Angular S3: More Forms

Después de esta lección podrás:

1. Entender los formularios reactivos en angular.
2. Crear Formularios con validaciones en el front.

Angular permite la creación de formularios de dos tipos diferentes, mediante ***templates*** o de ***forma reactiva***, la que os vamos a enseñar. Crearemos estructuras con las que Angular creará los formularios, manteniendo la lógica de nuestra aplicación web en una sola parte, haciendo que el código sea más fácil de manejar y de mantener.

Formularios reactivos

Para usarlos tenemos que declararlos en el `@NgModule`:

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

@NgModule({
  imports: [
    ...,
    ReactiveFormsModule
  ],
```

```
  declarations: [...],
  bootstrap: [...]
})
export class AppModule {}
```

Antes de continuar es importante definir qué es un **FormControl** y un **FormGroup**.

FormControl es un objeto qué se usa en los formularios para tener un control sobre su valor y su estado en el formulario. Para usarlo:

```
const ctrl = new FormControl('some value');
console.log(ctrl.value); // 'some value'
```

FormGroup es un cojunto de FormControls, el estado de este objeto depende del estado de todos sus objetos, es decir, si uno de los FormControl es inválido, el grupo entero es inválido. Para usarlo:

```
const avengersForm = new FormGroup({
  name: new FormControl('Steve Rogers', Validators.minLength(2)),
  power: new FormControl('Super Soldier'),
});
```

¿Y cómo podemos conectar estos objetos con los formularios del HTML? Es bastante sencillo, gracias al `formControlName` que usaremos para los formControl y la etiqueta `[formGroup]` que usaremos para enganchar nuestro `formGroup` declarado en el TS, por ejemplo:

```
<form novalidate [formGroup]="avengersForm">
  Name: <input type="text" formControlName="name">
  Power: <input type="text" formControlName="power">
</form>
```

En el componente creamos un formGroup con el nombre **avengersForm** con dos FormControl, name y power.

Para entender todo esto mejor vamos a usar un ejemplo real.

Ejemplo

Imaginemos que queremos crear un formulario de registro, para ello creamos la siguiente interfaz:

```
export interface User {
  name: string;
  password: string;
  passwordRepeat: string,
}
```

Ahora en nuestro componente, si no tenemos uno lo creamos, declaramos un `FormGroup` con `FormControl` para cada uno de los campos:

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup } from '@angular/forms';

@Component({...})
export class SignupFormComponent implements OnInit {
  user: FormGroup;
  ngOnInit() {
    this.user = new FormGroup({
      name: new FormControl(''),
      password: new FormControl(''),
      passwordRepeat: new FormControl('')
    });
  }
}
```

Ahora creamos el formulario en el HTML del template y conectamos los campos como hemos hecho antes:

```
<form novalidate [formGroup]="user">
  <label>
    <span>Full name</span>
    <input
```

```
      type="text"
      placeholder="Your full name"
      formControlName="name">
  </label>
  <label>
      <span>Email address</span>
      <input
        type="password"
        placeholder="Your passwornd"
        formControlName="password">
    </label>
    <label>
      <span>Confirm address</span>
      <input
        type="password"
        placeholder="Confirm your password"
        formControlName="passwordRepeat">
    </label>
  <button type="submit">Sign up</button>
</form>
```

¿Y qué pasa con el botón de **submit**? Pues se enlaza mediante el evento `ngSubmit`:

```
<form novalidate (ngSubmit)="onSubmit()" [formGroup]="user">
...
</form>
```

Validadores

Lo primero es importar en el componente los **validadores**:

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

Para validar la información de los campos del formulario, podemos usar las funciones que nos ofrece Angular o implementar las nuestras propias, para ello cambiamos los objetos (para tener limpio el `ngOnInit` os aconsejo llevarlo a una función y que está sea llamada desde el `ngOnInit`):

```
2//Opción A - Directo en el OnInit

ngOnInit() {
  this.user = new FormGroup({
    name: new FormControl('', [Validators.required, Validators.minLength(2)]),
      password: new FormControl('', Validators.required),
      passwordRepeat: new FormControl('', Validators.required)
  });
}

//Opción B - Función auxiliar / esto ayuda a activar o desactivar formularios.

ngOnInit() {
  this.userFormActivate();
}

function userFormActivate() {
   this.user = new FormGroup({
    name: new FormControl('', [Validators.required, Validators.minLength(2)]),
    password: new FormControl('', Validators.required),
    passwordRepeat: new FormControl('', Validators.required)
  }, { validators: passwordMatchValidator });
}
```

Con `Validators.required` aseguramos que el dato exista y con `minLength` aseguramos que al menos el usuario introduzca dos caracteres. Como puedes ver, si queremos añadir más de un validador, lo podemos hacer mediante un array de validadores. Existen más validadores que podéis usar, para ello os aconsejamos que visitéis la documentación oficial de angular: **AngularValidators**

Si os fijáis, existe una función *custom*, de validación de contraseñas:

```
function passwordMatchValidator(g: FormGroup) {
   return g.get('password').value === g.get('passwordRepeat').value
      ? null : {'mismatch': true};
}
```

También podemos hacer que el botón de submit esté deshabilitado mientras que no esté todo validado:

```
<form novalidate (ngSubmit)="onSubmit()" [formGroup]="user">
  ...
  <button type="submit" [disabled]="user.invalid">Sign up</button>
</form>
```

Para mostrar, por ejemplo, mensajes de error personalizados:

```
<div
  class="error"
  *ngIf="user.get('name').hasError('required') && user.get('name').touched">
  Nombre requerido
</div>
```

FormBuilder

Ahora que entendemos bien estos conceptos, podemos que dejar que Angular añada su mágia para que cree el **FormGroup** y el **FormControl**.

Para ello vamos a hacer uso del **FormBuilder**, primero lo importamos junto con lo que vamos a necesitar:

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

A continuación lo inyectamos en el constructor:

```
constructor(private fb: FormBuilder) {}
```

Ahora refactorizamos el código para usar el FormBuilder:

```
ngOnInit() {
  this.user = this.fb.group({
    name: ['', [Validators.required, Validators.minLength(2)]],
    password: ['', Validators.required],
    passwordRepeat: ['', Validators.required]
  });
}
```

Como veis como no añade funcionalidad nueva, simplemente es una manera de dejar el código más limpio y escalable.

Propuesta Ejercicio:

Este ejercicio nos ayudará a comprender mejor los formularios y poder trabajar con soltura dentro de un proyecto de Angular, además reforzaremos conocimientos adquiridos hasta ahora.

Por ello os iremos detallando por puntos o iteraciones lo que esperamos de Formularios Reactivos:

Iteración 1

Crea un componente Student-form dentro del componente padre student-list

Iteración 2

Crea un formulario reactivo con sus validaciones.

Iteración 3

Envía la información obtenida del componente a student-list.

Iteración 4

Renderiza student-list cada vez que añades un nuevo student.

Iteración 5

Botón disable hasta que no se rellenen todos los campos.

Iteración 6

Añade mensajes de error dentro del formulario.

Bonus

Estilos al formularios usando BEM y Flex (Diseño a tu elección).