**Execution:**

tsc <<filename.ts>>

node <<filename>>

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

**JSON:**

(JavaScript Object Notation-- Key, Value Pair)

emp=[

{"firstname":"sruthi","age":"21"}

{"firstname":"aaradhya", "age":"11"}

];

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

**XML:**

<?xml version="1.0"?>

<employees>

<employee>

<firstname>sruthi</firstname>

<age>21</age>

</employee>

<employee>

<firstname>aaradhya</firstname>

<age>11</age>

</employee>

</employees>

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

**XML and JSON:**

Similarities:

suppoerted for Data Transmission

are Hierarchical

supported in web service

wil support XMLHTTPRequest

Differences:

To parse XML, we need to use XMLDOM, JSON can be parsed using JSON.Parse() which is a javascript -- met

JSON structure is easy to understand compare to XML

JSON currently supports most of the language

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

get, post, put, patch, delete

Head, Connect, Options, Trace

connect--used to establish the end to end http tunnel through proxy server

options--set of instructions to be followed or maintained to connect service

trace--diagnostics information

http status code--

404 - server not found

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

**Struct in typescript:**

var Employee:{

id:number;

name:string;

};

Employee={

id:100,

name:"ram"

}

console.log(Employee);

let data1: string ="hi";

function check()

{

let data2: string="welcome";

{

//let data3:string="bye";

var data3:string="bye";

}

console.log(data1);

console.log(data2);

console.log(data3);

}

check();

var---> function scoped

let---> block scoped

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

**Datatypes in typescript:**

number,string,var,tuples,array,any,union

let num1:number;

let data1:string;

let data3:boolean;

//let names:string[]=["sruthi","mandalapu"];

let names:Array<string>;

names=["sruthi","mandalapu"];

console.log(names);

//arrays

let nums:number[]=[1,2,3,4];

let stringss:string[]=["hii","this","is","a","string array"];

console.log(nums);

console.log(stringss);

for(var info in stringss){

console.log(stringss[info]);

}

console.log("-------------------------");

for(var i=0;stringss.length<5;i++){

console.log(stringss[i]);

}

//tuple

var product:[number,string]=[100,"Oil"];

var ticket:[number,string,boolean,number]=[12,"ABC",true,5];

console.log(product);

console.log(ticket);

console.log("The product name is: "+product[1]);

console.log("Is ticket confirmed: "+ticket[2]);

//union -- can use different data types

//union

let identityproof:(string|number|boolean|number[]);

identityproof=123456;

identityproof="ABS780";

identityproof=true;

identityproof=[10,20,20];

console.log(identityproof);

let mydata:any="Hi";

mydata=20;

mydata=true;

console.log(mydata);

let info:any[]=["Ram",30,12,1200,true];

console.log(info);

var mydata=[10,"data"];

console.log(mydata);

mydata.push("sruthi");

console.log(mydata);

let data:any=123;

let info=data as number;

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

**Functions:**

//function with name

function greet(){

console.log("hii");

}

greet();

function calculate(x:number,y:number):number{

return x+y;

}

let result=calculate(5,9);

console.log(result);

//function without name

let fun=function(){

console.log(calculate(2,3));

}

fun();

let func=function(x:number,y:number):number{

return x+y;

}

console.log(func(1,2));

function greet(val1:string,val2?:string):string{

return val1+" "+val2+"!";

}

console.log(greet("hii ","sruthi ","wt are u doing"));

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

**Arrow Functions:**

Arrow function => (Tends to) => In other languages we can call it as lambda expression

(param1, param2, param3, …….) => Expression

//Arrow functions

function add(num1:number,num2:number):number{

return num1+num2;

}

console.log("Using normal function: "+add(1,2));

let res=function(num1:number,num2:number):number{

return num1+num2;

}

console.log("Using anonymous function: "+res(3,4));

let l=(num1:number,num2:number):number => {

return num1+num2;

}

console.log("Using arrow function: "+l(5,6));

//Arrow function having no parameters and return type

let res=()=>{

console.log("sruthi");

}

res();

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

**REST Parameter:**

Passing n-number of parameters easily to a type script function.

If the number of parameters for a function will be vary or not known

REST parameter can be represented using …

Rule: Make sure rest parameter must come last in the function definition

//example

function process(data:string,...nums:number[]):void{

let total:number=0;

for(var val in nums){

total+=nums[val];

}

console.log(data+" "+total.toString());

}

process("Sruthi",90,67,89);

process("Tarun",30,20,40,48);

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

**Typescript - Interface:**

Interface contains structure that will define template for your requirement.

It will specify how the classes should flow or follow. If any class using the interface, then it must follow the structure provided by the interface.

Interface contains properties, methods:

1. Interface as type

2. Interface as function

//Interface demo examples

//Interface as type

interface IProduct

{

pid:number;

pname:string;

}

let p1:IProduct={pid:1,pname:'Sruthi'};

console.log(p1);

//Interface as function

interface IArithmetic

{

(val1:number,val2:number):void;

}

function Sub(num1:number,num2:number):void{

console.log("The Result is: "+(num1-num2));

}

function Add(data1:number,data2:number):void{

console.log("The Value is: "+(data1+data2));

}

let calc:IArithmetic=Sub;

calc(1,2);

calc=Add;

calc(3,4);

//Readonly Property

interface IAadhar

{

state:string;

readonly aadharnumber:number;

}

let mem1:IAadhar={state:"AP",aadharnumber:123456};

console.log(mem1);

mem1.state="UP";

// mem1.aadharnumber=345474; //It is a readonly property, cannot be assigned

console.log(mem1);

//Declaring optional property in interface

interface IPostOffice

{

accountNumber:number;

name:string;

referencePerson?:string; //declaration of optional property

}

let Post1:IPostOffice={accountNumber:100,name:"Ram"};

let Post2:IPostOffice={accountNumber:400,name:"Sruthi",referencePerson:"Sruthi"};

console.log(Post1);

console.log(Post2);

Inheriting interface 🡪 Supports multiple inheritance

interface first{

data1:string;

data2:string;

}

interface second{

data3:boolean;

}

interface third{

data4:any;

}

interface fourth extends first,second,third{ //supports multiple inheritance

}

var four:fourth={data1:"sruthi",data2:"mandalapu",data3:true,data4:20};

console.log(four);

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

**Typescript Class:**

ECMA Script6 or later of ES2015 supports class.

Before that ECMA6, JS 🡪 is a Functional Programming

Inheritance 🡪 Prototype based

Functions 🡪 used to build reusable components

From ECMA6 onwards, JS introduced/supported with object-oriented class-based approach.

Class 🡪 declare variables, properties, methods

Class object can be initialized using new keyword

//class demonstration

class Supplier

{

supcode:number;

supname:string;

constructor(code:number,name:string){

this.supcode=code;

this.supname=name;

}

showSupply():number

{

return 200;

}

}

let sup=new Supplier(10,"sruthi");

console.log(sup);

console.log("The supplier code is: "+sup.supcode);

console.log("The supplier name is: "+sup.supname);

console.log("The supplier supplied value is: "+sup.showSupply());

//classes cannot hold multiple inheritance

//the below code results in error

class Supplier

{

supCode:number;

supName:string;

constructor(code:number,name:string){

this.supCode=code;

this.supName=name;

}

}

class Product{

}

class Customer extends Supplier,Product{

}

//example of inheritance

class Supplier

{

supCode:number;

supName:string;

constructor(code:number,name:string){

this.supCode=code;

this.supName=name;

}

}

class Product extends Supplier{

pName:string;

price:number;

constructor(code:number,name:string,pName:string,price:number){

super(code,name);

this.pName=pName;

this.price=price;

}

calctotal(qty:number):number{

return qty\*this.price;

}

}

let prod=new Product(10,"Sri","Paste",100);

let res=prod.calctotal(50);

console.log("The supplier "+prod.supName+" supplied "+prod.pName+" with total price: "+res);

//class with interface

interface INewsPaper

{

reading():string;

}

class Library implements INewsPaper{

libraryName:string;

loc:string;

capacity:number;

constructor(name:string,loc:string,capacity:number){

this.libraryName=name;

this.loc=loc;

this.capacity=capacity;

}

reading():string{

return "The library "+this.libraryName+" is located in "+this.loc+" having capacity of "+this.capacity;

}

}

let lib:Library=new Library("books","guntur",200);

console.log(lib.reading());

//method overriding

class Shape {

calculatearea():void{

console.log("Every shape has area");

}

}

class Square extends Shape{

calculatearea():void{

console.log("Area is side\*side");

}

}

class Rectangle extends Shape{

calculatearea():void{

console.log("Area is length\*breadth");

}

}

let myShape:Shape=new Shape();

myShape.calculatearea();

let Shape1:Square=new Square();

Shape1.calculatearea();

let Shape2:Rectangle=new Rectangle();

Shape2.calculatearea();

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

**Typsescript Access Modifiers:**

public 🡪 Default

Private

Protected

//Access Modifiers

class Customer{

public custId:number=10; //public

custName:string="sruthi"; //default

// private custName:string="sruthi"; //private

}

let cust:Customer=new Customer();

console.log("The customer id is: "+cust.custId);

console.log("The customer name is: "+cust.custName);

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

**Static in Typescript:**

Any static information can be accessible in the class

//example of static property

class ShareMarket

{

static goldRate:number=456.98;

calcGoldPrice(gram:number):number{

return gram\*ShareMarket.goldRate;

}

}

let sm:ShareMarket=new ShareMarket();

var result:number=sm.calcGoldPrice(5);

console.log("The Total Amount is: "+result);

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

**TS/Angular Module: (Import and Exporting modules)**

It provides to organize or categorize or group related code together.

We can export any declaration (variable, function, class, interface and type) using “export” keyword.

If any file contains top level import or export statement, then it will be considered as module.

As module cannot be executed through online compiler (TS Playground etc..), we have to run through VS code.

1. VS Code 🡪 Create a file and save it with .ts extension

2. Compile the file using tsc command 🡪 tsc <<filename.ts>> 🡪 if no error it will create Javascript file

3. Execute the necessary js files using “node” command.

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

**Typescript Namespace:**

It is a logical group of functionalities. It can include interface, class, function and variables

It will avoid naming conflicts

Example:

🡪Person1.ts 🡪Person2.ts 🡪Person3.ts

namespace Person1{ namespace Person2{ import {Titanic} from “./Person1”;

class Titanic{ class Titanic{ import {Titanic} from “./Person2”;

//ship description //movie description //let t:Titanic=new Titanic();

} } //there is a name collision

} } Let t=new Person2.Titanic();

Below command used to add the import a namespace where no export keyword present in the namespace: ///<reference path=”filename.ts”/>

Once namespace included in the target file, we can use below command to include the source file

tsc --target es6 <<filename.ts>> --outfile <<filename1.js>>

node filename1

tsc --target es6 namespace\_export2.ts --outfile second.js

node second

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

**Typescript Generics:**

1. It is Type Safe

2. Instead supplying type at compile time, it will replace type during execution

Example:

Hammer, Screw Driver, Spanner, Drilling Machine 🡪 Tools

🡪Generics always represented using <T>

Example:

// function printData(data:number|string|boolean){ // data:any

function printData<T>(data:T){

console.log("Data: "+data);

}

printData(5); // since 5 is passed T will be treated as int and data will be of number type

printData("sruthi");

printData(true);

printData([1,2,3]);

printData([2,3,4,,8,"Hi"]);

printData({name:"Ram",id:2});

function printData<T,U>(data1:T,data2:U){

console.log("The output is: "+data1+" "+data2);

}

printData(2,78);

printData(647,"Sruthi");

printData([26,"sruthi"],8839);

🡪Generics can be applicable for class, variable, function and interface

Example:

class keyvalue<T,U>

{

private key?:T;

private value?:U;

setKeyValue(key:T,value:U){

this.key=key;

this.value=value;

}

displayValues(){

console.log("The key is: "+this.key+" The value is: "+this.value);

}

}

let kv=new keyvalue<number,string>;

kv.setKeyValue(3,"sruthi");

kv.displayValues();

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

**Angular:**

Reference - angular.io

JS binding framework, single page web application

Binds HTML and Typescript

Dependency Injection, API Calls, routing etc

It is open-source JS based Framework used to develop web applications. This is one of the best framework for developing any SPA(Single Page Application)

It is MVC based structure framework for developing SPA applications, support client-side templating feature, facility to perform unit test so that code can be tested before deployment.

🡪Angular folder structure:

To create a new angular project, we have to use the below command:

ng new <<project name>>

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

**Angular Component:**

It is the building block of UI. It can be viewed as a collection of components and one component can be viewed/handled the view or part of the view

It contains data, HTML Markup, logic required for the view

Every angular application should have at least one component that used to display the data on the view

Every angular application is composed of 3 things:

1. Angular class – typescript class
2. Template (HTML Template/Template URL)
3. Decorator (Information of component)

**Component process:**

Command to create new component: ng g c|component <<component name>>

1. Go to app.module.ts 🡪 you will find bootstrap - modify with newly created component “ComponentnameComponent”
2. Go to componentname.component.ts 🡪 you will find the selector, copy it
3. Go to index.html 🡪within body change the tag to selected selector which was copied above

The selector name will be 🡪 app-[name of the component given]

Eg: if the name of the component is samp selector is 🡪 app-samp

**Build an Angular Project:**

To compile project 🡪 ng build

To compile and run the project 🡪 ng server -o 🡪 Angular project runs with default port of 4200

**Creating Parent/Child Component or Nesting Component:**

One component is called in another component

**Passing data from Parent Component to Child and vice versa:**

@Input 🡪 Pass data from Parent to Child

@Output 🡪 Pass data from Child to Parent

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

**Databinding:**

In angular communication between component and view happened through Databinding

**Interpolation:**

{{ }} whatever variables associated with values, can be displayed using interpolation operator

**Property/Attribute Binding: (Controller to View)**

.html file: <img [src]=”picture”>

.ts file: var picture:string=”./demo.png”;

.html file: <span [innerText]=”value”></span>

.ts file: var value=”This is a span value”;

**Event Binding: (View to Controller)**

.html file: <button (click)=”onClick()”></button>

.ts file: onClick(){ alert(“Hii”); }

**Class Binding:**

.html file:

<span class="red">This is in red color</span><br>

<span [className]="'red size'">This is in red color</span><br>

.css file:

.red{ color:red;} .size{ font-size:24px; }

.html file:

<span [ngClass]="'red size'">ngClass Binding</span><br> //ngClass with string

<span [ngClass]="['red','size']">ngClass Binding</span><br> //ngClass with Array

<span [ngClass]="{'red':true,'size':false}">ngClass Binding</span><br> //ngClass with Object

.css file: similar file the above mentioned for class binding

**Two-way Data Binding:**

(Event + Property)

.html file:

Enter your name: <input [(ngModel)]="fullName" /><br>

<p>{{ fullName }}</p>

.ts file:

var fullName:string=”sruthi”;

app.module.ts file:

import {FormsModule} from “@angular/forms”;

and in @NgModule you will find a imports option mention as FormsModule

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

**Angular Directive:**

These angular elements used to change the behavior or appearance or layout of the DOM element

3 types:

1. Structural 🡪 \*ngIf, \*ngFor, \*ngSwitch
2. Attribute 🡪 ngstyle, ngclass
3. Component 🡪 @Component

**Adding bootstrap to the project:**

Command to install bootstrap 🡪 npm install bootstrap

Import statement in styles.css 🡪 @import “~bootstrap/dist/css/bootstrap.css”

**ngIf Directive:**

It is used to add or remove HTML elements and its descendent element from the DOM layout at runtime conditionally.

Syntax 🡪 \*ngIf = ‘Expression’

Example:

.html file:

<div>

<input type="radio" name="rb" (click)="ChangeData(true)" checked>Valid

<input type="radio" name="rb" (click)="ChangeData(false)">Invalid

</div>

<div \*ngIf="isValid">Valid is checked</div>

<div \*ngIf="!isValid">Invalid is checked</div>

.ts file:

isValid:boolean=true;

ChangeData(valid:boolean){

this.isValid=valid; }

**ngFor Directive:**

Simillar to for… loop . It is used to iterate over a collection of data

Syntax 🡪 \*ngFor = let <value> of <collection>

**ngSwitch Directive:**

Example:

.html file:

<select (change)="SetDropDownValue($event)">

<option value="">Select</option>

<option value="in">India</option>

<option value="us">US</option>

<option value="uk">UK</option>

</select>

<div [ngSwitch]="dropDownValue">

<h4 \*ngSwitchCase="'in'">You have selected India</h4>

<h4 \*ngSwitchCase="'us'">You have selected United States</h4>

<h4 \*ngSwitchCase="'uk'">You have selected United Kingdom</h4>

<h4 \*ngSwitchDefault="">You haven't selected any Country</h4>

</div>

.ts file:

public dropDownValue="";

SetDropDownValue(drpValue:any){

this.dropDownValue=drpValue.target.value;

}

**ngStyle Directive:**

Example:

.html file:

<!-- ngStyle usage -->

<div>

<button [ngStyle]="AddButtonCSSStyles()">Virtusa</button>

</div>

.ts file:

//example for ngStyle Directive

AddButtonCSSStyles(){

let CssStyles={

'color':'red',

'font-weight':'bold',

'font-size.px':59

};

return CssStyles;

}

**ngClass Directive:**

Example1:

.html file:

<!-- ngClass usage-->

<div [ngClass]="AddCSSClasses('type1')">Sruthi</div>

<div [ngClass]="['one','three']">Sruthi</div>

<div [ngClass]="'one two'">sruthi</div>

.ts file:

//example for ngClass Directive

AddCSSClasses(flag:string){

let classstyles1={'one':true,'two':true,'three':true};

let classstyles={'one':false,'two':true,'three':true};

if(flag=="type1")

return classstyles;

else

return classstyles1;

}

Example2:

.html file:

<span [ngClass]="'red size'">ngClass Binding</span><br>

<span [ngClass]="['red','size']">ngClass Binding</span><br>

<span [ngClass]="{'red':true,'size':false}">ngClass Binding</span><br>

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

**Angular Forms:**

Building block of an application. It provides user friendly environment and also gives hiding the complexity of the application.

In angular 2 different ways to collect data (it will perform validation) from the user

1. Template-Driven Form
2. Reactive Form or Model Driven Form

**Template Driven Form:**

It is a simple form which can be used to develop application forms, since everything we are going to use it in an application, we need to define in the template that we are using along with the component.

Here we need to import FormsModule in the app.module.ts

1. Form elements must be provided with ngForm directive. It helps to create the control group inside the directive. It is attached with <Form> element in the HTML and nested forms
2. ngModel: when we add this directive all the input elements are registered with ngForm. It will create instance of FormControl class and assign FormControl elements
3. ngSubmit: directive will submit the form when we hit the enter key or submit (button type)

app.module.ts:

import {FormsModule} from ‘@angular/forms’; and imports need to be included with FormsModule

Example:

In .html file:

<form #studentForm="ngForm" (ngSubmit)="RegisterStudent(studentForm)">

<div class="form-group">

<label for="firstName">First Name</label>

<input id="firstName" type="text" class="form-control" name="firstName" ngModel>

</div>

<div>

<label for="lastName">Last Name</label>

<input type="text" name="lastName" id="lastName" class="form-control" ngModel>

</div>

<div>

<label for="email">Email</label>

<input type="text" name="email" id="email" class="form-control" ngModel>

</div>

<div>

<label>Gender</label>

<label>Male</label><input name="gender" value="male" type="radio" ngModel>

<label>Female</label><input name="gender" value="female" type="radio" ngModel>

</div>

<div>

<input type="submit" value="SUBMIT">

</div>

</form>

In .ts file:

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

RegisterStudent(studentForm:NgForm){

console.log(studentForm.value);

alert(studentForm.value);

alert("The student name is: "+studentForm.controls['firstName'].value);

}

Giving a default value to form and printing them:

By using Two-way Data Binding:

[(ngModel)]=”name”

**Reactive Form:**

It is used to define the structure of the form in Angular Component(class) itself.

To implement reactive forms we need to include import ReactiveForms module

The 2 things used for implementation of Reactive Forms

1. Form Control: It will encapsulate state of single form element. It stores the value and state of the form element and helps to interact with them using control properties and methods
2. Form Group: Represents collection of form control. It can also contain form groups

app.module.ts file:

import {ReactiveFormsModule} from ‘@angular/forms’;

and imports need to be included with Reactive

Example:

.html file:

<form [formGroup]="contactForm" (ngSubmit)="onSubmit()">

<div class="form-group">

<label for="firstName">First Name</label>

<input id="firstName" type="text" class="form-control" name="firstName" formControlName="firstname">

</div>

<div>

<label for="lastName">Last Name</label>

<input type="text" name="lastName" id="lastName" class="form-control" formControlName="lastname">

</div>

<div>

<label for="email">Email</label>

<input type="text" name="email" id="email" class="form-control" formControlName="email">

</div>

<div>

<label>Gender</label>

<label>Male</label><input name="gender" value="male" type="radio" formControlName="gender">

<label>Female</label><input name="gender" value="female" type="radio" formControlName="gender">

</div>

<div>

<label for="isMarried">Married</label>

<input type="checkbox" name="isMarried" formControlName="isMarried">

</div>

<div>

<label for="country">country</label>

<select id="country" name="country" formControlName="country">

<option value="1">India</option>

<option value="2">USA</option>

<option value="3">England</option>

<option value="4">Singapore</option>

<option value="5">UK</option>

</select>

</div>

<div>

<input type="submit" value="SUBMIT">

</div>

</form>

.ts file:

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

contactForm=new FormGroup({

firstname:new FormControl(),

lastname:new FormControl(),

email:new FormControl(),

gender:new FormControl(),

isMarried:new FormControl(),

country:new FormControl()

});

onSubmit(){

alert(this.contactForm.value);

alert(this.contactForm.controls.firstname.value);

}

Giving a default value to form and printing them:

In html it is printed 🡪 {{form\_obj.value.name}}

In ts 🡪 form\_obj=new FormGroup({

name: new FormControl(“sruthi”), // name: new FormControl();

email: new FormControl(“sruthi@gmail.com”)

});

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

**Angular Form Control State:**

For Angular Validation we can use our HTML5 validation syntax like required, minlength, maxlength, pattern

Untouched 🡪 the control not yet touched

Touched 🡪 the control is untouched

Pristine 🡪 the control value is not yet modified

Dirty 🡪 control value is changed

Invalid 🡪 control value is not valid

Valid 🡪 control value is valid

Submitted 🡪 the form is submitted

**Reactive form Validations:**

.html file:

<div class="container">

<h1>Welcome!!</h1>

<form [formGroup]="angForm" novalidate>

<div>

<label>Name: </label>

<input type="text" formControlName="name">

</div>

<div \*ngIf="angForm.controls['name'].invalid && (angForm.controls['name'].dirty || angForm.controls['name'].touched)" class="alert alert-danger">

<div \*ngIf="angForm.controls['name'].errors?.['required']">

Name is required!

</div>

</div>

<div>

<label>Address: </label>

<input type="text" formControlName="address">

</div>

<div \*ngIf="angForm.controls['address'].invalid && (angForm.controls['address'].dirty || angForm.controls['address'].touched)" class="alert alert-danger">

<div \*ngIf="angForm.controls['address'].errors?.['required']">

Address is required!

</div>

</div>

<div>

<button type="submit" [disabled]="angForm.pristine || angForm.invalid" class="btn btn-success">save</button>

</div>

</form>

<p>Form Value: {{angForm.value | json}}</p>

<p>Form Status: {{angForm.status | json}}</p>

</div>

.ts file:

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

angForm:FormGroup;

constructor(private fb:FormBuilder){

this.angForm=this.fb.group({

name:['',Validators.required],

address:['',Validators.required]

});

}

**Template Driven Form Validations:**

.html file:

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

**Angular Builder:**

FormBuilder API makes it easier to build reactive forms

**Angular Pipe:**

Take the raw data as input and then transform it to the desired output.

Using (|) symbol/operator, we can apply pipe features to any of the property in angular application

Built-in pipe: String, Date, Math, Currency, JSON

Pipe with chaining: FirstName | first 5 characters | display in uppercase

Example:

.html file:

<table border="1">

<tr>

<th>Student ID</th>

<th>Name</th>

<th>DOB</th>

<th>Gender</th>

<th>Course Fee</th>

</tr>

<tr \*ngFor="let student of students">

<td>{{student.ID | uppercase}}</td>

<td>{{student.Name | titlecase}}</td>

<td>{{student.DOB | date:'dd/MM/yyyy'}}</td>

<td>{{student.Gender | lowercase}}</td>

<td>{{student.CourseFee | currency:'USD':true}}</td>

</tr>

</table>

<hr/>

<p>Date Pipe: {{today | date}}</p>

<p>Full Date: {{today | date:'fullDate'}}</p>

<p>Mediate Date: {{today | date:'medium'}}</p>

<p>Short Date: {{today| date:'short'}}</p>

<p>Date: (dd/MM/yyyy): {{today | date:'dd/MM/yyyy'}}</p>

<p>Time: {{today | date:'h:mm a z'}}</p>

<p>Medium Time: {{today|date:'mediumTime'}}</p>

<hr/>

<p>Currency USD in Symbol: {{salary|currency:'USD':true}}</p>

<p>Currency INR in Symbol: {{salary|currency:'INR':true}}</p>

<p>Currency USD in Code: {{salary|currency:'USD':false:'4.2'}}</p>

<p>Currency INR in Code: {{salary|currency:'INR':false:'11.3'}}</p>

.ts file:

students:any[]=[

{ID:'std01',Name:'Sruthi',DOB:'05/28/2001',Gender:'Female',CourseFee:8328.45},

{ID:'std02',Name:'Aari',DOB:'12/20/2010',Gender:'Female',CourseFee:7782.83},

{ID:'std03',Name:'George',DOB:'05/30/2003',Gender:'Male',CourseFee:3728.83},

{ID:'std04',Name:'Swarup',DOB:'09/09/1997',Gender:'Male',CourseFee:9268.83},

{ID:'std05',Name:'Lekhana',DOB:'02/02/1990',Gender:'Female',CourseFee:2799.83}

];

today:number=Date.now(); //Data Pipe

salary:number=123456723.50; //Currency Pipe

**User defined pipe (Custom Pipe):**

To create a user defined pipe: ng g pipe <<pipe name>>

Example:

.html file:

<!-- Mr and Mrs tag attaching based on gender, And also calculating age based on data of birth-->

<table border="1">

<tr>

<th>Student ID</th>

<th>Name</th>

<th>DOB</th>

<th>Age</th>

<th>Gender</th>

<th>Course Fee</th>

</tr>

<tr \*ngFor="let student of students">

<td>{{student.ID | uppercase}}</td>

<td>{{student.Name | title:student.Gender}}</td>

<td>{{student.DOB | date:'dd/MM/yyyy'}}</td>

<td>{{student.DOB | age}}</td>

<td>{{student.Gender | lowercase}}</td>

<td>{{student.CourseFee | currency:'USD':true}}</td>

</tr>

</table>

.ts file:

import {TitlePipe} from './title.pipe';

import {AgePipe} from './age.pipe';

students:any[]=[

{ID:'std01',Name:'Sruthi',DOB:'05/28/2001',Gender:'Female',CourseFee:8328.45},

{ID:'std02',Name:'Aari',DOB:'12/20/2010',Gender:'Female',CourseFee:7782.83},

{ID:'std03',Name:'George',DOB:'05/30/2003',Gender:'Male',CourseFee:3728.83},

{ID:'std04',Name:'Swarup',DOB:'09/09/1997',Gender:'Male',CourseFee:9268.83},

{ID:'std05',Name:'Lekhana',DOB:'02/02/1990',Gender:'Female',CourseFee:2799.83}

];

Creating a user defined pipes using command:

ng g pipe title

ng g pipe age

.title.pipe.ts file:

transform(name: string, gender:string): unknown {

if(gender.toLowerCase()=="male")

return "Mr. "+name;

else

return "Mrs. "+name;}

.age.pipe.ts file:

transform(value: any): unknown {

let currentYear:any=new Date().getFullYear();

let userBirthYear:any=new Date(value).getFullYear();

let userAge=currentYear-userBirthYear;

return userAge;

}

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

**Angular Service:**

It is piece of code or logic that are used to perform specific task. Normally a function can contain a value or function or both.

The angular services are injected to the application using Dependency Injection (DI)

Whenever we need to use the same data and logic across multiple components a of application, then we need to use Angular Service

DRY (Don’t Repeat Yourself) concept is to be eliminated

The logic or data is implemented in a service and the service can be used across multiple components of our application

Steps to create Angular Service:

1. Export the class
2. Declare the class with @injectable decorator
3. We need to import the injectable decorator from the angular core library

Syntax: for creating a service 🡪 ng g service <<service name>>

Example:

.html file:

<table border="1">

<tr>

<th>ID</th>

<th>First Name</th>

<th>Last Name</th>

<th>Branch</th>

<th>DOB</th>

<th>Gender</th>

</tr>

<tr \*ngFor="let student of students">

<td>{{student.ID}}</td>

<td>{{student.FirstName}}</td>

<td>{{student.LastName}}</td>

<td>{{student.Branch}}</td>

<td>{{student.DOB}}</td>

<td>{{student.Gender}}</td>

</tr>

</table>

.ts file:

import {StudentService} from './student.service';

students:any[];

//can be declared using constructor

constructor(private \_StudentService: StudentService){

this.students=\_StudentService.getStudents();

}

/\*

private \_StudentService=new StudentService();

ngOnInit(){

this.students=this.\_StudentService.getStudents();

}

\*/

Creating a service:

ng g service student

student.service.ts file:

getStudents():any[]{

return [

{ID:'std01',FirstName:'Sruthi',LastName:'Mandalapu',Branch:'IT',DOB:'28/05/2001',Gender:'Female'},

{ID:'std02',FirstName:'Sireesha',LastName:'Mandalapu',Branch:'CSE',DOB:'20/11/1977',Gender:'Female'},

{ID:'std03',FirstName:'Karun',LastName:'Yadav',Branch:'ECE',DOB:'12/02/1990',Gender:'Male'},

{ID:'std04',FirstName:'Krithika',LastName:'Akshi',Branch:'EEE',DOB:'08/09/1968',Gender:'Female'},

{ID:'std05',FirstName:'Rachana',LastName:'Goud',Branch:'CIV',DOB:'13/12/2002',Gender:'Female'},

{ID:'std06',FirstName:'Samnith',LastName:'Kasi',Branch:'MECH',DOB:'03/06/2005',Gender:'Male'}

];

}

app.module.ts file:

import {StudentService} from './student.service';

In providers declare 🡪 StudentService

Difference between constructor and ngOnInit():

Whenever we create an instance of class, constructor automatically called which is similar to other programming languages. In Angular also, constructor used to initialize the number of classes and its dependency

The ngOnInit() is a lifecycle hook method provided by Angular which will be called after the constructor and it is generally used to perform tasks related to Angular bindings.

Services can be added either in component level (app.component.ts) of Module level (app.module.ts)

Example:

.html file:

<h1>{{title}}</h1>

<table border="1">

<tr>

<th>ID</th>

<th>First Name</th>

<th>Last Name</th>

<th>Branch</th>

<th>DOB</th>

<th>Gender</th>

</tr>

<tr \*ngFor="let student of students">

<td>{{student.ID}}</td>

<td>{{student.FirstName}}</td>

<td>{{student.LastName}}</td>

<td>{{student.Branch}}</td>

<td>{{student.DOB}}</td>

<td>{{student.Gender}}</td>

</tr>

</table>

.ts file:

import {StudentService} from './student.service';

title:string;

students:any[];

//can be declared using constructor

constructor(private \_StudentService: StudentService){

this.students=[];

this.title="";

}

//private \_StudentService=new StudentService();

ngOnInit(){

this.students=this.\_StudentService.getStudents();

this.title=this.\_StudentService.getTitle();

}

Creating a service:

ng g service student

student.service.ts file:

getTitle(){

return “Dependency Injection in Angular”;

}

getStudents():any[]{

return [

{ID:'std01',FirstName:'Sruthi',LastName:'Mandalapu',Branch:'IT',DOB:'28/05/2001',Gender:'Female'},

{ID:'std02',FirstName:'Sireesha',LastName:'Mandalapu',Branch:'CSE',DOB:'20/11/1977',Gender:'Female'},

{ID:'std03',FirstName:'Karun',LastName:'Yadav',Branch:'ECE',DOB:'12/02/1990',Gender:'Male'},

{ID:'std04',FirstName:'Krithika',LastName:'Akshi',Branch:'EEE',DOB:'08/09/1968',Gender:'Female'},

{ID:'std05',FirstName:'Rachana',LastName:'Goud',Branch:'CIV',DOB:'13/12/2002',Gender:'Female'},

{ID:'std06',FirstName:'Samnith',LastName:'Kasi',Branch:'MECH',DOB:'03/06/2005',Gender:'Male'}

];

}

app.module.ts file:

import {StudentService} from './student.service';

In providers declare 🡪 StudentService

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

**Dependency Injection:**

It is a technique where one subject supplies the dependencies of another object

Dependency means it is an object service that is going to be used by another object

Injection means it is a process of passing the dependency objection to the dependency object. Always it will create a new instance of the class along with its required dependencies

Example:

.html file:

<form #form\_obj="ngForm">

<div>

<label>Enter First Number: </label>

<input type="text" name="num1" [(ngModel)]="num1">

</div><br>

<div>

<label>Enter Second Number: </label>

<input type="text" name="num2" [(ngModel)]="num2">

</div><br>

<button (click)="add()">Add</button> &nbsp;&nbsp;&nbsp;&nbsp;

<button (click)="sub()">Subtract</button> &nbsp;&nbsp;&nbsp;&nbsp;

<button (click)="mul()">Multiply</button> &nbsp;&nbsp;&nbsp;&nbsp;

<button (click)="div()">Divide</button><br><br>

<p>The Result is: {{result}}</p>

</form>

.ts file:

import {ArithmeticService} from './arithmetic.service';

num1:number=0;

num2:number=0;

result:number=0;

\_calculate=new ArithmeticService();

add(){

this.result=this.\_calculate.add(this.num1,this.num2);

}

sub(){

this.result=this.\_calculate.subtract(this.num1,this.num2);

}

mul(){

this.result=this.\_calculate.multiply(this.num1,this.num2);

}

div(){

this.result=this.\_calculate.divide(this.num1,this.num2);

}

arithmetic.service.ts:

add(num1:number,num2:number):number{

return parseInt(num1.toString())+parseInt(num2.toString(0));

}

subtract(num1:number,num2:number):number{

return num1-num2;

}

multiply(num1:number,num2:number){

return num1\*num2;

}

divide(num1:number,num2:number){

return num1/num2;

}

app.module.ts:

import {ArithmeticService} from './arithmetic.service';

In providers declare 🡪 ArithmeticService

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

**Angular Routing:**

It is a mechanism or method which is used to navigating pages and displaying appropriate /relevant component/pages on the browser.

Routing used to navigate across the application from one view to another. It will also maintain the state, module implementation and loads the module based on user roles.

Angular project creation:

1. Create Angular Project (Verify the router modules in app.module.routing.ts)
2. Create Components
3. Import RouterModule and routes for route supporting in the project
4. Specify the component declaration through the route type
5. Define the router link or router outlet

2.

Angular Router used to provide navigate from one view to another based on process provided

4. Example:

const myRoutes:Routes[

{ path:”Customer”, component:CustomerComponent },

{ path:”Supplier”, componet:SupplierComponent },

{ path:”Product”, component:ProductComponent}

];

export default myRoutes;

All these routes should get registered either in app.module.ts or app-routing.module.ts

RouterModule.forRoot(myRoutes); 🡪 (Very Very Important)

5. Example:

<div>

<nav>

<a routerLink=”Product”>Click for Product</a>

<a routerLink=”Supplier”>Supplier Details</a>

<a routerLink=”Customer”>View Customers</a>

</nav>

</div>

<router-outlet></router-outlet>

Including routing to current angular project (which is non routing project):

To manually include routing feature in a project after it gets created

ng g module app-routing -flat -module=app

-flat 🡪 puts the file in specific folder. It will not create any default folder

-module 🡪 It will instruct Angular CLI to register it in the import array of AppModule

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

**Angular HTTP Client:**

HttpClient is used in Angular which is used to fetch external data, post and update, etc..

1. get() 🡪 read the details
2. post() 🡪create the details
3. put() 🡪 update the details
4. delete() 🡪 delete the details

To implement HTTP Client, we need to include HttpClientModule inside app.module.ts and also need to include HttpClient in “Component.ts” file.

Observables: It is a function that converts ordinary stream of data in observable stream of data. Provide details through subscribe.

Example:

app.component.html file:

<p>{{val}}</p>

app.component.ts file:

//getting data from demo text file

import HttpClient from ‘@angular/common/http’;

val:string="";

getdata(info:string){

this.val=info;

}

constructor(private http:HttpClient){

this.http.get("assets/demo.txt",{responseType:'text'}).subscribe(data => console.log(data));

this.http.get("assets/demo.txt",{responseType:'text'}).subscribe(data=>this.getdata(data));

}

app.component.ts file:

//getting text from url

import HttpClient from ‘@angular/common/http’;

val:any;

getdata(info:string){

this.val=info;

}

constructor(private http:HttpClient){

this.http.get(‘<http://jsonplaceholder.typicode.com/users>‘).subscribe(data=>getdata(data));

}

app.component.ts file:

//getting text from a json file

import data from ‘./users.json’;

val=data;

//for importing using json file need to modify the options in tsconfig.json file – (compiler options)

//in compiler options need to add the following requirements

//”resolveJsonModule”:true

//”esModuleInterop”:true

app.module.ts file:

import HttpClientModule from ‘@angular/common/http’;

In angular new feature called JSON server which is an npm package that allows us to create REST JSON web service backed by a simple database.

Install a JSON server package 🡪 npm install -g json-server

The default port number is 4200, but JSON it gets changed, we use 🡪 json-server --watch src/app/db.json --port 3000

Example:

AJAX 🡪 Asynchronous Javascript And XML

RXJS 🡪 Reactive Extensions for JavaScript 🡪 which will work with asynchronous data stream

RXJS has two major players:

1. Observable
2. Observer (Subscriber)

Observable is a function that converts ordinary stream of data to observable stream of data

Observable emits values from stream asynchronously. Observers communicate with the observable using callbacks (invoking functionally through events like keyboard events like keyup, keydown, keypress. Form events like value change, user notifications etc……)

The callback contains for methods:

next() 🡪 will be called whenever value arrives in the stream

error() 🡪 throws error during callback

complete() 🡪 when callback completed their work

ASP.NET Web API Service with SQL Server 🡪 Rest Service 🡪 Angular Observables

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

**JSON Server:**

Mock-API 🡪 json-server

Install json-server 🡪 npm install -g json-server

To load the JSON 🡪 json-server --watch <Device Specific Location/Project Specific Location> --port <portNo> (default port number is 3000)

Loading data from a json server

Adding a new data to a json server

Editing data from a json server

**------------------------------------------------------------------------------------------------------------------------------------------**

**Angular Material:**

It is a UI component library 🡪 for Angular Application

Supports Constructive, consistent and functional web page

Google developed Angular Material in 2014 which is tagged for Angular JS

Once again Google wrote code from scratch which is tagged to Angular from Sep 2016

Features:

1. Inbuilt rwd. Compatible with Android, iPhone, tablets and laptop
2. Angular Material has standard CSS
3. Angular Material is free to use

Installations: (in the angular application)

1. @angular/material
2. @angular/cdk
3. @angular/animations

Installation command: npm install @angular/material @angular/cdk @angular/animations –save

ng add @angular/material

AutoComplete:

It is a normal text input enhanced by a panel of suggested options

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

**CRUD Operations by using json-server:**

app.module.ts file:

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

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

app-routing.module.ts file:

import {HomeComponent} from './home/home.component';

import {CreateComponent} from './create/create.component';

import {EditComponent} from './edit/edit.component';

const routes: Routes = [

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

{path:"create",component:CreateComponent},

{path:"edit/:id",component:EditComponent}

{path:"delete/:id",component:DeleteComponent}

];

Create components HomeComponent, CreateComponent, EditComponent, DeleteComponent

home.component.html file:

<table border="1">

<tr>

<th>ID</th>

<th>Name</th>

<th>Age</th>

<th>Edit</th>

</tr>

<tr \*ngFor="let val of value">

<td>{{val.id}}</td>

<td>{{val.name}}</td>

<td>{{val.age}}</td>

<td><a [routerLink]="['/edit',val.id]">Edit</a></td>

</tr>

</table>

<button><a [routerLink]="['/create']">Add an Employee</a></button>

home.component.ts file:

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

export class HomeComponent {

value:any;

constructor(private http:HttpClient){

this.http.get("http://localhost:3000/employees").subscribe((data)=>{this.value=data});

}

}

edit.component.html file:

<form>

<label>Id</label>

<input type="text" name="id" [(ngModel)]="value.id"><br><br>

<label>Name</label>

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

<label>Age</label>

<input type="text" name="age" [(ngModel)]="value.age"><br><br>

</form>

edit.component.ts file:

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

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

export class EditComponent {

value={

id:0,

name:"",

age:0

};

id=0;

constructor(private http:HttpClient,private router:Router,private act:ActivatedRoute){

this.act.paramMap.subscribe((param)=>{this.id=Number(param.get('id'));});

alert(this.id);

this.http.get<Employees>(`http://localhost:3000/employees/${this.id}`).subscribe((data)=>{this.value=data;});

}

submit(){

this.http.put<Employees>(`http://localhost:3000/employees/${this.id}`,this.value).subscribe({

next:(data)=>{this.router.navigate(['/home']);}

});

}

}

interface Employees{

id:number,

name:string,

age:number

}

create.component.html:

<form>

<label>ID</label>

<input type="text" name="id" [(ngModel)]="value.id"><br><br>

<label>Name</label>

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

<label>Age</label>

<input type="text" name="age" [(ngModel)]="value.age"><br><br>

<input type="submit" (click)="submit()">

</form>

create.component.ts file:

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

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

export class CreateComponent {

value={

id:0,

name:'',

age:0

}

constructor(private http:HttpClient,private router:Router) {

}

submit(){

this.http.post("http://localhost:3000/employees",this.value).subscribe({ next:(data)=>{this.router.navigate(['/home'])} });

}

}

delete.component.ts:

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

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

export class DeleteComponent {

id:number=0;

constructor(private http:HttpClient,private router:Router,private act:ActivatedRoute){

this.act.paramMap.subscribe((param)=>{this.id=Number(param.get('id'));});

this.http.delete(`http://localhost:3000/tickets/${this.id}`).subscribe({

next:()=>{this.router.navigate(['/home']);alert("Deleted Successfully")}

});

}

}