Clase 6

Formularios reactivos y validaciones

¿Qué son los Formularios Reactivos?

Piensa en un formulario como una conexión en vivo

Imagina una hoja de cálculo que recalcula automáticamente cuando cambias un valor. Los formularios reactivos funcionan igual: **reaccionan** a los cambios en tiempo real.

¿Por qué "Reactivos"?

1. **Escuchas cambios** en tiempo real (como un stream de datos)
2. **Respondes automáticamente** a esos cambios
3. **Controlas todo desde TypeScript** (no desde el HTML)

¿Por qué usar Formularios Reactivos?

Template-driven (enfoque tradicional):

```
searchTerm: string = '';
onSearchChange() { this.filtrar(); } // Manual
```

Reactive Forms (enfoque reactivo):

```
searchControl = new FormControl('')
// El formulario REACCIONA cuando el usuario escribe
searchControl.valueChanges // Observable stream
  .subscribe((searchTerm) => {
    this.filtrarResultados(searchTerm) // Reacción automática
  })
```

Enfoque Progresivo de Aprendizaje

Vamos a construir conocimiento paso a paso:

Paso 1: `FormControl` - El bloque básico (1 campo)
Paso 2: `FormGroup` - Agrupar campos
Paso 3: `Validators` - Validación de datos
Paso 4: `FormArray` - Listas dinámicas
Paso 5: `FormBuilder` - Crear formularios fácilmente

FormControl - El Bloque Básico

```
import { FormControl } from '@angular/forms'

// Crear un control para el campo de búsqueda
searchControl = new FormControl('') // '' = valor inicial

// Obtener el valor actual
console.log(this.searchControl.value) // ''

// Escuchar cambios
this.searchControl.valueChanges.subscribe((valor) => {
  console.log('Nuevo valor:', valor)
})
```

Concepto clave: FormControl representa UN solo campo

FormControl en el Template

Conectar el FormControl con el HTML:

```
// Component
export class DashboardComponent {
  searchControl = new FormControl('')
}
```

```
<!-- Template -->
<input [formControl]="searchControl" placeholder="Buscar..." />
```

¡Eso es todo! Angular conecta automáticamente el input con el control.

FormGroup - Agrupar Campos

FormGroup = Contenedor de múltiples FormControls

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

filterForm = new FormGroup({
  searchTerm: new FormControl(''), // FormControl #1
  department: new FormControl('todos'), // FormControl #2
  availability: new FormControl('todos'), // FormControl #3
})

// Obtener valores
console.log(this.filterForm.value)
// { searchTerm: '', department: 'todos', availability: 'todos' }
```

FormGroup en el Template

```
// Component
filterForm = new FormGroup({
  searchTerm: new FormControl(''),
  department: new FormControl('todos'),
})
```

```
<!-- Template -->
<form [formGroup]="filterForm">
  <input formControlName="searchTerm" />
  <select formControlName="department">
    <option value="todos">Todos</option>
  </select>
</form>
```

FormBuilder - El Atajo

```
// ❌ SIN FormBuilder (más código)
filterForm = new FormGroup({
  searchTerm: new FormControl(''),
  department: new FormControl('todos')
});

// ✅ CON FormBuilder (menos código)
constructor(private fb: FormBuilder) {}

filterForm = this.fb.group({
  searchTerm: [''],       // [valor inicial]
  department: ['todos']
});
```

Estados del Formulario - Conceptos Clave

Los formularios trackean 3 tipos de estados:

1. **¿El usuario ha tocado el campo? →** `touched` / `untouched`
2. **¿El usuario ha modificado el valor? →** `dirty` / `pristine`
3. **¿El valor es válido? →** `valid` / `invalid`

Estos estados te ayudan a decidir **cuándo mostrar errores**.

Estados: Touched vs Untouched

touched = El usuario hizo clic en el campo (aunque no escribió nada)
untouched = El usuario nunca hizo clic en el campo

```
// Verificar si el usuario tocó el campo
if (this.filterForm.get('searchTerm')?.touched) {
  console.log('El usuario hizo clic en este campo')
}
```

Uso común: Mostrar errores solo DESPUÉS de que el usuario interactuó con el campo.

Estados: Dirty vs Pristine

dirty = El usuario cambió el valor (escribió algo)
pristine = El valor está sin modificar (igual que el inicial)

```
// Verificar si el formulario ha sido modificado
if (this.filterForm.dirty) {
  console.log('El usuario cambió algo en el formulario')
}
```

Uso común: Advertir al usuario antes de salir si hay cambios sin guardar.

Estados: Valid vs Invalid

valid = Todos los validadores pasaron
invalid = Al menos un validador falló

```
// Verificar si el formulario es válido
if (this.filterForm.valid) {
  console.log('¡Formulario válido! Podemos enviar los datos')
} else {
  console.log('Hay errores en el formulario')
}
```

Uso común: Habilitar/deshabilitar el botón de envío.

Ejemplo Visual de Estados

```
nameControl = new FormControl('');

// ESTADO INICIAL
nameControl.untouched → true    (no tocado)
nameControl.pristine → true     (sin modificar)
nameControl.valid → true        (válido, sin validadores)

// Usuario hace clic en el campo
nameControl.touched → true      (✓ tocado)
nameControl.pristine → true     (aún sin modificar)

// Usuario escribe "Juan"
nameControl.dirty → true        (✓ modificado)
nameControl.value → "Juan"
```

Validators - Validación de Datos

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

// FormControl con validadores
nameControl = new FormControl('', [
  Validators.required, // Campo obligatorio
  Validators.minLength(2), // Mínimo 2 caracteres
])

// Verificar si es válido
if (nameControl.valid) {
  console.log('¡Nombre válido!')
}
```

Los validadores se pasan como el segundo parámetro

Validadores Built-in Comunes

Angular incluye validadores predefinidos:

```
this.fb.group({
  name: ['', Validators.required], // Obligatorio
  email: ['', Validators.email], // Formato email
  age: [0, Validators.min(18)], // Mínimo 18
  password: ['', Validators.minLength(8)], // Mínimo 8 caracteres
  website: ['', Validators.pattern(/^https/)], // Patrón regex
})
```

Puedes combinar múltiples validadores en un array

Validadores con FormBuilder

```
filterForm = this.fb.group({
  searchTerm: [
    '',
    [
      Validators.minLength(2), // Buscar solo si escribió 2+ caracteres
    ],
  ],
  department: [
    'todos',
    [
      Validators.required, // Siempre debe tener un valor
    ],
  ],
  availability: ['todos', Validators.required],
})
```

Mostrar Errores de Validación

```
// Component
isFieldInvalid(fieldName: string): boolean {
  const field = this.filterForm.get(fieldName);
  return !!(field?.invalid && field?.touched);
}
```

```
<!-- Template -->
<input
  formControlName="searchTerm"
  [class.error]="isFieldInvalid('searchTerm')"
/>

<span *ngIf="isFieldInvalid('searchTerm')" class="error-message">
  Mínimo 2 caracteres para buscar
</span>
```

Obtener el Mensaje de Error Específico

```
getFieldError(fieldName: string): string {
  const field = this.filterForm.get(fieldName);

  if (field?.touched && field?.errors) {
    if (field.errors['required']) return 'Campo requerido';
    if (field.errors['email']) return 'Email inválido';
    if (field.errors['minlength']) {
      return `Mínimo ${field.errors['minlength'].requiredLength} caracteres`;
    }
  }
  return '';
}
```

FormArray - Listas Dinámicas

FormArray = Lista de controles que crece o se reduce

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

memberForm = this.fb.group({
  name: ['', Validators.required],
  skills: this.fb.array([])  // Array vacío inicialmente
});

get skillsArray(): FormArray {
  return this.memberForm.get('skills') as FormArray;
}
```

Agregar y Eliminar Elementos del FormArray

```
// Agregar una nueva habilidad
addSkill() {
  this.skillsArray.push(
    this.fb.control('', Validators.required)
  );
}

// Eliminar una habilidad por índice
removeSkill(index: number) {
  this.skillsArray.removeAt(index);
}
```

FormArray en el Template

```
<div formArrayName="skills">
  <!-- Iterar sobre cada control del array -->
  <div *ngFor="let skill of skillsArray.controls; let i = index">
    <input [formControlName]="i" placeholder="Habilidad" />
    <button type="button" (click)="removeSkill(i)">x</button>
  </div>

  <!-- Mensaje si no hay habilidades -->
  <div *ngIf="skillsArray.length === 0">No hay habilidades. Agrega una.</div>
</div>

<button type="button" (click)="addSkill()">+ Agregar Habilidad</button>
```

FormArray: Validar Cantidad Mínima

```
// Requiere al menos 1 habilidad
memberForm = this.fb.group({
  skills: this.fb.array([], [
    Validators.required,
    Validators.minLength(1)  // Mínimo 1 elemento
  ])
});

// En el template
<span *ngIf="memberForm.get('skills')?.invalid &&
             memberForm.get('skills')?.touched">
  Se requiere al menos una habilidad
</span>
```