</form>

</au-tab>

<au-tab title="Contract">

<form>

<div class="form-field">

<label>Email:</label><input>

</div>

<div class="form-field">

<label>Message:</label>

<textarea></textarea>

</div>

</form>

</au-tab>

</au-tab-panel>

Now what we want is we want to pass title from au-tab to au-tab-panel, so that we can display those names in tabs-container. We did this in au-tab-panel.component.ts-

import {QueryList} from '@angular/core';

export class AuTabPanelComponent implements OnInit, AfterContentInit {

@ContentChildren(AuTabPanelComponent) tabs: QueryList<AuTabPanelComponent>

constructor() { }

ngOnInit() {

}

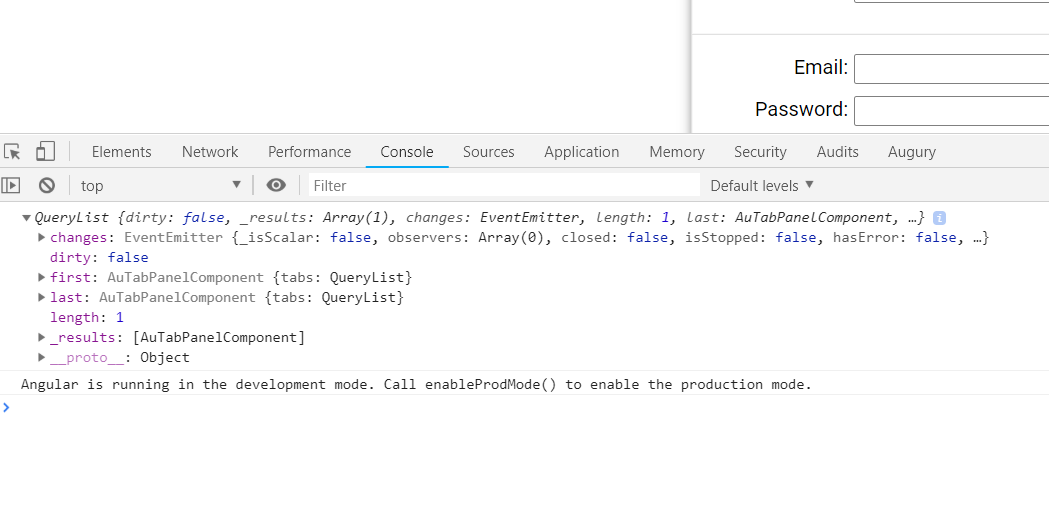
ngAfterContentInit() {

console.log(this.tabs);

}

}

Output-



So now QueryList is type defined in @angular/core. It is iterable. Click on QueryList in IDE to see what method it provides. queryList is iterable so w ecanpass it directly to the template.

Then we use it on our template like this-

Au-tab-panel.component.html-

<ul class="tab-panel-buttons" \*ngIf="tabs">

<li \*ngFor="let tab of tabs">{{tab.title}}</li>

<!-- <li class="selected">Login</li>

<li>Sign Up</li>

<li>Contact</li> -->

</ul>

<ng-content></ng-content>

We use ngIf to avoid flickering effect.

41)The Tab Container Component - See The Initial Implementation Up And Running

First we define property called selected on each tab. If that property is false, then we do not show anything in our that tab.

Au-tab.component.ts-

<div class="tab" \*ngIf="selected">

<ng-content></ng-content>

</div>

The in panel component we find which tab is selected. As querySelector is iteratable evry property that is avalaible on array is avalible on it. In panel component we select a first element if no elemnt is selected. Code-

ngAfterContentInit() {

const selected = this.tabs.find(tab => tab.selected);

if (!selected) {

this.tabs.first.selected = true;

}

}

So herew e are selecteing the first tab. Now we wnt to selected different tab on each click. So we call a function on each click.

Tab-panel.component.html-

<ul class="tab-panel-buttons" \*ngIf="tabs">

<li \*ngFor="let tab of tabs" [ngClass]="{selected:tab.selected}" (click)="selectTab(tab)">

{{tab.title}}

</li>

</ul>

<ng-content></ng-content>

Tab-panel.component.ts-

selectTab(selectedTab: AuTabComponent) {

this.tabs.forEach(tab => tab.selected = false);

selectedTab.selected = true;

}

This is initial version of our tab up and running.

Lets say we want to select second tab, we can do this by-

if (!selected) {

const secondTab: AuTabComponent = this.tabs.find((tab: AuTabComponent, i: number) => i === 1);

secondTab.selected = true;

}

42. Angular Advanced Features - ng-template , ng-container, ngTemplateOutlet

We are gong to add option of overriding the default look and style of tab container. So not only the style but ability to override the HTML itself. So if we do not provide anything than default tab container buttons will be used, but if we provide any html than that will be used.

Au-tab-panel.component.html-

<ng-template #defaultTabsHaeder lets-tabs="tabs">

<ul class="tab-panel-buttons" \*ngIf="tabs">

<li \*ngFor="let tab of tabs" [ngClass]="{selected:tab.selected}" (click)="selectTab(tab)">

{{tab.title}}

</li>

</ul>

</ng-template>

<ng-container \*ngTemplateOutlet="defaultTabsHaeder;"></ng-container>

<ng-content></ng-content>

In html we define a template. We give it a name. then we define input property for this template. We call it tabs and assign value of tabs variable from component. Now we wnt see tab component on screen because we are not using our template.it is just definition of section of our component template. We now need to take ng-template and instantiate it. There are multiple ways of consuming templates on of them is to use ng-container.

Ng-container will not result in any DOM element to be rendered to the screen but it will allow us to for example to apply structural directive on top of it such as \*ngIf, \*ngFor. So we can use it as container tag to apply structural directives. Here we use it to apply a structural directives that is \*ngTemplateOutlet. as a first argument we pass template that we want to render.as a second argument we can pass it optional context object. so this context object will contain inputs that are needed by template. Like this-

<ng-container \*ngTemplateOutlet="estimateTemplate;context:ctx">

</ng-container>

But in our case we need only one property tabs and we are already filling it at level of ng-template tag. But we could not have provided that value there if we did not know it upfront and we could alternatively provided in a completely programmatic way. Leys see how could we do it.

Lets see how we could have done it.

In tml-

<ng-template #defaultTabsHaeder lets-tabs="tabs">

<ul class="tab-panel-buttons" \*ngIf="tabs">

<li \*ngFor="let tab of tabs" [ngClass]="{selected:tab.selected}" (click)="selectTab(tab)">

{{tab.title}}

</li>

</ul>

</ng-template>

<ng-container \*ngTemplateOutlet="defaultTabsHaeder; context: tabsContext"></ng-container>

<ng-content></ng-content>

I ts-

get tabsContext() {

return {

tabs: this.tabs

};

}

So now we have same look and feel as before but we could now replace this template(defaultHeaderTemplate) by another template that was externally provided to the component.

For more details on tese directives refer these artciles-

<https://alligator.io/angular/reusable-components-ngtemplateoutlet/>

<https://blog.angular-university.io/angular-ng-template-ng-container-ngtemplateoutlet/>

<http://www.bentedder.com/angular-4-templates-passing-methods-context-ngTemplateOutlet-ngTemplateOutletContext/>

43. Learn Angular Template References And Input Template Partials

Lets learn how a angular component can not only have its own template but also receive template partials as inputs. One way to implement that is.

We declare a variable in component-

headerTemplate: TemplateRef<any>;

TemplateRef is refrence to angular template such as this-

<ng-template #defaultTabsHaeder lets-tabs="tabs">

<ul class="tab-panel-buttons" \*ngIf="tabs">

<li \*ngFor="let tab of tabs" [ngClass]="{selected:tab.selected}" (click)="selectTab(tab)">

{{tab.title}}

</li>

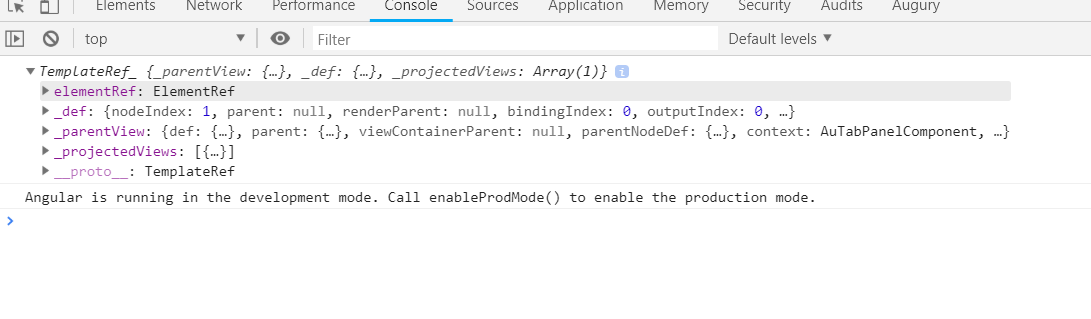
</ul>

</ng-template>

To better understand how ng- template works lets do a view child query to the tab-panel.component. we are going to query by template ref type. As in html we have only one ng-template tag, that is the instance that will be associated to our variable. we can query template by-

@ViewChild(TemplateRef) headerTemplate: TemplateRef<any>;

So this shows that variable of type TemplateRef will correspond to an instance of ng-template element. This is what we get if we print it on console-



This is what we get on console.

To learn more abput templateREf –

In our current usecase what we want is instead of retrieving this as viewchild query we want to retrieve template ref as component input using @Input. So we use this setup-

Au-tab-panel.component.ts-

@Input() headerTemplate: TemplateRef<any>;

Here we get the input paraemeter which is of type TemplateRef

Au-tab-panel.compoent.html-

<ng-template #defaultTabsHaeder lets-tabs="tabs">

<ul class="tab-panel-buttons" \*ngIf="tabs">

<li \*ngFor="let tab of tabs" [ngClass]="{selected:tab.selected}" (click)="selectTab(tab)">

{{tab.title}}

</li>

</ul>

</ng-template>

<ng-container \*ngTemplateOutlet="headerTemplate ?headerTemplate: defaultTabsHaeder; context: tabsContext"></ng-container>

<ng-content></ng-content>

Here we check whther tempalteRef is passed, if it is passed then use that otherwise we use defaultTabsHeader

In app.component.html-

<div class="container">

<ng-template #headerButtons>

<button>Login</button>

<button>Signup</button>

</ng-template>

<au-tab-panel [headerTemplate]="headerButtons">

<au-tab title="Login">

<form>

<div class="form-field">

<label>Email:</label><input>

</div>

<div class="form-field">

<label>Password:</label><input>

</div>

</form>

</au-tab>

<au-tab title="Sign Up">

<form>

<div class="form-field">

<label>Email:</label><input>

</div>

<div class="form-field">

<label>Password:</label><input>

</div>

<div class="form-field">

<label>Confirm Password:</label><input>

</div>

</form>

</au-tab>

<au-tab title="Contract">

<form>

<div class="form-field">

<label>Email:</label><input>

</div>

<div class="form-field">

<label>Message:</label>

<textarea></textarea>

</div>

</form>

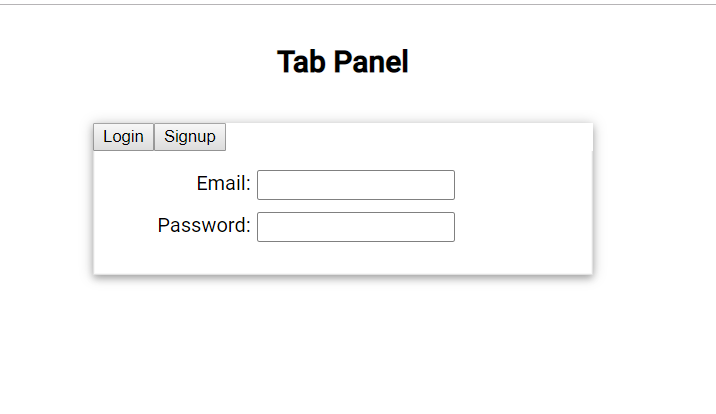
</au-tab>

</au-tab-panel>

</div>

Here we can see we are passing atemplate ref to au-tab-panel.component.ts. so now this template reference will be used.

If we run our program now, we can see that-



So now we are seeing the template that we are passing.

Now we want to show tabs depending upon which header button is clicked. This is how we do it, we use local refrence to call the method of au-tab-panel component, then we use local refrence to app au-tab component to that method. Code-

<h2>

Tab Panel

</h2>

<div class="container">

<ng-template #headerButtons>

<button (click)="tabPanel.selectTab(loginTab)">Login</button>

<button (click)="tabPanel.selectTab(signUpTab)">Signup</button>

</ng-template>

<au-tab-panel #tabPanel [headerTemplate]="headerButtons">

<au-tab title="Login" #loginTab>

<form>

<div class="form-field">

<label>Email:</label><input>

</div>

<div class="form-field">

<label>Password:</label><input>

</div>

</form>

</au-tab>

<au-tab title="Sign Up" #signUpTab>

<form>

<div class="form-field">

<label>Email:</label><input>

</div>

<div class="form-field">

<label>Password:</label><input>

</div>

<div class="form-field">

<label>Confirm Password:</label><input>

</div>

</form>

</au-tab>

<au-tab title="Contract">

<form>

<div class="form-field">

<label>Email:</label><input>

</div>

<div class="form-field">

<label>Message:</label>

<textarea></textarea>

</div>

</form>

</au-tab>

</au-tab-panel>

</div>

44) Testing The Tab Container Component

Here we ran some basic tests to test the functionality our component. to run test, run this command-

**npm test**

app.compoennt.spec.ts-

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

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

import { DebugElement } from '@angular/core';

import { AuTabPanelComponent } from './au-tab-panel/au-tab-panel.component';

import { AuTabComponent } from './au-tab/au-tab.component';

import { By } from '@angular/platform-browser';

describe('AppComponent', () => {

let component: AppComponent,

fixture: ComponentFixture<AppComponent>,

el: DebugElement,

tabPanel: DebugElement;

beforeEach(async(() => {

TestBed.configureTestingModule({

declarations: [

AppComponent, AuTabPanelComponent, AuTabComponent

],

}).compileComponents();

}));

beforeEach(() => {

fixture = TestBed.createComponent(AppComponent);

component = fixture.debugElement.componentInstance;

el = fixture.debugElement;

tabPanel = el.query(By.css('#tab-panel'));

fixture.detectChanges();

});

it('should create the test application', async(() => {

expect(component).toBeTruthy();

}));

it('should find only one tab inside the tab container', async(() => {

const tabs = tabPanel.queryAll(By.css('.tab'));

expect(tabs).toBeTruthy();

expect(tabs.length).toBe(1);

}));

it('should find login button marked as active', async(() => {

const selectedButton = tabPanel.query(By.css('.tab-panel-buttons li.selected')).nativeElement;

expect(selectedButton).toBeTruthy();

expect(selectedButton.textContent).toBe('Login');

}));

it('should display the Login Tab', async(() => {

const contactEmail = tabPanel.query(By.css('.login-email'));

expect(contactEmail).toBeTruthy();

}));

});

These test are self explanatory. In next lecture we will simulate some user interaction in test cases.

You can go thorugh lecture at once it wnt take much time.

45. Angular Component Testing - How To Simulate User interaction

Here we have introduced a new test cases where we click on button-

it('should switch to SignUp Tab', async(() => {

const tabButtons = tabPanel.queryAll(By.css('.tab-panel-buttons li'));

tabButtons[1].nativeElement.click();

fixture.detectChanges();

const signUpEmail = tabPanel.query(By.css('.signup-email'));

expect(signUpEmail).toBeTruthy();

const selectedButton = tabPanel.query(By.css('.tab-panel-buttons li.selected')).nativeElement;

expect(selectedButton).toBeTruthy();

expect(selectedButton.textContent).toBe('Sign Up');

}));