**Lesson18 Angular reactive forms cross field validation (Email and Confirm Email)**

**Notes:-**

**When we means cross field validation we mean that there are form control depend on another form control 🡺 we do the following steps:-**

**1-we must to create inner form group to contain the 2 form control**

**2-we have to create custom validator to compare between 2 form controls**

**3-we have to declare error message for the form group that define that the email and confirm email values does not match**

**Note:-**

**1-we have to know that the built in validator such as required have the same name when define value for required validation message**

validationMessages = {

'fullName': {

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

'minlength': 'Full Name must be greater than 2 characters',

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

**2-while in custom validator, we can rename the return type as the same of the name of the custom validator or in different name**

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) {return null;}

//This is the name of the return with key called matchEmailsMsg

else {return { 'matchEmailsMsg': true };}}

//we define the emailGroup validation binding to apply the custom validation with it

formErrors = {

'fullName': '',

'emailGroup':'',

'email': '',

'confirmEmail':''};

// Include required error message for phone form control

validationMessages = {

'fullName': {

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

'minlength': 'Full Name must be greater than 2 characters',

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

},

'emailGroup':{

//This is the return name that used when applying custom validator

'matchEmailsMsg': 'email and confirm email must match',

},

'email': {

'required': 'Email is required.',

'emailDomain': 'Email domian should be pragitmatch.com',},

'confirmEmail': {

'required': 'Confirm Email is required.',

}};

ngOnInit() {

// Include FormControls for contactPreference, email & phone

// contactPreference has email as the default value

this.employeeForm = this.fb.group({

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

contactPreference: ['email'],

//we apply the custom validator function

emailGroup: this.fb.group({

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

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

}, { validator: matchEmails }),

phone: [''],

skills: this.fb.group({

skillName: ['', Validators.required],

experienceInYears: ['', Validators.required],

proficiency: ['', Validators.required]

})});

**Steps:-**

**1-In the employees-comp.component.html we add the following**

<form class="form-horizontal" [formGroup]="employeeForm" (ngSubmit)="onSubmit()">

<div class="panel panel-primary"><div class="panel-heading">

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

<div class="panel-body">

<div class="form-group">

<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)="logValidationErrors()">

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

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

<div class="form-group">

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

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

<label class="radio-inline">

<input type="radio" value="email" formControlName="contactPreference">Email</label>

<label class="radio-inline">

<input type="radio" value="phone"formControlName="contactPreference">Phone

</label></div></div>

//we create the inner form group called emailGroup to carry the validation designed for the email and confirm email equal

<div formGroupName="emailGroup">

<!-- Email input element -->

<div class="form-group">

<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)="logValidationErrors()">

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

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

<div class="form-group">

<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)="logValidationErrors()">

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

{{formErrors.confirmEmail ? formErrors.confirmEmail : formErrors.emailGroup}}

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

</div>

<!-- Phone input element -->

<div class="form-group">

<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)="logValidationErrors()">

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

{{formErrors.phone}}

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

<div class="panel-footer">

<button class="btn btn-primary" type="submit">Save</button>

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

</div>

</div>

</form>

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

// Import FormGroup and FormControl classes

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

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

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

@Component({

selector: 'app-employees-comp',

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

styleUrls: ['./employees-comp.component.css']

})

export class EmployeesCompComponent implements OnInit {

// Include phone property

formErrors = {

'fullName': '',

//we declare the email Group to carry the validation message

'emailGroup':'',

'email': '',

'confirmEmail':'',

'phone': '',

'skillName': '',

'experienceInYears': '',

'proficiency': ''

};

// Include required error message for phone form control

validationMessages = {

'fullName': {

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

'minlength': 'Full Name must be greater than 2 characters',

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

//This is the validation message of emailGroup

'emailGroup':{

'emailMismatch': 'email and confirm email must match'},

'email': {

'required': 'Email is required.',

'emailDomain': 'Email domian should be pragitmatch.com'},

'confirmEmail': {

'required': 'Confirm Email is required.'},

'phone': {

'required': 'Phone is required.'},

'skillName': {

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

'experienceInYears': {

'required': 'Experience is required.'},

'proficiency': {

'required': 'Proficiency is required.'}};

employeeForm:FormGroup;

constructor(private fb:FormBuilder){}

ngOnInit() {

// Include FormControls for contactPreference, email & phone

// contactPreference has email as the default value

this.employeeForm = this.fb.group({

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

contactPreference: ['email'],

//we see that we apply the validation of the matchEmails on the level of //the form Group

emailGroup: this.fb.group({

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

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

}, { validator: matchEmails }),

phone: [''],

skills: this.fb.group({

skillName: ['', Validators.required],

experienceInYears: ['', Validators.required],

proficiency: ['', Validators.required]

}),});

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

this.logValidationErrors(this.employeeForm);

});

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

this.onContactPrefernceChange(data);})}

// If the Selected Radio Button value is "phone", then add the

// required validator function otherwise remove it

onContactPrefernceChange(frmGroup:string){

debugger;

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

if(frmGroup === 'phone'){

obj.setValidators(Validators.required);}

else{

obj.clearValidators();}

obj.updateValueAndValidity();}

logValidationErrors(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)) {

const messages = this.validationMessages[key];

for (const errorKey in abstractControl.errors) {

if (errorKey) {

debugger;

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

}}}

if (abstractControl instanceof FormGroup) {

this.logValidationErrors(abstractControl);}});}}

**3-on the emailValidation.ts we add the new two functions**

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

export {emailDomain,matchEmails}

export class EmailValidators {}

function emailDomain(domainName: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){

return null;}

else{

debugger;

return {'emailDomain':true};}}}

//we define custom validator that compare between the emailcontrol and confirmEmailControl

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) {

return null;

}

else {return { 'emailMismatch': true };}}