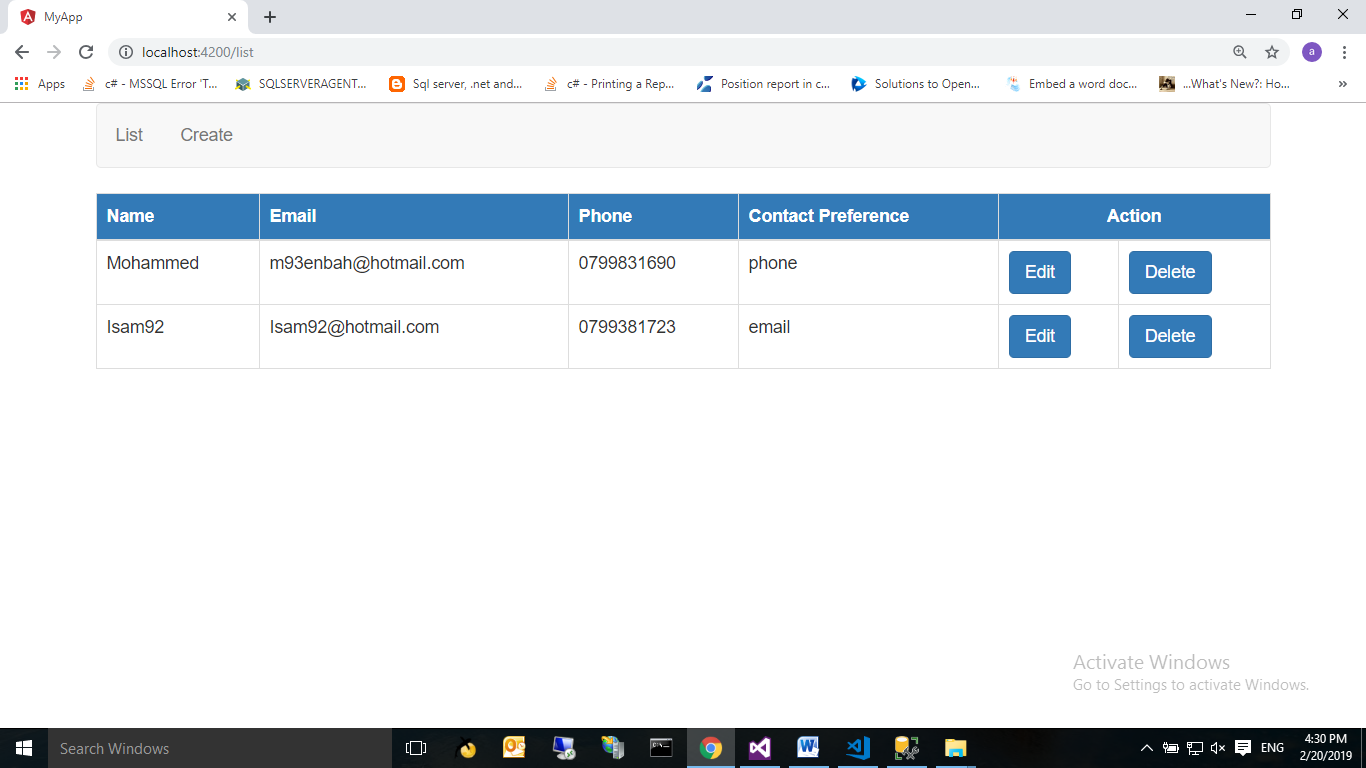
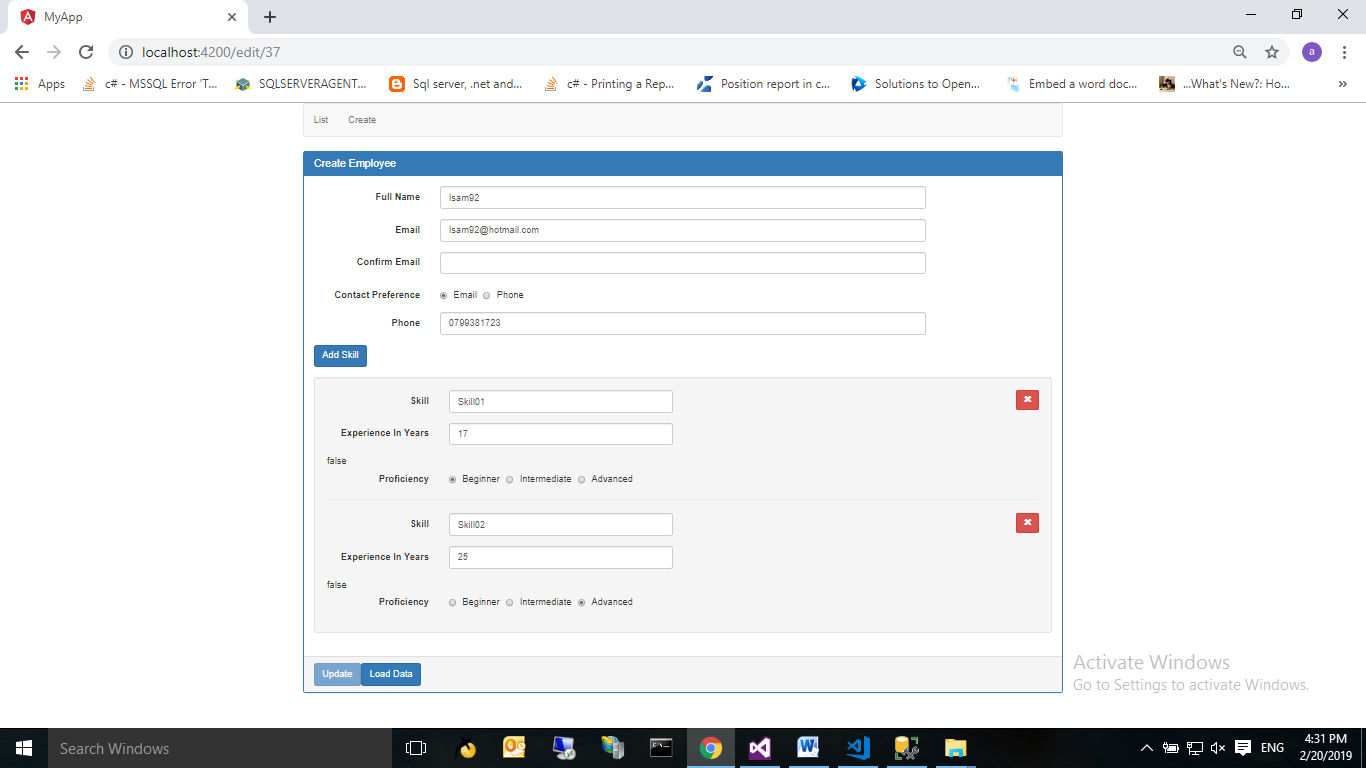
**Company Project Sample**

**Notes:-**

**1-In the following page we do different operation, we do page with full CRUD operation (insert, update, delete, load By ID, load All)**



**2-When Click Edit Button we see the following info like the following**



**3-we can make delete of specific skill of the group of skills**

**4-we can insert multiple skills on the same employee**

**4-we can update as I want of the skills and employee info**

**5-we can delete employee with all its info (such as the skills related with the employee)**

**Steps:-**

**1-we create skills class and create IEmployee class**

//on the ISkill class we write the following code

export class ISkill {

skillName: string;

experienceOfYear: number;

proffeicency: string;

empId:number;

}

//on the IEmployee class we write the following code

import {ISkill} from './skills';

export class IEmployee {

id: number;

fullName: string;

email: string;

confirmEmail:string;

phone: number;

contractPreference: string;

Skills: Array<ISkill>;}

**2-In the app.module we import the following 2 component and apply the routing to the following 2 component**

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

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

import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';

import { EmployeeEntryCompComponent } from './Components/employee-entry-comp/employee-entry-comp.component';

import { EmployeeLstCompComponent } from './Components/employee-lst-comp/employee-lst-comp.component';

import { MenuCompComponent } from './components/menu-comp/menu-comp.component';

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

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

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

@NgModule({

declarations: [

AppComponent,

EmployeeEntryCompComponent,

EmployeeLstCompComponent,

MenuCompComponent],

imports: [

BrowserModule,

AppRoutingModule,

FormsModule,

ReactiveFormsModule,

HttpClientModule],

providers: [],

bootstrap: [AppComponent]})

export class AppModule { }

**//In App Routing Module we apply the Routing Needed for the 2 component**

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

import { Routes, RouterModule } from '@angular/router';

import { EmployeeEntryCompComponent } from './Components/employee-entry-comp/employee-entry-comp.component';

import { EmployeeLstCompComponent } from './Components/employee-lst-comp/employee-lst-comp.component';

const routes: Routes = [

{ path: 'list', component: EmployeeLstCompComponent },

{ path: 'create', component: EmployeeEntryCompComponent },

{ path: 'edit/:id', component: EmployeeEntryCompComponent },

{ path: '', redirectTo: '/list', pathMatch: 'full' }];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]})

export class AppRoutingModule { }

**3-On the Service we apply the definition of the services required**

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

import { IEmployee } 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 EmployeeServiceService {

baseUrl = 'http://localhost:34173/api/Employee';

employees:IEmployee[];

employee:IEmployee;

constructor(private httpClient: HttpClient) { }

getEmployees(): Observable<IEmployee[]> {

const httpOptions ={

headers: new HttpHeaders({

'accept': 'application/json'})};

return this.httpClient.get<IEmployee[]>(this.baseUrl+'/loadEmployees',httpOptions)

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

getEmployee(id: Number): Observable<IEmployee> {

const httpOptions ={

headers: new HttpHeaders({

'accept': 'application/json'})};

return this.httpClient.get<IEmployee>(this.baseUrl+'/LoadEmployeeById?EmpID='+id,httpOptions)

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

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

const httpOptions ={

headers: new HttpHeaders({'accept': 'application/json'})};

return this.httpClient.post<IEmployee>(this.baseUrl+'/PostEmployee', employee, httpOptions

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

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

const httpOptions ={

headers: new HttpHeaders({

'accept': 'application/json'})};

return this.httpClient.put<void>(this.baseUrl+'/PutEmployee?id='+employee.id, employee, httpOptions).pipe(catchError(this.handleError));}

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

return this.httpClient.delete<void>(this.baseUrl+'/DeleteEmployee?empID='+id)

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

deleteSkill(id: Number): Observable<void> {

return this.httpClient.delete<void>(this.baseUrl+'/DeleteSkill?skillID='+id)

.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.');}}

**4-we create folder Validators and write the following code**

import { AbstractControl } from "@angular/forms";

export {emailDomain,matchEmails}

function emailDomain(domainName: string,domainName2: string) {

return (control: AbstractControl): { [key: string]: any } | null => {

const email: string = control.value;

const domain = email.substring(email.lastIndexOf('@') + 1);

if (email === '' || domain.toLowerCase() === domainName || domain.toLowerCase() === domainName2) {return null;} else {return { 'emailDomain': true };}}}

function matchEmails(group: AbstractControl): { [key: string]: any } | null {

const emailControl = group.get('email');

const confirmEmailControl = group.get('ConfirmEmail');

if (emailControl.value === confirmEmailControl.value || (confirmEmailControl.pristine && confirmEmailControl.value === '')) {

return null;} else {return { 'CompareEmails': true };}}

**5-we create folder component and we create component called**

**employee-lst-comp.component.html we write the following code**

<div class="table-responsive">

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

<thead>

<tr class="bg-primary">

<th>Name</th>

<th>Email</th>

<th>Phone</th>

<th>Contact Preference</th>

<th colspan="2" style="text-align:center;">Action</th>

</tr>

</thead>

<tbody>

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

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

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

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

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

<td><button class="btn btn-primary" (click)="editButtonClick(employee.id)">Edit

</button></td>

<td><button class="btn btn-primary" (click)="deleteButtonClick(employee.id)">

Delete</button></td>

</tr>

</tbody>

</table>

</div>

**And in the employee-lst-comp.component.ts we write the following code**

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

import { EmployeeServiceService } from 'src/app/Services/employee-service.service';

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

@Component({

selector: 'app-employee-lst-comp',

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

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

})

export class EmployeeLstCompComponent implements OnInit {

constructor(private empService:EmployeeServiceService,private \_route:Router) { }

ngOnInit() {

debugger;

this.empService.employees = null;

this.LoadEmployees();}

LoadEmployees(){

this.empService.getEmployees().subscribe(

(employeesLst) => {

this.empService.employees = employeesLst;

debugger;},

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

//when the user click to edit button we navigate to employee-entry component

editButtonClick(employeeId:Number){this.\_route.navigate(['/edit', employeeId]);}

//when the user click to delete button we called Load Employees Service

deleteButtonClick(employeeId:Number){

this.empService.deleteEmployee(employeeId)

.subscribe(() => this.LoadEmployees() ,(err: any) => console.log(err)

);;}}

**6-we create the employee-entry component html part and write the code needed.**

<form class="form-horizontal" [formGroup]="employeeForm">

<div class="panel panel-primary">

<div class="panel-heading">

<h3 class="panel-title">Create Employee</h3>

</div>

<div class="panel-body">

//In the below expression we make condition if the fullName form control is invalid and touched then apply the has-error class to the whole div

<div class="form-group" [ngClass]="{'has-error':

employeeForm.get('fullName').invalid && employeeForm.get('fullName').touched}">

<label class="col-sm-2 control-label" for="fullName">Full Name</label>

<div class="col-sm-8">

<input id="fullName" type="text" class="form-control" formControlName="fullName" (blur)="logControlsInsideFormGroup()">

<span class="help-block" \*ngIf="((employeeForm.get('fullName').touched ||

employeeForm.get('fullName').dirty) && employeeForm.get('fullName').errors)">

//If the fullName is not null then show the error message

<span class="help-block" \*ngIf="formErrors.fullName">

{{formErrors.fullName}}</span></span>

</div></div>

//we make inner form group called emailGroup to apply Custom Validation to the //email and confirm email form controls

<div formGroupName="emailGroup">

<div class="form-group" [ngClass]="{'has-error':

employeeForm.get('emailGroup').get('email').invalid && employeeForm.get('emailGroup').get('email').touched}">

<label class="col-sm-2 control-label" for="email">Email</label>

<div class="col-sm-8">

<input id="email" type="text" class="form-control" formControlName="email" (blur)="logControlsInsideFormGroup()">

<span class="help-block" \*ngIf="((employeeForm.get('emailGroup').get('email').touched ||

employeeForm.get('emailGroup').get('email').dirty) &&

employeeForm.get('emailGroup').get('email').errors)">

<span class="help-block" \*ngIf="formErrors.email">

{{formErrors.email}}

</span></span></div></div>

<div class="form-group" [ngClass]="{'has-error':

employeeForm.get('emailGroup').get('email').invalid && employeeForm.get('emailGroup').get('ConfirmEmail').touched}">

<label class="col-sm-2 control-label" for="ConfirmEmail">Confirm Email</label>

<div class="col-sm-8">

<input id="ConfirmEmail" type="text" class="form-control" formControlName="ConfirmEmail" (blur)="logControlsInsideFormGroup()">

<span class="help-block" \*ngIf="formErrors.emailGroup || formErrors.ConfirmEmail">

//we make condition if the form Errors. Confirm Email is not null or email Group that represent the validation message of confirm email then show it

{{formErrors.ConfirmEmail ? formErrors.ConfirmEmail : formErrors.emailGroup}}

</span></div></div></div>

<div class="form-group" [ngClass]="{'has-error':

employeeForm.get('contactPreference').invalid && employeeForm.get('contactPreference').touched}">

<label class="col-md-2 control-label">Contact Preference</label>

<div class="col-md-8">

<label class="radio-inline">

//we make event to set validation on phone form control to be required or not

<input type="radio" value="email" formControlName="contactPreference" (click)="onContactPrefernceChange('email')">Email

</label>

<label class="radio-inline">

<input type="radio" value="phone" formControlName="contactPreference" (click)="onContactPrefernceChange('phone')">Phone

</label>

</div>

</div>

<div class="form-group" [ngClass]="{'has-error':

employeeForm.get('phone').invalid && employeeForm.get('phone').touched}">

<label class="col-sm-2 control-label" for="phone">Phone</label>

<div class="col-sm-8">

<input id="phone" type="text" class="form-control" formControlName="phone" (blur)="logControlsInsideFormGroup()">

<span class="help-block" \*ngIf="((employeeForm.get('phone').touched ||

employeeForm.get('phone').dirty) &&

employeeForm.get('phone').errors)">

<span class="help-block" class="help-block" \*ngIf="formErrors.phone">

{{formErrors.phone}}

</span></span></div></div>

//The below button is used for add Dynamic form group inside form array

<div class="form-group">

<div class="col-md-2">

<button type="button" class="btn btn-primary" (click)="addSkillButtonClick()" [disabled]="employeeForm.get('skills').invalid">

Add Skill</button></div></div>

<div class="well">

//we makes for each loop of the entire inner form group inside form array

<div formArrayName="skills" \*ngFor="let skill of employeeForm.get('skills').controls; let i = index">

//we set i that represent the form group index

<div [formGroupName]="i">

<hr \*ngIf="i>0">

//we make unique id for each form control in each form array

<input type="hidden" class="form-control" [id]="'id'+i" formControlName="id">

<input type="hidden" class="form-control" [id]="'empId'+i" formControlName="empId">

//As you see, we get the control by access it by skill variable to reach it’s value or it’s state to detect the status it has been

<div class="form-group" [ngClass]="{'has-error':

skill.get('skillName').invalid && skill.get('skillName').touched}">

<label class="col-sm-2 control-label" for="skillName" [attr.for]="'skillName'+i">

Skill</label>

<div class="col-sm-4">

//we see that we make event called (blur) which means that when release the form //control it will check the validation of the that form control

<input type="text" class="form-control" [id]="'skillName'+i" placeholder="Name" formControlName="skillName"(blur)="logControlsInsideFormGroup()">

<span class="help-block" \*ngIf="skill.get('skillName').errors?.required &&

skill.get('skillName').touched">

<span class="help-block" \*ngIf="formErrors.skillName">{{formErrors.skillName}}

</span></span></div>

<div class="col-sm-6" >

//as you can see , we can pass the value of each form control value

<button type="button" class="btn btn-danger btn-sm pull-right" [disabled]="skill.invalid || employeeForm.get('skills').length <1"

title="Delete Skill" (click)="removeSkillButtonClick(i,skill.get('id').value)">

<span class="glyphicon glyphicon-remove"></span>

</button></div></div>

<div class="form-group" [ngClass]="{'has-error':

skill.get('experienceOfYear').invalid && skill.get('experienceOfYear').touched}">

<label class="col-sm-2 control-label" [attr.for]="'experienceOfYear'+i">

Experience In Years</label>

<div class="col-sm-4">

<input type="text" [id]="'experienceOfYear'+i" placeholder="Experience in Years" class="form-control"

formControlName="experienceOfYear" (blur)="logControlsInsideFormGroup()">

<span class="help-block" \*ngIf="skill.get('experienceOfYear').errors?.required &&

skill.get('experienceOfYear').touched">

<span class="help-block" \*ngIf="formErrors.experienceOfYear">

{{formErrors.experienceOfYear}}

</span></span></div></div>

{{skill.get('proffeicency').touched}}

<div class="form-group" [ngClass]="{'has-error':

skill.get('proffeicency').invalid && skill.get('proffeicency').touched}">

<label class="col-md-2 control-label" [attr.for]="'proffeicency'+i">Proficiency</label>

<div class="col-md-8">

<label class="radio-inline">

<input [id]="'proffeicency'+i" type="radio" value="Beginner" formControlName="proffeicency" (blur)="logControlsInsideFormGroup()">Beginner

</label>

<label class="radio-inline">

<input [id]="'proffeicency'+i" type="radio" value="Intermediate" formControlName="proffeicency" (blur)="logControlsInsideFormGroup()">Intermediate

</label>

<label class="radio-inline">

<input [id]="'proffeicency'+i" type="radio" value="Advanced" formControlName="proffeicency" (blur)="logControlsInsideFormGroup()">Advanced

</label>

<span class="help-block" \*ngIf="formErrors.proffeicency">

{{formErrors.proffeicency}}

</span></div></div></div></div></div></div>

<div class="panel-footer">

<button class="btn btn-primary" type="button" (click)="SaveEmployee()" [disabled]="employeeForm.invalid" [ngClass]="SetSaveButtonState()">Save</button>

<button class="btn btn-primary" type="button" (click)="UpdateEmployee()" [disabled]="employeeForm.invalid" [ngClass]="SetUpdateButtonState()" >Update</button>

<button class="btn btn-primary" type="button" (click)="onLoadDataClick()">Load Data</button>

</div>

</div>

</form>

<!-- <table>

**7-In the component called employee-entry-comp.component.ts we write the following code**

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

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

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

import {emailDomain,matchEmails} from '../../Validators/custom-validators';

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

import { EmployeeServiceService } from 'src/app/Services/employee-service.service';

import { IEmployee } from 'src/app/Models/employee';

import { ISkill } from 'src/app/Models/skills';

@Component({

selector: 'app-employee-entry-comp',

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

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

})

export class EmployeeEntryCompComponent implements OnInit {

// employee:IEmployee;

constructor(private fb: FormBuilder,private \_route:ActivatedRoute,private router:Router,private empService:EmployeeServiceService)

{

}

public employeeForm: FormGroup;

Arry:FormArray;

BtnSaveStatus:boolean;

BtnUpdateStatus:boolean;

empId:Number;

BtnState:string = "Save";

formErrors = {

'fullName': '',

'emailGroup':'',

'email': '',

'ConfirmEmail':'',

'phone':'',

'skillName': '',

'experienceOfYear': '',

'proffeicency': ''

};

// This object contains all the validation messages for this form

validationMessages = {

'fullName': {

'required': 'Full Name is required.',

'minlength': 'Full Name must be greater than 5 characters.',

'maxlength': 'Full Name must be less than 10 characters.',

},

'emailGroup':

{

'CompareEmails':'Email And Confirm Email Must Match',

},

'email': {

'required': 'Email is required.',

'emailDomain': 'Email domian should be hotmail.com or gmail.com',

},

'ConfirmEmail':

{

'required':'Confirm Email is required',

},

'phone': {

'required': 'phone is required.',

},

'skillName': {

'required': 'Skill Name is required.',

},

'experienceOfYear': {

'required': 'Experience is required.',

},

'proffeicency': {

'required': 'Proficiency is required.',

},

};

SetSaveButtonState(){

if(this.BtnSaveStatus){let x ={Showutton : true};return x;}

else{let x ={HideButton : true};return x;}}

SetUpdateButtonState(){

if(this.BtnUpdateStatus){let x ={Showutton : true};return x;}

else{let x ={HideButton : true};return x;}}

ngOnInit() {

//At the page load we reset the employee object to null and then we get the parameter id to check it if contain value or not to detect which mode it will be

this.empService.employee = null;

this.\_route.paramMap.subscribe(params => {

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

//If the employee id is not null then the page is in edit mode

if (this.empId > 0) {

//we initialize the form group with empty form array that will be filled by

this.employeeForm = this.fb.group({

fullName: ['', Validators.compose([Validators.required, Validators.minLength(5)])],

emailGroup: this.fb.group({

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

ConfirmEmail: ['', [Validators.required]]

}, { validator: matchEmails }),

contactPreference: ['email'],

phone: [''],

skills: this.fb.array([])

});

//the below method it’s used to dynamically create form group and then fill them //all to allow the user to edit them

this.getEmployee(this.empId);

this.BtnUpdateStatus = true;

this.BtnSaveStatus = false;

// this.SetButtonState();}

else {

//We set the page in ADD Mode with initialize the form array with single form //group only

this.employeeForm = this.fb.group({

fullName: ['', Validators.compose([Validators.required, Validators.minLength(5)])],

emailGroup: this.fb.group({

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

ConfirmEmail: ['', [Validators.required]]

}, { validator: matchEmails }),

contactPreference: [''],

phone: [''],

skills: this.fb.array([

this.addSkillFormGroup()])});

this.BtnUpdateStatus = false;

this.BtnSaveStatus = true;}});

//we used value Changes event when the user change every character it will check //the entire form group validation

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

this.logControlsInsideFormGroup(this.employeeForm);});}

getEmployee(id: Number) {

this.empService.getEmployee(id)

.subscribe(

(res: IEmployee) => this.LoadEmployee(res),

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

//In the below method we make dynamically create form group in the form array //based on the Skills count and then fill it dynamically

LoadEmployee(emp:IEmployee){

for(const item of emp.Skills){

(<FormArray>this.employeeForm.get('skills')).push(this.addSkillFormGroup());}

this.empService.employee = emp;

this.employeeForm.patchValue({

fullName: emp.fullName,

contactPreference: emp.contractPreference,

emailGroup: {

email: emp.email,

confirmEmail: emp.email},

phone: emp.phone,

skills:emp.Skills

});}

//The Following method it will call the service method to call the web api //service method to save employee

SaveEmployee(): void {

this.mapFormValuesToEmployeeModel();

this.empService.addEmployee(this.empService.employee).subscribe(

() => this.router.navigate(['list']),

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

//The Following method it will call the service method to call the web api //service method to update employee

UpdateEmployee(): void {

this.mapFormValuesToEmployeeModel();

this.empService.updateEmployee(this.empService.employee).subscribe(

() => this.router.navigate(['list']),

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

//In the below method we check if the employee object is not null or not if it’s //contain then push it inside the Skills property in the employee object

mapFormValuesToEmployeeModel() {

if(this.empService.employee != null){

this.empService.employee.fullName = this.employeeForm.value.fullName;

this.empService.employee.contractPreference = this.employeeForm.value.contactPreference;

this.empService.employee.email = this.employeeForm.value.emailGroup.email;

this.empService.employee.confirmEmail = this.employeeForm.value.emailGroup.email;

this.empService.employee.phone = this.employeeForm.value.phone;

this.empService.employee.Skills = new Array<ISkill>();

//In the below method we get the skills in the skills form array variable and //then push it inside the employee object

let skills = <FormArray>this.employeeForm.controls.skills;

let i:number =0;

for(const ctrl of skills.controls){

const skill = new ISkill();

skill.skillName = ctrl.get('skillName').value;

skill.experienceOfYear = ctrl.get('experienceOfYear').value;

skill.proffeicency = ctrl.get('proffeicency').value;

this.empService.employee.Skills.push(skill);}}

//When the user in the save mode then it will create new instance of employee and

//filling the array of skills inside it

else{

const emp = new IEmployee();

emp.fullName = this.employeeForm.value.fullName;

emp.contractPreference = this.employeeForm.value.contactPreference;

emp.email = this.employeeForm.value.emailGroup.email;

emp.confirmEmail = this.employeeForm.value.emailGroup.email;

emp.phone = this.employeeForm.value.phone;

emp.Skills = new Array<ISkill>();

let skills = <FormArray>this.employeeForm.controls.skills;

let i:number =0;

for(const ctrl of skills.controls){

const skill = new ISkill();

skill.skillName = ctrl.get('skillName').value;

skill.experienceOfYear = ctrl.get('experienceOfYear').value;

skill.proffeicency = ctrl.get('proffeicency').value;

emp.Skills.push(skill);}

this.empService.employee = emp;}}

addSkillFormGroup(): FormGroup {

return this.fb.group({

id:[''],

skillName: ['', Validators.required],

experienceOfYear: ['', Validators.required],

proffeicency: ['',Validators.required],

empId:['']

});}

//In the below method we get the skills and disable the previous form group and //then add new form group inside it

addSkillButtonClick(): void {

this.Arry = this.employeeForm.get('skills') as FormArray;

this.DisableControlsInFormGroup(this.Arry);

(<FormArray>this.employeeForm.get('skills')).push(this.addSkillFormGroup());}

DisableControlsInFormGroup(frmArry:FormArray){

for(const item of frmArry.controls){

item.get('skillName').markAsTouched();

item.get('skillName').disable();

item.get('experienceOfYear').markAsTouched();

item.get('experienceOfYear').disable();

item.get('proffeicency').markAsTouched();

item.get('proffeicency').disable();}}

removeSkillButtonClick(skillGroupIndex: number,skillID:number): void {

if(skillID > 0){

this.empService.deleteSkill(skillID).subscribe(

() => (<FormArray>this.employeeForm.get('skills')).removeAt(skillGroupIndex),

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

else{

(<FormArray>this.employeeForm.get('skills')).removeAt(skillGroupIndex);}}

//In the below method we make looping of each control in the main form group to get the validation error and store it in the validation Error Message variable

logControlsInsideFormGroup(group: FormGroup = this.employeeForm): void {

Object.keys(group.controls).forEach((key: string) => {

const abstractControl = group.get(key);

this.formErrors[key] = '';

// Loop through nested form groups and form controls to check

// for validation errors. For the form groups and form controls

// that have failed validation, retrieve the corresponding

// validation message from validationMessages object and store

// it in the formErrors object. The UI binds to the formErrors

// object properties to display the validation errors.

if (abstractControl && !abstractControl.valid

&& (abstractControl.touched || abstractControl.dirty || abstractControl.value !== '')) {

const messages = this.validationMessages[key];

for (const errorKey in abstractControl.errors) {

if (errorKey) {

this.formErrors[key] += messages[errorKey] + ' ';}}}

if (abstractControl instanceof FormGroup) {

this.logControlsInsideFormGroup(abstractControl);}

if (abstractControl instanceof FormArray) {

for (const control of abstractControl.controls) {

if (control instanceof FormGroup) {

this.logControlsInsideFormGroup(control);}}}});}

//on the below method we set the phone form control required or not based on the value passed on the contract preference form control

onContactPrefernceChange(option:string){

const ctrl = this.employeeForm.get('phone');

if(option === 'phone'){

ctrl.setValidators(Validators.required);}

else{

ctrl.clearValidators();}

ctrl.updateValueAndValidity();}}

**Web API Part && SQL Server Part**

**1-In the SQL Server we create the following 2 table**

**Create table Employee(**

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

**fullName nvarchar(50) not null,**

**email nvarchar(50) not null,**

**confirmEmail nvarchar(50) not null,**

**contractPreference nvarchar(50) not null,**

**phone nvarchar(50))**

**Create table Skills(**

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

**skillName nvarchar(50) not null,**

**experienceOfYear nvarchar(50) not null,**

**proffeicency nvarchar(50) not null,**

**empId int reference Employee(id))**

**2-In the WebAPIConfig.cs we update the following code**

**public static void Register(HttpConfiguration config){**

**// Web API configuration and services**

**// Configure Web API to use only bearer token authentication.**

**EnableCorsAttribute cors = new EnableCorsAttribute("\*", "\*", "\*");**

**config.EnableCors();**

**config.SuppressDefaultHostAuthentication();**

**config.Filters.Add(new HostAuthenticationFilter(OAuthDefaults.AuthenticationType));**

**// Web API routes**

**config.MapHttpAttributeRoutes();**

**config.Routes.MapHttpRoute(**

**name: "DefaultApi",**

**routeTemplate: "api/{controller}/{action}/{id}",**

**defaults: new { id = RouteParameter.Optional });**

**//In the below 2 code we remove xml response format (set only Jason format) and ignore the reference looping**

**config.Formatters.Remove(config.Formatters.XmlFormatter);**

**GlobalConfiguration.Configuration.Formatters.JsonFormatter.SerializerSettings.Re‌​ferenceLoopHandling = ReferenceLoopHandling.Ignore;}**

**3-In the EmployeeController.cs we write the following code**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Web;**

**using System.Web.Http.Cors;**

**using CompanyAPI.Models;**

**using System.Web.Http;**

**using System.Net.Http;**

**using System.Net;**

**using System.Data.Entity;**

**namespace CompanyAPI.Controllers{**

**[EnableCorsAttribute("\*", "\*", "\*")]**

**public class EmployeeController : ApiController{**

**[HttpGet]**

**public HttpResponseMessage LoadEmployees(){**

**using (CompanyEntities db = new CompanyEntities()){**

**var entity = from emp in db.Employees**

**//join skill in db.Skills on emp.id equals skill.empId**

**select new{**

**emp.id,**

**emp.fullName,**

**emp.email,**

**emp.confirmEmail,**

**emp.contractPreference,**

**emp.phone,**

**emp.Skills};**

**if (entity != null){**

**return Request.CreateResponse(HttpStatusCode.OK, entity.ToList());}**

**else{**

**return Request.CreateErrorResponse(HttpStatusCode.NotFound,**

**"Employees not found");}}}**

**[HttpGet]**

**// GET: Employee**

**public HttpResponseMessage LoadEmployeeById([FromUri] int EmpID){**

**using (CompanyEntities db = new CompanyEntities()){**

**var entity = from emp in db.Employees**

**join skill in db.Skills on emp.id equals skill.empId into ps**

**from p in ps.DefaultIfEmpty()**

**where emp.id == EmpID**

**select new{**

**emp.id,**

**emp.fullName,**

**emp.email,**

**emp.confirmEmail,**

**emp.contractPreference,**

**emp.phone,**

**emp.Skills};**

**if (entity != null){**

**return Request.CreateResponse(HttpStatusCode.OK, entity.FirstOrDefault());}**

**else{**

**return Request.CreateErrorResponse(HttpStatusCode.NotFound,**

**"Employee with Id " + EmpID.ToString() + " not found");}}}**

**//In the below web method at first we make load the old and new skills and passing to //new skills values to be in replace of the old skills and then update them**

**[HttpPut]**

**public HttpResponseMessage PutEmployee(int id, Employee employee){**

**try{**

**using (CompanyEntities db = new CompanyEntities()){**

**var emp = db.Employees.Include("Skills").FirstOrDefault();**

**var oldlst = from Skills in db.Skills where Skills.empId == emp.id select Skills;**

**var newlst = employee.Skills;**

**List<Skill> oldSkills = oldlst.ToList();**

**List<Skill> newSkills = newlst.ToList();**

**if (emp == null){**

**return Request.CreateResponse(HttpStatusCode.NotFound, "Employee With Id " + id.ToString() + " not found");}**

**else{**

**int i = 0;**

**int count = newSkills.Count-1;**

**foreach (var NewSkill in newSkills){**

**if (oldSkills.Count > 0 && oldSkills.Count > i){**

**oldSkills[i].skillName = NewSkill.skillName;**

**oldSkills[i].experienceOfYear = NewSkill.experienceOfYear;**

**oldSkills[i].proffeicency = NewSkill.proffeicency;**

**db.Entry(oldSkills[i]).State = EntityState.Modified;**

**db.SaveChanges();}**

**else{**

**Skill sk = new Skill();**

**sk.skillName = NewSkill.skillName;**

**sk.experienceOfYear = NewSkill.experienceOfYear;**

**sk.proffeicency = NewSkill.proffeicency;**

**sk.empId = id;**

**db.Skills.Add(sk);**

**db.SaveChanges();}**

**i++;}**

**emp.id = employee.id;**

**emp.fullName = employee.fullName;**

**emp.email = employee.email;**

**emp.confirmEmail = employee.confirmEmail;**

**emp.contractPreference = employee.contractPreference;**

**emp.phone = employee.phone;**

**// emp.Skills = employee.Skills;**

**db.Entry(emp).State = EntityState.Modified;**

**db.SaveChanges();**

**return Request.CreateResponse(HttpStatusCode.OK, "The Employee with Old Name " + employee.fullName + " Is Updated to New Name " + emp.fullName);}}}**

**catch (Exception ex){**

**return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex.Message);}}**

**[HttpPost]**

**public HttpResponseMessage PostEmployee([FromBody]Employee employee){**

**try{**

**using (CompanyEntities db = new CompanyEntities()){**

**db.Employees.Add(employee);**

**db.SaveChanges();**

**return Request.CreateResponse(HttpStatusCode.OK, "The Employee with Full Name "+ employee.fullName + "is Created");}}**

**catch (Exception ex){**

**return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex.Message);}}**

**//In the below method service at first we make looping of skills and delete each one of //them and then delete the employee object**

**[HttpDelete]**

**public HttpResponseMessage DeleteEmployee(int empID){**

**try{**

**using (CompanyEntities db = new CompanyEntities()){**

**Employee employee = db.Employees.Include("Skills").Where(x => x.id == empID).FirstOrDefault();**

**var oldLst = employee.Skills;**

**List<Skill> odlSkills = oldLst.ToList();**

**int i = 0;**

**foreach (var skill in odlSkills){**

**db.Skills.Remove(skill);**

**db.SaveChanges();**

**i++;}**

**if (employee == null){**

**return Request.CreateErrorResponse(HttpStatusCode.NotFound, "The Employee with id " + employee.id.ToString() + " not found");}**

**else{**

**db.Employees.Remove(employee);**

**db.SaveChanges();**

**return Request.CreateResponse(HttpStatusCode.OK, "The Employee with id " + employee.id.ToString() + " is Deleted");}}}**

**catch (Exception ex){**

**return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex.Message);}}**

**[HttpDelete]**

**public HttpResponseMessage DeleteSkill(int skillID){**

**try{**

**using (CompanyEntities db = new CompanyEntities()){**

**Skill skill = db.Skills.Where(x => x.id == skillID).FirstOrDefault();**

**if (skill == null){**

**return Request.CreateErrorResponse(HttpStatusCode.NotFound, "The Skill with id " + skillID.ToString() + " not found");}**

**else{**

**db.Skills.Remove(skill);**

**db.SaveChanges();**

**return Request.CreateResponse(HttpStatusCode.OK, "The Skill with id " + skillID.ToString() + " is Deleted");}}}**

**catch (Exception ex){**

**return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex.Message);}}}}**