Formularios en Angular

UT4 - Introducción al desarrollo de aplicaciones hibridas con Ionic

Objetivos de aprendizaje

- Comprender los tipos de formularios que podemos usar en Angular
- Aplicar las ayudas que nos proporciona Angular para el manejo de formularios

- Angular proporciona 2 tipos de formularios
- Ambos permiten:
  - Recogen los datos desde etiquetas input
  - Validan los datos
  - Envían datos al componente

Tipo de formulario	Donde se crea	Uso
De plantilla (template-driven)	En la plantilla	Páginas Web con pocos formularios
Reactivos	En el componente	Páginas Web con muchos formularios

Formularios template- driven

UT4 - Introducción al desarrollo de aplicaciones hibridas con Ionic

Formularios template-driven

- Requiere importar FormsModule directamente en el componente

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

@Component({
  standalone: true,
  imports: [FormsModule]
});

export class MiComponente { }
```

micomponente.component.ts

Formulario template-driven (plantilla)

Evento de envío

Indico a Angular que haga seguimiento de los cambios en el formulario

Usados para validaciones

```
<form (ngSubmit)="submit()" #usuarioForm="ngForm">
    <div class="form-group">
        <label for="usuario">Nombre:</label>
        <input type="text" name="nombre"
                           #nombre="ngModel"
                           required
                           [(ngModel)]="usuario.nombre">
    </div>
    <div class="form-group">
        <label for="password">Contraseña:</label>
        <input type="password" name="password"
                           #password="ngModel"
                           required minlength="4"
                           [(ngModel)]="usuario.password">
    </div>
    <button type="submit">ENVIAR</button>
</form>
```

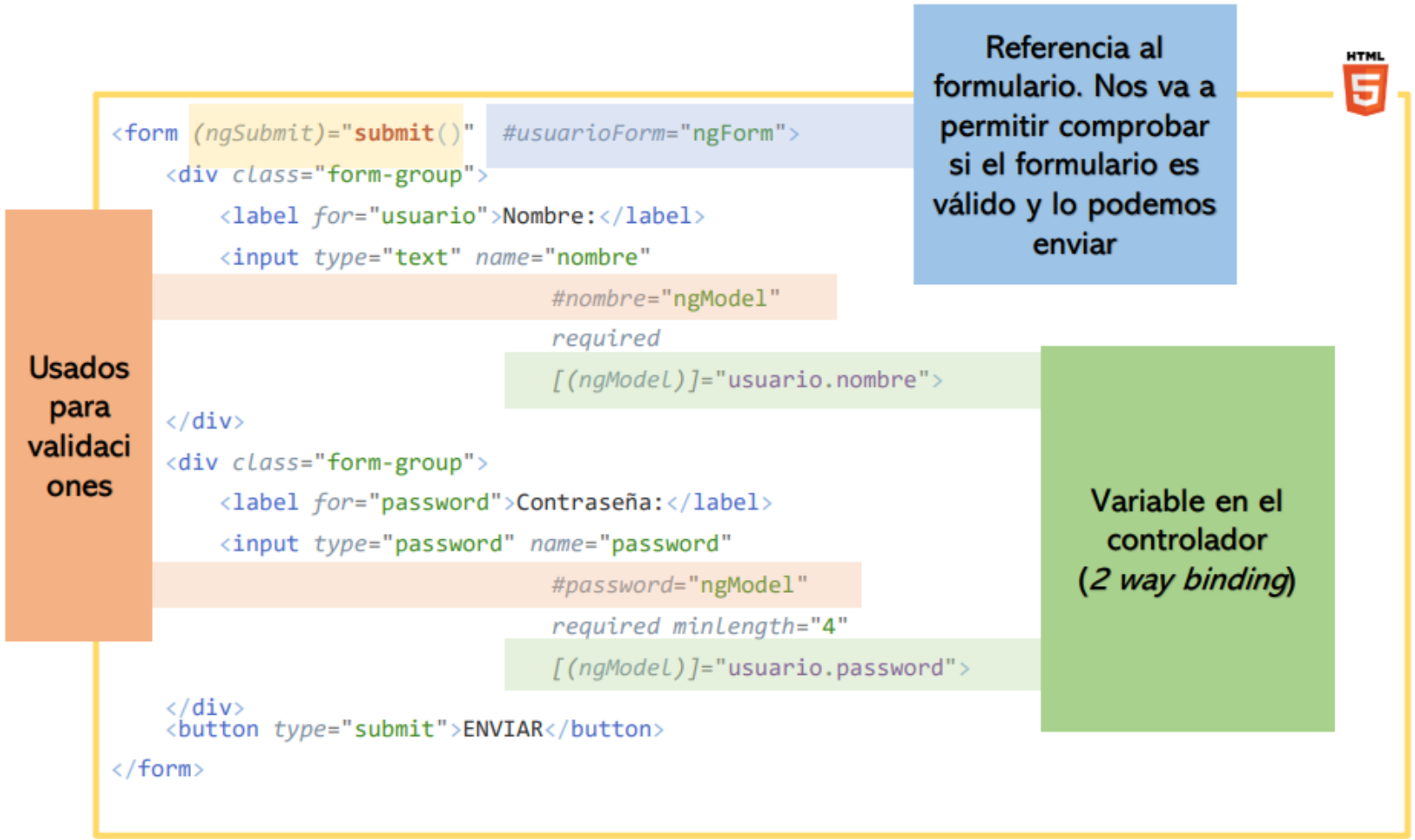
Formulario template-driven (componente)

```ts
export class LoginComponent {
  usuario ={
    nombre:"Pepe",
    password:''
}
public submit(){
    ...
  }
```

Enlazado con el formulario a través de [(ngModel)]

Evento de envío

Formulario template-driven (plantilla)




```
<form (ngSubmit)="submit()" #usuarioForm="ngForm">
    <div class="form-group">
        <label for="usuario">Nombre:</label>
        <input type="text" name="nombre"
                                #nombre="ngModel"
                                required
                                [(ngModel)]="usuario.nombre">
    </div>
    <div class="form-group">
        <label for="password">Contraseña:</label>
        <input type="password" name="password"
                                #password="ngModel"
                                required minlength="4"
                                [(ngModel)]="usuario.password">
    </div>
    <button type="submit">ENVIAR</button>
</form>
```

Validación de formularios

- Recordemos que los input pueden tener algunos campos para validar su contenido

Atributo	Significado
required	Obligatorio
minlength	Longitud mínima
maxlength	Longitud máxima

- Además, podemos establecer atributos “a medida” para realizar las validaciones (no los veremos)

Validación de formularios

El campo que hemos identificado como password tiene alguno de sus campos de validación inválidos

Evitan que mostremos errores cuando aún no se ha introducido nada:
dirty -> El input tiene texto
touched -> El input ha tenido el foco
pristine → El input no ha cambiado desde su valor inicial

```
<div @if="password.invalid && (password.dirty || password.touched)"
        class="alert">
        <div @if="password.errors?.['required']">Password obligatorio</div>
        <div @if="password.errors?.['minlength']"> El password debe tener al
                                                   menos 4 caracteres</div>
</div>
```

Acceso a los errores que se han producido
Los errores se definen en las etiquetas de la plantilla

Validación y clases CSS

- Podemos utilizar y definir clases CSS para dar estilos a los campos del formulario en función del estado de los mismos
- Estas clases las gestiona Angular pero nosotros debemos definir los estilos

`ng-valid`	`ng-invalid`
`ng-dirty`	`ng-untouched`
`ng-touched`	`ng-pristine`

Validación y envío de formularios

- **Opción A**. Solo habilitamos el botón de envío si los campos son correctos

```
<form (ngSubmit)="submit()"  #usuarioForm="ngForm">
    ...
<button type="submit"  [disabled]="!usuarioForm.valid">ENVIAR</button>

</form>
```

Es válido si todos los campos del formulario son válidos

Validación y envío de formularios

- **Opción B**. Validamos la corrección desde el componente

```html
<form (ngSubmit)="submit(usuarioForm)"  #usuarioForm="ngForm">
```

```ts
public submit(formulario:NgForm){
    if(formulario.valid){
        ...
    }
  }
```

Formularios template-driven: Resumen

Propiedad/valor		Donde se aplica	Utilidad
`#formulario=ngForm`		`<form>`	Comprobación de validez con formulario.valid
`[(ngModel)]=modelo.campo`		`Campos del formulario`	Actualización del modelo desde el controlador para posterior manipulación del modelo (almacenamiento, envío a otro componente o a un servicio
`#campo=ngModel`		`Campos del formulario`	Acceso a campo desde la plantilla (validaciones)

`campo.invalid`
`campo.required`
`campo.errors`
	`campo.pristine`
	`campo.touched`
	`campo.dirty`
		`Mostrando errores`	Permite mostrar etiquetas en función de ciertas validaciones

`(ngSubmit)=envio()`		`<form>`	Evento de envio

Formularios reactivos

UT4 - Introducción al desarrollo de aplicaciones hibridas con Ionic

Formularios reactivos

- Declaramos el formulario como si fuera un objeto
- Declaramos cada elemento del formulario y de sus valores
- Importamos los componentes de validación
- Asignamos métodos
- Trabajamos con observables
- Facilitan mucho el testeo

Formularios reactivos

- Requieren importar `ReactiveFormsModule` en el componente

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


@Component({
  standalone: true,
  selector: 'app-registro',
  imports: [ReactiveFormsModule],
  templateUrl: './registro.component.html',
})
export class RegistroComponent {}
```

Clases/etiquetas usadas

Clase	Uso
`FormGroup`	Permite crear el formulario
`FormControl`	Cada uno de los campos del formulario
`FormBuilder`	Sintaxis abreviada para definir formulario y campos

Usamos `FormControl` de manera individual cuando tenemos pocos campos (separados) y `FormGroup`/`FormBuilder` cuando queremos manejar un formulario con más campos

FormControl (componente)

```ts
export class LoginComponent {
nombre:FormControl;

 constructor(){
    this.nombre=new FormControl('');
  }

  ngOnInit(){
    this.nombre.valueChanges.subscribe((valor)=>{
      console.log(valor);
    })
  }
 onSubmit(){
    console.log(this.nombre.value);
  }
}
```

Definición

Inicialización (permite indicar valor por defecto)

Acceso con subscripción

Acceso sin subscripción

FormControl (plantilla)

```
<input type="text" id="nombre" name="nombre [formControl]="nombre"  >
```

Asocio campo del formulario con variable FormControl en el componente

```
<p>
    Nombre: {{nombre.value }}
</p>
```

Acceso

FormGroup (componente)

TS

```ts
export class LoginComponent {
 usuario:FormGroup;
  constructor(){
    this.usuario=new FormGroup({
      nombre:new FormControl(''),
      apellidos:new FormControl('')
    });
  }

  ngOnInit(){
    this.usuario.valueChanges.subscribe((valor)=>{
      console.log(valor);
      console.log(valor.nombre+ ' '+ valor.apellidos);
    })
  }
 }
```

Definición

Inicialización

Acceso con subscripción

FormGroup (componente) (II)

```ts
export class LoginComponent {
 usuario:FormGroup;
  constructor(){
    this.usuario=new FormGroup({
      nombre:new FormControl(''),
      apellidos:new FormControl('')
    });
  }

  onSubmit(){
    console.log(this.usuario.value);
    console.log(this.usuario.value.nombre+ ' '+ this.usuario.value.apellidos);

  }
}
```

Definición

Inicialización
(

Acceso sin
subscripción

FormGroup (plantilla)

Asocio formulario con variable FormGroup en el componente

```
<form  [formGroup]="usuario" (ngSubmit)="onSubmit()">

 <input type="text" id="nombre" name="nombre" formControlName="nombre"  >
 <input type="apellidos" id="apellidos" name="apellidos" formControlName="apellidos"  >
```

Asocio campos de formulario con variables FormControl en el componente

```
 <button [disabled]="!usuario.valid" type="submit">Enviar</button>
```

Validando formularios reactivos

```ts
constructor(){
    this.usuario=new FormGroup({
      nombre:new FormControl('Pepito',
        [Validators.required,
          Validators.minLength(10),
          Validators.maxLength(20)
        ]
      ),
      email:new FormControl(null,
        [Validators.required,
          Validators.email
        ])
    });
  }
```

Inicialización

Validadores

Al definir los validadores en el componente no es necesario especificarlos en la plantilla (HTML)

Validando formularios reactivos (plantilla)

Acceso a campo del formulario

```
<input type="text" id="nombre" name="nombre" formControlName="nombre"  >
        <div @if="usuario.get('nombre')?.invalid && usuario.get('nombre')?.dirty"  class="mt-2 alert alert-danger">
              <div @if="usuario.get('nombre')?.errors?.['required']">Campo obligatorio</div>
                <div @if="usuario.get('nombre')?.errors?.['minlength']">Longitud mínima 10 caracteres</div>
               <div @if="usuario.get('nombre')?.errors?.['maxlength']">Longitud máxima 20 caracteres</div>
        </div>

<button [disabled]="!usuario.valid" type="submit" class="btn btn-info btn-lg">Enviar</button>
```

Acceso a validadores

Los campos del formulario tienen los mismos atributos que en formularios template-driven: dirty, touched, pristine…

Validando formularios reactivos (refactorizado)

```html
<input type="text" id="nombre" name="nombre" formControlName="nombre"  >
        <div @if="usuario.get('nombre')?.invalid && usuario.get('nombre')?.dirty"  class="mt-2 alert alert-danger">
              <div @if="compruebaError('nombre','required'">Campo obligatorio</div>
               <div @if="compruebaError('nombre', 'minlength'">Longitud mínima 10 caracteres</div>
               <div @if="compruebaError('nombre', 'maxlength'">>Longitud máxima 20 caracteres</div>
        </div>

<button [disabled]="!usuario.valid" type="submit" class="btn btn-info btn-lg">Enviar</button>
```

```typescript
  public compruebaError(campo:string,error:string){
                  return this.formulario.get(campo)?.hasError(error) &&
                         this.formulario.get(campo)?.touched
  }
```

FormBuilder

- Sintaxis alternativa a FormGroup que simplifica la creación de formularios

```
import { Component, inject } from '@angular/core';
import { NonNullableFormBuilder, ReactiveFormsModule, Validators } from '@angular/forms';

@Component({
  standalone: true,
  selector: 'app-formbuilder',
  imports: [ReactiveFormsModule],
  templateUrl: './formbuilder.component.html',
})
export class FormbuilderComponent {
  private fb = inject(NonNullableFormBuilder);

  usuario = this.fb.group({
    nombre: ['Pepito', [Validators.required, Validators.minLength(5)]],
    email: ['', [Validators.required, Validators.email]],
  });
}
```

Validadores

Validador	Significado
`required`	Campo obligatorio
`maxlength`	Longitud máxima. Recibe un entero
`minlength`	Longitud mínima. Recibe un entero
`email`	El campo es un email
`max`	Valor máximo. Recibe un entero
`min`	Valor mínimo. Recibe un entero
`pattern`	Patron. Recibe expresión regular

Formularios reactivos: Resumen

Propiedad/valor	Donde se aplica	Utilidad
`formulario=new FormGroup()`	`Componente`	Inicializo formulario reactivo
`[formGroup]=formulario`	`<form>`	Asocio etiqueta de formulario con formulario reactivo
`nombre:new FormControl( valorInicial, [validadores])`	`Componente`	Inicializo campo del formulario, con sus correspondientes validadores
`formulario.get("campo")`	`Plantilla`	Acceso a campo desde la plantilla (validaciones)
`campo.invalid` `campo.required` `campo.errors` `campo.pristine` `campo.touched` `campo.dirty`	`Mostrando errores`	Permite mostrar etiquetas en función de ciertas validaciones
`(ngSubmit)=envio()`	`<form>`	Evento de envio

Formularios ionic

Creación de formularios con Ionic

Formularios

- Vamos a utilizar formularios utilizando el modelo reactivo
  - Son formularios que creamos mediante código en el controlador.
- Requisito previo: Importar ReactiveFormsModule en la página donde vamos a usarlo

```
import { Component } from '@angular/core';
import { ReactiveFormsModule } from '@angular/forms';
import { IonicModule } from '@ionic/angular/standalone';
import { CommonModule } from '@angular/common';

@Component({
  standalone: true,
  selector: 'app-nuevo-contacto',
  imports: [CommonModule, IonicModule, ReactiveFormsModule],
  templateUrl: './nuevo-contacto.page.html',
})
export class NuevoContactoPage {}
```

Clases/etiquetas usadas

Clase	Uso
`FormGroup`	Permite crear el formulario
`FormControl`	Cada uno de los campos del formulario
`<ion-input>`	Campos del formulario (vista)

Definiendo la vista del formulario

```
<form [formGroup]="formulario" (ngSubmit)="onSubmit()">
    <ion-item>
     <ion-input label= "Nombre" type="text" formControlName="nombre"></ion-input>
    </ion-item>
    <ion-item>
      <ion-input label= "Edad" type="number" formControlName="edad"></ion-input>
    </ion-item>
    <ion-item>
     <ion-input label= "EMail"  type="email" formControlName="email"></ion-input>
    </ion-item>
    <ion-item>
     <ion-input label= "Password" type="password" formControlName="password"></ion-input>
    </ion-item>
    <ion-item>
      <ion-input label= "DNI" type="text" formControlName="dni"></ion-input>
    </ion-item>
      <ion-button type="submit“[disabled]="!formulario.valid" class="ion-margin">Nuevo contacto</ion-button>
```

Definiendo los datos del formulario

```ts
@Component({
  selector: 'app-nuevo-contacto',
  standalone: true,
  imports: [CommonModule, IonicModule, ReactiveFormsModule],
  templateUrl: './nuevo-contacto.page.html',
})
export class NuevoContactoPage {
  private fb = inject(NonNullableFormBuilder);

  formulario = this.fb.group({
    nombre: ['', [Validators.required]],
    apellidos: [''],
    edad: [0, [Validators.min(0)]],
    email: ['', [Validators.required, Validators.email]],
    password: ['', [Validators.required, Validators.minLength(4)]],
    dni: [''],
  });

  onSubmit() {
    if (this.formulario.invalid) {
      this.formulario.markAllAsTouched();
      return;
    }
    console.log(this.formulario.getRawValue());
  }
}
```

Uso de validadores predefinidos

```ts
export class NuevoContactoPage {
  private fb = inject(NonNullableFormBuilder);
  formulario = this.fb.group({
    nombre: [
      'Pepito',
      [
        Validators.required,
        Validators.minLength(10),
        Validators.maxLength(15)
      ]
    ],
    apellidos: [''],
    edad: [
      18,
      [
        Validators.min(18),
        Validators.max(65)
      ]
    ]
  });
}
```

Mostrando errores

```
<ion-item>
  <ion-label position="stacked">Nombre</ion-label>

  <ion-input
   type="text"
   formControlName="nombre">
  </ion-input>

  @if (formulario.get('nombre')?.hasError('required') &&
      formulario.get('nombre')?.touched) {

   <ion-note color="danger">
    El campo nombre es obligatorio
   </ion-note>
  }

  @if (formulario.get('nombre')?.hasError('minlength') &&
      formulario.get('nombre')?.touched) {

   <ion-note color="danger">
    El campo nombre debe tener como mínimo 10 caracteres
   </ion-note>
  }

 </ion-item>
```

Mostrando errores (refactorizado)

```
<ion-item>
   <ion-label position="stacked">Nombre</ion-label>
   <ion-input type="text" formControlName="nombre"></ion-input>

   @if (compruebaError('nombre', 'required')) {
    <ion-note slot="error">El campo nombre es requerido</ion-note>
   }

   @if (compruebaError('nombre', 'minlength')) {
    <ion-note slot="error">El campo nombre debe tener como mínimo 10 caracteres</ion-note>
   }
 </ion-item>
```

```
public compruebaError(campo: string, error: string): boolean {
  const control = this.formulario.get(campo);
  return !!control && control.hasError(error) && control.touched;
}
```

Validadores

Validador	Significado
`required`	Campo obligatorio
`maxlength`	Longitud máxima. Recibe un entero
`minlength`	Longitud mínima. Recibe un entero
`email`	El campo es un email
`max`	Valor máximo. Recibe un entero
`min`	Valor mínimo. Recibe un entero
`pattern`	Patrón. Recibe expresión regular