A javascript framework for developing SPA.

Developed by Google

Uses typescript/javascript/DART

**Features**

Modularity - can use individual modules and can combine to develop large Apps

Easy

Dependency Injection support

Bindings

Routing

REST Api Support

Event handling

Template Support

Angular CLI used to create templates and components

Angular Supports many IDE like Visual studio and so on

Tools – Angular CLI , IDE

Why Angular

* React is not dedicated for SPA, Uses lots of Third Party tools , not suitable for large apps ,
* As angular provide many built in features for routing , binding , dependency injection and etc

Requirements - NodeJS, TypeScript , Angular CLI

Npm install –g @angular/cli

Once intall the angularCLI we wil get a command ng command

Ng new ,Ng generate

Angular1 is purely uses Javascript , uses MVVM patter , and contains scopes , filters, controller, Directives

**Angular2**

Components(Replacement of scope and controllers)

Pipes (Replacement of filters )

Services

Directives

***Architecture***

Module is the main building block ,

Application is combination of different modules

Module = collection of classes . (its package or namespace )

**An application can have more than one module**

Must have only one main module

**Angular Module**

**Component**

Code{}

Html<>

**Pipes**

|currency

**Services**

Class data{

}

**Directives**

Code{}

Other Modules

Built in , Routing,..

**AppModule :**

---------------------------------------------

@ngModule (

Declaration:[AppComponent,HomeComponent, AboutComponent,.. Components, pipes, directives ],

Providers :[MovieService,UserService, Services],

Imports:[UserModule,BuiltinModules,BrowserModule, all the modules],

Exports :[ModuleNames to be exported ,..],

entryComponets :[dynamic Components ]

Bootstrap:[AppComponent]// setting the startup

)

Export class AppModule {

}

Let us consider in a project which is having first, second , third Module

@ngModule for FirstModule

Declarations – Compoenenets for First Module

Imports – secondmodule , third Module , other module

Exports – Exports the modules to other modules which is consuming appmodule

Importing some module , adding some more functionality and export

Convention

Module names should ends with modules

Service names should ends with service

There must be one Root Component , generally it will be with Name APP

Main module should contain Root Component

Bootstrap Sequence

Index.html - <app></app> -🡪 main.ts 🡪 appModule 🡪 finds app and display

**Component**

@component({

Selector:home,

Template:`html code tags`, // template is used when we want render some inline html

Templateurl:’/htmlpageurl.html’, // hmtl file url

Styles:`[ all styles can be written ]`,

StylesUrls:[styleFilename.css,home.css,..All the css file required ]

})

Export class Home{

Constructor(){}

doClick(){}

}

On home page u can have

<div>

<button id=”btn” (click)=’doClick’ />

</div>

* Angular Events ie click must be in ().

**Ex: to Create a new component**

Create a new file in app folder >> name it as sample.component.ts

import {Component} from '@angular/core'

@Component({

selector:'sample-comp',

template:'<h1>Sample Component</h1>',

styles:["h1{color:red}"]

})

export class SampleComponent{

title='Sample Component'

}

**Now at appModule.ts we need to register the component . write the following**

import {SampleComponent } from './sample.component'

@NgModule({

declarations: [

AppComponent,SampleComponent

],

})

**To Dispaly the sample component we need to use the selector**

At appcomponent.html include the selector tag for samplecomponent

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

<h1>

Welcome to {{ title }}!

</h1>

</div>

<h2>Here are some links to help you start: </h2>

<ul>

</ul>

**<sample-comp></sample-comp>**

**Bindings**

Rendering the data onto html controls is called binding

**Types of Bindings :**

One way binding

Interpolation

Property binding

Two way binding

Event binding

**One way Binding :**

Just Display the data on to the page . Changes in the value will not reflect

Ex: Interpolation binding , Property Binding

**Interpolation**  ( Expression binding) - one Way Binding

Use {{ }} Ex: <p>{{name}}</p> wil print the value of name variable in paragraph

Ex:

export class HomeComponent implements OnInit {

private name : string ="Rama";

constructor() { }

ngOnInit() {

}

}

Html View

<p>

Hi Mr. {{name}}

</p>

**Property binding**

Mostly used for binding form controls like textboxes , list and etc

Use [] ex : [Property] = “nameofVar”

Ex:

export class HomeComponent implements OnInit {

private name : string =”Rama”;

constructor() { }

ngOnInit() {

}

}

Html view

<p>

User name : <input type=”text” [value] = “name” />

</p>

**Event Binding**

Used to bind the events of html controls. Use () to bind the event

Syn : (eventName ) = ‘functionName()’

Ex:

export class HomeComponent implements OnInit {

constructor() { }

btnClick(){

alert('Hi Button Clicked...')

}

ngOnInit() {

}

}

Html view

<button (click)="btnClick()">Click Me </button>

**Two way binding**

After binding data , if any change in the source will be reflect at target

We use [(ngModel)] to bind the data

* If we are using ngModel , we need to import

Another way of implementing two way binding by refering the html element by using #

Use keyup event to send the data to the component variable and bind that in html

Ex:

export class HomeComponent implements OnInit {

private twb: string=""

constructor() { }

orgPress(orgVal)

{

this.twb= orgVal.value;

}

}

Html View

<input type="text" id="txtOrgName" #org (keyup)="orgPress(org)" />

<h1 [style.color]="txtColor">MANIDEEP</h1>

**Directive**

A directive is a customTag or a attribute

Structural Directives - used to adding elements to the dom basing on condition

\*ngIf and, \*ngFor, \*ngSwitch

Behavioural directive – not changing the dom , but changing the behavior of the element

If we add click event handler to a button , that means we adds something for

Button element

Ex: highlight , click , ..

Component Directive - Directives with template

A custom tag which is associated with additional view . generally component we are creating are component directives

**\*ngif** - conditionally display the element

Ex:

<p \*ngIf="dept==='IT'">

Happy teachers day

</p>

The above paragraph will appear only if the dept value is IT , and disappear if false

\*ngforEx:

Used to repeat the loop for number of times . we can iterate through each item of the source

Class

{

Private courses : string[];

Constructor()

{

This.courses =[‘Angulrar’,’Java’,’React’,’Mongo’];

}

}

<ul>

<li \*ngfor=”let cr of courses”>{{cr}}</li>

</ul>

**Note :**

**Ng generate component components/ComponentName**

**Will generate a component with complete .ts, .css, .html files**

**We can generate pipes, service, component and etc with ng generate command**

**ngStyle** – use to apply dynamic style for html elements

<p [ngStyle]={'color':'colorName'} >

If we use ngStyle we can provide css properties with variable value

Ex:

Component.html

Enter U r color : <input type="text" #clrName (keyup)="kuClr(clrName)"/>

<h1 [ngStyle]="{'color':ColorName}">ANGULAR2</h1>

Componenet.ts

export class HomeComponent implements OnInit {

private ColorName: string;

constructor() { }

kuClr(cl)

{

this.ColorName = cl.value;

}

ngOnInit() {

}

}

**ngClass**

Apply the styles based on a condition

Component.ts

export class ExampleComponent implements OnInit {

private Employees:any[];

constructor() {

this.Employees=[{"id":"1001","name":"Rama","city":"Hyd"},{"id":"1002","name":"Raja","city":"Bang"},{"id":"1003","name":"Rana","city":"Hyd"}]

}

ngOnInit() {

}

}

Component.css

.highlight

{

color: black;

background-color: yellow

}

Component.html

<div \*ngFor="let em of Employees" >

<p>Id:{{emid}}</p>

<h3>Name:{{em.name}}</h3>

<p [ngClass]="{'highlight':em.city==='Hyd'}">City:{{em.city}}</p>

</div>

**ngSwitch**

* **Decides which element to be displayed in the view**
* **We need to define the styles first at .css file**

component.html

<div \*ngFor="let em of Employees" [ngSwitch]="em.city" >

<p>Id:{{emid}}</p>

<h3>Name:{{em.name}}</h3>

<p class="hc" \*ngSwitchCase="'Hyd'">{{em.city}}</p>

<p class ="bc" \*ngSwitchCase="'Bang'">{{em.city}}</p>

<p \*ngSwitchDefault>{{em.city}}</p>

</div>

Component.css

.hc

{

color: black;

background-color: yellow

}

.bc

{

color: black;

background-color: red

}

**ngNonBindable** - to disable the binding for any element we can use ngNonBindable

**Sample Example** of EventManagement Example

.row-display-flex{

Display:flex;

Flex-wrap:wrap;

}

.thumbnail{

Height:100%

\}

.event-title{

// for showing tool tip of a heading of

White-space : nowrap;

Overflow:hidden;

Text-overflow: ellipsis

}

**Pipes**  - the pipes are formatters , used to format the data

Uppercase

Lowercase

Date

Number

Currency

**Working with Forms**

**Reactive Forms :**

A form building technology for angular Applications

Uses the **ReativeFormsModule** from **@angular/forms Library**

With reactive forms, you create a tree of Angular form control objects in the component class and bind them to native form control elements in the component template,

Advantage is that value and validity updates are [always synchronous and under your control](https://angular.io/guide/reactive-forms#async-vs-sync)

**Steps to work with reactive forms**

1. Import ReactiveFormsModule at appModule.ts from @angular/forms library
2. And specify that ReactiveFormsModule at imports section of @ngModule({})
3. At Component import FormGroup, FormBuilder, Validators from a@angular/forms
   1. import {FormGroup, FormBuilder, Validators} from '@angular/forms'
4. Create a FormGroup variable at .ts file and add the controls using FormBuilder

this.frmGrp = this.fb.group({

fname: ['', Validators.required],

lname: ['', Validators.required]})

1. ControlName at FormBuilder in .ts file should match the formControlName at html file

<input t type="text" formControlName="fname" />

1. Bind the formGroup property of <form> with FormGroup Variable
   1. <form [formGroup]= “frmGrp” >
2. all validations can be checked by using formGroup.valid
3. all the values can get by using formGroup.value | json or formGroup.value.Controlname

**To validate an individual control**

We can use formcontrol validation properties

Valid

Invalid

Dirty

Prestine

Each formcontrol accessed using formGroup

* this.frmGrp.controls["fname"].invalid – return true if any of the validations on that formcontrol evaluates false

Ex:

**At appModule.ts**

import { ReactiveFormsModule, FormGroup} from '@angular/forms'

@NgModule({

imports: [

BrowserModule, ReactiveFormsModule

],

})

export class AppModule { }

At Component.ts

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

import {FormGroup, FormBuilder, Validators} from '@angular/forms'

@Component({

selector: 'app-employee',

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

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

})

export class EmployeeComponent implements OnInit {

private frmGrp : FormGroup;

private frmBdr : FormBuilder;

private venomsg : string

constructor() {

this.venomsg="";

this.frmBdr = new FormBuilder();

}

btnSubmit()

{

let veno = this.frmGrp.controls["eno"];

if(veno.invalid==true || veno.untouched == true || veno.dirty== true)

{

this.venomsg = "Plz Check Eno .."

}

else

{

this.venomsg="";

}

//alert(this.frmGrp.valid)

}

ngOnInit() {

this.frmGrp = this.frmBdr.group({

eno:['',Validators.required],

ename:['', Validators.required],

deptId:['', Validators.required],

mngrId:['', Validators.required],

city :[''],

salary:['', Validators.required]

})

}

}

**Component.html**

<div class = "container">

<div class="row">

<h2>Employee Register</h2>

</div>

<div class ="row">

<div class="col-md-offSet-3 col-md-6">

<form [formGroup] = "frmGrp">

<div class="form-group">

Eno:<input type="text" class="form-control" formControlName="eno" /><span style ="color:red">{{venomsg}}</span>

</div>

<div class="form-group">

Name:<input type="text" class="form-control" formControlName="ename" />

</div>

<div class="form-group">

DepartmentID:<input type="text" class="form-control" formControlName="deptId" />

</div>

<div class="form-group">

ManagerId:<input type="text" formControlName="mngrId" class="form-control" />

</div>

<div class="form-group">

City:<input type="text" formControlName="city" class="form-control"/>

</div>

<div class="form-group">

Salary:<input type="text" formControlName="salary" class="form-control"/>

</div>

<div class="form-group">

<button class="btn btn-primary" (click)="btnSubmit()">Save</button>

</div>

</form>

</div>

</div>

</div>

**To Apply Bootstrap in Agular2 Application**

First install bootstrap and jquery

Npm install jquery –-save

Npm install bootstrap –-save

At anglar.json File include

**"styles": [**

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

**"src/styles.css"**

**],**

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

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

**Working with LifeCycle Hooks**

An angular component under goes different stages in its life cycle