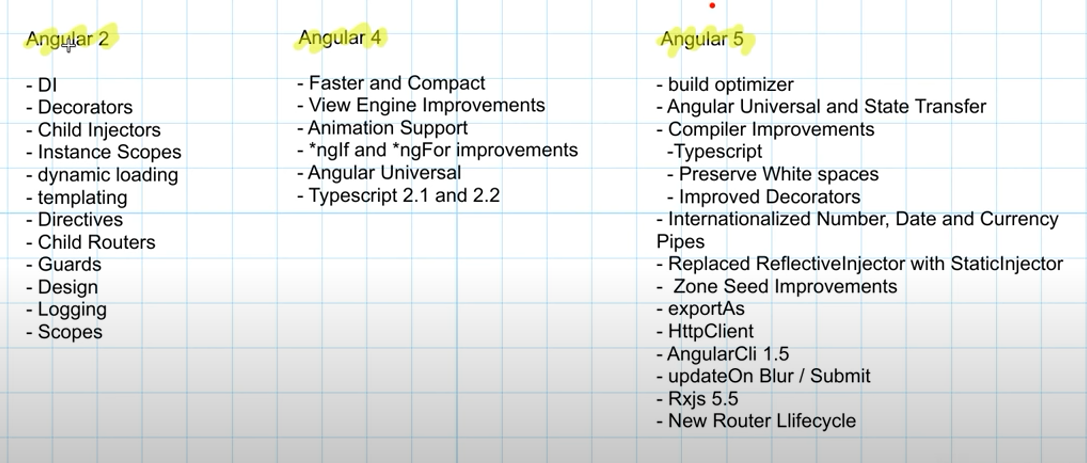
**Angular 2 Notes**

* Is A Free Open Source Component-Based Ui Framework From Google
* Angular 1 Or Angular Js Is The First Version
* Angular 13 Is Latest As Of 2022

Features Of Angular 2

* Angular 2+ Is The Complete Rewrite Of Angular Js ,5 X Faster Than Angular1
* Mobile Oriented
* Cli Support
* Data Bindi
* Can Use Any Of Programing Languages Like Ecmascript5,Ecmascript6,Typescript,Dart,Purescript Etc



**Typescript**

* Free Opensource Programing Language Made By Microsoft
* Extends Javascript By Adding Types To Javascript To Make It Stongly Typed
* Transpilation Is The Process That Compiles Typescript Code To Javascript Code
* Gives Oop Environment For Development
* Features Are :Intellisense,Autocompilation,Strong Typing, Classes,Interface,Inheritance

**Cli**

Command Line Interface

Helps To Create New Angular Application,Components Etc

**Initial Setup Of Angular In Visual Studio Code**

Step 1.

* Install Node 4.6.X Or Greater --Https://Nodejs.Org/En/
* Install Npm 3.X.X Greater -- Npm Install -G Npm
* Install Angular Cli --Npm Install –G @Angular/Cli
* Install Typescript 2.2.0 Greater --Npm Install -G Typescript
* To Get Versions In Command Window Node –V Npm -V

Step 2.

Create A Folder In Your Local Drive, Open Visual Studio Code And Select The Folder

Step 3.

* Create Angular Application And Components
* Ng New Myapp --Creates New Application Called Myapp
* Ng Gc Componentname --Generates A New Component
* Cd Myapp --Navigate To Folder

Step 5.

Ng Serve – To Run The Application

Ng Build – To Build Application

**Deployment**

Ng Buid --After Running, Dist Folder Will Be Created

Ng Build --Prod --In Production

Another Way Using Git Repo

Download Git,

Npm Install -G Angular-Cli-Ghpages

Create New Repo And Add Source Code To Git Repo

Ng Build --Prod --Base-Href="Github  Repo Url"

Angular-Cli-Ghpages  //Will Deploy The Site In Url

**Npm**

Node Package Manager

Helps To Install Any Js Frameworks Like Jquery

All Files Are Savend Under Node\_Module Folder

**Architecture Of Angular**

* View Template
* Components
* Modules
* Bindings
* Directives
* Services
* Di

**Components**

Building Block Of Angular App

Contains Following 3 Parts

* Component Decorator --Adds Metadata To Class Making It A Component Eg, @Component({})

Selector,

Template Or Templateurl --Defines User Interface ,Html, If Html Code >3 Line, Save In External File

Style Or Styleurl

* Class –Contains Code Required For Template Like Variables, Functions Etc
* By Default Root Component Of Angular Application Is Appcomponent

**Module**

Collection Of Components

**Constructor**

Function That Is Called When Instance Of A Component Is Created ,Used For Initializing Variables,Dependancy Injection

Oninit

Lifecycle Hook Method , Called After A Component Is Created,Used For Service Calls

**Bindings In Angular**

* **Interpolation{{}}**
* **Property[]**
* **Event()**
* **2 Way[()]**
* Interpolation
* Property Binding
* Atrribute Binding
* Class Binding
* Style Binding
* Event Binding
* 2 Way Data Binding

Properties And Attributes Are Different In Html.Angular Recomends Property Binding Wherever Possible

**Interpolation** - Moving Data In One Direction

One Way --Component Class Properties To View Template--Interpolation

One Way -- View Template To Component Class Properties

Syntax : {{Firstname}}

      <span>{{interpolationMessage}}</span>

**Property Binding**

Property In Square Brackets, Value In Double Or Single Quotes

Syntax : [Propertyname]=”variable name”

        <Img [src]='propertyBindingImageURL'>

**Event Binding**

-Data Flow From Html Element To Class

Syntax : (Eventnmae)=”Functionname()”

        <input type="button" (click)="ShowEventBinding()" class="btn btn-success" value="Event Binding" />

**2 Way Data Binding**

--Add Reference To FormsModule In Module.Ts.

<Input [(Ngmodel)]='Name'>

<input type="text" class="txt" placeholder="Please type your name" [(ngModel)]="twoWayBindingValue" />

        <span>hello  {{twoWayBindingValue}}</span>

**Attibute Binding**

<Th [Attr.Colspan]="Columnspan">

<Th Attr.Colspan="{{Colspan}}">

**Class Binding**

<Div [Class]="Classname"> --Remove Single Existing Class And Add New Class

<Div [Class.Boldclass]="Isaddclass"> --Add New Single Class To Existing Class

Add Remove Multiple Class

<Div [Ngclass]="Getclass()">

**Style Binding**

<Button [Style.Font.Weight]="Bold">

<Button [Style.Fontweight]="Bold">

<Button [Style.Size.Px]="30">

<Button [Ngstyle]="Getfontstyles()">

**Decorator In Angular**

Makes A Class An Angular Component,Otherwise It Will We A Normal Typescript Class.

To Create Component We Use @Component

To Create Module We Use @Ngmodule

**Directives In Angular**

Changes The Behaviour ,Apperance Or Behaviour Of Html Dom Elements

Angular Syntax Inside Html

3 Types

* + **Structural**-Change Structure Of Html Dom.Add,Remove Or Modify Html Layout

Eg,ngIf,ngFor,ngSwitch

* + **Attribute** –Changes The Apperance Or Behaviour Of Html Elements,No Change In

Structure.Eg [ngClass],[ngStyle],[ngModel],[hidden]

* + **Component**—User Defined Template Or Control Having A Ui,Associated Style And Data

Eg <employee-list>/<employeel-ist>

**Structural Directive**

**\*ngIf**

        <div \*ngIf="ngIfValue" > sample div

**\*ngFor**

  <ul \*ngFor="let item of F1Teams">

            <li>

                {{item}}

            </li>

        </ul>

**\*ngSwitch**

 <div [ngSwitch]="dayNo">

            <span \*ngSwitchCase="0">Hi, today is Monday</span>

            <span \*ngSwitchCase="1">Hi, today is Tuesday</span>

            <span \*ngSwitchCase="2">Hi, today is Wednesday</span>

            <span \*ngSwitchCase="3">Hi, today is Thursday</span>

            <span \*ngSwitchCase="4">Hi, today is Friday</span>

            <span \*ngSwitchCase="5">Hi, today is Saturday</span>

            <span \*ngSwitchCase="6">Hi, today is Sunday</span>

            <span \*ngSwitchDefault="-1"></span>

        </div>

**\*ngForTrackBy**

Keeps Track Of Current Elements By Id Or Code, Later If Data Changes, Only New Elements Added Instead Of Loading Entire Dom Elements

**Attribute Directive**

**[ngClass]**

        <div  [ngClass]="className"> sample div</div>

**[ngStyle]**

        <div  [ngStyle]="style"> sample div</div>

// In ts file

 this.style={

      "background-color": "rgb(226, 131, 43)", "color": "white","height": "50px","width": "150px"

    }

**[(ngModel)]**

<input type="text" [(ngModel)]="ngModelValue" placeholder="Please type here">

        {{ngModelValue}}

        <br />

**Component Directive**

 <app-test > </app-test>

**Pipes In Angular**

--Transform Data Before Display

<Td>{{Employee.Name | Uppercase}}</Td>

<Td>{{Employee.Name | Date:'Fulldate' |Uppercase}}</Td>

<Td>{{Employee.Salary | Currency :'Usd'| }}</Td>

**Component Interaction**

**Communication Between Related Components**

**1.Parent To Child Using @Input**

**Inside Parent Component**

 <p>parent component</p>

    <input type="text" [(ngModel)]="valueFromParent" placeholder="please type here">

    <app-child [valueFromParent]="valueFromParent"></app-child>

**Inside Child Component**

1.Import Input From Angular/Core

2.Declare @Input() Public Parentdata

Or Set Inside Component Decorator

@Component({

Selector:

Template:

Inputs:[ Parentdata]

})

  @Input() public valueFromParent:any

3.Add In Template <H2>{{Parentdata}}</H2>

  <p>child component</p>

    Messageue from parent  : {{valueFromParent}}

**2.Child To Parent Using Event And  @Output**

**In Child Component**

Import { Output,Eventemitter } From '@Angular/Core';

1.Create Eventemitter @Output Public Childevent=New Eventemitter();

  @Output()  ChildEvent : EventEmitter<any>=new EventEmitter();

2.Inside Function,This.Childevent.Emit("Data From Child");

  Valuechanged():void{

this.ChildEvent.emit(this.valueFromChild);

  }

3.set event in html template

   <p>child component</p>

    <input type="text" [(ngModel)]="valueFromChild"   (ngModelChange)="Valuechanged()" placeholder="please type here">

**In Parent Component**

1. Capture Event In Selector <Child-Component (Childevent)="Messagetoparent=$Event"> </Child-Component
2. <app-child1 (ChildEvent)="GetValueFromChild($event)" ></app-child1>

2.define the function and get value

  GetValueFromChild(event:any):void{

    this.childMessage=event

  }

**Communication Between Unrelated Components**

Component Interaction Using Service

--If Hirerchy Is Too Complicated Not Easy To Use @Input And @Output Method

--Create A Subject Inside A Service.

1.Create Service Ng G S Service

2. Declare subject

3. Create Function To Send Msg

export class appService {

  public message = new Subject<string>();

  constructor() {

  }

  sendMessage(value: string) {

    this.message.next(value); //it is publishing this value to all the subscribers that have already subscribed to this message

  }

**In Parent Component**

Access service and use the function to send message to subject

  sendMessage():void{

    this.\_service.sendMessage(this.message)

  }

[**In**](http://5.in/)**Child Component**

Access the service and subscribe to the function

 ngOnInit() {

    this.\_service.message.subscribe(

      (msg) => {

        this.message = msg

      }

    );

  }

**Interfaces—**

## Interface as Type

Interface in TypeScript can be used to define a type and also to implement it in the class.

Interface class

export interface IStudent{

    id:number,

    name:string,

    email:string,

    gender:string,

    status:string

}

**Child class**

 student:IStudent={id:0,name:"",email:"",gender:"",status:""};

 constructor(private \_service:appService) {

    this.student.name="Phill Coulson"

    this.student.email="abc@gmail.com"

    this.student.gender="M"

    this.student.status="Active"

  }

## Interface as Function Type

TypeScript interface is also used to define a type of a function. This ensures the function signature

Interface Class

export interface IClass

{

    GreetStudent(name:string):any

}

Component class

export class InterfaceComponent implements IClass {

 GreetStudent(name: string): void {

    this.message="Hello "+name;

  }

}

**Lifecycle Hook**

* Create Component
* Render Component
* Create And Render Component Children
* Checks When The Componend Data Bound Properties Change
* Destroys The Component Before Removing From Dom

**8lifecycle Hooks**

* 1. **ngOnChanges ()**  -When Input Properties Are Changed --Receives Cur And Prev Property Values
* 2. **ngOnInit ()**  --Initialize The Component After Input Properties Are Ser
* 3. **ngDoCheck ()**  --When Angular Cant Detect Its Own Change
* 4. **ngAfterContentInit ()**  --Respond After Angular Project External Content Into Components View
* 5. **ngAfterContentChecked ()** --After Checking The External Content Projected Into Components View
* 6. **ngAfterViewInit ()** -- After Angular Initialize The Component Views And Child Views
* 7. **ngAfterViewChecked ()** -- After Angular Checks The Component Views And Child Views
* 8. **ngOnDestroy ()**  --Cleanup Purpose. Called   Before Angular Destroy The Component,Unsubscribe

Observables,Detach Event Handlers

  ngOnChanges(): void {

    this.onChangesmessage = "onChanges() triggered at :" + new Date()

  }

  ngOnInit(): void {

    this.onInitmessage = "onInit() triggered at :" + new Date()

  }

  ngDoCheck(): void {

    this.onDoCheckmessage = "onDoCheck() triggered at :" + new Date()

  }

  ngAfterContentInit(): void {

    this.afterContentInit = "onAfterContentInit() triggered at :" + new Date()

  }

  ngAfterContentChecked(): void {

    this.afterContentChecked = "ngAfterContnetChecked() triggered at :" + new Date()

  }

  ngAfterViewInit(): void {

    this.afterViewInit = "ngAfterViewInit() triggered at :" + new Date()

  }

  ngAfterViewChecked(): void {

    this.afterViewChecked = "ngAfterViewChecked() triggered at :" + new Date()

  }

  ngOnDestroy(): void {

    this.onDestroy = "ngOnDestroy() triggered at :" + new Date()

    //cleanup code here

  }

**Angular Service**

**Http Calls In Angular**

In App.Module.Ts

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

Add [Httpclientmodule](https://angular.io/api/common/http/HttpClientModule) In Imports

In Service Class,

Import HttpClient,add in constructor,define function to call url

import { HttpClient } from '@angular/common/http';

 constructor(private \_http:HttpClient) {

  }

 GetEmployees():any{

    return this.\_http.get<any>("http://dummy.restapiexample.com/api/v1/employees");

  }

In Component

Ngoninit Is Best Place To Call Service Calls

Add service in constructor,subscribe to the method

  constructor(private \_appService:appService) { }

  ngOnInit(): void {

    this.\_appService.GetEmployees().subscribe((data: any) =>{

      this.gridData=data

    }

Observable--Asynchronous Pattern

Observable ------Observer Or Subscriber

**Using Bootstrap and jquery In Angular**

npm install jquery –save

npm install popper.js --save

npm install ngx-bootstrap bootstrap@4.1.1

In angular.json file, add the scripts and styles

 "styles": [

              "src/styles.css",

              "./node\_modules/bootstrap/dist/css/bootstrap.min.css"

            ],

            "scripts": ["./node\_modules/jquery/dist/jquery.min.js",

              "./node\_modules/popper.js/dist/umd/popper.min.js",

              "./node\_modules/bootstrap/dist/js/bootstrap.min.js"  ]

          },

Or in style.css , import the css file

@**import** './assets/css/bootstrap.min.css';

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

**Angular Routing**

**Defines A Collection Of Routes,Means Which Component Should Be Shown Based On Path Or Url**

* Navigating From One Componenet To Another
* Defined In App.Module.Ts
* 2 Parts : Path And Component

**Step 1**. Add the code <Base Href="/Src"> In Index.Html --To Set Base Relative Path, May Change In Production

**Step 2**. Import {Routermodule,Routes} From Angular/Core In App.Module.Ts

**Step 3**. Declare routes

const AppRoutes=[

  {path:"home",component: HomeComponent},

  {path:"bindings",component:BindingsComponent},

  {path:"directives",component:DirectivesComponent},

  {path:"lifecycleHooks",component:LifeCycleHooksComponent},

  {path:'',redirectTo:"/home",pathMatch:'full'},

]

**Step 4.** Inside Imports Routermodule.Forroot(Approutes)

**Step 5.** In Template

<a class="nav-link" routerLink="home" >Home</a>

<router-outlet></router-outlet> --Like @Renderbody In Mvc

Or In Class

This.\_Router.Navigate(['/Employeelist']);

**Routing With Parameters**

**Using routerLink**

In app Module,Set Route Path

 {path:"routing/:id",component:RoutingComponent},

In parent component

 <a class="nav-link"   routerLink="routing/1" >Routing</a>

**Using Navigate method**

In Parent Ts, import Router and Inject Route In Constructor,

import { Router } from '@angular/router';

 constructor(private \_router:Router) { }

inside function, call navigate method

GoToRoutingComponent():void{

    this.\_router.navigate(['routing',5])

  }

**To get route parameters inside child component**

import { ActivatedRoute } from '@angular/router';

  Id:any

  constructor(private \_activatedRoute:ActivatedRoute) { }

  ngOnInit(): void {

  this.Id=  this.\_activatedRoute.snapshot.params['Id'];

  }

**Angular Dependancy Injection**

Service Can Be Injected In Module Level Or Component Level Using Providers In Ngmodule Decorator

No Need To Create Any New Instance Of The Service In Each Component

Instance Is Received From Injecter.

Injected Service Can Be Accessed In Component Constructor

Constructor(Private \_Service:Employeeservice)

{}

Angular Component Can Be Injected In

Root App Injector

@Ngmodule({Providers:[Servicename]})

Component Injector

@Component({Providers:[Servicename]})

Forms In Angular

Import {Formsmodule} From @Angular

**Template Driven**

**Model Driven**

**Template Driven**

**Ex**

**In Appmodule.Ts**

**Add Ngform, Ngmodel**

**In Html**

**<Form #Employeform=”Ngform” (Ngsubmit)=”Save(Employeform)”>**

**<Input Type=”Text” [(Ngmodel)] >**

**</Form>**

**In Class**

**Save(Employeform:Ngform):Void{**

**Employeform.Value -- Has All The Form Control Values As Json**

**}**

**Employeform Is The Template Reference Form,Or Check Form Valid Etc**

**{{Employeform.Valid}}**

**Form Validations**

Touched Untouched

Dirty, Pristine

Valid ,Invalid

Add Template Variable Or Local Variable #Catnamecontrol="Ngmodel"

Use Angular Form Validation Properties Like Touched And Valid

<Input Required Matinput Name="Catname" [(Ngmodel)]="Catname" #Catnamecontrol="Ngmodel">

<Span Class="Errormsg" \*Ngif="Catnamecontrol.Touched && Catnamecontrol.Invalid">Name Is Required</Span>

Disable Submit Button If Not Valid

      <Button Id="Btnsave" Type="Submit" Mat-Raised-Button Color="Primary" [Disabled]="Productdetails.Invalid">Save</Button>

If We Are Using Models To Bind Instead Of Autogenerated Fields,Add Model Into Ngsubmit

<Form #Productdetails="Ngform"  (Ngsubmit)="Saveproduct(Product)">

Save(Employeform:Product):Void{

Employeform.Value -- Has All The Form Control Values As Json

}

Promise Vs Observable

Promise—Return Single Value

Observable—Like A Stream That Emits Multiple Data Over A Period Of Time,Need To Subscribe