**Angular**

**Steps to Create a Angular project and Run**

1) Open command prompt and move to required location and type the command as to create project.

ng new project\_name

ng --- angular

new --- saying new

Ex: ng new Project3

2) Once the project is created move into the project directory in command prompt.

cd project\_name

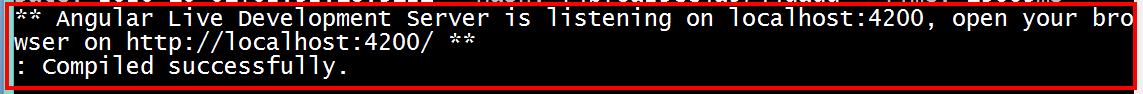
cd ---- change directory

Ex: cd Project3

3) To start the server, active into the project directory and type the command as

ng serve

ng -- angular



4) Open the browser and type the url as

http://localhost:4200

Note: Step 3 and 4 can be done in 1 Step

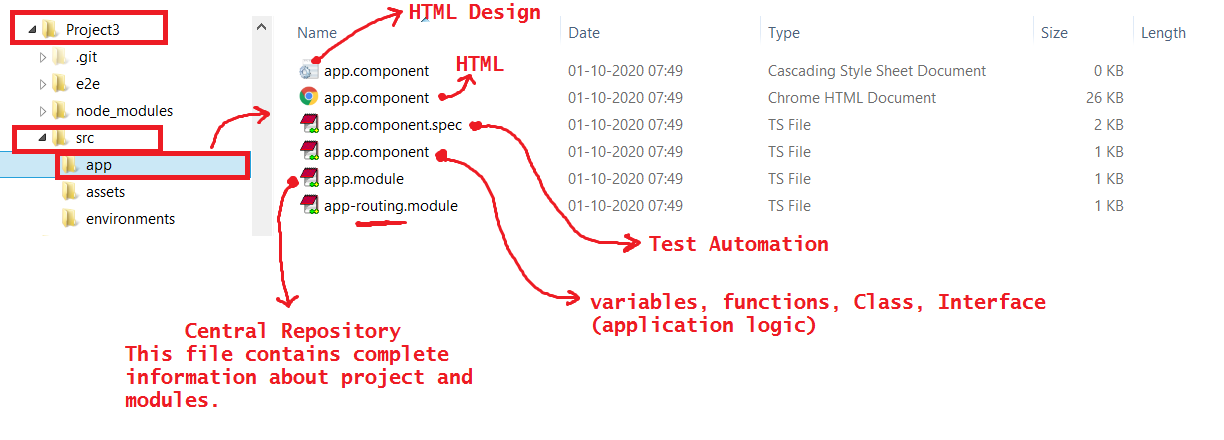
ng s -o

ng --- angular

s --- serve

-o --- Open

**After Creating the Project the Directory Structure**



**1) app.component.css**

In this file we write css code to design the HTML

**h1**{  
 **text-align**: **center**;  
 **background-color**: **orangered**;  
 **color**: **white**;  
 **font-family**: **"Agency FB"**;  
}

**2) app.component.html**

In this file we write presentation logic

<**h1**>Welcome Angular Application</**h1**>  
  
<**h3**>Student IDNO : {{ **s\_idno** }}</**h3**>  
<**h3**>Student Name : {{ **s\_name** }}</**h3**>

**3) app.component.ts**

In this file we write application logic (Business logic)

**import** { Component } **from '@angular/core'**;  
  
@Component({  
 **selector**: **'app-root'**,  
 **templateUrl**: **'./app.component.html'**,  
 **styleUrls**: [**'./app.component.css'**]  
})  
**export class** AppComponent {  
 **s\_name**:**string** = **"Ravi Kumar"**;  
 **s\_idno**:**number** = 101;  
}

**4) app.module.ts**

This file is a **central repository**, where we register all the components of your application.

**Example**

**import** { AppComponent } **from './app.component'**;  
  
  
@NgModule({  
 **declarations**: [AppComponent],

.......})

**5) Decorator**

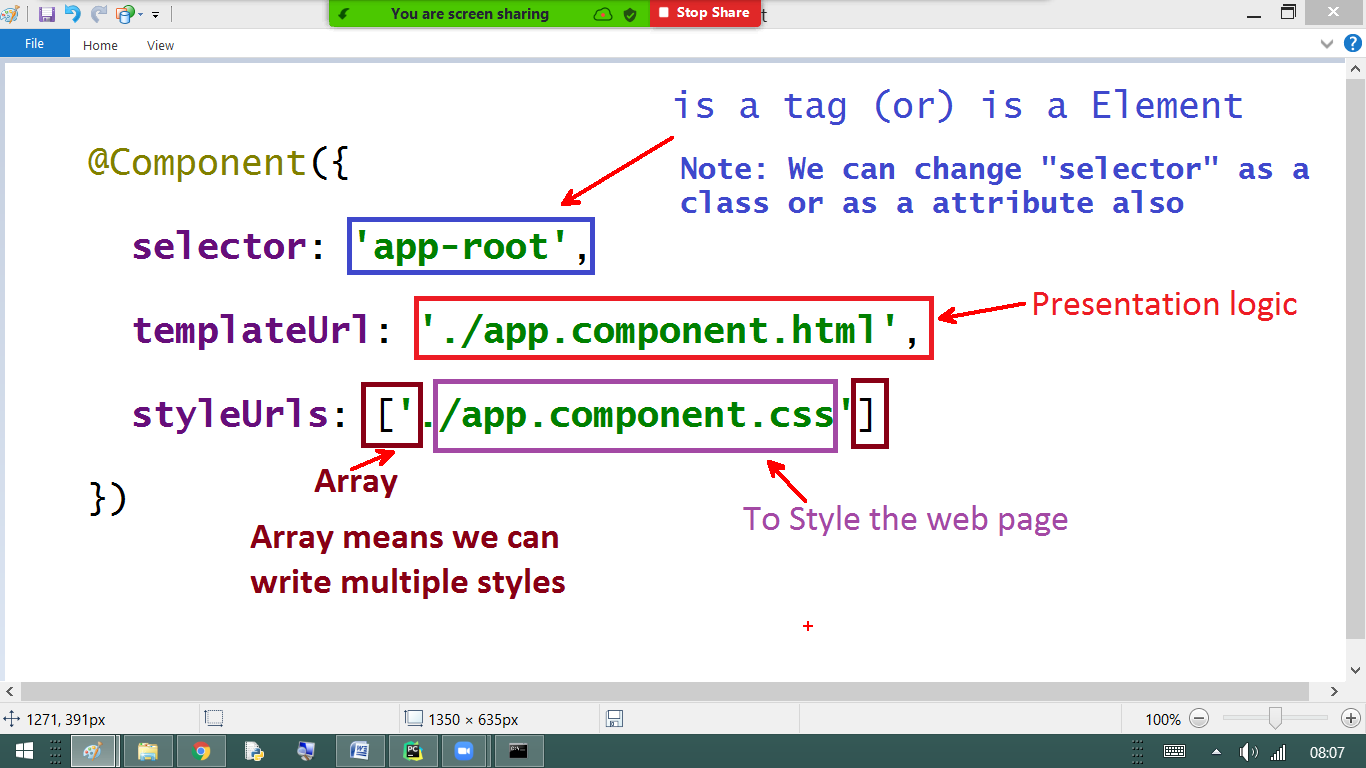
Every component will have a decorator.

The Decorator will specify the 3 parts

1) Selector (**Element Name)**

2) TemplateUrl (**HTML Name**)

3) StyleUrls (**Styling (Css) name**.)



**Selector**

We can write the selector in 3 ways

**1)** selector : <**tag-name**>

This tag name we use in HTML (index.html)

In TypeScript ---- **selector**: **'app-root'**,

In HTML ---- <**app-root**></**app-root**>

**2)** selector : < .**class\_name** >

The class name is used inside the tag.

In TypeScript ---- **selector**: **'.app\_sample'**,

In HTML ---- <**div class="app\_sample"**></**div**>

**3)** selector : <**'[attribute\_name]'**>

Use the attribute name in a tag

In TypeScript ---- **selector**: **'[app\_sample]'**,

In HTML ---- <**div app\_sample**></**div**>

**TemplateUrl**

We can write the TemplateUrl in 2 ways

**1)** as a template file (HTML file)

In TypeScript : **templateUrl**: **'./app.component.html'**,

In app.component.html <**h1**>Welcome Angular Application</**h1**>

**2)** as inner Template (or) inner HTML

In TypeScript : **template** : **"<h1>This is Inner HTML</h1>"**,

Note: If we write inner template in Double quotes we cannot write multiple lines, to do so we use back ticks.

Example

In TypeScript : **template** : **`<h1>Welcome to Template HTML</h1>  
 <h2>This is Multi Line Example</h2>`**,

**StyleUrls**

We can write the TemplateUrl in 2 ways

**1)** as a css file

In TypeScript : **styleUrls**: [**'./app.component.css'**],

In app.component.html <**h1**>Welcome Angular Application</**h1**>

**2)** as inner Css

In TypeScript :

**styles**: [**`h1{  
 background-color: blue;  
 color: white;  
 text-align: center;  
}  
h2{  
 background-color: brown;  
 color: wheat;  
 font-family: Chiller;  
 text-align: center;  
}`**]

Note: If we write inner css in Double quotes we cannot write multiple lines, to do so we use back ticks.

**Interpolation and template expressions**

Interpolation allows you to incorporate calculated strings into the text between HTML element tags and within attribute assignments.

**Example :** {{ ...... }}

Template expressions are what you use to calculate those strings.

**Example :**

10 + 20

"Sathya" + "Tech"

"Python With" + "Naveen"

Note: We cannot write expression directly into HTML, to do so we use interpolation.

**Example :**

**<h1>{{ 10+20 }}</h1>  
<h1>{{ "Sathya" + "Tech" }}</h1>  
<h1>{{ "sathya" + s\_name }}</h1>**

**Using Class properties into Interpolation**

Class properties means **variables** and **methods**

**Variables Example**

1) Define a Typescript class in "app.component.ts"

**export class** AppComponent

{  
 *// instance variables* **s\_name**:**string** = **'Ravi'**;  
 **s\_age**:**number** = 23;  
 **s\_marks**:**number**[] = [89,98,75];  
}

2) Using interpolation we can show these properties into HTML.

**<h1>Student Name : {{ s\_name }}</h1>  
<h1>Student Age : {{ s\_age }}</h1>  
<h1>Student Marks : {{ s\_marks }}</h1>**

**Methods Example**

1) Define a Typescript class in "app.component.ts"

**export class** AppComponent

{  
 *// instance method* show():**string** {  
 **return "Naveen"**;  
 }  
  
}

2) Using interpolation we can show these properties into HTML.

**<h1>{{ show() }}</h1>**

**Property binding [property]**

Use property binding to *set* properties of target elements or directive @[Input](https://angular.io/api/core/Input)() decorators.

1) Define a Typescript class in "app.component.ts"

2) In defined class, define some variables which you want to bind with HTML elements.

**Example**

**export class** AppComponent {  
 *// instance variables* **s\_name**:**string** = **'Ravi'**;  
}

3) Binding the class property to an input element.

**<input [value]="s\_name"/>**

**Q) Can we use interpolation instead of property binding ?**

A) **Yes** we can use, but the interpolation will not work for boolean values (like true or false).

**Example 1**

**Code in HTML**

**<input [value]="s\_name"/> <br><br>  
<input value="{{ s\_name }}"/>**

**Code in TypeScript Class.**

**export class** AppComponent {  
 *// instance variables* **s\_name**:**string** = **'Ravi'**;  
}

**Example 2**

**Code in HTML**

**<input value="{{ s\_name }}" disabled="true"/>  
<input value="{{ s\_name }}" disabled="{{ status }}"/>  
<input value="{{ s\_name }}" [disabled]="status" />**

**Code in TypeScript class.**

**export class** AppComponent {  
 *// instance variables* **status**:**boolean** = **false**;  
  
}

**Other syntax for Property binding " bind- "**

**Code in HTML**

**<button [disabled] ="status"> Click Me </button>  
<br><br>  
<button bind-disabled ="status"> Click Me </button>**

**Code in TypeScript Class**

**export class** AppComponent {  
 *// instance variables* **status**:**boolean** = **false**;  
  
}

**Class binding**

Here's how to set the class attribute without a binding in plain HTML**:**  **<div class="foo bar">Some text</div>**

You can also add and remove CSS class names from an element's class attribute with a class binding.

To create a single class binding, start with the prefix class followed by a dot (.) and the name of the CSS class (for example, [class.foo]="hasFoo").

**Example**

**Code in HTML**

**<h1 [class.status1]="mystatus">I am a Django Developer</h1>**

**Code in Styles**

**styles**: [**`  
 .status1{background-color: brown;color: white}   
`**]

**TypeScript Class**

**export class** AppComponent {  
  
 **my\_status1** = **"status1"**;  
 **my\_status2** = **"status2"**;  
 **mystatus** = **false**; / true  
}

Angular adds the class when the bound expression is truthy, and it removes the class when the expression is falsy (with the exception of undefined, see [styling delegation](https://angular.io/guide/attribute-binding#styling-delegation)).

To create a binding to multiple classes, use a generic [class] binding without the dot (for example, [class]="classExpr").

**Example 1**

**Code in HTML**

**<h1 [class]="my\_status1">Naveen</h1>**

**Code in styles**

**styles**: [**`  
 .status1{background-color: brown;color: white}**

**`**]

**Code in TypeScript**

**export class** AppComponent {  
  
 **my\_status1** = **"status1"**;  
}

The expression can be a space-delimited string of class names, or you can format it as an object with class names as the keys and truthy/falsy expressions as the values. With object format, Angular will add a class only if its associated value is truthy.

**Example 2**

**Code in HTML**

**<h1 [ngClass]="checkstatus"> Welcome to Angular </h1>  
 <h1></h1>**

**Code in styles**

**styles**: [**`  
 .status\_yes{background-color:green;color: white }  
 .status\_no{background-color: red;color: white}  
`**]

**Code in Typescript Class**

**export class** AppComponent {  
 **my\_status** = **true**;  
  
 **public checkstatus** = {  
 **'status\_yes'**: **this**.**my\_status**,  
 **'status\_no'** : !**this**.**my\_status** };  
}

**Style Binding**

Here's how to set the [style](https://angular.io/api/animations/style) attribute without a binding in plain HTML:

**<h1 style="color: blue"> Welcome to Angular </h1>**

You can also set styles **dynamically** with a **style binding**.

To create a single style binding, start with the prefix [style](https://angular.io/api/animations/style) followed by a dot (.) and the name of the CSS style property (for example, [style.width]="width").

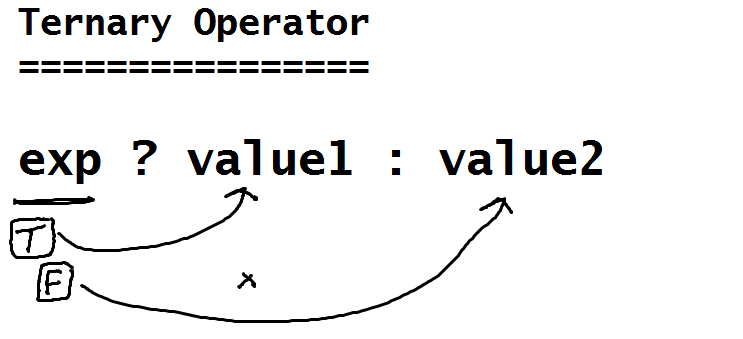
**Example**

**<h1 [style.color]="'red'" > Welcome to Angular </h1>**

**<h1 [style.fontSize]="'50px'" > Welcome to Angular </h1>  
  
<h1 [style.fontFamily]="'Chiller'" > Welcome to Angular </h1>**

The property will be set to the value of the bound expression, which is normally a string. Optionally, you can add a unit extension like em or %, which requires a number type.

**Style binding conditionally**



**Example**

**Code in HTML**

**<h1 [style.color]="status? 'green' : 'red' " > Welcome to Angular </h1>**

**Code in Typescript Class**

**export class** AppComponent {  
  
 **public status** = **true**;  
}

**Multiple Style Binding**

**Example**

Code in HTML

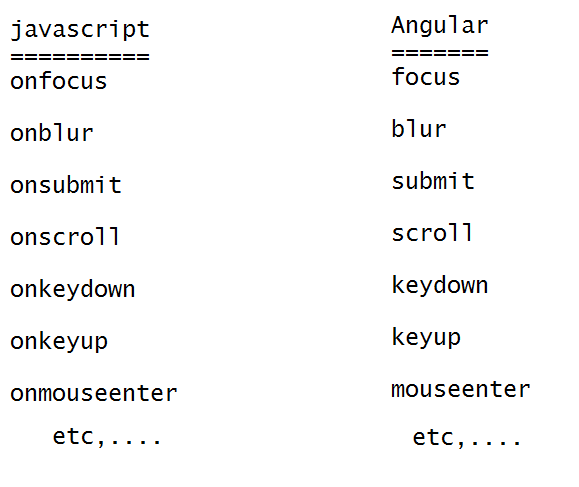
**<h1 [ngStyle]="my\_heading1"> Welcome to Angular </h1>**

Code in Typescript Class

**export class** AppComponent {  
  
 **public status** = **true**;  
  
 **public my\_heading1** = {  
 **color** : **'orangered'**,  
 **fontFamily** : **'Chiller'**,  
 **fontSize** : **'50px'** }  
}

**Event binding (event)**

Event binding allows you to listen for certain events such as keystrokes, mouse movements, clicks, and touches.



**Syntax**



Angular event binding syntax consists of a **target event name** within parentheses on the left of an equal sign, and a quoted **template statement** on the right.

**Example**

Code in Template

**<button (click)="display()" type="submit">Click Me</button>  
  
<h1> {{ message }}</h1>**

**Code in TypeScript**

**export class** AppComponent {  
  
 **public message** = **"..."**;  
 **public** display()  
 {  
 **this**.**message** = **"Hello Angular"**;  
 }  
}

**Template reference variables (#var)**

A template reference variable is often a reference to a DOM element within a template.

It can also refer to a directive (which contains a component), an element, [TemplateRef](https://angular.io/api/core/TemplateRef), or a [web component](https://developer.mozilla.org/en-US/docs/Web/Web_Components).

Use the hash symbol (#) to declare a reference variable.

**Example**

**Code in Template**

**<input type="text" placeholder="Name" (keyup)="show(*username*.value)" #username/>  
  
<h2> {{ sample\_text }}</h2>**

**Code in TypeScript**

**export class** AppComponent {  
  
 **sample\_text** = **" ... "**;  
 show(given\_username)  
 {  
 **this**.**sample\_text** = given\_username;  
 }  
}

Example on **Event Binding** and **Template Reference Variable**.

Code in Template

**<input type="number" placeholder="1st No" (keyup)="first\_number(*first*.value)" #first/>  
  
<input type="number" placeholder="2nd No" (keyup)="second\_number(*second*.value)" #second/>  
  
<h1>{{ fno }} + {{ sno }} = {{ fno+sno }}</h1>  
<h1>{{ fno }} - {{ sno }} = {{ fno-sno }}</h1>  
<h1>{{ fno }} \* {{ sno }} = {{ fno\*sno }}</h1>  
<h1>{{ fno }} / {{ sno }} = {{ fno/sno }}</h1>**

Code in Typescript

**export class** AppComponent {  
  
 **public fno**:**number** = 0;  
 **public sno**:**number** = 0;  
  
 first\_number(no1)  
 {  
 **this**.**fno** = no1;  
 }  
  
 second\_number(no2)  
 {  
 **this**.**sno** = no2;  
 }  
}

**Two-way binding [(...)]**

Two-way binding gives your app a way to share data between a component class and its template.

Two-way binding does two things:

1. Sets a specific element property.
2. Listens for an element change event.

Angular offers a special *two-way data binding* syntax for this purpose, [()].

The [()] syntax combines the brackets of **property binding, [],** with the parentheses of **event binding, ().**

**[( )] = BANANA IN A BOX**

Visualize a *banana in a box* to remember that the parentheses go *inside* the brackets.

Note 1 : To work with two-way Binding we need to import "FormsModule".

Note 2: Open "app.module.ts" file and do the import

**import** { FormsModule } **from '@angular/forms'**;

Note 3 : After import add the "Forms Module" into "imports" array.

**imports**: [  
 BrowserModule,  
 AppRoutingModule,  
 FormsModule  
],

Note 4 : If You do the Note 2 and 3 then only "ngModel" element will work else will return error.

Example Program

Code in HTML

**<input type="number" [(ngModel)]="no1"/>  
<input type="number" [(ngModel)]="no2"/>  
  
<h3>{{ no1 }} + {{ no2 }} = {{ no1+no2 }}</h3>  
<h3>{{ no1 }} - {{ no2 }} = {{ no1-no2 }}</h3>  
<h3>{{ no1 }} \* {{ no2 }} = {{ no1\*no2 }}</h3>  
<h3>{{ no1 }} / {{ no2 }} = {{ no1/no2 }}</h3>**

Code in Typescript

**export class** AppComponent {  
  
 **public no1**:**number** = 0;  
 **public no2**:**number** = 0;  
  
}

**Working With Array**

**Example-1 : On Array**

Code in Typescript

**export class** AppComponent {  
  
 **public names**:**string**[] = [**"Ravi"**,**"Kumar"**,**"Mohan"**,**"Murali"**,**"Krishna"**];

}

Code in Template

<**table align="center" border="2"**>  
  
 <**tr**><**th**>SNO</**th**><**th**>NAME</**th**></**tr**>  
  
 <**tr \*ngFor="let *x* of names; let *i* = *index*"**>  
 <**th**>{{ ***i*** }}</**th**>  
 <**th**>{{ ***x*** }}</**th**>  
 </**tr**>  
  
</**table**>

**Example-2 : On Nested Array**

Code in Typescript

**export class** AppComponent {  
   
 **public details**:[**string**,**number**][] = [[**'Ravi'**,456],[**'Kumar'**,556],[**'Mohan'**,386],[**'Murali'**,560],[**'Krishna'**,406]];

}

Code in Template

<**table align="center" border="2"**>  
  
 <**tr**><**th**>SNO</**th**><**th**>NAME</**th**><**th**>Total Marks</**th**></**tr**>  
  
 <**tr \*ngFor="let *x* of details; let *i*=*index*"**>  
 <**th**>{{ ***i***+1 }}</**th**>  
 <**th**>{{ ***x***[0] }}</**th**>  
 <**th**>{{ ***x***[1] }}</**th**>  
 </**tr**>  
  
</**table**>

**Example - 3 : On Nested Array using "if" and "else".**

Code in Typescript

**export class** AppComponent {  
  
 **public marks**:[**string**,**number**[]][] = [[**'Ravi'**,[45,65,75,89,8,95]],[**'Kumar'**,[56,48,15,78,99,56]],[**'Mohan'**,[96,69,78,85,16,48]]]  
  
}

Code in Template

<**table align="center" border="2"**>  
 <**tr**><**th colspan="8"**> >=35 are pass marks else Fail </**th**></**tr**>  
 <**tr**><**th rowspan="2"**>SNO</**th**><**th rowspan="2"**>NAME</**th**><**th colspan="6"**> Marks</**th**></**tr**>  
 <**tr**> <**th**>Sub-1</**th**><**th**>Sub-2</**th**><**th**>Sub-3</**th**><**th**>Sub-4</**th**><**th**>Sub-5</**th**><**th**>Sub-6</**th**> </**tr**>  
  
 <**tr \*ngFor="let *x* of marks; let *i*=*index*"**>  
 <**th**>{{ ***i***+1 }}</**th**>  
 <**th**>{{ ***x***[0] }}</**th**>  
 <**th \*ngFor="let *y* of *x*[1]"**>  
 <**h1 \*ngIf="*y*>=35;then *pass*;else *fail*"**></**h1**>  
 <**ng-template #pass**>  
 {{ ***y*** }} -- Pass  
 </**ng-template**>  
  
 <**ng-template #fail**>  
 {{ ***y*** }} -- Fail  
 </**ng-template**>  
 </**th**>  
 </**tr**>  
</**table**>

**Working With 1 - Object -- (Object Initialization)**

Code in Template

**export class** AppComponent {  
  
 **public emp1**:Employee = {**e\_idno**:101,**e\_name**:**"Ravi"**,**e\_salary**:185000.00};

}  
  
**class** Employee  
{  
 *// instance variables of a class.* **public e\_idno**:**number**;  
 **public e\_name**:**string**;  
 **public e\_salary**:**number**;  
  
}

Code in Typescript

<**h1 class = one**> One Object Example </**h1**>  
<**h1**> Employee- Object Variable = {{ **emp1** }}</**h1**>  
<**h2**> Employee IDNO = {{ **emp1**.**e\_idno**}}</**h2**>  
<**h2**> Employee NAME = {{ **emp1**.**e\_name**}}</**h2**>  
<**h2**> Employee SALARY = {{ **emp1**.**e\_salary**}}</**h2**>

**Working With Array of Objects ( Objects Initializations )**

Code in Typescript

**export class** AppComponent {  
  
 **public employees**:Employee[] = [  
 {**e\_idno**:101,**e\_name**:**"Ravi"**,**e\_salary**:185000.00},  
 {**e\_idno**:102,**e\_name**:**"Kumar"**,**e\_salary**:285000.00},  
 {**e\_idno**:103,**e\_name**:**"Mohan"**,**e\_salary**:200000.00},  
 {**e\_idno**:104,**e\_name**:**"Prasad"**,**e\_salary**:205000.00},  
 {**e\_idno**:105,**e\_name**:**"Krishna"**,**e\_salary**:105000.00}  
 ];  
}  
  
  
**class** Employee  
{  
 *// instance variables of a class.* **public e\_idno**:**number**;  
 **public e\_name**:**string**;  
 **public e\_salary**:**number**;  
  
}

Code in HTML

<**h1 class="one"**>Array of Employee Objects</**h1**>  
<**h1**> Employee - Array Object Variable = {{ **employees** }}</**h1**>  
<**table align="center" border="2"**>  
 <**tr style="background-color**: **black**;**color**: **orange"**>  
 <**th**>IDNO</**th**>  
 <**th width="200"**>NAME</**th**>  
 <**th width="100"**>SALARY</**th**>  
 </**tr**>  
  
 <**tr \*ngFor="let *emp* of employees"**>  
 <**th**>{{ ***emp***.**e\_idno** }}</**th**>  
 <**th**>{{ ***emp***.**e\_name** }}</**th**>  
  
<**div \*ngIf="*emp*.e\_salary>=200000; then *one*;else *two*"**></**div**>  
  
 <**ng-template #one**>  
<**th style="background-color**: **red**;**color**: **white"**>

{{ ***emp***.**e\_salary** }}

</**th**>  
 </**ng-template**>  
  
 <**ng-template #two**>  
<**th style="background-color**: **green**;**color**: **white"**>

{{ ***emp***.**e\_salary** }}

</**th**>  
 </**ng-template**>  
  
 </**tr**>  
</**table**>

**Working with 1 Object ( Using "new" Keyword)**

Code in Typescript

**export class** AppComponent {  
 *//single object* **public stu1**:Student = **new** Student(101,**"Ravi"**,[10,20,30,40,50,60]);

}

**class** Student  
{  
 **public s\_idno**:**number**;  
 **public s\_name**:**string**;  
 **public s\_marks**:**number**[];  
  
 **constructor**(id,na,marks)  
 {  
 **this**.**s\_idno** = id;  
 **this**.**s\_name** = na;  
 **this**.**s\_marks** = marks;  
 }  
  
 **public** total\_marks():**number** {  
 **let** sum:**number** = 0;  
 **for**(**let** x **of this**.**s\_marks**)  
 {  
 sum+=x;  
 }  
 **return** sum;  
 }  
}

Code in HTML

<**h1**>Student Object = {{ **stu1** }}</**h1**>  
<**h1**>Student IDNO = {{ **stu1**.**s\_idno** }}</**h1**>  
<**h1**>Student Name = {{ **stu1**.**s\_name** }}</**h1**>  
<**h1**>Student Marks = {{ **stu1**.**s\_marks** }}</**h1**>  
<**h1**> Total Marks = {{ **stu1**.total\_marks() }}</**h1**>

**Working with Array of Objects ( Using "new" keyword )**

Code in Typescript

**export class** AppComponent {  
   
 **public students**:Student[] =[  
 **new** Student(101,**"Ravi"**,[10,20,30,40,50,60]),  
 **new** Student(102,**"Kumar"**,[11,22,33,44,55,66]),  
 **new** Student(103,**"Mohan"**,[77,88,99,10,20,30]),  
 **new** Student(104,**"Murali"**,[55,66,88,22,11,44]),  
 **new** Student(105,**"krishna"**,[10,1,20,4,55,65])  
 ];  
}

**class** Student  
{  
 **public s\_idno**:**number**;  
 **public s\_name**:**string**;  
 **public s\_marks**:**number**[];  
  
 **constructor**(id,na,marks)  
 {  
 **this**.**s\_idno** = id;  
 **this**.**s\_name** = na;  
 **this**.**s\_marks** = marks;  
 }  
  
 **public** total\_marks():**number** {  
 **let** sum:**number** = 0;  
 **for**(**let** x **of this**.**s\_marks**)  
 {  
 sum+=x;  
 }  
 **return** sum;  
 }  
}

Code in HTML

<**table align="center" border="2"**>  
 <**tr**>  
 <**th rowspan="2"**>Student ID</**th**>  
 <**th rowspan="2"**>Student NAME</**th**>  
 <**th colspan="6"**>Student marks</**th**>  
 <**th rowspan="2"**>Total Marks</**th**>  
 </**tr**>  
 <**tr**>  
 <**th**>Subj-1</**th**><**th**>Subj-2</**th**><**th**>Subj-3</**th**><**th**>Subj-4</**th**><**th**>Subj-5</**th**><**th**>Subj-6</**th**>  
 </**tr**>  
  
 <**tr \*ngFor="let *s* of students"**>  
 <**th**> {{ ***s***.**s\_idno** }}</**th**>  
 <**th**> {{ ***s***.**s\_name** }}</**th**>  
 <**th \*ngFor="let *m* of *s*.s\_marks"**>  
 <**h1 \*ngIf="*m* >=35; then *pass*; else *fail*"**></**h1**>  
 <**ng-template #pass**>  
 <**span style="background-color**: **black**;**color**: **gold"**> {{ ***m*** }}</**span**>  
 </**ng-template**>  
  
 <**ng-template #fail**>  
 <**span style="background-color**: **#ff0900**;**color**: **#fff3fc"**> {{ ***m*** }}</**span**>  
 </**ng-template**>  
 </**th**>  
 <**th**>{{ ***s***.total\_marks() }}</**th**>  
 </**tr**>  
</**table**>

**Note :** same example we can implement on imported class also.

In "\*ngFor" we can use 4 variables like.

1) index --- by default it will start from Zero

2) even --- if the index position is even it will return "true".

3) odd --- if the index position is odd it will return "true".

4) count --- It will return the count of your loop.

**Example**

<**h2 \*ngFor="let *no* of students; let *i* = *index*;let *e* = even;let *o*=odd;let *c* = *count*"**>  
 {{ ***no*** }} -- {{ ***i*** }} -- {{ ***e*** }} -- {{ ***o*** }} -- {{ ***c*** }}  
</**h2**>

**Angular Pipe's**

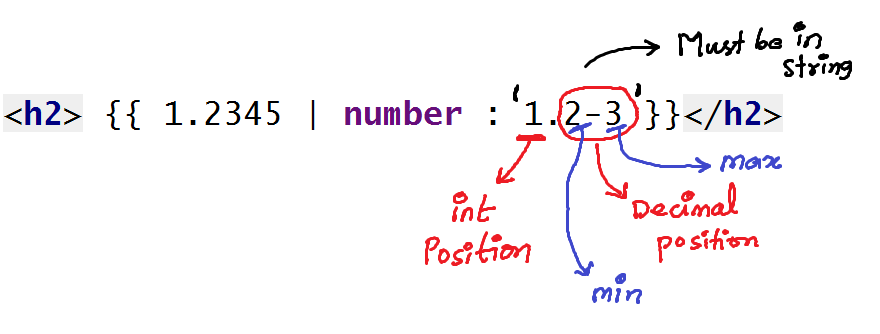
Angular Pipe's is used to transform the data, It means it will change from one format to other format.

Angular Pipes will change the data format not data.

Angular Pipes are **strings**, **numbers**, **currency**, **percent**, **date** and **slice(:) .**

**Example**

<**h1 class="one"**>**Example on Date Format** </**h1**>  
<**h2**> {{ **current\_date** }}</**h2**>  
<**h2**> {{ **current\_date** | **date**:'short' }}</**h2**>  
<**h2**> {{ **current\_date** | **date**:'shortDate' }}</**h2**>  
<**h2**> {{ **current\_date** | **date**:'shortTime' }}</**h2**>  
  
<**h2**> {{ **current\_date** | **date**:'medium' }}</**h2**>  
<**h2**> {{ **current\_date** | **date**:'mediumDate' }}</**h2**>  
<**h2**> {{ **current\_date** | **date**:'mediumTime' }}</**h2**>  
  
<**h2**> {{ **current\_date** | **date**:'long' }}</**h2**>  
<**h2**> {{ **current\_date** | **date**:'longDate' }}</**h2**>  
<**h2**> {{ **current\_date** | **date**:'longTime' }}</**h2**>  
  
<**h2**> Custom DATE : {{ **current\_date** | **date** : "dd-MM-yyyy"}}</**h2**>  
<**h4**> NOTE: In javascript small "mm" represent min's</**h4**>  
<**h4**> NOTE: In javascript small "MM" represent month</**h4**>  
<**h2**> Custom DATE : {{ **current\_date** | **date** : "MM-dd-yyyy"}}</**h2**>  
  
<**h1 class="one"**>**Example on Percent Format** </**h1**>  
<**h2**> Product {{ **product\_no** }} Dicount is {{ **discount\_percent** }}% </**h2**>  
<**h2**> Product {{ **product\_no** }} Dicount is {{ **discount\_percent** | percent }} </**h2**>  
  
  
<**h1 class="one"**>**Example on Currency Format** </**h1**>  
<**h2**> Salary Per Month is {{ **salary** }} </**h2**>  
<**h2**> Salary Per Month is {{ **salary** | **currency** }} </**h2**>  
<**h4**> Note : The Default Currency format is USD </**h4**>  
<**h2**> Salary Per Month is {{ **salary** | **currency**:'GBP' }} </**h2**>  
<**h2**> Salary Per Month is {{ **salary** | **currency**:'GBP':'code' }} </**h2**>  
<**h2**> Salary Per Month is {{ **salary** | **currency**:'INR' }} </**h2**>  
  
  
<**h1 class="one"**>**Example on Number Format** </**h1**>



<**h2**> {{ 1.2345 }}</**h2**>  
<**h2**> {{ 1.2345 | **number** : '1.2-3' }}</**h2**>  
<**h2**> {{ 1.2306 | **number** : '1.2-3' }}</**h2**>  
<**h2**> {{ 1.2306 | **number** : '3.2-3' }}</**h2**>  
<**h2**> {{ 78945.2306 | **number** : '3.2-3' }}</**h2**>  
<**h2**> {{ 5.2 | **number** : '1.2-3' }}</**h2**>  
  
<**h2**> Add = {{ **no1** + **no2** }}</**h2**>  
<**h2**> Sub = {{ **no1** - **no2** }}</**h2**>  
<**h2**> Mul = {{ **no1** \* **no2** }}</**h2**>  
<**h2**> Div = {{ **no1** / **no2** }}</**h2**>  
<**h2**> Div = {{ **no1** / **no2** | **number** : '1.2-4'}}</**h2**>  
  
<**h1 class="one"**>**Example on String Format** </**h1**>  
<**h2**> {{ **employee\_name** }}</**h2**>  
<**h2**> {{ **employee\_name** | uppercase }}</**h2**>  
<**h2**> {{ **employee\_name** | lowercase }}</**h2**>  
<**h2**> {{ **employee\_name** | titlecase }}</**h2**>

<**h1 class="one"**>**Example on Slice Format** </**h1**>  
<**h2**> {{ **employee\_name** | *slice* :3 }}</**h2**>  
<**h2**> {{ **employee\_name** | *slice* :2:6 }}</**h2**>

**Custom Pipes**

1) Define custom pipe class with **"@pipe()"** decorator.

2) The defined class must implement **"PipeTransform"** interface.

3) The **"PipeTransform"** interface contains one un-implemented method i.e **"transform".**

4) The **"transform"** method must be implemented by your class.

5) Register the class into **"declarations"** of **"app.module.ts"** file.

**Note:** the above given five steps will be automated by using the below command.

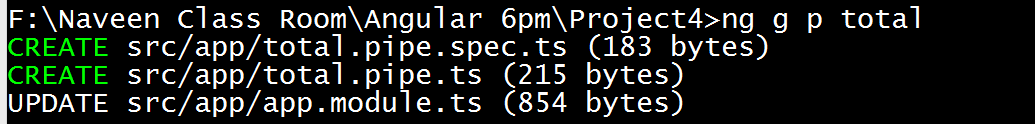
To create custom pipe we need to use a command like "**ng g p <pipe\_name>"** .

ng --- angular

g --- generate

p --- pipe

**Example**

****

"total" Custom pipe is created

Code in app/total.pipe.ts

**import** { Pipe, PipeTransform } **from '@angular/core'**;  
  
@Pipe({  
 **name**: **'result'**})  
**export class** TotalPipe **implements** PipeTransform {  
  
 transform(total\_marks:**number**):**string** {  
 **if** (total\_marks >= 350)  
 {  
 **return "Pass"**;  
 }  
 **else** {  
 **return "Fail"**;  
 }  
 }  
}

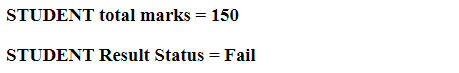
Code in typescript file

**export class** StudentsComponent **implements** OnInit {  
  
 **public total\_marks** = 150;  
  
 **constructor**() { }  
  
 ngOnInit(): **void** {  
 }  
}

Using custom pipe in template

<**h3**>STUDENT total marks = {{ **total\_marks** }}</**h3**>  
<**h3**>STUDENT Result Status = {{ **total\_marks** | **result** }}</**h3**>

**Output**

****

**Example on Slice Operator in Array Of OBJECTS with Event Binding.**

Code in Typescript : app/employees\_information.ts

**export class** Employees{  
 **public employee\_idno**:**number**;  
 **public employee\_name**:**string**;  
 **public employee\_salary**:**number**;  
  
 **constructor**(idno,name,salary)  
 {  
 **this**.**employee\_idno** = idno;  
 **this**.**employee\_name** = name;  
 **this**.**employee\_salary** = salary;  
 }  
}

Code in Typescript : app/app.component.ts

**export class** AppComponent {  
  
 **public emps**:Employees[] = [  
 **new** Employees(101,**"Ravi"**,185000.00),  
 **new** Employees(102,**"Ravi"**,185000.00),  
 **new** Employees(103,**"Ravi"**,185000.00),  
 **new** Employees(104,**"Ravi"**,185000.00),  
 **new** Employees(105,**"Ravi"**,185000.00),  
 **new** Employees(106,**"Ravi"**,185000.00),  
 **new** Employees(107,**"Ravi"**,185000.00),  
 **new** Employees(108,**"Ravi"**,185000.00),  
 **new** Employees(109,**"Ravi"**,185000.00),  
 **new** Employees(110,**"Ravi"**,185000.00),  
 **new** Employees(111,**"Ravi"**,185000.00),  
 **new** Employees(112,**"Ravi"**,185000.00),  
 **new** Employees(113,**"Ravi"**,185000.00),  
 **new** Employees(114,**"Ravi"**,185000.00),  
 **new** Employees(115,**"Ravi"**,185000.00),  
 **new** Employees(116,**"Ravi"**,185000.00),  
 **new** Employees(117,**"Ravi"**,185000.00),  
 **new** Employees(118,**"Ravi"**,185000.00),  
 **new** Employees(119,**"Ravi"**,185000.00),  
 **new** Employees(120,**"Ravi"**,185000.00),  
 **new** Employees(121,**"Ravi"**,185000.00),  
 **new** Employees(122,**"Ravi"**,185000.00),  
 **new** Employees(123,**"Ravi"**,185000.00),  
 **new** Employees(124,**"Ravi"**,185000.00),  
 **new** Employees(125,**"Ravi"**,185000.00),  
 **new** Employees(126,**"Ravi"**,185000.00)  
 ];  
  
 **public start\_number**:**number** = 0;  
 **public end\_number** : **number** = 5;  
  
  
 showNext() {  
 **this**.**start\_number** = **this**.**start\_number** + 5;  
 **this**.**end\_number** = **this**.**end\_number** + 5;  
  
 **if** (**this**.**start\_number** >= **this**.**emps**.**length**)  
 {  
 **this**.**start\_number** = 0;  
 **this**.**end\_number** = 5;  
 }  
 }  
}

Code in HTML

<**table align="center"**>  
 <**tr**><**th colspan="3"**>All Employees Information</**th**></**tr**>  
 <**tr bgcolor="yellow" style="font-family**: **'Agency FB'**;**font-size**: 30**px**;**text-align**: **center"**>  
 <**th width="100"**> IDNO</**th**>  
 <**th width="250"**> NAME</**th**>  
 <**th width="150"**> SALARY</**th**>  
 </**tr**>  
  
 <**tr \*ngFor="let *e* of emps |** *slice* **: start\_number : end\_number; let *c* = *count*"**>  
 <**th**>{{ ***e***.**employee\_idno** }}</**th**>  
 <**th**>{{ ***e***.**employee\_name** }}</**th**>  
 <**th**>{{ ***e***.**employee\_salary** }}</**th**>  
 </**tr**>  
  
 <**tr**><**th colspan="3" style="text-align**: **center**;**height**: 50**px"**>  
 <**button type="submit" (click)="showNext()"**> Show Next </**button**>  
 </**th**></**tr**>  
</**table**>

**Example on "push": Adding elements to array dynamically**

Code in Typescript

**export class** FacultyComponent **implements** OnInit {  
  
 **public idno**;  
 **public ename**;  
 **public age**;  
 **public gender**;  
 **public contact**;  
 **public salary**;  
 **public designaion**;  
  
  
 **public employees**:EmployeeInformation[] = [  
 {**"employee\_idno"**:101,**"employee\_name"**:**"Ravi"**,**"employee\_age"**:30,**"employee\_gender"**:**"Male"**,**"employee\_contact"**:9876543210,**"employee\_salary"**:185000.00,**"employee\_designation"**:**"Developer"**},  
 {**"employee\_idno"**:102,**"employee\_name"**:**"Kumar"**,**"employee\_age"**:32,**"employee\_gender"**:**"Male"**,**"employee\_contact"**:9876543211,**"employee\_salary"**:285000.00,**"employee\_designation"**:**"SR-Developer"**},  
 ];  
  
 **constructor**() { }  
  
 ngOnInit(): **void** {  
 }  
  
 **public** add\_employee()  
 {  
 **this**.**employees**.push({**"employee\_idno"**:**this**.**idno**,**"employee\_name"**:**this**.**ename**,**"employee\_age"**:**this**.**age**,**"employee\_gender"**:**this**.**gender**,**"employee\_contact"**:**this**.**contact**,**"employee\_salary"**:**this**.**salary**,**"employee\_designation"**:**this**.**designaion**})  
 }  
}  
  
**class** EmployeeInformation  
{  
 **public employee\_idno**;  
 **public employee\_name**;  
 **public employee\_age**;  
 **public employee\_gender**;  
 **public employee\_contact**;  
 **public employee\_salary**;  
 **public employee\_designation**;  
}

Code in HTML

<**table align="center" width="1000"**>  
 <**tr class="title"**><**th colspan="7"**>Employees Information</**th**></**tr**>  
 <**tr id="one"**>  
 <**th**>Idno</**th**>  
 <**th**>Name</**th**>  
 <**th**>Age</**th**>  
 <**th**>Gender</**th**>  
 <**th**>Contact</**th**>  
 <**th**>Salary</**th**>  
 <**th**>Designation</**th**>  
 </**tr**>  
  
 <**tr \*ngFor="let *emp* of employees"**>  
 <**th**>{{ ***emp***.**employee\_idno** }}</**th**>  
 <**th**>{{ ***emp***.**employee\_name** }}</**th**>  
 <**th**>{{ ***emp***.**employee\_age** }}</**th**>  
 <**th**>{{ ***emp***.**employee\_gender** }}</**th**>  
 <**th**>{{ ***emp***.**employee\_contact** }}</**th**>  
 <**th**>{{ ***emp***.**employee\_salary** }}</**th**>  
 <**th**>{{ ***emp***.**employee\_designation** }}</**th**>  
 </**tr**>  
</**table**>  
  
<**table align="center" border="2"**>  
 <**tr class="title"**><**th**>Employee Registration</**th**>  
  
 <**tr**><**th**><**input type="number" [(ngModel)]="idno" placeholder="Idno"**></**th**></**tr**>  
  
 <**tr**><**th**><**input type="text" placeholder="Name" [(ngModel)]="ename"**></**th**></**tr**>  
  
 <**tr**><**th**><**input type="number" placeholder="Age" [(ngModel)]="age"**></**th**></**tr**>  
  
 <**tr**><**th**>Gender <**input type="radio" name="gen" value="male" checked [(ngModel)]="gender"**> Male  
 <**input type="radio" name="gen" value="female" [(ngModel)]="gender"**> Female  
 </**th**></**tr**>  
  
 <**tr**><**th**><**input type="number" placeholder="Contact No" [(ngModel)]="contact"**></**th**></**tr**>  
  
 <**tr**><**th**><**input type="number" placeholder="Salary" [(ngModel)]="salary"**></**th**></**tr**>  
  
 <**tr**><**th**>Designation<**select [(ngModel)]="designaion"**>  
 <**option**>Developer</**option**>  
 <**option**>Tester</**option**>  
 <**option**>Designer</**option**>  
 </**select**></**th**></**tr**>  
 <**tr**><**th**><**button type="submit" (click)="add\_employee()"**>Save</**button**> </**th**></**tr**>  
</**table**>

**Working With Components**

Components are the main building block for Angular applications.

Each component consists of:

* An HTML template that declares what renders on the page.
* A Typescript class that defines behaviour.
* A CSS selector that defines how the component is used in a template
* Optionally, CSS styles applied to the template.

Steps to create and configure an Angular component.

Prerequisites

To create a component, verify that you have met the following prerequisites:

1. Install the Angular CLI.
2. Create an Angular project. If you don't have a project, you can create one using ng new <project-name>, where <project-name> is the name of your Angular application.

**Creating a component**

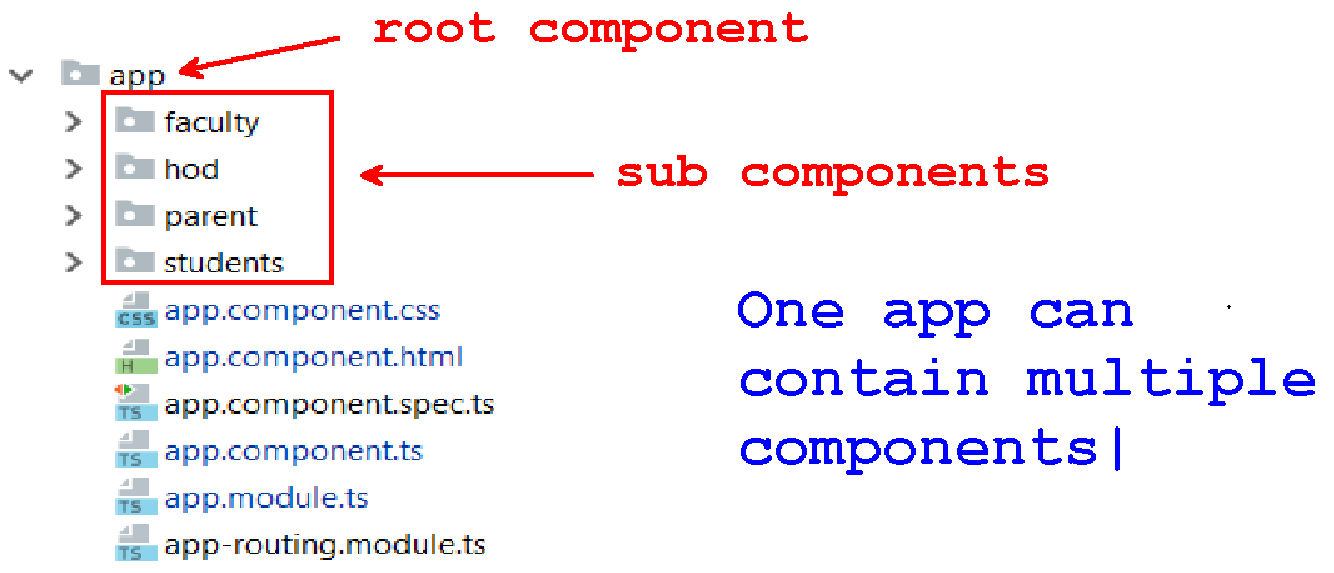
The easiest way to create a component is with the Angular CLI. You can also create a component manually.

Creating a component using the Angular CLI

1. From a terminal window, navigate to the directory containing your application.
2. Run the ng generate component <component-name> command, where <component-name> is the name of your new component.

By default, this command creates the following:

* A folder named after the component
* A component file, <component-name>.component.ts
* A template file, <component-name>.component.html
* A CSS file, <component-name>.component.css
* A testing specification file, <component-name>.component.spec.ts



**Custom Elements**

To write custom elements we need fallow the below steps

1) In app/app.module.ts

**import** {NgModule,CUSTOM\_ELEMENTS\_SCHEMA} **from '@angular/core'**;

2) In app/app/modules.ts

@NgModule({

......

......  
 **schemas**:[CUSTOM\_ELEMENTS\_SCHEMA],  
})

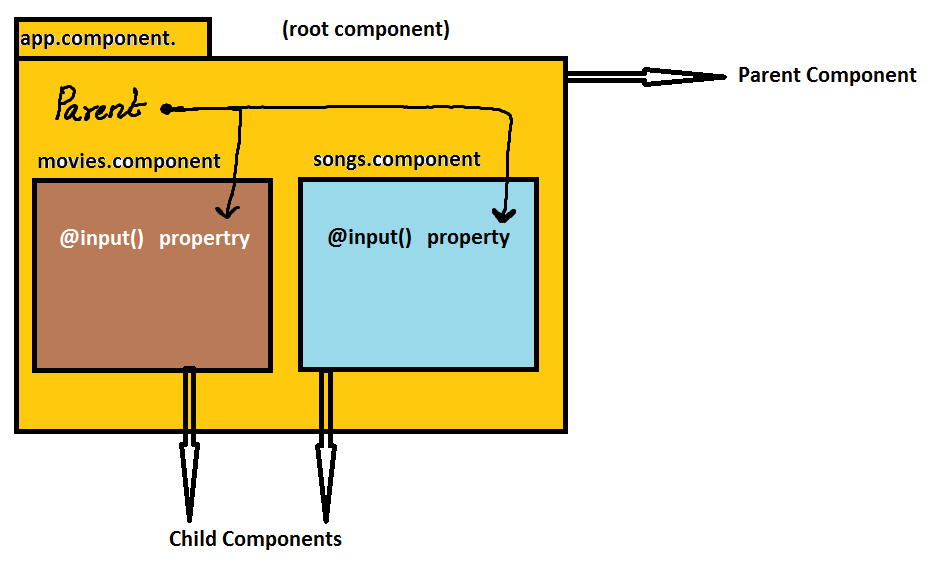
3) In app/app.component.html -- we can write any custom element.

<**pythonwithnaveen**></**pythonwithnaveen**>

**Sending Data From One Component to Other Component**

**1) Sending Data from parent component to child component.**

In child component we use decorator that is **"@input()".**

****

**Example**

**Code in app.component.ts file (root component)**

**export class** AppComponent {  
 **public parent\_name** : **string** = **"Sathya"**;  
}

**Code in app.coponent.html file (root template file)**

<**h1**>I am Parent Component </**h1**>  
  
<**app-movies [parent\_data]="parent\_name"**></**app-movies**>

**Code in movies.component.ts file ( child component)**

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

**export class** MoviesComponent **implements** OnInit {  
  
 @Input() **public parent\_data**;

.............  
}

**Code in movies.component.html file ( child template )**

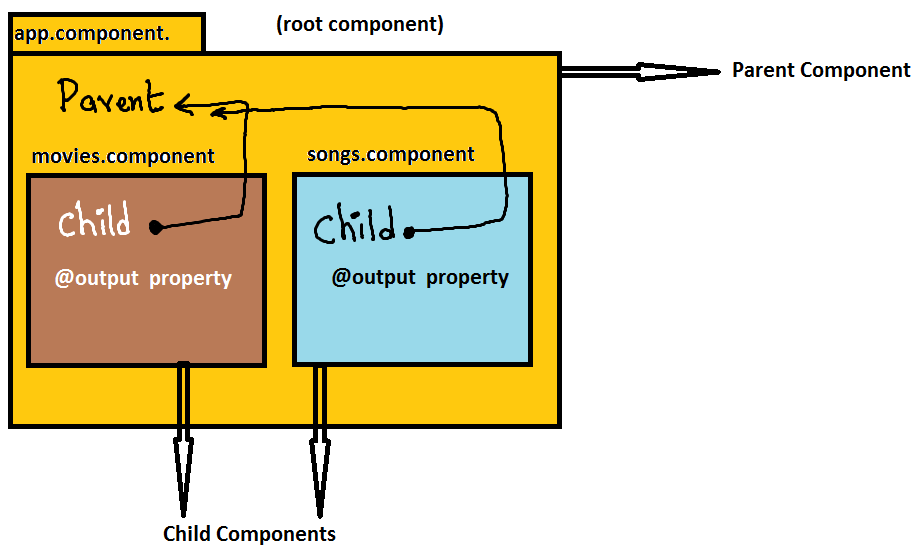
<**h1**> {{ **parent\_data** }} - Movies</**h1**>

**2) Sending Data from child component to parent component**

In child component we use decorator that is **"@output()".**

In child component we use **"EventEmitter"** object to **"emit"** the data.

emit means sending.



**Code in movies.component.ts ( child component)**

**import** {Component, EventEmitter, OnInit, Output} **from '@angular/core'**;  
  
**export class** MoviesComponent **implements** OnInit {  
  
 @Output() **public child\_data** = **new** EventEmitter();  
  
 **public** send\_data()  
 {  
 **this**.**child\_data**.emit(**"Hello I am Movies"**);  
 }  
}

**Code in movies.component.html (Child Template)**

<**button (click)="send\_data()" type="submit"**>Send</**button**>

**Code in app.component.ts (Parent Component)**

**public message**;

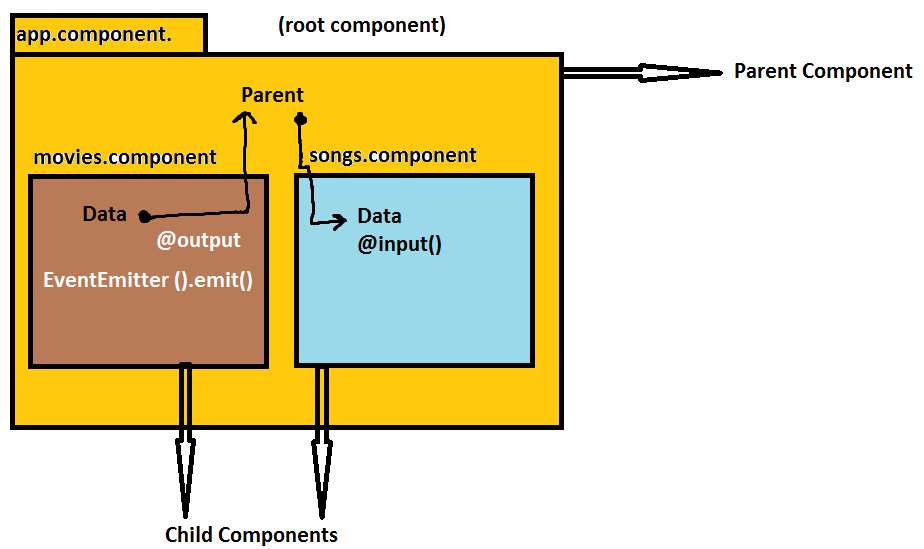
**Code in app.component.html (parent template)**

<**h1**>{{ **message** }}</**h1**>  
  
<**app-movies (child\_data)="message=*$event*"**></**app-movies**>

**3) Sending Data from child component to child component**

To do so we use **@output()** decorator with

**EventEmitter.emit()** function and we use **@input** decorator.



....................................................

....................................................

....................................................

....................................................

**Angular Services**

A service is a **"reusable logic"** that can be shared across the application.

To create a service use the command as

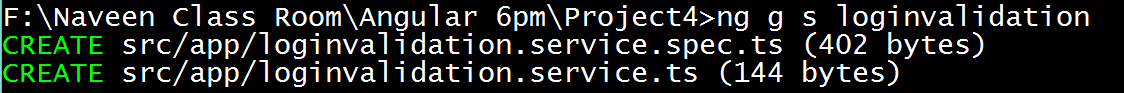
**"ng g s service\_name".**

This command will generate 2 ".ts" files

1) service\_name.service.spec.ts

2) service\_name.service.ts

Note: the **"service\_name.service.ts"** file contains the service  **"reusable logic".**

****

**Example**

**app/loginvalidation.service.ts**

**import** { Injectable } **from '@angular/core'**;  
  
@Injectable({  
 **providedIn**: **'root'**})  
**export class** LoginvalidationService {  
  
 validation(username:**string**,password:**string**):**string** {  
 **if** (username == **"naveen"** && password == **"kumar"**)  
 {  
 **return "valid user"**;  
 }  
 **else**{  
 **return "invalid user"**;  
 }  
 }  
}

**Angular is providing pre defined services like.**

**1) HTTP Client Service**

**2) Activated Route Service**

**3) Router Service.**

**HTTP Client Service**

HTTP Client Service is used to communicate with server to get data from server.

Server is providing the data in "JSON" format.

JSON is a light weight data interchange format.

Using **Http Client Service** we can implement **"Ajax"** calls in Angular.

**Ajax** call means making **Asynchronous call**.

**Synchronous vs Asynchronous**

**Synchronous:** We can send one request at one time and have to wait for the response before send the second request. i.e. It blocks the client browser until operation completes.

**Asynchronous:** We can send other request before getting the response of first request. i.e. It does not blocks the client browser.

**Steps to inject Http Client Service into a component.**

The HttpClient class is providing different methods for different operations.

**Like :**

**request.get();** *for reading 1 or all records from application*

**request.post();**  *for inserting the record*

**request.patch();**  *for updating particular cols in 1 record*

**request.put();**  *for updating all the cols in 1 records*

**request.delete();**  *for deleting 1 or multiple record from application*

**Example on get method**

1) import into aap/app.module.ts

**import** { HttpClientModule } **from '@angular/common/http'**;

2) add into "imports" array in app/app.module.ts

**imports**: [

......

HttpClientModule,  
],

3) Inject into a component

**export class** StatesComponent **implements** OnInit  
{  
 **public res\_data**;  
  
 **constructor**(**private covid\_request**:HttpClient)  
 {  
 **let** url:**string** = **"https://api.covid19india.org/state\_district\_wise.json"**;  
 **this**.**covid\_request**.get(url).subscribe((response\_data:**any**)=>{  
  
 ***console***.log(response\_data);  
 **this**.**res\_data** = response\_data;  
  
 });  
 }  
 ngOnInit(): **void** {  
 }

}

4) Use the response in template

<**table align="center" border="2"**>  
 <**tr**><**th**><**h1**>Sates</**h1**></**th**></**tr**>  
 <**tr \*ngFor="let *x* of res\_data | keyvalue"**>  
 <**td**> {{ ***x***.**key** }}</**td**>  
 </**tr**>  
</**table**>

**Example on post method**

**Template**

<**table align="center" border="2"**>  
 <**tr**><**td**>Add a Product</**td**></**tr**>  
 <**tr**><**th**>

<**input placeholder="PName" [(ngModel)]="n2"**></**th**></**tr**>  
 <**tr**><**th**>

<**input placeholder="PPrice" [(ngModel)]="n3"**></**th**></**tr**>  
 <**tr**><**th**>

<**input placeholder="PQuantity" [(ngModel)]="n4"**></**th**></**tr**>  
 <**tr**><**th**>

<**button type="submit" (click)="saveData()"**>Save</**button**> </**th**></**tr**>  
</**table**>

**Typescript**

**export class** ProductsComponent **implements** OnInit  
{  
   
 **public n2**;  
 **public n3**;  
 **public n4**;  
  
**constructor**(**private request**:HttpClient) { }  
  
 ngOnInit(): **void** { }  
  
  
 **public** saveData()  
 {  
  
 **let** data = {**"name"**: **this**.**n2**,**"price"**: **this**.**n3**,**"quantity"**: **this**.**n4** };

**let** url = **"http://127.0.0.1:8000/insertproduct/"**;  
  
**this**.**request**.post(url,data).subscribe((result:**any**)=>{  
 ***console***.log(**"saved"**);  
});

============== or ==============

**var** form = **new *FormData***();  
 form.append(**"name"**,**this**.**n2**);  
 form.append(**"price"**,**this**.**n3**);  
 form.append(**"quantity"**,**this**.**n4**);  
  
 **let** url = **"http://127.0.0.1:8000/insertproduct/"**;  
  
 **this**.**request**.post(url,form).subscribe((result:**any**)=>{  
 ***console***.log(**"saved "**);  
 });  
  
 }

}

**Example on post method for Images (or any kind of files)**

**Template**

<**table align="center" border="2"**>  
 <**tr**><**td**>Add a Product</**td**></**tr**>  
 <**tr**><**th**>

<**input placeholder="PName" [(ngModel)]="n2"**></**th**></**tr**>  
 <**tr**><**th**>

<**input placeholder="PPrice" [(ngModel)]="n3"**></**th**></**tr**>  
 <**tr**><**th**>

<**input placeholder="PQuantity" [(ngModel)]="n4"**></**th**></**tr**>  
 <**tr**><**th**>

<**input type="file" [(ngModel)]="n5" (change)="showimage(*$event*)"**>

</**th**></**tr**>  
<**tr**><**th**>

<**img [src]="localimage" \*ngIf="localimage"**>

</**th**></**tr**>

<**tr**><**th**>

<**button type="submit" (click)="saveData()"**>Save</**button**> </**th**></**tr**>  
</**table**>

**Typescript**

**export class** ProductsComponent **implements** OnInit  
{  
   
 **public n2**;  
 **public n3**;  
 **public n4**;  
 **public n5**;  
 **public img**;

**public localimage**;  
  
**constructor**(**private request**:HttpClient) { }  
  
 ngOnInit(): **void** { }

**public** showimage(event:**any**)  
 {  
 **this**.**img** = event.**target**.**files**[0];  
  
 **if** (event.**target**.**files** && event.**target**.**files**[0]) {  
 **var** reader = **new *FileReader***();  
 reader.**onload** = (event: **any**) => {  
 **this**.**localimage** = event.**target**.**result**;  
 };  
 reader.readAsDataURL(event.**target**.**files**[0]);  
 }  
}

**public** saveData()  
{  
**var** form = **new *FormData***();  
 form.append(**"name"**,**this**.**n2**);  
 form.append(**"price"**,**this**.**n3**);  
 form.append(**"quantity"**,**this**.**n4**);  
 form.append(**"photo"**,**this**.**img**);  
  
 **let** url = **"http://127.0.0.1:8000/insertproduct/"**;  
  
 **this**.**request**.post(url,form).subscribe((result:**any**)=>{  
 ***console***.log(**"saved with image"**);  
 });  
}}

**Example on delete method**

**Template**

<**table align="center"**>  
 <**tr**><**th**>Delete Operations</**th**></**tr**>  
 <**tr**><**th**><**input type="number" required [(ngModel)]="d1" placeholder="Pno"**></**th**></**tr**>  
 <**tr**><**th**><**button type="submit" (click)="deleteProduct()"**>Delete</**button**> </**th**></**tr**>  
</**table**>

**Typescript**

**export class** ProductsComponent **implements** OnInit  
{  
 **public d1**:**number**;  
 **constructor**(**private request**:HttpClient) { }  
  
 ngOnInit(): **void** { }  
  
 **public** deleteProduct()  
 {  
 **let** url = **"http://127.0.0.1:8000/deleteproduct/"** + **this**.**d1**;  
  
 **this**.**request**.delete(url).subscribe((result:**any**)=>{  
 ***console***.log(**"deleted"**);  
 });  
 }

**Example on Put method (for updating a record)**

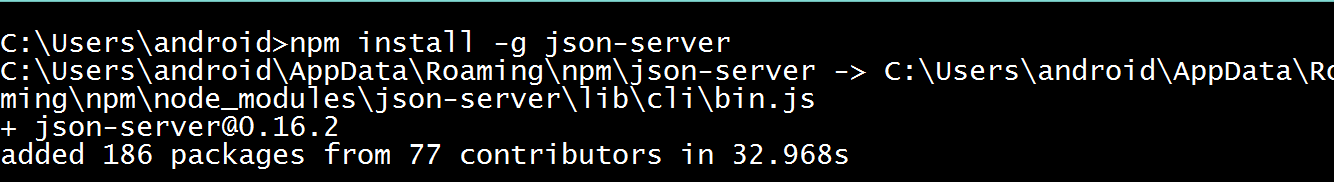
**......................**

**Working with json server**

Website : https://www.npmjs.com/package/json-server

1) Install "json server"

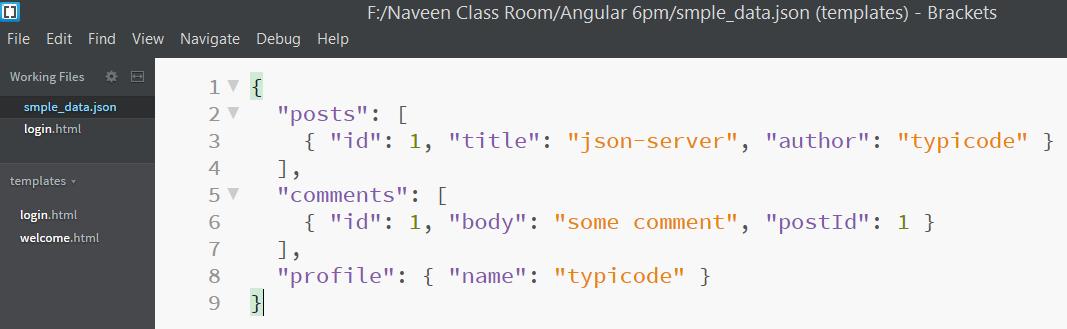
command : npm install -g json-server



2) To check installed or not use type "json-server" in command prompt.

3) prepare a json file and write some json code into it.

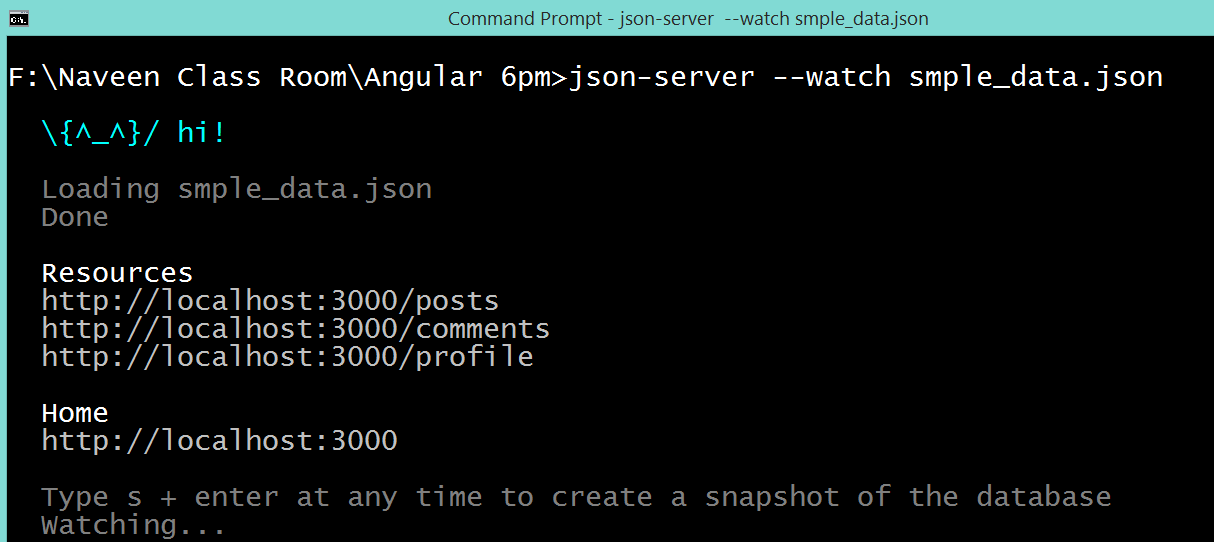
Example



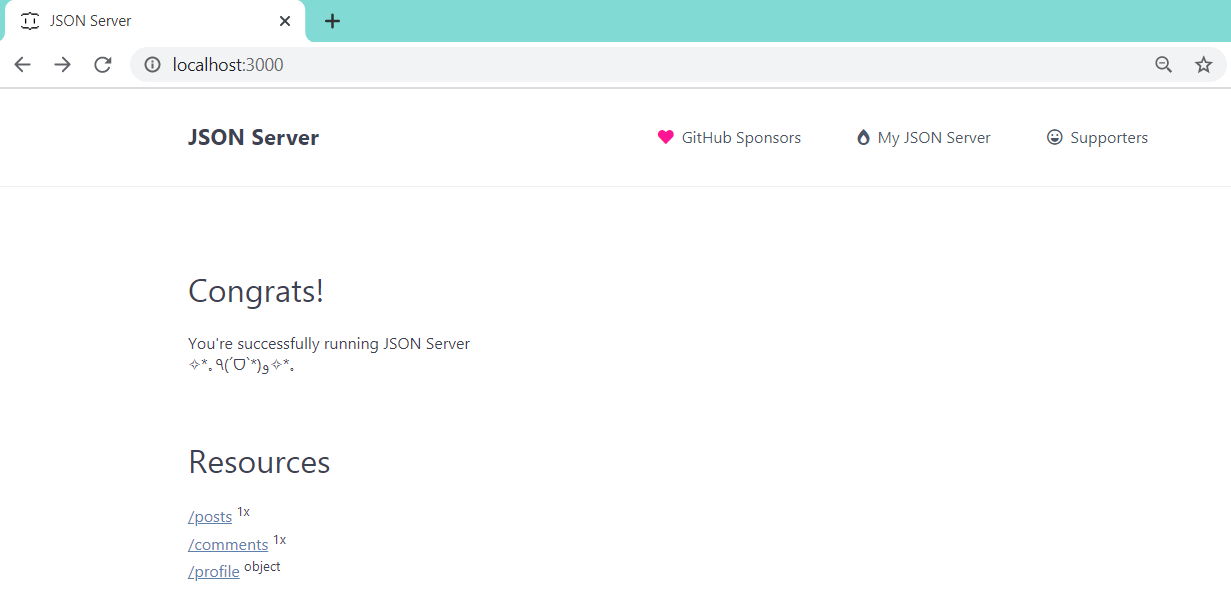
4) To run json-server, open command prompt and move into the file location and type the command as

json-server --watch "file\_name.json"

Example :



5) Open a browser and type the url to get the json result



**Angular FORMS**In angular we can define forms in 2 ways

**1) Template Driven Forms**

In Template Driven Forms we write the complete logic in template

Example : app.componenets.html

<**form #form1="ngForm"** >  
  
<**input minlength="3" #a="ngModel" name="one" type="text" placeholder="Name" required="true" [(ngModel)]="t1"**>  
  
 <**span class="error\_messages" \*ngIf="a.invalid && a.touched"**>Please Enter Name</**span**>  
 <**span class="error\_messages" \*ngIf="a.errors?.minlength"**>Minimum 3 chars </**span**>  
  
<**br**><**br**>  
<**input #b="ngModel" name="two" type="number" placeholder="Age" required min="20" max="60" [(ngModel)]="t2"**>  
  
 <**span class="error\_messages" \*ngIf="b.invalid && a.touched"**>Please Enter Age</**span**>

<**br**><**br**>  
<**button type="submit" (click)="saveDetails()" [disabled]="form1.invalid"** >Save</**button**>  
  
</**form**>

**2) Model Driven Forms**

In Model Driven Forms we write the complete logic in ".ts" file.

The Model Driven Forms are compatible for testing.

The Model Driven Forms are also called as **"Reactive Forms"**.

1) import ReactiveFormsModule into "app.module.ts"

**import** { ReactiveFormsModule } **from '@angular/forms'**;

2) Add imported module into "imports array".

**imports**: [  
 .......,

.......,

ReactiveFormsModule,  
],

3) open "app.component.ts" file

**import** { FormGroup, FormControl, Validators } **from '@angular/forms'**;

**export class** AppComponent {  
  
 **public employeeForm** : FormGroup = **new** FormGroup({  
 **emp\_idno** : **new** FormControl(**null**, [Validators.*required*, Validators.*min*(100)]),  
 **emp\_name** : **new** FormControl(**null**, Validators.*required*),  
 **emp\_salary** : **new** FormControl(**null**, Validators.*required*)  
 });  
  
 **public** display()  
 {  
 *alert*(**"oK"**)  
 }  
}

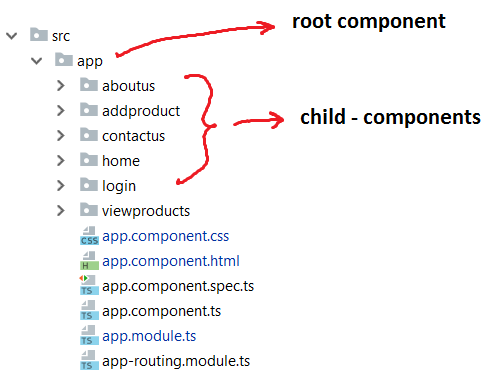
4) open "app.conponent.html"

<**form [formGroup]="employeeForm"**>  
<**table align="center" border="2"**>  
 <**tr**><**th**>Employee Registration</**th**></**tr**>  
 <**tr**><**th**>  
 <**input type="number" placeholder="Idno" formControlName="emp\_idno"**>  
<**span \*ngIf="employeeForm.controls.emp\_idno.invalid && (employeeForm.controls.emp\_idno.dirty || employeeForm.controls.emp\_idno.touched)"**>Idno is Required</**span**>  
 </**th**></**tr**>  
  
 <**tr**>  
 <**th**>  
 <**input type="text" placeholder="Name" formControlName="emp\_name"**>  
 <**span \*ngIf="employeeForm.controls.emp\_name.invalid && employeeForm.controls.emp\_name.dirty"**>Name is Required</**span**>  
 </**th**>  
 </**tr**>  
  
  
 <**tr**><**th**><**button type="submit" (click)="display()" [disabled]="employeeForm.invalid"**>  
 Save  
 </**button**> </**th**></**tr**>  
</**table**>  
</**form**>

**Angular Routing**

Angular Routing is used to switch in between the components .

1) Prepare required no of components.



2) open "app.module.ts" file

**import** {Routes, RouterModule} **from '@angular/router'**;

3) in **"app.modules.ts"** file declare **routers array** with path and component name.

**const** myroutes: Routes = [  
 {**path**: **''**, **component**: HomeComponent},  
 {**path**: **'aboutus'**, **component**: AboutusComponent},  
 {**path**: **'contactus'**, **component**: ContactusComponent},  
 {**path**: **'login'**, **component**: LoginComponent},  
 {**path**: **'view'**, **component**: ViewproductsComponent},  
 {**path**: **'add'**, **component**: AddproductComponent}  
];

4) in **"app.module.ts"** file into @ngModel add the router array variable.

@NgModule({  
 **declarations**: [.............],  
 **imports**: [  
 BrowserModule,  
 AppRoutingModule,  
 RouterModule.*forRoot*(myroutes),  
 ],  
})

5) In "app.component.html" define a menu with hyperlink to switch the components.

<**div id="menu"**>  
 <**nav class="navbar navbar-expand-lg navbar-dark bg-primary"**>  
 <**button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup" aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation"**>  
 <**span class="navbar-toggler-icon"**></**span**>  
 </**button**>  
 <**div class="collapse navbar-collapse" id="navbarNavAltMarkup"**>  
 <**div class="navbar-nav"**>  
 <**a class="nav-link active" [routerLink]="['/']"**>Home <**span class="sr-only"**>(current)</**span**></**a**>  
 <**a class="nav-link active" [routerLink]="['/aboutus']"**>About Us <**span class="sr-only"**>(current)</**span**></**a**>  
 <**a class="nav-link active" [routerLink]="['/contactus']"**>Contact Us <**span class="sr-only"**>(current)</**span**></**a**>  
 <**a class="nav-link active" [routerLink]="['/login']"**>Login <**span class="sr-only"**>(current)</**span**></**a**>  
 <**a class="nav-link active" [routerLink]="['/view']"**>View Products <**span class="sr-only"**>(current)</**span**></**a**>  
 <**a class="nav-link active" [routerLink]="['/add']"**>Add Product <**span class="sr-only"**>(current)</**span**></**a**>  
 </**div**>  
 </**div**>  
</**nav**>  
</**div**>

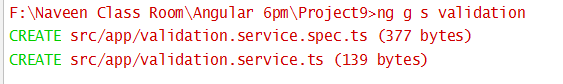
6) To view the components we need <router-outlet> element

<**router-outlet**></**router-outlet**>

**Authentication**

To provide authentication we need to fallow some steps

1) Create a service



2) Service logic.

a) Your service class must implement "**CanActivate**" interface.

b) The **"CanActivate"** interface contains one unimplemented method that is **"canActivate()".**

c) Your service class must implement the **"canActivate()"** method.

d) know write your service logic into this **"canActivate()"** method.

Example

**import** {CanActivate} **from '@angular/router'**;  
  
**export class** ValidationService **implements** CanActivate{  
  
 **constructor**() { }  
  
 **public** canActivate(): **boolean** {  
 **return false**;  
 }  
}

3) open "app.module.ts" and add **"canActivate"** attribute to path.

**import** { ValidationService } **from './validation.service'**;

{**path**: **'add'**, **component**: AddproductComponent, **canActivate** : [ValidationService] },

**Web Storage**

Web Storage is used to store data in client browser.

1) Local Storage

2) Session Storage.

Session storage is temporary, it means once you close the browser the session will be deleted automatically.

a) To write into session.

**sessionStorage.setItem("key","value);**

i) key must be string type and any name you can give.

ii) value can be anything.

b) To read from session.

**sessionStorage.getItem("key")**

i) To getItem we will provide the key so the getItem will return the value.

ii) If the given key is not available the getItem will return **"null".**

c) To delete from session

**sessionStorage.removeItem("key")**

**Angular Material**

https://material.angular.io/

1) Create an Angular project

2) Move into the project in command prompt.

3) Install the material library into an existing project.

**command** : ng add @angular/material

While add material library into the project select the below the points.

a) Select the theme

b) Select the Material (Fonts)

c) Select the browser animation

4) Once the library is installed we can use the material in our application.

https://www.angularjswiki.com/angular/angular-material-icons-list-mat-icon-list/#mat-icon-list-category-editor

**To use the Material in your application**

1) in app.module.ts file

**import** {MatButtonModule} **from '@angular/material/button'**;

**imports**: [  
 MatButtonModule

],

2) in app.component.html

<**button**> Normal Button </**button**>  
  
<**button mat-button**> Mat Button Example </**button**>

**Note :**

1. If 'mat-icon' is an Angular component, then verify that it is part of this module.

2. If 'mat-icon' is a Web Component then add 'CUSTOM\_ELEMENTS\_SCHEMA' to the '@NgModule.schemas' of this component to suppress this message.

3. import {MatIconModule} from '@angular/material/icon';

4. add into the imports array.

**Angular Animation**

1) open "app.module.ts" file

**import** { BrowserAnimationsModule } **from '@angular/platform-browser/animations'**;

2) Add the imported Module into imports array.

**imports**: [

.................  
 BrowserAnimationsModule  
],

3) Open a **"component.ts"** where you need animation.

a) import *state*, *style*, *transition*, *trigger, animate*

b) Inside component decorator add a new array called "animation".

**import** {*animate*, *state*, *style*, *transition*, *trigger*} **from '@angular/animations'**;  
  
@Component({  
 **selector**: **'app-root'**,  
 **templateUrl**: **'./app.component.html'**,  
 **styleUrls**: [**'./app.component.css'**],  
 **animations** : [  
 *trigger*(**'myanimation1'**, [  
 *state*(**'start'**, *style*({ **backgroundColor** : **'red'**, **color** : **'white'**})),  
 *state*(**'end'**, *style*({**backgroundColor** : **'green'**, **color** : **'white'**})),  
 *transition*(**'start => end'**, *animate*(**'2000ms'**)),  
 *transition*(**'end => start'**, *animate*(**'2000ms'**))  
 ]),  
 *trigger*(**'myanimation2'**, [  
 *state*(**'start'**, *style*({ **backgroundColor** : **'blue'**, **color** : **'white'**})),  
 *state*(**'end'**, *style*({**backgroundColor** : **'white'**, **color** : **'black'**})),  
 *transition*(**'start => end'**, *animate*(**'2000ms'**)),  
 *transition*(**'end => start'**, *animate*(**'2000ms'**))  
 ]),  
 *trigger*(**'myanimation3'**, [  
 *state*(**'start'**, *style*({ **transform** : **'rotate(0deg)'**})),  
 *state*(**'end'**, *style*({**transform** : **'rotate(180deg)'**, **opacity** : **'0.1'**})),  
 *transition*(**'start => end'**, *animate*(**'2000ms'**)),  
 *transition*(**'end => start'**, *animate*(**'2000ms'**))  
 ])  
 ]  
})

4) Open **"component.html"** and use **"property binding"** to apply animation to element.

<**button type="submit" (click)="doChange()"**>Animate</**button**>  
  
<**h1 [@myanimation1]=" status "**> Welcome to Angular Animation</**h1**>  
<**h1 [@myanimation2]=" status "**> Welcome to Angular Animation</**h1**>  
  
<**img src="assets/images/admin.png" width="150" height="150" [@myanimation3] = "status"**>

5) in "component.ts" on click change code

**public status**: **string** = **'start'**;  
  
**public** doChange()  
{  
 **if** (**this**.**status** == **'end'**)  
 {  
 **this**.**status** = **'start'**;  
 }  
 **else** {  
 **this**.**status** = **'end'**;  
 }  
}