18 - Formularios reactivos : validaciones estándares de Angular

Una funcionalidad muy importante de un formulario es la validación de las entradas de datos del usuario y la información que se le provee al mismo indicando los datos mal cargados.

Cuando utilizamos formularios reactivos la lógica de la validación se encuentra en la clase (archivo *.js) y no en la vista (archivo *.html)

El framework de Angular provee una clase llamada 'Validators' que contiene una serie de métodos estáticos para validar entradas de datos muy comunes como la estructura de un email, valores numéricos máximos y mínimos, cantidad mínima y máxima de caracteres etc.

La asignación de los métodos de validación se hace cuando se crea el objeto de la clase FormControl. Veremos con un ejemplo como configuramos distintos métodos de validación a los controles de un formulario.

Problema

Confeccionar un formulario de contacto que permita cargar el nombre, mail y un mensaje. Implementar las siguientes validaciones a los controles del formulario:

- Los tres controles no pueden quedar vacíos.
- El nombre debe tener como mínimo 10 caracteres.
- El email ingresado debe ser correcto.
- El mensaje no puede superar los 500 caracteres.

Cuando se presione un botón mostrar un mensaje indicando si todos los controles se encuentran correctamente cargados.

- Crearemos primero el proyecto:

```
ng new proyecto013
```

- Modificamos la vista de la componente que muestra el formulario reactivo y eventualmente los mensajes de error de entrada de datos (app.component.html):

```
<form [formGroup]="formularioContacto" (ngSubmit)="submit()">
  <p>Nombre:
    <input type="text" formControlName="nombre" required>
    @if(this.formularioContacto.get('nombre')?.errors?.['required'
]) {
    <span>(El nombre no puede quedar vacío)</span>
    }
    @if(this.formularioContacto.get('nombre')?.errors?.['minlength
']) {
    <span>(Debe tener como mínimo 10 caracteres)</span>
    }
  </p>
  <p>Mail:
    <input type="text" formControlName="mail">
    @if(this.formularioContacto.get('mail')?.errors?.['required'])
 {
    <span>(El mail no puede quedar vacío)</span>
    }
    @if(this.formularioContacto.get('mail')?.errors?.['email']) {
    <span>(El mail no es válido)</span>
    }
  </p>
  <p>Mensaje<br>
    <textarea rows="10" cols="70" formControlName="mensaje"></text
area>
  </p>
  @if(this.formularioContacto.get('mensaje')?.errors?.['required']
) {
  <div>(El mensaje no puede quedar vacío)</div>
  }
  @if(this.formularioContacto.get('mensaje')?.errors?.['maxlength'
]) {
  <div>(Debe tener como máximo 500 caracteres)</div>
  }
  <button type="submit">Confirmar</button>
</form>
<div>{{resultado}}</div>
<router-outlet />
```

Analizaremos este archivo en conjunto luego de presentar 'app.component.ts'

- La clase asociada a la vista es (app.component.ts):

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { ReactiveFormsModule, FormControl, FormGroup, Validators }
 from '@angular/forms';

@Component({
  selector: 'app-root',
  imports: [RouterOutlet, ReactiveFormsModule],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  resultado='';

  formularioContacto = new FormGroup({
    nombre: new FormControl('', [Validators.required, Validators.m
inLength(10)]),
    mail: new FormControl('', [Validators.required, Validators.ema
il]),
    mensaje: new FormControl('', [Validators.required, Validators.
maxLength(500)])
  });

  submit() {
    if (this.formularioContacto.valid)
      this.resultado = "Todos los datos son válidos";
    else
      this.resultado = "Hay datos inválidos en el formulario";
  }
}
```

Lo primero que debemos hacer es importar la clase 'Validators' y las otras que ya conocemos:

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

Cuando creamos los objetos de la clase FormControl debemos pasar como segundo parámetro del constructor los nombres de métodos que se le aplicarán a cada control de formulario:

```
  formularioContacto = new FormGroup({
    nombre: new FormControl('', [Validators.required, Validators.minLength(10)]),
    mail: new FormControl('', [Validators.required, Validators.email]),
    mensaje: new FormControl('', [Validators.required, Validators.maxLength(500)])
  });
```

Por ejemplo para el 'nombre' configuramos que se apliquen las validaciones: [Validators.required, Validators.minLength(10)], luego en la vista podemos verificar que validaciones se cumplen y cuales no:

```
<p>Nombre:
  <input type="text" formControlName="nombre" required>
  @if(this.formularioContacto.get('nombre')?.errors?.['required']) {
  <span>(El nombre no puede quedar vacío)</span>
  }
  @if(this.formularioContacto.get('nombre')?.errors?.['minlength']) {
  <span>(Debe tener como mínimo 10 caracteres)</span>
  }
</p>
```

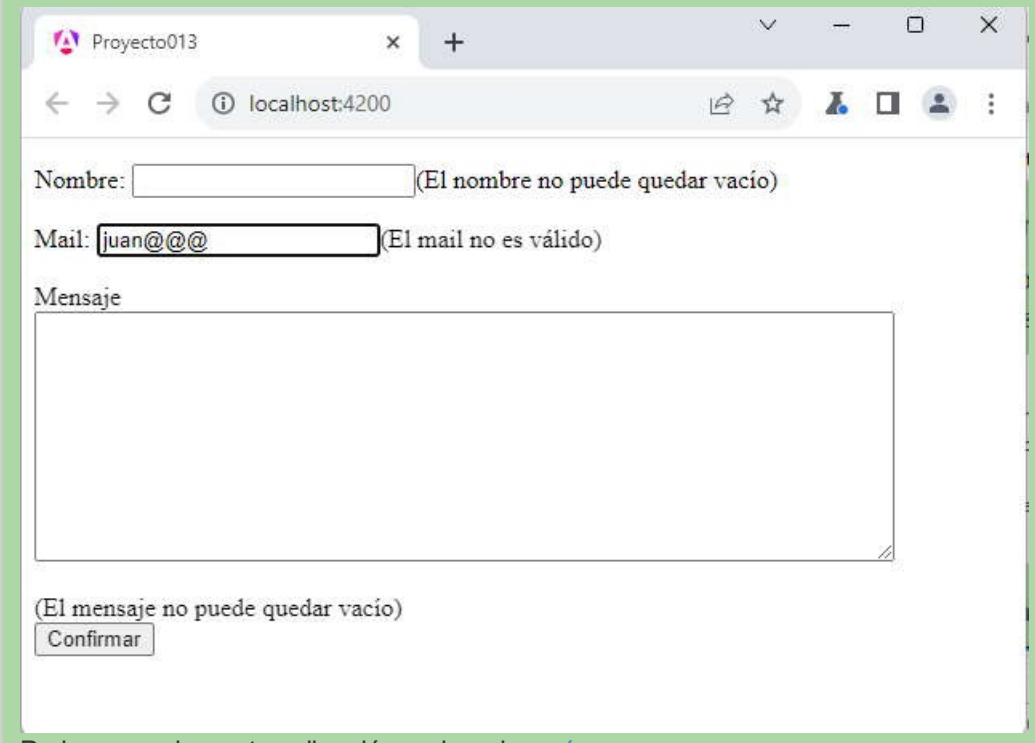
Mediante comandos @if validamos si se debe mostrar el mensaje de error dependiendo si el atributo this.formularioContacto.get('nombre')?.errors? existe o no.

De forma similar se hacen las validaciones para los otros controles.

Cuando se presiona el botón analizamos la propiedad 'valid' del objeto FormGroup, si almacena true significa que todos los controles de validación del formulario se verifican correctos:

```
submit() {
  if (this.formularioContacto.valid)
    this.resultado = "Todos los datos son válidos";
  else
    this.resultado = "Hay datos inválidos en el formulario";
}
```

Si ejecutamos la aplicación tenemos una interfaz similar a:



Podemos probar esta aplicación en la web aquí.

Acotaciones

Para conocer todos los métodos de validación que nos provee Angular podemos visitar su documentación oficial y analizar la clase Validators.

Retornar