46)Modal Component and Structural Directives - Section Introduction

Here we will be building reuseable modal. We will use our previous components. We will be using structural directives and we will are going to show how the multiple directives of modules can be combined togather and can interact in a very transparent way by using module specific service. First we need to get the code of new branch, run-

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

then we go to au-modal component and run **yarn** command. In package.josnw e also have our components-

"au-input": "^3.0.1",

"au-tab-panel": "^2.0.1",

So we will use these components in our new component.

47)How To Implement a Modal Component - Reviewing The HTML and CSS

When ir modal is not displayed we dnt want to hide it, we want to remove it from dom.

The content of modal, the body of modal should freely interact with any of elements of page where modal is inserted ,for example if from inside of modal, we would like to call a service in application component, that should be very easy to implement.

also modal needs to be in middle of screen and its width will depend upon its content. also it should be very easy to pass input data to modal. Lets see how we can do it.

First in app.component.html we add this div-

<div class="modal-overlay">

</div>

We define modal-overlay class like this-

.modal-overlay {

display: inline-block;

position: fixed;

width: 100%;

height: 100%;

left: 0;

top: 0;

background: rgba(51,51,51,0.7);

z-index: 10;

}

We give width and height 100% so that it takes full width and height. W ekept its positon fixed, so that even if we scroll, the overlay is still there. Then we define z –index so that div is higher than content of page, so that it will prevent any clicking on elements underneath. Aslo it will detect the click outside the modal which you would like to respond by closing the modal. Also we give transparent background to div.we do this by giving fourth argument to rgba.

Inside this div we define div with modal-body class. This div will be centred and it will contain the content of modal. It should have any predefine size it should adapt to the size of content.

<div class="modal-overlay">

<div class="modal-body">

</div>

</div>

Inside this second dive we paste the au-tab-panel component which we made in last section. final html looks like this-

<div class="modal-overlay">

<div class="modal-body">

<au-tab-panel>

<au-tab title="Login">

<form>

<div class="form-field">

<au-fa-input id="email-field" icon="envelope">

<input auInput type="email" name="email" placeholder="Email" autocomplete="off">

</au-fa-input>

</div>

<div class="form-field">

<au-fa-input icon="lock">

<input auInput type="password" name="email" placeholder="Password" autocomplete="off">

</au-fa-input>

</div>

</form>

</au-tab>

<au-tab title="Sign Up">

<form>

<div class="form-field">

<au-fa-input icon="envelope">

<input auInput type="email" name="email" placeholder="Email" autocomplete="off">

</au-fa-input>

</div>

<div class="form-field">

<au-fa-input icon="lock">

<input auInput type="password" name="email" placeholder="Password" autocomplete="off">

</au-fa-input>

</div>

<div class="form-field">

<au-fa-input icon="lock">

<input auInput type="password" name="email" placeholder="Confirm Password" autocomplete="off">

</au-fa-input>

</div>

</form>

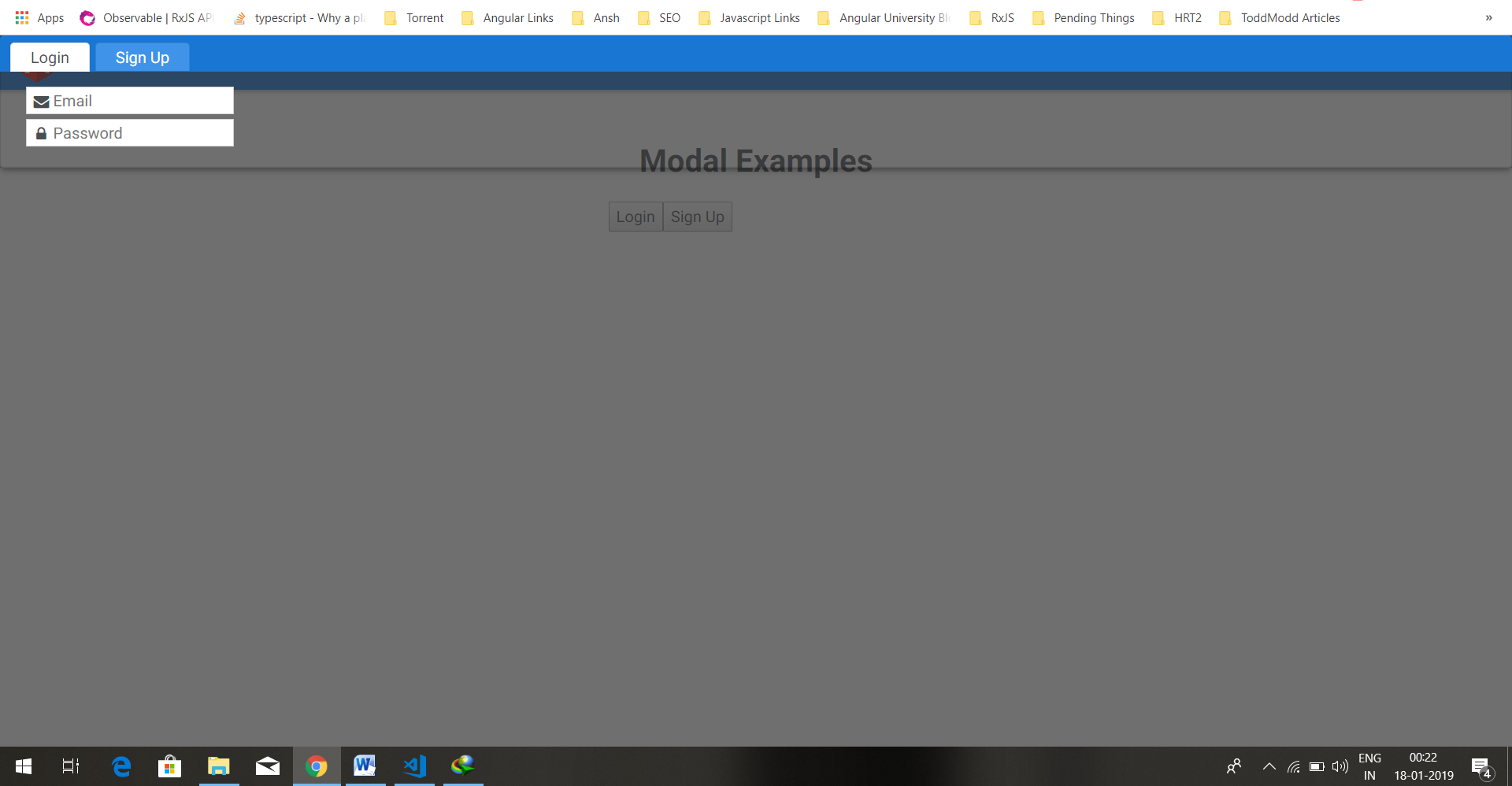
</au-tab>

</au-tab-panel>

</div>

</div>

This is what we get-



Then create a new component, au-modal.

With this we have all html and css we have . next we will work on typescript part of our component.

48. Implementing a Modal Component - The First Version Up And Running

We use content projection to have our first version of modal component up and running.

Au-modal.component.html-

<div class="modal-overlay">

<div class="modal-body">

<ng-content></ng-content>

</div>

</div>

Au-modal.component.scss-

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

.modal-body {

position: absolute;

top: 50%;

left: 50%;

transform: translate(-50%, -50%);

z-index: 20;

$border: 1px solid $secondary;

border:$border;

border-radius: 4px;

background: white;

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);

}

.modal-overlay {

display: inline-block;

position: fixed;

width: 100%;

height: 100%;

left: 0;

top: 0;

background: rgba(51,51,51,0.7);

z-index: 10;

}

We used transform property in modal-body class to center the modal. In app.component.html-

We place the au-tabel component into au-modal component.

<au-modal>

<au-tab-panel>

<au-tab title="Login">

<form>

<div class="form-field">

<au-fa-input id="email-field" icon="envelope">

<input auInput type="email" name="email" placeholder="Email" autocomplete="off">

</au-fa-input>

</div>

<div class="form-field">

<au-fa-input icon="lock">

<input auInput type="password" name="email" placeholder="Password" autocomplete="off">

</au-fa-input>

</div>

</form>

</au-tab>

<au-tab title="Sign Up">

<form>

<div class="form-field">

<au-fa-input icon="envelope">

<input auInput type="email" name="email" placeholder="Email" autocomplete="off">

</au-fa-input>

</div>

<div class="form-field">

<au-fa-input icon="lock">

<input auInput type="password" name="email" placeholder="Password" autocomplete="off">

</au-fa-input>

</div>

<div class="form-field">

<au-fa-input icon="lock">

<input auInput type="password" name="email" placeholder="Confirm Password" autocomplete="off">

</au-fa-input>

</div>

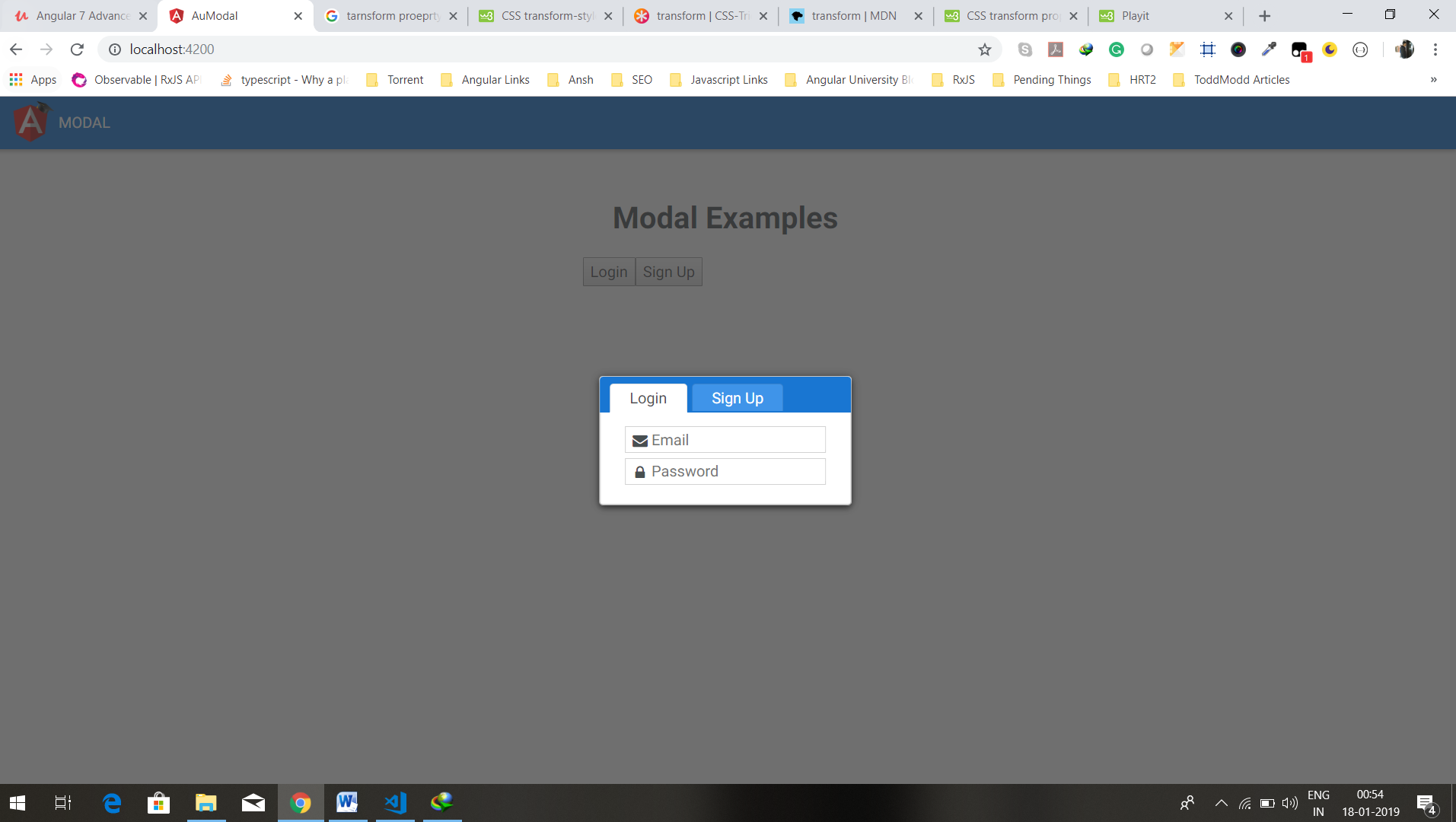
</form>

</au-tab>

</au-tab-panel>

</au-modal>

This is what we get-



Lets now improve our implementation and support also input templates. Once we have that in place we are going to introduce structural directives.

49. Configurable Angular Components - Content Projection and Input Templates

Here we are checking that whether we are passing templte to our component through property binding if yes, we display that component in body in body of our component. if not then we use content projection. Au-modal.component.html-

<div class="modal-overlay">

<div class="modal-body">

<ng-container \*ngIf="body else projectedContent">

<ng-container \*ngTemplateOutlet="body"></ng-container>

</ng-container>

<ng-template #projectedContent>

<ng-content></ng-content>

</ng-template>

</div>

</div>

Au-modal.component.html-

@Input() body: TemplateRef<any>;

App.component.html-

<ng-template #authModalBody>

au-tab-panel component goes here

</ng-template>

<au-modal class="auth-modal" [body]="authModalBody">

</au-modal>

Here is html of au-tab-panel-

<au-tab-panel>

<au-tab title="Login">

<form>

<div class="form-field">

<au-fa-input id="email-field" icon="envelope">

<input auInput type="email" name="email" placeholder="Email" autocomplete="off">

</au-fa-input>

</div>

<div class="form-field">

<au-fa-input icon="lock">

<input auInput type="password" name="email" placeholder="Password" autocomplete="off">

</au-fa-input>

</div>

</form>

</au-tab>

<au-tab title="Sign Up">

<form>

<div class="form-field">

<au-fa-input icon="envelope">

<input auInput type="email" name="email" placeholder="Email" autocomplete="off">

</au-fa-input>

</div>

<div class="form-field">

<au-fa-input icon="lock">

<input auInput type="password" name="email" placeholder="Password" autocomplete="off">

</au-fa-input>

</div>

<div class="form-field">

<au-fa-input icon="lock">

<input auInput type="password" name="email" placeholder="Confirm Password" autocomplete="off">

</au-fa-input>

</div>

</form>

</au-tab>

</au-tab-panel>

So now we have 2 ways of passing modal body to our component.

50. Structural Directives - How Do They Work ? Understanding Their Syntax

We are going to implement structure directive as a part of our modal. Lets start implementing now the functionality for opening and closing the modal in response to different events. This could be clicking over button or menu item or this could be hitting escape key if modal is open or clicking elsewhere on page outside the modal. We will be providing the auxiliary structural directive to support those very common use cases.

But note that if have some condition which is not covered by directive we can use \*ngIf in app.component.html to open and close the modal depending upon some condition.

So we want to build directive like this-

<au-modal

class="auth-modal"

[body]="authModalBody"

\*auModalOpenOnClick="[loginButton,signUpButton]">

</au-modal>

<div class="modal-buttons">

<button #loginButton>Login</button>

<button #signUpButton>Sign Up</button>

</div>

To auModalOpenOnClick we pass local refrence. If we want to pass more than one local refrence than we can pass array. By clicking on these local refrences, modal will be opened.

51. Step-By-Step Implementation of a Structural Directive - Learn ViewContainerRef

Here we created a structural directive. We know about desugaring mechanism. Refer to max notes on structural diectives. Here are some diffrences-

We inject 2 things in our directives, that all structural directives need. First one is refrence to the template itself. We know \* will converted into ng-template thing. So templateRef is handle to that refrence. So we can use it to instantiate the component that is inside it. Then we inject another thing called vieContainerRef. View container represents a container where one or more views can be attached.

We have various methods on viewContainerRef. One of these methods is createComponent. It creates a component and attach it to this view container.

Another method is createEmbededView,it takes templateRef as argument. it creates a embedded view and attaches it to element on which directive is placed.

Like this-

export class AuModalOpenOnClickDirective {

constructor(

private templateRef: TemplateRef<any>,

private viewContainer: ViewContainerRef) {

viewContainer.createEmbeddedView(templateRef);

}

}

Now lets see how we are going to use it in our template. Our directive will be become like this-

<ng-template [auModalOpenOnClick]="loginButton">

<au-modal class="auth-modal" [body]="authModalBody">

</au-modal>

</ng-template>

loginButton is localrefrence of login button, we want to open modal on click of this button.

Here templateRef that we have injected in our directive is refrence of wrapping ng-template and viewContainer corrosponds to the element on which directive is placed. Question-

<https://www.udemy.com/angular-advanced-masterclass/learn/v4/questions/6166356>

Au-modal-open-on-click.directive.ts-

import { Directive, TemplateRef, ViewContainerRef, Input } from '@angular/core';

@Directive({

selector: '[auModalOpenOnClick]'

})

export class AuModalOpenOnClickDirective {

constructor(

private templateRef: TemplateRef<any>,

private viewContainer: ViewContainerRef) {}

@Input() set auModalOpenOnClick(el: HTMLBaseElement) {

el.addEventListener('click', () => {

this.viewContainer.clear();

this.viewContainer.createEmbeddedView(this.templateRef);

});

}

}

We receive value from app.componnt.html using @Input , We define typescript setter that has same name as input property.

Value that we pass from app.component.html is local refrence of login button, now because there is no directive applied here to this button element. loginButton will actually be a HTML refrence to login button so it will be refrence to a DOM element. So what we are receiving in directive is DOM element directly . it is HTMLBaseElement. We define this type in directive. So this element does not have to be html button but then it could be any clickable element on page.

On click first we cclear anything which could be instantiated in view container, we clear it. Then we create a embeddedView. We instantiate the templte onto which this directive is applied.

Question-

<https://www.udemy.com/angular-advanced-masterclass/learn/v4/questions/6166520>

We use local refrence to add event listener to elements. Then on click listener we display our elements. We use setter so that we can execute this function whenever local refrence changes.

Now when we click on button, we can see that modal is displayed. This is how we can use it in app.component.html-

<ng-template [auModalOpenOnClick]="loginButton">

<au-modal class="auth-modal" [body]="authModalBody">

</au-modal>

</ng-template>

<div class="modal-buttons">

<button #loginButton>Login</button>

<button #signUpButton>Sign Up</button>

</div>

Now one thing is that we are attaching the event listener but we are not removing them. Now because whole element will be removed from the page, in principle this will not cause the problem but it’s always a good idea not to rely on implicit removal of any listener that we add. Later in this course we are going to refactor theses in order to remove these listerners implicitly. Right now lets concentrate on what we are building. This is open and close functionality of modal.

Lets add possibility of receiving multiple local refrences. This is how we do it-

import { Directive, TemplateRef, ViewContainerRef, Input } from '@angular/core';

@Directive({

selector: '[auModalOpenOnClick]'

})

export class AuModalOpenOnClickDirective {

constructor(

private templateRef: TemplateRef<any>,

private viewContainer: ViewContainerRef) {}

@Input() set auModalOpenOnClick(els) {

let elements: HTMLBaseElement[];

if (els.length) {

elements = els;

} else {

elements = [els];

}

elements.forEach(el => {

el.addEventListener('click', () => {

this.viewContainer.clear();

this.viewContainer.createEmbeddedView(this.templateRef);

});

});

}

}

Now we can pass refrence of buttons like this-

<ng-template [auModalOpenOnClick]="[loginButton,signUpButton]">

<au-modal class="auth-modal" [body]="authModalBody">

</au-modal>

</ng-template>

<div class="modal-buttons">

<button #loginButton>Login</button>

<button #signUpButton>Sign Up</button>

</div>

</div>

Now we want to pass parameter to control, whoch tab is opened when modal is displayed. We will do that in next lecture. But now have you noticed one thing? How do we implement the functionality for closing the model. Because the directive that controls if modal is displayed does not have information about if there was click outside the modal. This information is avalible at the level of modal component which is completely separate from open and close directive. So lets see how we can make 2 diretives communicate with each other in a transparent way.

52. Running Into A Design Issue While Implementing The Modal Close Functionality

Here we are going to implement the functionality of closing the modal. It seems simple task, but we are going to run into a major component directive design issue while implementing it. We are going to have design discussion about that in this lesson.

We want if we click outside the modal(that is overlay mask), we want our modal to close.

In au-component.html, we call a method, **closeModal** on click-

<div class="modal-overlay" (click)="closeModal()">

<div class="modal-body">

<ng-container \*ngIf="body else projectedContent">

<ng-container \*ngTemplateOutlet="body"></ng-container>

</ng-container>

<ng-template #projectedContent>

<ng-content></ng-content>

</ng-template>

</div>

</div>

This function is at level of au-modal.component.ts, we want this function to trigger something in our structural directive, which is separate directive and there is no direct way for these do communicate or is it there?

If we see app.component.html, -

<ng-template [auModalOpenOnClick]="[loginButton,signUpButton]">

<au-modal class="auth-modal" [body]="authModalBody">

</au-modal>

</ng-template>

As you can see that modal component is in content part(content projection) of ng-template and structural directive by definition is applied at level of element(here we are treating ng-template as a element). So lets try to inject the component at level of directive by using the ContentChild. Here we assume that ng-content is our element, and we are using content projection to have lets see if this is working.

Au-modal-open-on-click.directive.ts-

@ContentChild(AuModalComponent) modal: AuModalComponent;

ngAfterContentInit() {

console.log(this.modal);

}

Here we try to get directive refrence and then we print it. However we get undefined printed on console. So this is not working. there can be muptiple explanations why this is not working. Best explanations is it means this is simply not meant to be used for this use case and in general ,we have no guarantee whenever we have multiple related directives on a page we have no guarantee that the directive auModalOpenOnClick will always be applied in the parent element of another directive that we want to interact with. So the directive could be placed at multiple places on the page and we would still like to have some interaction between the directives without ensuring certain nesting structure of those diretives. So let’s see how we can get generic mechanism for directive interaction.

Before that lets use \*syntax for structural directives. In app.component.html-

<!-- <ng-template [auModalOpenOnClick]="[loginButton,signUpButton]">

<au-modal class="auth-modal" [body]="authModalBody">

</au-modal>

</ng-template> -->

<ng-template >

<au-modal class="auth-modal" [body]="authModalBody" \*auModalOpenOnClick="[loginButton,signUpButton]">

</au-modal>

</ng-template>

So let’s see how we can get generic mechanism for directive interaction.

53. Implementing A General Communication Mechanism For Directive Interaction

Lets implement mechanism for component and directive interaction. We are going to use most generic mechanism possible which is shared service. So crate a service ModalService in au-modal folder. now lets provision it in our dependency injection system because, this is not done by angular-cli because it can be done in number of ways.

We go to au-modal module, we are going to define service here at level of au-module but we don’t want to unnecessarily add this service here to providers property of this module, because that could cause problems with applications that lazy load this module. So we want service to be defined globally at the level of the root application module as a application wide singleton. This assume ofcourse that we are displaying one dialog at a time which is reasonable assumption for this type of components.

So instead of adding service in providing array of AuModalModule, we create a static method. This method by convention is called **forRoot**. This means this is static initializer that we should be using in the application module. So lets do it-

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

FormsModule,

HttpModule,

AuInputModule,

AuTabPanelModule,

// AuModalModule,

AuModalModule.forRoot()

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

In app.module instead of importing AuModalModule, we call forRoot instead.

Now in AuModalModule, this forRoot method needs to returns a type which is ModuleWithProviders, this is plain jaavscript object with couple of properties. First propery is **ngModule**, ngModule is going to be our modal module. Then there is property which is providers of this module. So here at level of these providers we want to pass on our service.

Au-modal.module.ts-

import {ModuleWithProviders, NgModule} from '@angular/core';

import { CommonModule } from '@angular/common';

import { AuModalComponent } from './au-modal.component';

import { AuModalOpenOnClickDirective } from './au-modal-open-on-click.directive';

import { AUModalService } from './modal.service';

@NgModule({

declarations: [AuModalComponent, AuModalOpenOnClickDirective],

imports: [

CommonModule

],

exports: [AuModalComponent, AuModalOpenOnClickDirective]

})

export class AuModalModule {

static forRoot(): ModuleWithProviders {

return {

ngModule: AuModalModule,

providers: [AUModalService]

}

}

}

So right now whenever this service is being used by both root module, so hthis application does not have lazy loaded modules but it could have in large applications. In that case forRoot is going to prevent use from running into situtaions where there are multiple instances of this service at same time on same application which is something that we want to avoid. So lets see how this service can help us to solve problem.

Moda.service..ts-

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

import { Observable, Subject } from 'rxjs';

// @Injectable({

// providedIn: 'root'

// })

export class AUModalService {

private subject = new Subject();

close$: Observable<any> = this.subject.asObservable();

constructor() { }

close() {

this.subject.next();

}

}

So we have subject from there we derive a observable.so now we have public observable that we can consume and we have private subject that we can use to emit events. So its important to keep the subject private to the service so that we don’t expose the ability of emitting events to outside users .then we inject this service in both au-modal.component and our directive. The we communicate via observable.

Au-modal.component.ts-

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

import { AUModalService } from './modal.service';

@Component({

selector: 'au-modal',

templateUrl: './au-modal.component.html',

styleUrls: ['./au-modal.component.scss']

})

export class AuModalComponent implements OnInit {

@Input() body: TemplateRef<any>;

constructor(private modalService: AUModalService) { }

ngOnInit() {

}

closeModal() {

this.modalService.close();

}

cancelClick(event: KeyboardEvent) {

// event.preventDefault();

event.stopPropagation();

}

}

In au-modal.component we call the the close method of service. we can directy use subject to emit value, in component file also. to keep the internal details of service private, here we call only function of service.

Au-modal.component.html-

<div class="modal-overlay" (click)="closeModal()">

<div class="modal-body" (click)="cancelClick($event)">

<ng-container \*ngIf="body else projectedContent">

<ng-container \*ngTemplateOutlet="body"></ng-container>

</ng-container>

<ng-template #projectedContent>

<ng-content></ng-content>

</ng-template>

</div>

</div>

Here we use stopPropgation to prevent event from reaching div with class modal-overlay. This is because we do not want to close the modal when we click inside the modal.

Then in our directive we subscribe to this observable, and we clear the viewContainer-

import { Directive, TemplateRef, ViewContainerRef, Input, ContentChild, AfterContentInit, OnInit } from '@angular/core';

import { AuModalComponent } from './au-modal.component';

import { AUModalService } from './modal.service';

@Directive({

selector: '[auModalOpenOnClick]'

})

export class AuModalOpenOnClickDirective implements AfterContentInit, OnInit {

@ContentChild(AuModalComponent) modal: AuModalComponent;

constructor(

private templateRef: TemplateRef<any>,

private viewContainer: ViewContainerRef,

private modalService: AUModalService) {}

ngOnInit() {

this.modalService.close$.subscribe(() => this.viewContainer.clear());

}

ngAfterContentInit() {

console.log(this.modal);

}

@Input() set auModalOpenOnClick(els) {

let elements: HTMLBaseElement[];

if (els.length) {

elements = els;

} else {

elements = [els];

}

elements.forEach(el => {

el.addEventListener('click', () => {

this.viewContainer.clear();

this.viewContainer.createEmbeddedView(this.templateRef);

});

});

}

}

Lets now add possibility of closing it by escape key. Lets improve the api of modal and lets introduce a new concept multi slot content projection. We are going to see how to pass parameter to modal as well and also we will look how to prevent memory leak by unregistering the event listener that we have registered so far.

54. Angular Global Keyboard Handling With EventManager

We want to close modal when we press esc. lets start by having some global keyboarad handling behaviour for escape key. So lets see how can we implement that . in angular in order to handle global keyboard events, we have to inject a dependency **EventManager**, this is built in angular injectable that is specific to browser platform module. This is global event manager. We import it like this-

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

then we use it like this in ngOnit-

constructor(private modalService: AUModalService,

private eventManager: EventManager) { }

ngOnInit() {

this.eventManager.addGlobalEventListener('window', 'keyup.esc', () => {

this.closeModal();

});

}

Here we register a event. On document object. regarding using keyup.esc, read this article-

<https://alligator.io/angular/binding-keyup-keydown-events/>

so on esc we are closing the modal. Now we want to make this thing configurable i.e to close modal when we press esc key. also we want to make this thing configurable, close modal when we click outside the modal body. This is how we do this, we add 2 new properties to our modal api.

Au-modal.componen.ts-

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

import { AUModalService } from './modal.service';

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

@Component({

selector: 'au-modal',

templateUrl: './au-modal.component.html',

styleUrls: ['./au-modal.component.scss']

})

export class AuModalComponent implements OnInit {

@Input() body: TemplateRef<any>;

@Input() hideOnEsc = true;

@Input() hideOnClickOutside = true;

constructor(private modalService: AUModalService,

private eventManager: EventManager) { }

ngOnInit() {

this.eventManager.addGlobalEventListener('window', 'keyup.esc', () => {

if (this.hideOnEsc) {

this.closeModal();

}

});

}

onClickOutsideModal() {

if (this.hideOnClickOutside) {

this.closeModal();

}

}

closeModal() {

this.modalService.close();

}

cancelClick(event: KeyboardEvent) {

// event.preventDefault();

event.stopPropagation();

}

}

Here we set default values of these 2 inputs to true, because we want this as default behaviour

Au-modal.component.html-

<div class="modal-overlay" (click)="onClickOutsideModal()">

<div class="modal-body" (click)="cancelClick($event)">

<ng-container \*ngIf="body else projectedContent">

<ng-container \*ngTemplateOutlet="body"></ng-container>

</ng-container>

<ng-template #projectedContent>

<ng-content></ng-content>

</ng-template>

</div>

</div>

App.component.html-

<au-modal class="auth-modal" [body]="authModalBody"

\*auModalOpenOnClick="[loginButton,signUpButton]"

[hideOnEsc]="false"

[hideOnClickOutside]="false">

</au-modal>

Lets keep adding new features to our modal. We are going to see how we can provide a close icon on the top right of the modal via multi slot content projection. We will see how we can add parameters to the model. And also we will prevent a few memory leaks that could occur because we were adding event listeners manually.

55. Learn Angular Multi-Slot Content Projection

Now what we want to do is we want to allow user to provide template for closing icon. this template for closing icon is customizable. That is user can provide any template.

Like this-

<au-modal class="auth-modal" [body]="authModalBody"

\*auModalOpenOnClick="[loginButton,signUpButton]"

[hideOnEsc]="false"

[hideOnClickOutside]="false">

<i class="fa fa-times"></i>

</au-modal>

Here i tag is template for closing tag. Then in au-modal.component.html, we display cross icon using ng-content property but we use select property. Here we select only icon tag.

Au-modal.component.ts-

<div class="modal-overlay" (click)="onClickOutsideModal()">

<div class="modal-body" (click)="cancelClick($event)">

<ng-content select="i"></ng-content>

<ng-container \*ngIf="body else projectedContent">

<ng-container \*ngTemplateOutlet="body"></ng-container>

</ng-container>

<ng-template #projectedContent>

<ng-content></ng-content>

</ng-template>

</div>

</div>

Now with -

<ng-content select="i"></ng-content>

We select only icon tags which are present in content part of modal. So in our case , this corresponds to only this icon tag-

<i class="fa fa-times"></i>

And this new ng-content tag with select attribute will not interfere with use of ng-content later in template. So the way this works is ng-content without selector will select everything that is not selected by any of selectors.

We can have many ng-conten with different selectors, we can select based on attribute, based on css class. But here we are selecting based on html tag.

Now see video, now we passed the tab-panel via content projection and it worked fine. Initially we passed it via body property in to modal component. see video from 02:38. This was don to show that next ng-content selects the html that is not selected by ng-content tags with select attribute.

Then we add some styling to our close icon. Au-modal.component.scss-

:host .close-icon {

width: 100%;

height: 20px;

}

:host .close-icon ::ng-deep i {

cursor: pointer;

position: fixed;

right: 5px;

top: 5px;

height: 15px;

}

Now we want to close the modal, when we click on close icon. But we have icon in app.compoent.html and we want to call method in au-modal.component.ts for this we can use local refrence.

App.component.html-

<au-modal class="auth-modal" [body]="authModalBody"

\*auModalOpenOnClick="[loginButton,signUpButton]"

[hideOnEsc]="true"

[hideOnClickOutside]="true" #modal>

<i class="fa fa-times" (click)="modal.closeModal()"></i>

</au-modal>

Now we will see how we can pass some inout prameters to our modal, so that we can define which tab will be selected depending upon which button was clicked.

56. Completing the Authentication Dialog - Learn How To Use ng-template Inputs

Lets continue improving the implementation of our modal, lets have here possibility of passing parameters to the modal, in case we are using an input template.

Now we want to make tabs active depending upon which button you clicked, so this means that we need to find a way of passing a parameter to the model. So if we would be using here content projection instead of template that would be really very simple(we are talking about how we are passing au-tab-panel to au-modal.

Lets see how we could have done it(here we are still using template but this pparoch will work in case of content projection also).

Au-modal.component.html-

<ng-template #authModalBody>

<au-tab-panel>

<au-tab title="Login" [selected]="loginActive">

<form>

<div class="form-field">

<au-fa-input id="email-field" icon="envelope">

<input auInput type="email" name="email" placeholder="Email" autocomplete="off">

</au-fa-input>

</div>

<div class="form-field">

<au-fa-input icon="lock">

<input auInput type="password" name="email" placeholder="Password" autocomplete="off">

</au-fa-input>

</div>

</form>

</au-tab>

<au-tab title="Sign Up" [selected]="!loginActive">

<form>

<div class="form-field">

<au-fa-input icon="envelope">

<input auInput type="email" name="email" placeholder="Email" autocomplete="off">

</au-fa-input>

</div>

<div class="form-field">

<au-fa-input icon="lock">

<input auInput type="password" name="email" placeholder="Password" autocomplete="off">

</au-fa-input>

</div>

<div class="form-field">

<au-fa-input icon="lock">

<input auInput type="password" name="email" placeholder="Confirm Password" autocomplete="off">

</au-fa-input>

</div>

</form>

</au-tab>

</au-tab-panel>

</ng-template>

Here we are passing selected attribute to au-tabs. We set value of these by click of button.

<div class="modal-buttons">

<button #loginButton (click)="setLoginActive(true)">Login</button>

<button #signUpButton (click)="setLoginActive(false)">Sign Up</button>

</div>

In app.component.ts-

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

@Component({

selector: 'app-root',

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

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

})

export class AppComponent {

loginActive = true;

setLoginActive(loginActive) {

this.loginActive = loginActive;

}

}

Now modal is working a sexpected. As we see really don’t need any other mechanism for passing parameters to content of modal. Because content of modal, this ng-template has implicit access to to anything accessible at level of app.component.so interior of modal can for example call a service that can save data on backend.

now because this is ng-template and we are passing this template to au-modal via body property, it would also be good thing to allow passing parameters to ng-tempate, just in case that for example this ng-template is defined dynamically and to support more use cases. So the way we are going to implement this is we are going to go to au-modal.component.html, here to templateOutlet we pass context object. this context object will contain the parameters of the context. We can use them if required. Then we define context property at level of au-modal.component. now context object can contain multiple parameters like title of modal and any other property like which tab should be active.

Au-modal.compoent.html-<div class="modal-overlay" (click)="onClickOutsideModal()">

<div class="modal-body" (click)="cancelClick($event)">

<div class="close-icon">

<ng-content select="i"></ng-content>

</div>

<ng-container \*ngIf="body else projectedContent">

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

</ng-container>

<ng-template #projectedContent>

<ng-content></ng-content>

</ng-template>

</div>

</div>

Au-modal.component.ts-

@Input() context: any;

Then at level of ng template , we define multiple properties that we need.

App.component.ts-

<ng-template #authModalBody let-title="title" let-loginTabActive="loginTabActive">

<au-tab-panel>

<au-tab title="Login" [selected]="loginTabActive">

<form>

<div class="form-field">

<au-fa-input id="email-field" icon="envelope">

<input auInput type="email" name="email" placeholder="Email" autocomplete="off">

</au-fa-input>

</div>

<div class="form-field">

<au-fa-input icon="lock">

<input auInput type="password" name="email" placeholder="Password" autocomplete="off">

</au-fa-input>

</div>

</form>

</au-tab>

<au-tab title="Sign Up" [selected]="!loginTabActive">

<form>

<div class="form-field">

<au-fa-input icon="envelope">

<input auInput type="email" name="email" placeholder="Email" autocomplete="off">

</au-fa-input>

</div>

<div class="form-field">

<au-fa-input icon="lock">

<input auInput type="password" name="email" placeholder="Password" autocomplete="off">

</au-fa-input>

</div>

<div class="form-field">

<au-fa-input icon="lock">

<input auInput type="password" name="email" placeholder="Confirm Password" autocomplete="off">

</au-fa-input>

</div>

</form>

</au-tab>

</au-tab-panel>

</ng-template>

Then we pass these properties to au-modal component like this-

<au-modal class="auth-modal" [body]="authModalBody"

\*auModalOpenOnClick="[loginButton,signUpButton]"

[hideOnEsc]="true"

[hideOnClickOutside]="true"

[context]="{title: 'Login or Sign Up', loginTabActive: loginActive}"

#modal>

<i class="fa fa-times" (click)="modal.closeModal()"></i>

</au-modal>

Now e have added mechanism of passing input properties to template just in case it is necessary in more advanced use cases. In general as we saw, we can take proeprties directly from app.component(as we did in starting of this lecture) using normal template expressions with this modal is almost ready.

Let’s visit the implementation of open and close structural directive. We are going to remove event listeners that we are adding. We are going to that as a precaution at directive destruction time.

57)Preventing Memory Leaks

In our directive, we are adding the events but we are not removing them.

Au-modal-open-on-click.directive.ts-

import { Directive, TemplateRef, ViewContainerRef, Input, ContentChild, AfterContentInit, OnInit } from '@angular/core';

import { AuModalComponent } from './au-modal.component';

import { AUModalService } from './modal.service';

@Directive({

selector: '[auModalOpenOnClick]'

})

export class AuModalOpenOnClickDirective implements AfterContentInit, OnInit {

@ContentChild(AuModalComponent) modal: AuModalComponent;

constructor(

private templateRef: TemplateRef<any>,

private viewContainer: ViewContainerRef,

private modalService: AUModalService) {}

ngOnInit() {

this.modalService.close$.subscribe(() => this.viewContainer.clear());

}

ngAfterContentInit() {

console.log(this.modal);

}

@Input() set auModalOpenOnClick(els) {

let elements: HTMLBaseElement[];

if (els.length) {

elements = els;

} else {

elements = [els];

}

elements.forEach(el => {

el.addEventListener('click', () => {

this.viewContainer.clear();

this.viewContainer.createEmbeddedView(this.templateRef);

});

});

}

}

In principle this should not cause problem because whenever we hide the modal, this should also me removed(event handlers) because the elements to which we have added event listeners were also removed. But lets show how we would remove listeners manually instead of relying on their implicit removal.

So we add onDestory life cycle hook . this will be called when component on which we have added directive is destroyed. Herew e will remove evnt handlers. Lets see how we can do it, first we have to refactor implementation of setter. We define elments variable at level of component. now we not only need refrence to elements(so that we can call remove listeners on them) but we also need refrence to event handling functions themselves. Now because we are defining these functions as inline we do not have refrence to them. So we refactor this and create function clickHandler. Then we use this function to set event handler.

So with this in place we are no longer creating click handler each time on the fly but we are always passing same refrence to same function. so function is only defined once. Then in ngOndestroy we remove click event. There we use refrence of function.

Au-modal-open-on-click.diretive.ts-

import { Directive, TemplateRef, ViewContainerRef, Input, ContentChild, AfterContentInit, OnInit, OnDestroy } from '@angular/core';

import { AuModalComponent } from './au-modal.component';

import { AUModalService } from './modal.service';

@Directive({

selector: '[auModalOpenOnClick]'

})

export class AuModalOpenOnClickDirective implements AfterContentInit, OnInit, OnDestroy {

elements: HTMLBaseElement[];

@ContentChild(AuModalComponent) modal: AuModalComponent;

constructor(

private templateRef: TemplateRef<any>,

private viewContainer: ViewContainerRef,

private modalService: AUModalService) {}

ngOnInit() {

this.modalService.close$.subscribe(() => this.viewContainer.clear());

}

ngAfterContentInit() {

console.log(this.modal);

}

@Input() set auModalOpenOnClick(els) {

if (els.length) {

this.elements = els;

} else {

this.elements = [els];

}

this.elements.forEach(el => {

el.addEventListener('click', this.clickHandler);

});

}

clickHandler() {

this.viewContainer.clear();

this.viewContainer.createEmbeddedView(this.templateRef);

}

ngOnDestroy() {

this.elements.forEach(el => el.removeEventListener('click', this.clickHandler));

}

}

Lets try to run it. We get this error-

**TypeError: Cannot read property 'clear' of undefined**

Error occurs in first line of clickHandler function. so it measn clickHandler that we are passing or regsietring in input call(in setter), this clickHandler is passing refrence to fucntio in our component. but this keywod is not being correctly populated. This is actually normal and expected because whenever the browser calls this.click handler, this is not pointing to this class. The browser does not have refrence to this class. Here this points to global window obect. This is not what we want. To avoid this we define clickHandler in slightly different way. We define it as member variable and we assign to it an inline function.

import { Directive, TemplateRef, ViewContainerRef, Input, ContentChild, AfterContentInit, OnInit, OnDestroy } from '@angular/core';

import { AuModalComponent } from './au-modal.component';

import { AUModalService } from './modal.service';

@Directive({

selector: '[auModalOpenOnClick]'

})

export class AuModalOpenOnClickDirective implements AfterContentInit, OnInit, OnDestroy {

elements: HTMLBaseElement[];

@ContentChild(AuModalComponent) modal: AuModalComponent;

constructor(

private templateRef: TemplateRef<any>,

private viewContainer: ViewContainerRef,

private modalService: AUModalService) { }

ngOnInit() {

this.modalService.close$.subscribe(() => this.viewContainer.clear());

}

ngAfterContentInit() {

console.log(this.modal);

}

@Input() set auModalOpenOnClick(els) {

if (els.length) {

this.elements = els;

} else {

this.elements = [els];

}

this.elements.forEach(el => {

el.addEventListener('click', this.clickHandler);

});

}

clickHandler = () => {

console.log(this);

console.log('In your fucntion');

this.viewContainer.clear();

this.viewContainer.createEmbeddedView(this.templateRef);

}

ngOnDestroy() {

this.elements.forEach(el => el.removeEventListener('click', this.clickHandler));

}

}

Now everything works fine. But in video vasco, propeosed this solution, he wrapped the body into parenthesis, so effectively creating expression and then he applied bind operator. like this-

clickHandler = (() => {

console.log(this);

console.log('In your fucntion');

this.viewContainer.clear();

this.viewContainer.createEmbeddedView(this.templateRef);

}).bind(this);