

In Normal HTML + JS application In Html we write the paragraph and div but in JS we write code to get logic.

**Eg 1 : If we have to show current date and time**

In HTML : Add a div and paragraph where time will be displayed

In JS : we write the code to get date/time. And then get the paragraph DOM element to update that value.

**Eg 2 : If we have a button which refreshes the page**

In HTML : Add Button

In JS : Code function to handle click

In Angular we have Component based approach to develop web application .The Component based approach involves try to figure out what are different portions in this application that we can separate out and create component . i.e each individual segment is created as a component. i.e a header component, a sidebar component, a main area component and a footer component.

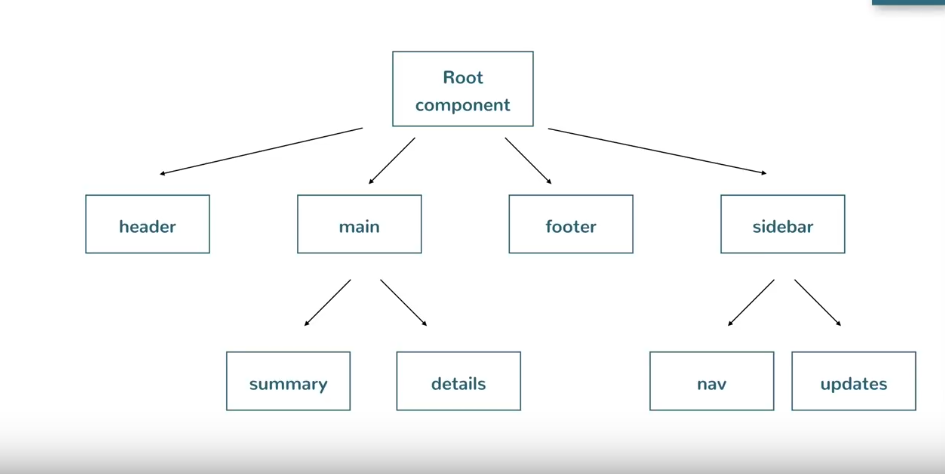
Eg: If we are going to create a Header component it means I am going to create an entity which contains html and js code together. Component in isolation doesn’t mean anything. That should be used/rendered/update. How somebody else uses this component.

* Every component can be assigned a tag or a selector which is basically somebody use this component. The way To do that we assign or we give a name to the component when we are creating . let us say we create a selector **header-section** of a component **header.** A selector is a consumer to use to call and render that component. This selector **“header-section”** can be used as a tag in any html in our angular code and if somebody uses this tag the header is rendered.

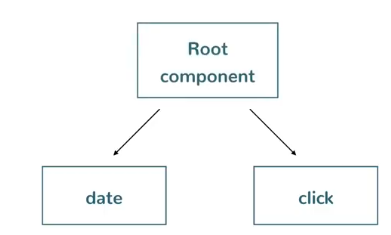
We register all the component in angular with a specific name like **header-section, main-section, side-bar, footer-section.**

Each section can have multiple sub section. like body section can have multiple section(like info-section, details-section etc) and these all subsections are also component and is treated as a tree component of that main component**.**

**Each component has a root component**



So as per our requirement we have :

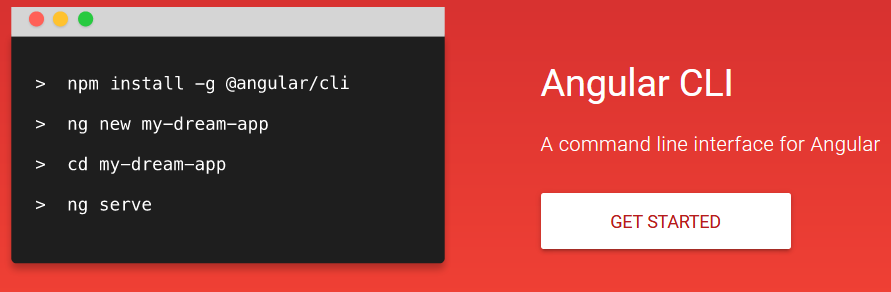


**Angular 6 Setting up :**

**There are three things we need to install to get started with Angular**

1. **Node Js : Node Js is the runtime environment in which we get running on your development tool .**

After installing check : **node –v** and **npm –v** version from cmd prompt

1. **Visual studio code : IDE**
2. **Angular CLI :** A command line interface for Angular. It allows us to generate a building block of Angular application by just typing some commands . It makes the development quicker easier and also its use id best practices.
3. To install Angular CLI run these command
4. 
5. To check installed angular cli version >> **ng –v**

**Hello world Angular Application :**

**ng new my-project :** To create our own project. A new folder named my-project is created in Angular folder

**cd my-project :** to navigate inside the project folder

**ng serve : to** run your application

http://localhost:4200 : this will navigate you to a html page(Hello world )

**Angular What and Why?**

# Angular is framework that is used to create a client side application.

# It is created to build a SPA(Single page application) it means parts of the page refreshed asynchronysly without reloading entire page.

# Angular framework has a modular approach and application build will have a clear structure.

#Re-usable code : Making use of component we can have a lot reusable code

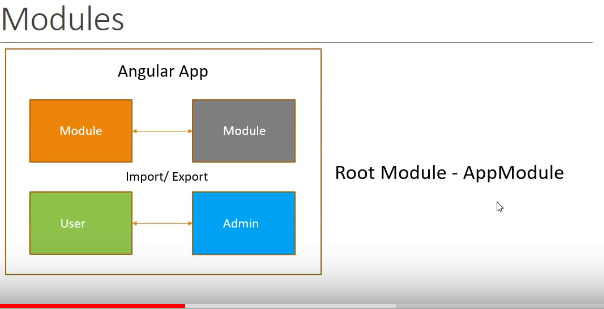
# Development quicker and easier : Angular has lot of capability such as validation ,routing and so on which makes its development quicker and easier

#Unit testable and maintainable code

**Angular Architecture :**

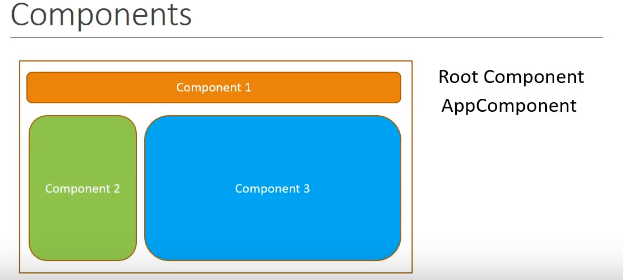
1.Module :

Angular application are modular in nature. Angular application is just a collection of individual modules. Every module represents a feature area on your application. eg : We have a user module that is related to application User and an admin module that is related to application admin. Angular module is just line of code that is imported or exported. Every Angular application has at least one module that is the **root module** and by convention this is called an **Appmodule**. Each module is made up of **component and Services**

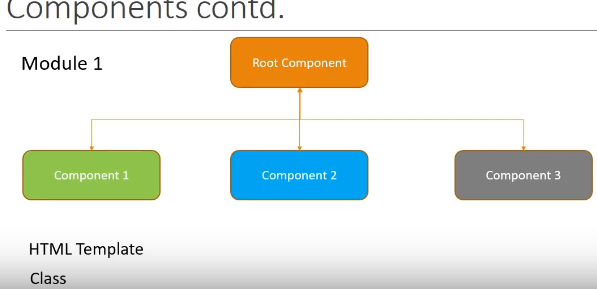


**Components** : A component controls the portion of the view on browser.

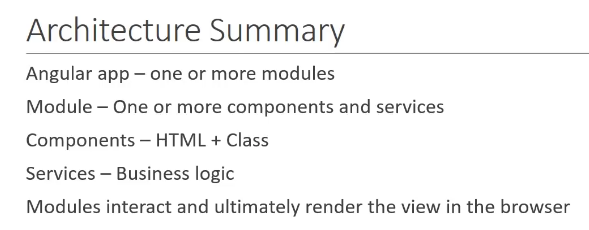
Our Angular application has at least one component called Root component or AppComponent



All components are nested in this root component and each component has an HTML template to represent a view on the browser, and a class that controls the logic of that particular view

\

**Servicses: Module will also have services this is basically a class that contains business logic of our application**

****

**Our newly created application has lot of files out of which we are discussing some of them**

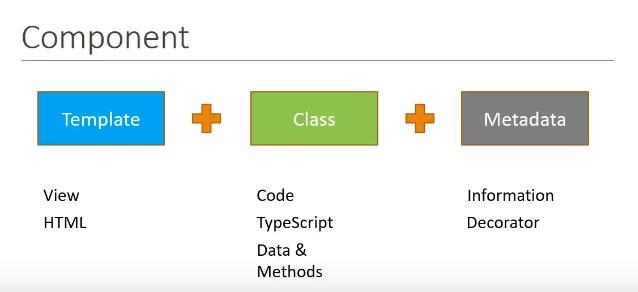
**1.package.json** : This file contains the dependency and the devDependencies which are nothing but library or module that are required for your Angular application to work. the packages listed here will be installed when we run the command **ng new hello-world** and all packages gets installed are in **node\_module folder.** there are lot of scripts are available in **package.json file**

**2.** We have a src folder, inside that src folder we have **main.ts** file which are the entry point to our angular application.

**3.** Under src -> app folder we have **app.module.ts** file which is the root module of our application. and also **app.component.ts** file which is the root component of our application.

When we run the command **ng serve** to start our application the execution comes to **main.ts** file. over here it bootstrap or quick start the **AppModule(corresponds to app.module.ts)** , in **app.module.ts** file it bootstrap or quick start **appComponent(corresponds to app.component.ts)** and this **appComponent** has two things html(**app.Componet.html**) template for view and the class to control the view logic(**app.Component.ts**) file. In this **app.Component.ts** file there is a property defined title which is bound in **app.Componet.html………**

**Component : It is made up of three parts**

****

**1. Teamplate :** it represents the view . this is created using html

**2. Class :** It contains the code that supports the view. this is created using typescript. The class like any other programming language contains data properties and method contains the logic to control the view. eg : We have a methods to show or hide a value based on the properties

**3.Metadata:** This is the information that angular needs to decide if a particular class is in fact an angular component or just a regular class. The metadata is defined using decorator which is a feature in typescript. Decorator is just a function which provides the information about a class attached to it. And for component we use component decorator

**app.component.ts:**

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

@Component({ ---------Metadata which decides that this is component decorator.ie angular class

selector: 'app-root', // Selector is to define the html name for which this class is designed

templateUrl: './app.component.html', //points to a html file representing view for this component

styleUrls: ['./app.component.css'] //Css file applied for this component

})

export class AppComponent {

title = 'myAng';

}

**In our index.html file :**

<body>

<app-root></app-root>

</body>

And this **app-root** selector is the name to render **the app.component.html file (**as defined in templateUrl for app-root selector is app.component.html)

To generate a new component we run this command **: ng g c myComponent**

**g for -> generate**

**c for -> component**

on running this command 4 new files are created **and an update on app-module.ts file.**

**D:\Angular\myAng>ng g c test**

**CREATE src/app/test/test.component.html (23 bytes)**

**CREATE src/app/test/test.component.spec.ts (614 bytes)**

**CREATE src/app/test/test.component.ts (261 bytes)**

**CREATE src/app/test/test.component.css (0 bytes)**

**UPDATE src/app/app.module.ts (388 bytes)**

In the app folder a new folder named **test** is created automatically which is responsible for this component. within that test folder 4 new files are created . **test.component.spec.ts** is created for testing but we don’t need it now so we just delete it. even we notice our component name is test component is just added with all file. This is the naming convention for Angular and angular CLI.

Note : Whenever a new component is added our application should be aware of it. So in **app.module.ts** we import the TestComponent and be added in **declarations array** . (this all works are done automatically by Angular itself). The declaration array contains all the components used by application.

In **app.module.ts :**

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

import { TestComponent } from './test/test.component';

@NgModule({

declarations: [

AppComponent,

TestComponent

],

**Remember** : our Angular application will have one root component called **app component** and all the other components will fall under this app component , that’s why we include **app-test** component inside app component html.

app.component.html

<div style="text-align:center">

<h1>

Welcome to {{ title }}!

</h1>

<app-test></app-test>

</div>

Test.component.html :

<p>

test works!

</p>

On running the application using **ng serve or npm start**

**Welcome to Hello world -------**

**test works!**

**Different ways of changes in a component :**

**1. Selector specification :** There are three ways to specify a selector

**a.** We use a selector as **app-test** and use it as a custom tag in html

In test.component.ts

@Component({

selector: 'app-test', ----🡪 created a selector

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

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

})

In app.component.html

<div style="text-align:center">

<h1>

Welcome to {{ title }}!

</h1>

<app-test></app-test> ------🡪 used that selector in html

</div>

**b.We can use selector as a class :** We begin your selector with a dot character and you can use this selector as a class name.

In test.component.ts

@Component({

selector: '.app-test', -------🡪define our selector as a class

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

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

})

In app.component.html

<div style="text-align:center">

<h1>

Welcome to {{ title }}!

</h1>

<div class="app-test"></div> ---------🡪 using this class under div tag

</div>

**C.** **Enclose your selector within square bracket:**

selector: '[app-test]', -🡪 in component.ts

**in html we use app-test as an attribute**

<div app-test></div> ----🡪 used as attribute

Next thing we can change the **templateUrl** property : It points to the file that html contains the html like **test.component.html** ,but it is also possible to specify the template inline meaning in the same typescript file, and for that we use the **template** property so you can change **templateurl** to just **template** and now we can specify the html right here.

@Component({

selector: '[app-test]',

template: '<div>Inline template </div>',

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

})

**But sometime my inline html might span a couple of lines and for that we make use of back tags( ` )**

@Component({

selector: '[app-test]',

template: `<div>

Inline template

</div>`,

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

})

**We can also change styleUrls with styles tags and style in customize way :**

@Component({

selector: '[app-test]',

template: `<div>

Inline template

</div>`,

styles: [`

div{

color:red;

}

`]

})

**InterPolation :** Using Interpolation we bind data from the class to the template and the syntax is {{}}.within curly bracket we have a property or expression. so we can evaluate any javascript property and the result will be displayed in the browser. We can use String concatenation and we can call any method that is defined in a class. Also we can access javascript global variable with the help of interpolation.

If we have requirement to show the dynamic value of name who is logged in, we must have to define that variable in component and under template tag we put it under {{ name }} --🡪 this expression {{}} in Angular is called interpolation

By using interpolation we are asking angular to evaluate the content inside {{ }} and display the value when the component is rendered in the browser.

@Component({

selector: 'app-test',

template: ` <h2>

Welcome {{name}}

</h2>`,

styles: []

})

export class TestComponent implements OnInit {

public name = "Bharat kumar";

constructor() { }

This is the simplest way to bind a data from a class to a template.

**Things We can do in interpolation :**

@Component({

selector: 'app-test',

template: ` <h2>

Welcome {{name}} Deepak

</h2>

<h2>{{2+2}}</h2>

<h2>{{2+2 +name}}</h2>

<h2>{{"Welcome " + name}}</h2>

<h2>{{name.length}}</h2>

<h2>{{name.toUpperCase()}}</h2>

<h2>{{greetUser()}}</h2>

`

,

styles: []

})

export class TestComponent implements OnInit {

public name = "Bharat kumar";

constructor() { }

ngOnInit() {

}

greetUser(){

return “Hello : ” + this.name ;

}

}

**Things We can not do in interpolation :**

1. **Assigning an expression to a variable : {{** a=2+2}}----🡪You cant do this it gives template parsing error because after calculation that value is assigned to a so this assignment not possible in interpolation {{}}
2. **Cannot use java script global variables** like window, screen so on -🡪 gives error

Eg : if we try to find the url of current page **using {{window.location.href}} -🡪**gives error cant read property location of TestComponent.html:9 undefined

So to do it you define it in a class and bind that property in component

<h2>{{greetUser()}}</h2>

<h2>{{siteUrl}}

`

,

styles: []

})

export class TestComponent implements OnInit {

public name = "Bharat kumar";

public siteUrl = window.location.href;

**o/p :** http://localhost:4200/

**Property Binding in Angular :**

# **Attribute vs Property :**

# Attribute and property are not same.

# Attributes are defined by **HTML** but property are defined by **DOM(Document Object Model)**

**#** Attributes is the initialized value in html but property value is the current value defined by DOM. Attributes value can’t changed once they are initialized.

# Properties value however can change

In a class we define a property and bind it with component

public myId = "testId"; // defining a property in class

<input [id]="myId" type = "text" value="Deepak"/> // binding that property to component

Note: The same binding can be done using interpolation {{ }}

<input id="{{myId}}" type = "text" value="Deepak"/>

Then what is the need of property binding.

Answer: **Interpolation can bind only string values**.

If our requirement is to bind a Boolean value. like if we want to make any field disable

<input disabled="false" id="{{myId}}" type = "text" value="Deepak"/>

Here if we change the property disabled = “false” and binding it with interpolation {{}} it will not work so in this case to bind Boolean value we need property binding concept like

<input [disabled]="false" id="{{myId}}" type = "text" value="Deepak"/>

There is an alternate syntax for property binding .instead **of using [] we can use bind-attributeName**

<input bind-disabled="false" id="{{myId}}" type = "text" value="Deepak"/>

**Class Binding to html element in Angular:**

Let us suppose three different classes in **text-success** having color green, **text-danger** class having color red and **text-special** class having font-style italic and in template we have h2 element that have codevolution

styles: [

` .text-success {

color: green;

}

.text-danger {

color: red;

}

.text-special {

color: italic;

}

`

]

**The regular way to apply this class on html element under template :**

<h2 class = "text-success">CodeEvolution </h2>

**To use class binding we need to declare a property and assign to it that class name and in our template we can use [] to bind that class like**

export class TestComponent implements OnInit {

public successClass= "text-success";

**and in template :**

<h2 [class]="successClass"> Codevaluation</h2>

**Q-> what happens when class binding and class attribute both used in the same html element?**

<h2 [class]="successClass" class="text-special"> Codevaluation</h2>

## O/P: Codevaluation 🡪 Here the Text is green but not italic

It means a regular class attribute becomes a dummy attribute in presence of class binding. It means you have to use one of them not both

**Another way to define class binding based on property :**

**We define some Boolean property in class and based on its true/false value the class will be applied**

export class TestComponent implements OnInit {

public hasError= true;

and in Component :

<h2 [class.text-danger]="hasError"> Code Evolution based on Boolean property hasError </h2>

**This is applied with conditionally applied single class but if we want to apply multiple classes we need to use ngClass directive**

<h2 [ngClass]="messageClasses">CodeEvolution using ng directive</h2>

**In our component class :**

export class TestComponent implements OnInit {

public hasError= true;

public isSpecial = true;

public messageClasses = {

"text-success": !this.hasError,

"text-danger": this.hasError,

"text-special": !this.isSpecial

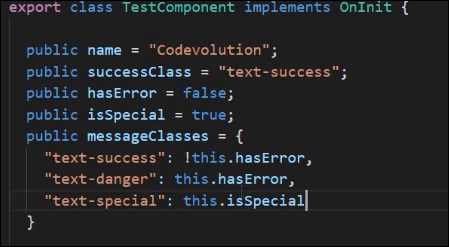
}

}

**Conditionally apply multiple classes:**

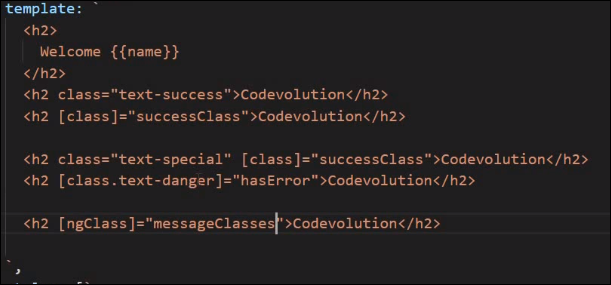
Using ng-directive to apply multiple classes as per given below:

1. create new property and class object as per show below:



used to set the class on the basis of values of class properties hasError & isSpecial

2. using ngClass directive to set class for h2:



**What happens when angular identifies the messageClasses applied to ngClass?**

a. in the object it checks which classes are set to true in above example hasError set to false and isSpecial to true so the text-success and text-special class will be applied.

Note: class binding is useful because it allows you to add or remove classes dynamically to html element based on certain user interaction.

**Style Binding : In angular Style binding is used to apply inline css to html element**

**Method 1 : We are tring bind color property with h2 elememt.**

<h2 [style.color]="'orange'"> Style Binding </h2>

**Method 2 : We can also use ternary operator for style binding :**

<h2 [style.color]="hasError ? 'red' : 'green' "> Style Binding </h2>

export class TestComponent implements OnInit {

public hasError= false;

}

**Method 3 : We can apply CSS with defining a new property**

**In** @Component({

<h2 [style.color]="hasError ? 'red' : 'green' "> Style Binding </h2>

<h2 [style.color]="highLightColor"> Style 2 Binding </h2>

})

export class TestComponent implements OnInit {

public hasError= false;

public highLightColor = "yellow";

}

**Method 4 : We can apply multiple style at a time on any html attribute :**

export class TestComponent implements OnInit {

public hasError= false;

public highLightColor = "yellow";

public titleStyles = {

color : "blue",

fontStyle : "italic"

}

**And to apply this we use ngStyle directive :**

<h2 [style.color]="hasError ? 'red' : 'green' "> Style Binding </h2>

<h2 [style.color]="highLightColor"> Style 2 Binding </h2>

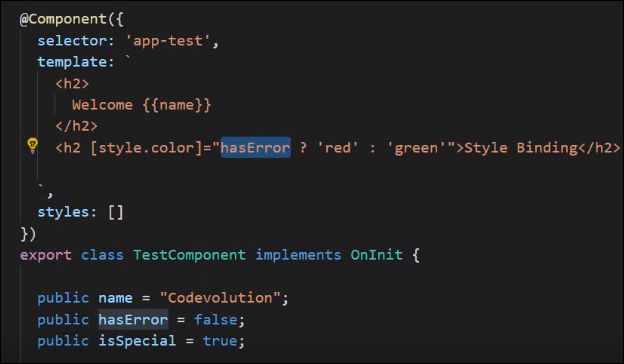
<h2 [ngStyle] ="titleStyles"> Style 3 binding <h2>

**Style Binding: by Pradeep**

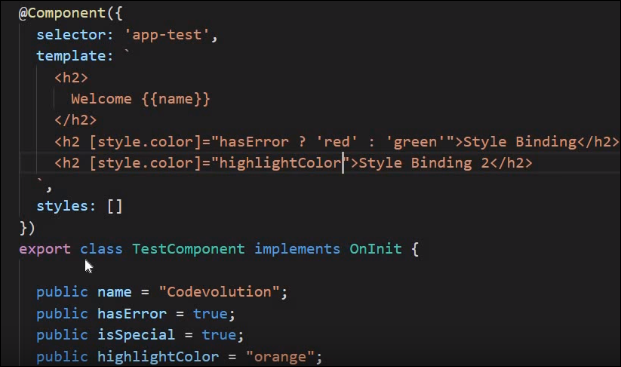
It is used to apply inline style to html elements and it is similar to class binding.



Using conditional operator to bind style as shown below:



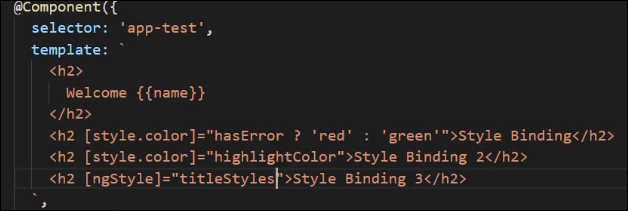
Using component class properties during binding:



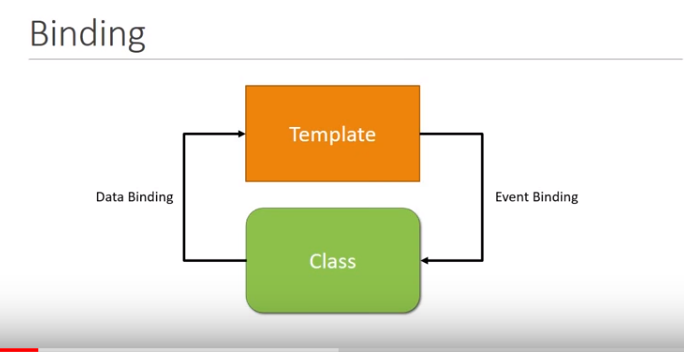
to apply multiple styles use ngStyle directive:



We cannot use - separator here because this are styles so we use camel case



**Event Binding : As far we have seen the data flow from component class to component template.Any update on the class property the view gets updated. But sometimes to respond to user event such as mouse click or keyboard event we need the data flow in the other direction as well ie from the template to the class. So to capture events we may use event bindings.**

****

**Let us suppose an example : We create a button named Greet <button> Greet </button> .**

if user click on this button this should be displayed “Welcome to codeevolution” we need to listen on click event on this button

To do so within the opening tag of event we start with a dom event on which we want to listen to …..

for example we want to listen on click event. On this click event lets define a event handler **onClick()** which is defined in Component class

@Component({

selector: 'app-test',

template: `

<h2>

Welcome {{name}}

</h2>

<button (click)="onClick()"> Greet </button>

`,

styles: []

})

export class TestComponent implements OnInit {

public name = “Deepak”;

ngOnInit() {

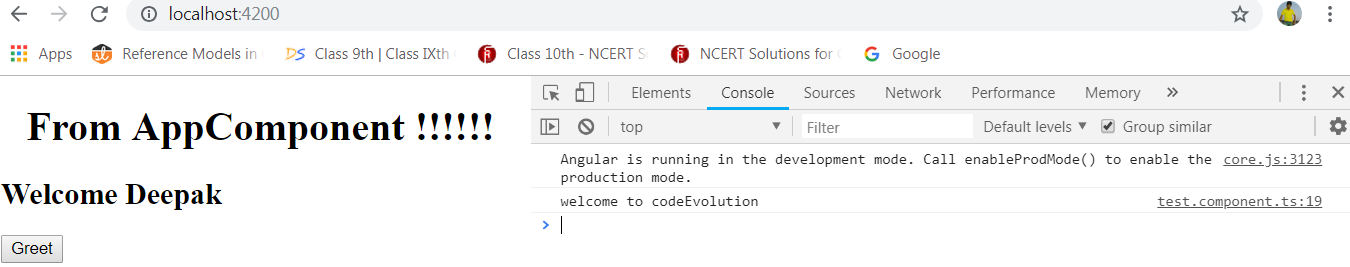
}

onClick(){

console.log('welcome to codeEvolution');

}

}

****

**\*\* It is also possible to set a property on mouse click event**

@Component({

selector: 'app-test',

template: `

<h2>

Welcome {{name}}

</h2>

<button (click)="onClick()"> Greet </button>

{{greetings}}

`,

styles: []

})

export class TestComponent implements OnInit {

public name = “Deepak”;

public greetings = "";

ngOnInit() {

}

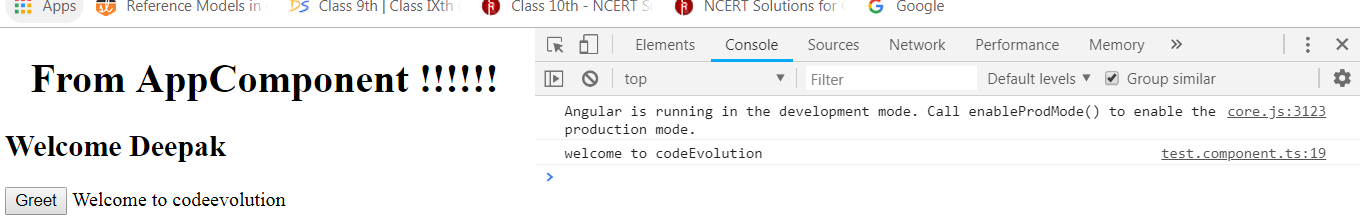
onClick(){

console.log('welcome to codeEvolution');

this.greetings= "Welcome to codeevolution";

}

}

****

**Sometimes we want information about the event itself :**

To do so we simple pass a parameter in the event handler this parameter is **$event and this is a special variable for angular. $event gives you all the information about the dom event that was raised.**

@Component({

selector: 'app-test',

template: `

<h2>

Welcome {{name}}

</h2>

<button (click)="onClick($event)"> Greet </button>

{{greetings}}

`,

styles: []

})

export class TestComponent implements OnInit {

public name= "Deepak";

public greetings = "";

ngOnInit() {

}

onClick(event){

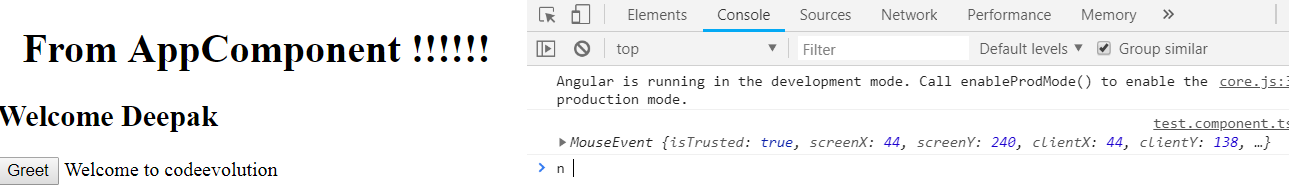
console.log(event);

this.greetings= "Welcome to codeevolution";

}

}

Now on loging event we get the information about that, it is a **mouseEvent**

****

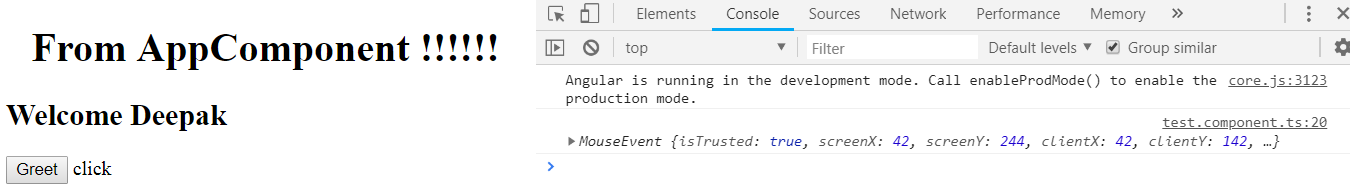
**On expanding we can get all the information that this mouse event captured like X and Y co-ordinate ,target, the type of the event (in this example click event ) and so on. And we can use any of this in our application for example : we use**

onClick(event){

console.log(event);

this.greetings= event.type;

}

**o/p : **

**Sometime if event handler body part is very small : we haveno need to create a separate event handler rather we define the required event handling code in html template only :**

@Component({

selector: 'app-test',

template: `

<h2>

Welcome {{name}}

</h2>

<button (click)="onClick($event)"> Greet </button>

<button (click)="greetings='Welcome to world of Angular'"> Greet Angular </button>

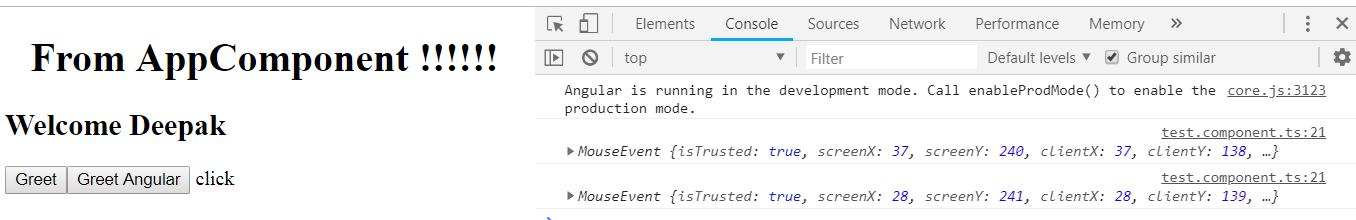
{{greetings}}

`,

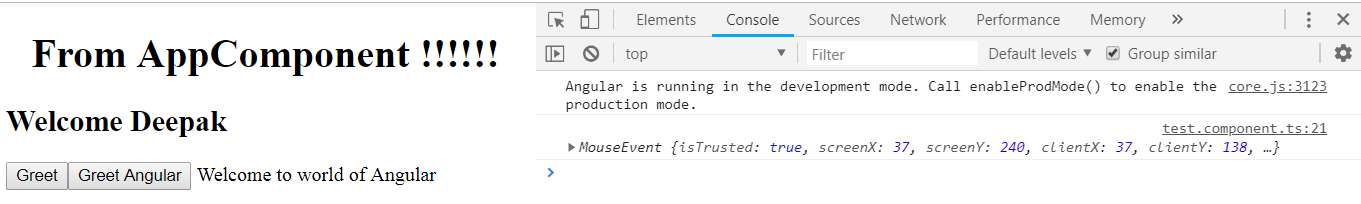
styles: []

})

**On hitting Greet Button :**

****

**On hitting Greet Angular Button :**

****

**Template Reference Variable: Binding from UI to class**

If we want to pass some data to flow from view to the class to perform an operation.eg : We may require some input text field to perform some validation so to easily access dom element and their properties angular provides with **template reference variable.**

@Component({

selector: 'app-test',

template: `

<h2>

Welcome {{name}}

</h2>

<input #myInput type="text">

<button (click)="logMessage(myInput.value)">Log</button>

`,

styles: []

})

export class TestComponent implements OnInit {

public name= "Deepak";

ngOnInit() {

}

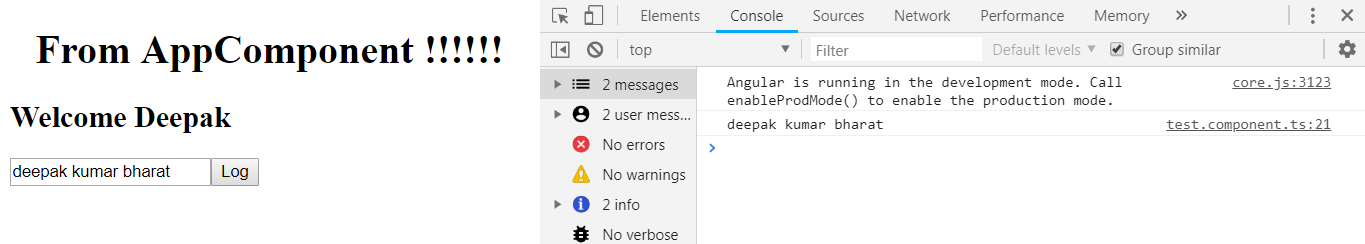
logMessage(value){

console.log(value);

}

}

**O/P :**

****

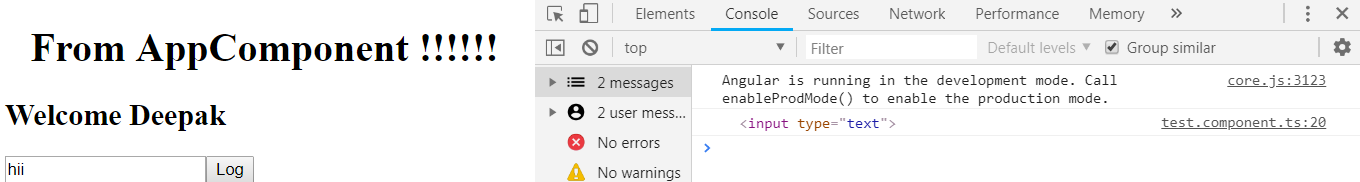
**Note : In the above case if we pass**

<button (click)="logMessage(myInput)">Log</button>

**Instead of**

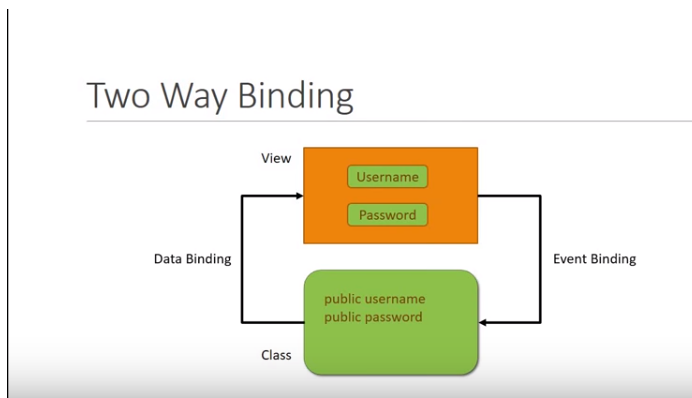
<button (click)="logMessage(#myInput)">Log</button>

**O/p:**

****

* **Meaning only input element is logged not the input value..**
* The reference variable used to refer the html element and to all the dom properties

**Two way Binding : When you work with form inputs it is essential that your model and your view are always in synk otherwise the data might not be consistent.**

****

**Here** there are two things first one is view having two field Username and Password and a component class Having two property username and password if we update some field value from UI it should automatically update its class attribute (event binding) and if we update some data value from model this should automatically update its view page(data binding) .This is called two way binding.Angular has a feature to allow two way binding using **ngModel directive**

<input [(ngModel)]="name" type="text">

**We use [] -🡪for Property binding which is data flow from the class to its template**

**And () 🡺 for the data binding which is a data flow from the template to the class …**

**Here its two way data flow and hence its two way binding :**

@Component({

selector: 'app-test',

template: `

<h2>

Welcome {{name}}

</h2>

<input [(ngModel)]="name" type="text">

{{name}}

`,

styles: []

})

export class TestComponent implements OnInit {

public name= "";

ngOnInit() {

}

}

**On using this we get an error :::::**

compiler.js:1021 Uncaught Error: Template parse errors:

Can't bind to 'ngModel' since it isn't a known property of 'input'. ("

Welcome {{name}}

</h2>

<input [ERROR ->][(ngModel)]="name" type="text">

{{name}}

Reason for this error Angular is not aware of ngModel directive because it is in a separate module called form module so lets add that module to our application.

Open **app.module.ts** : Import @angular/forms

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

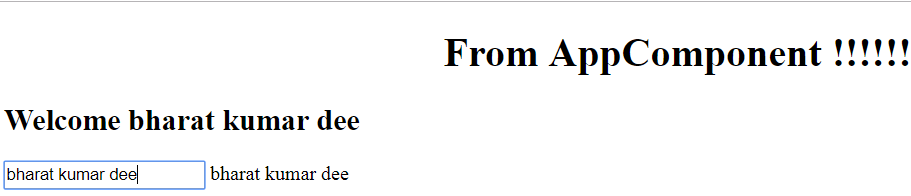
and in imports array add **FormsModule**

imports: [

BrowserModule,

FormsModule

],



**Structural Directive : Structural directive are the directives which will let you add or remove the html elements from the dom.**

**There are three common basic Structural directives are : 1.ngIf 2.ngSwitch and 3.ngFor**

The First two directive ngIf and ngSwitch are conditionally rendered html element whereas the ngFor is rendered a list of html element

**1. ngIf directive :**

@Component({

selector: 'app-test',

template: `

<h2> Welcome {{name}} </h2>

<h2 \*ngIf="displayName">CodeEvolution</h2>

`,

styles: []

})

export class TestComponent implements OnInit {

public name = "Deepak";

displayName ="true";

constructor() { }

ngOnInit() {

}

}

**Toggling using ngIf directive :**

@Component({

selector: 'app-test',

template: `

<h2> Welcome {{name}} </h2>

<h2 \*ngIf="displayName; else elseBlock">

CodeEvolution

</h2>

<ng-template #elseBlock>

<h2> Name is Hidden </h2>

</ng-template>

`,

styles: []

})

export class TestComponent implements OnInit {

public name = "Deepak";

displayName = false;

constructor() { }

ngOnInit() {

}

}

**How it works internally : First it checks is displaName is true then**

<h2 \*ngIf="displayName; else elseBlock">

CodeEvolution

</h2>

**Is executed but if displayName is false then angular checks is there any else statement in our case it is available so it checks in else statement case which html element needs to be rendered.in our case it is elseBlock which is the reference of ng-template tag.**

<ng-template #elseBlock>

<h2> Name is Hidden </h2>

</ng-template>

**is executed**

**Another syntax provided for ngIf directive :**

@Component({

selector: 'app-test',

template: `

<h2> Welcome {{name}} </h2>

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

<ng-template #thenBlock>

<h2> Code Evolution </h2>

</ng-template>

<ng-template #elseBlock>

<h2> Name is Hidden </h2>

</ng-template>

`,

styles: []

})

export class TestComponent implements OnInit {

public name = "Deepak";

displayName = true;

constructor() { }

ngOnInit() {

}

}

**2.ngSwitch : this is used mainly when our requirement is to compare against multiple value.if our requirement is to pick the color red , blue or green**

@Component({

selector: 'app-test',

template: `

<div [ngSwitch]= "color">

<div \*ngSwitchCase=" 'red'"> You Picked Red Color </div>

<div \*ngSwitchCase=" 'blue'"> You Picked Blue Color </div>

<div \*ngSwitchCase=" 'green'"> You Picked Green Color </div>

<div \*ngSwitchDefault> Pick again </div>

</div>

`,

styles: []

})

export class TestComponent implements OnInit {

public name = "Deepak";

public color = "green";

constructor() { }

ngOnInit() {

}

}

**3.ngFor : it is similar to for loop in any programming language the difference is here we render a html instead of executing some logic**

@Component({

selector: 'app-test',

template: `

<div \*ngFor="let color of colors">

<h2> {{color}} </h2>

</div>

`,

styles: []

})

export class TestComponent implements OnInit {

public colors = ["red" , "blue" ,"green","yellow"];

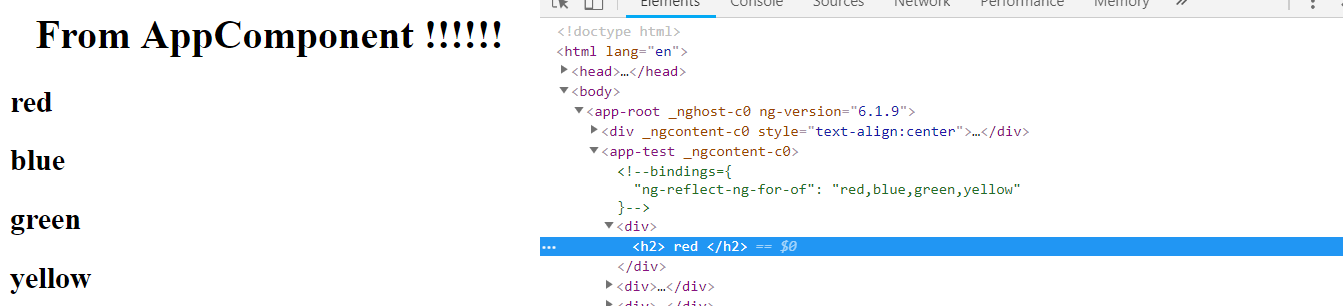
constructor() { }

ngOnInit() {

}

}

Here colors is nothing but a data source having array of colors and we are declaring a variable using **let** **keyword** which is ES2015 feature and variable name is color so color will refer to each item during the iteration



**If we want this value with index :**

@Component({

selector: 'app-test',

template: `

<div \*ngFor="let manohar of colors ;index as i">

<h2>{{i}} {{manohar}} </h2>

</div>

`,

styles: []

})

export class TestComponent implements OnInit {

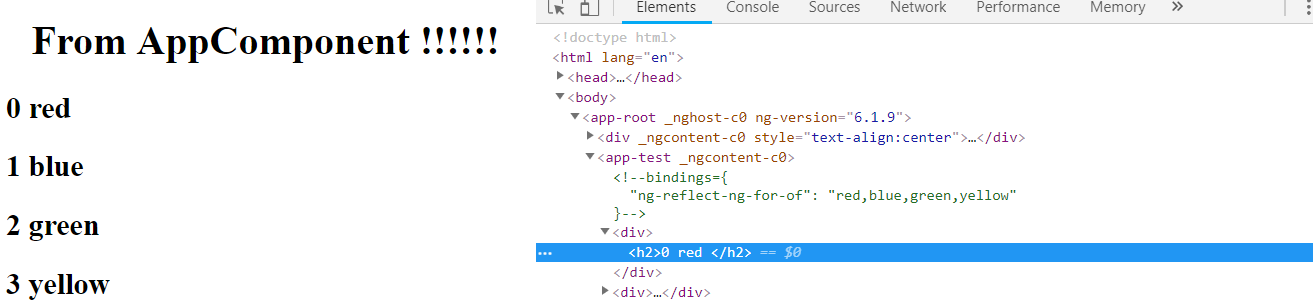
public colors = ["red" , "blue" ,"green","yellow"];

constructor() { }

ngOnInit() {

}

}

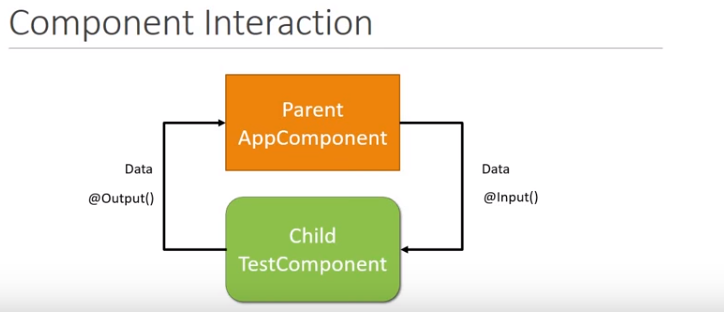


**Component interaction : -**

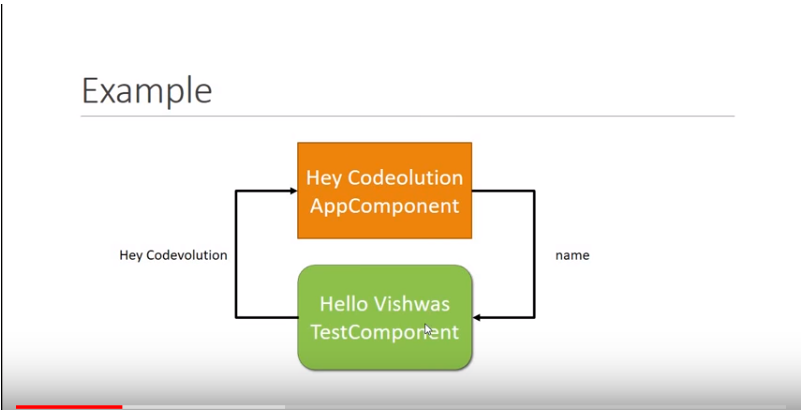
In our application there are two components AppComponent(Parent Component) and TestComponent(Child Component) The Parent component can send some data to child component (**using @Input() decorators**)and the child component can also send some data to parent component(**using @Output() decorators**)

Using @Input() decorator : The child can accept the data from parent component

Using @Output() decorator : The child can send the data to parent component



In our application requirement is we send the name from AppComponent to TestComponent and sending hello name from TestComponent to AppComponent



**Step 1: Sending data from parent to child :----**

**In app.component.ts :** First we define a new property in **AppComponent Class** which is to bind in TestComponent

@Component({

selector: 'app-root',

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

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

})

export class AppComponent {

title = 'structure-directives';

public name = "Deepak";

}

To send this app property to **test-component** we need to add this name property under opening tag of  **<app-test> selector in app.component.html. so we are sending the data from AppComponet to TestComponent.**

<div style="text-align:center">

<h1>

From AppComponent !!!!!!

</h1>

</div>

<app-test [parentData]="name"></app-test>

**Now to receive this sent data into TestComponent :**

**First we need to define that property in TestComponent class under test.component.ts file with that same property name as the appComponent is sending and map that data in template.**

**But We need to inform the TestComponet this is not a normal property rather it is an input property and you will receive this value from the parent . and the way we do that using @Input () decorator.we also need to import it from @angular/core**

**Now we bind this parentData property to template tagusing interpolation {{ }}**

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

@Component({

selector: 'app-test',

template: `

<h2 > {{"Hello : "+parentData}} </h2> `,

styles: []

})

export class TestComponent implements OnInit {

@Input() public parentData;

constructor() { }

ngOnInit() {

}

}

**Note : We can also take an alias name instead of using parentData property like**

@Component({

selector: 'app-test',

template: `

<h2 > {{"Hello : "+name}} </h2> `,

styles: []

})

export class TestComponent implements OnInit {

@Input('parentData') public name;

constructor() { }

ngOnInit() {

}

}

**Step 2 : Data flow from Child component to parent component**

In the previous case the parent component html we have child component selector so we can easily bind data this way but in a child component we don’t have any parent component selector so we cant send the data same way. So the way a child component is sending data to parent is using **event**

**Lets send a message from Test component to Appcomponent and display that into Test component so first create an event there we can send it to AppComponnet**

So in the TestComponent class First create a new instance of **EventEmitter class**

**Public childEvent = new EventEmmiter();**

**And also import that EventEmmiter into @angular/core**

Now to able to send this childEvent to the parent we use the **@Output() decorator**

**@Output() Public childEvent = new EventEmmiter(); and import that decorator also**

**Now our event is set up but to fire this event lets create a button so in template we create a button over here**

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

@Component({

selector: 'app-test',

template: `

<h2 > {{"Hello : "+name}} </h2>

<button (click)="fireEvent()"> Send Button </button> `,

styles: []

})

export class TestComponent implements OnInit {

@Input('parentData') public name;

@Output() public childEvent = new EventEmitter();

constructor() { }

ngOnInit() {

}

fireEvent(){

this.childEvent.emit('Hey CodeEvolution');

}

}

**Now lets capture this event on parent component:**

**In app.component.html we just capture that event in <app-test> selector**

<app-test (childEvent)="message=$event" ></app-test>

Here **$event** variable will go to refer to **“Hey Codeevolution” this is** the String message we want to send on parent component . so now we have a property named **message .we define that property in AppComponnet class and binfd it to html**

**App.component.ts**

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

@Component({

selector: 'app-root',

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

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

})

export class AppComponent {

title = 'structure-directives';

public name = "Deepak";

public message;

}

App.component.html :

<div style="text-align:center">

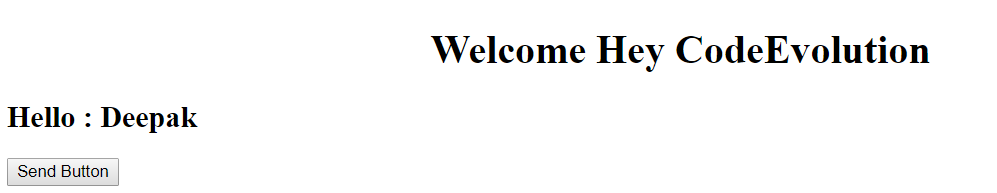
<h1>

Welcome {{message}}

</h1>

</div>

<app-test (childEvent)="message=$event" [parentData]="name"></app-test>

O/P : 

**Pipes in Angular : Pipes allows us to transform data before displaying into the view**

**Example of Some Built in pipe for String :**

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

@Component({

selector: 'app-test',

template: `

<h2> {{ name }}</h2>

<h2> {{ name | lowercase}}</h2>

<h2> {{ name | uppercase}}</h2>

<h2> {{ message | titlecase}}</h2> <!--this pipe converts all first letter in to upper case.-->

<h2> {{ name | slice:3}}</h2> <!-- this starts with 3rd place index started with 0 -->

<h2> {{ name | slice:3 : 5}}</h2> <!-- this starts with 3rd place till 5th(exclusive) of name -->

<h2> {{ person | json }}</h2> <!-- this converts a person into json format -->

`,

styles: []

})

export class TestComponent implements OnInit {

public name = "Codevolution";

public message = "Welcome to codevolution";

public person = {

"firstName": "Deepak",

"lastName" : "Kumar"

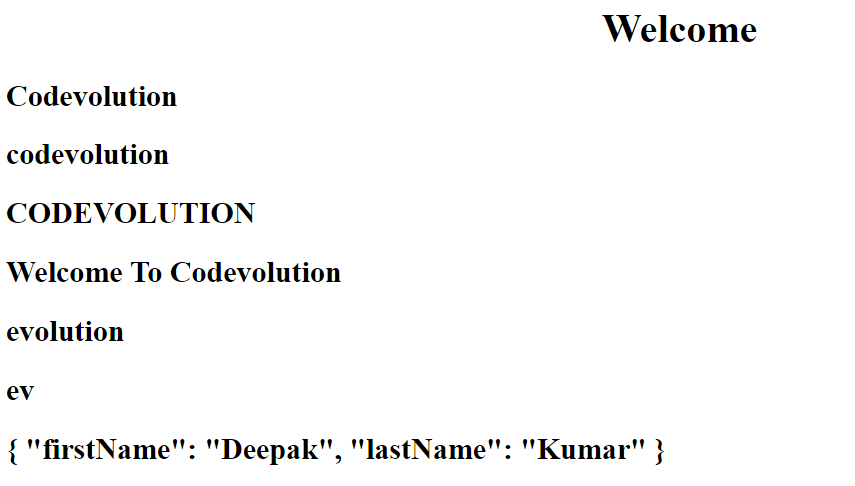
}

constructor() { }

ngOnInit() {

}

}

****

**Pipes applicable for numbers :**

@Component({

selector: 'app-test',

template: `

<h2> {{ 5.678 | number : '1.2-3'}} <!-- it represents minimum 1 digit at integer place and minimum 2 digit at decimal palce and maximum 3 digits at decimal place -->

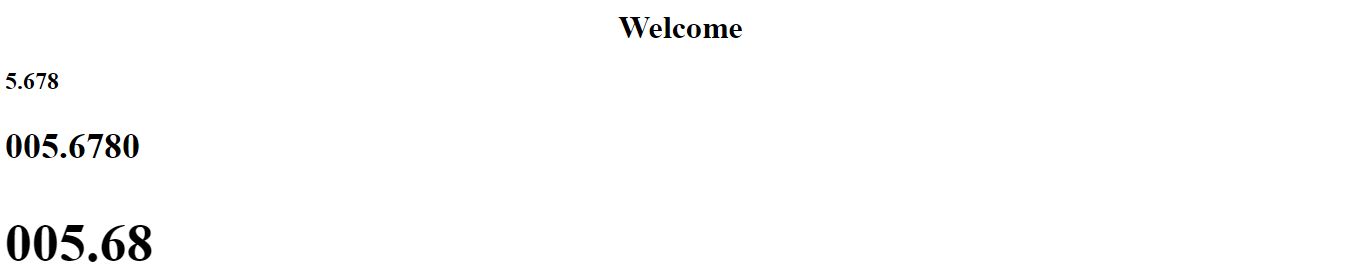
<h2> {{ 5.678 | number : '3.4-5'}}

<h2> {{ 5.678 | number : '3.1-2'}}

`,

styles: []

})

****

**Pipes with Currency :**

@Component({

selector: 'app-test',

template: `

<h2> {{0.25 | percent }}</h2>

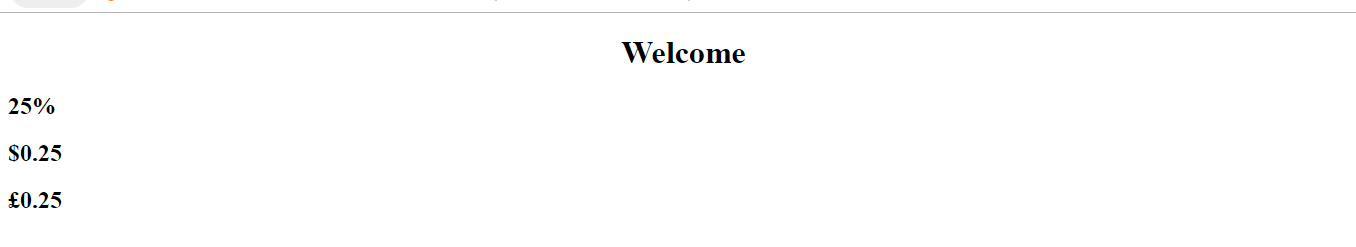
<h2> {{0.25 | currency }}</h2> <!-- by default the currency is us $ but we can change it by passing argument -->

<h2> {{0.25 | currency: 'GBP' }}</h2>

`,

styles: []

})

****

**Pipes with date :**

@Component({

selector: 'app-test',

template: `

<h2> {{ date }}</h2>

<h2> {{ date | date:'short' }}</h2>

<h2> {{ date | date:'shortDate' }}</h2>

<h2> {{ date | date:'shortTime' }}</h2>

<h2> {{ date | date:'medium' }}</h2>

<h2> {{ date | date:'mediumDate' }}</h2>

<h2> {{ date | date:'mediumTime' }}</h2>

<h2> {{ date | date:'long' }}</h2>

<h2> {{ date | date:'longDate' }}</h2>

<h2> {{ date | date:'longTime' }}</h2>

`,

styles: []

})

export class TestComponent implements OnInit {

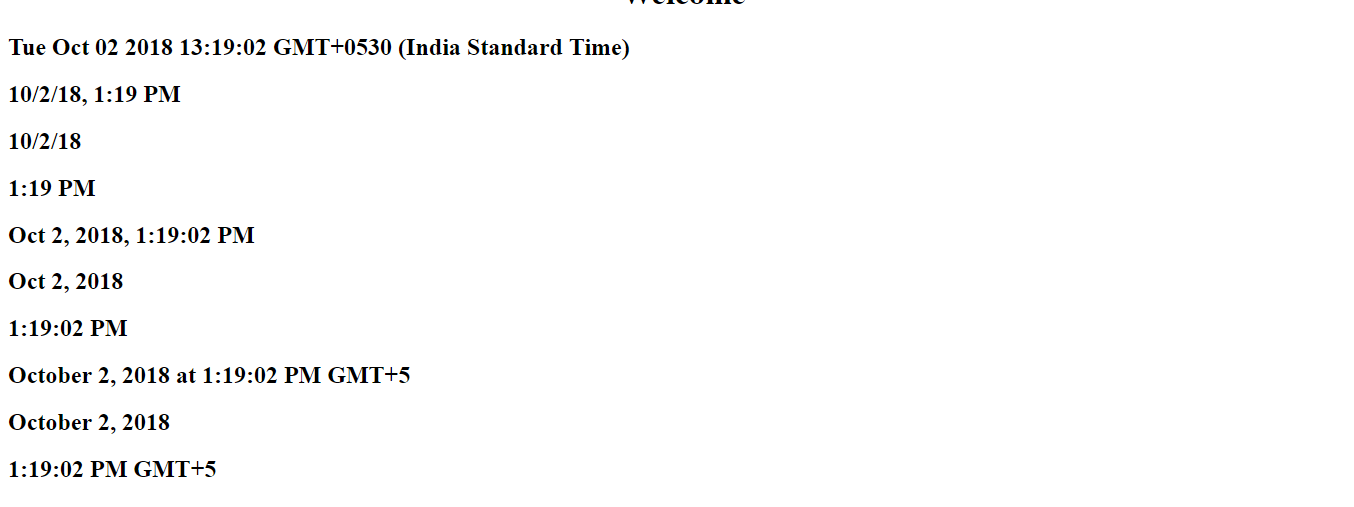
public date = new Date();

constructor() { }

ngOnInit() {

}

}

****

**Important point** : The pipe operator transforms the data only for the view .it doesnot change the value in the property in the class

**Services In Angular :**

Service is a class with specific purpose .

**Why do we need Services?**

1. with the help of service we need to share the Data across multiple component
2. To implement application logic : lets an employee gives his DOB and we need to calculate his age , so we need to write the logic to do that and logic will be independent with any individual component..so in this case we create a the logic in a service and finally we use that logic in any individual component.
3. External interaction : Once logic is defined and implemented we can interact with external component

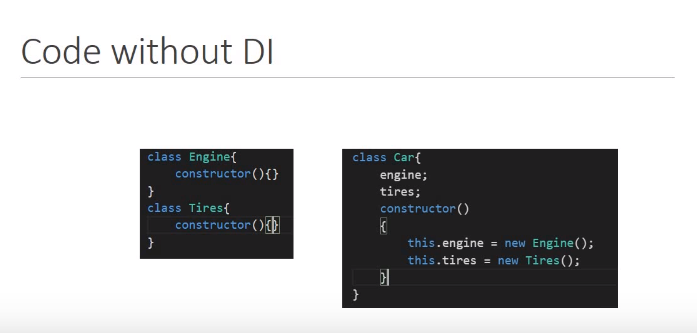
**Naming convention - .service.ts eg:** for employee service employee.service.ts is the service file name and name of class would be EmployeeService

**How we use the Service in Angular?**

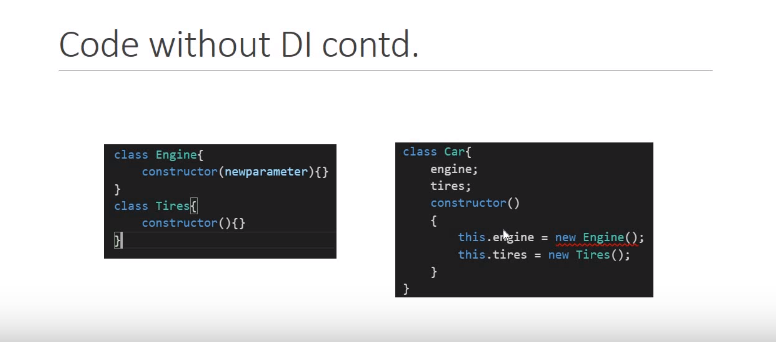
**Ans - Using Dependency injection**

**We are learning DI into three parts :**

1. **Code without DI**
2. **DI as a design pattern**
3. **DI as a framework provided by Angular**
4. **Code without DI :** Let us suppose three classes Engine , Tyres and Car. let us suppose for simplicity To create a car we need only tyres and Engines

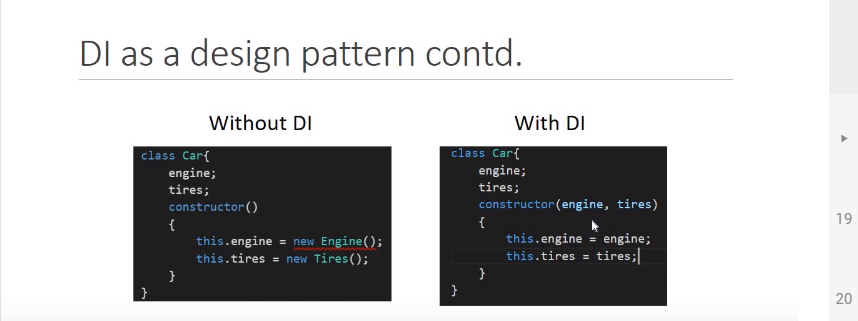
****

**Disadvantage :** Let’s suppose in future if our Engine constructor accepts some parameter like Diesel engine or Petrol engine we will have to change in Car class constructor also otherwise it will give an error when we construct Car class object , similarly if we try to change Tire constructor then also we need to change the Car class constructor . it means we can say that our code is not flexible means any time if dependency change the car class needs to be changed as well.

****

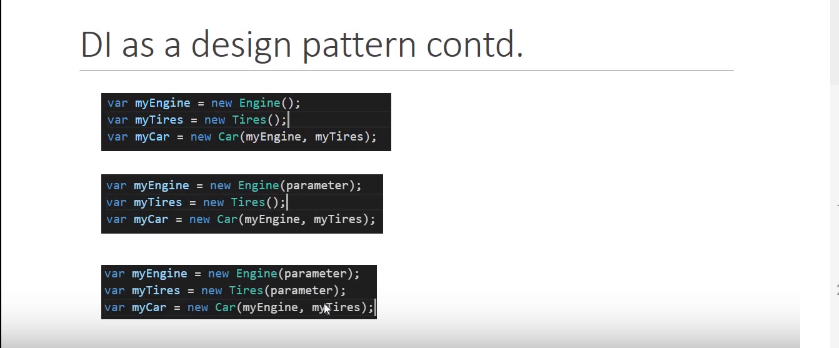
The second drawback is this code is not suitable for testing anytime you instantiate a new car you get same engine and same set of tyres,then how will we teat our code with diesel engine ,petrol engine ,old tyres , new tyres ad so on.. its not possible this can be overcome using DI as a design pattern

**2. DI as design pattern :** DI is coding pattern in which a class receives its dependencies from external sources rather than creating them itself.

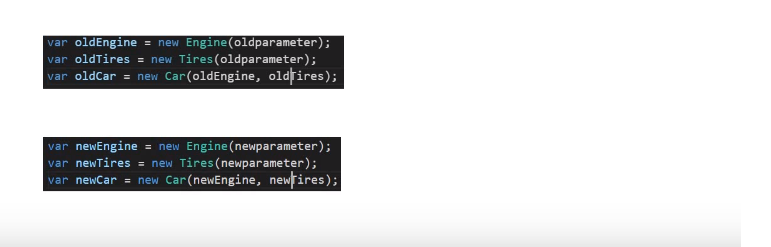


In this example using with DI Car class doesn’t create the instance of engine and tires and it just consumes them. the creation of this dependency is external to this class and by doing so both the drawbacks are now solved.

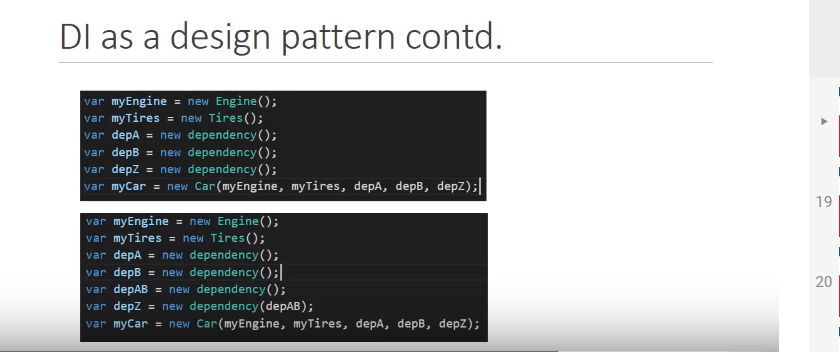
To creation of instance of Car we go by following ways :

In this way first we create an instance of Engine, then tire and lastly we pass both the instances as argument into Car. Even in this case if we have to pass any parameter in Engine instance creation we have no change in instantiating Car . and same for tire also.

**And the same advantage is also for testing purpose like we can create oldEngine,oldTireand with this old set of engine and tire we can create an old Car instance similarly with new set of engine and tire we can create a new instance of Car.**

****

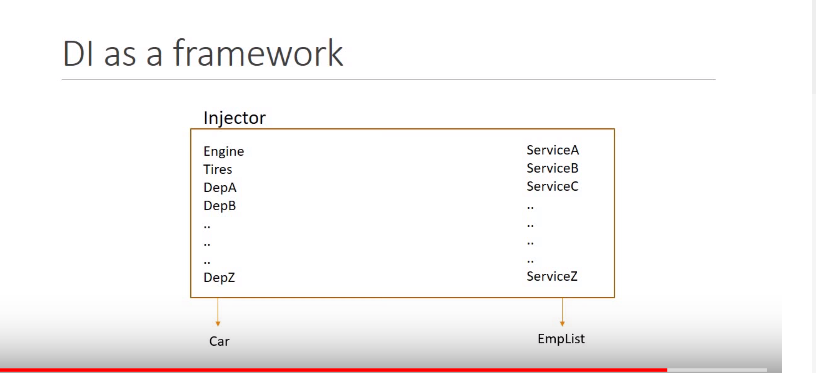
**Disadvantage :** In the above case we had only two dependency to instantiate the Car.so as a developer our first duty is the instantiate all the dependency but if the number of dependency increases then it will be difficult to manage that all … for example



To overcome this overhead Angular introduces DI as Framework

**3.DI as a framework provided by Angular**

DI framework has something called **Injector** where we register all our dependency. so Injector is like a container containing all the dependency like Engine, Tires, DepA, DepB, … DepZ. So If we want to instantiate **Car** we(as a developer) have no need to instantiate any dependent object rather Injector is there to provide all the required dependency

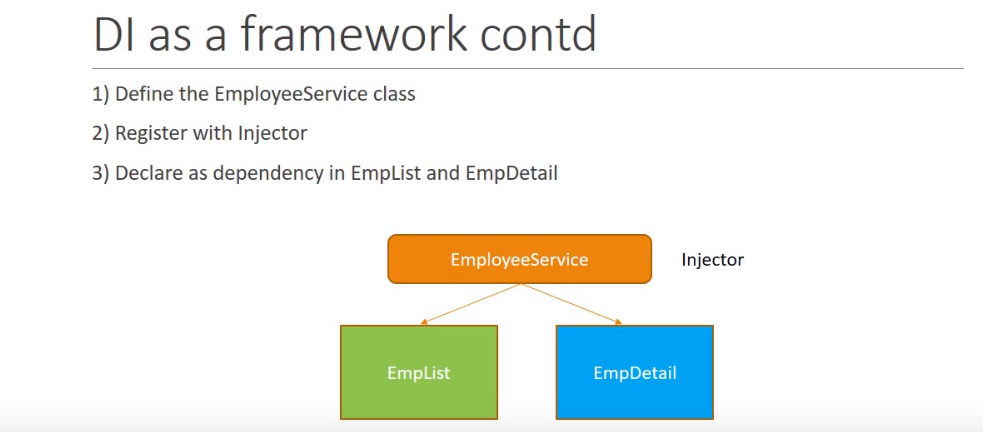
****

**How we use a service in Angular :**

**Step 1 : To create a service like EmployeeService Class**

**Step 2 : Register that service with Angular built in injector**

**Step 3 : Declare that service as the dependency in the class that need it. in our case EmpDetails and EmpList are the classes we need to declare our service class EmployeeService class**

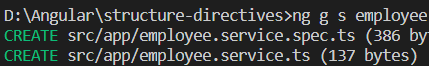
****

**Step 1 : Create an Employee service :** For this we use a command **ng g s employee**

**g 🡪 generate**

**s 🡪 service**

**after running this command a new file is generated**

****

**Employee.service.ts**

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

@Injectable({

providedIn: 'root'

})

export class EmployeeService {

constructor() { }

getEmployees(){

return[

{"id": 1, "name" : "Deepak", "age": 29 },

{"id": 2, "name" : "Sonia", "age": 33 },

{"id": 3, "name" : "Pradeep", "age": 30 },

{"id": 4, "name" : "Pooja", "age": 25 },

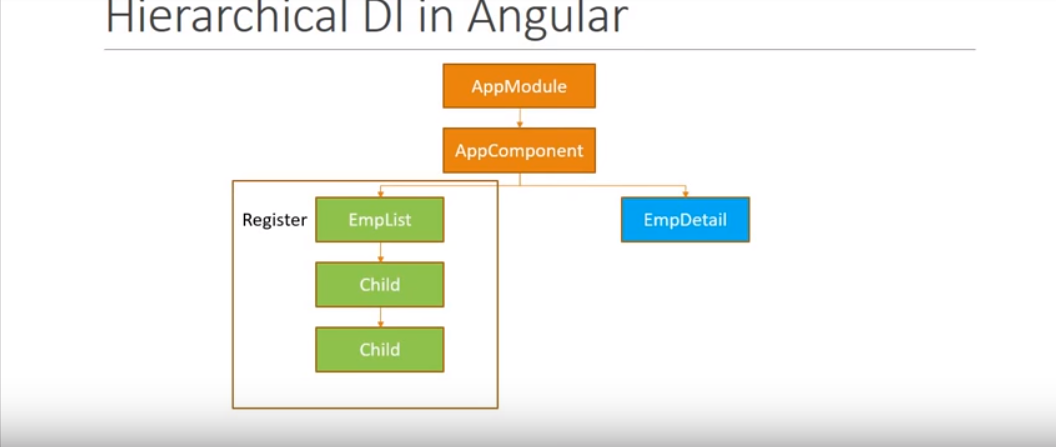
];

}

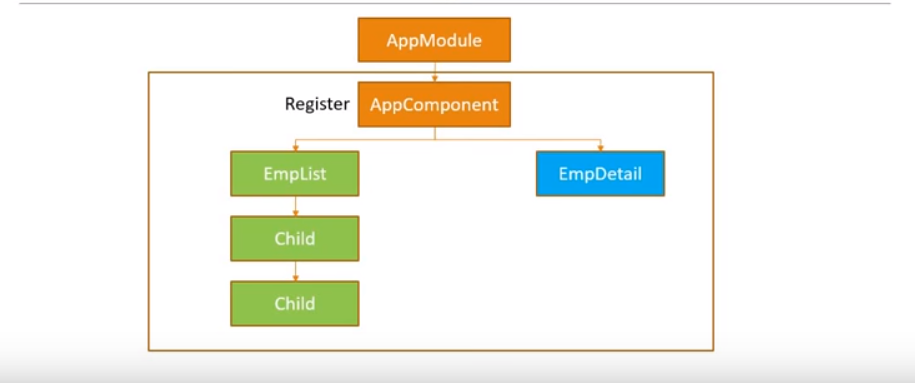
}

**Step 2: Register that service to Injector : There are multiple place to register that service class in Angular but the palce we register is important because Angular has Hierarchical DI system.**

**In our application we have AppModule that has AppComponent and This AppComponent has two components EmpList and EmpDetails.If we register our service in EmployeeList Component ,The service will be used by EmployeeList Componet and its children no other component not ven EmpDetails can use this service**

****

**So this is not s good choice**

**But if it registered with AppComponent then AppComponent as well as all its child can use this Service, so it is a good choice but the best way to register at AppModule.**

**App.module.ts :**

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

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

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

import { TestComponent } from './test/test.component';

import { EmployeeDetailsComponent } from './employee-details/employee-details.component';

import { EmployeeListsComponent } from './employee-lists/employee-lists.component';

import { EmployeeService } from './employee.service';

@NgModule({

declarations: [

AppComponent,

TestComponent,

EmployeeDetailsComponent,

EmployeeListsComponent

],

imports: [

BrowserModule

],

providers: [EmployeeService],

bootstrap: [AppComponent]

})

export class AppModule { }

**Step 3 : Declare that service as dependency wherever it is needed**

**To do that we go to constructor part of different component like :::**

**Employee.details.component.ts :**

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

import { EmployeeService } from '../employee.service';

@Component({

selector: 'app-employee-details',

templateUrl: './employee-details.component.html',

styleUrls: ['./employee-details.component.css']

})

export class EmployeeDetailsComponent implements OnInit {

public employees = [];

constructor(private \_employeeService : EmployeeService) { }

ngOnInit() {

this.employees = this.\_employeeService.getEmployees();

}

}

**Now How to get data of Employee from EmployeeService :**

EmployeeService has a method named getEmployee() that returns employees Data, so in our EmployeeDetailsComponent We have an instance of employeeService (**\_employeeService**) it calls the **getEmployees**() method and returns to **employees** array that belongs to the EmployeeDetailsComponent

**Now let’s repeat the same thing in EmpoyeeList Component :**

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

import { EmployeeService } from '../employee.service';

@Component({

selector: 'app-employee-lists',

templateUrl: './employee-lists.component.html',

styleUrls: ['./employee-lists.component.css']

})

export class EmployeeListsComponent implements OnInit {

public employees = [];

constructor(private \_employeeService : EmployeeService) { }

ngOnInit() {

this.employees = this.\_employeeService.getEmployees();

}

}

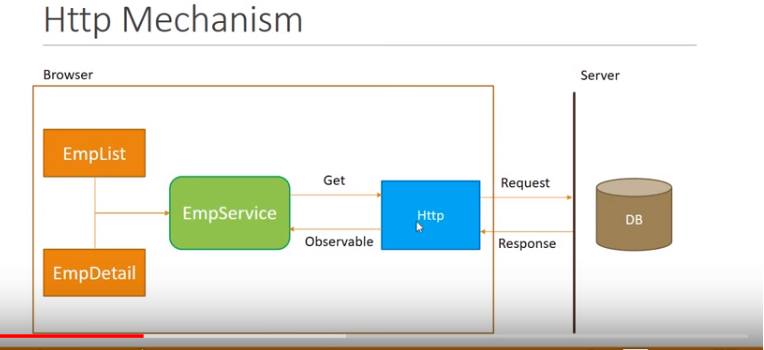
**Now Run the application :**

**O/P :**

**What is the meaning of @Injectable decorator:**

Injectable decorator tells Angular this service might itself have injected dependencies.so if you ever want to inject a service into another service @Injectable decorato is must. Right now EmployeeService diesnot have dependency so @Injectable decorator is not necessary.but employeeService might have dependency in future and that is why we recommended to use @Injectable decorator as soon as we create aservice class and since Angular CLI follows best practices it adds automatically @Injectable decorator when we generate anew Service.

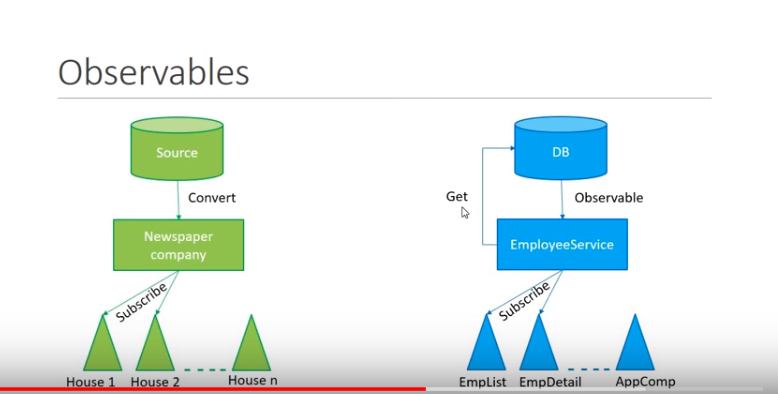
**Http and Observables in Angular : In our current scenario we are fetching the data which is hard coded in our EmployeeService but in real world application data is fetched by calling web servers.**

****

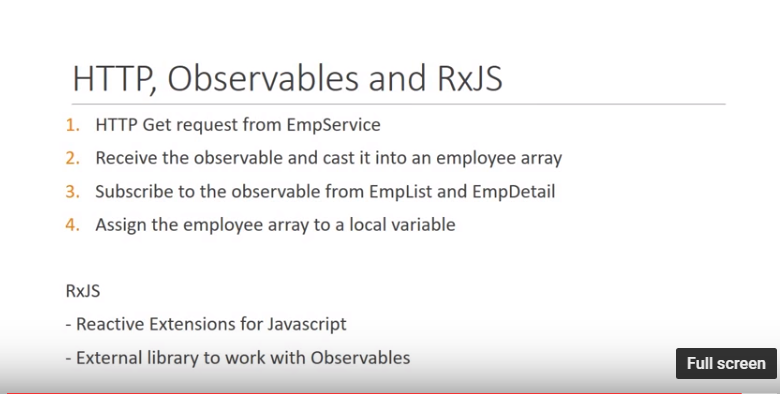
**In this mechanism there is mainly two step :**

1. **sending http request for web service call to get data from Database and**
2. **getting a http response from web server/DB server and response will be back from the http call is an Observable Employeeservice needs to cast this observable into an array of employee and returns the same to EmpList and empDetail components.**

**What is Observables :**

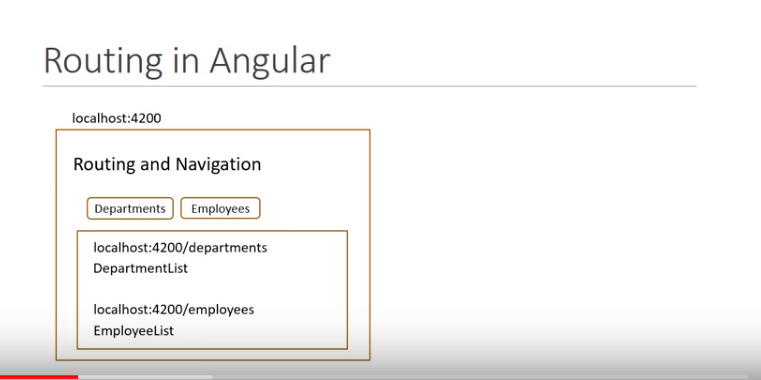
****

**Conclusion :**

****

**Routing and Navigation :**

**How to navigate between two views on button clicks :**

****

**Steps to Routing app:**

**Step 1: Generate a project with routing option :**in terminal we type **ng new routing-demo – routing**

ng new routing-demo -🡪 this commands create a new project

--routing command is used to routing option

Note : we might add routing option with existing application also

**Step 2 : Generate departmentList and employeeList components that needs to be navigate**

**ng g c empl-list –it –is and ng g c dept-list –it –is**

here it is for inline template and –is for inline style

Step 3 : Configure the routes for the application :

In **app.routing-module.ts** file :

We have a constant routes which is of strongly type Routes from the Router package. here we define all possible routes for our application and each route is nothing but an object. Object contains a path which is reflected in the url and the component to be rendered when we navigate to the corresponding path. Now we need to configure two routs for our application the first one is department and second one is employee

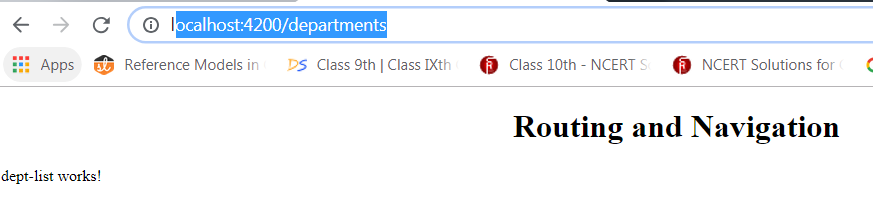
const routes: Routes = [

{path:'departments' , component : DeptListComponent},

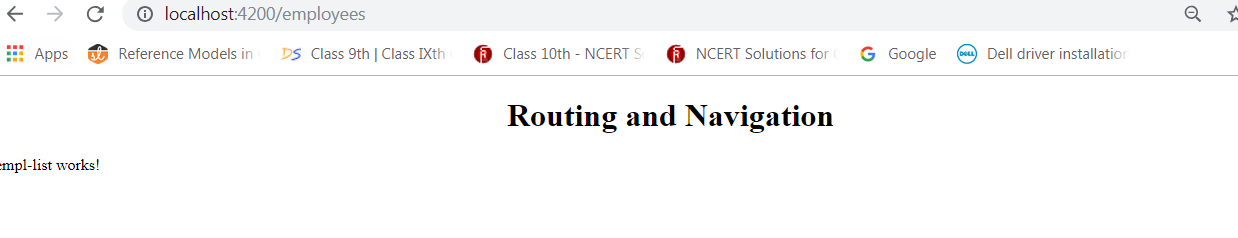
{path:'employees' , component : EmplListComponent}

];

On hitting url



**And**



**Lets add two button to navigate between two views :**

**In app.component.html :**

**We add a <nav> tag and within this <nav> tag two anchor**

**<a> Department</a> and**

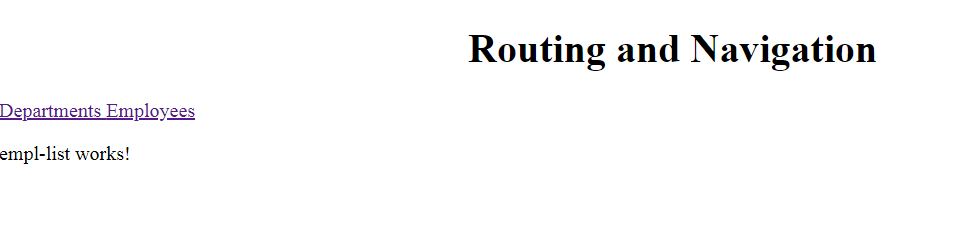
**<a> Employee</a>**

<nav>

<a routerLink= "/departments">Departments </a>

<a routerLink="/employees"> Employees</a>

</nav>

****

**WildCard routes and Redirecting routes :**

In the above case we have configured routing for two url if we try to route any other url apart from above two this will give an error in console **Error: Cannot match any routes. URL Segment: 'test'** obviously it is not favorable. so a better way to handle this invalid urls is using the Wildcard routes .By using this wildcard routes we can navigate user to a 404 page-not-found component if the url doesn’t match with any configured browse.

**To do so we create a new component page-not-found**

**ng g c page-not-found –it –is**

**now on newly created page-not-found.component.ts we add first <h3> Page not found </h3> under template.**

@Component({

selector: 'app-page-not-found',

template: `

<h3> Page Not Found </h3>

`,

styles: []

})

**Now we add the wildcard route :**

**In app.routing.module.ts**

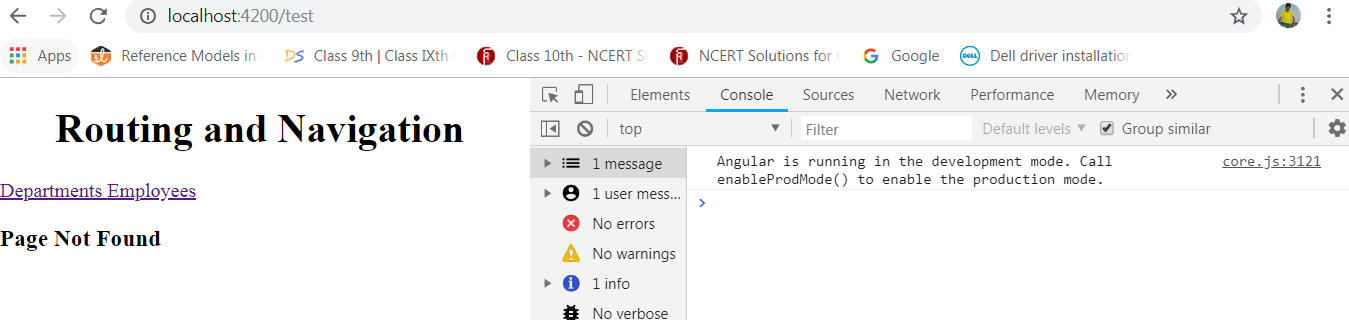
const routes: Routes = [

{path:'departments' , component : DeptListComponent},

{path:'employees' , component : EmplListComponent},

{path: "\*\*" , component : PageNotFoundComponent}

];

****

**Note : Wildcart route will be always be the last route in the configuration it is because the router start to match the path from the top.if the wildcard route will be at the top then it will math with every url and calls PageNotFoundomponent.**

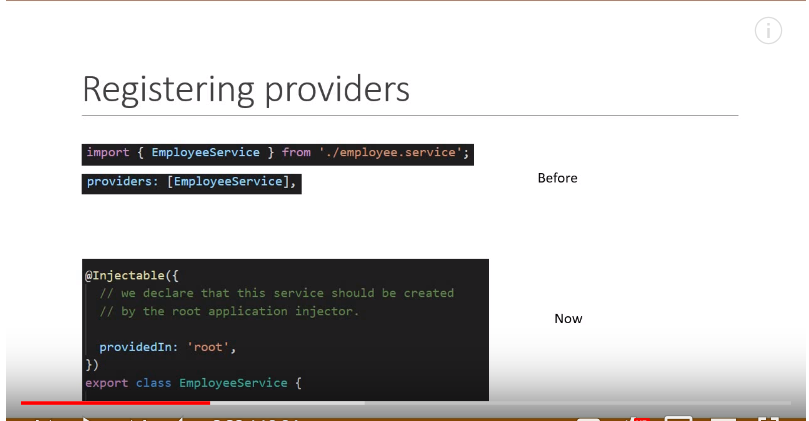
**New features in Angular 6.0.0 release :**

* **Animations:**

**Fisrt all the Animation related code was present in import from @angular/core but now all in import from @angular/animations**

* **Version 6 removes the support of <template> tag instead using <ng-template> tag should be used in html component page.**

**Registering providers :**

****