Batch started on 12/10/18

31/10/18

Ex1:

interface Interface1{

fun\_one( ): string;

};

interface Interface2{

fun\_two( ): string;

};

interface Interface3 extends Interface1, Interface2{

fun\_three( ): string;

};

class my\_class implements Interface3{

public fun\_one( ): string{

return “Data from OracleDB soon…!”;

};

public fun\_two( ): string{

return “Data from MySQLDB soon…!”;

};

public fun\_three( ): string{

return “Data from MongoDB soon…!”;

};

};

var obj:my\_class = my\_class( );

document.write(obj).fun\_one()+”<br>”+ obj).fun\_two()+”<br>”+ obj).fun\_three());

index.html

<html>

<script src = “classes.ts” </script>

</html>

Ex2:

interface myInterface{

fun\_one( ):strubg;

};

abstract class class\_one implements myInterface{

public fun\_one( ):string{

return “Data from Oracle DB Soon..”;

};

public abstract fun\_two( ):string;

};

class class\_two extends class\_one{

public fun\_two( ):string{

return “Data from MongoDB soon…”;

};

};

var obj:class\_two = new class\_two()’

document.write(obj.fun\_one()+”<br>”+obj.fun\_two());

Modifiers in TypeScript:

In TypeScript, we have following modifiers.

1) public

2) private

3) proteted

The default Modifier in TypeScript is : "public"

1) "public" modifier:

----> public members available anywhere in application.

----> "public" modifier applicable to variables, functions, classes and

constructors also.

----> public members we can access through class objects / class references.

2) "private" modifier:

----> private memebrs we can access within class only.

----> private members we can access within class by using "this" keyword.

----> private modifier applicable to variables, functions, classes and constructors.

----> The recommended modifier for variables in classes is "private".

----> we can't access private members through class objects because of security.

----> we can initialize private members with constructors.

3) "protected" modifier:

----> protected members available to all child classes.

----> protected members we can’t access through class objects.

Example:

class class\_one{

// declare the protected variable

protected var\_one:string;

// initialize the protected members with the help of parameterized constructor.

constructor(arg1:string){

//assign arg1 to var\_one

this.var\_one=arg1;

};

//declare the protected function.

protected fun\_one():string{

//return protected variable

return this.var\_one;

};

};

class class\_two extends class\_one{

public myFun():any{

return this.fun\_one();

};

};

var obj:class\_two = new class\_two("Hello");

document.write(obj.myFun());

Example:

class class\_one{

private var\_one:string;

constructor(arg1:string){

this.var\_one=arg1;

};

};

var obj:class\_one = new class\_one("Hello");

obj.var\_one; // invalid, because, private members we can't access through class objects.

Example

class class\_one{

private var\_one:string="Hello";

};

class class\_two extends class\_one{

public myFun():string{

return this.var\_one; // invalid - private members of the parent

class won't available to child classes.

};

};

Example: combination of both protected and private members.

// create the class\_one

class class\_one{

// declare the private variable

private var\_one:string;

// initialize the "var\_one" with the help of parameterized constructor

constructor(arg1:string){

this.var\_one=arg1;

};

};

// create the public function

public fun\_one():string{

return this.var\_one;

};

};

// create the child class

class class\_two extends class\_one{

//declare the protected variable

protected var\_two:string;

// contructor

constructor(arg1:string, arg2:string){

super(arg1);

this.var\_two=arg2;

};

//create the public function

public fun\_two():string{

return this.var\_two;

};

};

var obj:class\_two = new class\_two("AngularJS", "Angular6");

document.write(obj.fun\_one() + "<br>"+obj.fun\_two());

01/111/18

JSON

----> JSON stands for Java Script Object Notation.

----> JSON is used to transfer the data over the Network.

----> JSON is network friendly format.

----> Parsing (Reading) of JSON is easy compared to XML.

----> JSON is light weight compared to XML.

**Syntax:**

1. Objects ---- { }
2. Arrays ---- [ ]
3. Data ----- key & value pairs.

Key & value separated by using “:”\.

Each “key & value” pairs are separated by using “,”.

Example-1: Create the JSON Object by using the following variable.

data.

JSON Object contain the following keys.

sub\_one

sub\_two

sub\_three

Values are:

Angular7

Node JS

MongoDB

Folder: JSON

File Noe: json.ts

var data:any = {

sub\_one : “Angular7”,

sub\_one : “Node JS”,

sub\_one : “MongoDB”

};

document.write(data.sub\_one+”<br>” + data.sub\_two+”<br>”+data.sub\_three));

index.js

<!DOCTYPE html>

<script src=”json.js”/>

</html>

Example-2: Create the JSON object by using the variable – “data”.

JSON keys are: Login, Logout, setCredentials, ClearCredentials.

JSON values are following Named Functions Definitions.

Login, Logout, setCredentials, ClearCredentials.

“Login” function return following message.

“Welcome to Login Module…!”

“Logout” function return following message.

“Welcome to Logout Module…!”

“setCredentials” function return following message.

“Welcome to setCredentials Module…!”

“clearCredentials” function return following message.

“Welcome to clearCredentials Module…!”

var data:any = {

Login : Login,

Logout : Logout,

SetCredentials : SetCredentials,

ClearCredentials : ClearCredentials

};

// values are function definitions.

function Login():string{

return “Welcome to Login Module…!”;

};

function Logout():string{

return “Welcome to Logout Module…!”;

};

function SetCredentials():string{

return “Welcome to SetCredentials Module…!”;

};

function ClearCredentials():string{

return “Welcome to ClearCredentials Module…!”;

};

document.write(data.Login()+”<br>” +

data.Logout()+”<br>”+

data.setCredentials()+”<br>+

data.clearCredentials());

Note: If we don’t specify “ () ” (i.e. data.Login+……) in the above statement, we will get function definitions. Check the output.

Example-3: Create the JSON object by using the variable – “data”.

JSON object contain the following keys:

oracle

mysql

mssql

mongodb

JSON values are arrow functions definitions.

First arrow function return “Oracle connection soon…”

Second arrow function return “MySql connection soon…”

Third arrow function return “SQL Server connection soon…”

Last arrow function return “MongoDB connection soon…”

var data : any = {

oracle : ( ):string=>{ return “Oracle connection soon..”},

mysql : ( ):string=>{ return “MySQL connection soon..”},

mssql : ( ):string=>{ return “SQL Server connection soon..”},

mongodb : ( ):string=>{ return “MongoDB connection soon..”}

};

document.write(data.oracle()+”<br>” +

data.mysql()+”<br>”+

data.mssql()+”<br>+

data.mongodb());

Transpilation process: view-->Terminal--> tsc json.ts

Run index.js.

Example-4: Create the following named function.

Fun\_one

“Fun\_one” return JSON object.

JSON key is : wish

JSON value is: “Welcome to JSON…!”

function Fun\_one():any{

return {

wish : “Welcome to JSON …!”

};

// document.write(Fun\_one()); --- check the output. It is JSON object.

document.write(Fun\_one().wish);

Example-5: Create the arrow function with following variable .

my\_fun

“my\_fun” return JSON object.

JSON key is : link

JSON value is : goo.gl/n1SkfM (whatsapp group link)

Call the arrow function.

Var my\_fun=( ):any=>{

return {

link : “goo.g1/nlSkfM”

};

};

document.write(my\_fun().link);

Example-6: Create the class with the following name.

class\_one

“class\_one” contain following function.

myFun( )

“myFun( )” returns JSON object.

JSON key is: wish

JSON value is: “Welcome to Angular 7”

Create the object to “class\_one”.

// create class

class class\_one{

// create function

public myFun( ):any {

return {

wish : “Welcome to Angular7….”

};

};

};

document.write( new class\_one( ).myFun( ).wish); // this is shortcut code.

index.html

<!DOCTYPE html>

<html>

<script src=”json.js”> </script>

</html>

01/11/18.

Example-7: create the JSON Array by using the following variable.

@data

JSON Array contain 5 JSON Objects.

Each JSON Object contain following keys.

@p\_id

@p\_name

@p\_cost

Code:

var data:Array<any> = [

{ “p\_id” : 111, “p\_name”: “p\_one”, “p\_cost”: 10000},

{ “p\_id” : 222, “p\_name”: “p\_two”, “p\_cost”: 20000},

{ “p\_id” : 333, “p\_name”: “p\_three”, “p\_cost”: 30000},

{ “p\_id” : 444, “p\_name”: “p\_four”, “p\_cost”: 40000},  
{ “p\_id” : 555, “p\_name”: “p\_five”, “p\_cost”: 50000}

];

for (var i:number=0; i<data.length; i++){

var obj:any = data[i];

document.write(obj.p\_id+”……..”+obj.p\_name+”……”+obj.p\_cost+”<br>”);

};

Example -8: Read the JSON data from the following URL.

URL: <https://restcountries.eu/rest/v2/all>

To know the structure of the JSON data, visit this website and “<http://jsonviewer.stack.hu/> “ and paste the JSON data, and click on Viewer tab to look the structure .

We got the following result.

--- Initially we have array.

--- Array contain 250 JSON Objects.

--- Each JSON object contain following keys.

1. name
2. capital
3. flag
4. currencies ==> Array ==> 0th position ==> Objects ==> code is the key in 0th position JSON Object.

Diagram:

var data:Array<any>= “copy the data from URL”;

for (var i:number=0; i<data.length;i++){

var obj:any = data[i];

document.write(obj.name+”…..”+

obj.capital+”…..”+

obj.currencies[0].code+”…..”+

”<img width=’100px’

height=’50px’ src=” + obj.flag+”> <br>”

Example-9: Read the JSON data from the following URL:

URL: <https://www.w3schools.com/angular/customers.php>

We got the following result.

-- Initially we have JSON Object.

-- JSON Object contain “records” Array.

-- “records” Array contain 15 JSON Objects.

-- Each JSON Object contain the following keys.

1) Name

2) City

3) Country

Design-2

var data : any = “copy the data from URL”;

for (va i:number=0; i<data.records.length; i++){

var obj : any = data.records[i];

document.write(obj.Name+”…”+

obj.City+”…”+

obj.Country+” <br> <br>” );

};

**Installation of Softwares to Build Angular7 Applications.**

1. Download and install NodeJS.

-- NodeJS is the scripting language.

-- NodeJS is used to Build the Servers.

-- Angular Environmental Setup depending on Node Server.

-- So, as a Angular Developer, we must download and install NodeJS.

Website: <https://nodejs.org/en/download/>

File: node-v10.13.0-x64.msi

**Note:** We can check whether Node JS is installed or not by executing

below commands at command prompt.

1. c:\> node -v

output: v10.13.0

2) c:\> npm –v

output: 6.4.1

1. Download and Install Git.

-- Git is the Automation Tool.

-- Git tool is used to interact with the GitHub.

-- GitHub is the open source repository.

-- Angular depending on GitHub Repository to download modules.

-- So, as a Angular Developer, we must download and install Git.

Website: <https://git-scm.com/download/win>

File: Git-2.19.1-64-bit.exe

**Note:** We can check whether Git is installed or not by executing

below command at command prompt

C:\> git --version

Output: git version 2.19.1.windows.1

1. **Download and install Visual Studio Code.**

-- Visual Studio Code is the IDE used to build the Software Applications like Angular, Node, React,…..

-- Visual Studio Code IDE is recommended to build Angular Applications.

Website: <https://code.visualstudio.com/docs/?dv=win>

File : vscodeUserSetup-x64-1.28.2.exe

1. Install yarn Package Manager:

-- yarn package manager used to download the node modules in faster manner.

-- we can install yarn packaging manager by using the following command.

> npm install -g yarn@latest

“npm” stands for node packaging manager.

“-g” stands for global installation.

Note: We can execute the above command at VSC ->

view->Terminal also

1. Install Angular Software:

-- we can install Angular by using the following command.

> npm install –g @angular/cli@latest ( in place of latest we can specify number like 2, 4, 5, 6 to install specific versions)

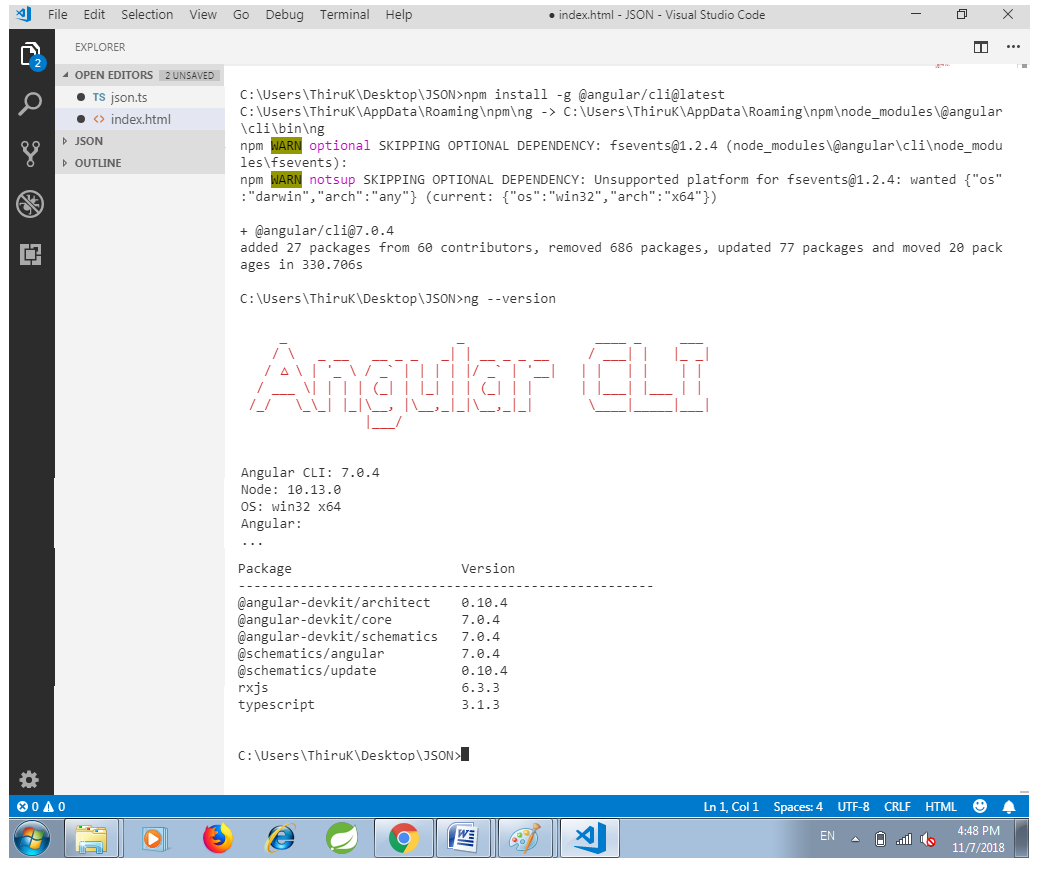
“cli” stands for command line interface

Note: We can execute the above command at VSC ->

view->Terminal also

we can check Angular 7 installation by using the command:

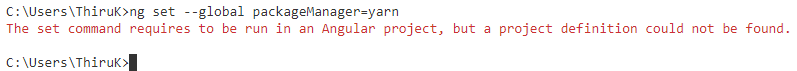
> ng --version



1. Set the yarn availability to Angular: i.e. [configure Yarn as the default package manager for Angular CLI?](https://stackoverflow.com/questions/49132455/how-do-i-configure-yarn-as-the-default-package-manager-for-angular-cli)

> ng set -- global packageManager=yarn (execute this command at VSC -> view->Terminal) (This command is applicable for Angular6. Use below command for Angular 7)

Error: The set command requires to be run in an Angular project, but a project definition could not be found.



> ng config -g cli.packageManager yarn

**Note:** Windows 10 may not allow to download through the

above command.

1. Create the Angular Application:

> ng new firstApp

-- automatically “firstApp” will be created.

-- where “firstApp” is the angular application.

8) Switch to Angular application.

> cd firstApp

-- automatically we can switch to Angular Application.

9) Execute Angular Application.

> ng serve --open (it is serve hyphen hyphen only, not server)

-- automatically “firstApp” will be deployed into “lite” server.

-- “lite” is the server given by Microsoft.

-- “lite” server is the web server used to execute the angular

applications.

-- by default lite server running on port no. 4200.

05/11/18

1) e2e folder:

In general testing is categorized to two types.

1. Manual Testing
2. Automation Testing.

Automation Testing is categorized into three types.

1. Unit Testing
2. Integration Testing
3. End-to-End Test.

End-to-End testing also called as e2e testing.

We will write e2e test cases in e2e folder in angular application

2) node\_modules:

node\_modules folder contains pre-defined modules.

These modules helps to execute the Angular7 Applications.

3) src/app:

App folder used to deploy our Angular 7 Applications.

4) src/app/app.module.ts:

It is similar to web.xml in JAVA Servlets.

This file called as configuration file.

This file is used to register our Angular Applications.

Angular Framework will execute our applications based configuration file.

5) src/assets:

This directory is used to deploy the static assets. Examples are Logos, fonts, headers, footers and so on.

6) environments:

In general, we have two types of environments.

1. Development Environment.
2. Production Environment.

We will configure required environment in “environments” directory.

7) **browserslist file**

This file shows supporting browsers of Angular7. i.e. it will display list of browsers supported by Angular7.

8) **fevicon.ico**

fevicon.ico representing the default logo of Angular Framework.

9) **src/index.html:**

index.html file is the main html file in Angular Project. Angular Framework starts the execution from this index.html file.

10) src/karma.conf.js:

“karmawithjasmine” is the automation tool.

Karmawithjasmine automation tool used to write the Unit Test cases to Angular applications.

“karma.conf.js” representing the configuration file for the karmawithjasmine tool.

11) main.ts:

Angular Framework starts the execution from index.html file.

index.html file called as main template in Angular project.

main.html also called as landing template in Angular project.

index.html file internally executes main.ts file.

main.ts internally executes configuration file (i.e. app.module.ts file, called as configuration file)

configuration file contains registration of our Angular Application / Project.

Angular Framework automatically executes our applications.

Application Execution flow:

index.html 🡪main.ts🡪app.module.ts🡪our Angular Application.

i.e. finally our application will be executed by Angular Framework with help of “app.module.js” only because this file contains registration of our Angular Application.

12) src/polyfills.ts:

In general, we will run Angular Applications in Browsers.

Browsers need help to execute Projects into Browser Engine.

“polyfills.ts” file provides help to Browser Engine to execute Angular Applications.

13) src/styles.css:

styles.css file is used to define the global styles.

styles.css file related styles applicable to whole angular project.

These styles called as global styles.

14) src/test.ts:

This file representing sample Unit Testing file.

From Angular7 onwards this file containing sample code of Unit Testing.

Upto Angular6 this file contains nothing.

15) src/tsconfig.app.json

src/tsconfig.spec.json

src/tslint.json:

These three files representing configuration files of TypeScript.

16) firstApp/angular.json:

This file explains the directory structure of our Angular Project/Application.

If we create the Angular Project by using npm automatically “angular.json” fill will be created in current path.

If we create the Angular Project by using yarn, automatically “angular-cli.json” file will be created in current path.

This file name may be changed based on Operating System also.

17) package.json file.:

In future, if Angular application wants to communicates with NodeJS, then this file is required.

package.json file used to download the node modules.

Examples: express, mongodb, multer, and so on.

Angular can communicate with NodeJS by using above modules.

**06/11/18**

**Component:**

-- Simple Typescript class behaves like component.

-- As a Angular developer we can create more than one Component.

-- As a Angular Developer, we can provide communication between Components.

-- Angular Applications are Component based Applications.

-- Because of Components code re-usability is high in angular.

-- Component acting as interface between View and Service.

-- The communication between View and Component is called as “Two-Way Data Binding”.

-- The communication between Component and Service called as “Dependency Injection”.

Project Directory Structure:

FirstApp

src

app

first.component.ts

first.component.html

app.module.ts (Registration file)

index.html (Landing Template)

-- where “first.component.ts” file used to create the TypeScript class. TypeScript class behaves like Component.

-- where “first.component.html” file acting as external template.

-- where “app.module.ts” file acting as configuration file.

-- where “index.html” file acting as landing template.

first.component.ts

/\*

-- import Component class.

-- Component class is the predefined class.

-- Component class available in “@angular/core” package.

-- Component class used to convert Typescript standards / Typescript code to HTML standards / HTML code.

-- we will use predefined classes with the help of “ @ “.

-- using predefined classes with “ @ “ , technically called as Decorator.

-- the argument for Component class constructor is JSON Object.

-- “selector” is the json key used to define custom tag or our own defined tag.

-- “templateUrl” is the json key used to link the external template to Components.

import {Component } from “@angular/core”;

@Component({

selector: “first”:,

templateUrl: “./first.component.html”

})

// “export” is the keyword in typescript.

// “export” keyword used to export the classes.

// anyone can import the exported classes.

export class firstComponent{

//declare the variables

// private modifier recommended for variables.

private sub\_one:string;

private sub\_two:string;

private sub\_three:string;

// initialize the above variables by using constructor

constructor( ){

// “this” is the keyword.

// “this” keyword used to refer the current class members.

this.sub\_one=”Angular7”;

this.sub\_two=“NodeJS”;

this.sub\_three=“Mongodb”;

};

// declare the getSubOne() function

public getSubOne():string{

return this.sub\_one;

};

// declare the getSubTwo() function

public getSubTwo():string{

return this.sub\_two;

};

// declare the getSubThree() function

public getSubThree ():string{

return this.sub\_three;

};

};

**first.component.html**

**<!—**

**{{ }} – used to display the result on templates.**

**{{ }} technically called as Expression / Interpolation.**

**-->**

**<h1 style=”color:red”> {{getSubOne()}}</h1>**

**<h1 style=”color:green”> {{getSubTwo()}}</h1>**

**<h1 style=”color:blue”> {{getSubThree()}}</h1>**

Index.html

**<body>**

**<first> </first>**

**</body>**

**app.module.js:**

**/\***

**-- BrowserModule is used to execute the applications into different browsers. The output of the “polyfills.ts” file is BrowserModule.**

**-- NgModule is used to create the Custom Modules.**

**-- AppComponent is the default Component.**

**-- firstComponent is our own Component.**

**-- declarations[ ] array used to register the Components, pipes, directives,…**

**-- imports[ ] array used to register the modules.**

**-- providers[ ] array used to register the services.**

**-- bootstrap[ ] array used to execute particular component.**

**\*/**

**import ( BrowserModule ) from ‘@angular/platform-browser’;**

**import ( NgModule ) from ‘@angular/core’;**

**import ( AppComponent ) from ‘./app.component’;**

**import ( firstComponent) from “./first.component”;**

**@NgModule( {**

**declarations: [**

**AppComponent, firstComponent**

**],**

**imports: [**

**BrowserModule**

**],**

**providers: [ ],**

**bootstrap: [ firstComponent]**

**})**

**export class AppModule{ }**

**08/11/18**

**09/11/18.**

map( ) – it is predefined function, used to catch the positive responsive given by the server.

catch( ) – it is also a predefined function, used to catch the error response given by the server.

throw( ) – it is also a predefined function, used to throw the error to the caller.

-- throw( ) function available in Observable class.

-- Observable class available in rxjs/Observable package.

map( ), catch( ) and throw( ) functions are available in rxjs/Rx package.

Design:

Directory structure:

Create Project by using below command

> ng new preSer1

preSer1

src

app

services

countries.service.ts

components

countries.component.ts

countries.component.html

app.module.ts

index.html

countries.service.ts:

//import Injectable

// Injectable is the pre-defined class.

// Injectable class used to create the Custom Service.

import { Injectable } from ‘@angular/core’;

// import Http

// Http is the predefined Service in Angular2.

// Http is used to make the Rest API calls.

// import Response class.

// Response is the pre-defined class in Angular2

// both Positive Response & Error Response, given by the Server, are “Response” Type.

Import { Http, Response } from “@angular/http”;

// import Observable class.

// throw( ) function available in Observable class

// throw( ) function used to throw the error to the caller.

import { Observable } from “rxjs/Observabe”;

// import “rxjs/Rx” package.

// map( ), catch( ) & throw( ) function’s implementations are available in the “rxjs/Rx” package.

import “rxjs/Rx”;

// use Injectable

@Injectable( )

// export the class

export class countriesService{

// create the instance to Http. Technically we call it “Dependency Injection”

// where “\_http” is the reference of Http.

constructor( private \_http:Http) { }

// create the Custom Function.

public getCountries( ):any{

// where getCountries( ) is the custom function.

//make the REST API call i.e. hitting the Server.

return this.\_http.get(<https://restcountries.eu/rest/v2/all>)

.map(this.fun\_one)

.catch(this.fun\_two)

// map( ) function argument is “arrow function definition”

// where “fun\_one” is the arrow function.

// where “fun\_one” is used to catch the positive response / positive result.

// catch( ) function argument is “arrow function definition”

// where “fun\_two” is the arrow function.

// where “fun\_two” is used to catch the error response / error result.

};

// create the arrow function i.e. “fun\_one”

public fun\_one = (res.Rsponse): any => {

// return res in the form of a JSON object

return res.json( );

};

// create another arrow function i.e. “fun\_two”

public fun\_two = (error.Response): any =>{

// throw the error

return Observable.throw(err || “Server Side Errors”); // sometimes there is no error then in this case we need to throw “Server Side Errors”

};

Rough:

// create the \_successCallBack

public \_successCallBack = (res.Response):any =>{

return res

// create the \_errorCallBack

public \_errorCallBack

----- end.

countries.component.ts:

// import Component

//Component class is used to create the Building Block of Angular Application.

import { Component } from “@angular/core”;

// import countriesService

import { countriesService } from “../services/countries”;

// import HttpErrorResponse

// “HttpErrorResponse” is the pre-defined class in Angular4.

// “HttpErrorResponse” class used to categorize the errors.(countriesService throws the errors and countriesComponent will catch these errors. We need to categorize these errors based on the client side errors or server side errors.

Import { HttpErrorResonse } from “@angular/common/http”;

// use Component

@Component( {

selector: “countries”,

templateUrl: “./countries.component.html”

})

// export class

export class countriesComponent{

// declare the variable to hold the Positive Result.

private result:any;

// “Component” class depending on subscribe( ) function which is used **to catch the REST data from Service.** The return type of subscribe ( ) function is Object type.

// declare the variable to hold the Object.

private countriesSubcribe: any;

// create the object to countriesService

// dependency injection.

constructor(private \_service: countiesService){ }

// where “\_service” is the countriesService reference.

// First life cycle hook.

ngOnInit( ){

this.countriesSubscribe = this.\_service.getCounties( )

.subscribe( this.\_succssCallBack, this.\_errorCallBack);

// where \_succssCallBack & \_errorCallBack are arrow function definitions.

};

// create the \_succssCallBack

public \_succssCallBack = (res): any => {

this.result = res;

};

// create the \_errorCallBack

public \_errorCallBack = (err:HttpErrorResponse):any=> {

if(err.error instanceof Error) // meaning is if value of “err.error” is of tye “Error”

{

console.log(“Client Side Errors !!!”);

} else {

console.log(“Server Side Errors !!!”);

}

};

// last life cycle hook, to kill the object

ngOnDestroy( ){

this.countriesSubscribe.unsubscribe( );

};

};

**12/11/18**

Drag and drop the folder Services

Create new project using below command:

> ng new preSer1

app-> services-> countries.service.ts

import { Injectable } from “@angular/core”;

import { Http, Response } from “@angular/http”;

import {Observable} from “rxjs/Observabe”;

Note: if there is any problem (i.e. this problem is not coding problem, it is a Os problem) while importing Observable, execute the following command

switch to Project

preServ1> npm install rxjs-compat@latest –save

import “rxjs/Rx”;

@Injectable( )

export class countriesService{

constructor(private \_http:Http){ };

// to consume url i.e. <https://restcountries.eu/rest/v2/all> we need to create one function, i.e. below function.

public getCountries( ):any{

return this.\_http.get(“<https://restcountries.eu/rest/v2/all>”)

.map(this.fun\_one) // if we got any problem use like:

pipe(map(this.fun\_one))

.catch(this.fun\_two); // catchError(this.fun\_two)

// here fun\_one and fun\_two are Anonymous function

};

public fun\_one = (res:Response):any=>{

return res.json( );

};

public fun\_two = (err:Response):any=>{

return Observable.throw(err || “Internal Server Error”);

};

};

Stept#2: Create Component file.

app/components/countries.component.ts

import { Component } from “@angular/core”;

import { countriesService } from “../services/countries.service”;

import {HttpErrorResponse } from “@angular/common/http”

//HttpErrorResonse is used to categorize the errors.

@Component( {

selector: “countries”,

templateUrl:”./countries.component.html”

})

export class countriesComponent{

private result:any;

private countriesSubscribe:any;

constructor(private \_service:countriesService) { } // technically this line is

called as Dependency Injection.

ngOnInit( ){

this.countriesSubscribe = this.\_service

.getCountries( )

.subscribe( this.\_successCallBack,

this.\_errorCallBack);

};

public \_successCallBack = (res):any=>{

this.result=res;

};

public \_errorCallBack= (err:HttpErrorResponse):any=>{

// categorize the error.

If(err.error instanceof Error) {

console.log(“Client Side Errors!!!!”);

} else{

console.log(“Server Side Errors !!!!”);

}

};

};

**Step#3**:app/components/countries.components.html

<table border=”1”

cellpadding=”10px”

cellspacing=”10px”

align=”center”

style=”font-size:20px;text-align:center”>

<thead style=”background-color: gray”>

<tr>

<th> Name</th>

<th> Capital</ th>

<th> Code</ th>

<th> Flag</ th>

</tr>

</thead>

<tbody>

<tr \*ngFor = “let x of result”> //\*ngFor is a structural type pre-defined directive.

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

<td> {{x.capital}}</td>

<td> {{x.currencies[0].code}}</td>

<td> <img width=”100px” height=”50px” src=”{{x.flag}}”</td>

</tr>

</tbody>

</table>

Step#4: register in app.module.ts

……

……

import {countriesComponent} from “./components/countries.component”;

import {counriesService} from “./services/countries.service”;

import { HttpModule } from “@angular/http”;

@NgModule( {

declarations : [ AppComponent, countriesComponent],

imports : [ BrowserModule, HttpModule],

providers : [ countriesService],

bootstrap : [coountriesComponent]

})

export class AppModule{ }

**Step#4:** index.html

<body>

<countries> </countries>

</body>

Step#5: Run the application

preServ1> ng serve –open

**13/11/18**

**Angular4**

**HttpClient**

-- HttpClient is the predefined class.

-- HttpClient is the predefined service introduced in Angular4.

-- HttpClient is used to make the REST API calls.

-- HttpClient available in “HttpClientModule”.

-- HttpErrorResonse is the predefined class introduced in Angular4.

-- HttpErrorResonse class used to categorize the Errors.

-- HttpClient, HttpClientModule & HttpErrorResponse available in “@angular/common/http” package.

-- Http deprecated in Angular4 to Angular7.

-- HttpClient applications are easy to write compared to Http Applications.

-- HttpClient Applications have more performance compared to Http Applications.

-- HttpClient Applications are easily Debuggable Applications.

-- Writing test cases to HttpClient Applications are easy compared to Http Applications.

-- HttpClient converts the Response to JSON Automatically.

-- HttpClient won’t depends on “rxjs” package.

**Example:**

URL: https://www.w3schools.com/angular/customers.php

Design.

Directory structure

preSer2

src

app

services

customers.service.ts

components

customers.component.ts

customers.component.html

app.module.ts

index.html

Create Folder: Services (next drag & drop into VSC)

Create Project > ng new preSer2

app/services/customers.service.ts:

// import Injectable

//Injectable class used to create the Custom Service

Import { Injectable } from “@angular/common/http”;

// use Injetable

@Injectable( )

// export class

export class customersService{

// create the object to the HttpClient

// Technically called as Dependency Injection.

constructor(private \_http:HttpClient) { }

// where \_http is the reference of HttpClient.

// create the custom function.

public getCustomers( ): any{

// make the REST API Call

return this.\_http.get(<https://www.w3schools.com/angular/customers.php>);

};

};

2) create component class.

**app/components/customers.component.ts:**

// import Component

// Component class used to create the building block of Angular application.

import { Component } from “@angular/core”;

// import customersService.

Import { customersService } from “../services/customers.service”;

// import HttpErrorResponse

import {HttpErrorResponse } from “@angular/common/http”

// use Component

@Component( {

selector: “customers”,

templateUrl:”../customers.component.html”

})

export class customersService{

// declare the result variable

// result variable used to hold the REST API result.

private result:any;

//Component catch the Rest Data by using subscribe() function.

// subscribe() function returns object.

// declare a variable to hold the subscribe function return object.

private customersSubscribe:any;

//create the object to the customersService

// Technically called as Dependency Injection.

constructor(private \_service: customersService) { }

// where \_service is the reference of Service.

// by using \_service we can call getCustomers( ) function.

// first life cycle hook.

// first life cycle hook is used to write Business logic.

ngOnInit() {

this.customersSubscribe = this.\_service.getCustomers( )

.subscribe(this.\_successCallBack,

this.\_errorCallBack);

};

// create the \_successCallBack

public \_successCallBack = (res):any=> {

this. result=res;

};

// create the \_errorCallBack

public \_errorCallBack=(err:HttpErrorResponse):any=>{

if(err.error instanceof Error){

console.log(“Client Side Error…”);

} else {

console.log(“Server Side Error….”);

}

};

// last life cycle hook

ngOnDestroy(){

// unsubscribe the Observable

// to overcome the memory leakage in application

this.customersSubscribe.unsubscribe();

};

};

**customers.component.html**

<table border= “1”

cellpadding = “10px”

cellspacing= “10px”

align = “center”

style=”font-size:30px; text-align:center”>

<thead style=”background-color:gray”>

<tr>

<th> SNO</ th>

<th> Name</ th>

<th> City</ th>

<th> Country</th>

</tr>

</thead>

<tbody>

<tr \*ngFor =”let x of result.records; let I = index”>

<td> {{i}}</td>

<td> {{x.Name}}</ td>

<td> {{ x.City}}</ td>

<td>{{ x.Country}} </ td>

</tr>

</tbody>

</table>

<!—

-- “index” is the constant in angular.

-- “index” constant will return indexes for each iteration.

-- “\*ngFor” is the structural directive.

-- \*ngFor directive used to iterate the elements from array.

-->

app.module.ts

import customersComponent, customersService, HttpClientModule (@angular/common/http)

declarations: cC

imports: HCM

providers: cS

bootstrap: cC

export class AppModule { }

index.html

<body>

<customers></customers>

Run the application.

preSer2>ng serve –open

**14/11/18**

**HttpPost:**

POST URL: <http://test-routes.herokuapp.com/testuppercase>

Directory Structure:

preSer3

src

app

services

uppercase.service.ts

components

uppercase.comonent.ts

uppercase.comonent.html

app.module.ts

indx.html

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

uppercase.service.ts

// import Injectable

//Injectable class used to create the Custom Service

import { Injectable } from “@angular/core”;

// import HttpClient

// HttpClient is used to make the REST API calls.

import {HttpClient} from “@angular/common/http”;

// use Injetable

@Injectable( ) // it is a Injectable Decorator

// export class

export class uppercaseService{

// create the object / instance to the HttpClient

// Technically called as Implicit Dependency Injection.

constructor(private \_http:HttpClient) { }

// where \_http is the reference of HttpClient.

// create the custom function.

// converToUppercae(-) is the Custom Function

// “obj” is the argument

// “obj” should come from Component through DI (as per MVC design)

public converToUppercae( obj: any):any{

// send the obj to the server.

return this.\_http.post(“<http://test-routes.herokuapp.com/testuppercase>”,

obj );

};

};

2) create component class.

**app/components/uppercase.component.ts:**

// import Component

// Component class used to create the Building Block of Angular application.

import { Component } from “@angular/core”;

// import uppercaseService.

import { uppercaseService } from “../services/uppercase.service”;

// import HttpErrorResponse

// HttpErrorResponse is used to categorize the Errors.

import { HttpErrorResponse } from “@angular/common/http”

// use Component

@Component( {

selector : “uppercase”,

templateUrl : ”../uppercase.component.html”

} )

// export the class

export class uppercaseComponent{

// declare variable to hold the result

private result:any;

// declare one more variable to hold Observable or return type of the subscribe() function.

private uppercaseSubscribe:any;

//create the object to the uppercaseService

// Technically called as Dependency Injection.

constructor(private \_service: uppercaseService) { }

// where \_service is the reference of Service.

// by using \_service we can call convertToUppercase( ) function.

// implement the clickMe( )

// clickMe( ) function will execute whenever we click the button

public clickMe(obj:any) : any {

this.uppercaseSubscribe = this.\_service.convertToUpperCase(obj)

.subscribe(this.\_successCallBack,

this.\_errorCallBack);

};

// implement the \_successCallBack

public \_successCallBack = (res):any=> {

this. result=res;

};

// implement the \_errorCallBack

public \_errorCallBack=(err:HttpErrorResponse):any=>{

if(err.error instanceof Error){

console.log(“Client Side Error…”);

} else {

console.log(“Server Side Error….”);

}

};

// last life cycle hook

ngOnDestroy(){

// unsubscribe the Observable, to overcome the memory leakage in application

this.uppercaseSubscribe.unsubscribe();

};

};

Note: Don’t us ngOnInit( ){….} because in UI, Button is there. When we click on button event will be raised. We need to handle that event by using clickMe( ) function.

**uppercase.component.html**

<input type = “text” [( ngModel )] =”my\_message”.

// ngModel directive will store the data, whatever we entered into text box, into my\_message variable.

<button (click) =”clickMe({‘message’: my\_message})”>Send</button>

<br>

<h1> {{ result | json }} </h1>

/\*

-- [(ngModel)] is the predefined directive.

-- [(ngModel)] directive present in FormModule.

-- [(ngModel)] directive created the application data with user defined variable.

-- [( )] symbol recognizing two day binding in angular.

-- (click) is the predefined directive.

-- ( ) symbol representing even in angular.

app.module.ts

// import uppercaseComponent

Import {uppercaseComponent } from “./components/uppercase.component”;

// import uppercaseService

import { uppercaseService } from “./services/uppercase.service”;

//Import HttpClientModule

import {HttpClientModule } from “@angular/common/ttp”;

// import FormsModule

//[(ngModel)] directive available in FormsModule.

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

@NgModule({

declarations : [ AppComponent, uppercaseComponent ],

imports : [ BrowserModule, HttpClientModule, FormModule],

providers : [ uppercaseService ],

bootstrap : [ uppercaseComponent ]

} )

export class AppModule { }

index.html

<body>

<uppercase></uppercase>

Run the application.

preSer3>ng serve –open

**15/11/18**

**Calls ( Series & Parallel )**

-- If we make the server calls with dependencies between them called as “Series calls”.

Call1

Call2

Call3

Call4

Call5

If Call1 success then Call2, if Call2 success then Call3, if Call3 success then Call4 etc…

-- If we make the server calls without any dependencies between them called as “Parallel calls”.

Call1

Call2 Call3

Call4 Call5

No relation between Call1 to Call2 and Call2 to Call3 and Call3 to Call4 etc.

All the Calls will be executed parallel. ( It is similar to MultiThread in Java)

URL1 –<https://restcountries.eu/rest/v2/all>

URL2 – <https://www.w3schools.com/angular/customers.php>

Directory Structure:

preSer4

src

app

services

countries.service.ts

customers.service.ts

components

series.component.ts

series.comonent.html

parallel.component.ts

parallel.component.html

app.module.ts

index.html

**countries.service.ts:**

import { Injectable } from “@angular/core”;

import { HttClient } from “@angular/common/http”;

@Injectable( )

export class countriesService{

constructor(private\_http:HttpClient) { };

public getCountries( ): any{

return this.\_http.get(“<https://restcountries.eu/rest/v2/all>”);

};

};

**customers.service.ts:**

import { Injectable } from “@angular/core”;

import { HttClient } from “@angular/common/http”;

@Injectable( )

export class customersService{

// create the object to HttpClient

constructor(private\_http:HttpClient) { };

// create custom function

public getCustomers( ): any{

return this.\_http.get(“<https://www.w3schools.com/angular/customers.php>”);

};

};

**series.component.ts:**

import { Component } from ”@angular/core”

import { countriesService } from ”../services/countries.service”;

import { customersService } from ”..services/customers.service”;

import { HttpErrorResponse } from ”@angular/common/http”;

@Component( {

selector :”series”,

templateUrl : “./series.component.html”

})

export class seriesComonent{

// declare variables to hold the results

private result\_one:any:

private result\_two:any:

// declare variables to holds subscribe( ) function return type.

private countriesSubscribe:any;

private customersSubscribe:any

// create the objects to Services

// Dependency Injection

constructor(private \_service1:countriesService,

private \_service2:customersService) { };

ngOnInit( ) {

this.countriesSubscribe= this.\_service1.getCountries()

.subscribe(this.\_countriesCallBack,

this.\_errorCallBack);

};

// create the \_countriesCallBack

public \_countriesCallBack = (res):any=> {

this.result\_one=res;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// making call2 (with the above line executed successfully means Call1 executed successfully.

this.customersSubscribe=this.\_service2.getCustomers( )

.subscribe(this.\_customersCallBack,

this.\_errorCallBack);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

};

// create the \_errorCallBack

public \_errorCallBack = (err:HttpErrorResponse):any=>{

if(err.error instanceof Error){

console.log(“Client side errors…”);

}else{

console.log(“Serverside Error…”);

}

};

// implement \_customersCallBack

pubic \_customersCallBack= (res)=> {

this. result\_two=res;

};

// last life cycle hook.

ngOndestroy( ){

this.countriesSubscribe.unsubscribe( ):

this.customersSubscribe.unsubscribe( ):

};

};

**series.component.html**

<h2 style=”color:red”> {{ result\_two | json}} </h2>

<h2 style=”color:green”> {{ result\_one | json}} </h2>

**app.module.js**

import { countriesService} from ”./services/countries.service “;

import { customersService} from ”./ services/customers.service “;

import { seriesComponent} from ” ./components/series.component “;

import { HttpClientModule} from” @angular/common/htt “;

@NgModule( {

declarations: [AppComponent, seriesComponent ],

imports : [ BrowserModule, HttpClientModule ],

providers : [ countriesService, customersServie ],

bootstrap :[ seriesComponent ]

} )

export class AppModule { }

**index.html:**

<body>

<series> </series>

</body>

Run the application:

preSer4> ng serve - -open

**16/11/18**

**parallel.component.ts:**

// import Component

// Component used to convert the TypeSscript standards to equal HTML standards.

import { Component } from ”@angular/core”

// import countriesService

import { countriesService } from ”../services/countries.service”;

// import customersService

import { customersService } from ”..services/customers.service”;

// import HttpErrorResponse

// HttpErrorResponse is used to categorize the Errors

import { HttpErrorResponse } from ”@angular/common/http”;

// import Observable Class

// we will use forkJoin( ) function to make parallel calls.

// forkJoin( ) function is available in Observable Class.

Import {Observable} from “rxjs/Observable”;

// import rxjs

Import “rxjs/Rx”;

//use Component

@Component( {

selector :”parallel”,

templateUrl : “./parallel.component.html”

})

export class parallelComonent{

// declare two variables to hold the results

private result\_one:any:

private result\_two:any:

// declare variables to hold Observable

private parallelSubscribe:any;

// create the objects / instances to Services

// Dependency Injection

constructor(private \_service1:countriesService,

private \_service2:customersService) { };

ngOnInit( ) {

// make parallel call

this.parallelSubscribe =

Observable.forkJoin([this.\_service1.getCountries(), this.\_service2.getcustomers()])

.subscribe(this.\_successCallBack, this.\_errorCallBack);

};

// create the \_successCallBack

public \_successCallBack = (res):any=>{

this.result\_one = res[0];

this.result\_two = res[1];

};

// create the \_errorCallBack

public \_errorCallBack = (err:HttpErrorResponse):any=> {

if(err.error instanceof Error){

console.log(“Client Side Error…!”);

}else{

console.log(“Server Side Error..!);

}

};

// last life cycle hook.

ngOndestroy( ){

this.parallelSubscribe.unsubscribe( ):

};

};

**parallel.component.html**

<h2 style=”color:red”> {{ result\_one | json}} </h2>

<h2 style=”color:green”> {{ result\_two | json}} </h2>

**app.module.js ( needs to complete, just copied, check it)**

import { countriesService} from ”./services/countries.service “;

import { customersService} from ”./ services/customers.service “;

import { seriesComponent} from ” ./components/series.component “;

import { HttpClientModule} from” @angular/common/htt “;

@NgModule( {

declarations: [AppComponent, seriesComponent ],

imports : [ BrowserModule, HttpClientModule ],

providers : [ countriesService, customersServie ],

bootstrap :[ seriesComponent ]

} )

export class AppModule { }

**index.html:**

<body>

<series> </series>

</body>

Run the application:

preSer4> ng serve - -open

**Java Integration**

[ i.e. Consuming Java Application ]

-- “EmployeeDetailRestResource” is the Java Webservices Application.

-- We will deploy “EmployeeDetailRestResource” Application in Tomcat Server.

-- The following URL representing the Get Request.

<http://localhost:9090/EmployeeDetailRestResource/api/empService/getAll>

-- The above URL gives the JSON Array, i.e. Array of JSON objects, as Response.

-- The following URL representing the POST requests

<http://localhost:9090/EmployeeDetailRestResource/api/empService/addEmp>

-- The above URL gives the JSON Array as Response.

**Directory Structure:**

preSer5 (Get request)

src

app

services

java.service.ts

components

java.component.ts

java.component.html

app.module.ts

index.html.

java.service.ts:

import { Injectable } from “@angular/core”;

import { HttpClient } from “@angular/common/http”;

@Injectable( )

export class javaService{

constructor( private \_http:HttpClient) {};

public getEmployees():any{

Return this.\_http.get(“<http://localhost:9090/EmployeeDetailRestResource/api/empService/getAll> “);

};

};

java.component.ts:

import { Component } from “@angular/core”;

import {JavaService } from “../services/java.service”;

import { HttpErrorResonse } from “@angular/common/http”;

// use Component

@Component( {

selector: “java”,

templateUrl: “./java.component.html”

})

// export class

export class javaComponent{

// declare the variable to hold the Positive Result.

private result:any;

// “Component” class depending on subscribe( ) function which is used **to catch the REST data from Service.** The return type of subscribe ( ) function is Object type.

// declare the variable to hold the Object.

private javaSubcribe: any;

// create the object to javaService. Object name is: “\_service”

// dependency injection.

constructor(private \_service: javaService){ }

// First life cycle hook.

ngOnInit( ){

this.javaSubscribe = this.\_service.getEmployees( )

.subscribe( this.\_successCallBack, this.\_errorCallBack);

// where \_successCallBack & \_errorCallBack are arrow function definitions.

};

// create the \_successCallBack

public \_succssCallBack = (res): any => {

this.result = res; // result contains JSON array which contains Employee objects with 5 keys, which is coming from JAVA application.

};

// create the \_errorCallBack

public \_errorCallBack = (err:HttpErrorResponse):any=> {

if(err.error instanceof Error) // meaning is if value of “err.error” is of tye “Error”

{

console.log(“Client Side Errors !!!”);

} else {

console.log(“Server Side Errors !!!”);

}

};

// last life cycle hook.

ngOnDestroy( ){

// We need to kill the object i.e. “javaSubscrie” by using “unsubscribe( )” function.

this.javaSubscribe.unsubscribe( );

};

};

**java.component.html:**

<table border=”1”

cellpadding=”10px”

cellspacing=”10px”

align=”center”

style=”font-size:20px; “font-wight:bold”>

<thead style=”background-color: gray”>

<tr>

<th> eno</th>

<th> ename</ th>

<th> sal</ th>

<th> dept</ th>

<th> addrs</ th>

</tr>

</thead>

<tbody>

<tr \*ngFor = “let x of result”> //\*ngFor is a structural type pre-defined directive.

<td> {{x.eno}} </td>

<td> {{x.ename}} </td>

<td> {{x.sal}} </td>

<td> {{x.dept}} </td>

<td> {{x.addrs}} </td>

</tr>

</tbody>

</table>

**app.module.ts**

-----

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

import {javaComponent} from “./components/java.component”;

import {javaService} from “./services/java.service”;

import { HttpClientModule } from “@angular/common/http”;

@NgModule( {

declarations : [ AppComponent, javaComponent],

imports : [ BrowserModule, HttpClientModule],

providers : [ javaService],

bootstrap : [javaComponent]

})

export class AppModule{ }

**index.html:**

<body>

<java> </java>

</body>

**Note:**

-- Java Application running in Tomcat Server.

-- Tomcat Server running on port no. 9090.

-- Angular Application running on Lite server with port no. 4200.

-- Enabling the communication between different Servers called as “CORS”

(Cross Orign resource sharing)

-- We can enable the CORS at Browser side by using CORS Plugin.

(Serach in google with CORS plugin to Google Chrome) -> click on “Add to Chrome”

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Directory Structure for 6th application.**

preSer6 (POST request)

src

app

services

emp.service.ts

components

emp.component.ts

emp.component.html

app.module.ts

index.html.

**19/11/18**

**emp.service.ts**

// import Injectable

//Injectable class is used to create the Custom Service

import { Injectable } from “@angular/core”;

// import HttpClient

// HttpClient is used to make the REST API calls.

import {HttpClient} from “@angular/common/http”;

// use Injetable

@Injectable( ) // it is a Injectable Decorator

// export class

export class empService{

// create object / instance to the HttpClient

// Technically called as Implicit Dependency Injection.

constructor(private \_http:HttpClient) { }

// where \_http is the reference of HttpClient.

// create the custom function.

// where “addEmp” is the Custom Function

// “obj” is the argument and it is JSON object which contain eno, ename, sal, addrs, dept.

// “obj” should come from Component through DI (as per MVC design)

public addEmp( obj: any):any{

// send the obj to the server.

return this.\_http.post(“<http://localhost:9090/EmployeeDetailRestResource/api/empService/addEmp> ”, obj );

};

};

2) create component class.

**app/components/emp.component.ts:**

// import Component

// Component class used to create the Building Block of Angular application i.e. used to convert Typescript code to equivalent JavaScript code.

import { Component } from “@angular/core”;

// import empService.

import { empService } from “../services/emp.service”;

// import HttpErrorResponse.

// HttpErrorResponse is predefined Class introduced in Angular4.3

// HttpErrorResponse class is used to categorize the Errors.

import { HttpErrorResponse } from “@angular/common/http”

// use Component

@Component( {

selector : “emp”,

templateUrl : ”../emp.component.html”

} )

// export the class

export class empComponent{

// declare variable to hold the result

private result:any;

// declare one more variable to hold Observable or return type of the subscribe() function.

private empSubscribe:any;

//create the object to the empService

// technically called as Dependency Injection.

constructor(private \_service: empService) { }

// where \_service is the reference of Service.

// by using \_service we can call addEmp( ) function.

// implement the function.

// register( ) function will execute with click / submit event.

// argument for the register( ) function is “obj”.

// “obj” should come from View (because currently we are in Component).

public register(obj:any) : any {

// send obj to the service.

this.empSubscribe = this.\_service.addEmp(obj)

.subscribe(this.\_successCallBack,

this.\_errorCallBack);

};

// implement the \_successCallBack

public \_successCallBack = (res):any=> {

this. result=res;

};

// implement the \_errorCallBack

public \_errorCallBack=(err:HttpErrorResponse):any=>{

if(err.error instanceof Error){

console.log(“Client Side Error…”);

} else {

console.log(“Server Side Error….”);

}

};

// last life cycle hook

ngOnDestroy(){

// unsubscribe the Observable, to overcome the memory leakage in application

this.empSubscribe.unsubscribe();

};

};

Note: Don’t use ngOnInit( ){….} because in UI, Button is there. When we click on button event will be raised. We need to handle that event by using register( ) function.

**emp.component.html**

<div class = “container”>

// where “container” is a class in Bootstrap used to display the content at centre of the screen.

<form (ngSubmit) = “register({

‘eno’: eno,

‘esal’: esal,

‘addrs’: addrs,

‘dept’: dept

})”>

<!-------eno ------->

<div class=”form-group”>

<label> Eno. </

<input type =”number”

class=”form-control”

name=”eno”

[(ngModel)] = “eno”>

</div>

<!-------ename ------->

<div class=”form-group”>

<label> Ename. </

<input type =”text”

class=”form-control”

name=”ename”

](ngModel)] = “ename”>

</div>

<!-------sal ------->

<div class=”form-group”>

<label> Sal. </

<input type =”number”

class=”form-control”

name=”sal”

](ngModel)] = “sal”>

</div>

<!-------addrs ------->

<div class=”form-group”>

<label> Address. </

<input type =”number”

class=”form-control”

name=”addrs”

](ngModel)] = “addrs”>

</div>

<!-------dept ------->

<div class=”form-group”>

<label> Dept. </

<input type =”number”

class=”form-control”

name=”dept”

](ngModel)] = “dept”>

</div>

<!-------submit ------->

<div class=”form-group” align=”center”>

<input type=”submit” class=”btn btn-success”>

</div>

</form>

<h2 style=”color:red”> {{result | json}} ></h2>

</div>

app.module.ts

import { BroserModule} from “ @angular/platform-browser”;

import {NgModule }angular/core;

import AppComponent from

HttpClientModule

FormModule

// import empComponent

Import {empComponent } from “./components/emp.component”;

// import uppercaseService

import { uppercaseService } from “./services/uppercase.service”;

//Import HttpClientModule

import {HttpClientModule } from “@angular/common/ttp”;

// import FormsModule

//[(ngModel)] directive available in FormsModule.

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

@NgModule({

declarations : [ AppComponent, empComponent ],

imports : [ BrowserModule, HttpClientModule, FormModule],

providers : [ empService ],

bootstrap : [ empComponent ]

} )

export class AppModule { }

index.html

<head>

link rel=”stylesheet”

href=”<https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.js>“

</head>

<body>

<emp></emp>

</body>

Run the application. (Make sure before executing this application, enable CORS in chrome browser).

preSer3>ng serve –open

**28/11/18**

Drag and drop Services folder into VSC.

**server.js**

// import express module

var express = require(“express”);

// import body-parser module

var bodyparser=require(“body-parser”);

// import cors module

var cors= require(“cors”);

// create the REST Object.

var app=express( );

// set the JSON as MIME Type

app.use(bodyparser.json( ));

// take the permission to read POST Parameters

app.use(bodyparser.urlencoded({extended:false}));

// use cors module.

app.use(cors( ));

// import login module

var login=require(“./login/login”);

// use login module

app.use(“/login”, login); (first argument is “url” and second argument is

login module name

// assign the port no.

app.listen(8080);

console.log(“Server listening the port no. 8080”);

**Step#5:** Start the Node Server.

>cd preSer10

>cd server

>node server

-- automatically node server will start on port no. 8080.

**Step#6:** Test the following REST API by using Postman tool. (post request)

url: <http://localhost:8080/login>

enter below json object in body part

{

“uname”: “admin”

“upwd”: “admin”

}

**Stemp#7:** Create the Custom Service:

preSer10

src

app

services

login.service.ts

**login.service.ts**

// import Injectable

import { Injectable } from “@angular/core”;

// import HttpClient

import { Httpclient } from “@angular/common/http”;

// use Injectable

@Injectable( )

// export the class

export class loginServie{

// create the object to HttpClient

Constructor(private \_http:HttpClient){ }

//create the custom function

public authenticate(obj:any):any{

// where “authenticate( )” is the custom function.

// the argument for authenticate function is “obj”

// “obj” is the JSON object which contain uname & upwd.

// “obj” coming from the Component through Dependency Injection

// send the obj to the Server

return this.\_http.post(“<http://localhost:8080/login>”, obj);

};

};

**Step#8:** Create the Custom Component.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

preSer10

src

app

components

login.component.ts

login.component.html

app.modules.ts

index.html (landing template)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**login.component.ts:**

// import Component

import { Component } from “@angular/core”;

// import loginService

Imort { loginService } from “../services/login.service”;

// import HttpErrorResponse

import { HttpErrorResponse } from “@angular/common/http”;

// use Component

@Component( {

selector : “login”,

templateUrl : ”./login.component.html’

} )

//export the class

export class loginComponent{

// declare variable to hold result

private result:any;

//declare variable to hold Observable

private lognSubscribe:any;

// create the object to loginService

constructor(private \_service:loginService) { }

// create the event function

public login(obj:any):any{

// the name of custom function is “login”

// “login( )” function will execute whenever we click on the “login” button.

// the argument for “login( )” function is obj.

/ obj is the JSON Object

// obj contain uname and upwd.

//obj should come from View through Two-way Data Binding.

// make the service call

this.loginSubscribe =

this.\_service.authenticate(obj)

.subscribe(this.\_successCallBack, this.\_errorCallBack);

};

// create the successCallBack

pubic \_successCallBack = (res):any =>{

this.result=res;

};

// create the errorCallBack

public \_errorCallBack=(err:HttpErrorResponse): any=>{

if(err.error instanceof Error){

console.log(“Client Side Error …”);

}else{

console.log(“Server Side Error….”);

}

};

// last life cycle hook

ngOnDestroy(){

this.loginSubscribe.unsubscribe( );

};

};

**login.component.html:**

<div class=”container”>

<form (ngSubmit)=”login({‘uname’: uname, ‘upwd’: upwd})”>

<!—uname -->

<div class=”form-group”>

<label>Uname</label>

<input type=”text” name=”uname”

class=”form-control”

[(ngModel)]=”uname”>

</div>

<!—upwd -->

<div class=”form-group”>

<label>Upwd</label>

<input type=”password” name=”upwd”

class=”form-control”

[(ngModel)]=”upwd” >

</div>

<!—submit -->

<div class=”form-group” align=”center”>

<input type=”submit” class=”btn btn-success”>

</div>

</form>

<h2>{{ result }} | json }} </h2>

</div>

Note: In this html file “class” indicates Bootstrap.

app.module.ts

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

import { NgModule } from @angular/core;

import { AppComponent } from “./app.component”;

import { loginComponent } from “./components/login.component”;

import { loginService } from “./service/login.service”;

import { HttpClientModule } from “@angular/common/http”;

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

@NgModule( {

declarations : [Appcomponent, loginComponent ],

imports : [ BrowserModule, HttpClientModule, FormModule ],

providers: [ loginService],

bootstrap: [ loginComponent]

} )

export class AppModule{ }

**index.html:**

<head>

<link rel=”stylesheet”

href=”<https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css>”>

<link rel=”icon” type=”image/x-icon” href=”favicon.ico”>

</head>

<body>

<login> </login>

</body>

</html>

Note: After executing the above application we will get Token code, we will use this code in future applications.

**MongoDB CRUD Operations:**

-- MongoDB is the light weight database.

-- MongoDB is document oriented NoSQL database.

-- MongoDB follows the Client - Server Architecture.

-- Mongo Server running on default Port No: 27017.

-- MongoDB follows the “mongodb” protocol.

-- MongoDB supports JSON.

-- MongoDB is JSON based database.

-- As a MongoDB developer we can perform CRUD operations on JSON.

**Installation of MongodB**

**Step#1:** Download and install MongoDB by using following website.

WebSite : <https://www.mongodb.com/download-center#community>

File Name: mongodb-win32-x86\_64-2008plus-ssl-4.0.4-signed.msi

**Step#2:** Create the following directory structure

Ex: c:/data/db

-- above directory structure used to maintain the data

backup.

**Step#3:** Set the “path” environmental variable.

path=C:\Program Files\MongoDB\Server\4.0\bin

**Steps to implement the above application**

Drag &Drop “services” folder into vsc

Step#1: Make the mongodb ready for CRUD operations.

**MongoDB queries:**

1. **Start the server**

> mongod

-- automatically mongo server will start on port no.27017.

-- mongo server follows the “mongodb” protocol.

1. **Connect to Server:**

> mongo

-- automatically we are able to connect to server.

1. **Create and switch to databse**

> use angular11am;

-- automatically angular11am database will be created and switches also.

4) **Create the collection (table)**

> db.createCollection(“products”);

-- automatically “products” collection will be created.

5) **insert the JSON Object:**

> db.products.insert({“p\_id”:111, “p\_name”: “p\_one”, “p\_cost”:10000});

> db.products.insert({“p\_id”:222, “p\_name”: “p\_two”, “p\_cost”:20000});

> db.products.insert({“p\_id”:333, “p\_name”: “p\_three”, “p\_cost”:30000});

> db.products.insert({“p\_id”:444, “p\_name”: “p\_four”, “p\_cost”:40000});

> db.products.insert({“p\_id”:555, “p\_name”: “p\_five”, “p\_cost”:50000});

-- automatically five JSON objects will be inserted.

6) **fetch the data from collection**

> db.products.find( );

-- automatically we are able to fetch the data from products collection.

7) **show the available databases:**

> show dbs;

-- automatically it will show the available databases.

8) **drop the database**

>db.dropDatabase( );

-- automatically we are able to drop the database.

Note: to drop the table first we need to switch to the database using ”use” command.

**Step#2: Download the following Node modules.**

1. express
2. [mongodb@2.2.32](mailto:mongodb@2.2.32)
3. body-parser
4. cors

-- “express” module used to develop the Rest APIs.

-- “mongodb” module used to interact with the mongodb database.

-- the stable version of mongodb is 2.2.32.

-- “body-parser” module used to read the post parameters.

-- “cors” module used to enable the CORS.

> yarn add express [mongodb@2.2.32](mailto:mongodb@2.2.32) body-parser cors - -save

**Step#3: Develop the Rest APIs by using Node**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Directory structure:**

preSer11

server

fetch

fetch.js

insert

insert.js

update

update.js

delete

delete.js

server.js

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**fetch.js:**

// import express module

// express module used to develop the Rest APIs

var express=require(“express”);

// import mongodb module

// mongodb module used to interact with the MongoDB database.

var mongodb=require(“mongodb”);

// create the mongo client.

// MongoDB follows the client server architecture. So we need to create Mongo Client.

var nareshIT=mongodb.MongoClient;

// where nareshIT is the mongo client.

// create the router instance.

// router instance used to create the modules.

var router = express.Router( );

// create the Rest API

router.get(“/”, (req, res)=>{

nareshIT.connect(“mongodb://localhost:27017/angular11am”, (err, db)=>{

db.collection(“products”)

.find()

.toArray((err, array)=>{

/\* where toArray() is used to store all the json objects of “products” collection. These Json objects may store or may not store. “err” variable in the first argument is used to store error messages and “array” variable in the second argument used to story success messages. \*/

if(err){

res.send(“Error while fetching the data…”)

}else{

res.send(array);

}

} );

} );

} );

// export the router.

module.exports = router;

**insert.js**

var express = require(“express”);

var mongodb = require(“mongodb”);

var nareshIT = mongodb.MongoClient;

var router = express.Router( );

router.post(“/”, (req, res) => {

// read p\_id, p\_name & p\_cost coming from angular.

// “body” is the key in request object used to read the post parameters.

var p\_id = req.body.p\_id;

var p\_name = req.body.p\_name;

var p\_cost = req.body.p\_cost;

// create the JSON object.

var obj = { “p\_id” : p\_id, “p\_name” : p\_name, “p\_cost” : p\_cost } ;

nareshIT.connect(“mongodb://localhost:27017/angular11am”, (err, db)=>{

// the above arrow function is used to handle the situation i.e. mongodb client

i.e. nareshIT may connect or or may not connect to angular11am db

db.collection(“products”).insertOne(obj, (err, result)=>{

if(err){

res.send( { “insert” : “fail” } ) ;

}else{

res.send( { “insert” : “success” } );

}

} );

} );

} );

// export the router.

module.exports = router;

**update.js**

var express = require(“express”);

var mongodb = require(“mongodb”);

var nareshIT = mongodb.MongoClient;

var router = express.Router( );

router.post(“/”, (req, res) => {

// read p\_id, p\_name & p\_cost coming from angular.

// “body” is the key in request object used to read the post parameters.

var p\_id = req.body.p\_id;

var p\_name = req.body.p\_name;

var p\_cost = req.body.p\_cost;

// We need to update “p\_name” & “p\_cost” based on “p\_id”. For this we have to create two json objects.

// create first JSON object.

var obj\_one = { “p\_id” : p\_id } ;

// create second JSON object.

var obj\_two = { $set: { “p\_name” : p\_name, “p\_cost” : p\_cost } } ;

// create MongoDB client

nareshIT.connect(“mongodb://localhost:27017/angular11am”, (err, db)=>{

// the above arrow function is used to handle the situation i.e. mongodb client,

i.e. nareshIT, may connect or may not connect to “angular11am” db.

db.collection(“products”)

.updateOne(obj\_one, obj\_two (err, result)=>{

if(err){

res.send( { “update” : “fail” } ) ;

}else{

res.send( { “update” : “success” } );

}

} );

} );

} );

// export the router.

module.exports = router;

**delete.js**

var express = require(“express”);

var mongodb = require(“mongodb”);

var nareshIT = mongodb.MongoClient;

var router = express.Router( );

router.post(“/”, (req, res) => {

// read p\_id coming from angular.

// “body” is the key in request object used to read the post parameters.

var p\_id = req.body.p\_id;

// create JSON object.

var obj = { “p\_id” : p\_id } ;

// create MongoDB client

nareshIT.connect(“mongodb://localhost:27017/angular11am”, (err, db)=>{

// the above arrow function is used to handle the situation i.e. mongodb client,

i.e. nareshIT, may connect or may not connect to “angular11am” db.

db.collection(“products”)

.deleteOne(obj, (err, result)=>{

if(err){

res.send( { “delete” : “fail” } ) ;

}else{

res.send( { “delete” : “success” } );

}

} );

} );

} );

// export the router.

module.exports = router;

**30/11/18**

**server.js**

// import express module

var express = require(“express”);

// import body-parser module

var bodyparser=require(‘body-parser”);

// import cors module

var cors=require(“cors”);

// create the Rest Object

var app=express();

// set the JSON as MIME type

app.use(bodyparser.json());

// take the permission to read post parameters

app.use(bodyparser.urlencoded({extended:false}));

// use the cors module

app.use(cors());

//import fetch module

var fetch=require(“./fetch/fetch”);

// use fetch module

app.use(“/fetch”, fetch);

// import insert module

var insert = require(“./insert/insert”);

// use insert module

app.use(“/insert”, insert);

// import update module

var update = require(“./update/update”);

// use update module

app.use(“/update”, update);

// import delete module

var remove = require(“./delete/delete”);

// use delete module

app.use(“/delete”, remove);

// assign the port no.

app.listen(8080);

console.log(“Server Listening the port no. 8080”);

**Step#4: Start the Node server**

C:\user\thiruk\desktop\services\preSer11\server> node server

**--** automatically Node server will listen the port no 8080.

**Step5: Test the following four Rest APIs by using Postman tool.**

<http://localhost:8080/fetch> (GET Request)

<http://localhost:8080/insert> (POST Request)

<http://localhost:8080/update> (POST Request)

<http://localhost:8080/delete> (POST Request)

**Step#6: Create the Custom Services for below Rest APIs. (Similar to Consumer Application in Java)**

**Directory Structure**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

preSer11

src

app

services

fetch.service.ts

insert.service.ts

update.service.ts

delete.service.ts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**fetch.service.ts:**

//import Injectable

// Injectable is the pre-defined class.

// Injectable class used to create the Custom Service.

import { Injectable } from ‘@angular/core’;

// import HttpClient

Import { HttpClient } from “@angular/common/http”;

// use Injectable

@Injectable( )

// export the class

export class fetchService{

// create the object to HttpClient

constructor( private \_http:HttpClient) { }

// create custom function

public getProducts( ):any{

return this.\_http.get(“[http://localhost:8080/fetch”](http://localhost:8080/fetch));

};

};

**insert.service.ts:**

//import Injectable

// Injectable is the pre-defined class.

// Injectable class used to create the Custom Service.

import { Injectable } from ‘@angular/core’;

// import HttpClient

Import { HttpClient } from “@angular/common/http”;

// use Injectable

@Injectable( )

// export the class

export class insertService{

// create the object to HttpClient

constructor( private \_http:HttpClient) { }

//create the custom function.

// “saveProduct() is the Custom Function

// The argument for saveProduct(-) is obj.

//obj contain p\_id, p\_name & p\_cost

// obj should come from Component through Dependency Injection.

public saveProduct( obj:any):any{

//send the obj to the server

return this.\_http.post(“[http://localhost:8080/insert”](http://localhost:8080/insert), obj);

};

};

**update.service.ts:**

//import Injectable

// Injectable is the pre-defined class.

// Injectable class used to create the Custom Service.

import { Injectable } from ‘@angular/core’;

// import HttpClient

Import { HttpClient } from “@angular/common/http”;

// use Injectable

@Injectable( )

// export the class

export class updateService{

// create the object to HttpClient

constructor( private \_http:HttpClient) { }

//create the custom function.

// “updateProduct()” is the Custom Function

// The argument for updateProduct(-) is obj.

//obj contain p\_id, p\_name & p\_cost

// obj should come from Component through Dependency Injection.

public updateProduct( obj:any):any{

//send the obj to the server

return this.\_http.post(“[http://localhost:8080/update”](http://localhost:8080/update), obj);

};

};

**delete.service.ts:**

//import Injectable

// Injectable is the pre-defined class.

// Injectable class used to create the Custom Service.

import { Injectable } from ‘@angular/core’;

// import HttpClient

Import { HttpClient } from “@angular/common/http”;

// use Injectable

@Injectable( )

// export the class

export class deleteService{

// create the object to HttpClient

constructor( private \_http:HttpClient) { }

//create the custom function.

// “deleteProduct() is the Custom Function

// The argument for deleteProduct(-) is obj.

//obj contain p\_id, p\_name & p\_cost

// obj should come from Component through Dependency Injection.

public deleteProduct( obj:any):any{

//send the obj to the server

return this.\_http.post(“[http://localhost:8080/delete”](http://localhost:8080/delete), obj);

};

};

**Step#7: Create the Custom Components**

**Directory Structure:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

preSer11

src

app

components

mongodb.component.ts

mongodb.component.html

app.module.ts

index.html

**Step#1:** create mongodb.component.ts

( app/components/mongodb.component.ts )

import { Component } from “@angular/core”;

import { fetchService } from “../services/fetch.service”;

import { insertService } from “../services/inser.service”;

import { updateService } from “../services/update.service”;

import { deleteService } from “../services/delete.service”;

import {HttpErrorResponse } from “@angular/common/http”

//HttpErrorResonse is used to categorize the errors.

@Component( {

selector: “mongodb”,

templateUrl:”./mongodb.component.html”

})

export class mongodbComponent{

// take or declare four variable for 1. storing records 2. insert status message 3. update status message 4. delete status message

// declare variable to hold the records

private records:any;

// declare the variable to hold insert status

private insertStatus:any;

// declare the variable to hold update status

private updateStatus:any;

// declare the variable to hold delete status

private deleteStatus:any;

// declare variables to hold subscribe( ) function return types.

private fetchSubScribe:any;

private insertSubScribe:any;

private updateSubScribe:any;

private deleteSubScribe:any;

// create the instances to services.

constructor(private \_fetch:fetchService,

private \_insert:insertService,

private \_update:updateService,

private \_delete:deleteService) ){ }

// first life cycle hook

ngOnInit( ){

this.fetchSubscribe = this.\_fetch.getProducts( )

.subscribe( this.\_fetchCallBack,

this.\_errorCallBack);

};

// create the \_fetchCallBack

public \_fetchCallBack = (res):any=>{

this.records=res;

};

// create the \_errorCallBack

public \_errorCallBack= (err:HttpErrorResponse):any=>{

// categorize the error.

If(err.error instanceof Error) {

console.log(“Client Side Error!!!!”);

} else{

console.log(“Server Side Error !!!!”);

}

};

// declare variable to hold record

private record:any;

// implement the insert()

public insert(obj:any): any{

// make the service call

this.insertSubScribe = this.\_insert.saveProduct(obj)

.subscribe(this.\_insertCallBack, this.\_errorCallBack);

this.record=obj;

};

//implement \_insertCallBack

public \_insertCallBack = (res):any=>{

if(res.insert== “succsss”) {

this.records.push(this.record);

};

this.insertStatus=res;

};

**// declare variable to hold record to implement two-way data binding**

**private updateRecord:any;**

**// implement update( )**

public update(obj:any): any{

// make the service call

this.updateSubScribe = this.\_update.updateProduct(obj)

.subscribe(this.\_updateCallBack, this.\_errorCallBack);

**this.updateRecord=obj;**

};

//implement \_updateCallBack

public \_updateCallBack = (res:any):any=>{

//res contains update:success or update:fail

**// two-way data binding logic implementation**

**If(res.update==”success”{**

**for(var i:number=0; i<this.records.length; i++){**

**if(this.records[i].p\_id==this.updateRecord.p\_id){**

**this.records[i].p\_name=this.updateRecord.p-name;**

**this.records[i].p\_cost=this.updateRecord.p\_cost;**

**};**

**};**

**};**

res.updateStatus=res;

};

// implement remove()

public remove(obj:any): any{

this.deleteSubScribe = this.\_delete.deleteProduct(obj)

.subscribe(this.\_deleteCallBack, this.\_errorCallBack);

};

//implement \_deleteCallBack

public \_deleteCallBack = (res):any=>{

res.deleteStatus=res;

};

// last life cycle hook

ngOnDestroy(){

this.fetchSubScribe.unsubscribe();

this.insertSubScribe.unsubscribe();

this.updateSubScribe.unsubscribe();

this.deleteSubScribe.unsubscribe();

};

**Step#2**: create mongodb.components.html

( app/components/mongodb.components.html )

<h2> {{ records | json }} <h2>

<hr style = “height: 2px; background-color: gray”>

**<!—Inset Form -->**

<label> Product ID: </label> &emsp; &&emsp;emsp; &emsp; &emsp;

<input type =”number” [(ngModel)]=”insert\_pid”>

<br> <br>

<label> Product Name: </label> &emsp; &&emsp;emsp; &emsp; &emsp;

<input type =”text” [(ngModel)]=”insert\_pname”>

<br> <br>

<label> Product Cost: </label> &emsp; &&emsp;emsp; &emsp; &emsp;

<input type =”number” [(ngModel)]=”insert\_pcost”>

<br> <br>

<button (click) = “insert({‘p\_id’:insert\_pid,

‘p\_name:insert\_pname,

‘p\_cost:insert\_pcost})”> Insert </button>

<h2 style =”color: red”> {{ insertStatus | json}} </h2>

<hr style=”height:2px; background-color:gray”>

**<!—Update Form -->**

<br> <br>

<label> Product ID: </label> &emsp; &&emsp;emsp; &emsp; &emsp;

<input type =”number” [(ngModel)]=”update\_pid”>

<br> <br>

<label> Product Name: </label> &emsp; &&emsp;emsp; &emsp; &emsp;

<input type =”text” [(ngModel)]=”update\_pname”>

<br> <br>

<label> Product Cost: </label> &emsp; &&emsp;emsp; &emsp; &emsp;

<input type =”number” [(ngModel)]=”update\_pcost”>

<br> <br>

<button (click) = “update({‘p\_id’:update\_pid,

‘p\_name:update\_pname,

‘p\_cost: update \_pcost})”> Update </button>

<h2 style =”color: red”> {{updateStatus | json}} </h2>

<hr style=”height:2px; background-color:gray”>

**<!—Delete Form -->**

<br> <br>

<label> Product ID: </label> &emsp; &&emsp;emsp; &emsp; &emsp;

<input type =”number” [(ngModel)]=”delete\_pid”>

<br> <br>

<button (click) = “remove({‘p\_id’:delete\_pid})”> Delete </button>

<br> <br>

<h2 style =”color: red”> {{deleteStatus | json}} </h2>

<hr style=”height:2px; background-color:gray”>

**Step#3:** register in app.module.ts

Import{ BrowserModule } from ‘@angular/platform-browser’;

Import { NgModule } from ‘@angular/core’;

Import { AppComponent } from ‘./app.component’;

import { mongodbComponent} from “./components/mongodb.component”;

import { fetchService } from “./services/fetch.service”;

import { insertService} from “./services/insert.service”;

import {updateService} from “./services/update.service”;

import {deleteService} from “./services/delete.service”;

import { HttpClientModule } from “@angular/common/http”;

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

@NgModule( {

declarations : [ AppComponent, mongodbComponent],

imports : [ BrowserModule, HttpClientModule, FormsModule],

providers : [ fetchService, insertService, updateService,

deleteService],

bootstrap : [mongodbComponent]

})

export class AppModule{ }

**Step#4:** index.html

<body>

<mongodb> </mongodb>

</body>

Step#5: Run the application

preServ1> ng serve –open

Step#5: Start mongdb server.

> mongod

Step#6: Switch to Project, Server and start Node Server.

…> node server

Step#7: Switch to Project and start Node server

> ng serve - -open

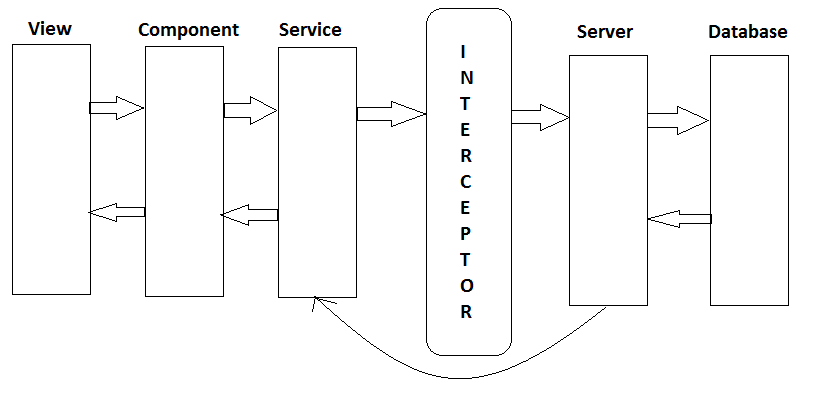
**Task: implement two-way data binding in Delete operation also.**

**Interceptors**

-- Interceptors are used to validate the http requests before hitting the Server.

-- If request is valid then only request bypassing to Server, otherwise Interceptors won’t bypass to Servers.

**Diagramatic Representation**



-- HttpInterceptor is the predefined Interface.

-- HttpInterceptor used to implement the Interceptor

-- intercept( ) is the unimplemented method in HttpInterceptor.

-- HttpRequest is the predefined class in Angular.

-- HttpRequest class used to receive the requests from Service.

-- HttpHandler is the predefined class.

-- HttpHandler class used to bypass the requests to Server.

-- HttpEvent is the predefined class in Angular.

-- HttpEvent class used to handle the Events raised by the Interceptors.

-- We will use Observables also to handle the events smoothly**.**

**URL:** [**https://www.w3schools.com/angular/customers.php**](https://www.w3schools.com/angular/customers.php)**.**

Directory Structure:

**preSer12**

**src**

**app**

**services**

**customers.service.ts**

**components**

**customers.component.ts**

**customers.component.html**

**interceptors**

**my.interceptor.ts**

**app.module.ts**

**index.html**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

my.interceptor.ts:

**// import Injectable.**

**// Injectable is used to create the Custom Service.**

**// We will implement interceptors with the help of Custom Services.**

**import { Injectable }** from “@angular/core”;

// import HttpInterceptor

// HttpInterceptor used to implement the Interceptors.

// Import HttpRequest

// HttpRequest is used to take the requests from service.

// import HttpHandler

// HttpHandler is used to bypass the requests to Servers.

// Import HttpEvent.

// HttpEvent is used to handle the Events Raised by Interceptors

import { HttpIntercetor, HttpRequest, HttpHandler, HttpEvent } from

“@angular/common/http”;

//Import Observable

@Injectable()

// export the class

export class myInterceptor implements HttpInterceptor{

intercept(req:HttpRequest<any>, handler:HttpHandler):Observable<httpEvent<any>>{

console.log(“---- In Interceptor---);

var requ1=req.clone({

setHeaders:{

“authorization”:”adkfjskfjskdfs”

}

} );

return hadler.handle(req10;

};

}

**customers.service.ts:**

import { Injectable} from “@angular core”;

import { HttpClient } from “@angular/common/http”;

@Injectable()

export class customersService{

constructor(private \_http:HttpClient){}

public getCustomers()”:any{

return this.\_http.get(“<https://www.w3cschools.com/angular/customers.php>”};

};

};

**customers.component.ts:**

import {Component} from “@angular/core”;

import { customersService} from “../service/customers.service”;

import { HttpErrorResponse } from “@angular/common/http”;

@Component ({

selector: “customers”,

templateUrl:”./customers.component.html”

})

export class customersComponent{

private result:any;

privatecustomersSubscribe:any;

constructor(private \_service:customersService){}

ngOnInit() {

this.customersSubscribe = this.\_servce.getCustomers()

.subscribe(this.\_successCallBack,

this.\_errorCallBack) ;

};

}

Public \_successCallBack = (res):any=>{

this.result=res;

};

public \_errorCallBack=(err:HttpErrorResponse):any=>{

If(err.error instanceof Error){

console.log(“Client side Error…”);

}else{

console.log(“Server side Eror….”);

}

};

ngOnDestroy(){

this.customers.subscribe.unsubscribe();

};

};

**customers.component.html:**

<table border=”1”

cellpadding=”10px”

cellspacing=”10px”

align=”center”

style=”font-size:30px;text-align:center”>

<thead style=”background-color: gray”>

<tr>

<th> Name</th>

<th> City</ th>

<th> Country</ th>

</tr>

</thead>

<tbody>

<tr \*ngFor = “let x of result.records”>

//\*ngFor is a structural type pre-defined directive.

<td>{{x.Name}}</td>

<td> {{x.City}}</td>

<td> {{x.Country}</td>

</tr>

</tbody>

</table>

**app.module.ts**

import { BrowserModule } “@angular/platform-browser”;

import {NgModule } from “@angular/core”;

import {AppComponent” from “./app.component”;

import {customersService} from “./services/customers.service”;

import {customersComponent} from “./components/customers.component”;

import{myInterceptor} from “./interceptors/my.interceptor”;

import { HttpClientModule, HTTP\_INTERCEPTORS } from “@angular/http”;

@NgModule( {

declarations : [ AppComponent, customersComponent],

imports : [ BrowserModule, HttpClientModule],

providers : [ customersService, {

// we need to register interceptor by using json object.

provide : HTTP\_INTERCEPTORS,

useClass : myInterceptor,

multi : true

} ],

bootstrap : [customersComponent]

})

export class AppModule{ }

**Note:**

-- HTTP\_INTERCEPTORS is the constant in angular used to register the interceptors

-- “useClass” as JSON key used to register the interceptor by “class name”.

-- “multi” is the key, used to enable multiple interceptors in application.

Bydefault “multi” key is disabled.

**Step#4:** index.html

<body>

<customers> </customers>

</body>

Run the application.

preserv12> ng serve - -open

If we send the req1, it won’t execute in server, because req1 contains corrupted data.

We need to send req only.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PIPES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- Pipes are used to format the data according to application requirement.

-- We will apply pipes by using “ | “ symbol.

-- We have two types of pipes.

1) Predefined pipes.

2) Custom pipes.

The pipes given by angular framework called as predefined pipes.

The pipes developed by us based on application requirement called as custom pipes.

**Predefined Pipes:**

1) **titlecase:**

--“ titlecase” pipe used to make the first letter of each word as capital.

**Example:**

private var\_one:string = “naresh it”;

{{ var\_one }} // naresh it

{{ var\_one | titlecase }} // Naresh It

2) **currency:**

-- “currency” pipe used to apply the currencies symbol to numerical values.

**Example:**

private var\_two:number= 100;

{{ var\_two}} // 100

{{ var\_two | currency }} // $100.00

{{ var\_two | currency:”INR” }} // R100.00

{{ var\_two | currency:”GBP”}} // ..100.00

{{ var\_two | currency: “EUR” }} // … 100.00

- - -

- - -

3) **number:**

-- “number” pipe used to manipulate the numerical values.

**Example:**

private var\_three:number = 100.12345;

{{ var\_three | number:”3.1-2”}} // 100.12 (3 indicates no.of digits, 1 indicates minimum no.of digits, 2 indicates maximum no.of digits.

{{ var\_three | number:”4.2-4”}} // 0100.1234

4) **percent:**

-- “percent” pipe used to convert the fractions to equivalent percentages.

**Example:**

private var\_four:number = 0.9;

{{ var\_four }} // 0.9

{{ var\_four | percent }} // 90%

5) **json:**

-- “json” pipe used to convert the JSON objects to JSON Strings

**Example:**

private var\_five: any = { “p\_id” : 111, “p\_name”: “p\_name”, “p\_cost”:10000”};

{{ var\_five }} // [ Object object ]

{{ var\_five | json }} // { “p\_id” : 111, “p\_name”: “p\_name”, “p\_cost”:10000”};

6) **uppercase:**

-- “uppercase” pipe used to convert lowercase characters to uppercase characters.

**Example:**

private var\_six:string = “naresh it”;

{[ var\_six }} // naresh it

{{ var\_six | uppercase }} // NARESH IT

7)  **lowercase:**

--“lowercase” pipe used to convert uppercase characters to lowercase characters.

**Example:**

private var\_seven:string = “NARESH IT”;

{[ var\_s seven }} // NARESH IT

{{ var\_six | lowercase }} // naresh it

8) **date:**

-- “date” pipe used to manipulate the date according to application requirement.

**Example**

private var\_eight:Date= new Date();

{[ var\_eight }}

{{ var\_eight | date:’dd-MM-yy’ }}

{{ var\_eight | date:’dd-MMM-yyyy’ }}

{{ var\_eight | date:’shortDate’ }}

{{ var\_eight | date:’mediumDate’ }}

{{ var\_eight | date:’fulldate’ }}

9) **slice:**

-- This pipe used to manipulate the numerical arrays.

**Example:**

private var\_nine:Array<number> = [10, 20, 30, 40, 50];

{{ var\_nine }} // 10 20 30 40 50

{{ var\_nine | slice:1:3}} // 20 30

{{ var\_nine | slice:1 }} // 20 30 40 50

{{ var\_nine | slice: 1:-1 }} // 20 30 40

**04/12/18**

**Example:** pipesDemo

C:\...desktop\pipesDemo> ng new pipesDemo

**app.component.ts:**

import {Component } from “@angular/core”;

@Component({

selector: ‘app-root’,

templateUrl: ‘’./app.component.html’,

styleUrls: [‘.app.component.css’]

})

export class AppComponent{

private var\_one :string = “naresh it”;

private var\_two :string = “NARESH IT”;

private var\_three :string = “naresh it”;

private var-four :number = 100.12345;

private var\_five:number = 0.9;

private var\_six :number = 100;

private var\_seven : { “p\_id”:111, “p\_name”:”p\_one”, “p\_cost”:10000};

private var\_eight : Date= new Date();

privater var\_nine :Array<number>=[10, 20,30,40,50];

private var\_ten:any;

public getPromise():any{

return new Promise((resolve, reject)=> {

setTimeout( ( )=>{

resolve(new Date().toLocaleTimeString( ))

},1000);

});

};

constructor (){

this.var\_ten = this.getPromise();

};

}

**app.comonent.html:**

<h1 style=”color:teal”> {{ var\_ten | async}} </h1>

<h1 style=”color:skyblue”> {{var\_nine | slice:1}}</h1>

<h1 style=”color:skyblue”> {{var\_nine | slice:1:-1}}</h1>

<h1 style=”color:skyblue”> {{var\_nine | slice:1:3}}</h1>

<h1 style=”color:skyblue”> {{var\_nine }}</h1>

<h1 style=”color:seagreen”> {{var\_eight | date:’shortTime’}}</h1>

<h1 style=”color: seagreen”> {{var\_eight | date:’fullDate’}}</h1>

<h1 style=”color: seagreen”> {{var\_eight | date:’mediumDate’}}</h1>

<h1 style=”color: seagreen”> {{var\_eight | date:’dd/MMM/yy’}}</h1>

<h1 style=”color: seagreen”> {{var\_eight | date:’dd-MMM-yyyy’}}</h1>

<h1 style=”color: seagreen”> {{var\_eight | date:’dd-MM-yy’}}</h1>

<h1 style=”color: seagreen”> {{var\_eight}}</h1>

<h1 style=”color:purple”> {{var\_seven |json}}</h1>

<h1 style=”color:purple”> {{var\_seven }}</h1>

<h1 style=”color:peru”> {{var\_six |currency:”EUR”}}</h1>

<h1 style=”color:peru”> {{var\_six |currency:”GBP”}}</h1>

<h1 style=”color:peru”> {{var\_six |currency:”INR”}}</h1>

<h1 style=”color:peru”> {{var\_six |currency}}</h1>

<h1 style=”color:palevioletred”> {{var\_five |percent}}</h1>

<h1 style=”color:brown”> {{var\_four |number:”4.1-1”}}</h1>

<h1 style=”color:brown”> {{var\_four |number:”3.1-2”}}</h1>

<h1 style=”color:brown”> {{var\_four |number}}</h1>

<h1 style=”color:blue”> {{var\_three |titlecase}}</h1>

<h1 style=”color:green”> {{var\_two |lowercase}}</h1>

<h1 style=”color:red”> {{var\_one |uppercase}}</h1>

**Output:**

**\*\*\*\*\*\*\* CUSTOM PIPES \*\*\*\*\*\*\***

-- Creating our own pipe based on application requirement called as

-- Pipe is the predefined class in angular.

-- Pipe class used to assign logical name to Custom Pipe.

-- PipeTransform is the predefined interface in Angular.

-- PipeTransform used to implement the Custom Pipe.

-- transform( ) is the unimplemented method in PipeTransform interface.

Example:

**App.reverse.pipe.ts:**

// import Pipe

// Pipe is the predefined class.

// Pipe class used t assign the logical name to custom pipe.

// import PipeTransform.

// PipeTransform is the interface.

// PipeTransform interface used to implement the Custom Pipe.

import {Pipe, PipeTransform } from “@angularcore”;

@Pipe({

name:”reverse”

})

export class reversePipe implements PipeTransform{

transform(arg1){

var data:string=” “;

for ( var i:number=0; i<arg1.length; i++ ) {

data = arg1[i] + data;

};

return data;

};

};

**app.component.html**

<h1> {{“Hello” | reverse}}</h1>

**app.module.ts:**

import BrowserModule

import NgModule

import AppComponent

import reversePipe from “.reverse.pipe”;

@NgModule({

Declarations: AppComponent, reversePipe,

Imports: BrowserModule,

Providers: [ ],

Bootstrap;AppComponent

})

export class AppModule { }

**Communication between Components:**

-- Simple TypeScript class called as Component.

-- Component acting as interface between View and Service.

-- Because of Components code Reusability is high in Angular7.

-- Angular7 applications are Component based applications.

-- We can create more than one Component.

-- As a Angular Developer we can provide communication between Components.

-- In Angular we can provide communication in four ways.

1) @Input( ) (Parent to Child)

2) @Output( ) (Child to Parent)

3) @ViewChild( ) (to get reference of one Component in another Component)

4) @ViewChildren( )

**05/12/18**

**@Input**

-- “@Input( )” is the Decorator.

-- “@Input( )” Decorator used to share the data from parentComponents to childComponents.

-- Input is the Predefined class.

-- Input class available in @angular/core package.

**Steps to Transfer the data from parentComponent to childComponent:**

**Step#1:** Create the childComponent with @Input( ) properties.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* **Directory Structure:**

\* InputAndOutputEx

\* src

\* app

\* components

\* child.component.ts

\* child.component.html

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**app/components/child.component.ts:**

import { Component, Input } from “@angular/core”;

@Component( {

selector : “child”,

templateUrl : “./child.component.html”

})

export class childComponent(

@Input( ) p\_id;

@Input( ) p\_id;

@Input( ) p\_id;

**Step2:** Create the parentComponent with application data.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* **Directory Structure:**

\* InputAndOutputEx

\* src

\* app

\* components

\* parent.component.ts

\* parent.component.html

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**app/components/parent.component.ts**

**Step#3:** Call the childComponent selector in parent Component template.

**Step#4:** Bootstrap the parentComponent

**Step#5:** Add<parent> tag (selector) in index.html

**Step#6:** Run the application

ng serve - -open

**Step#7:** Open browser and enter url: <http://localhost:4200>.

**2) @Output( ):**

-- “@Output( )” is the predefined Decorator.

-- @Output( ) Decorator used to share data from childComponent to parentComponent.

-- Output is the predefined class.

-- Outpu class available in @angular/core package.

**Steps to transfer the Data from childComponent to parentComponent:**

**Step#1:** Declare the @Output( ) event in childComponent.

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

**Step#2:** Fire the event, whenever we click the button, with help of emit( ) function.

**Step#3:** map the even in parent component template.

**Complete Code:**

**app/components/child.component.ts:**

// import Component.

//Component used to create the building block of angular application.

// import Input.

// Input is used to receive the data from parentComponent.

//Import Output.

// Output is used to share / send the data to parentComponent.

// Import EventEmitter.

// EventEmitter is the predefined class.

// EventEmitter class helps to Output Class to share the Data from

childComponent to parentComponent.

import { Component, Input, Output, EventEmitter } from “@angular/core”;

@Component( {

selector : “child”,

templateUrl : “./child.component.html”

} )

export class childComponent(

/\* declare properties.

p\_id;

p\_name;

p\_cost;

These properties should come from parentComponent.

\*/

@Input( ) p\_id;

@Input( ) p\_name;

@Input( ) p\_cost;

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

public clickMe( ):any{

// emi() is the predefined function used to fire the event.

this.sendData.emit(this.p\_id + this.p\_name + p\_cost);

};

};

**child.component.html:**

<ht> Product Id <span style=”color:red”> {{p\_id}} </span></h2>

<ht> Product Id <span style=”color:red”> {{p\_name}} </span></h2>

<ht> Product Id <span style=”color:red”> {{p\_cost}} </span></h2>

<button (click=”clickMe()”> <b>Send</b> </button>

<hr style=”background-color:gray;height:2px;”>

**parent.component.ts:**

import Component from angular/core

@Component({

Selector parent

templateUrl “./parent.component.html

})

Export class parentComponent{

Private my\_array:Array<any>= [

{“p\_id”:111,”p\_name”:”p\_one”, “p\_cost”:10000},

{“p\_id”:222,”p\_name”:”p\_one”, “p\_cost”:20000},

{“p\_id”:333,”p\_name”:”p\_one”, “p\_cost”:30000},

{“p\_id”:444,”p\_name”:”p\_one”, “p\_cost”:40000},

{“p\_id”:555,”p\_name”:”p\_one”, “p\_cost”:50000},

];

public my\_fun(data0:any{

alert(data);

};

};

**parent.component.html:**

// we will use [ ] for Input properties.

// we will use ( ) for Output properties.

<child

[ p\_id] =”x.p\_id”

[ p\_name] =”x.p\_name”

[ p\_cost] =”x.p\_cost”

(sendData)=”my\_fun($event)”

\*ngFor=”let x of my\_array”>

</child>

**06/12/18**

**3)** **@ViewChild:**

-- “@ViewChild( )” is the predefined Decorator.

-- @ViewChild( ) Decorator used to get the reference of one Component to another Component.

-- ViewChild is the predefined class.

-- ViewChild class is available in @angular/core package.

**Rough:**

--ViewChild has the capability to manipulate the DOM elements also ( like <input> element, etc…) with help of ElementRef class.

**Example:**

<input type =”text” #fname>

<input type =”text” #lname>

Here ViewChild can manipulate #fname & #lname.

**Disadvantage of ViewChild:**

-- ViewChild effect goes to first occurrence only.

**Example:** <child> </child>

<child> </child>

<child> </child>

Here ViewChild effects goes to first occurrence only i.e. first line only.

**Note:** we can modify parentComponent properties with childComponent properties with the help of @ViewChild Decorator.

**Rough endSteps to implement the ViewChild Example:**

**Step#1:** Create the childComponent with public properties.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* **Directory Structure:** \*

\* \*

\* ViewChildEx \*

\* src \*

\* app \*

\* components \*

\* child.component.ts \*

\* child.component.html \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**src/app/components/child.component.ts:**

import {Component } from “@angular/core”;

@Component({

selector : “child”;

templateUrl : “./child.comonent.html”

})

export class childComponent{

public var\_one: string;

public var\_two: string;

constructor(){

this.var\_one=”Message from childComponent”;

this.var\_two=”Marquee Test from Child Component…”

};

};

**src/app.components/child.component.html:**

<h1 style=”color:red”> {{var\_one}}</h1>

<h2 style=”color:green”><marquee>{{var\_two}} </marquee> </h2>

Step#2: Get the Reference of childComponent in parent Component by using @ViewChild() Decorator.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* **Directory Structure:** \*

\* \*

\* ViewChildEx \*

\* src \*

\* app \*

\* components \*

\* parent.component.ts \*

\* parent.component.html \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**src/app/components/parent.component.ts:**

import {Component } from “@angular/core”;

@Component( {

selector : “parent”,

templateUrl: “./parent.component.html”

});

export class parentComponent{

@ViewChild(childComponent)

private\_child:childComponent;

public clickMe():any{

this.\_child.var\_one=”Message from Parent Component…”;

this.\_child.var\_two=”Marquee from parent Component…”;

};

};

**Step#3:** Call the childComponent selector in parentCoponent Template

**src/app/components/parent.component.html:**

<child> </child>

<button (click)=”clickMe( )”><b> ClickMe</b> </button>

**Step4:** Bootstrap the parentComponent

Open app.module.ts and add below lines of code.

Import {parentComponent} from “./components/parent.component”;

Import {childComponent} from “./components/child.component”;

…..

**Complete Example:**

**src/app/components/child.component.ts:**

import {Component } from “@angular/core”;

@Component({

selector : “child”;

templateUrl : “./child.comonent.html”

})

export class childComponent{

public var\_one: string;

public var\_two: string;

constructor(){

this.var\_one=”Message from childComponent”;

this.var\_two=”Marquee Test from Child Component…”;

};

};

**src/app.components/child.component.html:**

<h1 style=”color:red”> {{var\_one}}</h1>

<h2 style=”color:green”><marquee>{{var\_two}} </marquee> </h2>

**src/app/components/parent.component.ts:**

import {Component, ViewChild, ElementRef } from “@angular/core”;

import {childComponent} from “./child.component”;

@Component( {

selector : “parent”,

templateUrl: “./parent.component.html”

});

export class parentComponent{

@ViewChild(childComponent)

private\_child:childComponent;

@ViewChild(“fname”)

private \_fname:ElementRef;

@ViewChild(“lname”)

private \_lname:ElementRef;

public clickMe():any{

this.\_child.var\_one=”Message from Parent Component…”;

this.\_child.var\_two=”Marquee from parent Component…”;

this.\_fname.nativeElement.style.backgroundColor=”green”;

this.\_fname.nativeElement.style.color=”red”;

this.\_lname.nativeElement.style.backgroundColor=”red”;

this.\_lname.nativeElement.style.color=”green”;

};

};

// ElementRef is the predefined class.

// ElementRef class available in @angular/core package.

// ElementRef class used to manipulate the DOM elements

**src/app/components/parent.component.html:**

<child> </child>

<child> </child>

<child> </child>

<input type = “text” #fname> <br>

<input type = “text” #lname> <br>

<button (click)=”clickMe( )”><b> ClickMe</b> </button>

-- if we have more than one occurrence, only first occurrence will be effected by ViewChild.

-- In order to overcome the above limitation we will use ViewChildren.

-- Here “ # ” is used to assign the logical name to DOM elements.

**app.module.ts (configuration file)**

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

import { NgModule } from “@angular/core”;

import { AppComponent } from “./app.component”;

Import { parentComponent } from “./components/parent.component”;

Import { childComponent } from “./components/child.component”;

@NgModule({

declarations : [AppComponent, parentComponent, childComponent],

imports: [BrowserModule],

providers: [ ] ,

bootstrap: [ parentComponent ]

})

export class AppModule{ }

**index.html:**

<parent> </parent>

**4) @ViewChildren:**

Example:

**src/app/components/parent.component.ts:**

import {Component, ViewChild, ViewChildren, ElementRef } from “@angular/core”;

import {childComponent} from “./child.component”;

@Component( {

selector : “parent”,

templateUrl: “./parent.component.html”

});

export class parentComponent{

@ViewChildren(childComponent)

private \_child.QueryList<childComponent> = new QueryList();

private my\_array:Array<any>=[ ];

ngAfterViewInit() {

this.my\_array=this.\_child.toArray();

};

public clickMe():any{

for(var i:number=0; I<this.my\_array.length; i++){

this.my\_array[i]var\_one=”Hello”;

this.my\_array[i]var\_two=”Hi”;

};

};

Remaining code is same from the previous example.

**10/12/18**

-- “@ViewChildren( )” Decorator used to get the reference of one Component to another Component.

-- ViewChildren is the predefined class.

-- ViewChildren class is available in @angular/core package.

**Steps to implement the ViewChildren Decorator Example:**

Create the Project using below command

> ng new ViewChildrenEx

Step#1: Create the childComponent with two public properties.

**Directory Structure:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ViewChildrenEx

src

app

components

child.component.ts

child.component.html

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**C:\...\Demo\ViewChildrenEx> src/app/components/child.component.ts:**

import {Component} from “@angular/core”;

@Component( {

selector : “child”,

templateUrl: “./child.component.html”

});

export class childComponent{

public var\_one:string;

public var\_two:string;

constructor(){

this.var\_one=”Message from Child Component….”;

this.var\_two=”Marque Text from Child Component….”;

};

};

**child.component.html:**

<h1 style=”color:red”> {{var-one}} <h1>

<h1 style=”color:green> <marquee> {{var-two}} </marquee> <h1>

**Step#2:** Call the childComponent selector in parentComponent template.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ViewChildrenEx

src

app

components

parent.component.ts

parent.component.html

app.module.ts

index.html

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**C:\...\Demo\ViewChildrenEx> src/app/components/parent.component.html:**

<child> </child>

<child> </child>

<child> </child>

<button (click)=”clickMe()”> ClickMe </button>

**Step#3:** Get the Reference of childComponent to parentComponent by using @ViewChildren( ) decorator.

import {Component, QueryList, ViewChildren } from “@angular/core”;

import {childComponent} from “./child.component”;

@Component( {

selector : “parent”,

templateUrl: “./parent.component.html”

});

export class parentComponent{

@ViewChildren(childComponent)

// get the references of childComponent using QueryList

private \_child.QueryList<childComponent> = new QueryList();

// here QueryList will get all the occurrences of childComponent and will store the same in Map Object. This Map object will be stored in \_child.

private my\_array:Array<any>=[ ];

ngAfterViewInit() {

this.my\_array=this.\_child.toArray();

};

public clickMe():any{

for(var i:number=0; I<this.my\_array.length; i++){

// modify var\_one

this.my\_array[i]var\_one=”Message from Parent Component”;

// modify var\_two

this.my\_array[i]var\_two=”Marque Text from Parent Component”;

};

};

};

**Step#4:** Bootstrap the parentComponent

**app.module.ts (configuration file)**

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

import { NgModule } from “@angular/core”;

import { AppComponent } from “./app.component”;

Import { parentComponent } from “./components/parent.component”;

Import { childComponent } from “./components/child.component”;

@NgModule( {

declarations : [ AppComponent, parentComponent, childComponent ],

imports : [ BrowserModule ],

providers : [ ] ,

bootstrap : [ parentComponent ]

} )

export class AppModule{ }

**Step#5:** Add <parent> selector in landing template.

**index.html:**

<body>

<parent> </parent>

</body>

**Step#6:** Run the application using below command.

View-> Terminal-> ViewChildrenEx> ng serve - -open

**Step#7:** Check the output on Browser using below url.

<http://localhost:4200>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\* Directives \*\*\*\*\*\*\*\*\*\*\*\*\*\***

-- Directives enhance the view capabilities.

-- We have two types of Directives.

1) Predefined Directives.

2) Custom Directives.

-- The Directives given by Angular Framework called as Predefined Directives.

-- The Directives developed by us based on application requirement called as Custom Directives.

**Two Way Data Binding:**

-- Automatic Data Synchronization between Model and View with the supervision of Component called as Two Way Data Binding.

i.e. changes in Model will reflect in View and also changes in View will reflect in Model with supervision of Component.

-- All the TypeScript variables in Component behaves like Model.

-- We can use HTML, HTML5, Python & PERL as a View in Angular.

**Directory Structure:**

TwoWayDataBindingEx

src

app

components

click.component.ts

click.component.html

app.module.ts

index.html

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**click.component.html:**

<label> Uname.</label>&emsp; &emsp; &emsp; &emsp;

<input type=”text” [(ngModel)]=”obj.uname”>

<br> <br>

<label> Upwd.</label>&emsp; &emsp; &emsp; &emsp;

<input type=”password” [(ngModel)]=”obj.upwd”>

<br> <br>

<button (click)=”clickMe()”> Login</button>

**click.component.ts:**

import {Component } from “@angular/core”;

@Component( {

selector : “click”,

templateUrl: “./click.component.html”

});

export class clickComponent{

private obj:any = { “uname” : “ ”, “upwd”: “ “};

public clickMe():any{

if(this.obj.uname=”admin” && this.obj.upwd=”admin”) {

alert(“Login Success…”);

} else {

alert (“Login Fail…”);

};

};

};

**app.module.ts (configuration file)**

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

import { NgModule } from “@angular/core”;

import { AppComponent } from “./app.component”;

Import { clickComponent } from “./components/click.component”;

Import { FormModule } from “@angular/forms”;

@NgModule( {

declarations : [ AppComponent, clickComponent ],

imports : [ BrowserModule, FormsModule ],

providers : [ ] ,

bootstrap : [ clickComponent ]

} )

export class AppModule{ }

**Step#5:** Add <click> selector in landing template.

**index.html:**

<body>

<click> </click>

</body>

**Step#6:** Run the application using below command.

View-> Terminal-> TwoWayDataBindingEx> ng serve - -open

**Step#7:** Check the output on Browser using below url.

<http://localhost:4200>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**“ # “ Operator:**

-- “ # “ Operator is used to assign the logical name to DOM elements.

**Example:** consider previous example with below modifications (but it is not a two-way data binding example)

**click.component.html:**

<label> Uname.</label>&emsp; &emsp; &emsp; &emsp;

<input type=”text” #uname”>

<br> <br>

<label> Upwd.</label>&emsp; &emsp; &emsp; &emsp;

<input type=”password” #upwd >

<br> <br>

<button (click)=”clickMe(uname.value, upwd.value)”> Login</button>

**click.component.ts:**

import {Component } from “@angular/core”;

@Component( {

selector : “click”,

templateUrl: “./click.component.html”

});

export class clickComponent{

private obj:any = { “uname” : “ ”, “upwd”: “ “};

public clickMe(arg1:any, arg2:any):any{

if(arg1==”admin” && arg2==”admin”) {

alert(“Login Success…”);

} else {

alert (“Login Fail…”);

};

};

};

Why Directives in Angular?

How many types of Predefined Directives?

Explain Two Way Data Binding with example?

What is Structural Directive?

Explain Custom Directives?

**11/12/18**

**[ngSwitch]:**

-- This directive used to write the switch cases in angular applications.

Syntax:

[ngSwitch]=”expression”

\*ngSwitchCase=”value1”

// Business Logic

\*ngSwitchCase=”value2”

// Business Logic

* - - -
* - - -

\*ngSwitchDefault

// Business Logic

**\*ngIf**:

-- This directive used to write the conditions.

**Syntax-1:**

<div \*ngIf=”condition; else elseBlock”> …. </div>

<ng-template #elseBlock>……</ng-template>

**Syntax-2:**

<div \*ngIf=”condition; then thenBlock else elseBlock”> …. </div>

<ng-template #thenBlock>……</ng-template>

<ng-template #elseBlock>……</ng-template>

**Example:**

app.component.html:

<input type =”number” [(ngModel)]=”my\_number”><br> <br>

<div [ngSwitch]=”my\_number”>

<div “ngSwitchCase=”1” class=”btn btn-success”> {{ my\_number }}</div>

<div “ngSwitchCase=”2” class=”btn btn-danger”> {{ my\_number }}</div>

<div “ngSwitchCase=”3” class=”btn btn-warning”> {{ my\_number }}</div>

<div “ngSwitchCase=”4” class=”btn btn-default”> {{ my\_number }}</div>

<div “ngSwitchDefault=”1” class=”btn btn-button”> {{ my\_number }}</div>

</div>

Note: btn btn-success, btn btn-danger, btn btn-warning, btn btn-default, btn btn-button - all are predefined classes in bootstrap.

**app.component.ts**:

import { Component } from “@angular/core”;

@Component({

Selector: ‘app-root’,

templateUrl: ‘./app.component.html’,

styleUrs:’./app.component.css’]

})

export class AppComponent{

my\_number:number=0;

}

**Index.html:**

<head>

<link rel=”stylesheet” href=<https://maxcdn.bootstracdn.com/bootstrap/3.3.7/css/bootstra.min.css>>

<link rel=”icon” type=”image/x-icon” href=”favicon.ico”>

</head>

**app.module.ts:**

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

import { NgModule } from “@angular/core”;

import { AppComponent } from “./app.component”;

Import { FormModule } from “@angular/forms”;

@NgModule( {

declarations : [ AppComponent],

imports : [ BrowserModule, FormsModule ],

providers : [ ] ,

bootstrap : [ ]

} )

export class AppModule{ }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**app.component.html**:

<div \*ngIf=”my\_string==’Naresh’; else elseBlock”> Hello…! </div>

<ng-template #elseBlock> Hi….</ng-template>

**app.component.ts:**

import { Component } from “@angular/core”;

@Component({

Selector: ‘app-root’,

templateUrl: ‘./app.component.html’,

styleUrs:’./app.component.css’]

})

my\_number:number=0;

my\_string:string=”Naresh”;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**app.component.html:**

<div \*ngIf=”my\_string==’Naresh’; then thenBlock else elseBlock”> Hello</div>

<ng-template #thenBlock> <b> Technologies</b> </ng-template>

<ng-template #elseBlock> <b> Angular7</b> </ng-template>

-- If condition satisfies automatically then block will be executed.

-- If condition fails automatically else block will be executed.

-- In both the cases the presentation logic present in “if block” won’t executes.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**app.component.html:**

<input type = “radio” name=”gender” (click) = “my\_fun(true)”><b> Male</b>

<input type = “radio” name=”gender” (click) = “my\_fun(true)”><b> Female</b>

<br> <br>

<div \*ngIf=”status;then thenBlock else elseBock”> </div>

<ng-template #thenBlock> <b> Male </b> </ng-template>

<ng-template #elseBlock> <b> FeMale </b> </ng-template>

**app.component.ts**:

import { Component } from “@angular/core”;

@Component({

Selector: ‘app-root’,

templateUrl: ‘./app.component.html’,

styleUrs:[’./app.component.css’]

})

export class AppComponent{

my\_number:number=0;

my\_string:string=”Naresh”;

private status:boolean;

public my\_fun(isBoolean:boolean):any{

this.status=isBoolean:

};

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**[ngStyle]:**

This Directive is used to apply the CSS Properties to DOM Elements.

**Example**

<div [ngStyle] = “{‘background-color’: ‘green’, ‘color’ : ’red’}> Hello</div>

**Conditional Code:**

<div [ngStyle] = “{‘background-color’: ‘green’, ‘color’ : ’red’}> Hello</div>

<div [ngStyle] = “{‘color’: my\_string===’Naresh’ ? ‘red’ : ‘pink’ } ” > Hello</div>

<b> Welcome to ngStyle with Condition </b>

</div>

**app.component.html:**

<div [ngStyle] = “{‘color’: my\_string===’Naresh’ ? ‘red’ : ‘pink’ } ” > Hello</div>

<b> Welcome to ngStyle with Condition </b>

</div>

<div [ngStyle] =”{‘color’:getColor()}”>

Hello This is NgStyle with Custom Function

</div>

**app.component.ts**:

import { Component } from “@angular/core”;

@Component({

Selector: ‘app-root’,

templateUrl: ‘./app.component.html’,

styleUrs:[’./app.component.css’]

})

export class AppComponent{

my\_number:number=0;

my\_string:string=”Naresh”;

private status:boolean;

public my\_fun(isBoolean:boolean):any{

this.status=isBoolean:

// custom function call

public getColor():string{

return “blue”;

};

};

}

From the above examples discussed in today’s class, we have discussed below three possibilities:

Direct assignment

Condition assignment

Function call assignment

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**[ngClass]** :

-- This directive used to apply the bootstrap classes to DOM Elements directly.

**app.component.ts**:

import { Component } from “@angular/core”;

@Component({

Selector: ‘app-root’,

templateUrl: ‘./app.component.html’,

styleUrs:[’./app.component.css’]

})

export class AppComponent{

my\_number:number=0;

my\_string:string=”Naresh”;

private status:boolean;

public my\_fun(isBoolean:boolean):any{

this.status=isBoolean:

// custom function call

public getColor():string{

return “blue”;

};

private my\_array:Array<any>=[

{“id” : 111, “name” : “AngularJS”},

{“id” : 222, “name” : “Angular7”},

{“id” : 333, “name” : “ReactJS”},

{“id” : 444, “name” : “NodeJS”},

{“id” : 555, “name” : “MongoDB”},

];

}

**app.component.html:**

<ul \*ngFor=”let x of my\_array”>

<li [ngClass]=”{ ‘text-success’ : x.id == 111,

‘text-danger’ : x.id == 222},

‘text-primary’ : x.id == 333},

‘text-warning’ : x.id == 444},

‘text-default’ : x.id == 555}

} “ >

{{x.id}} ….. {{x.name }}

</li>

</ul>

**15/12/18**

**Single Page Applications**

-- Loading one template into another template without refreshing the whole webpage called as Single Page Application.

-- Loading one template into another template in Single Page Application called as “Routing”.

-- “Routes” is the predefined Array.

-- “Routes” array used to implement the Routing in Single Page Application.

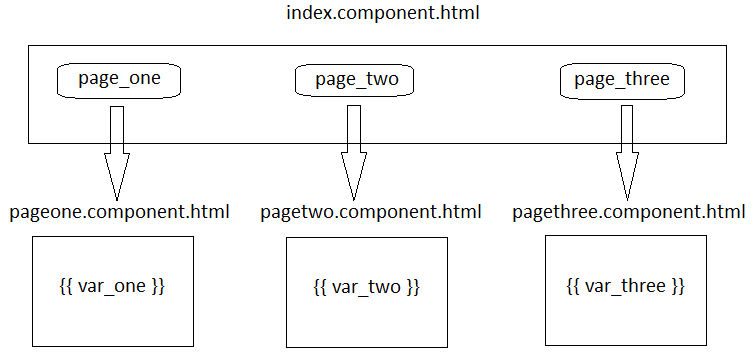
-- “RouterModule” is the predefined Module.

-- “RouterModule” used to load the “Routings” into Memory.

-- “Router” is the predefined class.

-- “Router” class used to navigate from one template to another template in Single Page Application.

-- “Routes”, “RouterModule” and “Router” class available in @angular/router” package”



[ binding means printing. ]

In the above application we have the following components.

1. IndexComponent.
2. PageoneComponent.
3. PagetwoComponent.
4. PagethreeComponent

-- Where IndexComponent is the source component in Single Page Application.

-- Where as “PageoneComponent”, “PagetwoComponent” & “PagethreeComponent” are Target components in Single Page Application.

**Steps to implement the above Application:**

**Step#1:** Generate the components by using the following command.

> ng g c components/index - -spec=false -is

> ng g c components/pageone - -spec=false -is

> ng g c components/pagetwo - -spec=false -is

> ng g c components/pagethree - -spec=false -is

-- “ g “ stands for generate.

-- “ c “ stands for Component.

-- “- -spec=false” means avoiding the creation of unit testing file.

-- “-is ” stands for inline styles.

-- “ -is “ meaning is avoiding the external style sheets.

**Step#2:** Implement the Business Logic in Target Components:

**SPADemo1/src/app/components/pageone/pageone.component.html:**

<h1 style=”color:red’> {{ var\_one}} </h1>

**SPADemo1/src/app/components/pageone/pageone.component.ts:**

private var\_one:string;

constructor(){

this.var\_one=”I am from Page One !!! “;

}

**SPADemo1/src/app/components/pageone/pagetwo.component.html:**

<h1 style=”color:green’> {{ var\_two}} </h1>

**SPADemo1/src/app/components/pageone/pagetwo.component.ts:**

private var\_two:string;

constructor(){

this.var\_two=”I am from Page Two !!! “;

}

**SPADemo1/src/app/components/pageone/pagethree.component.html:**

<h1 style=”color:blue’> {{ var\_three}} </h1>

**SPADemo1/src/app/components/pageone/pagethree.component.ts:**

private var\_three:string;

constructor(){

this.var\_three=”I am from Page Three !!! “;

}

**Step#3:** Implement the presentation logic in Source Component Template (i.e. index.component.html).

**SPADemo1/src/app/components/pageone/index.component.html:**

<!—[routerLink] is the predefined directive.

[routerLink] is used to load the page without refreshing.

<router-outlet> is the predefined directive.

<router-outlet> behaves like magnet to hold the target pages.

-->

<a [routerLink]=”[ ‘/ ’ ] “ > <b> Page\_one</b> </a> &emsp; &emsp; &emsp; &emsp;

<a [routerLink]=”[ ‘/page\_two ’ ] “ > <b> Page\_two</b> </a> &emsp; &emsp; &emsp; <button (clickMe) = “clickMe()”> <b> Page\_Three </button>

<br> <br>

<router-outlet></router-outlet>

**SPADemo1/src/app/components/pageone/index.component.ts:**

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

import { Router } from “@angular/router”;

@Component( {

selector : ‘app-index’;

templateUrl : ‘./index.component.html’,

styleUrls : [‘./index.component.css’]

} )

export class IndexComponent implements OnInit {

constructor(private \_router:Router) { }

ngOnInit( ) {

}

public clickMe( ):any{

// defining the URL dynamically

this.\_router.navitage([“/page\_three”]);

};

}

**Step#4:**  Implement the Routing in Single Page Applications by using “Routes” Array.

/ ==> PageoneComponent

/page\_two ==> PagetwoComponent

/page\_three ==> PagethreeComponent

i.e. route url should map to PageoneComponent and

“/page\_two” url should map to PagetwoComponent

Directory structure:

SPADemo1

src

app

routings

app.routes.ts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SPADemo1/src/app/routings/app.routes.ts:

import { Routes } from “@angular/router”;

import { PageComponent } from “../components/pageone/pageone.component”;

import { PagetwoComponent } from “../components/pagetwo/pagetwo.component”;

import { PagethreeComponent } from “../components/pagethree/pagethree.component”;

export const appRoutes:Routes = [

{ path ; “ “, component:PageoneComponent},

{ path ; “page\_two “, component:PagetwoComponent},

{ path ; “page\_three “, component:PagethreeComponent},

};

**Step#5:**  Load “appRoutes” in memory by using “RouterModule”.

**SPADemo1/src/app/app.module.ts:**

import { AppComponent } from “@angular/platform-browser”;

import { IndexComponent } from “

import { PageComponent } from “../components/pageone/pageone.component”;

import { PagetwoComponent } from “../components/pagetwo/pagetwo.component”;

import { PagethreeComponent } from “../components/pagethree/pagethree.component”;

import {RouterModule} from “@angular/router”;

import {appRoutes} from “./routings/app.routes”;

@NgModule({

declarations : [AppComponent, IndexComponent, PageoneComponent,

PagetwoComponent, PagethreeC

]

Imports : [ BroserModule, RouterModule.forRoot(appRoutes) },

Bootstrap : [IndexComponent }

-- forRoot() is the predefine function in RouterModule.

-- forRoot() function used to load the routes into memory.

**Index.html:**

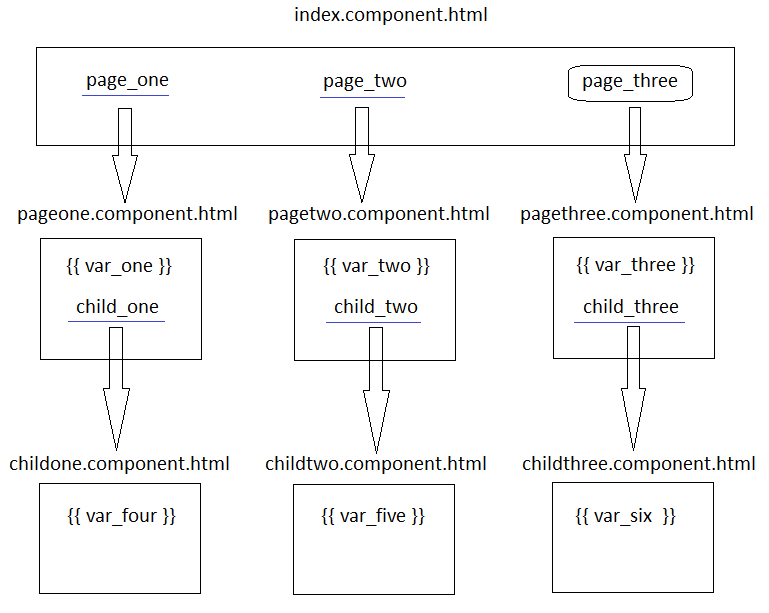
<body>

<app-index> </app-index>

</body>

**Nested Routing**

-- If we have routing more than one level called as Nested Routing in Single Page Application.



**Steps to implement the Nested Routing in Single Page Application:**

**Step#1:** Generate the Components.

> ng g c components/childone - -spec=false –is

> ng g c components/childtwo - -spec=false –is

> ng g c components/childthree - -spec=false –is

**Step#2:**  Implement the business logic in target components.

**SPADemo1/src/app/components/childone/childone.component.html:**

<h2 style=”color:brown”> {{ var\_four }} </h2>

**SPADemo1/src/app/components/childone/childone.component.ts:**

export class ChildoneComponent implements OnInit {

private var\_four:string;

constructor() [

this.var\_four=”I am from child one !!!”;

}

**SPADemo1/src/app/components/childtwo/childtwo.component.html:**

<h2 style=”color:coral”> {{ var\_five }} </h2>

**SPADemo1/src/app/components/childtwo/childtwo.component.ts:**

export class ChildtwoComponent implements OnInit {

private var\_five:string;

constructor() [

this.var\_five=”I am from child two !!!”;

}

**SPADemo1/src/app/components/childthree/childthree.component.html:**

<h2 style=”color:darkgreen”> {{ var\_six }} </h2>

**SPADemo1/src/app/components/childthree/childthree.component.ts:**

export class ChildthreeComponent implements OnInit {

private var\_six:string;

constructor() [

this.var\_six=”I am from child three !!!”;

}

**Step#3:**  Define Hyperlinks.

**SPADemo1/src/app/components/pageone/pageone.component.html**

<h1 style=”color:red”> {{ var\_one}}</h1>

<a [routerLink] = “[‘child\_one’]” > <b> Child\_One</b> </a>

<br>

<br>

<router-outlet> </router-outet>

**SPADemo1/src/app/components/pagetwo/pagetwo.component.html**

<h1 style=”color:green”> {{ var\_two}}</h1>

<a [routerLink] = “[‘child\_two’]” > <b> Child\_Two</b> </a>

<br> <br>

<router-outlet> </router-outet>

**SPADemo1/src/app/components/pagethree/pagethree.component.html**

<h1 style=”color:blue”> {{ var\_three}}</h1>

<a [routerLink] = “[‘child\_three’]” > <b> Child\_Three</b> </a>

<br> <br>

<router-outlet> </router-outet>

**Step#4:** Implement the Routings.

pageone ==> childone

pagetwo ==> childtwo

pagethree ==> childthree

**app.routes.ts:**

import ChildoneComponent } from “../component/childone/childone.component”;

import ChildoneComponent } from “../component/childone/childone.component”;

import ChildoneComponent } from “../component/childone/childone.component”;

export const appRoutes:Routes = [

{ path:” “, component:PageoneComponent, children:[{path:”child\_one”, component:ChildoneComponent}]},

{ path:” page\_two“, component:PagetwoComponent, children:[{path:”child\_two”, component:ChildtwoComponent}]},

{ path:”page\_three “, component:PagethreeComponent, children:[{path:”child\_three”, component:ChildthreeComponent}]}

];

**Route Guards**

1) **CanActivate:**

-- CanActivate is the Route Guard.

-- CanActivate is the predefined interface.

-- canActivate( ) is the boolean unimplemented method in CanActivate Interface.

-- CanActivate Route Guard used to give the permission to enter into Main Routes in Single Page Applications.

-- CanActivate interface is available in “@angular/router” package.

2) **CanDeactivate:**

-- CanDeactivate is the Route Guard.

-- CanDeactivate is the predefined interface.

-- canDeactivate( ) is the unimplemented boolean method in CanDeactivate interface.

-- CanDeactivate Route Guard used to give the permission to leave main Routings in Single Page Applications.

-- CanDeactivate interface is available in “@angular/router” package.

3) **CanActivateChild:**

-- CanActivateChild is the Route Guard.

-- CanActivateChild is the predefined interface.

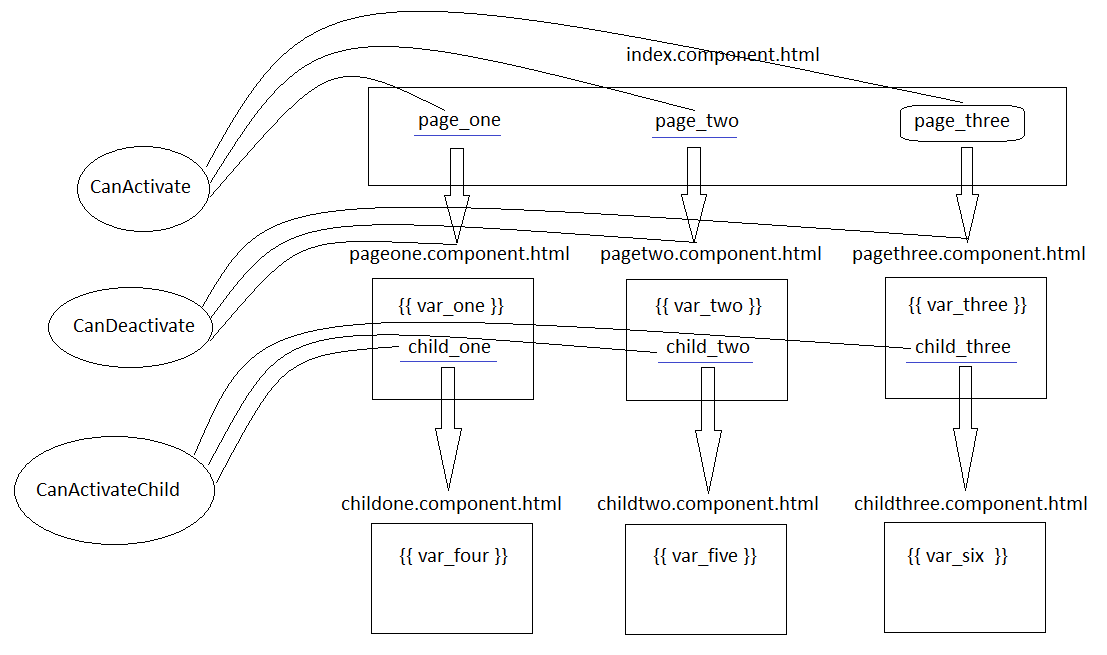
--canActivateChild( ) is the unimplemented boolean method in CanActivateChild interface.

-- CanActivateChild Route Guard used to give the permission to enter into Child Routes in Single Page Application.

-- CanActivateChild interface available in @angular/router packge.

Note: We will implement the Route Guards by using Custom Services.

**Steps to implement the Route Guards in Single Page Applications.**

****

**Step#1:**  implement the Route Guards by using Custom Service.

**Directory Structure**

SPADemo1

src

app

guards

// import Injectable

// Injectable is used to create Custom Service.

// we will implement Route Guards by using Custom Services only.

import { Injectable } from “@angular/core”;

// import CanActivate.

// CanActivate is used to give the permission to enter into main routes.

// import CanDeactivate.

// CanDeactive is used to give the permission to leave the main Routes.

// import CanActivateChild

// CanActivateChild is used to give the permission to enter into child route.

import { CanActivate, CanDeactivate, CanActivateChild }from “@angular/router”;

//import PagetwoComponent.

// we want to apply CanDeactivate to PagetwoComponent.

Import { PagetwoComponent } from “../components/pagetwo/pagetwo.component”;

// use Injectable

@Injectable()

// export the class

export class authGuards implements CanActivate, CanDeactivate<PagetwoComponent>,

CanActivateChild {

canActivate():boolean {

return confirm(“Do you want to enter into page\_two ??”);

};

canDeactivate():boolean {

return confirm(“Do you want to leave page\_two ??”);

};

canActivateChild(): boolean {

return confirm(“Do you want to enter into child\_one ??”);

};

};

**Step#2:**  Register the “authGuards” in module.

app.module.ts:

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

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

import {authGuards } from “./guards/auth/guards”;

-----

----

Providers: [authGuards]

-----

-----

**Step#3:** Apply the route guards to components.

PagetwoComponent ==> CanActivate

PagetwoComponent ==> CanDeactivate

ChildoneComponent ==> CanActivateChild

**SPADemo1/src/app/routings/app.routes.ts:**

…..

export const appRoutes:Routes = [

{ path:” “, component:PageoneComponent,

Children: [ {path: “child\_one”, component:ChildoneComponent}],

canActivateChild:[authGuards]},

{ path:”page\_two “, component:PagetwoComponent,

Children: [ {path: “child\_two”, component:ChildtwoComponent}],

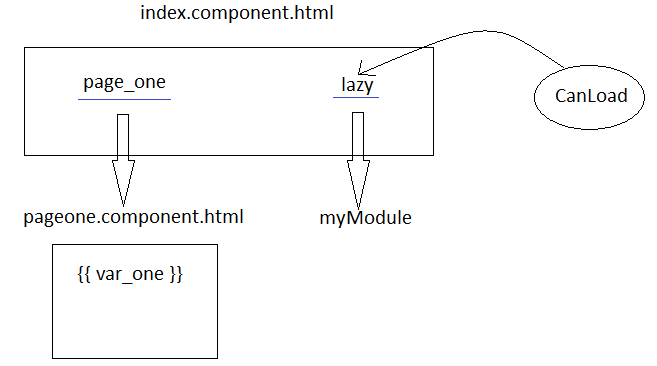
canActivate:[authGuards], canDeactivate: [authGuards]},

{ path:”page\_three “, component:PagethreeComponent,

Children: [ {path: “child\_three”, component:ChildthreeComponent}]}

};

**Example-1:**



-- In the above diagram we are navigating to Custom Module.

-- where “myModule” is the Custom Module

-- “myModule” making the Rest API Call.

-- Assume “myModule” is the heavy Component (Module).

-- Assume “myModule” receiving the images from Server.

-- If we load myModule at booting time automatically application booting time will be increased and application performance will be degraded.

-- So, as a Angular developer we must load “myModule” based on demand.

-- Loading myModule based on demand called as Lazy Loading.

**Steps to implement the lazy loading in Single Page Application.**

**Step#1:** Implement the Components.

Directory Structure:

SPADemo2

src

app

pageone.comonent.ts

pageone.comonent.html

index.component.ts

index.component.html

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**index.component.html:**

<a [routerLink]= “[‘/’]” <b> Page\_One </b> </a> &emsp;

<a [routerLink]= “[‘/lazy’]” <b> lazy </b> </a> &emsp;

<br> <br>

<router-outlet> </router-outlet>

**Index.component.ts:**

**pageone.component.html:**

**pageone.component.ts:**

import {component } from angular/core;

@Component

pageone

“./pageone.component.html”

})

Export class pageOneComponent{

private var\_one:string;

constructor() {

this.var\_one=” I am from Page-One”;

}

**Step#2:** implement myModule.

Directory structure:

SPADemo2

src

app

countries.service.ts

countries.component.ts

countries.component.html

my.module.ts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

URL: <https://restcountries.eu/rest/v2/all>

countries.service.ts:

import {Injectable} from “@angular/core”;

import {HttpClient } from “@angular/common/http”;

@Injectable()

export class countriesService{

constructor(private \_http:HttpClient) {};

public getCountries(){

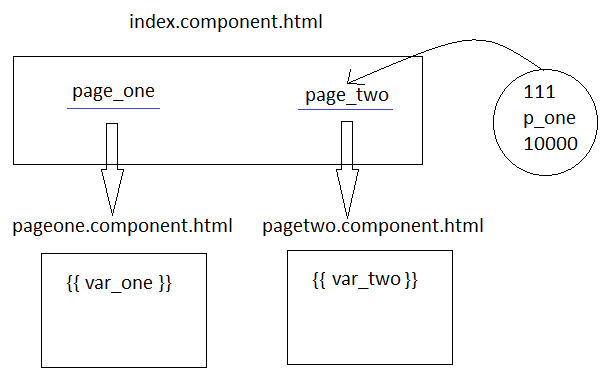
return this.\_http.get(“<https://restcountries.eu/rest/v2/all>”);

};

};

Refer the notes for continuation.

**Passing Routing parameters in Single Page Application**



Steps to implement the above application:

**Step#1**: Generate the Comonents.

index

pageone

pagetwo

> ng g c components/index - -spec=false –is

> ng g c components/pageone - -spec=false –is

> ng g c components/pagetwo - -spec=false -is

**Stepe#2:** Implement the presentation logic in source Component template.

**Index.comonent.html**

<a [router-link] =”[‘/’]”> <b> Page\_One </a> &emsp; &emsp; &emsp;

<a [router-link] =”[‘/page\_two’, 111, ‘p\_one’, 10000]”> ;

<b> Page\_Two</b>

<br> <br>

<router-outlet> </router-outlet>

**Step#3:** Implement the Routing.

**Directory Structure:**

SPADemo3

src

app

routing

app.routes.ts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**app.routes.ts**

import {Routes } from “@angular/router”};

import {PageoneComponent } “../components/pageone/pageone.component”;

import {PagetwoComponent } “../components/pagetwo/pagetwo.component”;

export const appRoutes:Routes = {

{path:” “, component : PageoneComponent},

{path: “page\_two/:p\_id/:p\_name/:p\_cost”, component: PagetwoComponent}

};

**Step#4:** Implement the business logic in target Component:

**pageone.component.html:**

<h1 style=”color:red”> {{ var\_one }} </h1>

**pageone.component.ts:**

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

@Component( {

selector : ‘app.pageone’,

templateUrl: ‘./pageone.component.html’,

styles : [ ]

} )

export class PageoneComponent implement OnInit{

private var\_one:string;

constructor(){

this.var\_one=”I am from Page One”;

}

ngOnInit() {

}

**pagetwo.component.html:**

<h1 style=”color:green”> {{ var\_two }} </h1>

**pagetwo.component.ts:**

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

// import ActivatedRoute.

// ActivatedRoute is the predefined class.

// ActivatedRoute class is used to read the routing parameters.

import {ActivatedRoute } from “@angular/router”;

@Component( {

selector : ‘app.pagetwo’,

templateUrl: ‘./pagetwo.component.html’,

styles : [ ]

} )

export class PagetwoComponent implement OnInit{

private var\_two:string;

constructor(private \_route:ActivatedRoute){ }

ngOnInit() {

/\* below syntax is related to Angular2

this.var\_two = this.\_route.snapshot.paras[“p\_id”] + “…… “ +

this.\_route.snapshot.paras[“p\_name”] + “….” +this.\_route.snapshot.paras[“p\_cost”]

\*/

// below syntax is related to Angular4 onwards

this.\_route.paramMap.subscribe(res=> {

this.var\_two = res.get(“p\_id”) + “….. “ +

res.get(“p\_name”) +”……” +res.get(“p\_cost”)

} ) ;

}

}

**Step#5:**  Load the routes into memory.

**app.module.ts:**

import BrowserModule

import NgModule

import PageoneComponent

import PagetwoComponent

import Component, OnInit}

import RouterModule from angular/router

import appRoutes from “./routing/app.routes”;

@NgModule( {

declaration : [ AppComponent, IndexComponent, PageoneComponent,

PagetwoComponent ],

imports : [BrowserModule, RouteModule.forRoot(appRoutes)],

providers : [ ],

bootstrap : [IndexComponent]

})

export class AppModule { }

**index.html:**

<body>

<app-index> </app-index>

</body>

Run the above application using below command:

SPADemo3> ng serve - -open

**url:** http:Localhost:4200/page\_two/111/p\_one/10000

**Rough**

**Installation of JSON Server:**

We can install json server by using below command.

C:\users\thiruk> npm install -g json-server

Create below file on desktop and save with “db.json”

{

“products”: [

{ “p\_id”:111, “p\_name”:”p\_one”, “p\_cost”:10000},

{ “p\_id”:222, “p\_name”:”p\_two”, “p\_cost”:20000},

{ “p\_id”:333, “p\_name”:”p\_three”, “p\_cost”:30000},

]

}

Deploy the above db.json file in JSON Server by using below command.

C:\users\thiruk\desktop>Json-server - -watch db.json

Test above file using below url using POSTMAN tool with get request.

http://localhost:3000/products

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Integration with PHP application

http://localhost:8085/api-provider/rest/employees

api-provider is the PHP application.

Test the application using POSTMAN tool

http://localhost:8085/api-provider/rest/employees/add

Get request

Keys:

ename

sal

addrs

dept

**Rough end**

**18/12/18**

**PHP Integration**

-- “api-provider” is the php application.

-- “api-provider” application fetch the data from “MySQL” Database.

-- We will deploy the “api-provider” application in xampp server.

--The following URL Representing the absolute path of “php” application in xampp server.

<http://localhost:8085/api-provider/rest/employees>

-- Above URL gives the JSON as response.

**\*\*\*\*\*\*\*\*\*\*\*\*\* JSON Server Integration \*\*\*\*\*\*\*\*\*\*\*\*\***

-- “json-server” is the light weight server.

-- “json-server” supports JSON.

-- By using “json-server” we can develop Rest API.

**Installation of JSON Server:**

> npm install json-server –g

-- automatically “json-server” installed globally.

**Create the database (JSON data) file:**

**Ex:** db.json

{

“products”: [

{ “p\_id”:111, “p\_name”:”p\_one”, “p\_cost”:10000},

{ “p\_id”:222, “p\_name”:”p\_two”, “p\_cost”:20000},

{ “p\_id”:333, “p\_name”:”p\_three”, “p\_cost”:30000},

{ “p\_id”:444, “p\_name”:”p\_four”, “p\_cost”:40000},

{ “p\_id”:555, “p\_name”:”p\_five”, “p\_cost”:50000}

]

}

**Load JSON file (ie. db.json) into server:**

> cd desktop

C:\Desktop>json-server - -watch db.json

-- After loading the above file into json-server, it will give below url which represents the Absolute path of JSON Server.

**http:// localhost:3000/products**

-- The above URL gives the JSON as Response.

**Example:** preSer13:

**preSer13/src/app/services/my.service.ts**:

import Injectable from @angular/core;

import HttpClient from @angular/common/http;

@Injectable( )

export class myService {

constructor(private \_http:HttpClient) { }

public getPhpData( ): any{

return this.\_http.get(“<http://localhost:8085/api-provider/rest/employees>”);

};

public getJSONData( ):any {

return this.http.get(“<http://localhost:3000/products>”);

};

};

**preSer13/src/app/services/my.component.ts**:

import { Component } from “@angular/core”;

import { myService }from “../services/my.service”;

import { HttperrorResponse } from “@angular/common/http”;

import {Observable } from “rxjs/Observable’;

import “rxjs/Rx”

@Component( {

selector:”json”,

templateUrl:”./my.component.html”

} )

export class myComponent{

private result\_one :any;

private result\_two: any;

public jsonSubscribe:any;

}

// create object to myService

constructor(private \_service:myService) { }

// first life cycle hook

ngOnInit( ){

this.jsonSubscribe = Observable.forkJoin([this.s-service.getPhpData( ),

this.\_service.getJSONData( )])

.subscribe(this.\_successCallBack, this.errorCallBack);

} ;

public \_successCallBack = (res) : (any) =>{

this.result\_one = res[0];

this.result\_two = res[1];

};

public \_errorCallBack = (err:HttpErrorResponse) : any => {

if (err.error instanceof Error) {

console.log(“Client Side Error”);

} else {

console.log(“Server Side Error”);

}

};

ngOnDestroy(){

this.jsonSubscribe.unsubscribe();

};

};

**my.component.html**

<h5 style = “color: green”> {{ result\_one | json}} </h5>

<h5 style = “color: indianred”> {{ result\_twos | json}} </h5>

**app.module.ts**

import BrowserModule from “@angular/platform-broser”;

import NgModule from “@angular/core”;

import Appcomponent } from

import myService from “./services/my.service”;

import myComponent from “./components/my.component”;

import HttpClientModule from “@angular/common/http”;

@NgModule( {

declrations: [ AppComponent, mycomponent],

imports: [ BrowserModule, HttpClientModue],

providers: [myService],

bootstrap : [myComponent]

})

Index.html:

<body>

<json> </json>

</body>

**19/12/18**

**FORMS**

**Model Driven FromExample**

**app.component.html:**

<div class =”container” novalidate>

<form [formGroup] = “profileData” (ngSubmit) = “register( )”>

<!-- Define Form Elements like First Name, Last Name, Age, Email, Address,

Gender etc.. -->

<!—First Name -->

<div class = “form-group”>

<label> First Name </label>

<input type=”text”

class=”form-control”

name=”fname”

formControlName=”fname”

required>

</div>

<div \*ngIf=”profileData.control[‘fname’].hasError(‘required’)”

class=”alert alert-danger”>

\*\*\*\*\* Can’t left blank \*\*\*\*\*

</div>

<div \*ngIf=”profileData.control[‘fname’].hasError(‘minlength’)”

class=”alert alert-danger”>

\*\*\*\*\* Minimum 3 Characters Are Required \*\*\*\*\*

</div>

<div \*ngIf=”profileData.control[‘fname’].hasError(‘maxlength’)”

class=”alert alert-danger”>

\*\*\*\*\* Maximum 6 Characteres Are Allowed \*\*\*\*\*

</div>

<!—Last Name -->

<div class = “form-group”>

<label> Last Name </label>

<input type=”text”

class=”form-control”

formControlName=”lname”

</div>

<!—Age -->

<div class = “form-group”>

<label> Age </label>

<input type=”text”

class=”form-control”

formControlName=”uage” >

</div>

<div \*ngIf=”profileData.controls[‘uage’].hasError(‘ageRange’)”

class = “alert alert-danger”>

\*\*\*\*\*\* Age Not Valid \*\*\*\*\*\*

</div>

<!—Email -->

<div class = “form-group”>

<label> Email </label>

<input type=”email”

class=”form-control”

formControlName=”uemail” >

</div>

<!—Sub Group -->

<div class = “form-group” formGroupName=”addr”>

<!—ucity -->

<div class=”form-group”

<label> City </label>

<input type=”text”

class=”form-control”

formControlName=”ucity” >

</div>

<!—Address Lane -->

<div class=”form-group”

<label> Address Lane </label>

<input type=”text”

class=”form-control”

formControlName=”uaddress” >

</div>

<!—Gender -->

<div class=”form-group”

<input type=”radio”

class=”form-control”

name = “gender”

value=”male”

formControlName=”gender” ><b> Male </b>

<input type=”radio”

class=”form-control”

name = “gender”

value=”female”

formControlName=”gender” ><b> FeMale </b>

</div>

<!—ucountry -->

<div class=”form-group”>

<select class = “form-control” formControlName=”ucountry”>

<option value=”india”> India </option>

<option value=”usa”> USA </option>

<option value=”uk”> UK </option>

<option value=”japan”> Japan </option>

<option value=”canada”> Canada</option>

</select>

</div>

<!—Submit Button -->

<div class=”form-group” align=”center”>

Input type=”submit” class=” btn btn-success”>

</div>

</form>

</div>

**app.component.ts:**

import Component from angular/core

import FormGroup, FormControl, Validators, AbstractControl, ValidatorFn} from angular/forms”;

@Component( {

selector : ‘app-root’,

tempateUrl : ‘./app.component.html’,

styleUrls : [‘./app.component.css’]

} )

export class AppcComponnt {

profileData: FormGroup;

constructor( ) {

this.profileData = new FormGroup( {

fname : new FormControl(“Naresh”, [Validator.required,

Validator.minLength(3), Validator.maxLength(6)]),

lname : new FormControl( ),

uage : new FormControl( 20, [this.ageRangeValidator( 25, 40) ] ),

uemail : new FormControl( ),

addr : new FormControl( {

ucity : new FormControl( ),

uaddress : new FormControl( )

} ),

gender : new FormControl( ),

ucountry : new FormControl( )

}) ;

};

public register( ):any{

console.log(this.profileData.value);

};

ageRangeValidator(min:number, max:number): ValidatorFn {

return (control: AbstractControl): { [key:string]:boolean} | null => {

if (control.value !== undefined && (isNan(control.value) ||

control.value < min || control.value > max)){

return {‘ageRnge’ : true};

}

return null;

};

}

}

**app.module.ts:**

import ( BrowserModule ) from ‘@angular/platform-browser’;

import ( NgModule ) from ‘@angular/core’;

import ( AppComponent ) from ‘./app.component’;

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

@NgModule( {

declarations: [ AppComponent ],

imports: [ BrowserModule, ReactiveFormModule ],

providers: [ ],

bootstrap: [ appComponent]

})

export class AppModule { }

**20/12/18**

**Template Driven Forms Example**

**app.component.html:**

<div class = “container”>

<form #profileData=”ngForm” (ngSubmit) = “register(profileData.value)”>

**<!—Define the Form elements like First Name, Last Name, Email,…etc.-->**

**<!—First Name -->**

<div class = “form-group”>

<label> First Name </label>

<input type=”text”

class=”form-control”

name=”fname” ~~ngModel~~

[(ngModel)] = “fname1”

#fname=ngModel

required>

<!-- whatever we enter the data into text, will be stored in “fname” and fname will be stored in parent i.e. profileData because of “ngForm” directive. -->

</div>

<div [hidden] =”fname.valid” class =”alert alert-danger”>

\*\*\*\*\* Can’t Left Blank \*\*\*\*\*\*

</div>

**<!—Last Name -->**

<div class = “form-group”>

<label> Last Name </label>

<input type=”text”

class=”form-control”

name=”lname”

ngModel>

</div>

<!—Email -- >

<div class = “form-group”>

<label> Email</label>

<input type=”email”

name=”uemail”

class=”form-control”

ngModel>

</div>

<!—Sub Group -- >

<div class = “form-group” ngModelGroup=”addr”>

<!—ucity -- >

<div class = “form-group”>

<label> City</label>

<input type=”text”

name=”ucity”

class=”form-control”

ngModel>

</div>

<!—address lane -- >

<div class = “form-group”>

<label> Address Lane</label>

<input type=”text”

name=”uaddress”

class=”form-control”

ngModel>

</div>

</div>

<!—Gender -- >

<div class = “form-group”>

<label> Gender</label>

<input type=”radio”

class=”form-control”

name=”gender”

value=”male”

ngModel> <b> Male</b>

<input type=”radio”

class=”form-control”

name=”gender”

value=”female”

ngModel> <b> Female</b>

</div>

<!—Country Dropdown -- >

<div class = “form-group”>

<select class=”form-control” name=”ucountry’ ngModel>

<option value=”india”> India </option>

<option value=”usa”> USA </option>

<option value=”uk”> UK </option>

<option value=”canada”> Canada </option>

</select>

</div>

<!—submit button -->

<div class =”form-group” align=”center”>

<input type=”submit” class =” btn btn-success”>

</div.

</form>

</div>

**app.component.ts:**

import Component from angular/core;

@Component ({

selector :’app-root’;

templateUrl: ‘./app.component.html’;

styleUrls: [‘./app.component.css’]

})

export class AppComponent {

title = ‘app”;

public register(data):any{

console.log(data);

};

}

**app.module.ts:**

import ( BrowserModule ) from ‘@angular/platform-browser’;

import ( NgModule ) from ‘@angular/core’;

import ( AppComponent ) from ‘./app.component’;

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

@NgModule( {

declarations: [ AppComponent ],

imports: [ BrowserModule, FormModule ],

providers: [ ],

bootstrap: [ appComponent]

})

export class AppModule { }

Run the application:

TDFEx>ng serve - -open

Problem with this approach:

1. Less control to developer.

**21/12/18**

**MINI PROJECT**

Project Architecture:

**Steps to implement the above application**

**Step#1:** Make the MySQL ready for both authentication and home module data.

mysql> create schema miniproject

mysql> use miniproject

mysql> create table login\_details(uname varchar(20), upwd varchar(20));

mysql> insert into login\_details values (“admin”, “admin”);

mysql> select \* from login\_details.

mysql> create table products(p\_id integer, p\_name varchar(20), p\_cost integer);

mysql> insert into products values (111, ‘p\_one”, 10000);

mysql> insert into products values (222, ‘p\_two”, 20000);

mysql> insert into products values (333, ‘p\_three”, 30000);

mysql> insert into products values (444, ‘p\_four”, 40000);

mysql> insert into products values (555, ‘p\_five”, 50000);

mysql> select \* from products;

**i) Export database:**

* c:\Program Files (x86)\MySQL\MySQl Server\bin>mysqldump –u root –p root miniproject > miniproject.sql

Enter Password:

**ii) :** **Import database:**

**1)**  create the database

> create schema miniproject

2) run the following command

c:\Program Files (x86)\MySQL\MySQl Server\bin>

mysql –u root –p miniproject < miniproject.sql

mysql >drop schema miniproject

* show databases

**Step#2**: Make the MongoDB ready.

1. **Export MongoDB database.**
2. run terminal in administrator mode
3. navigate to bin location

C:\Program Files\MongoDB\Server\4.0\bin>

1. run the following command.

mongodump - -db miniproject - -out C:/users/thiruk/desktop/mongodb

i.e. C:\Program Files\MongoDB\Server\4.0\bin> mongodump - -db miniproject - -out C:/users/thiruk/desktop/mongodb

-- automatically “miniproject” exported

**ii) Import MongoDB database:**

mongorestore -d miniproject C:\users\thiruk\desktop\mongodb/miniproject

**Step#3: create sample.json file**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

miniproject@11am

static

sample.json

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**sample.json:**

[

{“p\_id”:111, “p\_name”: “p\_one”, “p\_cost”:10000},

{“p\_id”:222, “p\_name”: “p\_two”, “p\_cost”:20000},

{‘p\_id”:333, “p\_name”: “p\_three”, “p\_cost”:30000}

}

**Step#4:** Download the following Node modules.

1. express
2. body-parser
3. mysql
4. mongodb@2.2.32
5. jwt-simple
6. cors

>yarn add express mysql [mongodb@2.2.32](mailto:mongodb@2.2.32) cors body-parser jwt-simple - -save

**Rough:**

* show dbs;
* use miniproject
* db.createCollection(“products”);
* db.products.insert({“p\_id”:111, “p\_name”: “p\_one”, “p\_cost”:10000} );
* db.products.insert({“p\_id”:222, “p\_name”: “p\_one”, “p\_cost”:10000} );
* db.products.insert({“p\_id”:333, “p\_name”: “p\_one”, “p\_cost”:10000} );
* db.products.insert({“p\_id”:444, “p\_name”: “p\_one”, “p\_cost”:10000} );
* db.products.insert({“p\_id”:555, “p\_name”: “p\_one”, “p\_cost”:10000} );
* db.products.find( );

**26/12/18**

Import BrowserModule

LoginComponent ‘./components/login/login.component

DashboardComponent

HomeComponent ‘./components/home/home.component

AboutComponent ‘./components/about/about.component

ContactComponent

LoginService “./services/login.service”;

AboutService “./services/about.service”;

HomeService “./services/home.service”;

ContactService “./services/contact.service”;

LogoutService “./services/logout.service”;

appRoutes “./routing/app.routes”;

RouterModule ‘@angular/router”;

HttpClientModule “@angular/common/http”;

FormsModule “@angular/forms”;

@NgModule( {

declarations: [AppComponent, IndexComponent, LoginComponent,

DashboardComponent, HomeComponent, AboutComponent,

ContactComponent],

imports: [ BrowserModule, RouterModule.forRoot(appRoutes),

FormsModule, HttpClientModule ],

providers: [ LoginService, AboutService, HomeService, ContactService,

LogoutService],

bootstrap: [ IndexComponent]

})

export class AppModule { }

**index.html:**

<doctype html>

<html lang=”en”>

<head>

<title>