GOBIERNO DE ESPAÑA
MINISTERIO DE ECONOMÍA Y EMPRESA
red.es
PROFESIONALES DIGITALES | Formación Continua
El FSE invierte en tu futuro
UNIÓN EUROPEA

Formularios

En desarrollo web las aplicaciones usan formularios para loguear usuarios, actualizar un perfil, introducir información sensible y actualizar muchos otros datos.

Angular provee dos tipos de formularios:

- *Reactive forms*: son más robustos, son más escalables, reusables y testeables. Si la aplicación se base mayoritariamente en formularios, una zona de administración por ejemplo, se debe usar esta forma de generar formularios.
- *Template-driven forms*: son más útiles para añadir simples formularios.

	Reactive	***Template-driven***
Preparación	Se crea en el componente	Creado por directivas
Modelo	Estructural	No estructural
Previsibilidad	Síncrona	Asíncrona
Validación	Funciones	Directivas
Mutabilidad	Inmutable	Mutable
Escalabilidad	Acceso a API a bajo nivel	Abstracción de la API

Reactive Forms

Los formularios *reactive* utilizan un enfoque explícito e inmutable para administrar el estado de un formulario. Cada cambio en el estado del formulario devuelve un nuevo estado, que mantiene la integridad del modelo entre cambios. También se debe de tener en cuenta a la hora de trabajar con ellos que se crean bajo la lógica de *Observable* de Angular

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

@NgModule({
  imports: [
    // other imports ...
    ReactiveFormsModule
  ],
})
export class AppModule { }
```

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

@Component({
  selector: 'app-name-editor',
  templatel: '<label>  Name:
      <input type="text" [formControl]="name">
    </label>',
  styleUrls: ['./name-editor.component.css']
})
export class NameEditorComponent {
  name = new FormControl('');
}
```

FormGroup

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

@Component({
  selector: 'app-profile-editor',
  templateUrl: './profile-editor.component.html',
  styleUrls: ['./profile-editor.component.css']
})
export class ProfileEditorComponent {
  profileForm = new FormGroup({
    firstName: new FormControl(''),
    lastName: new FormControl(''),
  });
}
```

```
<form [formGroup]="profileForm">

  <label>
    First Name:
    <input type="text" formControlName="firstName">
  </label>

  <label>
    Last Name:
    <input type="text" formControlName="lastName">
  </label>

</form>
```

FormGroup anidados

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

@Component({
  selector: 'app-profile-editor',
  templateUrl: './profile-editor.component.html',
  styleUrls: ['./profile-editor.component.css']
})
export class ProfileEditorComponent {
  profileForm = new FormGroup({
    firstName: new FormControl(''),
    lastName: new FormControl(''),
    address: new FormGroup({
      street: new FormControl(''),
      city: new FormControl(''),
      state: new FormControl(''),
      zip: new FormControl('')
    })
  });
}
```

```
<div formGroupName="address">
  <h3>Address</h3>
  <label> Street:
    <input type="text" formControlName="street">
  </label>
  <label> City:
    <input type="text" formControlName="city">
  </label>
  <label> State:
    <input type="text" formControlName="state">
  </label>
  <label> Zip Code:
    <input type="text" formControlName="zip">
  </label>
</div>
```

FormBuilder

FormBuilder es una forma diferente de realizar también un formulario *reactive*.

```
import { Component } from '@angular/core';
import { FormBuilder } from '@angular/forms';

@Component({
  selector: 'app-profile-editor',
  templateUrl: './profile-editor.component.html',
  styleUrls: ['./profile-editor.component.css']
})
export class ProfileEditorComponent {
  profileForm = this.fb.group({
    firstName: [''],
    lastName: [''],
    address: this.fb.group({
      street: [''],
      city: [''],
      state: [''],
      zip: ['']
    }),
  });

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

Template-driven forms

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

@NgModule({
  imports: [
    // other imports ...
    FormsModule
  ],
})
export class AppModule { }
```

```
<form (ngSubmit)="onSubmit()" #heroForm="ngForm">
    <div class="form-group">
      <label for="name">Name</label>
      <input type="text" class="form-control" id="name" required [(ngModel)]="model.name" name="name"
            #name="ngModel">
      <div [hidden]="name.valid || name.pristine" class="alert alert-danger"> Name is required </div>
    </div>

    <div class="form-group">
      <label for="alterEgo">Alter Ego</label>
     <input type="text" class="form-control" id="alterEgo" [(ngModel)]="model.alterEgo"
           name="alterEgo">
    </div>

    <div class="form-group">
      <label for="power">Hero Power</label>
      <select class="form-control" id="power" required  [(ngModel)]="model.power" name="power"
            #power="ngModel">
        <option *ngFor="let pow of powers" [value]="pow">{{pow}}</option>
      </select>
      <div [hidden]="power.valid || power.pristine" class="alert alert-danger"> Power is required </div>
    </div>

    <button type="submit" class="btn btn-success" [disabled]="!heroForm.form.valid">Submit</button>
    <button type="button" class="btn btn-default" (click)="newHero(); heroForm.reset()">
          New Hero</button>
</form>
```

Validación - *Template-driven*

Para añadir validación a este tipo de formularios, se añadirá la misma forma de validación que cualquier formulario de HTML. Angular usa directivas para unir esos atributos con sus funciones de validación:

- Los atributos *required*, *minlenght* o *forbiddenName* en la etiqueta <input>
- *#var=”ngModel”* exporta la clase *NgModel* en una variable. Se usa para validar los estados de la etiqueta donde se coloque.

```
<input id="name" name="name" class="form-control" required minlength="4"
appForbiddenName="bob"
     [(ngModel)]="hero.name" #name="ngModel" >

<div *ngIf="name.invalid && (name.dirty || name.touched)" class="alert alert-danger">

  <div *ngIf="name.errors.required"> Name is required. </div>
  <div *ngIf="name.errors.minlength"> Name must be at least 4 characters long.  </div>
  <div *ngIf="name.errors.forbiddenName"> Name cannot be Bob. </div>

</div>
```

Validación - *Reactive form*

Para formulario *reactive* la validación se puede implementar también en el HTML pero *FormControl* tiene una opción para pasarle los parámetros de la validación:

```
ngOnInit(): void {
  this.heroForm = new FormGroup({
    'name': new FormControl(this.hero.name, [ Validators.required, Validators.minLength(4),
      forbiddenNameValidator(/bob/i) // <-- Here's how you pass in the custom validator.
    ]),
    'alterEgo': new FormControl(this.hero.alterEgo),
    'power': new FormControl(this.hero.power, Validators.required)
  });

}

get name() { return this.heroForm.get('name'); }

get power() { return this.heroForm.get('power'); }
```

```
<input id="name" class="form-control" formControlName="name" >

<div *ngIf="heroForm.get(‘name’).invalid && (heroForm.get(‘name’).dirty ||
heroForm.get(‘name’).touched)" class="alert alert-danger">

  <div *ngIf="heroForm.get(‘name’).errors.required"> Name is required. </div>
  <div *ngIf="heroForm.get(‘name’).errors.minlength"> Name must be at least 4 characters
long. </div>
  <div *ngIf="heroForm.get(‘name’).errors.forbiddenName"> Name cannot be Bob. </div>
</div>
```

Validación - CSS

Angular provee una serie de clases que se añaden automáticamente a los elementos de control, con ellas se puede editar el estilo del elemento según su estado de validación.

.ng-valid	.ng-invalid	.ng-pending	.ng-dirty
.ng-pristine	.ng-untouched	.ng-touched	

```
.ng-valid[required], .ng-valid.required  {
  border-left: 5px solid #42A948; /* green */
}

.ng-invalid:not(form)  {
  border-left: 5px solid #a94442; /* red */
}
```