**Lesson22 RXJS operators in angular services**

**Notes:-**

**1-RxJS 5 vs 6**  
An Angular 6 project, by default uses RxJS version 6. RxJS 6 has some breaking changes compared to RxJS 5.5 and older versions.   
The way we import some of the classes like Observable and Subject has changed in RxJS 6.   
In RxJS 5, we import Observable and Subject classes as shown below.

import { Observable } from 'rxjs/Observable';

import { Subject } from 'rxjs/Subject';

In **RxJS 6**, this has changed to 

import { Observable, Subject } from 'rxjs';

Similarly, the way we import operators also changed in RxJS 6. To import catchError operator in RxJS 5, we use the following 

import { catchError } from 'rxjs/operators/catchError';

In **RxJS 6**, it has changed to the following 

import { catchError } from 'rxjs/operators';

In RxJS 6, we import all the operators from 'rxjs/operators' 

import { map, delay, catchError } from 'rxjs/operators';

For example, in v5 to create an ErrorObservable we might use one of the following 

new ErrorObservable('Your error message');  
  
OR  
  
ErrorObservable.create('Your error message');

In v6, we use throwError() function to achieve this.  

return throwError('Your error message');

**Steps:-**

**1-execute the following sql statments**

**create table Skills(**

**ID int primary key identity(1,1),**

**proficiency nvarchar(50),**

**skillName nvarchar(50),**

**experienceInYears int) ;**

**create table Employee(**

**id int primary key identity(1,1),**

**fullName nvarchar(50),**

**contactPreference nvarchar(50),**

**phone nvarchar(50),**

**email nvarchar(50),**

**SkillID int references Skills(ID));**

**create table EmployeeSkills(**

**EmpID int references Employee(id),**

**SkillID int reference Skills(id),**

**Primary key (EmpID,SkillID)**

**);**

**2-we apply in the web.config the following code in the global.ascx**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Web;**

**using System.Web.Http;**

**using System.Web.Mvc;**

**using System.Web.Optimization;**

**using System.Web.Routing;**

**namespace APIPRO{**

**public class WebApiApplication : System.Web.HttpApplication{**

**protected void Application\_Start(){**

**AreaRegistration.RegisterAllAreas();**

**GlobalConfiguration.Configure(WebApiConfig.Register);**

**FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);**

**RouteConfig.RegisterRoutes(RouteTable.Routes);**

**BundleConfig.RegisterBundles(BundleTable.Bundles);**

**//the below code will resolve the query that get values from multiple tables**

**GlobalConfiguration.Configuration.Formatters.JsonFormatter.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore;**

**GlobalConfiguration.Configuration.Formatters.Remove(GlobalConfiguration.Configuration.Formatters.XmlFormatter);}}}**

**3-in the CompnayDBEntities1 we enable the lazy loading**

public partial class CompnayDBEntities1 : DbContext{

public CompnayDBEntities1()

: base("name=CompnayDBEntities1"){this.Configuration.LazyLoadingEnabled = true;}

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

import { Employee } from '../Models/employee';

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

import { Observable, throwError } from 'rxjs';

import { catchError } from 'rxjs/operators';

@Injectable({

providedIn: 'root'})

export class EmployeeServService {

baseUrl = 'http://localhost/APIPRO/api/Employees/';

constructor(private httpClient: HttpClient) {}

getEmployees(): Observable<Employee[]> {

return this.httpClient.get<Employee[]>(this.baseUrl+'/LoadEmployees')

.pipe(catchError(this.handleError));}

private handleError(errorResponse: HttpErrorResponse) {

if (errorResponse.error instanceof ErrorEvent) {

console.error('Client Side Error :', errorResponse.error.message);}

else {console.error('Server Side Error :', errorResponse);}

return throwError('There is a problem with the service. We are notified & working on it. Please try again later.');}

getEmployee(id: number): Observable<Employee> {

return this.httpClient.get<Employee>(this.baseUrl+"/"+id).pipe(catchError(this.handleError));}

addEmployee(employee: Employee): Observable<Employee> {

return this.httpClient.post<Employee>(this.baseUrl, employee, {

headers: new HttpHeaders({

'Content-Type': 'application/json'})})

.pipe(catchError(this.handleError));}

updateEmployee(employee: Employee): Observable<void> {

return this.httpClient.put<void>(this.baseUrl+"/"+employee.id, employee, {

headers: new HttpHeaders({

'Content-Type': 'application/json'})})

.pipe(catchError(this.handleError));}

deleteEmployee(id: number): Observable<void> {

return this.httpClient.delete<void>(this.baseUrl+"/"+id)

.pipe(catchError(this.handleError));}}

<div class="table-responsive">

<table class="table table-bordered" \*ngIf="employees && employees.length">

<thead>

<tr class="bg-primary">

<th>Name</th>

<th>Email</th>

<th>Phone</th>

<th>Contact Preference</th>

<th>Action</th>

</tr>

</thead>

<tbody>

<tr \*ngFor="let employee of employees">

<td>{{ employee.fullName }}</td>

<td>{{ employee.email }}</td>

<td>{{ employee.phone }}</td>

<td>{{ employee.contactPreference }}</td>

<td> <button class="btn btn-primary">Edit</button> </td>

</tr>

</tbody>

</table>

</div>

**=============================================================**

On the empService.ts we write the following code

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

import { EmployeeServService } from '../../services/employee-serv.service';

import { Employee } from '../../Models/employee';

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

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

@Component({

selector: 'app-employee-comp',

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

styleUrls: ['./employee-comp.component.css']})

export class EmployeeCompComponent implements OnInit {

employees: Employee[];

//we use this variable to load the form array

skillSets:Array<ISkill>;

skill:ISkill;

//we see that we apply dependency injection on the service

constructor(private fb: FormBuilder,

private route: ActivatedRoute,private employeeService: EmployeeServService) { }

ngOnInit() {

this.employeeForm = this.fb.group({

fullName: ['', Validators.compose([Validators.required, Validators.minLength(2), Validators.maxLength(10)])],

contactPreference: ['email'],

emailGroup: this.fb.group({

email: ['', Validators.compose([Validators.required,emailDomain('hotmail.com')])],

confirmEmail:['',[Validators.required]],}, { validator: matchEmails }),

phone: [''],

skills: this.fb.array([

this.AddSkillFormGrouping()])});

this.employeeForm.valueChanges.subscribe((data) => {

this.logValidationErrors(this.employeeForm);});

this.employeeForm.get('contactPreference').valueChanges.subscribe((data) =>{this.onContactPrefernceChange(data);})

//we see that we apply the Routing with parameter to get the Employee and //to fill the Skills array

this.route.paramMap.subscribe(params => {

const empId = +params.get('id');

if (empId) {

this.getEmployee(empId);

this.logValidationErrors();}});}

//we get the employee with it’s skills

getEmployee(id: number) {

this.employeeService.getEmployee(id).subscribe(

(res:Employee[]) => {

let no:number = res.length-1;

//we define the Array<ISkill> to insert the object to the Array<ISkill>

this.skillSets = new Array<ISkill>();

for(let i = 0;i <= no;i++){

this.skill = new ISkill();

this.skill.skillName = res[i].skillName;

this.skill.experienceInYears = res[i].experienceInYears;

this.skill.proficiency = res[i].proficiency;

this.skillSets[i] = this.skill;}

this.editEmployee(res[0]);

//we load the formArray and set the Array<Skills> inside it

this.employeeForm.setControl('skills', this.setExistingSkills(this.skillSets));},

(err: any) => console.log(err));}

//we load the object with its property in the form

this.employeeForm.patchValue({

fullName: emp.fullName,

contactPreference: emp.contactPreference,

emailGroup: {

email: emp.email,

confirmEmail: emp.email},

phone: emp.phone});}

//In the below method we push the object in the Array<>

setExistingSkills(skillSets: ISkill[]): FormArray {

const formArray = new FormArray([]);

skillSets.forEach(s => {

formArray.push(this.fb.group({

skillName: s.skillName,

experienceInYears: s.experienceInYears,

proficiency: s.proficiency}));});

return formArray;}