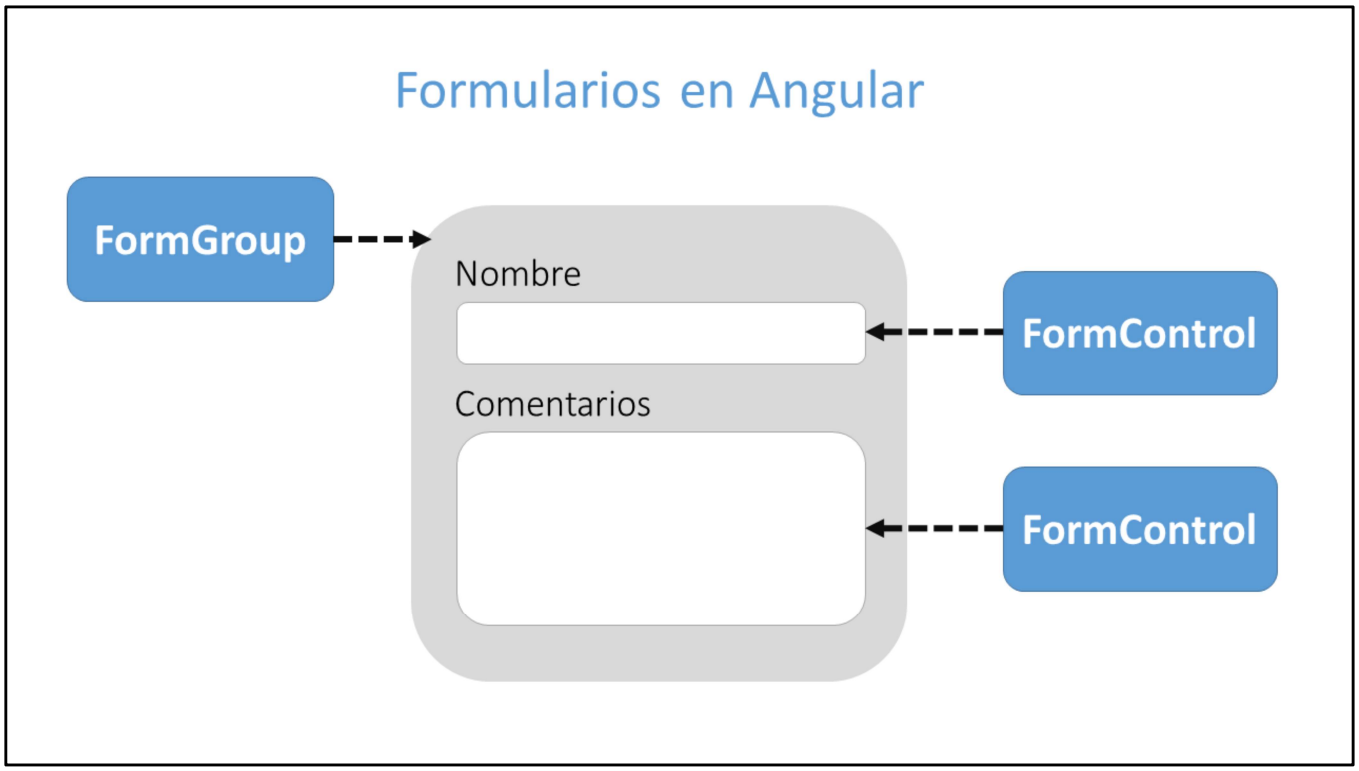
Tema 6 – Formularios dirigidos por plantilla

- Formularios en Angular
- Formularios dirigidos por plantilla
- Validaciones y manejo de errores
- Submit y recogida de datos
- Data binding de doble sentido
- Grupos de controles en un formulario.
- Controles habituales (Angular Material)

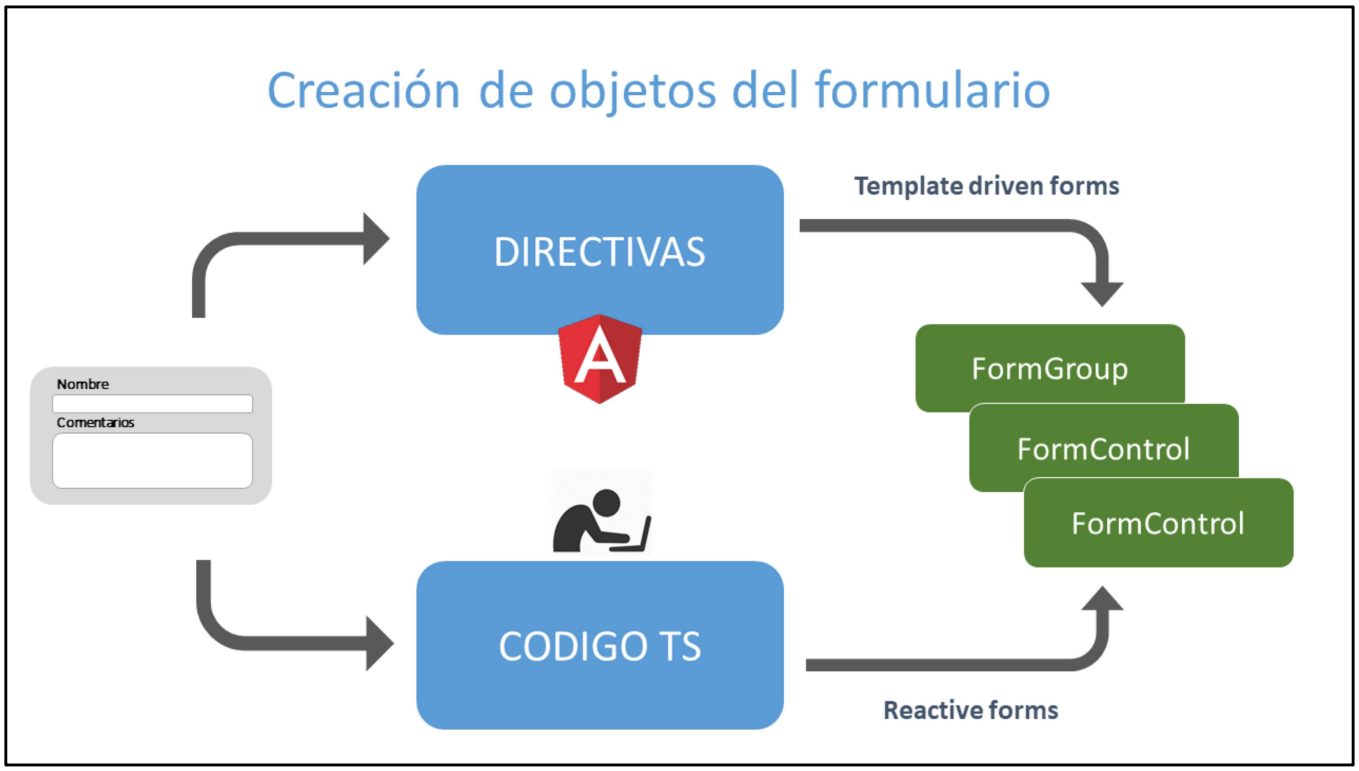


Los formularios en Angular no dejan de ser formularios HTML.

La diferencia es que tanto el propio formulario como los elementos del mismo, deben asociarse a objetos a los que accederemos en nuestro componente, para controlar el formulario y acceder a la información del mismo.

- Un formulario HTML se asocia a un objeto de tipo FormGroup;
- Cada control del formulario se asocia a un objeto de tipo FormControl.

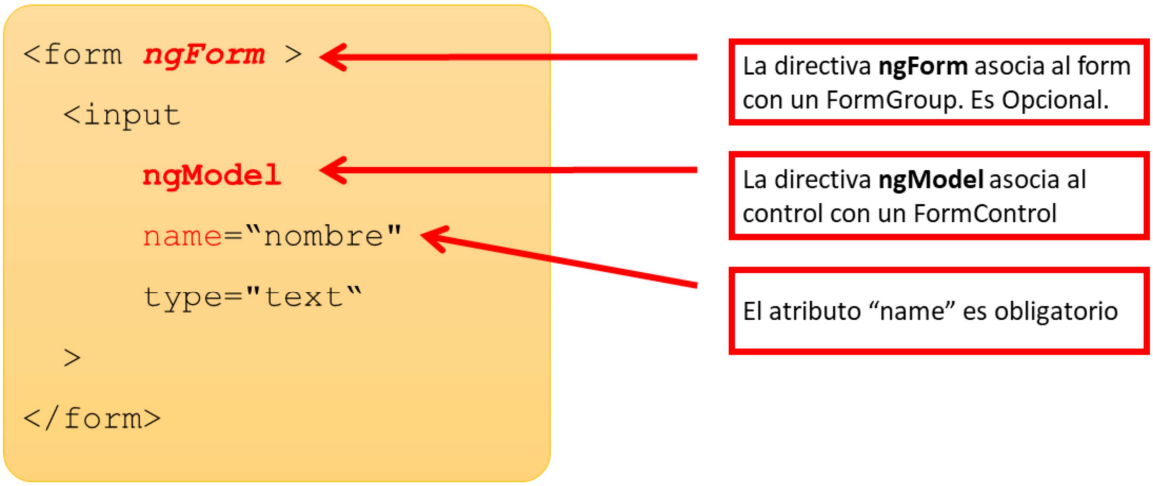
Estos objetos, a través de sus métodos y propiedades, nos permiten acceder y controlar el formulario en el componente.



¿Cómo creamos los objetos FormGroup y FormControl? Angular tiene 2 formas de hacerlo:

- Mediante directivas en la plantilla HTML. Se denominan formularios dirigidos por plantilla, o "template-driven forms". Están recomendados para formularios simples con validaciones básicas.
- Mediante código TS en el componente. Se denominan Formularios Reactivos, o "reactive forms" . Están recomendados para formularios y validaciones más complejos.

Formularios dirigidos por plantilla



La directiva **ngForm** asocia al form con un FormGroup. Por defecto Angular lo asocia a todos los elementos “form” de la plantilla, así que es opcional especificarlo.

Para que Angular cree el FormControl, se añade la directiva atributo "**ngModel**" al elemento HTML.

Es obligatorio especificar el atributo “name” de los controles con dicha directiva, ya que será a través de dicho nombre cómo obtengamos después su valor en el componente.

Validaciones y manejo de errores

- Los formularios dirigidos por plantilla admiten todas las validaciones estándar de HTML5:
  - *required, min, max, minlength, maxlength, pattern, step*

- No es posible usar validadores personalizados en formularios dirigidos por plantilla, pero sí en formularios reactivos
  - … no obstante, se pueden simular mediante manejadores de eventos

Validaciones y manejo de errores

- Los objetos FormGroup y FormControl tienen propiedades útiles para determinar su estado y su validez:
  - *valid / invalid*
    - *Validaciones se cumplen o no (incluyendo hijos)*
  - *touched / untouched*
    - *El usuario ha entrado o no al form o control*
  - *dirty / pristine*
    - *El form o control ha sido modificado o no*

Validaciones y manejo de errores

- Validación básica y mensaje de error:

```
<input
   ngModel
   #nombre="ngModel"
   name="nombre"
   type="text"
   required
>

<div *ngIf="nombre.touched && nombre.invalid"
class="alert alert-danger">
   El nombre es obligatorio
</div>
```

Variable de plantilla (#) para referenciar al objeto FormControl asociado

Propiedades “touched” e “invalid” del objeto FormControl asociado.
El div se muestra solo si el control ha sido accedido y no es válido

Validaciones y manejo de errores

- Validación múltiple y mensajes de error:

```
<input ngModel #nombre="ngModel" name="nombre" id="nombre" type="text"
required minlength=3>

<div *ngIf="nombre.invalid && nombre.touched" class="alert alert-
danger">
   <div *ngIf=" nombre.errors && nombre.errors.required">El nombre es
   obligatorio</div>
   <div *ngIf="nombre.errors?.minlength">
      La longitud minima es de
      {{ nombre.errors.minlength.requiredLength }} caracteres, pero
      tiene solo {{ nombre.errors.minlength.actualLength }}
   </div>
</div>
```

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

Si tenemos varias validaciones en un control, no nos vale simplemente saber si su estado es válido o no, posiblemente queramos saber qué validaciones en concreto no se cumplen.

Por ejemplo, si queremos mostrar errores personalizados en un control con 2 validaciones, requerido y longitud minima.

En el ejemplo tenemos un div general en donde vamos a incluir otros 2 div específicos para cada error posible. Este div general se mostrará, igual que antes, si el control ha sido accedido y no es válido.

Para controlar la visibilidad de los 2 div internos, vamos a chequear una propiedad de FormControl que se llama "**errors**". Esta propiedad contiene en tiempo de ejecución, un objeto con un atributo por cada uno de los errores de validación que tiene el control.

Si no hay errores, el atributo "errors" no está presente. Por tanto lo que hacemos es primero comprobar si existe "errors" en el control, y después comprobar si el objeto "errors" tiene el atributo "minlength".

Fijarse en como, para el segundo div, simplificamos la comprobación usando el operador ?., llamado "Safe Transversal Operator", u Operador transversal seguro. Es igual que lo

anterior, pero no provoca un error si “nombre.errors” es nulo.

El valor de los atributos de “errors”, depende del tipo de validación. Para “required” es true o false. Pero para “minlength” es un objeto que contiene a su vez 2 propiedades: requiredLength y actualLength, con los valores de la longitud requerida y la real. Para ver la estructura de cada tipo de error, ver https://angular.io/api/forms/Validators

Validaciones y manejo de errores

- Activar o desactivar el botón según la validez:

```
<form ngForm #f="ngForm">
...
   <button [disabled]="f.invalid"
   class="btn btn-
   primary">Submit</button>

</form>
```

Debemos agregar una variable de plantilla que referencie al FormGroup...

... para poder comprobar el estado valido o invalido del formulario en su conjunto

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

Vamos a activar o desactivar el botón, en base a la validez del formulario, usando binding del atributo “disabled” del botón.

Igual que usábamos #nombre=“ngModel” para obtener una referencia al FormControl, usamos #f=“ngForm” para obtener una referencia al FormGroup asociado al formulario.

La propiedad valid / invalid del formulario se calcula en base a la validez o no de todos los controles del formulario. Dicho de otra forma, form.valid es cierto solo si la propiedad valid de todos los objetos FormControl anidados es cierto.

Submit del formulario

- El FormGroup creado con “ngForm” tiene un evento “ngSubmit”, similar al “onsubmit” de HTML

HTML

```
<form #f="ngForm" (ngSubmit)="enviarForm(f)"
```

TS

```
enviarForm(f: NgForm){
     if (f.valid) console.log("OK");
}
```

En Angular, a diferencia de HTML normal, el submit de un formulario no implica el envío del formulario al servidor.

Capturamos el evento especial "ngSubmit" para capturar el evento "submit" del formulario Angular.

Recogida de valores de un formulario

Cómo recoger los valores del formulario:

A. Mediante el atributo “value” del FormGroup:

TS

```
enviarForm(f){
   let valorNombre = f.value.nombre;
}
```

B. Accediendo directamente a los objetos FormControl

TS

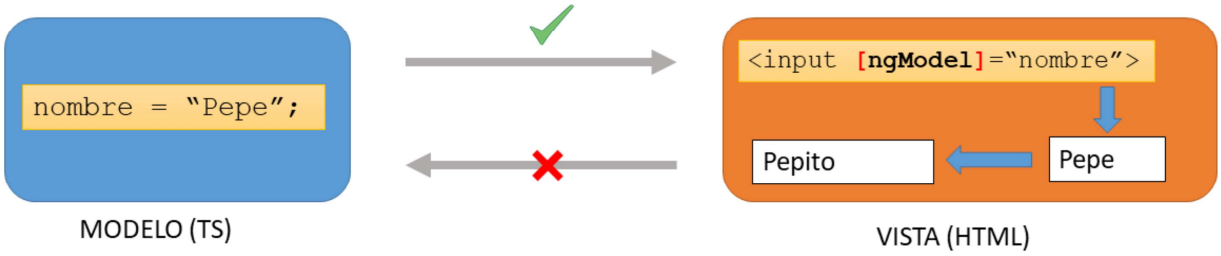
```
enviarForm(f){
   let valorNombre = f.form.get("nombre").value;
}
```

El objeto FormGroup tiene la propiedad "value", que es un objeto con un atributo por cada uno de los objetos FormControl. El nombre del atributo es el "name" del control, y el valor la propiedad "value".

Es importante ver cómo accedemos en el componente al objeto FormGroup que ha generado Angular. Los tenemos que pasar como argumentos en las llamadas desde la plantilla. Por eso a este tipo de formularios se les denomina "dirigidos por plantilla".

Binding entre componente y formulario

- Mediante property binding de **ngModel**, podemos controlar el valor de un FormControl desde el componente. Al modificar el modelo, se actualiza la vista.
- **[ngModel]** es un binding en un solo sentido. Cambios en la vista NO actualizan el modelo.
  - Por tanto al modificar el "input", no se modifica el valor en el modelo.



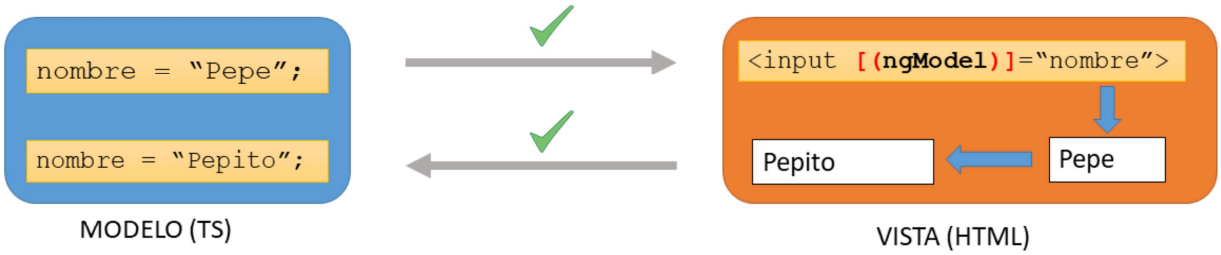
MODELO (TS)

VISTA (HTML)

El binding de propiedades es un binding de un solo sentido. ¿Qué quiere decir esto? Que la vista responde a los cambios del modelo, pero el modelo no se modifica si el control visual modifica el dato.

Binding de doble sentido

- **[(ngModel)]** → Binding de doble sentido
  - Cambios en el modelo actualizan la vista
  - Cambios en la vista actualizan el modelo



[( )] → Banana in a box

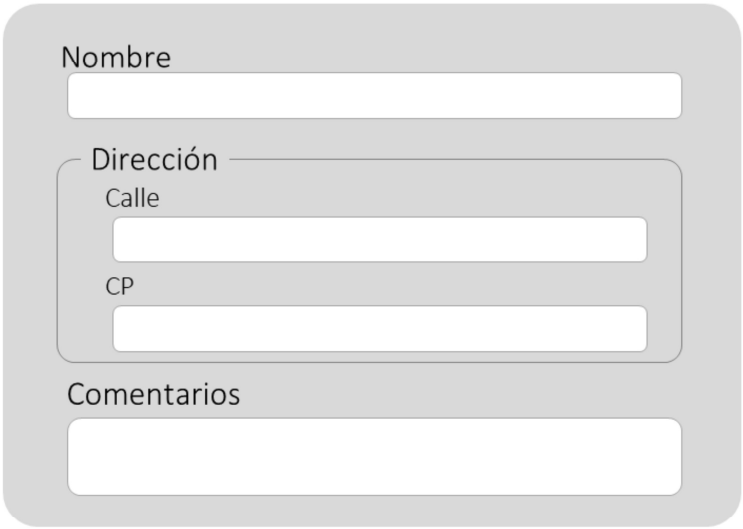
Para conseguir que el modelo y la vista estén sincronizados en ambos sentidos, es necesario utilizar el binding de doble sentido.

En lugar de usar simples corchetes, usamos la notación corchetes y paréntesis, que se suele denominar "banana in a box", por su forma.

De esta forma, los cambios en el modelo actualizan la vista, y los cambios en la vista actualizan el modelo.

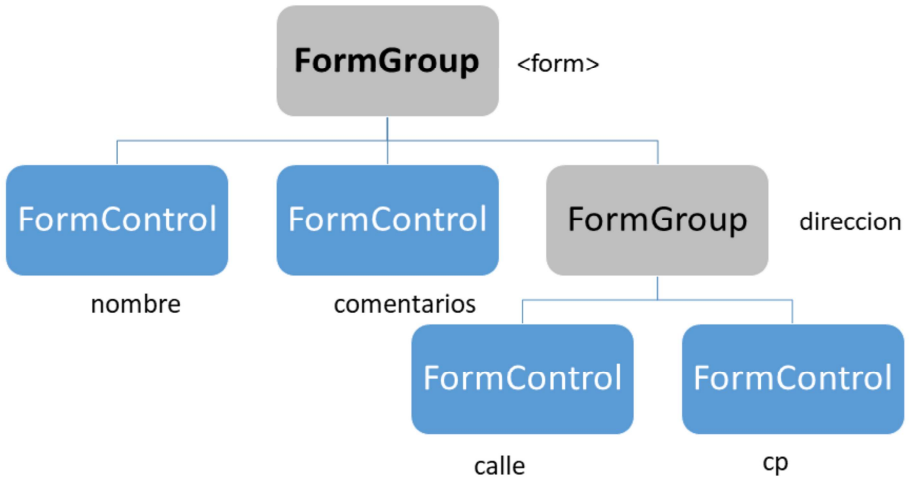
Este comportamiento es perfecto para los formularios, ya que evita tener que escribir código para inicializar y recoger los valores de los controles. Basta con hacer el binding, y en el componente usar las propiedades enlazadas.

Grupos de controles en un formulario



Vamos a modificar el formulario para hacerlo más complejo, añadiendo un grupo de controles para la dirección, con la direccion y el código postal

Árbol de objetos con grupos de controles



- Podemos crear agrupaciones de controles en formularios complejos. Es decir, los FormGroup pueden estar anidados a partir del FormGroup principal del formulario.

- Esta estructura en árbol es útil, por ejemplo, para las validaciones:
    - Los objetos de tipo FormGroup tienen las mismas propiedades valid/invalid que los controles.
    - Su valor se calculará en base a todos los controles que están agrupados. Será válido solo si todos los controles del grupo son validos. Este comportamiento se transmite hacia arriba en el árbol de FormGroup, de forma que el propio formulario será valido solo si todos sus grupos y controles lo son.

Grupos de controles en un formulario

- Directiva ngModelGroup:

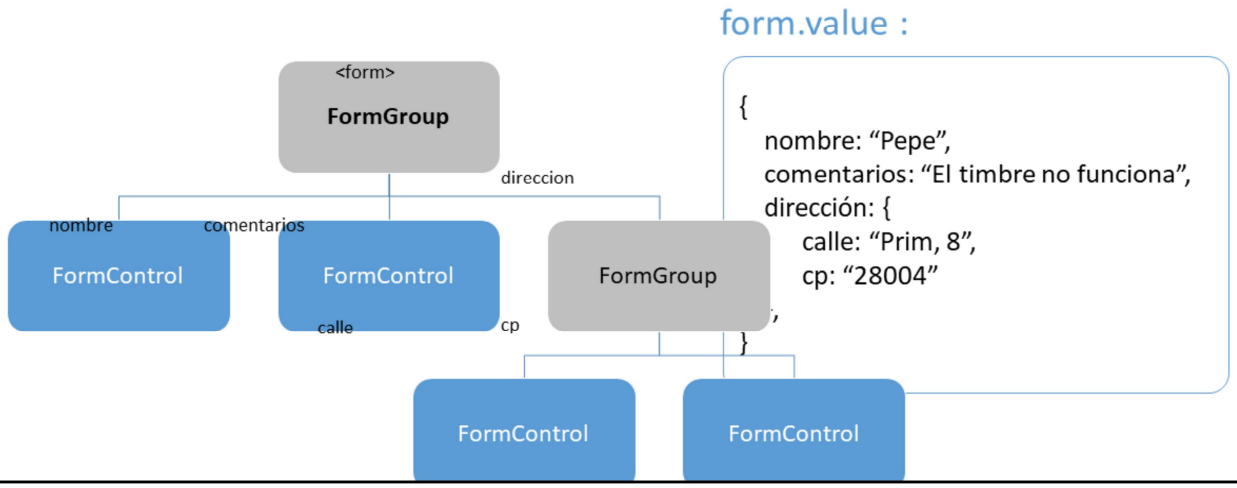
```
<form>
...
   <div ngModelGroup="direccion">
      <div>
         <label for="calle">Calle</label>
         <input ngModel name="calle" type="text">
         <label for="CP">CP</label>
         <input ngModel name="CP" type="text">
      </div>
   </div>
...
</form>
```

Creamos grupos de controles, con la directiva – atributo “ngModelGroup”, .asignándole un nombre al grupo.

La directiva "ngModelGroup" es igual que ngModel, pero en lugar de crear un objeto FormControl, crea uno FormGroup.

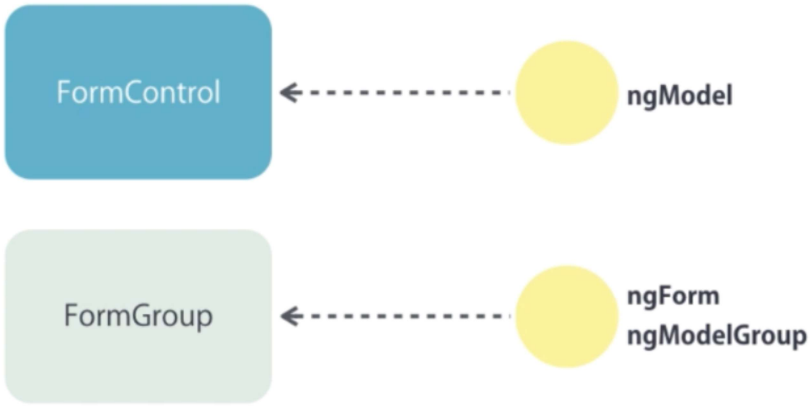
Grupos de controles en un formulario

- El valor de "value" en el form refleja la estructura del formulario:



La estructura del objeto form.value refleja la misma estructura de objetos FormGroup y FormControl que hemos puesto en la plantilla.

Resumen de directivas y clases



Recapitulando:

- FormGroup y FormControl son las clases que usa Angular para guardar el estado interno de los formularios y controles que hay en la plantilla HTML.
- Por cada etiqueta "form", se crea un objeto de tipo FormGroup, usando implicitamente la directiva "ngForm".
- Por cada control con el atributo "ngModel", se crea un objeto de tipo FormControl.
- Por cada elemento con el atributo "ngModelGroup", se crea un objeto FormGroup.
- La diferencia entre el FormGroup de ngForm y ngModelGroup, es que con ngForm, se crea un evento "ngSubmit" para el submit del formulario asociado.

Controles habituales (Angular Material)

- Cuadros de texto
- Checkboxes
- Listas desplegables (select)
- Radio buttons

Controles habituales (Angular Material)

- Cuadros de texto

```
import {MatInputModule} from '@angular/material/input';
import {MatFormFieldModule} from '@angular/material/form-field'
```

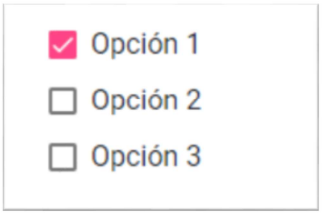
```
<form>
    <mat-form-field>
        <mat-label>Nombre</mat-label>
        <input
            matInput
            ngModel
            name="nombre">
    </mat-form-field>
</form>
```

Controles habituales (Angular Material)

- Checkbox

```
import {MatCheckboxModule} from '@angular/material/checkbox'
```

```
<form>
    <mat-checkbox checked>Opción 1</mat-
    checkbox>
    <mat-checkbox>Opción 2</mat-checkbox>
    <mat-checkbox>Opción 3</mat-checkbox>
</form>
```

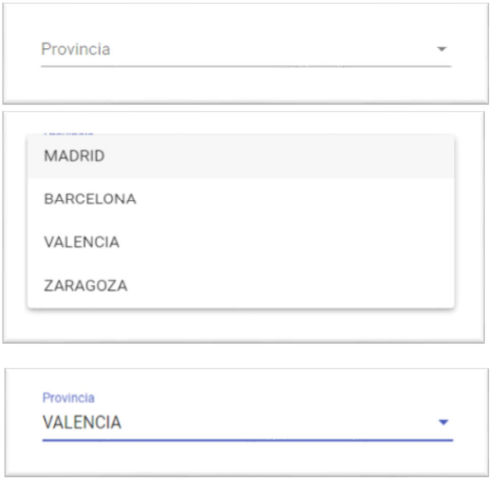


Controles habituales (Angular Material)

- Listas desplegables (select)

```
import {MatSelectModule} from '@angular/material/select';
```

```
<form>
    <mat-form-field>
        <mat-label>Provincia</mat-label>
        <mat-select>
            <mat-option value="MADRID">MADRID</mat-option>
            <mat-option value="BARCELONA">BARCELONA</mat-
            option>
            <mat-option value="VALENCIA">VALENCIA</mat-option>
            <mat-option value="ZARAGOZA">ZARAGOZA</mat-option>
        </mat-select>
    </mat-form-field>
</form>
```



Controles habituales (Angular Material)

- Radio buttons

```
import {MatRadioModule} from '@angular/material/radio';
```

```
<form>
  <mat-radio-group class='d-flex flex-column'>
    <mat-radio-button value="1" checked=true>Opción 1</mat-
radio-button>
    <mat-radio-button value="2">Opción 2</mat-radio-button>
  </mat-radio-group>
</form>
```