**Chapter: 3.** **Templates, Styles & Directives**

## Objective: In this chapter, our objective is to study below topics:

* Template Vs. templateURL
* Applying styles for templates
* Bootstrap Integration with Angular
* Directives Introduction and types
* Built-in Directives
* Custom Directives
* Multiple Choice Questions
* FAQ/Lab assignments
* Summery about the chapter.

# Template vs. Template URL

**template:**

Both of these are part of component directive **template** is used for inline template. To create inline template we use the command like

* In this technique the presentation, logic and styles all are maintained in a single file.
* Component is configured in only one file “.ts”
* In-line technique is good if your component is having a simple functionality.
* To create this by using angular cli command we use like



**Open the** [**home.component.ts**](http://home.component.ts) **file and write the following code**

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

@Component({

    selector: 'app-home',

    template: `

    <h2>Amazon Home</h2>

    <div id="offer">{{msg}}</div>

    <p>This is our first component</p>

    `,

    styles: ['h2{text-align:center;color:darkcyan}', '#offer{border:2px solid darkcyan; border-radius:20px; box-shadow:2px 3px 4px darkcyan}']

})

export class HomeComponent {

    public msg = '02-Aug-2020 to 04-Aug-2020 Monsoon Sale 70% Off';

}

**Go to “app.module.ts” and register your component and set in bootstrap.**

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

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

import { AppRoutingModule } from './app-routing.module';

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

import { HomeComponent } from './home/home.component';

@NgModule({

  declarations: [

    AppComponent,

    HomeComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule

  ],

  providers: [],

  bootstrap: [HomeComponent]

})

export class AppModule { }

**Go to “index.html” use selector to run HomeComponent**

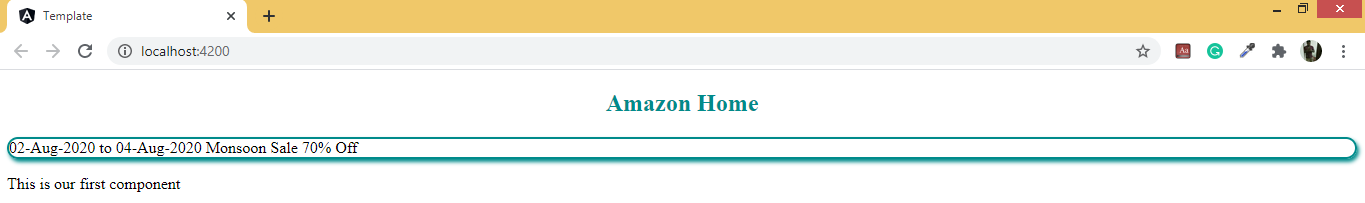
<body>

  <app-home></app-home>

</body>

**Start your project**> ng serve - -open

**Output:**



**templateUrl:**

And, we don’t need to write our template code inline all the time, and it can be clumsy as well, and messy. And we can store our HTML template files in another file, and then merely refer to it in our component with the help of the template URL property.

* In this technique the code, presentation and styles are maintained in separate files.
* It is easy to extend and test.
* If you have regular extensions to add into your functionality then you can go with “Code Behind” technique.
* However using multiple files will increase the load time.

**Applying styles for templates**

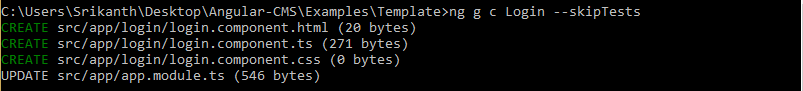
We are also allowed to write any custom CSS style for the component through the styles property. The styles ask for an array of strings and simply like the multi-line strings with the help of the back-ticks.

**StylesUrl**

Now suppose you do not want to write the style inline. There are no worries in that case as well. You can easily write the style in various formats like CSS, scss, etc., which you choose at the time of creating the app in a separate file, and then you can mention the URL of the style file in the stylesurl property in the decorator. And you are done.

**Example:**

1. Create a component by using angular cli command like



1. Add following files  
   - login.component.ts  
   - login.component.html  
   - login.component.css
2. Write the implementation in respective files like below

**login.component.ts:**

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

@Component({

    selector: 'app-login',

    templateUrl: 'login.component.html',

    styleUrls: ['login.component.css']

})

export class LoginComponent {

    public title = 'User Login';

}

**login.component.css:**

.form-login {

    width: 300px;

    margin:auto;

    align-items: center;

    justify-content: center;

    border:2px solid darkcyan;

    box-shadow: 2px 2px 3px darkcyan;

    border-radius: 10px;

}

button {

    background-color: darkcyan;

    color:white;

}

**Login.component.html:**

<div class="form-login">

    <h2>{{title}}</h2>

    <dl>

        <dt>User Name</dt>

        <dd>

            <input type="text">

        </dd>

        <dt>Password</dt>

        <dd>

            <input type="password">

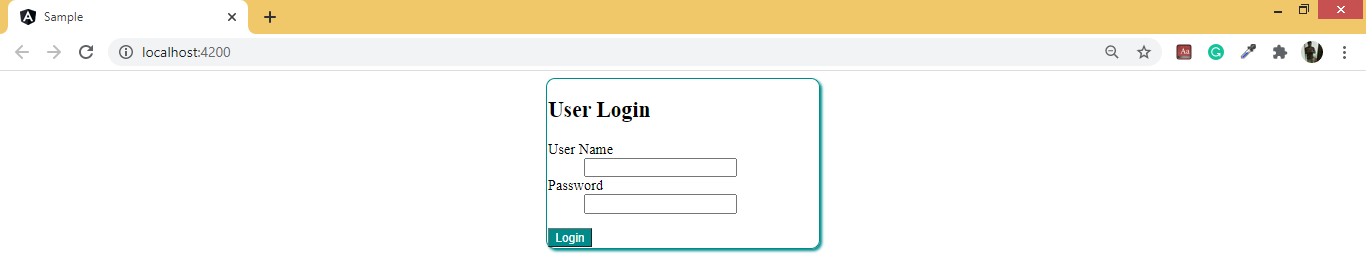
        </dd>

    </dl>

    <button>Login</button>

</div>

1. Bootstrap the **LoginComponent** from **app.module.ts** and use selector in **index.html** will get following output



**Installing and Enabling Bootstrap for Angular Project**

1. Open your workspace location in Terminal
2. Type the following command  
   > **npm install bootstrap**
3. Bootstrap repository is copied into **“node\_modules”**

node\_modules  
|\_bootstrap  
 |\_dist  
 |\_css  
 |\_boostrap.css

1. Make **bootstrap.css** as global for all components.
   1. Go to “**styles.css**”
   2. Import bootstrap css

**@import "../../../node\_modules/bootstrap/dist/css/bootstrap.css";**

**Example with Bootstrap:**

Modify the **login.component.ts** and write the following code by adding bootstrap classes and check the output

**Login.component.html:**

<div class="container-fluid">

    <h2 class="text-center text-primary">{{title}}</h2>

    <div class="form-group">

     <label>User Name</label>

     <div>

         <input type="text" class="form-control">

     </div>

    </div>

    <div class="form-group">

     <label>Password</label>

     <div>

         <input type="password" class="form-control">

     </div>

    </div>

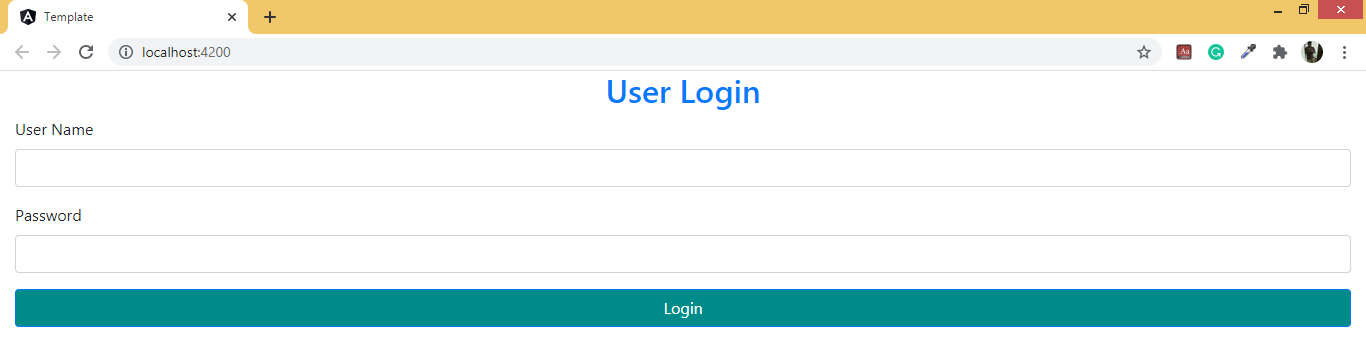
    <div class="form-group">

       <button class="btn btn-primary btn-block">Login</button>

    </div>

   </div>

**Output:**



**Note:** You can also import the css from the bootstrap that is installed through the npm by adding the below code in **angular.json.**

 "styles": ["node\_modules/bootstrap/dist/css/bootstrap.min.css"]

# Directives Introduction and types

Directives are as well the decorators. They are the instructions that bring about a change in the element, and the change can be structural, or in behavior. The **@component** brings about the change in component and it is as well a directive.

**What is an Angular Directive?**

1. If you want the same functionality each of the components, like change background color, you can move forward in two ways.
2. A common method is to explicitly write the code in each of the components for certain behavior. However, this method is complex and lengthy.
3. However, just like a function, you can write a piece of code and you can invoke it anytime and mention the behavior within it. Then, the behavior you mention, you can import this directive, Yes, it will be the directive.

**Why we need a Directive?**

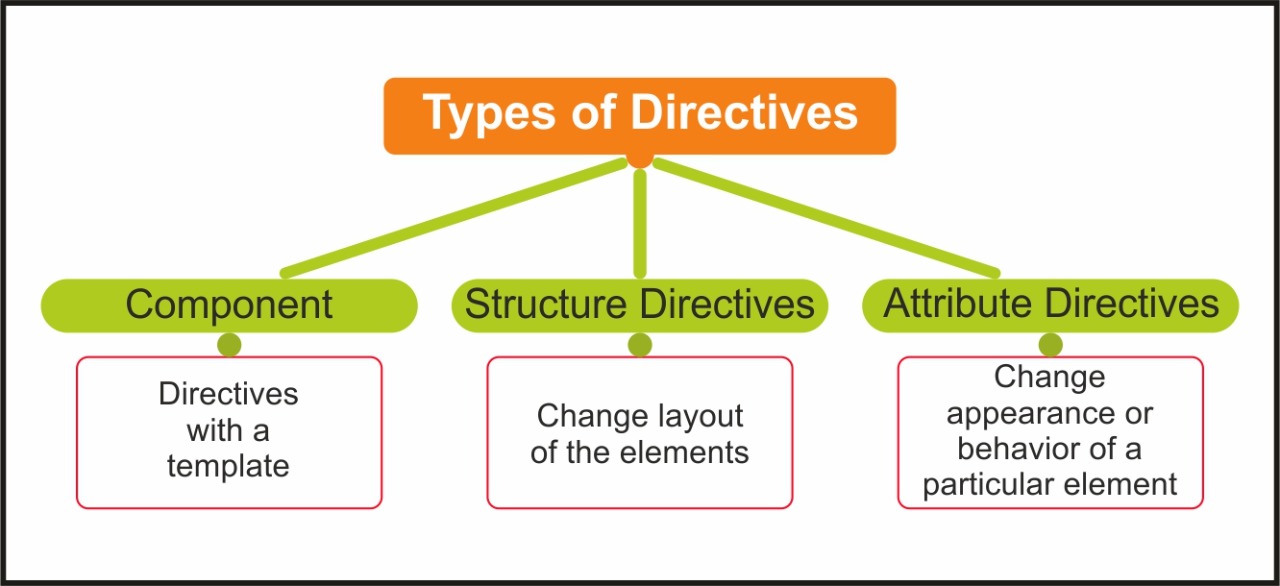
You should know that the Angular directive is a class with the **@Directive decorator**. These decorators are the functions the modify the Typescript classes. They attach to the classes the metadata, and it knows the class configuration and how they will be working.

And as mentioned above the component Is also a directive that is bundled with the template. It is a @Directive decorator that is extended with the template-oriented feature. And as the angular directive renders the directive, that changes the DOM as per the instructions that are given inside the directive. It appears inside the element tag much like the attribute.

**Types of Angular Directives:**

It can be classified into two types:

* attribute directive
* structural directive
* component directives



**Component Directives:**

1. The component directive is the most common directive in Angular.
2. It is a template that comprises logic, presentation and styles.
3. It is used for dynamically rendering HTML and handle interaction with user.

Ex: login.component.ts - logic

login.component.html - presentation

login.component.css - styles

<app-login> </app-login>

- We can access and use built-in components from a library called **"Angular Material"**

## 

## Structural Directives:

1. A structural directive is responsible for changing the DOM structure dynamically.
2. It can add or remove element, It can iterate over element, it can switch between elements etc.
3. Angular structural directives are
4. ngIf
5. ngIf then else

c) ngSwitch

d) ngFor

4. Structural directives are added to **HTML DOM** element by using **"\*"**

<div \*ngIf="">

<li \*ngFor="">

### \*NgIf Directive:

1. It is a structural directive uses to add or remove any DOM element dynamically.
2. It uses a Boolean condition or value to add and remove DOM element.
3. If the expression is true and then the ngIf will be true. And hence that part of DOM as specified by ngIf will be shown or else it will disappear.
4. Through the ngIf directive, we do not hide the element. It adds and then removes them from the DOM. This can be confirmed as well by using the browser developer tool which you can make use of for inspecting the DOM.
5. Syntax: **<tag \*ngIf="Boolean Value/Expression">**

**Example with ngIf Directive:**

Create a component with the name **IfDemo** and implement it like below

**Ifdemo.component.ts:**

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

@Component({

  selector: 'app-ifdemo',

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

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

})

export class IfdemoComponent {

   public product = {

     Name: 'Nike Casuals',

     Price: 4500.55,

     Photo: 'assets/shoe.jpg'

   };

   public showImage = false;

   public btnText = 'Show';

   public TogglePreview() {

      this.showImage = (this.showImage==false)?true:false;

      this.btnText = (this.btnText=='Show')?'Hide':'Show';

   }

}

**Ifdemo.component.html:**

<div class="container-fluid">

    <h2>Product Details</h2>

    <div class="row">

      <div class="col-3">

          <dl>

              <dt>Name</dt>

              <dd>{{product.Name}}</dd>

              <dt>Price</dt>

              <dd>{{product.Price}}</dd>

          </dl>

          <button (click)="TogglePreview()" class="btn btn-primary btn-sm btn-block">{{btnText}} Preview</button>

      </div>

      <div class="col-9">

        <div \*ngIf="showImage">

            <img [src]="product.Photo" height="200" width="200" >

        </div>

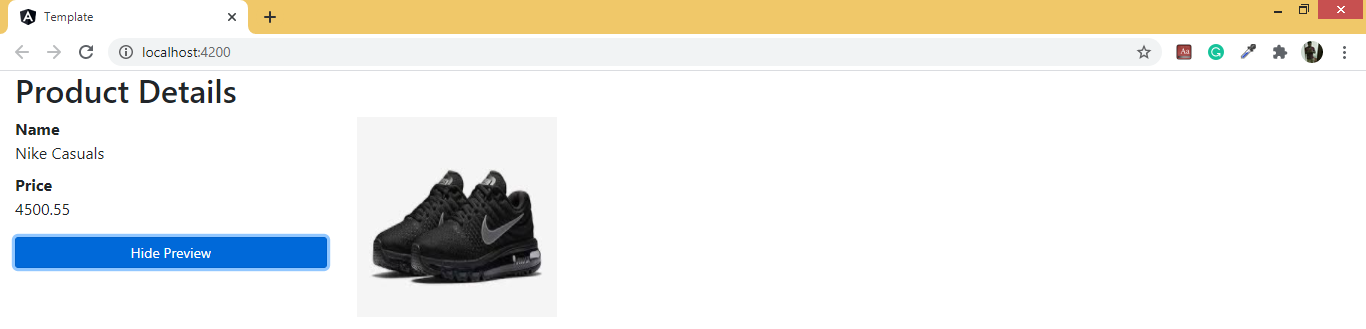
      </div>

    </div>

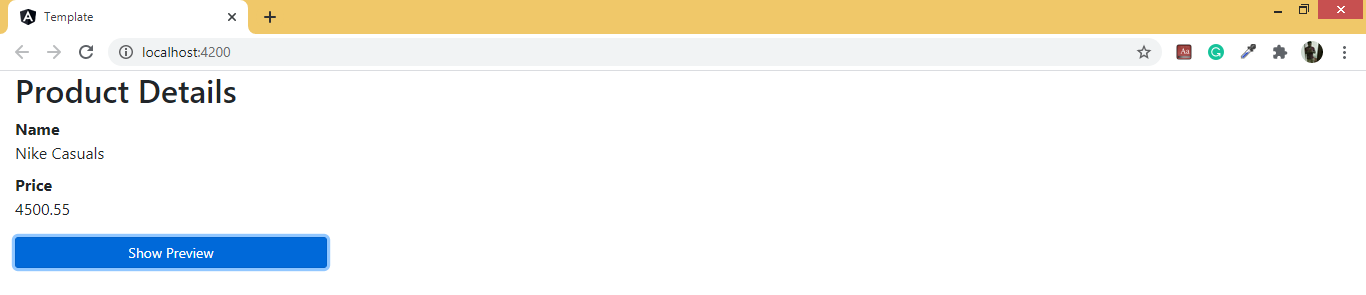
    </div>

Bootstrap the **IfdemoComponent** from **app.module.ts** and use selector in **index.html** will get following output

**Output: If we click on show button will get like**



# Output: If we click on hide button will get like



### \*NgIf else Directive:

1. ngIf with “then and else” block
2. This directive is used to provide the additional functionality for \*ngIf directive like if the condition is true it will render one template else it will render another template.
3. Syntax:

<div \*ngIf=”condition; then thenBlockId else elseBlockId”> </div>

<ng-template #thenBlockId>content to display when true </ng-template>

<ng-template #elseBlockId>content to display when false </ng-template>

1. Angular Containers can handle your interactions dynamically without effecting the DOM structure.
2. Angular provides several containers like
   * <ng-template> </ng-template>
   * <ng-container> </ng-container>

**Example with ngIf then else:**

Modify the **Ifdemo.component.ts** and write the following code:

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

@Component({

  selector: 'app-ifdemo',

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

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

})

export class IfdemoComponent {

   public instock = true;

   public product = {

     Name: 'Nike Casuals',

     Price: 4500.55,

     Photo: 'assets/shoe.jpg'

   };

   public showImage = false;

   public btnText = 'Show';

   public TogglePreview() {

      this.showImage = (this.showImage==false)?true:false;

      this.btnText = (this.btnText=='Show')?'Hide':'Show';

   }

}

**Ifdemo.component.html:**

<div class="container-fluid">

    <h2>Product Details</h2>

    <div class="row">

      <div class="col-3">

          <dl>

              <dt>Name</dt>

              <dd>{{product.Name}}</dd>

              <dt>Price</dt>

              <dd>{{product.Price}}</dd>

          </dl>

          <button (click)="TogglePreview()" class="btn btn-primary btn-sm btn-block">

              {{btnText}} Preview</button>

      </div>

      <div class="col-9">

        <div \*ngIf="showImage; then thenBlock else elseBlock">

        </div>

        <ng-template #thenBlock>

          <img [src]="product.Photo" height="200" width="200">

        </ng-template>

        <ng-template #elseBlock>

          <img alt="Click Preview" height="200" width="200" >

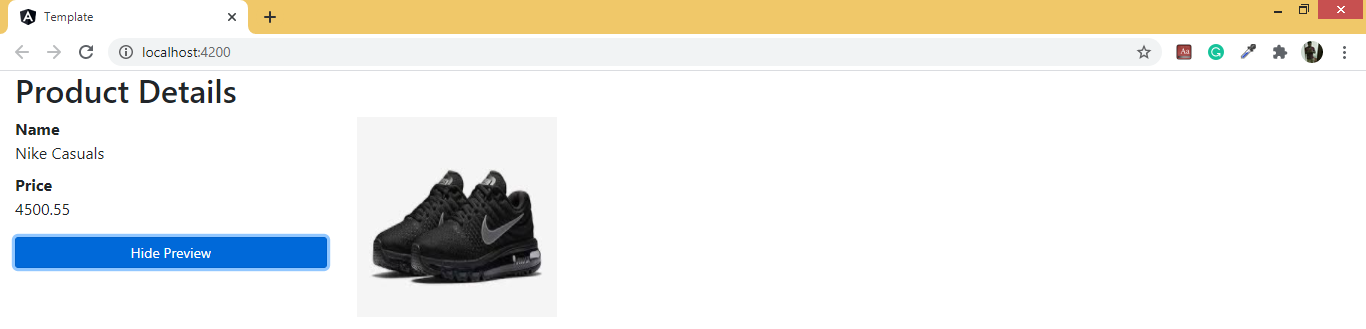
        </ng-template>

      </div>

    </div>

    </div>

**Output:**



### NgSwitch Directive:

1. It is a switch selector in UI.
2. It can display only the container that is required for specific situation.
3. It can allow to change the display dynamically.
4. You can define multiple containers in a page. NgSwitch will select only the container that is required for the situation and removes other containers from DOM.
5. Switch block is defined by using “**ngSwitch**”
6. Case block is defined by using “**ngSwitchCase**”
7. Default block is defined by using “**ngSwitchDefault**”.

**Syntax:**

<main-container [ngSwitch]=”value/expression”>

<child-container \*ngSwitchCase=”1”> </child-container>

<child-container \*ngSwitchCase=”2”> </child-container>

<child-container \*ngSwitchDefault> </child-container>

</main-container>

**Example with ngSwitch Directive:**

Create a component with the name **Switchdemo** and write the following code

**Switchdemo.component.ts:**

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

@Component({

  selector: 'app-switchdemo',

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

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

})

export class SwitchdemoComponent{

   public product = {

     Name: 'Nike Casuals',

     Price: 4500.55,

     Photo: 'assets/shoe.jpg',

     Description: 'Something about Nike Casuals...'

   };

   public selectedView = 'info';

   public views = ['info', 'preview', 'more'];

   public count = 0;

   public ChangeView(obj) {

      this.selectedView = obj.target.name;

   }

   public NextClick(){

     this.count++;

     this.selectedView = this.views[this.count];

   }

   public PrevClick(){

     this.count--;

     this.selectedView = this.views[this.count];

   }

}

**Switch.component.html:**

<div class="container-fluid">

    <h2 class="text-primary text-center"><span class="fa fa-shopping-cart"></span>Amazon Shopping</h2>

    <div class="btn-toolbar bg-danger justify-content-between">

     <div class="btn-group">

       <button (click)="ChangeView($event)" name="info" class="btn btn-danger">Basic Details</button>

       <button (click)="ChangeView($event)" name="preview" class="btn btn-danger">Preview</button>

       <button (click)="ChangeView($event)" name="more" class="btn btn-danger">More..</button>

     </div>

     <div class="btn-group">

       <button (click)="PrevClick()" class="btn btn-danger"><span class="fa fa-chevron-circle-left"></span></button>

       <button (click)="NextClick()" class="btn btn-danger"><span class="fa fa-chevron-circle-right"></span></button>

     </div>

    </div>

    <div class="row" style="margin: 20px;">

       <!--Main Container-->

     <div [ngSwitch]="selectedView">

       <!--info card-->

       <div class="card" \*ngSwitchCase="'info'">

           <div class="card-header">

               <h2>{{product.Name}}</h2>

           </div>

           <div class="card-body">

               <h4>{{product.Price}}</h4>

           </div>

       </div>

       <!--Preview card-->

       <div class="card" \*ngSwitchCase="'preview'">

           <div class="card-body">

               <img [src]="product.Photo" width="200" height="200" >

           </div>

       </div>

       <!--Description Card-->

       <div class="card" \*ngSwitchCase="'more'">

           <div class="card-header">

               <h3>Description</h3>

           </div>

           <div class="card-body">

               <p>{{product.Description}}</p>

           </div>

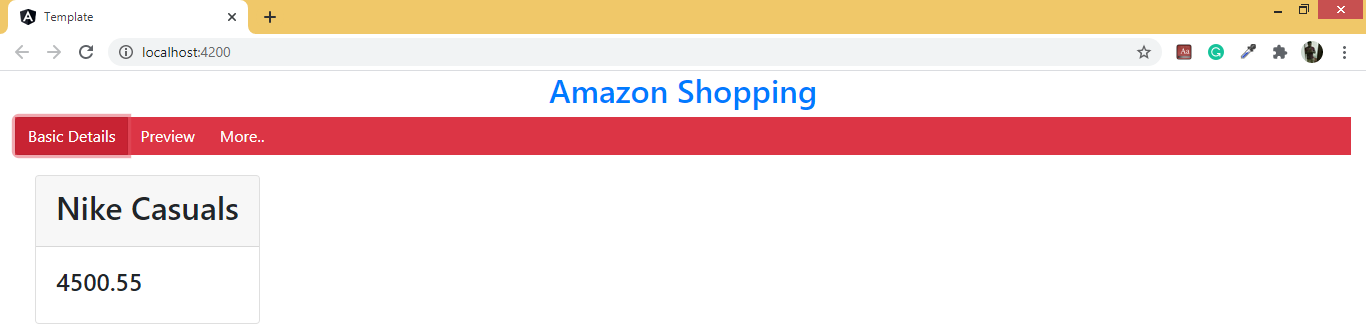
       </div>

     </div>

    </div>

   </div>

Bootstrap the **SwitchComponent** from **app.module.ts** and use selector in **index.html** will get following output



### \*ngFor Directive:

1. It is a repeater.
2. It repeats HTML elements based on an Iterator.
3. It uses “of” operator to read values from a collection and generate an HTML element for every value.

**Syntax:** <tag \*ngFor=”let item of collection”> </tag>

**Example with \*ngFor Directive:**

Create a component with the name **Fordemo** and write the following code

**Fordemo.component.ts:**

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

@Component({

  selector: 'app-fordemo',

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

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

})

export class FordemoComponent {

  public categories = ['Electronics', 'Footwear', 'Fashion'];

  public menudata = [

    {Category: 'Electronics', Products: ['Samsung TV', 'JBL Speaker']},

    {Category: 'Footwear', Products: ['Nike Casuals', 'Lee Cooper Boot']}

  ];

}

**Fordemo.component.html:**

<div class="container-fluid">

    <div class="row">

     <div class="col-3">

       <h3>Categories</h3>

       <ol>

           <li \*ngFor="let item of categories">{{item}}</li>

       </ol>

     </div>

     <div class="col-3">

       <h3>Categories</h3>

       <select class="form-control">

           <option \*ngFor="let item of categories">

               {{item}}

           </option>

       </select>

     </div>

     <div class="col-3">

       <h3>Categories</h3>

       <table class="table table-hover">

         <tbody>

             <tr \*ngFor="let item of categories">

                 <td><a href="#">{{item}}</a></td>

             </tr>

         </tbody>

       </table>

     </div>

    </div>

    <div class="row" style="margin-top: 20px;">

       <div class="col-3">

           <h3>Menu</h3>

           <ol>

               <li \*ngFor="let item of menudata">

                   {{item.Category}}

                   <ol type="a">

                       <li \*ngFor="let product of item.Products">

                           {{product}}

                       </li>

                   </ol>

               </li>

           </ol>

       </div>

       <div>

           <h3>Menu</h3>

           <select class="form-control">

               <optgroup \*ngFor="let item of menudata" label="{{item.Category}}">

                   <option \*ngFor="let product of item.Products">

                       {{product}}

                   </option>

               </optgroup>

           </select>

       </div>

       <div class="col-3">

           <h3>Menu</h3>

           <div \*ngFor="let item of menudata">

               <details>

                   <summary>{{item.Category}}</summary>

                   <ol>

                       <li \*ngFor="let product of item.Products">

                           {{product}}

                       </li>

                   </ol>

               </details>

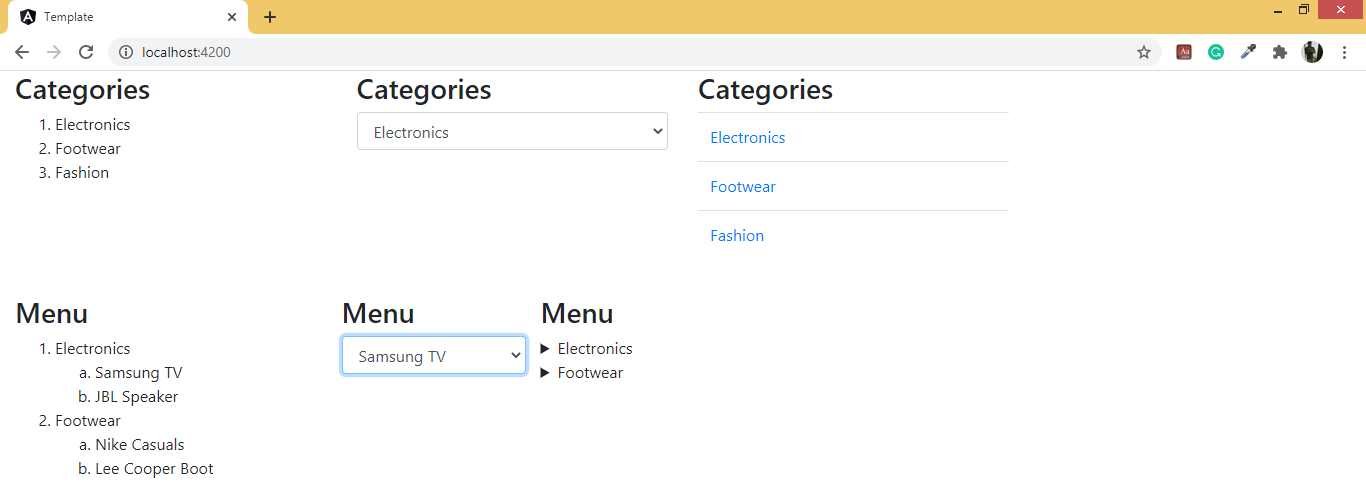
           </div>

       </div>

    </div>

   </div>

Bootstrap the **FordemoComponent** from **app.module.ts** and use selector in **index.html** will get following **output**



### Attribute Directive

Creating dynamic styles in web applications can be a real pain. Luckily with Angular we have multiple ways to handle our dynamic CSS and CSS classes with the new template syntax as well as some built in directives.

### [ngClass]

We can set class in style for a DOM element using [ngClass]. We can use the TypeScript methods, expressions to manipulate each attribute behavior in the class as well. And that gives us an added advantage.

**Syntax:**

You can use TypeScript methods to initialize CSS properties above.

We can also use the Angular property syntax to add CSS classes to elements.

<tag [ngClass]=”{css properties initialized}”></tag>

Alternate Syntax: [class.<class-name>]=”truthy expression”

<div [className]="'Green"> </div>

Now lets have a look at the **ngClass** directive and the options it caters for updating classes on our HTML element and components.

<div [ngClass]="['italic-text', 'maroon']">array of class</div>

<div [ngClass]="'bold-text,’green’'">string of classes</div>

<div [ngClass]="{'largel-text': true, 'blue': true}">object of classes</div>

Like ngStyle ngClass caters various ways for adding and toggling our CSS. And we can bind these classes directly to the component properties for updating them as need and dynamically. In between the new template syntax and a various other directives the angular apps are simpler than even before for styling.

**Example with Class Binding using [ngClass]:**

Create a component with the name **ClassBindingDemo** and write the following code

**Classbindingdemo.component.css:**

.borderEffects {

   border:2px solid darkcyan;

   box-shadow: 2px 3px 4px darkcyan;

}

.backgroundEffects {

    background-color: yellow;

}

.textEffects {

  text-align: center;

  padding: 10px;

}

**Classbindingdemo.component.ts:**

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

@Component({

  selector: 'app-classbinding',

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

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

})

export class ClassbindingComponent {

  public ONTextEffects = false;

  public ONBorderEffects = false;

  public ONBackgroundEffects = false;

}

**Classbindingdemo.component.html:**

<div class="container-fluid">

    <div>

      <div style="margin:10px">

          <ul class="list-unstyled">

              <li><input [(ngModel)]="ONTextEffects" type="checkbox"> Text Effects</li>

              <li><input [(ngModel)]="ONBackgroundEffects" type="checkbox"> Background Effects</li>

              <li><input [(ngModel)]="ONBorderEffects" type="checkbox"> Border Effects </li>

          </ul>

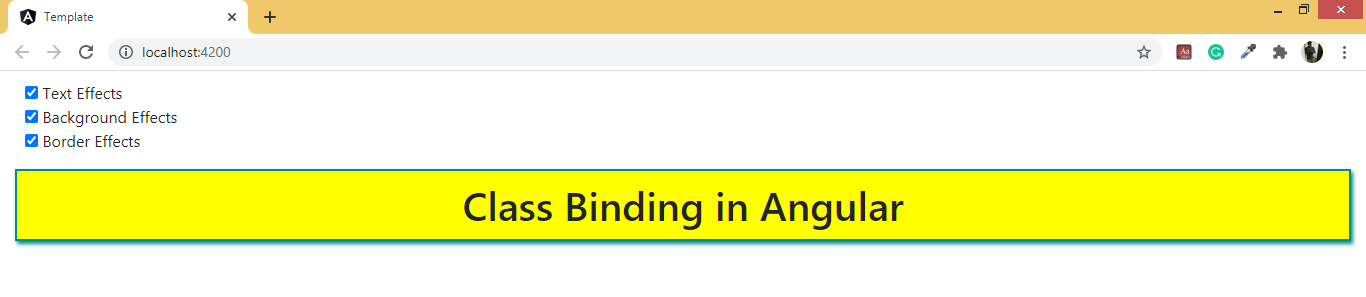
      </div>

      <h1 [ngClass]="{borderEffects:ONBorderEffects, textEffects:ONTextEffects, backgroundEffects:ONBackgroundEffects}">Class Binding in Angular</h1>

    </div>

  </div>

**Output:**



### [ngStyle] Directive:

This helps us set or change the style of a particular element of the DOM element. TypeScript methods can be used in this case as well.

**Syntax:**

Note/: ngStyle works for individual elements and ngClass works for multiple elements of DOM. You can pass the Object literal to each of them.

<tag [ngStyle]=”{css properties with value}>Hello</tag>

First have a look at how we use color property in pure JS,

letdemoDiv = document.getElementById(‘demo-div');

demoDiv.style.color = 'Red'; // we are updating div through its properties.

Now lets look at Angular syntax:

<div [style.color]="'orange'">

style using property syntax, this text is orange

</div>

**Example with Style Binding using [ngStyle]:**

Create a component with the name **Styledemo** and write the following code

**Styledemo.component.ts:**

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

@Component({

  selector: 'app-styledemo',

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

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

})

export class StyledemoComponent {

  public styleObj = {};

  public bgcolor;

  public forecolor;

  public align;

  public ApplyStylesClick() {

     this.styleObj = {

       'background-color': this.bgcolor,

       'color': this.forecolor,

       'text-align': this.align

     };

  }

}

**Styledemo.component.html**

<div class="container-fluid">

    <fieldset>

        <legend>Choose Effects</legend>

        <dl style="width: 300px; align-items: center; margin:auto">

            <dt>Background Color</dt>

            <dd>

                <select [(ngModel)]="bgcolor" class="form-control">

                    <option>Red</option>

                    <option>Yellow</option>

                    <option>Green</option>

                </select>

            </dd>

            <dt>Text Color</dt>

            <dd>

                <select [(ngModel)]="forecolor" class="form-control">

                    <option>White</option>

                    <option>Yellow</option>

                    <option>Red</option>

                </select>

            </dd>

            <dt>Text Align</dt>

            <dd>

                <input [(ngModel)]="align" type="radio" value="left" name="align"> Left

                <input [(ngModel)]="align" type="radio" value="center" name="align"> Center

                <input [(ngModel)]="align" type="radio" value="right" name="align"> Right

            </dd>

            <button (click)="ApplyStylesClick()" class="btn btn-primary btn-block">Apply Effects</button>

        </dl>

    </fieldset>

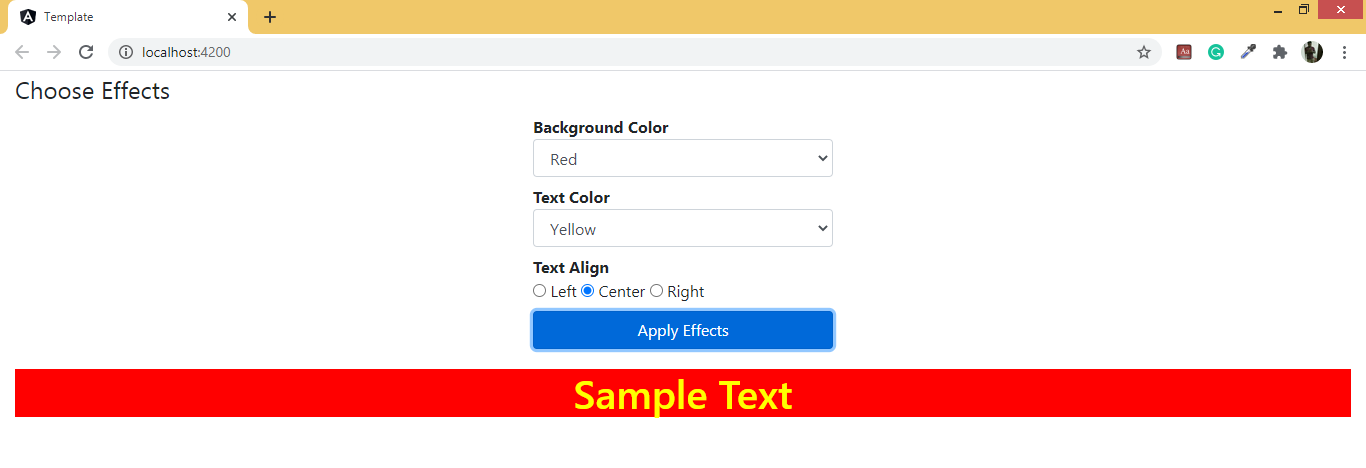
    <div class="form-group" style="margin-top: 20px;">

        <h1 [ngStyle]="styleObj">Sample Text</h1>

    </div>

</div>

**Output:**



## 

## Custom Directive

For making a custom directive, you need to import the directive decorator rather than a component decorator. Also import the Input, TemplateRef, and the ViewContainerRef Symbols which you will require for the structural directive.

Now add to the decorator the directive class.

Now set the CSSattribute selector which finds the directive when it is applied to a certain element within the template. This is the way to start making a directive.

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

@Directive({

selector: '[studentCustom]'

})

export class myStudentDirective {}

The directive selector is the attribute name in the square bracket. It must be spelled in lowerCamelCase and should start with a prefix. You are not required to use ng. It is for Angular. The class name of the directive class finishes with a directive.Notknown

**TemplateRef and ViewContainerRef**

The embedded view from <ng-template> is created by a simple structural directive. This view is inserted in the view container which is just the side of the directive original host element. You can acquire the ng-template content with the help of the TemplateRef and then access the view container with the help of ViewContainerRef. You can inject both of these in the directive constructor like a class private variable.

**Example with Custom Directives:**

**Step-1:** Create a directive with the name **MyDirective** by using a command **“ng g directive My --skipTests”**

**My.directive.ts:**

//import statements

/\* Directive: Used to provide logical name for the custom directive

    Input: Used to pass the dynemic data into the custom directive

    HostListner: Used to apply events

    ElementRef: used to find the elements where we applied the custom directive

\*/

import {Directive,Input,HostListener,ElementRef} from '@angular/core';

@Directive({

    selector:'[myDir]'

})

export class MyDirective{

    @Input() var\_one;

    @Input() var\_two;

    //Inject ElementRef

    constructor(private \_el:ElementRef){

    }

    public myFun(arg1){

        this.\_el.nativeElement.style.backgroundColor = arg1;

    }

    //Add the events by using HostListner

    @HostListener("mouseenter") onmouseenter(){

        this.myFun(this.var\_one);

    }

    @HostListener("mouseleave") onmouseleave(){

        this.myFun(this.var\_two);

    }

}

**Step-2:** Create the Structural Custom Directivewith the name **structural.directive.ts** and write the following code

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

@Directive({

    selector:'[hello]'

})

export class StructuralDirective{

    constructor(private \_templateRef:TemplateRef<any>,

                private \_viewContainerRef:ViewContainerRef){}

    @Input() set hello(arg1:boolean){

        if(arg1){

            this.\_viewContainerRef.createEmbeddedView(this.\_templateRef);

        }

        else{

            this.\_viewContainerRef.clear();

        }

    }

}

**Step-3:** Create a component and use the directive like

**App.component.html:**

<h1 [var\_one]="color\_one.value" [var\_two]="color\_two.value" myDir>Custom Attribute Directive</h1>

<br><br>

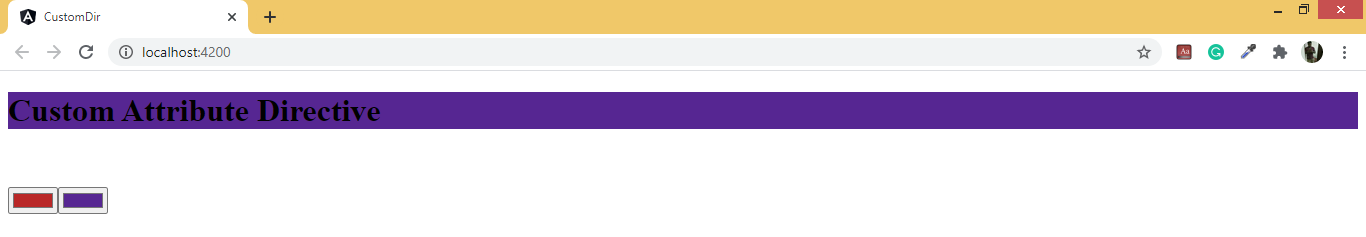
<input type="color" #color\_one>

<input type="color" #color\_two>

<br><br>

<h1 \*hello="false">Custom Structural Directive</h1>

**Step-4:** Register directive and component in **app.module.ts** and use component’s selector in **index.html** and run the application by using **“ng s -o”** will get the following **output** like



# Multiple Choice Questions:

**1. Template can be inline only in component.**

1. It can be explicit as well, a) Yes
2. Its alternate version is TemplateUrl used for explicit declaration.
3. None of these

**2. The output keeps showing appcomponent.html. Why.**

1. You have not declared routes.
2. You have not bootstrapped new component.
3. You have not updated app.module.ts d) None of these

**3. Which is the root component tag from these:**

1. app-root b) app-student c) app d) add-base

**4. Inline declaration of templates is good if:**

1. We need to add only few lines of code.
2. Code length is small. c) Its not possible. d) None of these.

**5. Is component a directive?**

1. Yes it’s the directive with @Component decorator.
2. No c) In some cases d) None of these

**6. What does structural directive does?**

1. It adds or removes certain controls from the DOM.
2. It modifies controls behavior
3. It modifies the css properties of control. d) It adds a button to the DOM.

**7. What does ngIf does**

1. It is the angular version of if. a) It is different from general if.
2. It outputs based on condition as per if fundamentals.
3. None of these

**8. What is ngSwitch?**

1. It is the angular version of switch statement.
2. It is different from general switch statement.
3. It is a structural directive. d) It can be customized.

**9. Attribute directives changes the behavior of the controls.**

1. Yes b) No
2. Sometimes and sometimes they add/remove controls as well
3. No, they add/remove controls

**10. Directive is the fundamental concept of Angular.**

1. Yes b) No
2. Yes, and component is also a directive. d) Yes, but its not fundamental.

**11. How many types of directives are there?**

1. 3 b) 4 c) 2 d) None of these

# FAQS/Lab Assignments

1. Explain briefly template and Template URL.
2. What is View Encapsulation? What are the three types of View Encapsulation? Explain briefly.
3. What is styles URL?
4. What is shadow DOM?
5. How is Bootstrap loaded in Angular?
6. What is built in directive?
7. How many types of built in directives are there? Explain each of them in detail.
8. How can you use the host listener to listen to host event?
9. How can you pass and retrieve argument inside the event handler method?
10. How can you listen to events that originates outside and inside a directive?
11. How can you make use of host binders to bind the host properties?
12. Explain in brief the NgIf, NgFor and NgSwitch.
13. What is structural directive?
14. Explain briefly TemplateRef and ViewContainerRef.

# Summary

We discussed following in this chapter:

1. We discussed template and template url.
2. We discussed styles and various cases.
3. We discussed how we can load bootstrap and various conditions.
4. We discussed directives and types of directives.
5. We discussed attribute directive
6. We discussed structural directives
7. We discussed custom directives.
8. And that concluded this chapter.