**FORMULARIOS REACTIVOS**

Para hacer uso de formularios reactivos se deben agregar en app.module.ts:

* import{ReactiveFormsModule} from '@angular/forms'
* Y en la sección de imports:

Imports: [ReactiveFormsModule]

**FORM CONTROL**

|  |
| --- |
| import{FormControl, Validators} from '@angular/forms' |
| import { debounceTime, debounce } from 'rxjs/operators'; |
| @Component({ |
| selector: 'app-product-form', |
| templateUrl: './product-form.component.html', |
| styleUrls: ['./product-form.component.css'] |
| }) |
| export class ProductFormComponent implements OnInit { |
| emailCtrl= new FormControl('',[Validators.required, Validators.maxLength(20), Validators.minLength(11)]); |
| constructor() { |
| this.emailCtrl.valueChanges.pipe( |
| debounceTime(350) |
| ).subscribe( |
| value=>{ |
| console.log(value); |
| } |
| ) |
| } |
| ngOnInit() { |
| } |
| getEmail(event:Event) |
| { |
| event.preventDefault(); |
| console.log(this.emailCtrl.value); |
| } |
| } |

Cuando se crea un form control, se debe especificar en un array 3 elementos,

emailCtrl= new FormControl('',[Validators.required, Validators.maxLength(20), Validators.minLength(11)]);

Los dos últimos son las validaciones Síncronas y Validaciones Asíncronas, respectivamente.

Para esta sección vamos a utilizar las validaciones síncronas.

* Validators.required Esta valida que esté lleno el campo.
* Validators.maxLength(20) Valida que el tamaño máximo del campo sea 20.
* Validators.minLength(11) Valida que el tamaño mínimo sea 11.

Una vez especificado en el archivo de Typescript, cuales van a ser los validadores, se procede a formarlo en el HTML

|  |
| --- |
| <p>  Email: <br/> |
| <input type="email" [formControl]="emailCtrl" > |
| </p> |
| <p \*ngIf="emailCtrl.errors"> |
| <small \*ngIf="emailCtrl.hasError('required') && emailCtrl.dirty">Este campo es requerido</small> |
| <small \*ngIf="emailCtrl.hasError('maxlength') && emailCtrl.dirty">Este campo se pasa de 20 caracteres</small> |
| <small \*ngIf="emailCtrl.hasError('minlength') && emailCtrl.dirty">Este campo tiene muy pocos caracteres</small> |
| </p> |

Como se puede observar, en el input se enlaza al formcControl, por medio de [formControl]=”emailCtrl”, y en la parte de abajo se verifica cual de las condiciones no cumple.

Para esto primero se verifica si tiene errores, sea cual sea por medio de emailCtrl.errors. Después de esto ya se verifica cual es el error que presenta:

\*ngIf="emailCtrl.hasError('required')

hasError('maxlength')

hasError('minlength')

**FORM GROUPS**

**Los formGroups es un conjunto de formControls, de esta manera se monitores un conjunto y no uno a uno, aunque también podría hacerse de forma explicita a un control miembro de ese group.**

**//Se inicializa:**

form: FormGroup;

**Podemos inicializar cada formControl, que es miembro del group:**

*this.form= new FormGroup({*

*name:new FormControl('',[Validators.required]),*

*date:new FormControl('',[Validators.required]),*

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

*text:new FormControl('',[Validators.maxLength(200)]),*

*category:new FormControl('',[Validators.required]),*

*gender:new FormControl('',[Validators.required])*

*});*

**Pero hay una forma más corta y es agregar el servicio de este constructor de forms, haciendo inyección así:**

constructor(private FormBuilder:FormBuilder) {

*this*.buildForm();

  }

private buildForm() {

*this*.form= *this*.FormBuilder.group({

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

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

      email:['',[Validators.email]],

      text:['',[Validators.maxLength(200)]],

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

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

    });

*this*.form.valueChanges.pipe(

      debounceTime(350)

      ).subscribe(

        value=>{

          console.log(value);

        }

      )

}

save(event:Event)

  {

    event.preventDefault(); *//se Cancela su comportamiento por defecto, evitando que se recargue la pagina*

    const value= *this*.form.value;

    console.log(value);

  }

**Especifico los validadores por cada formControl.**

**Ahora lo creo en el html**

<form [formGroup]="form" (ngSubmit)="save($event)">

    <p>

      Name: <br/>

      <input type="text" formControlName="name">

    </p>

    <p>

      Fecha: <br/>

      <input type="date" formControlName="date">

    </p>

    <p>

      Email: <br/>

      <input type="email" formControlName="email">

    </p>

    <p>

      Texto largo: ( {{form.get('text').value.length}} /200) <br/>

      <textarea cols="30" rows="10" formControlName="text"></textarea>

    </p>

    <div \*ngIf="form.get('text').errors && form.get('text').touched">

      <p \*ngIf="form.get('text').hasError('required')">

        Es un campo requierido

    </p>

    <p \*ngIf="form.get('text').hasError('maxlength')">

      Maximo tamaño 200 caracteres

    </p>

    </div>

    <p>

      Category: <br/>

      <select formControlName="category">

        <option value="1">Category 1</option>

        <option value="2">Category 2</option>

        <option value="3">Category 3</option>

      </select>

    </p>

    <p>

      Genero: <br/>

      <input type="radio" name="gender" formControlName="gender" value="male"> Male<br>

      <input type="radio" name="gender" formControlName="gender" value="female"> Female<br>

      <input type="radio" name="gender" formControlName="gender" value="other"> Other

    </p>

    <button type="submit" [disabled]="form.invalid" >Guardar</button>

  </form>

**Acceder al valor de un formControl de un formGroup:**

let text=*this*.form.controls['text'].value;

**Como vemos, se deshabilita el botón guardar, pero es confuso cuando el usuario no sabe por qué se da..  
para esto marcamos todos los botones como tocados, con el fin de que se muestren todos los errores en pantalla.**

  save(event:Event)

  {

    event.preventDefault(); *//se Cancela su comportamiento por defecto, evitando que se recargue la pagina*

    if(*this*.form.valid)

    {

      const value= *this*.form.value;

      console.log(value);

    }

    else *this*.form.markAllAsTouched();

    }

**Se recomienda hacer un getField para cada control, de tal manera que se pueda obtener este. Es buena práctica de programación.**

 getNameField()

    {

*this*.form.get('name');

    }

    getDateField()

    {

*this*.form.get('date');

    }

Existe un error cuando se coloca un botón distinto al guardar dentro del formulario, la acción que ejecuta el botón también ejecuta el guardar por defecto. Para solucionar esto, debemos aclarar que el botón es de **tipo button**, pues si se deja sin nada el lo coloca como tipo **submit**