**Angular ng-template, ng-container and ngTemplateOutlet - The Complete Guide to Angular Templates**

**1-An Attribute directive changes the appearance or behavior of a DOM element.**

**2-** **The ng-template directive and the related ngTemplateOutlet directive are very powerful Angular features that support a wide variety of advanced use cases.**

**3- These directives are frequently used with ng-container, and because these directives are designed to be used together.**

**Tables of Contents: -**

**1-Introduction to the ng-template directive**

**2-Template Input Variables**

**3-The ng-template directive use with ngIf**

**4-ngIf de-suggared syntax and ng-template**

**5-ng-template template references and the TemplateRef injectable**

**6-Configurable Components with Template Partial @Inputs**

**7-The ng-container directive, when to use it?**

**8-Dynamic Template with the ngTemplateOutlet custom directive**

**9-Template outlet @Input Properties**

**10-Final Combined Example**

**11-Summary and Conclusions**

**Introduction to the ng-template directive**

**1-** **the ng-template directive represents an Angular template: this means that the content of this tag will contain part of a template, that can be then be composed together with other templates in order to form the final component template.**

**2- ng-template under the hood in many of the structural directives that we use all the time: ngIf, ngFor and ngSwitch.**

**Example: -**

**This is normal and it's the expected behavior. This is because with the ng-template tag we are simply defining a template, but we are not using it yet.**

**(by default the ng-template is just template which is not show until you call it as below)**

//this means that if the variable lessons exist then show the inside content div else put the ng-template referenced by reference variable as below

<div \*ngIf="lessons else loading" ></div>

<ng-template #loading>

<button class="tab-button"

(click)="login()">{{loginText}}</button>

<button class="tab-button"

(click)="signUp()">{{signUpText}}</button>

</ng-template>

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

**@Component({**

**selector: 'app-template-comp',**

**templateUrl: './template-comp.component.html',**

**styleUrls: ['./template-comp.component.css']})**

**export class TemplateCompComponent implements OnInit {**

**showTemplate:boolean =true;**

**loginText = 'Login';**

**signUpText = 'Sign Up';**

**lessons1 = ['Lesson 1', 'Lessons 2'];**

**constructor() { }**

**ngOnInit() {}**

**login() {console.log('Login');}**

**signUp() {console.log('Sign Up');}}**

**Multiple Structural Directives**

**1-These directives are all very commonly used, so this means these templates are present everywhere in Angular, either implicitly or explicitly.**

**Example: -**

**The below example shows you that you cannot apply multiple structural directive binding on the same element like you cannot use \*ngIf and \*ngFor on the same div**

<div class="lesson" \*ngIf="lessons" \*ngFor="let lesson of lessons">

<div class="lesson-detail">

{{lesson | json}}</div></div>

**Solution: -**

**1-Solution 01: by using extra div element that contain one of the \*ngIF**

<div \*ngIf="lessons" >

    <div class="lesson" \*ngFor="let lesson of lessons">

        <div class="lesson-detail">

            {{lesson | json}}

        </div>

    </div>

</div>

**2-Solution 02: by using ng-container that represent placeholder to inject the template directly to the page**

<ng-container \*ngIf="lessons">

    <div class="lesson" \*ngFor="let lesson of lessons">

        <div class="lesson-detail">

            {{lesson | json}}

        </div>

    </div>

</ng-container>

**Dynamic Template Creation with the ngTemplateOutlet directive**

**Notes: -**

**1-we can apply add multiple ng-templates on the div by mapping the reference variable with the value of the \*ngTemplateOutlet directive**

<ng-container \*ngTemplateOutlet="loading2" >

<div class="lesson" \*ngFor="let lesson of lessons">

<div class="lesson-detail">{{lesson | json}}</div></div>

</ng-container>

<ng-template #loading>

<button class="tab-button" (click)="login()">{{loginText}}</button>

<button class="tab-button" (click)="signUp()">{{signUpText}}</button>

</ng-template>

<ng-template #loading2>

<button class="tab-button" (click)="login()">{{loginText2}}</button>

<button class="tab-button" (click)="signUp()">{{signUpText2}}</button>

</ng-template>

**Template Context**

**Notes: -**

**1-each template can also define its own set of input variables! Actually, each template has associated a context object containing all the template-specific input variables.**

**Example:-**

<ng-template #estimateTemplate let-lessonsCounter="estimate">

<div> Approximately {{lessonsCounter}} lessons ...</div>

</ng-template>

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

</ng-container>

export class TemplateCompComponent implements OnInit {

totalEstimate = 10;

ctx = {estimate: this.totalEstimate};}

**1-On the below example we assign local-variable called lessonsCounter with prefex command called let-**

**2-we see that we assign Content Content on the ng-container with defined variable that contain property called estimate that match the name on the ng-template to declare**

**Template References**

**1-we can use template references to access to html directive by using reference variable with view child command**

<ng-template #defaultTabButtons>

<button class="tab-button" (click)="login()">

{{loginText}}

</button>

<button class="tab-button" (click)="signUp()">

{{signUpText}}

</button>

</ng-template>

export class TemplateCompComponent implements OnInit {

@ViewChild('defaultTabButtons', {static: true}) private defaultTabButtonsTpl:

TemplateRef<any>;

ngOnInit() {

debugger;

console.log(this.defaultTabButtonsTpl);}}

**Passing ng-template from parent component to child component**

**Steps: -**

**1-on the parent component we set the following code**

<ng-template #customTabButtons>

<div class="custom-class">

<button class="tab-button" (click)="login()">{{loginText}}</button>

<button class="tab-button" (click)="signUp()">{{signUpText}}</button></div>

</ng-template>

//we pass the reference variable to the child component like below

<app-template-comp [headerTemplate]="customTabButtons"></app-template-comp>

export class AppComponent {

title = 'myapp';

loginText:string = "login ";

signUpText:string = "logout";}

**2-on the child component we set the following code**

<ng-template #defaultTabButtons>

<div class="default-tab-buttons">hello every one on the whole world</div>

</ng-template>

//this container will show the ng-template of input variable of default

//ng-template

<ng-container \*ngTemplateOutlet="headerTemplate ? headerTemplate: defaultTabButtons">

</ng-container>

export class TemplateCompComponent implements OnInit {

//this input variable is contain the ng-template

@Input() headerTemplate: TemplateRef<any>;

ngOnInit() {}}

**Conclusions: -**

**1-The core directives ng-container, ng-template and ngTemplateOutlet all combine together to allow us to create highly dynamical and customizable components.**

**2-We can even change completely the look and feel of a component based on input templates, and we can define a template and instantiate on multiple places of the application.**

**Example: -**

**In the below example we pass flag to detect which ng-template we want to preview**

**And we pass ng-template object to child component to another <ng-container>**

**On app.component.html**

<ng-template #comp3>Comp3 is here</ng-template>

//we pass the Boolean flag to show the ng-template we want to show

//and we pass the ng-template from parent component to child component

<app-template-comp [show]="state" [headerTemplate]="comp3"></app-template-comp>

export class AppComponent {

title = 'myapp';

state:boolean = true ;

loginText:string = "login ";

signUpText:string = "logout";}

//on the child component we type the following code

<ng-template #comp1>Comp1 is here</ng-template>

<ng-template #comp2>Comp2 is here</ng-template>

<ng-container \*ngTemplateOutlet="show?comp1:comp2"></ng-container>

<br/>

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

export class TemplateCompComponent implements OnInit {

@Input() show:boolean = true;

@Input() headerTemplate: TemplateRef<any>;

ngOnInit() {}}