**Chapter: 2. Components and Data Binding in Angular**

## Objective:

The main object of this chapter is to clarify the concepts like component and Data binding. The topics will be covering in the process are as below:

* What are Components in Angular
* Programming elements in the component
* How to create a component by using Angular CLI
* Files that are created when component is created using CLI.
* What is Data Binding?
* Types of Data binding
* Component data sharing techniques in Angular
* Multiple Choice Questions
* FAQ/Lab assignments
* Summery about the chapter.

## What are Components in Angular?

The Angular component controls the UI part of the application. As an example, it controls the footer, header, or the whole page, and all of this can be a component. And it depends upon how much reusability, you need in your code.

In technical terms, the Angular component happens to be the Typescript class which is decorated with the **@Component ()** decorator that is part of the Angular core.

There is an associated template of the component that is an HTML file, however, it also contains various special Angular template syntaxes that helps in displaying the data, and it binds the events of the components.

It has one or more associated style sheet files for implementing the styles in the component view. You can make use of the CSS, SAAS, LESS, and SASS.

## Programming elements in the component

### Import statements

Used to import the packages that are need to run the angular application

### Decorator:

Now open the **src/app/app.component.ts** file, and in that file the **AppComponent** class, and we decorate with the **@Component** decorator, which is imported from the @angular/core package that takes certain metadata like:

|  |  |
| --- | --- |
| Selector | This helps you to invoke the component from the HTML template or the file like the standard HTML such as <my-app></my-app> |
| Template | You can write an inline template code here |
| templateUrl | This details the path of the HTML view. |
| styles | This is for inline styling. |
| StyleUrls | This happens to be the array of relative paths where the component needs to look for the style for styling the HTML view. |
| Providers | Through providers, we inject the features to the services, and we have providers in @NgModule, which injects the features to the module. |

1. Export classes: This is where you customize the pipes, directives, services, etc. that the component comprises of. And thus, we are customizing the component to fit our requirements.

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

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'Sample';

}

After creating the component and run that in browser follow the below steps

**Step-1:** Open the **app.module.ts** file and **bootstrap** the component like

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

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

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

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

import { DemoComponent } from './demo/demo.component';

@NgModule({

  declarations: [

    AppComponent,

    DemoComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule

  ],

  providers: [],

**bootstrap: [AppComponent]**

})

export class AppModule { }

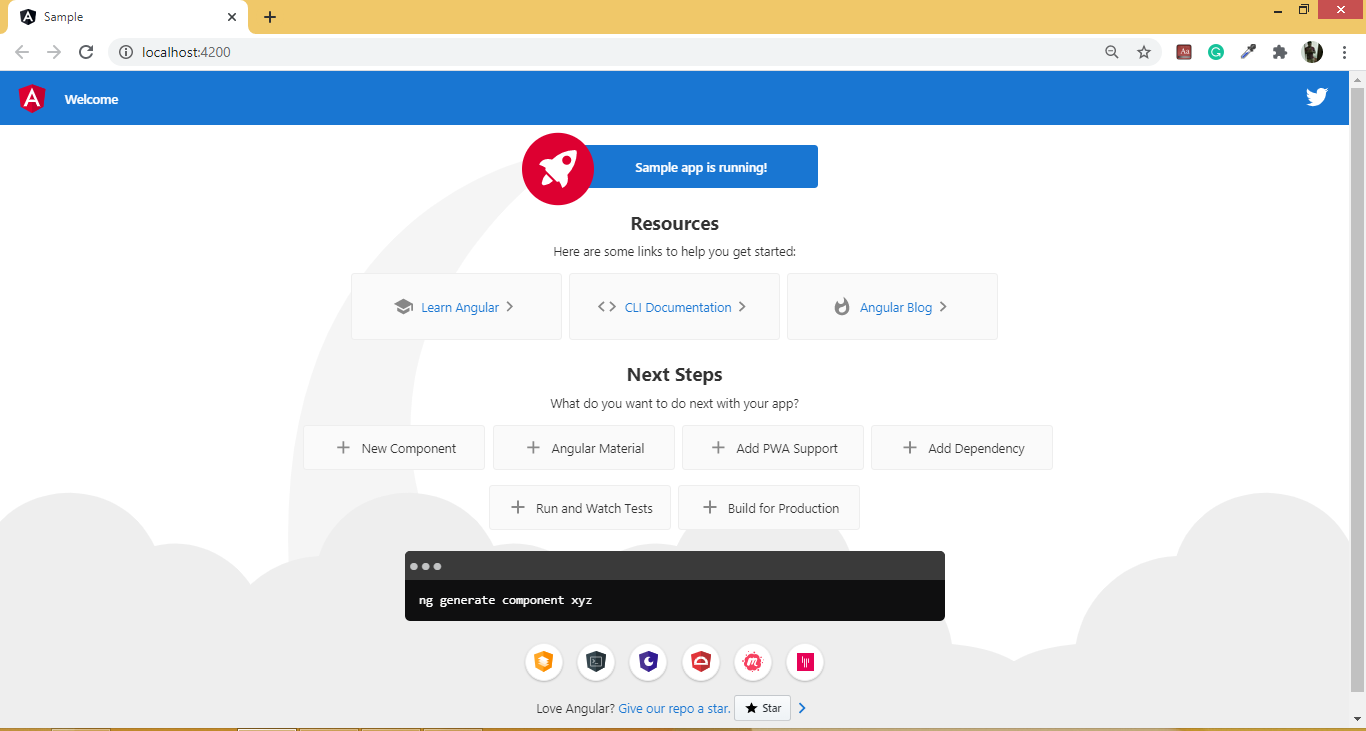
**Step- 2:** To invoke the component we use component’s **selector** in **index.html** like

<body>

  <app-root></app-root>

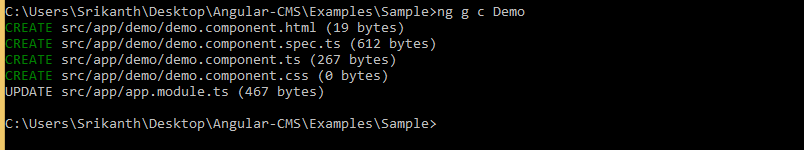
</body>

**Step-3:** Run the application by using a command **“ng s –o”** and check the output in browser



## Creating Components from CLI

The command is as below:

****

## Files that are created when component is created using CLI.

demo.component.css: This has the CSS meant for that particular component.

demo.component.ts: It handles the data as well as the functions.demo.component.html: This determines the view that is being presented to the user.

demo.component.spec.ts: Used to define the specification file mainly used for unit testing

## Example with Login Component:

## Step-1: Create a LoginComponent by using Angular CLI command like above

## Step-2: Open the login.component.ts and write the following code

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

@Component({

    selector: 'app-login',

    templateUrl: 'login.component.html',

    styleUrls: ['login.component.css']

})

export class LoginComponent {

    public title = 'User Login';

}

## Step-3: Open the login.component.html and write the following code

<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>

<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>

## Step-4: Provide the styles for the component’s template by modifying the code of 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;

}

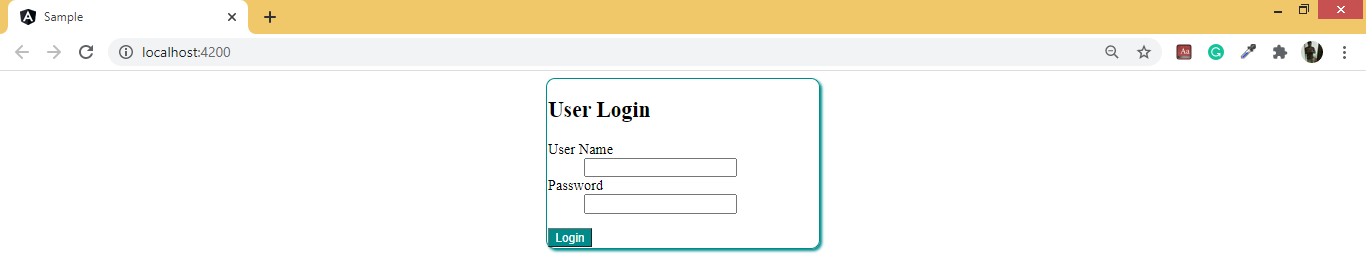
.button {

  background-color: darkcyan;

  color:white;

}

## Step-5: Modify the AppModule and index.html will get the following output



## Data Binding

Data binding binds the user interface or the UI to the models. With the help of it, you can manipulate the elements that are present on the website with the help of the browser. And as some of the variable changes, that change is to be reflected in the DOM.

In the case of Angular, the interaction amid the DOM and the components is being defined by the Data Binding. And Data binding has been the part of Angular since AngularJS, and you can also find it in Angular 10.

## Types of Data binding

There are two types of data binding in Angular. They are One-way data binding and two-way data binding.

### One-way data binding:

One-way data binding is the data binding where the data flows in one direction only and that that is from the data source to control. They are of different types like:

* String Interpolation of Interpolation binding
* Property Binding
* Event Binding
* Class Binding
* Style Binding

**Interpolation Binding:**

* This data binding mechanism is used to bind the data from component to the HTML template.
* In this we will bind component’s property to the HTML element by using “**{{}}”** delimiters
* **Syntax:** <tag>{{property}}</tag>

**Property Binding:**

* It is similar to interpolation where we can get the data from component to template, but it is used to change the values for html attributes or apply the values dynamically for the html attributes.
* We can bind the property to the html attribute by using **“[]”**.
* **Syntax:** <tag [attribute]=”property”></tag>

**Event Binding:**

* The event binding syntax derives with the target event name which is being inscribed inside the parenthesis on the left-hand side of the equal sign, and right of the cited statement.
* In order to use this event binding in practical we use **(eventname)** in the html template.
* Every event is handled by using a function and is called event handler that can be presented in component.
* We will discuss more about event binding later sections.

### Two-way Binding

* The angular allows two-way binding which allows you to perform data sharing in both the direction i.e. from the template to the component and vice versa.
* The two-way data binding is going to do two things which are setting the element property and the second one is to listen to the element change event.
* The two-way binding syntax is **[(ngModel)]** where ngModel is a directive and as you can view it’s the blend of the property binding syntax which is [] and the event binding syntax which is ().

**Note: ngModel** directive is available in **“FormsModule”,** so we need to import in the AppModule before we use

**Example with Data Binding:**

Create an Employee Component and write the following code

**Employee.component.ts:**

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

@Component({

    selector:'emp',

    templateUrl:'./employee.component.html'

})

export class EmployeeComponent{

    empname:string="Srikanth";

    empid:number=103;

    empage:number = 30;

    isActive:boolean = true;

    gender:string = "Male";

    country:string = "India";

    companyadd:string = "http://www.nareshit.in";

    ChangeData(){

        this.empid=102;

        this.empname = "Diyansh";

        this.empage=5;

        this.isActive=false;

        this.gender="Male";

        this.country="USA";

    }

}

**Employee.component.html:**

<div>

  <h3>Data Binding</h3>

  Employee ID: {{empid}} <br>

  Employee Name: {{empname}} <br>

  Age is: {{empage}}<br>

  Status: {{isActive}}<br>

  Gender: {{gender}}<br>

  Country: {{country}}<br>

  <!-- Company Address:<a href="{{companyadd}}">NareshIT</a> -->

  Company Address: <a [href]="companyadd">NareshIT</a>

  <br><br><hr>

  Employee ID: <input type="text" [(ngModel)]="empid"><br>

  Employee Name: <input type="text" [(ngModel)]="empname"><br>

  Select Age: <input type="number" [(ngModel)]="empage" min="18" max="40"><br>

  Status: <input type="checkbox" [(ngModel)]="isActive"><br>

  Gender:

  <input type="radio" [(ngModel)]="gender" value="Male"><span>Male</span>

  <input type="radio" [(ngModel)]="gender" value="Female"><span>Female</span>

  <br>

  Select Country:

  <select [(ngModel)]="country">

      <option>India</option>

      <option>USA</option>

      <option>UK</option>

      <option>Canada</option>

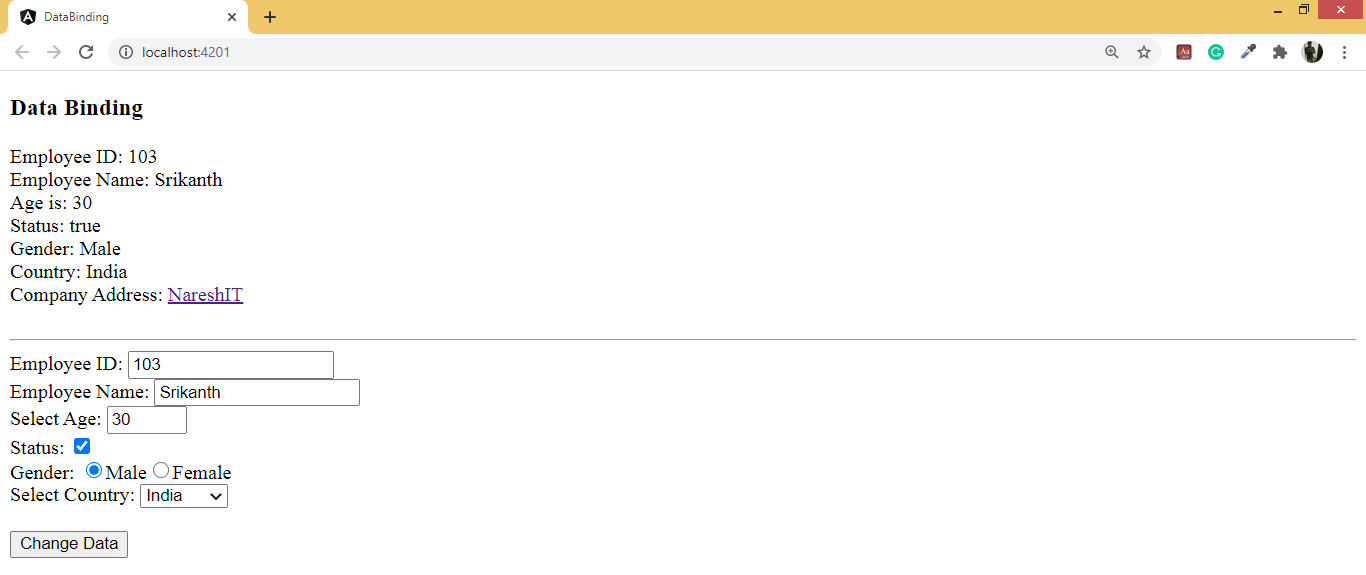
  </select>

  <br><br>

  <input type="button" value="Change Data" (click)="ChangeData()">

</div>

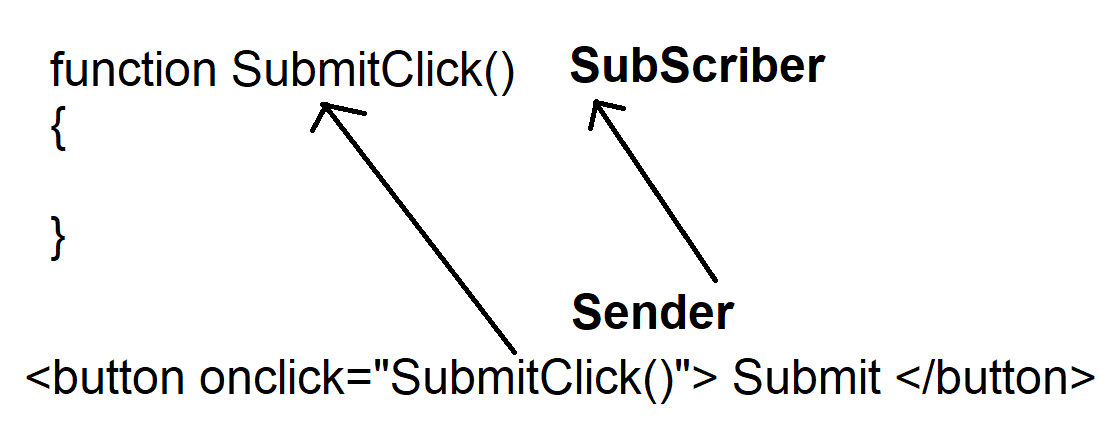
**Note:** Bootstrap the **EmployeeComponent** in **app.module.ts** and use selector of **EmployeeComponent** in **index.html** and run the application will get the following **output**.



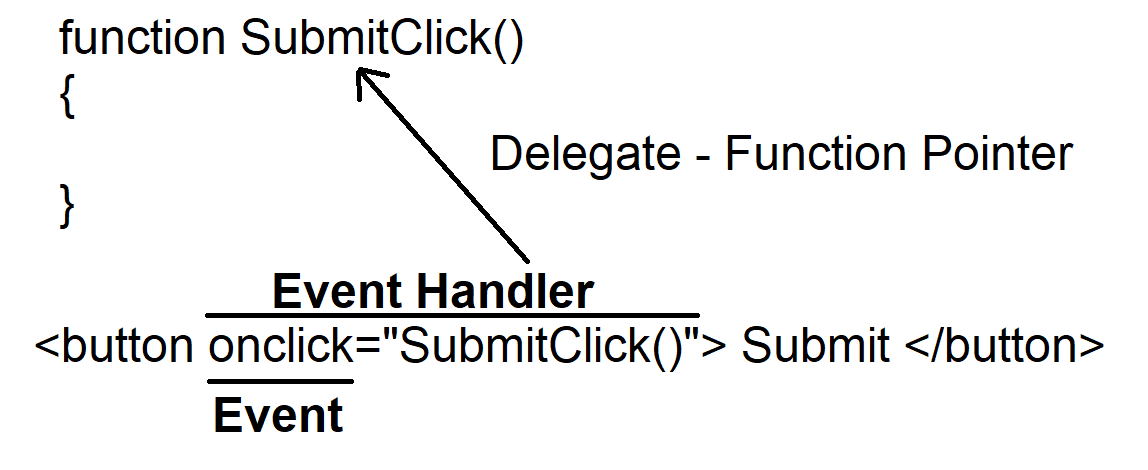
### 

### Event binding and In-depth:

1. Event is a message sent by sender to its **subscriber** in order to notify the change.
2. Event follows a software design pattern called “**Observer**”, which is a communication pattern. [Behavioral Patterns]



1. Event uses an Event Handling that follows “**Delegate**” [Function Pointer] mechanism.



1. Event handler requires definition for event arguments, even when you are not passing any argument you have define the memory for empty arguments.
2. Angular support different events like
   1. Key Events
   2. Mouse Events
   3. Timer Events
   4. Miscellaneous Events

**Example with Event Binding:**

### Create a component with the name Eventsdemo and write the following code

**Eventdemo.component.html:**

<div class="container-fluid">

  <h2>Events Demo</h2>

  <div class="form-group">

      <label>User Name</label>

      <div>

          <input (cut)="onCut()" (copy)="onCopy()" (paste)="onPaste()" [(ngModel)]="txtName" (focus)="OnFocus()" (blur)="OnBlur()" class="form-control" type="text" placeholder="Block Letters Only">

          <span>{{msg}}</span>

      </div>

  </div>

</div>

**Eventdemo.component.ts:**

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

@Component({

  selector: 'app-eventsdemo',

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

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

})

export class EventsdemoComponent {

    public msg;

    public txtName;

    public OnFocus() {

      this.msg = 'Name in Block Letters';

    }

    public OnBlur() {

      this.msg = '';

      this.txtName = this.txtName.toUpperCase();

    }

    public onCut(){

      this.msg = 'Removed and Placed on Clipboard';

    }

    public onCopy(){

      this.msg = 'Copied to Clipboard';

    }

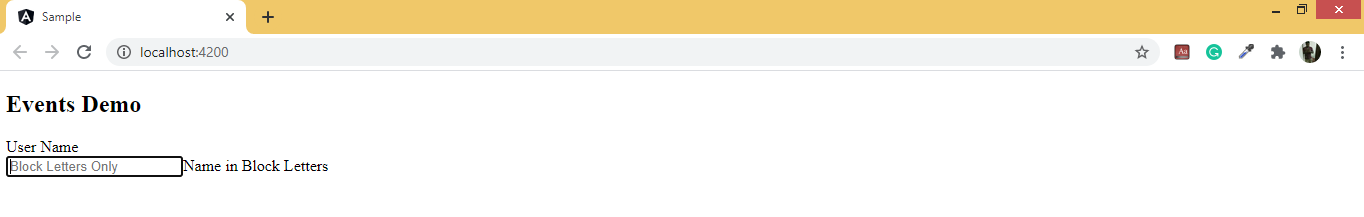
    public onPaste() {

      this.msg = 'Inserted from Clipboard';

    }

}

**Note:** Bootstrap the **Eventsdemo** component in **app.module.ts** and use selector of **Eventsdemo** in **index.html** and run the application will get the following output.



### Style Binding in Angular

1. We can use style binding for setting the style of certain view elements. And we can configure inline style with style binding.
2. Much similar to class as well as attribute binding, the style binding syntax is similar to property binding.
3. We can postulate the element in between the bracket.
4. We can implement the style binding in following ways
   1. [ngStyle]: Used to specify css properties dynemically
   2. [ngClass]: Used to specify css classes dynamically

**Example with [ngStyle] and [ngClass]:**

Create an **StylesDemo** Component and write the following code

**Styledemo.component.ts:**

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

@Component({

  selector: 'styledemo',

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

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

})

export class StyledemoComponent  {

  public styles;

  public foreColor='black';

  public alignment = 'left';

  public isApply=false;

  public ApplyStyles(){

    this.styles = {

      'color':this.foreColor,

      'text-align':this.alignment

    };

    return this.styles;

  }

}

**Styledemo.component.html:**

<div class="container">

  <h1 [ngStyle]="{'background-color':isApply ? 'red' : 'green' }">Welcome to NgStyle</h1>

  <fieldset>

      <legend>Apply Styles</legend>

      <dl>

          <dt>Fore Color</dt>

          <dd>

              <select [(ngModel)]="foreColor">

                  <option value="red">Red</option>

                  <option value="green">Green</option>

                  <option value="blue">Blue</option>

              </select>

          </dd>

          <dt>Alignment</dt>

          <dd>

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

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

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

          </dd>

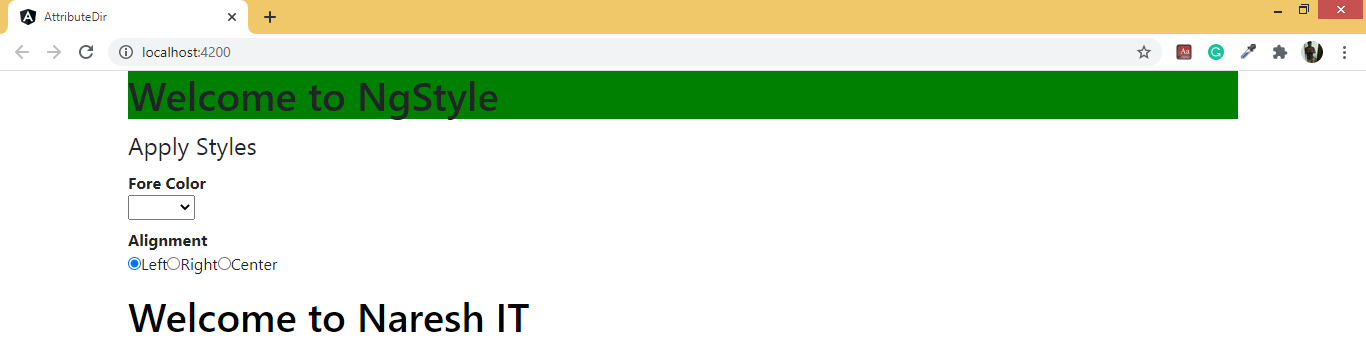
      </dl>

      <h1 [ngStyle]="ApplyStyles()">Welcome to Naresh IT</h1>

  </fieldset>

</div>

**Note:** Bootstrap the **Stylesdemo** component in **app.module.ts** and use selector of **Stylesdemo** in **index.html** and run the application will get the following output.



## Component data sharing techniques in Angular

### @input

In case we need to pass the data from the root to the leaf nodes or the child component, we can make use of the input binding via the **@input decorations**. Now you need to realize the component which has two input properties through the @input decorations. And if you will have a look at the example below, we are required to import the input from the @angular/core library.

The aliasing can also be used in the case of the @input binding. And as we find just above, the rootname is being aliased with the headName.

Now stopping the changes in the Input Property with the help of the Setter and the ngOnChange()the input property that is interrupted with thengOnchanges() and the Setter the interrupting input property support in serving to toil upon the root value.

**Example:**

Create two components like **ParentComponent** and **ChildComponent** and write the following code

**Child.component.ts**

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

@Component({

  selector: 'app-child',

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

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

})

export class ChildComponent {

      @Input() public msg;

}

**Child.component.html**

<div class="container" style="padding:30px; background-color:green; color:white; text-align:center">

  <h3>Child Component</h3>

  <p>{{msg}}</p>

</div>

**Parent.component.ts**

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

@Component({

  selector: 'app-parent',

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

  styles: [

  ]

})

export class ParentComponent {

  public hello = 'Hello ! from Parent Component';

}

**Parent.component.html:**

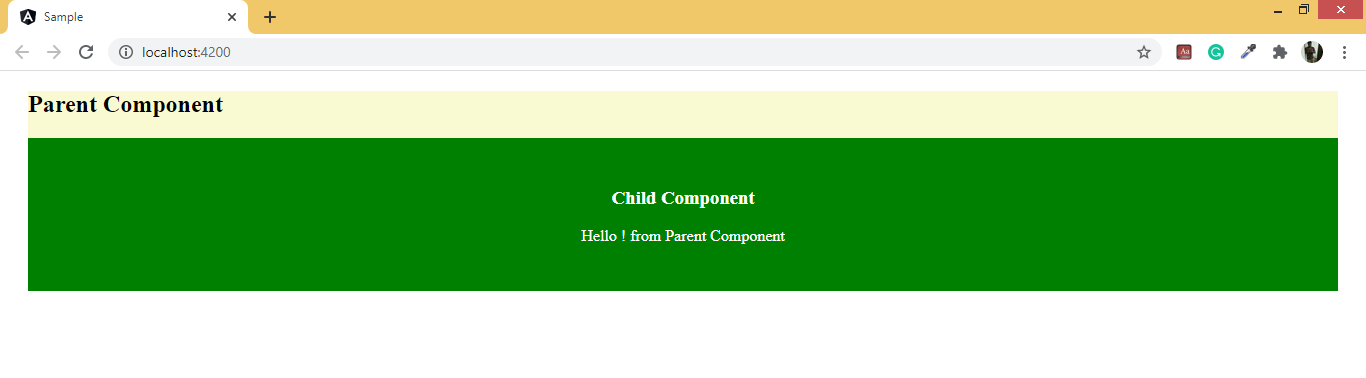
<div class="container-fluid" style="background-color: lightgoldenrodyellow; margin:20px; height: 200px;">

  <h2>Parent Component</h2>

  <app-child [msg]="hello" ></app-child>

</div>

Invoke the parent component and will get the following output:



### @Output Binding

The Output is the property that is being annotated with the @Output decorator; and the property always returns an Angular **EventEmitter**. The values come out of the component as events that are bind through event binding.

And in Angular, the component can emit the event with the help of the Output EventEmitter. And both of these lies in @angular/core. Let us consider an example where we emit the sum value from the component ChildComponent.

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

@Output() mesg:EventEmitter<string>=new EventEmitter();

  outputMessage:string="I am child component."

  sendValues(){

    this.mesg.emit(this.outputMessage);

 }

// Add above to studentresultComponent.ts

//add below code tpo app.component.ts

GetChildData(data){

    console.log(data);

 }

//Add this to app.component.html

<div>

    <app-studentresult (mesg)="GetChildData($event)"></app-studentresult>

</div>

**@ViewChild decorator**

Quite often we need to coordinate with multiple HTML elements and components in the template directly with the help of template references like @primary output or @primaryColorSample, and without the AppComponent class.

However, we do not find this as the case always. At times the AppComponent need the reference to the multiple elements which is contained inside the template, for mediating the interaction.

In that case, we can obtain the reference to the template elements and inject them in the AppComponent class through template querying. And this is what the @ViewChild is for.

**How to inject a reference to the component with the help of the @ViewChild**

Let's assume that AppComponent requires the reference to a certain component that it uses within the template, for calling the method on it directly.

In that scenario, the reference to the component instance can be injected with the help of @ViewChild.

@Component({

selector: 'app-root',

templateUrl: './app.component.html'

})

export class AppComponent {

.....

@ViewChild(StudentComponent)personComponent: StudentComponent;

.....

}

With the help of the @ViewChild, the personComponent member variable will be filled in by the Angular with the StudentComponent instance.

This StudentComponent instance which is injected is the one that is linked to the <app-student> custom element that is present in the template.

**When we should inject the variables with the help of the @ViewChild available?**

The value of the member variable injected is not available at once of the component construction time!

Angular is going to fill in the property automatically, however, only afterward in the component lifecycle, after the completion of the view initialization.

If we want to write the component initialization code which makes use if the references are injected via the @ViewChild then we need to do this inside the AfterViewinit lifecycle hook.

Below is an example.

@Component({

    selector: 'app-root',

    templateUrl: './app.component.html'

  })

  export class AppComponent implements  AfterViewInit {

    .....

    @ViewChildStudentComponent)

    personComponent: StudentComponent;

    ngAfterViewInit() {

      console.log('ngAfterViewInit() (Values on):');

      console.log("person:", this.studentComponent);

    }

    .....

  }

### @ViewChildren

If we want to access the multiple child references or variables and methods from multiple component (child component) references then we make use of the @ViewChildren.

<div >

  <div>

   <app-student></app-student> //Child component (Use 1st time)

 </div>

</div>

<div >

  <div>

   <app-student></app- student> //Child component (Use 2nd time)

  </div>

</div>

<div >

  <div>

   <button type="button" >Button</button>

 </div>

Now, in the above output, two elements are rendered and we can see two clocks, but in the console only one reference is present. So, now to get multiple children components we can use @ViewChildren() decorator.

Now let us examine this with example.

At first, we use ViewChild(). And then we first go to the first component ts file and change the ViewChild() to ViewChildren() like:

@ViewChildren(StudentComponent) myValue:QueryList<StudentComponent>;

Here, @ViewChildren is going to return a list of students, and hence we are using queryList. Hence we also require the queryList for getting the list of this kind of component.

And in ngAfterViewInit() we can get output using this.myValue.toArray();

ngAfterViewInit(){

    console.log(this.myValue.toArray());

}

# Multiple Choice Questions

**1. How many main types of data binding are there?**

a)2 b)4 c)6 d)8

**2. What are the four types of data sharing techniques in Angular?**

1. @input(), @output(), @viewChild(), @viewChildren()
2. @component, @NgModule, @injectable, @BootStrap()
3. We cannot share data
4. None of these

**3. Do we have a template and templateurl both in @Component?**

1. Both are allowed.
2. The only template is allowed.
3. The template is for inline, and the template is for explicit.
4. None of these

**4. Can we nest Components and Modules both?**

1. Only components can be nested.
2. Only Modules can be nested.
3. Both Components and Modules can be nested.
4. None of these.

**5. Can we pass data from parent to child as well as from child to parent components?**

1. Yes using @output and @input respectively.
2. Yes using @input and @output respectively.
3. No. d) None of these

**6. Do we need to add a component name anywhere before using it in another component or module?**

1. Yes in App.module.ts @NgModules section
2. Yes inAppModule.ts @NgModules/Declaration section.
3. No d) None of these

**7. Which is the root component included in all apps?**

1. AppComponent b) ParentComponent
2. RootComponent c) ChildComponent

**8. Style binding is a One-way binding. Is it so?**

1. Yes b) No c) It is a two-way binding d) None of these

**9. Unidirectional flow can be advantageous. Is it so?**

1. Yes
2. Yes, but having both one way and two-way binding is more advantages,
3. Not always. That is why Angular is better than React as we can implement both in angular.
4. No, and hence React is better than Angular. It’s a necessary feature for a good UI framework of library

.

**10. Why do you need a component?**

1. Better reusability. b) For better cohesion and coupling
2. For well defined and structured coding. d) All of these.

**11. How many components can we have in an app?**

1. 2 b) 3 c) 4 d) Any number

**12. What is the flag that we use for creating a component in a subdirectory but to show in the same folder as the root component folder?**

1. flat b) root d) dist d) None of these

# 

# FAQ/Lab assignments

1. What are angular essentials? Explain in brief.
2. Explain Angular components. What are its parts? Explain briefly each of them.
3. Explain briefly property and event binding.
4. Briefly explain property binding and Interpolation.
5. What are string interpolation and Interpolation binding? Explain in your own words.,
6. What is Angular data binding an Interpolation? Explain in brief.
7. What is a single binding?
8. What is two-way binding?
9. What is the input binding?
10. What is output binding?
11. What changes can be made with the setter in Angular?
12. What changes can you make with ngOnChange Property in Angular?
13. How is data fetched from child to parent using Output binding?
14. How is event data passed using Event binding?
15. What is the target event binding in Angular?
16. Explain briefly event handling and event handling statements in Angular.
17. What are event emitters? Explain briefly.
18. How do we bind custom events using Event Emitters?
19. Using two components explain the use of @ViewChild and @ViewChildren.
20. Create a component and add it to the first app you have created in chapter 1,
21. Explain with an app the concept of nested components.
22. Explain property binding with a code example. Do the same thing for event binding, style binding, string interpolation, and [(ngModel)].

# Summery about the chapter.

In this chapter we studied:

1. What are the components?
2. Of what Components are formed?
3. How we can create a component using CLI? Also, we studied each kind, how can component,
4. We studied the nesting of components.
5. We studied what is data binding.
6. We studied how many types of data binding are there.
7. We studied interpolation binding and the fact that it is also called string interpolation.
8. We studied property binding.
9. We studied event binding and various situations where we use it with syntax.
10. We studied four types of data sharing techniques in Angular.
11. We studied @input that is used for data flow from child to parent.
12. We studied @output which is just the opposite of the above.
13. We studied @ViewChild
14. We studied @ViewChildren.
15. And that concluded the 2nd chapter.