Angular2 form validation :

Template driven approach : We simplt setup our form in the template in html code.

vs reactive approach : In this we actually define the structure of our form in typescript code. More control over it.

**Template driven approach**

Make sure that in the app.module.ts , we do have the FormsModule from ‘’. It will create the javascript representation of the form.

Accessing the form with ViewChild.

@ViewChild signupForm : NgForm;

**Disable button when the form is not valid :**

<button [disabled]=”!f.valid”>Save Record</button>

Apply boredr color for ng-valid and ng-invalid state’s.

input.ng-invalid.ng-touched{

border : 1px solid red;

}

input.ng-dirty.ng-valid {

border : 1px solid green;

}

**Show the error message for the field :**

<p>

Email : <input name="email" id="email" ngModel required #email="ngModel">

<span \*ngIf="!email.valid && email.touched">Please enter a valid email!</span>

</p>

**Set default value for the field**

[ngModel]=’email’

Inside the component initialize the :

email=[abc@gmail.com](mailto:abc@gmail.com)

**Set 2 way data-binding**

[(ngModel)]=”comment”;

Use the expression tag to display the result.

**Grouping the fields:**

<div ngModelGroup=”userData”>

Usernamer : ..

Password : ..

</div>

**Radio button :**

Inside the component define the array of gender :

genders = ['male','female'];

Inside the view :

Gender :

<div \*ngFor="let gender of genders">

<label>

<input type="radio" name="gender" [value]="gender" required> {{gender}}

</label>

</div>

**Using form data :**

addRecord(form : NgForm){

console.log(this.signUp.value.userData.username);

//alert('add record');

}

**Resetting form :**

This.signUp.reset();

Some good examples :

* <form novalidate>
* <input type="text" name="name" ngModel required>
* <input type="text" name="street" ngModel minlength="3">
* <input type="text" name="city" ngModel maxlength="10">
* <input type="text" name="zip" ngModel pattern="[A-Za-z]{5}">
* </form>

Note : ngOnInit() is an interface

**Reactive approach : In reactive approach the form is created programatically**

Reactive setup :

We will create a property inside the component class that will actually hold our form there.

Import {FormGroup} from ‘@angular/forms’;

Class AbcComponent implements OnInit{

signupForm : FormGroup;

ngOnInit(){

signupForm = new FormGroup();

}

}

Inside the app.module.ts

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

imports : [ReactiveFormsModule]

**Now , to the component**

**Note :** Validators.minLength(5), Validators.maxLength(10)

Note 2 :

* this.form = new FormGroup({
* name: new FormControl('', Validators.required)),
* street: new FormControl('', Validators.minLength(3)),
* city: new FormControl('', Validators.maxLength(10)),
* zip: new FormControl('', Validators.pattern('[A-Za-z]{5}'))
* });

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

export class FormValidationReactiveComponent implements OnInit {

genders = ['male','female'];

constructor() { }

signUp : FormGroup;

ngOnInit() {

this.signUp = new FormGroup({

'username' : new FormControl(null),//First value is the field value , second parameter can specify some of the validator

'email' : new FormControl(null),

'gender' : new FormControl('male')

});

}

addRecord(){

console.log(this.signUp);

}

}

**Inside the template**

<form [formGroup]="signUp">

<p>

Username : <input name="username" id="username" formControlName="username">

</p>

<p>

Email : <input name="email" id="email" formControlName="email">

</p>

<p>

Gender :

<div \*ngFor="let gender of genders">

<label>

<input type="radio" formControlName="gender" name="gender" [value]="gender" required> {{gender}}

</label>

</div>

<p>

<button (click)="addRecord()">Add</button>

</p>

</form>

**Applying the validation : Fill the second argument of the FormControl**

this.signUp = new FormGroup({

'username' : new FormControl(null,Validators.required),//First value is the field value

'email' : new FormControl(null,[Validators.required,Validators.email]),

'gender' : new FormControl('male')

});

Now dive towards the form :

<form [formGroup]="signUp">

<p>

Username : <input id="username" formControlName="username">

<span \*ngIf="!signUp.get('username').valid && signUp.get('username').touched">Please enter the username</span>

<br>

Email : <input id="email" formControlName="email">

<span \*ngIf="!signUp.get('email').valid && signUp.get('email').touched">Please enter the email</span>

</p>

Gender :

<div \*ngFor="let gender of genders">

<label>

<input type="radio" formControlName="gender" name="gender" [value]="gender" required> {{gender}}

</label>

</div>

<p>

<button [disabled]="!signUp.valid" (click)="addRecord()">Add</button>

</p>

</form>

**Array of form controls :**

a) this.signUp = new FormGroup({

'username' : new FormControl(null,Validators.required),//First value is the field value

'email' : new FormControl(null,[Validators.required]),

'gender' : new FormControl('male'),

'hobbies' : new FormArray([])

});

b)

addHobby(){

const control = new FormControl(null,Validators.required);

(<FormArray>this.signUp.get('hobbies')).push(control)

}

c)Inside the form

<div formArrayName="hobbies">

Your Hobbies

<button (click)="addHobby()">Add Hobby</button>

<div \*ngFor="let x of signUp.get('hobbies').controls; let i = index">

<input type="text" [formControlName]="i">

</div>

</div>

**Adding own validators :**

**Using error codes :**

<span \*ngIf=”signUp.get(‘username’).errors[‘required’]”>..</span>

<span \*ngIf=”signUp.get(‘username’).errors[‘email]”>..</span>