DESARROLLO DE APLICACIONES CON ANGULAR

Angular

ANGULAR

- Formularios. Enfoques:
  - **TEMPLATE-DRIVEN**.
  - **REACTIVE**.
- Capacidades comunes:
  - Permitir la entrada de datos.
  - Validaciones.
  - Creación de un modelo de datos.
  - Capacidad de detección de cambios.

https://angular.io/guide/forms-overview

ANGULAR

- Formularios. Enfoques:
  - **TEMPLATE-DRIVEN**. Más rápidos y sencillos de crear. Válidos para formularios que no tengan complicaciones y que no vayan a escalar.
  - **REACTIVE**. Más escalables, reutilizables y tiene una mayor capacidad respecto a las validaciones.

https://angular.io/guide/forms-overview

ANGULAR

Key differences

The following table summarizes the key differences between reactive and template-driven forms.

	REACTIVE	TEMPLATE-DRIVEN
Setup of form model	Explicit, created in component class	Implicit, created by directives
Data model	Structured and immutable	Unstructured and mutable
Data flow	Synchronous	Asynchronous
Form validation	Functions	Directives

ANGULAR

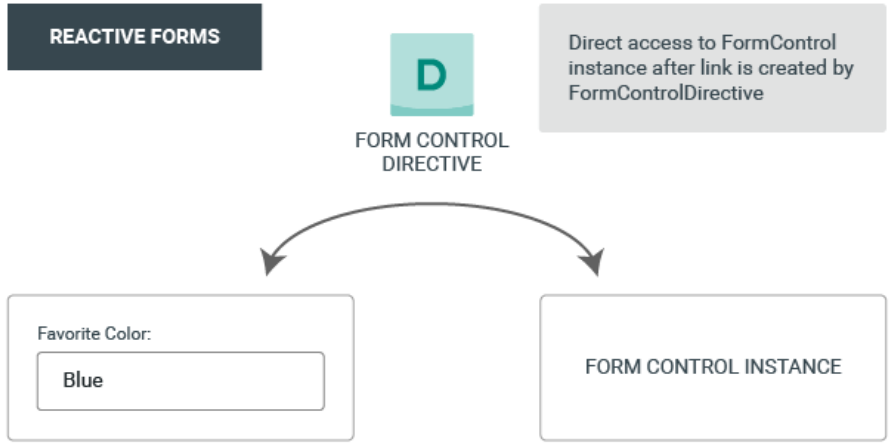
- Formularios. Elementos comunes:
  - FormControl. Controla el valor y el estado de un control de formulario individual.
  - FormGroup. Controla los valores y los estados de una colección de controles de formulario.
  - FormArray. Controla los valores y estados de un array de controles de formulario.
  - ControlValueAccesor. Crea un enlace entre el FormControl y los elementos del DOM.

ANGULAR

- Formularios reactivos:
  - Requieren una declaración explícita del FormControl en la clase del componente.
  - Permiten acceder al componente además de a su modelo de datos.

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

@Component({
  selector: 'app-reactive-favorite-color',
  template: `
    Favorite Color: <input type="text" [formControl]="favoriteColorControl">
  `
})
export class FavoriteColorComponent {
  favoriteColorControl = new FormControl('');
}
```



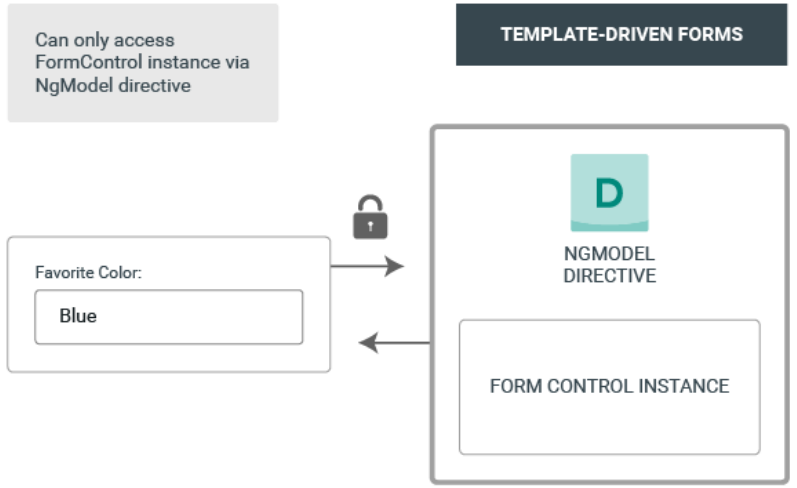
ANGULAR

Formularios template:

- **NO** requieren una declaración explícita del FormControl en la clase del componente.
- **NO** permiten acceder al componente además de a su modelo de datos.

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

@Component({
  selector: 'app-template-favorite-color',
  template: `
    Favorite Color: <input type="text" [(ngModel)]="favoriteColor">
  `
})
export class FavoriteColorComponent {
  favoriteColor = '';
}
```



ANGULAR

- Formularios reactivos:
  - **El flujo de datos**

ANGULAR

- Formularios template:
  - **El flujo de datos**

ANGULAR

FORMULARIOS REACTIVOS O MODEL-DRIVEN

https://angular.io/guide/reactive-forms

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - Agregar en app.module.ts:
    - import { ReactiveFormsModule } from '@angular/forms'
    - Incluir dentro de la sección imports.

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - En el html, se vinculan formulario y campos input con el formulario del TS:
    - En el form → [formGroup]="formulario"
    - En el input → formControlName="id"
    - En el input -> formControlName="password"
  - Crear el input de tipo submit.
  - {{formulario.value | json}} →Muestra los valores.

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - En el componente:
    - Atributo de tipo FormGroup → formulario:FormGroup;
    - En el constructor se crea el formulario con todos los componentes.

```
constructor() {
  this.formulario = new FormGroup({
    id: new FormControl(),
    password: new FormControl()
  }
```

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - Acceso a los valores de los campos.
    - Lectura:
      - `this.formulario.controls["email"].value`
    - Escritura:
      - `this.formulario.controls["email"].setValue("Nuevo valor");`

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - Valor por defecto:

```
nombre:new FormControl('Valor por defecto')
```

  - Validación:

```
nombre:new FormControl('',[
   Validators.required,
   Validators.minLength(longitud),
   Validators.maxLength(longitud),
   Validators.email,
   Validators.pattern(/reg-exp/)
]
```

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - Validación:

```
class Validators {
  static min(min: number): ValidatorFn
  static max(max: number): ValidatorFn
  static required(control: AbstractControl): ValidationErrors | null
  static requiredTrue(control: AbstractControl): ValidationErrors | null
  static email(control: AbstractControl): ValidationErrors | null
  static minLength(minLength: number): ValidatorFn
  static maxLength(maxLength: number): ValidatorFn
  static pattern(pattern: string | RegExp): ValidatorFn
  static nullValidator(control: AbstractControl): ValidationErrors | null
  static compose(validators: ValidatorFn[]): ValidatorFn | null
  static composeAsync(validators: AsyncValidatorFn[]): AsyncValidatorFn | null
}
```

https://angular.io/api/forms/Validators

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - Validación:
    - La validación ser manifiesta mediante la asignación de clases a los componentes. Estas clases cambian en función de si la validación se cumple o no:
      - Clases CSS de los inputs validados:
        - `ng-pristine/ng-dirty` → No ha sido modificado
        - `ng-untouched/ng-touched` → Indica si el campo ha recibido el foco o no.
        - `ng-valid/ng-invalid` → Es válido o no.

https://angular.io/api/forms/NgControlStatus

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - Validación:

Description

CSS classes applied

The following classes are applied as the properties become true:

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

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - Existen propiedades equivalentes a las clases CSS para interrogar al formulario y saber su estado:
    - *nombre-formulario.propiedad*
    - valid
    - invalid
    - dirty
    - pristine
    - touched
    - untouched

```
<button (click)="procesar()" [disabled]="formulario.invalid">Procesar</button>
```

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - También se puede acceder al estado de cualquier control del formulario:
    - *nombre-formulario.*controls*[nombre].propiedad*
    - *Ejemplo:*
      - `this.formulario.controls["nombre"].dirty`

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - Validación personalizadas:
    - A través de una función.

```
password: new FormControl('',
        [
          Validators.required,
          diccionarioValidator()
        ]),
```

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - Validación personalizadas.
    - Función:
      - Devuelve una función de validación.
        - La función de validación:
          - Tiene como entrada un elemento AbstractControl.
          - Devuelve un objeto ValidationErrors o null.

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - Validación personalizadas.
    - Función. Ejemplo.

```
export function diccionarioValidator(): ValidatorFn {
  const error_object = {error_message:"Palabra prohibida"}
  return (control: AbstractControl): ValidationErrors | null => {
    const palabras: string[] = ["alacena", "barril", "cubierta"];
    /*
    if (palabras.includes(control.value)) {
      return error_object
    }
    return null;
    */
    return (palabras.includes(control.value)) ? error_object : null;
  };
}
```

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - Mostrar errores:
    - Podemos condicionar la presentación de los contenedores de mensajes de error (p.e.):
      - `<div *ngIf="formulario.controls['email'].errors && formulario.controls['email'].touched">`

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - Mostrar errores:
    - La variable errors proporciona información referente al tipo de error.
      - Gestionando el contenido de errors se pueden personalizar los mensajes.

```
<div *ngIf="formularioLogin.controls['password'].errors?.['required']" class="mensaje-error">
        Contraseña obligatoria
</div>
<div *ngIf="formularioLogin.controls['password'].errors?.['minlength']" class="mensaje-error">
     Contraseña debe tener
     {{formularioLogin.controls['password'].errors?.['minlength']?.['requiredLength']}}
</div>
```

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - Ejemplos
    - Mensaje de advertencia condicionado a la aceptación de un checkbox:

```
<div *ngIf="formularioLogin.controls['condenado'].errors">
        Debe aceptar las condiciones del servicio.
</div>
```

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - Ejemplos
    - Botón de aceptar (submit) que se activa cuando el formulario al completo es válido:

```
<button [disabled]="!formularioLogin.valid“>Aceptar</button>
```

Botón de aceptar (submit) que cambia de color cuando el formulario es válido:

```
<button[ngClass]="{'red':!formularioLogin.valid}">Aceptar</button>
```

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - Validaciones en tiempo real:
    - Obtenemos la referencia la control.
    - Acceder a la propiedad valueChange (es un Observable) mediante suscripción. Recibirá cualquier cambio del valor del componente.

```
ngOnInit():void {
  const variable = this.formulario.controls.control-name;
  //const variable = this.formulario.controls['control-
  name'];//Alternativa
  variable.valueChanges.subscribe(value=> {
    //tratamiento del value
  }
```

    - Utilizando debounceTime se puede retardar la gestión del cambio del valor del input.

ANGULAR

- Formularios tipo MODEL DRIVEN.
  - Limpieza de formularios:
    - Los formularios se pueden limpiar desde código de manera global o a través de componentes individuales:
      - De manera global mediante el método reset():
        - `this.formulario.reset();`
      - De manera individual accediendo a los controles por su nombre y asignando un valor vacío:
        - `this.formulario.controls["nombre"].setValue("");`

ANGULAR

- Formularios: Local Storage
  - Local Storage es un espacio de almacenamiento local al navegador.
  - Se accede mediante pares clave-valor a través del objeto *localStorage*, mediante el método *setValue* y el atributo de solo lectura *value.*
    - Almacenamiento:

```
localStorage.setItem("email",this.formulario.controls["email"].value);
```

    - Recuperación:

```
if (localStorage.getItem("email")!=null){
      this.formulario.controls['email'].setValue(localStorage.getItem("email"));
}
```

ANGULAR

FORMULARIOS DE TIPO TEMPLATE

ANGULAR

- Formularios tipo TEMPLATE.
  - Agregar en app.module.ts:
    - import { FormsModule } from '@angular/forms'
    - Incluir dentro de la sección imports.
  - Los formularios deben de estar delimitados por <form>
    - <form #identificadorFormulario="ngForm" (ngSubmit)="método(identificadorFormulario.value)">
  - Hay que indicar los elementos input que forman parte del formulario mediante ngModel y el atributo name:
    - <input type="text" name="nombre" ngModel>

ANGULAR

- Formularios tipo TEMPLATE.
  - El método en el componente:
    - nombreMétodo(valores) → Recibe todos los ngModel del formulario.