



# Angular S3: Reactive Forms with FormBuilder

**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 que se usa en los formularios para tener un control sobre su valor y su estado en el formulario.

**FormGroup** es un conjunto 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.

¿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.

## Ejemplo de un Formulario Reactivo

Lo primero crearemos una **interfaz** para nuestro formulario:

```
export interface UserRegister {
  name: string;
  password: string;
  passwordRepeat: string;
}
```

Ahora en nuestro **componente formulario**:

```
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormBuilder, Validators } from '@angular/forms';
import { UserRegister } from '../model/user-register-model';

@Component({...})
export class SignupFormComponent implements OnInit {
  // Inicialización del formulario
  public userRegisterForm: FormGroup = null;
  // variable submitted a false
  public submitted: boolean = false;

  // Inicializamos FormBuilder en el constructor
  constructor(private formBuilder: FormBuilder) {
    // Nuestro formulario - sin campos por defecto
    // Podemos meter valores por defecto en las comillas
    this.userRegisterForm = this.formBuilder.group({
      name: ['', [Validators.required, Validators.maxLength(20)]],
      password: ['', [Validators.required, Validators.maxLength(20)]],
      passwordRepeat: ['', [Validators.required, Validators.maxLength(20)]],
    });
  }

  // El OnInit -> Vacío
  ngOnInit() { /* Empty */ }

  //Función accionada al clickar en submit
  public onSubmit(): void {
    // El usuario ha pulsado en submit->cambia a true submitted
    this.submitted = true;
    // Si el formulario es valido
    if (this.userRegisterForm.valid) {
      // Creamos un Usuario y lo emitimos
      const user: UserRegister = {
        name: this.userRegisterForm.get('name').value,
        password: this.userRegisterForm.get('password').value,
        passwordRepeat: this.userRegisterForm.get('passwordRepeat').value,
      };
    }
  }
}
```

```

        console.log(user);
        // Reseteamos todos los campos y el indicador de envío o submitted
        this.userRegisterForm.reset();
        this.submitted = false;
    }
}
}

```

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

```

<h2>Formulario de Registro:</h2>
<!-- formGroup es el nombre de nuestro formFroup del .ts -->
<form novalidate (ngSubmit)="onSubmit()" [formGroup]="userRegisterForm">
  <!-- Fieldset del primer campo - name -->
  <fieldset>
    <label for="name">Nombre:</label>
    <input type="text" id="name" formControlName="name" />
    <!-- Si el campo name es invalido / ha sido tocado / submitted = true -->
    <div *ngIf="!userRegisterForm.get('name').valid &&
      (userRegisterForm.get('name').dirty || submitted)">
      Has introducido mal tu nombre
    </div>
  </fieldset>

  <!-- Fieldset del segundo campo - password -->
  <fieldset>
    <label for="password">Password:</label>
    <input type="password" id="password" formControlName="password" />
    <!-- Si el campo password es invalido / ha sido password / submitted = true -->
    <div *ngIf="!userRegisterForm.get('password').valid &&
      (userRegisterForm.get('password').dirty || submitted)">
      Has introducido mal tu contraseña
    </div>
  </fieldset>

  <!-- Fieldset del segundo campo - repeatPassword -->
  <fieldset>
    <label for="passwordRepeat">Repeat password:</label>
    <input type="password" id="passwordRepeat" formControlName="passwordRepeat" />
    <!-- Si el campo repeatPassword es invalido / ha sido password / submitted = true -->
    <div *ngIf="!userRegisterForm.get('passwordRepeat').valid &&
      (userRegisterForm.get('passwordRepeat').dirty || submitted)">
      Tu contraseña no coincide
    </div>
  </fieldset>
</form>

```

```
<!-- Botón deshabilitado hasta que el formulario sea valid -->
<button type="submit" [disabled]="!userRegisterForm.valid">Submit</button>

</form>
```

## Validadores por defecto de Angular

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

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

Para **validar** la información de los **campos del formulario**, podemos usar las **validación Angular** o implementar una **validación personalizada**:

### Validación de Angular:

```
// Ejemplo con validación por defecto de Angular
// Para ver todas las opciones ver la docu de angular.io

this.userRegisterForm = this.formBuilder.group({
  name: ['', [Validators.required, Validators.maxLength(20)]],
  password: ['', [Validators.required, Validators.maxLength(20)]],
  passwordRepeat: ['', [Validators.required]],
});
```

### Validación Angular + personalizada:

Lo primero nos creamos un fichero .ts para trabajar en nuestras validaciones custom, en este caso queremos validar que la **password** es igual a **passwordRepeat**, para ello he creado dicho fichero con el nombre **customValidator.ts**:

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

// Función para validar la contraseña
// Entran dos valores por parametro
export function comparePassword (controlName: string, matchingControlName: string){
  return (formGroup: FormGroup) => {
    // Asignamos dos controladores a nuestros valores por param
    const control = formGroup.controls[controlName];
    const matchingControl = formGroup.controls[matchingControlName];
    // Control de errores
    if (matchingControl.errors && !matchingControl.errors.mustMatch) {
      return;
    }
    // Setter Errores
    if (control.value !== matchingControl.value) {
      matchingControl.setErrors({ mustMatch: true });
    } else {
      matchingControl.setErrors(null);
    }
  };
}
```

Y para usar nuestra validación custom tenemos que importarlo en nuestro componente e inicializar en el constructor, haremos lo siguiente:

```
import { comparePassword } from './customValidator';

constructor(private formBuilder: FormBuilder) {
  // Nuestro formulario - sin campos por defecto
  // Podemos meter valores por defecto en las comillas
  this.userRegisterForm = this.formBuilder.group({
    name: ['', [Validators.required, Validators.maxLength(20)]],
    password: ['', [Validators.required, Validators.maxLength(20)]],
    passwordRepeat: ['', [Validators.required, Validators.maxLength(20)]],
  },
  {

```

```
// Validación custom de password
validator: comparePassword('password', 'passwordRepeat')
});
}
```

## Ejercicio en clase

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).

 Angular S3: More Forms