Actividades Prácticas: Formularios en Angular 2+

Actividad 1: Formulario Template-Driven - Registro de Usuario Básico

Descripción

Crear un formulario dirigido por plantilla para el registro de un nuevo usuario. Este es el primer contacto con formularios en Angular y te permitirá entender cómo las directivas de Angular vinculan datos bidireccionales.

Objetivos de Aprendizaje

- Usar la directiva `ngModel` para vinculación bidireccional de datos
- Implementar validación básica con directivas HTML
- Mostrar mensajes de error usando validadores integrados
- Capturar y procesar datos del formulario

Práctica 1: Formulario de Registro

Archivo: `registro.component.ts`

```
import { Component } from '@angular/core';
import { NgForm } from '@angular/forms';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';

@Component({
selector: 'app-registro',
standalone: true,
imports: [CommonModule, FormsModule],
templateUrl: './registro.component.html',
styleUrls: ['./registro.component.css']
})
export class RegistroComponent {
usuario = {
nombre: '',
email: '',
password: '',
confirmPassword: ''
};
```

```
usuariosRegistrados: any[] = [];
mensajeExito: string = '';

onSubmit(form: NgForm) {
if (form.valid) {
if (this.usuario.password === this.usuario.confirmPassword) {
this.usuariosRegistrados.push({...this.usuario});
this.mensajeExito = ¡Bienvenido, ${this.usuario.nombre}!;
form.resetForm();
console.log('Usuarios registrados:', this.usuariosRegistrados);
} else {
alert('Las contraseñas no coinciden');
}
}
}
}
```

Archivo: `registro.component.html`

Formulario de Registro

Nombre Completo:

El nombre es requerido

Mínimo 3 caracteres

Email:

El email es requerido

Formato de email inválido

Contraseña:

La contraseña es requerida

Mínimo 6 caracteres

Confirmar Contraseña:

Registrarse

{{ mensajeExito }}

Usuarios Registrados

Conceptos clave abordados: Directivas `ngModel`, validadores HTML integrados, referencias de template, ciclo de vida del formulario, binding bidireccional.

Actividad 2: Formulario Reactivo - Gestión de Productos

Descripción

Crear un formulario reactivo para gestionar un catálogo de productos. Este enfoque te introduce a `FormBuilder`, `FormGroup` y `FormControl` programáticos, proporcionando mayor control y testabilidad.

Objetivos de Aprendizaje

- Utilizar `FormBuilder` para simplificar la creación de formularios
- Implementar validadores síncronos integrados
- Acceder a controles de formulario mediante referencias programáticas
- Gestionar estado del formulario y valores

Práctica 2: Gestor de Productos

Archivo: `productos.component.ts`

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators, ReactiveFormsModule } from
'@angular/forms';
import { CommonModule } from '@angular/common';

interface Producto {
id: number;
nombre: string;
descripcion: string;
precio: number;
cantidad: number;
categoria: string;
}

@Component({
selector: 'app-productos',
standalone: true,
imports: [ReactiveFormsModule, CommonModule],
templateUrl: './productos.component.html',
styleUrls: ['./productos.component.css']
})
export class ProductosComponent implements OnInit {
productosForm!: FormGroup;
productos: Producto[] = [];
proximoId = 1;

constructor(private fb: FormBuilder) {}

ngOnInit() {
this.inicializarFormulario();
}

inicializarFormulario() {
this.productosForm = this.fb.group({
nombre: ['', [Validators.required, Validators.minLength(3)]],
```

```
descripcion: ['', [Validators.required, Validators.maxLength(200)]],
precio: ['', [Validators.required, Validators.min(0.01)]],
cantidad: ['', [Validators.required, Validators.min(1), Validators.max(1000)]],
categoria: ['', Validators.required]
});
}

onSubmit() {
if (this.productosForm.valid) {
const nuevoProducto: Producto = {
id: this.proximoId++,
...this.productosForm.value
};
this.productos.push(nuevoProducto);
console.log('Productos:', this.productos);
this.productosForm.reset();
} else {
console.log('Formulario inválido');
}
}

obtenerProducto(id: number) {
return this.productos.find(p => p.id === id);
}

eliminarProducto(id: number) {
this.productos = this.productos.filter(p => p.id !== id);
}

calcularTotal(): number {
return this.productos.reduce((total, p) => total + (p.precio * p.cantidad), 0);
}
}
```

Archivo: `productos.component.html`

Gestor de Productos

Nombre del Producto:

Requerido

Mínimo 3 caracteres

Descripción:

Requerido

Máximo 200 caracteres

Precio (€):

Requerido

Debe ser mayor a 0

Cantidad:

Requerido

Mínimo 1

Categoría:

Requerido

Agregar Producto

Productos Ingresados

\begin{table} \begin{tabular}{|c|l|p{3cm}|c|c|c|} \hline ID & Nombre & Descripción & Precio & Cantidad & Acciones \\ \hline\hline \end{tabular} \caption{Listado de productos registrados} \end{table}

Total del Inventario: {{ calcularTotal() | currency:'EUR':'symbol':'1.2-2' }}

Conceptos clave abordados: `FormBuilder`, `FormGroup`, `FormControl`, validadores síncronos, acceso programático a controles, gestión de estado del formulario.

Actividad 3: Validadores Personalizados y FormArray

Descripción

Crear un formulario dinámico que agregue múltiples líneas de detalles de compra utilizando `FormArray`. Implementarás validadores personalizados para reglas de negocio específicas.

Objetivos de Aprendizaje

- Crear validadores personalizados síncronos y asincronos
- Utilizar `FormArray` para gestionar colecciones dinámicas de controles
- Agregar y eliminar campos dinámicamente
- Validar relaciones entre múltiples campos

Práctica 3: Formulario de Factura con Detalles Dinámicos

Archivo: `factura.component.ts`

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, FormArray, Validators, AbstractControl,
ValidationErrors } from '@angular/forms';
import { ReactiveFormsModule, FormsModule } from '@angular/forms';
import { CommonModule } from '@angular/common';

@Component({
selector: 'app-factura',
```

```
standalone: true,
imports: [ReactiveFormsModule, FormsModule, CommonModule],
templateUrl: './factura.component.html',
styleUrls: ['./factura.component.css']
})
export class FacturaComponent implements OnInit {
facturaForm!: FormGroup;
impuesto = 0.21; // IVA 21%

constructor(private fb: FormBuilder) {}

ngOnInit() {
this.inicializarFormulario();
}

inicializarFormulario() {
this.facturaForm = this.fb.group({
numeroFactura: ['', [Validators.required, this.validarNumeroFactura.bind(this)]],
cliente: ['', [Validators.required, Validators.minLength(3)]],
fecha: [new Date().toISOString().split('T')[0], Validators.required],
detalles: this.fb.array([])
});
}

validarNumeroFactura(control: AbstractControl): ValidationErrors | null {
const valor = control.value;
if (!valor) return null;

// Validar formato: FAC-XXXXXX
const regex = /^FAC-\d{6}$/;
return regex.test(valor) ? null : { formatoFactura: true };


}

get detalles() {
return this.facturaForm.get('detalles') as FormArray;
}

agregarDetalle() {
const detalleForm = this.fb.group({
producto: ['', Validators.required],
cantidad: [1, [Validators.required, Validators.min(1)]],
precioUnitario: ['', [Validators.required, Validators.min(0.01)]],
descuento: [0, [Validators.min(0), Validators.max(100)]]
});
this.detalles.push(detalleForm);
}

eliminarDetalle(index: number) {
this.detalles.removeAt(index);
}

calcularSubtotal(): number {
return this.detalles.controls.reduce((total, detalle) => {
```

```
const cantidad = detalle.get('cantidad')?.value || 0;
const precio = detalle.get('precioUnitario')?.value || 0;
const descuento = detalle.get('descuento')?.value || 0;
const descuentoAplicado = (precio * cantidad * descuento) / 100;
return total + (precio * cantidad - descuentoAplicado);
}, 0);
}

calcularIVA(): number {
return this.calcularSubtotal() * this.impuesto;
}

calcularTotal(): number {
return this.calcularSubtotal() + this.calcularIVA();
}

onSubmit() {
if (this.facturaForm.valid && this.detalles.length > 0) {
const factura = {
...this.facturaForm.value,
subtotal: this.calcularSubtotal(),
iva: this.calcularIVA(),
total: this.calcularTotal()
};
console.log('Factura:', factura);
alert('Factura guardada correctamente');
} else {
alert('Completa todos los campos requeridos y agrega al menos un detalle');
}
}
}
```

Archivo: `factura.component.html`

Generador de Facturas

Número de Factura:

Formato: FAC-XXXXXX (6 dígitos)

Fecha:

Cliente:

Requerido

Detalles de Factura

Producto:

Cantidad:

Precio Unitario (€):

Descuento (%):

Eliminar

+ Agregar Detalle

Subtotal: **{{ calcularSubtotal() | currency:'EUR':'symbol':'1.2-2' }}**

IVA (21%): **{{ calcularIVA() | currency:'EUR':'symbol':'1.2-2' }}**

Total: **{{ calcularTotal() | currency:'EUR':'symbol':'1.2-2' }}**

Guardar Factura

Conceptos clave abordados: Validadores personalizados, `FormArray`, validación dinámica, operaciones de cálculo, gestión de colecciones de controles.

Actividad 4: Validación Asincrónica y Integración con API

Descripción

Desarrollar un formulario que valide datos de forma asincrónica mediante llamadas simuladas a un servicio backend. Aprenderás patrones de validación moderna y gestión de promesas.

Objetivos de Aprendizaje

- Implementar validadores asincronos con promesas
- Simular llamadas a servicios HTTP
- Mostrar estados de carga durante validación
- Manejar errores de validación asincrónica

Práctica 4: Validador Asincrónico de Email Único

Archivo: `validators.service.ts`

```
import { Injectable } from '@angular/core';
import { AbstractControl, AsyncValidatorFn, ValidationErrors } from '@angular/forms';
import { Observable, of } from 'rxjs';
import { delay, map } from 'rxjs/operators';

@Injectable({
providedIn: 'root'
})
export class ValidatorsService {

// Simular base de datos de emails registrados
emailsRegistrados = ['admin@example.com', 'usuario@example.com', 'test@example.com'];

validarEmailUnico(): AsyncValidatorFn {
return (control: AbstractControl): Observable<ValidationErrors | null> => {
if (!control.value) {
```

return of(null);
}

```
  // Simular llamada a API con delay de 1 segundo
  return of(this.emailsRegistrados.includes(control.value)).pipe(
    delay(1000),
    map(existe => existe ? { emailExiste: true } : null)
  );
};
```

}

validarUsernameDisponible(): AsyncValidatorFn {
return (control: AbstractControl): Observable<ValidationErrors | null> => {
if (!control.value) {
return of(null);
}

```
  // Simular validación: usernames que comienzan con 'admin' están reservados
  return of(control.value.toLowerCase().startsWith('admin')).pipe(
    delay(800),
    map(reservado => reservado ? { usernameReservado: true } : null)
  );
};
```

}
}

Archivo: `registro-avanzado.component.ts`

import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators, ReactiveFormsModule } from '@angular/forms';
import { CommonModule } from '@angular/common';
import { ValidatorsService } from './validators.service';

@Component({
selector: 'app-registro-avanzado',
standalone: true,
imports: [ReactiveFormsModule, CommonModule],
templateUrl: './registro-avanzado.component.html',
styleUrls: ['./registro-avanzado.component.css']
})
export class RegistroAvanzadoComponent implements OnInit {
registroForm!: FormGroup;
enviando = false;

constructor(
private fb: FormBuilder,

```
private validatorsService: ValidatorsService
) {}

ngOnInit() {
this.inicializarFormulario();
}

inicializarFormulario() {
this.registroForm = this.fb.group({
username: [
'',
[Validators.required, Validators.minLength(3)],
[this.validatorsService.validarUsernameDisponible()]
],
email: [
'',
[Validators.required, Validators.email],
[this.validatorsService.validarEmailUnico()]
],
password: ['', [Validators.required, Validators.minLength(8)]],
terminos: [false, Validators.requiredTrue]
});
}

onSubmit() {
if (this.registroForm.valid) {
this.enviando = true;
// Simular envío a servidor
setTimeout(() => {
console.log('Registro completado:', this.registroForm.value);
this.enviando = false;
alert('¡Registro completado exitosamente!');
}, 2000);
}
}

validarCampo(nombreCampo: string): boolean {
const control = this.registroForm.get(nombreCampo);
return control?.invalid && control?.touched || false;
}

mostrarCargando(nombreCampo: string): boolean {
const control = this.registroForm.get(nombreCampo);
return control?.pending || false;
}
}
```

Archivo: `registro-avanzado.component.html`

Registro Avanzado con Validación Asincrónica

Username:

Verificando...

Requerido

Mínimo 3 caracteres

Username no disponible (reservado)

Email:

Verificando disponibilidad...

Requerido

Email inválido

Este email ya está registrado

Validando disponibilidad del email...

Contraseña:

Requerida

Mínimo 8 caracteres

Acepto los términos y condiciones

Debes aceptar los términos

{{ enviando ? 'Registrando...' : 'Crear Cuenta' }}

Validando información...

Conceptos clave abordados: Validadores asincronos, `AsyncValidatorFn`, simulación de promesas, manejo de estados de carga, Observable y operadores RxJS.

Actividad 5: FormGroup Anidados y Patrón Dinámico Complejo

Descripción

Desarrollar un formulario complejo con `FormGroup` anidados para modelar estructuras jerárquicas de datos, como un formulario de perfil de usuario con múltiples direcciones.

Objetivos de Aprendizaje

- Crear y manipular `FormGroup` anidados
- Implementar validación a nivel de grupo
- Gestionar complejidad en formularios de múltiples niveles
- Aplicar patrones avanzados de reactividad

Práctica 5: Perfil de Usuario Completo con Direcciones Múltiples

Archivo: `perfil-usuario.component.ts`

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, FormArray, Validators, ReactiveFormsModule } from
'@angular/forms';
import { CommonModule } from '@angular/common';

interface PerfilUsuario {
informacionPersonal: {
nombre: string;
apellido: string;
telefono: string;
fechaNacimiento: string;
};
direcciones: Array<{
tipo: string;
calle: string;
ciudad: string;
codigoPostal: string;
pais: string;
}>;
}

@Component({
selector: 'app-perfil-usuario',
standalone: true,
imports: [ReactiveFormsModule, CommonModule],
templateUrl: './perfil-usuario.component.html',
styleUrls: ['./perfil-usuario.component.css']
})
export class PerfilUsuarioComponent implements OnInit {
perfilForm!: FormGroup;
tiposDireccion = ['Residencial', 'Laboral', 'Otra'];

constructor(private fb: FormBuilder) {}

ngOnInit() {
this.inicializarFormulario();
}

inicializarFormulario() {
this.perfilForm = this.fb.group({
informacionPersonal: this.fb.group({
nombre: ['', [Validators.required, Validators.minLength(2)]],
apellido: ['', [Validators.required, Validators.minLength(2)]],
telefono: ['', [Validators.required, Validators.pattern(/[1]{9,}$/)]],
fechaNacimiento: ['', Validators.required]
}),
direcciones: this.fb.array([])
});
```

```
// Agregar una dirección por defecto
this.agregarDireccion();


}

get informacionPersonal() {
return this.perfilForm.get('informacionPersonal') as FormGroup;
}

get direcciones() {
return this.perfilForm.get('direcciones') as FormArray;
}

crearFormDireccion(): FormGroup {
return this.fb.group({
tipo: ['Residencial', Validators.required],
calle: ['', [Validators.required, Validators.minLength(5)]],
ciudad: ['', [Validators.required, Validators.minLength(2)]],
codigoPostal: ['', [Validators.required, Validators.pattern(/[2]{5}$/)]],
pais: ['España', Validators.required]
});
}

agregarDireccion() {
this.direcciones.push(this.crearFormDireccion());
}

eliminarDireccion(index: number) {
if (this.direcciones.length > 1) {
this.direcciones.removeAt(index);
} else {
alert('Debe mantener al menos una dirección');
}
}

calcularEdad(): number {
const fechaNacimiento = this.informacionPersonal.get('fechaNacimiento')?.value;
if (!fechaNacimiento) return 0;

const hoy = new Date();
const nacimiento = new Date(fechaNacimiento);
let edad = hoy.getFullYear() - nacimiento.getFullYear();
const mes = hoy.getMonth() - nacimiento.getMonth();

if (mes < 0 || (mes === 0 && hoy.getDate() < nacimiento.getDate())) {
  edad--;
}
return edad;


}

onSubmit() {
if (this.perfilForm.valid) {
const perfil: PerfilUsuario = this.perfilForm.value;
```

```
console.log('Perfil guardado:', perfil);
alert(¡Perfil de ${perfil.informacionPersonal.nombre} actualizado exitosamente!);
}
}

generarResumen(): string {
if (!this.perfilForm.valid) return '';

const info = this.informacionPersonal.value;
const edad = this.calcularEdad();
const numDirecciones = this.direcciones.length;

return `${info.nombre} ${info.apellido}, ${edad} años, ${numDirecciones} dirección(es) registrada(s)`;


}
}
```

Archivo: `perfil-usuario.component.html`

```
# Perfil de Usuario Completo

## Información Personal

Nombre:

Nombre requerido (mínimo 2 caracteres)

Apellido:

Apellido requerido (mínimo 2 caracteres)

Fecha de Nacimiento:

Edad: {{ calcularEdad() }} años

Teléfono:

Teléfono inválido (mínimo 9 dígitos)

## Direcciones ({{ direcciones.length }})

### Dirección {{ i + 1 }}

Tipo:

Eliminar Dirección

Calle:

Mínimo 5 caracteres

Ciudad:

Código Postal:

Formato: 5 dígitos
```

País:

+ Agregar Otra Dirección

Resumen del Perfil

{{ generarResumen() }}

Guardar Perfil

Conceptos clave abordados: `FormGroup` anidados, `FormArray` avanzado, validación a nivel de grupo, patrones complejos de reactividad, gestión jerárquica de datos.

Resumen de Competencias Adquiridas

- •Dominio de Template-Driven Forms y Reactive Forms
- •Implementación de validadores síncronos y asincronos
- •Gestión de colecciones dinámicas con FormArray
- •Diseño de FormGroups anidados para estructuras complejas
- •Integración de patrones de validación de negocio
- •Manejo de estados de formulario y reactividad
- •Buenas prácticas en diseño de componentes de formulario

Recomendaciones para Profundizar

- Investiga sobre FormControl avanzados y custom validators
- Explora la gestión de estado con NgRx o Signals
- Practica con validación de cross-field
- Estudia patrones de manejo de errores robusto
- Implementa pruebas unitarias para formularios
- Considera la accesibilidad (WCAG) en tus formularios[2]

Referencias

[1] Sincfusion. (2024). Angular Template Driven vs. Reactive Forms. https://www.syncfusion.com/blogs/post/angular-template-driven-vs-reactive-forms

[2] JavaScript in Plain English. (2024). Latest Angular Best Practices (2024–2025). https://javascript.plainenglish.io/latest-angular-best-practices-2024-2025-caa4e41cf0ab

---

1. 0-9 ↩️
2. 0-9 ↩️