El concepto de Angular Select es bastante amplio y hace referencia a cómo podemos gestionar el concepto de combo , select o desplegable dentro de una aplicación moderna de Angular algo que ha cambiado bastante desde la versión clásica . Los ejemplos más elementales son sencillos de utilizar sin embargo existen algunos ejemplos que pueden ser un poco más complejos y no tan sencillos de abordar por el desarrollador vamos a verlos . Para ello partiremos del ejemplo más básico creando un componente sencillo que incluya un select.

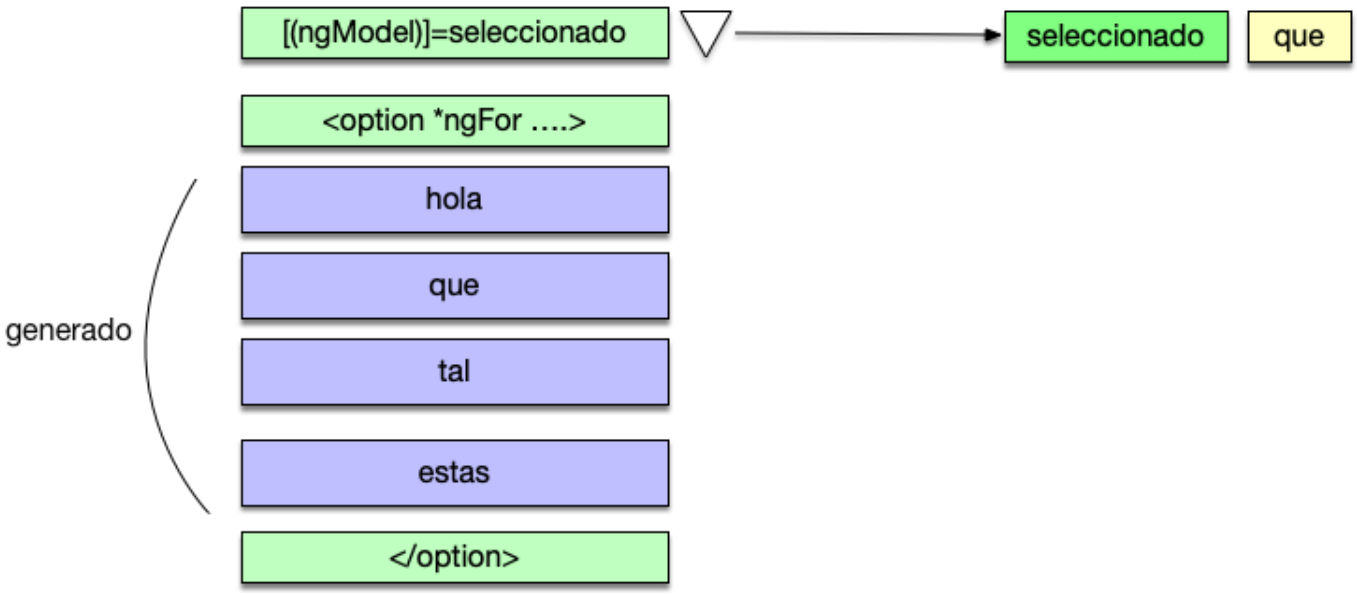
```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-c1',
  templateUrl: './c1.component.html',
  styleUrls: ['./c1.component.less']
})
export class C1Component implements OnInit {

lista:string[]=["hola","que","tal", "estas"];

  constructor() { }

  ngOnInit() {
  }

}
```

En este caso tenemos un componente que contiene una lista de elementos. Esta lista de elementos son simples textos. Si queremos referenciar a ellos en nuestro código y que el select funcione de forma correcta, no tenemos nada más que hacer lo siguiente:

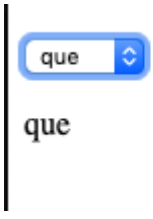
```
<p>
<select name="miselect" [(ngModel)]="seleccionado">
```

```
<option [value]="item" *ngFor="let item of lista">{{item}}</option>
</select>
</p>
{{seleccionado}}
```

En este caso acabamos de aplicar dos acciones. En primer lugar una directiva *ngFor que presenta en el select todos los elementos a nivel de options en segundo lugar un binding bidireccional que define una propiedad que almacena el elemento seleccionado.



Esto implicará que cuando nosotros en el select seleccionamos un elemento pues automáticamente queda elegido a nivel de ngModel y lo podemos imprimir usando interpolación con {{seleccionado}}

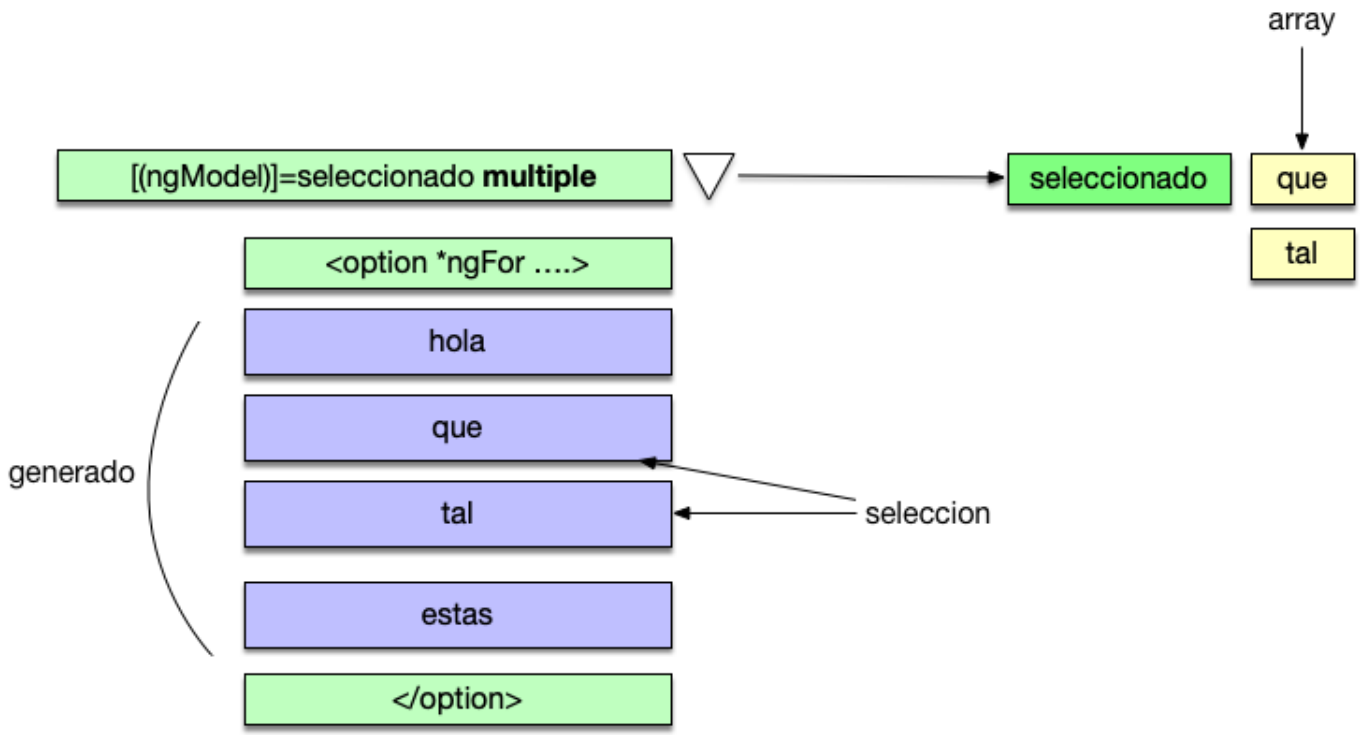


Angular Select Multiple

Otra de las opciones que tenemos es diseñar un select múltiple es decir un select en el cual se puedan elegir varias opciones de forma simultanea para ello activaremos la opción de multiple a nivel del código html:

```
<p>
    <select name="miselect" [(ngModel)]="seleccionados"
multiple="multiple">
    <option [value]="item" *ngFor="let item of
lista">{{item}}</option>
    </select>
    </p>

    <ul>
      <li *ngFor="let item of seleccionados">
        {{item}}
      </li>

    </ul>
```

Ademas de realizar esa operación añadiremos una lista de elementos seleccionados que aparezcan con una estructura de ul y li. De tal forma que en vez de tener un ngModel con un único elemento seleccionado tendremos un ngModel con un Array de elementos algo que es bastante diferente.



Eso es lo que debemos de añadir a nivel de código de TypeScript.

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

@Component({
  selector: 'app-c2',
  templateUrl: './c2.component.html',
  styleUrls: ['./c2.component.less']
})
export class C2Component implements OnInit {

  lista:string[]=["hola","que","tal", "estas"];
  seleccionados:string[]=[];
  constructor() { }
```

```
  ngOnInit() {
  }

}
```

De esta forma cuando seleccionemos en el desplegable tendremos la opción se seleccionar varios . Esto de forma automática quedara ligado al array de cadenas que hemos definido como “seleccionados” y que a través de interpolación mostramos en una lista sin numerar.

- hola
- que

Angular Selecciones y Objetos

Es totalmente válido disponer de las configuraciones anteriores , pero la realidad es que en muchas ocasiones al ser Angular un framework orientado a datos es más común trabajar con listas de objetos complejos y mostrar en el select alguna de sus propiedades de forma directa . Vamos a ver un ejemplo de como podemos construir una solución de este tipo. Para ello nos vamos a definir la clase Ciudad esta clase contendrá un identificador , un nombre y una descripción.

```
export class Ciudad {

    constructor(public id?:string, public nombre?:string, public
descripcion?:string) {
```

```
    }
}
```

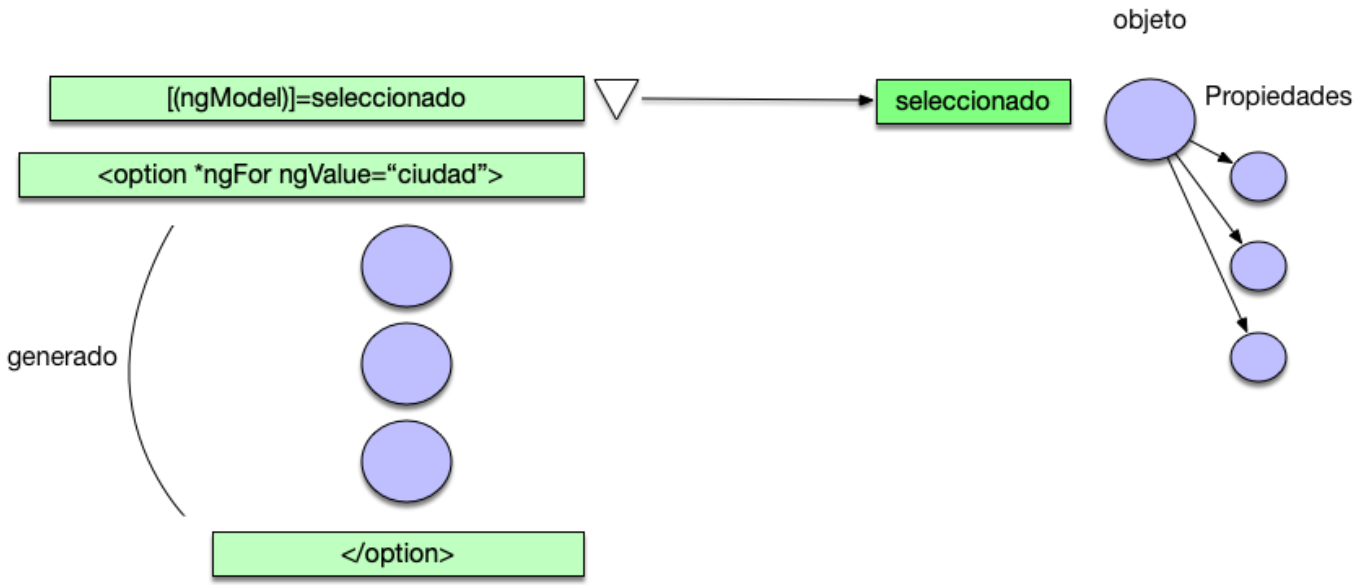
Hemos usado un constructor class expression para definir la clase . Es hora a usar el concepto de ciudad para generarnos un Angular Select que se nos permite gestionar una lista de objetos de forma rápida. Para ello nos generaremos una lista a nivel del propio componente.

```
import { Component, OnInit } from '@angular/core';
import { Ciudad } from '../ciudad';

@Component({
  selector: 'app-c3',
  templateUrl: './c3.component.html',
  styleUrls: ['./c3.component.less']
})
export class C3Component implements OnInit {

  lista:Ciudad[]=[];

  constructor() { }

  ngOnInit() {

    this.lista.push(new Ciudad("1","santander","ciudad al lado del
mar"))
    this.lista.push(new Ciudad("2","donosti","ciudad gastronomica"))
    this.lista.push(new Ciudad("3","bilbao","ciudad cultural"))
  }

}
```

Esta lista esta compuesta por objetos y debemos utilizar una plantilla para mostrar algunas de sus propiedades por pantalla.

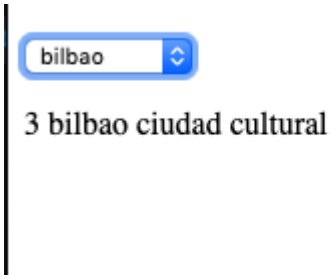
```
<p>
    <select name="miselect" [(ngModel)]="seleccionado">
    <option [ngValue]="objeto" *ngFor="let objeto of
lista">{{objeto.nombre}}</option>
    </select>
    </p>
    {{seleccionado?.id}}
    {{seleccionado?.nombre}}
    {{seleccionado?.descripcion}}
```

Nosotros seleccionemos un elemento del desplegable el cual ha realizado un binding a un objeto complejo no a una propiedad sencilla.



Demonos cuenta de la diferencia que existe entre usar [value] a nivel de property binding

.Lo que hace es enlazar una propiedad de tipo cadena o numero (es decir un tipo básico) a usar [ngValue] que nos permite enlazar un objeto complejo. De esta manera al seleccionar el nombre de la ciudad desplegaremos el objeto entero. Veamos cómo funciona a nivel de interface de usuario.



Como vemos seleccionamos un elemento de la lista que de entrada parece una cadena y lo que mostramos son todas las propiedades del objeto ya que es a lo que realmente hemos realizado binding.

Angular CompareWith

Existen situaciones en las que nos podemos encontrar con algunas cosas algo más complejas. Por ejemplo imaginémonos que disponemos de una lista de objetos como en este caso es el tema de las ciudades y ademas queremos que el objeto que tenemos a nivel de seleccionado salga marcado por defecto en el select cuando este se carga. Se trata de un código similar al anterior pero que tiene sus peculiaridades.

```
<p>
  <select name="miselect" [(ngModel)]="seleccionado"
[compareWith]="compararNombres">
    <option [ngValue]="objeto" *ngFor="let objeto of
lista">{{objeto.nombre}}</option>
  </select>
</p>
<p>
```

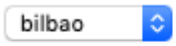
```
  {{seleccionado?.id}}
  {{seleccionado?.nombre}}
  {{seleccionado?.descripcion}}
</p>
```

En este caso para que el item salga marcado debemos construir una función "compararNombres" en el componente que nos permita indicar cuando dos items son iguales a la hora de compararlos . Vamos a ver su código:

```
import { Component, OnInit } from '@angular/core';
import { Ciudad } from '../ciudad';

@Component({
  selector: 'app-c4',
  templateUrl: './c4.component.html',
  styleUrls: ['./c4.component.less']
})
export class C4Component implements OnInit {

  seleccionado:Ciudad=new Ciudad("3","bilbao", "ciudad cultural");
  lista:Ciudad[]=[];


  constructor() { }

  ngOnInit() {


    this.lista.push(new Ciudad("1","santander","ciudad al lado del
mar"))
    this.lista.push(new Ciudad("2","donosti","ciudad gastronomica"))
    this.lista.push(new Ciudad("3","bilbao","ciudad cultural"))
```

```
  }

  compararNombres( ciudad1:Ciudad, ciudad2:Ciudad) {

    if (ciudad1==null || ciudad2==null) {
      return false;
    }
    return ciudad1.nombre===ciudad2.nombre;

  }

}
```

En este caso estamos seleccionando bilbao como ciudad por defecto que debe salir marcada . Una vez la tengamos en la variable seleccionado nos queda hacer uso de el método compararNombres que informa a Angular cuando dos items son idénticos.



3 bilbao ciudad cultural

Nada mas cargar la página nos aparece bilbao seleccionado como opción por defecto ya que es la coincidente. Acabamos de ver las diferentes opciones que Angular soporta a nivel de la gestión de los selects.

Otros artículos relacionados

1. Angular ngIf, opciones y componentes
2. Angular Lazy Loading Modules y sus opciones
3. Angular Sorting , extensibilidad con lodash

4. Angular ngFor la directiva y sus opciones
5. https://angular.io/