	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN	
CICLO: 02	GUIA DE LABORATORIO #05	
	Nombre de la Práctica:	Angular, parte II - crudAngularFirestore
	MATERIA:	Diseño y Programación de Software Multiplataforma

I. OBJETIVOS

Que el estudiante:

- Diseñe aplicaciones web utilizando funciones de Angular.
- Diseñe aplicaciones con navegación.
- Aprender hacer interfaz para que el usuario pueda interactuar con la aplicación.
- Aprender a crear componente personalizado.
- Hacer uso de binding en el diseño de interfaz angular.
- Aprender hacer interfaz para que el usuario pueda interactuar con la aplicación.
- Aprender a utilizar pipes dentro de la interfaz de usuario.
- Hacer uso de los componentes en el diseño de interfaz angular.

Contenido

Herramienta Angular CLI - comando: ng new	3
Herramienta Angular CLI - comando: ng generate.....	9
ng generate component	9
ng generate service	11
ng generate module.....	12
ng generate pipe	12
ng generate class	13
ng generate interface	13
ng generate enum	14
Herramienta Angular CLI - comando: ng serve.....	14
Opción --port.....	14
Opción --watch (alias: -w)	14
Opción --prod.....	15
Directiva estructural *ngFor - Sintaxis completa.....	15
Su sintaxis tiene partes obligatorias:	15
Su sintaxis extendida:.....	16
Microsintaxis	17
Directiva estructural *ngIf - Sintaxis completa	18
Directiva *ngIf con else	20
Uso de operadores lógicos.....	20
Llamada a un método.	21
Variante de sintaxis con then else.	21
Microsintaxis	22
Directivas estructurales *ngSwitchCase, *ngSwitchDefault y ngSwitch	23
Microsintaxis	25
Creación Crud Angular – Firebase.....	27

II. INTRODUCCION TEORICA

Herramienta Angular CLI - comando: ng new

Hemos visto que con el comando 'new' de la herramienta Angular CLI se crea un esqueleto de una aplicación Angular:

```
ng new proyecto017
```

Durante la generación del proyecto se nos consulta si queremos disponer rutas y el tipo de archivos para las hojas de estilo.

El comando new podemos pasar otros parámetros para adaptar la aplicación generada.

Si necesitamos administrar rutas en nuestra aplicación podemos crearla con la siguiente sintaxis (luego no se nos consulta en la generación del proyecto):

```
ng new proyecto017 --routing
```

Se crea el archivo 'app-routing.module.ts' donde debemos configurar las rutas, tema visto anteriormente.

Podemos especificar el prefijo a los selectores generados mediante el parámetro --prefix:

```
ng new proyecto017 --prefix ele
```

Luego se crea con prefijo 'ele' en lugar del valor por defecto 'app':

Si abrimos la componente creada por defecto podemos ver que el prefijo es 'ele':

```
@Component({  
  selector: 'ele-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})
```

En lugar de 'app':

```
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',
```

```
styleUrls: ['./app.component.css']
})
```

Luego si creamos una componente dentro del proyecto también se respetará el prefijo definido en la creación del proyecto:

ng generate component dado

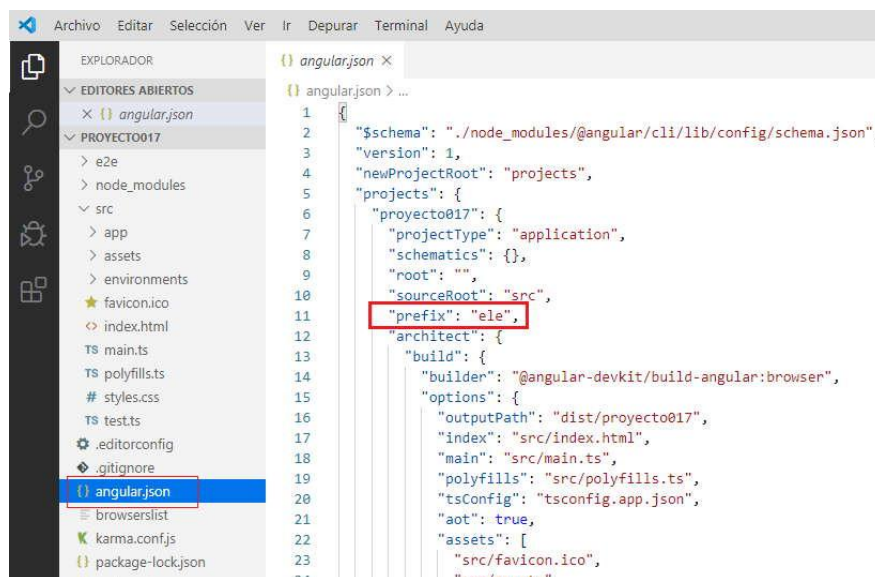
El selector de la componente dado queda con el siguiente nombre:

```
@Component({
  selector: 'ele-dado',
  templateUrl: './dado.component.html',
  styleUrls: ['./dado.component.css']
})
```

La definición de prefijos en las componentes de Angular permiten diferenciar las etiquetas nativas del navegador (ej. 'button') con las etiquetas propias de Angular (ej. ele-button)

Al crearse un proyecto uno de los archivos fundamentales que almacenan datos de configuración del mismo es 'angular.json' (se encuentra en la carpeta raíz del proyecto)

En el archivo 'angular.json' se encuentra almacenado el prefijo definido al crearse el proyecto:



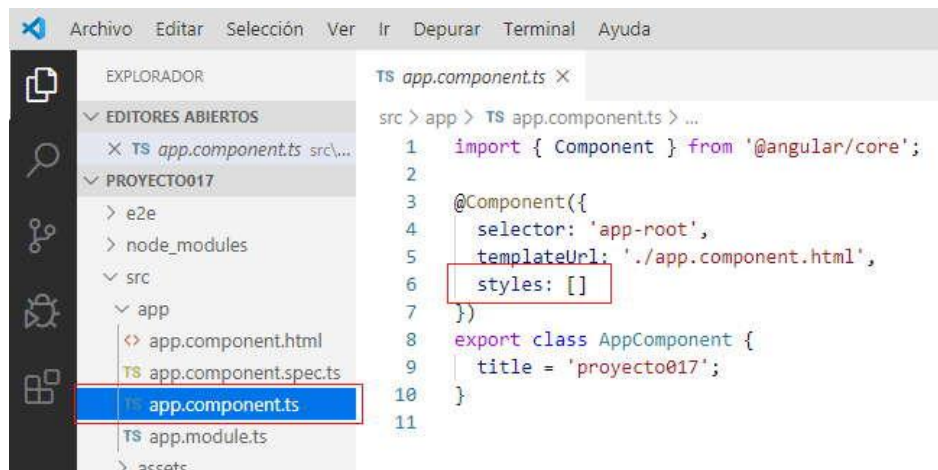
Si queremos escribir menos con Angular CLI al crear el proyecto podemos utilizar el alias -p en lugar de --prefix:

```
ng new proyecto017 -p ele
```

Mediante el parámetro `--inline-style` (alias: `-s`) podemos evitar que se cree el archivo `*.css` y la definición de los estilos se deba hacer directamente en el archivo `*.ts`:

```
ng new proyecto017 --inline-style
```

Con esto no se crearán los archivos `*.css` para las componentes. El archivo `*.ts` de la componente tienen un lugar donde definir los estilos:



Puede tener sentido si la componente es bastante sencilla y no requiere la definición de una hoja de estilo compleja.

Probar modificar los archivos `*.ts` y `*.html` de la componente del proyecto.

app.component.ts

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

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styles: ['.titulo { color:red; font-size:2rem }',
    '.parrafo { color:black; font-size:1.1rem }']
})
```

```
export class AppComponent {
  tit = 'Prueba de inline-style';
```

```
}
```

app.component.html

```
<div style="text-align:center">
<h1 class="titulo">{{tit}}</h1>
<p class="parrafo">Esto es un párrafo.</p>
</div>
```

Si ejecutamos el proyecto podemos ver que los estilos se están recuperando del archivo *.ts:

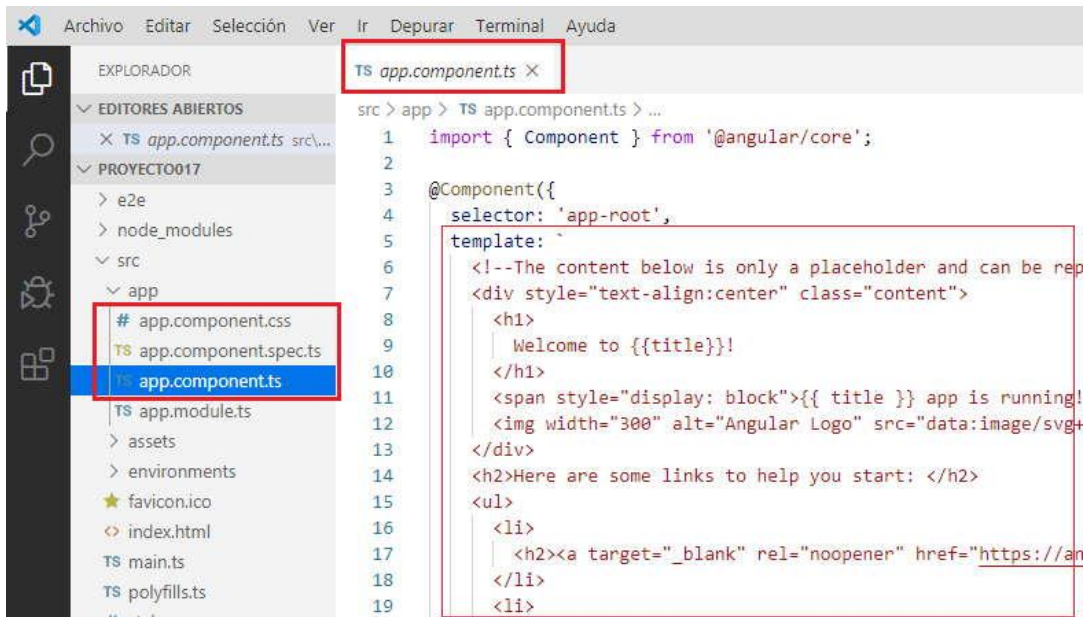


Mediante el parámetro `--inline-template` (alias: `-s`) podemos evitar que se cree el archivo *.html y la definición del HTML se hace directamente en el archivo *.ts (borre primero el proyecto017 antes de crearlo nuevamente con esta nueva configuración):

```
ng new proyecto017 --inline-template
```

Con este parámetro estamos indicando que no se cree el archivo *.html para la componente y la definición del HTML se haga en el mismo archivo *.js

Luego de creado el proyecto solo se han creado los archivos *.css y *.ts:



Nuevamente decimos que esto tiene sentido si la complejidad de la componente Angular no es grande.

Podemos inclusive evitar que se creen tanto el archivo *.css y *.html indicando ambos parámetros al crear la componente:

```
ng new proyecto017 --inline-style --inline-template
```

Si tratamos de crear un proyecto y ya existe uno en la carpeta actual se produce un error al ejecutar:

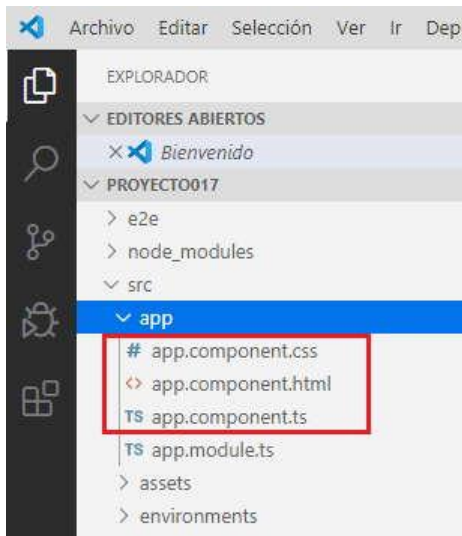
```
ng new proyecto017
```

Si queremos forzar la creación del proyecto y sobrescribir los archivos actuales debemos añadir el parámetro --force (alias -f):

```
ng new proyecto017 --force
```

Si por algún motivo no queremos que Angular.CLI nos genere el archivo de test debemos pasar el parámetro --skip-tests (alias -s):

```
ng new proyecto017 --skip-tests
```



Como podemos comprobar no se ha generado el archivo 'app.component.spec.ts'.

Otro parámetro más que podemos utilizar cuando vamos a crear un proyecto es `--dry-run` (alias: `-d`)

Mediante esta opción Angular CLI nos informa que archivos se crearán (no los crea) y a partir de estos datos poder tomar la decisión de crear o no el proyecto:

```
ng new proyecto017 --dry-run
```

```

Node.js command prompt
F:\angular>ng new proyecto017 --dry-run
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? CSS
CREATE proyecto017/angular.json (3607 bytes)
CREATE proyecto017/package.json (1288 bytes)
CREATE proyecto017/README.md (1028 bytes)
CREATE proyecto017/tsconfig.json (543 bytes)
CREATE proyecto017/tslint.json (1953 bytes)
CREATE proyecto017/.editorconfig (246 bytes)
CREATE proyecto017/.gitignore (631 bytes)
CREATE proyecto017/browserslist (429 bytes)
CREATE proyecto017/karma.conf.js (1023 bytes)
CREATE proyecto017/tsconfig.app.json (210 bytes)
CREATE proyecto017/tsconfig.spec.json (270 bytes)
CREATE proyecto017/src/favicon.ico (948 bytes)
CREATE proyecto017/src/index.html (297 bytes)
CREATE proyecto017/src/main.ts (372 bytes)
CREATE proyecto017/src/polyfills.ts (2838 bytes)
CREATE proyecto017/src/styles.css (80 bytes)
CREATE proyecto017/src/test.ts (753 bytes)
CREATE proyecto017/src/assets/.gitkeep (0 bytes)
CREATE proyecto017/src/environments/environment.prod.ts (51 bytes)
CREATE proyecto017/src/environments/environment.ts (662 bytes)
CREATE proyecto017/src/app/app.module.ts (314 bytes)
CREATE proyecto017/src/app/app.component.html (25673 bytes)
CREATE proyecto017/src/app/app.component.spec.ts (957 bytes)
CREATE proyecto017/src/app/app.component.ts (215 bytes)
CREATE proyecto017/src/app/app.component.css (0 bytes)
CREATE proyecto017/e2e/protractor.conf.js (808 bytes)
CREATE proyecto017/e2e/tsconfig.json (214 bytes)
CREATE proyecto017/e2e/src/app.e2e-spec.ts (644 bytes)
CREATE proyecto017/e2e/src/app.po.ts (301 bytes)

NOTE: The "dryRun" flag means no changes were made.
F:\angular>
  
```


Herramienta Angular CLI - comando: ng generate

Otro comando que hemos utilizado a lo largo de los primeros conceptos de este curso de Angular es 'generate'. Mediante el comando 'generate' de la herramienta Angular CLI podemos crear:

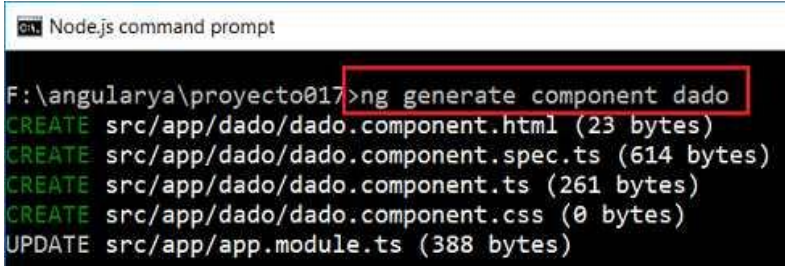
- component
- service
- module
- pipe
- class
- interface
- enum
- directive
- application
- library
- universal
- guard

NG GENERATE COMPONENT

Mediante el comando:

```
ng generate component dado
```

Se crea una componente 'dado' con los archivos respectivos y la modificación del archivo del módulo: Se informa en la consola los archivos creados y modificados:



```
Node.js command prompt
F:\angularya\proyecto017>ng generate component dado
CREATE src/app/dado/dado.component.html (23 bytes)
CREATE src/app/dado/dado.component.spec.ts (614 bytes)
CREATE src/app/dado/dado.component.ts (261 bytes)
CREATE src/app/dado/dado.component.css (0 bytes)
UPDATE src/app/app.module.ts (388 bytes)
```

Podemos pasar varias opciones cuando creamos la componente:

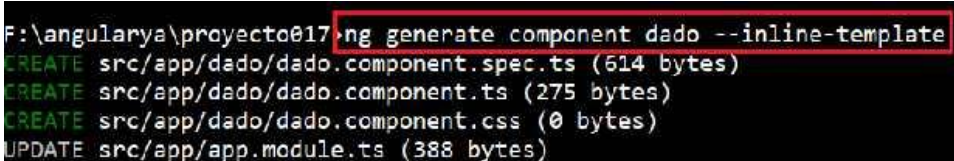
```
ng generate component dado --inline-style
```

Evita que se cree el archivo dado.component.css

Lo mismo podemos evitar que se cree el archivo HTML:

```
ng generate component dado --inline-template
```

Tenemos como resultado la no creación del archivo HTML:



```
F:\angularya\proyecto017>ng generate component dado --inline-template
CREATE src/app/dado/dado.component.spec.ts (614 bytes)
CREATE src/app/dado/dado.component.ts (275 bytes)
CREATE src/app/dado/dado.component.css (0 bytes)
UPDATE src/app/app.module.ts (388 bytes)
```

Podemos definir el prefijo para la componente mediante la opción --prefix (alias: -p):

```
ng generate component dado --prefix juego
```

Luego la componente que se crea tiene dicho prefijo:

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

@Component({
  selector: 'juego-dado',
  templateUrl: './dado.component.html',
  styleUrls: ['./dado.component.css']
})
export class DadoComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

Si queremos definir el nombre completo para el selector tenemos la opción --selector:

```
ng generate component dado --selector ju-dado
```

Luego la componente se crea con dicho nombre de selector:

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

@Component({
  selector: 'ju-dado',
  templateUrl: './dado.component.html',
  styleUrls: ['./dado.component.css']
})
export class DadoComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

Para almacenar la componente en un determinado módulo debemos utilizar la opción --module (alias: -m)

```
ng generate component elementos/dado --module elementos
```

Considerando que ya hemos creado un módulo llamado elementos (ng generate module elementos) estamos creando la componente dado en el modulo elementos.

Para evitar que se cree el archivo 'dado.spec.js' debemos insertar el comando --spec con el valor false:

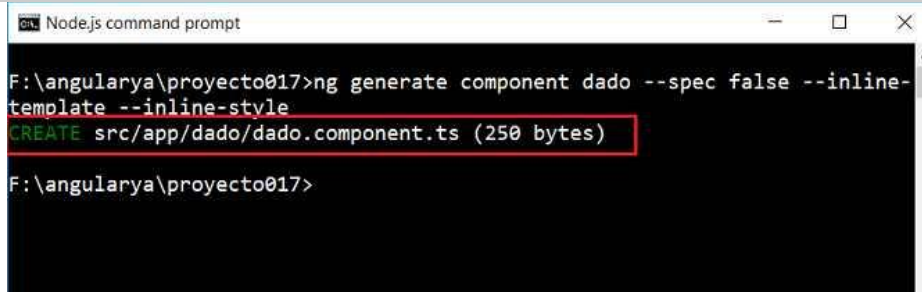
```
ng generate component dado --spec false
```

Luego no se genera el archivo 'dado.spec.js':

```
F:\angularya\proyecto017>ng generate component dado --spec false
CREATE src/app/dado/dado.component.html (23 bytes)
CREATE src/app/dado/dado.component.ts (261 bytes)
CREATE src/app/dado/dado.component.css (0 bytes)
```

Tengamos en cuenta que todas estas opciones se pueden combinar y ejecutar en forma simultanea, por ejemplo si queremos generar solo el archivo *.ts de la componente y que no genere el archivo spec, *.css y *.html:

```
ng generate component dado --spec false --inline-template --inline-style
```



```
Node.js command prompt
F:\angularya\proyecto017>ng generate component dado --spec false --inline-template --inline-style
CREATE src/app/dado/dado.component.ts (250 bytes)
F:\angularya\proyecto017>
```

Otras opciones posibles cuando creamos una componente son:

--force (alias: -f) Forzar la sobrescritura de los archivos existentes (se borra la componente anterior que tiene el mismo nombre)

--dry-run (alias: -d) Informa los archivos que se crearán sin hacer dicha actividad.

Angular CLI permite ingresar comandos en formato resumido utilizando el primer caracter:

```
ng g c dado
```

En lugar de escribir:

```
ng generate component dado
```

NG GENERATE SERVICE

Mediante el comando:

```
ng generate service articulos
```

Se crea una clase ArticulosService y se inyecta a nivel de 'root':

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

```
@Injectable({
  providedIn: 'root'
})
export class ArticulosService {

  constructor() { }
}
```

Disponemos de los siguientes opciones en este comando:

--dry-run

--force

--spec

NG GENERATE MODULE

Mediante el comando:

```
ng generate module administracion
```

Se crea la carpeta 'administracion' y dentro de ella el archivo 'administracion.module.ts'.

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

@NgModule({
  imports: [
    CommonModule
  ],
  declarations: []
})
export class AdministracionModule { }
```

Disponemos de los siguientes opciones en este comando:

--dry-run

--force

--spec

--routing

NG GENERATE PIPE

Mediante el comando:

```
ng generate pipe letras
```

Se genera el archivo 'letras.pipe.ts':

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'letras'
})
export class LetrasPipe implements PipeTransform {

  transform(value: any, args?: any): any {
    return null;
  }

}
```

Disponemos de los siguientes opciones en este comando:

--dry-run

--force

--spec

--module

NG GENERATE CLASS

Mediante el comando:

```
ng generate class articulo
```

Se crea el archivo 'articulo.ts':

```
export class Articulo {
}
```

NG GENERATE INTERFACE

Mediante el comando:

```
ng generate interface venta
```

Se crea el archivo 'venta.ts':

```
export interface Venta {
}
```

NG GENERATE ENUM

Mediante el comando:

```
ng generate enum operaciones
```

Se crea el archivo 'operaciones.enum.ts':

```
export enum Operaciones {  
}
```

Herramienta Angular CLI - comando: ng serve

Otro comando que hemos utilizado en cada uno de los proyectos que hemos implementado es 'serve' (desde el principio hemos utilizado la opción -o):

```
ng serve -o
```

Recordemos que debemos ejecutar el comando serve en la carpeta donde se haya nuestra aplicación Angular. Al disponer la opción -o se abre automáticamente el navegador web. La sintaxis larga pero que produce la misma acción es:

```
ng serve --open
```

Para conocer todas las opciones disponibles en un comando de Angular CLI debemos acceder a la opción --help:

```
ng serve --help
```

OPCIÓN --PORT

Por defecto el servidor web local que crea Angular CLI se ejecuta en el puerto 4200, si necesitamos que el servidor web se ejecute en otro puerto podemos indicarlo con la opción --port en el momento de iniciarlo:

```
ng serve -o --port 4444
```

OPCIÓN --WATCH (ALIAS: -W)

Por defecto cada vez que modificamos nuestro proyecto y grabamos los cambios el resultado se actualiza en el navegador en forma automática. En algunas situaciones si queremos que no se actualice debemos utilizar la opción 'watch' pasando el valor false:

```
ng serve -o -watch false
```

OPCIÓN --PROD

Ejecuta la aplicación con todas las optimizaciones que se hacen cuando se genera el código de producción.

```
ng serve -o --prod
```

Directiva estructural *ngFor - Sintaxis completa

Angular tiene por defecto varias directivas estructurales: *ngFor, *ngIf y *ngSwitchCase, las hemos estado utilizando constantemente desde los primeros conceptos de este curso, ahora veremos en más profundidad la directiva estructural *ng-For y sus posibilidades.

Recordemos que las directivas estructurales se aplican a elementos HTML que permiten añadir, manipular o eliminar elementos del DOM (Document Object Model).

Las directivas estructurales llevan el prefijo *.

La directiva *ngFor permite añadir elementos HTML de acuerdo a la etiqueta HTML que se le aplica: <tr>, etc.

SU SINTAXIS TIENE PARTES OBLIGATORIAS:

Si tenemos definido el siguiente arreglo en el archivo *.ts:

```
arreglo1 = [10, 20, 30, 40, 50];
```

Luego podemos iterar sus elementos para mostrarlos en la vista:

```
<ul>
<li *ngFor="let elemento of arreglo1">
  {{elemento}}
</li>
</ul>
```

En la primera iteración la variable 'elemento' almacena el valor 10, y mediante interpolación mostramos su valor dentro de la etiqueta 'li'.

Declaración de la variable que contiene el valor de la iteración mediante la palabra clave 'let' : 'let elemento'.

Palabra clave: 'of'

Variable a iterar: 'arreglo1'

Esta es la estructura mínima que podemos definir cuando utilizamos la directiva estructural *ngFor.

SU SINTAXIS EXTENDIDA:

Si tenemos definido el siguiente arreglo y método en el archivo *.ts:

```
arreglo1 = [10, 20, 30, 40, 50];  
rastrearPor(indice: number, elemento: number) {  
  console.log(indice, elemento);  
}
```

Luego podemos iterar sus elementos para mostrarlos en la vista junto con otros elementos de información que podemos rescatar:

```
<ul>  
<li *ngFor="let elemento of arreglo1; let indice=index; let impar=odd; trackBy: rastrearPor;let primo  
ro=first; let ultimo=last">  
  El {{elemento}} está en la posición {{indice}} se encuentra en una posición {{impar?'impar':'par'}}  
  {{primero?'es el primero del arreglo':''}} {{ultimo?'es el último del arreglo':''}}  
</li>  
</ul>
```

Separamos por punto y coma cada uno de los otros datos opcionales que nos suministra la directiva estructural *ngFor:

let indice=index : En la variable 'indice' se almacena el índice actual del elemento del arreglo1 que estamos procesando.

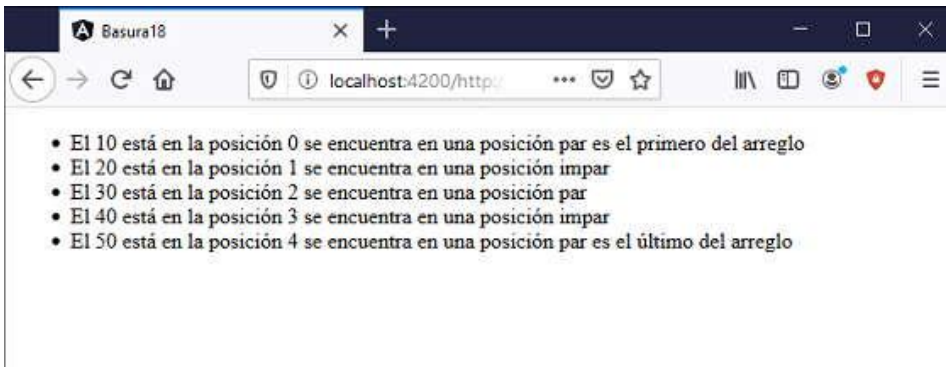
let impar=odd : En la variable 'impar' se almacena true si el elemento del arreglo1 se encuentra en una posición impar.

let primero=first : Almacena la variable 'primero' un valor true si estamos iterando el primer elemento del arreglo1.

let ultimo=last : Almacena la variable 'ultimo' un valor true si estamos iterando el último elemento del arreglo1.

trackBy: rastrearPor : Indicamos el nombre de una función que se ejecutará por cada iteración del arreglo1, pasando el índice y el elemento procesado.

Como resultado de ejecutar la aplicación tenemos como impresión:



La información extra que provee la directiva nos puede servir para tomar decisiones cuando tenemos que mostrar información, por ejemplo pintar de distinto color las filas pares e impares de una tabla, mostrar información extra para el primer elemento etc.

Microsintaxis

La microsintaxis de angular le permite configurar una directiva en una cadena compacta y amigable. El analizador de microsintaxis traduce esa cadena en atributos de una etiqueta <ng-template>.

Por ejemplo la sintaxis del problema anterior:

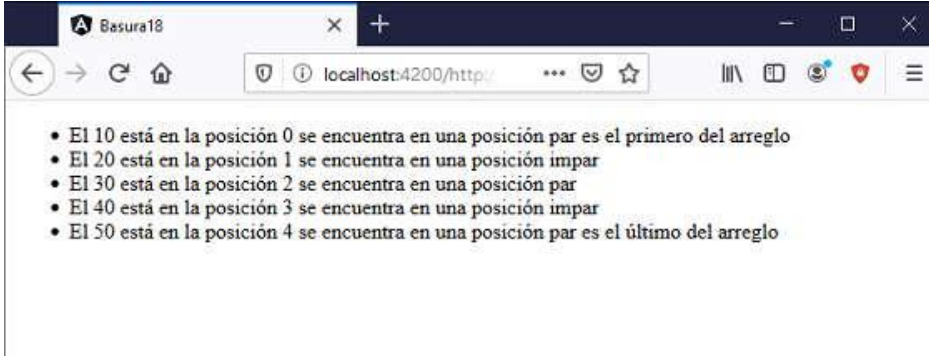
```
<ul>
<li *ngFor="let elemento of arreglo1; let indice=index; let impar=odd; trackBy: rastrearPor;let primero=first; let ultimo=last">
El {{elemento}} está en la posición {{indice}} se encuentra en una posición {{impar?'impar':'par'}}
{{primero?'es el primero del arreglo':''}} {{ultimo?'es el último del arreglo':''}}
</li>
</ul>
```

El transpilador de Angular procede a modificar la directiva estructural *ngFor por un elemento <ng-template> que envuelve a la etiqueta que le hemos definido la directiva estructural:

```
<ul>
<ng-template ngFor let-elemento [ngForOf]="arreglo1" let-indice="index" let-impar="odd" let-primero="first" let-ultimo="last" [ngForTrackBy]="rastrearPor">
<li>
El {{elemento}} está en la posición {{indice}} se encuentra en una posición {{impar?'impar':'par'}}
{{primero?'es el primero del arreglo':''}} {{ultimo?'es el último del arreglo':''}}
</li>
```

```
</ng-template>
</ul>
```

Si modificamos manualmente con el código anterior, el resultado es exactamente el mismo:



Si bien el resultado es el mismo, podemos comprobar que es mucho más legible utilizar la sintaxis de la directiva `*ngFor`:

```
<li *ngFor="let elemento of arreglo1; let indice=index; let impar=odd; trackBy: rastrearPor; let primero=first; let ultimo=last">
```

En lugar de definir la etiqueta `ng-template` con sus propiedades:

```
<ng-template ngFor let-elemento [ngForOf]="arreglo1" let-indice="index" let-impar="odd" let-primero="first" let-ultimo="last" [ngForTrackBy]="rastrearPor">
```

Traemos el concepto de microsinaxis para entender porque las directivas estructurales requieren obligatoriamente el caracter `'*` previo a su nombre. Angular sabe que las directivas estructurales deben ser transformadas previamente y expandidas en una etiqueta `ng-template`.

Siempre es más conveniente utilizar la sintaxis `*ngFor`, pero es importante entender como Angular transforma este tipo de directivas para los casos que tengamos que codificar nuestras propias directivas estructurales.

Directiva estructural `*ngIf` - Sintaxis completa

En el concepto anterior trabajamos en profundidad la sintaxis de la directiva estructural `*ngFor`, ahora nos toca trabajar con la directiva estructural `*ngIf`.

La directiva estructural `*ngIf` permite mostrar u ocultar elementos del DOM.

Si tenemos un atributo en el archivo `*.ts`:

```
alerta=true;
```

Luego en el archivo HTML mediante la directiva *ngIf:

```
<h1 *ngIf="alerta">Todos los precios se encuentran rebajados 50%</h1>
```

Como la variable almacena un 'true', luego la etiqueta 'h1' se muestra en la página. Si la variable 'alerta' se le asigna un 'false' la etiqueta 'h1' no se muestra en la página (no se inserta en el DOM)

Recordemos que una etiqueta solo puede tener una directiva estructural, luego si podemos disponer dentro de una etiqueta interior otro directiva estructural.

Por ejemplo si tenemos declarado el siguiente array en el archivo *.ts:

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  personas: Persona[] = [
    new Persona('juan', 33),
    new Persona('ana', 15),
    new Persona('luis', 56),
    new Persona('carla', 45)
  ];
}

class Persona {
  constructor(public nombre: string, public edad: number) { }
}
```

Luego podemos recorrer mediante un *ngFor el arreglo y verificar cada edad para ver si mostramos el elemento HTML 'span':

```
<table>
<tr>
<td>Nombre</td>
<td>Edad</td>
</tr>
<tr *ngFor="let persona of personas">
<td>{{persona.nombre}} <span *ngIf="persona.edad>=18">(Mayor)</span></td>
<td>{{persona.edad}}</td>
</tr>
</table>
```

Se verifica la edad de cada persona y si es mayor o igual a 18 luego la directiva estructural `*ngIf` se verifica verdadera por lo que la etiqueta 'span' se incorpora al DOM de la página web:

```
*ngIf="persona.edad>=18">(Mayor)</span>
```

DIRECTIVA *NGIF CON ELSE

Una parte opcional de la directiva estructural `*ngIf` es la sección del else. La sintaxis que debemos utilizar para el else es:

Si tenemos un atributo en el archivo `*.ts`:

```
estado=true;
```

Luego en el archivo HTML mediante la directiva `*ngIf` con else podemos mostrar uno u otro título:

```
<h1 *ngIf="estado; else fueradeservicio">Bienvenido al sitio</h1>
<ng-template #fueradeservicio><h1>Sitio fuera de servicio</h1></ng-template>
```

Es obligatorio cuando definimos la sección del else definir una etiqueta 'ng-template', ya que no podemos disponer #fueradeservicio en la etiqueta 'h1'.

USO DE OPERADORES LÓGICOS

Podemos disponer operadores lógicos en el valor asignado a la directiva estructural `*ngIf`, si tenemos los atributos definidos en la clase:

```
estado=true;
```

```
dia=9;
```

Luego podemos disponer una condición compuesta:

```
<h1 *ngIf="estado && dia>=10; else fueradeservicio">Bienvenido al sitio</h1>
<ng-template #fueradeservicio><h1>Sitio fuera de servicio</h1></ng-template>
```

En este caso se ejecuta el else debido a que el atributo dia tiene un valor que no es mayor o igual a 10.

LLAMADA A UN MÉTODO.

Podemos disponer la llamada a un método o función que retorne un valor boolean en la directiva estructural *ngIf:

```
export class AppComponent {
  edad = 52;
  mayorEdad(): boolean {
    if (this.edad >= 18)
      return true;
    else
      return false;
  }
}
```

Luego llamamos al método en la directiva:

```
<p *ngIf="mayorEdad()">Es mayor de edad</p>
```

Retorna true el método y por lo tanto se muestra el párrafo.

VARIANTE DE SINTAXIS CON THEN ELSE.

Hay otra sintaxis que podemos utilizar cuando tenemos tanto actividades por el verdadero como por el falso.

Si tenemos el atributo:

```
estado=true;
```

Luego tenemos la sintaxis:

```
<ng-template *ngIf="estado;then serviciocorrecto else fueradeservicio"></ng-template>
<ng-template #serviciocorrecto><h1>Bienvenido al sitio</h1></ng-template>
<ng-template #fueradeservicio><h1>Sitio fuera de servicio</h1></ng-template>
```

Es una alternativa poco usada, pero disponible.

Microsintaxis

La microsintaxis de angular le permite configurar una directiva en una cadena compacta y amigable. El analizador de microsintaxis traduce esa cadena en atributos de una etiqueta `<ng-template>`. La microsintaxis se aplica también para la directiva `*ngIf`.

Por ejemplo la sintaxis del problema anterior:

```
<h1 *ngIf="alerta">Todos los precios se encuentran rebajados 50%</h1>
```

El transpilador de Angular procede a modificar la directiva estructural `*ngIf` por un elemento `<ng-template>`:

```
<ng-template [ngIf]="alerta">
<h1>Todos los precios se encuentran rebajados 50%</h1>
</ng-template>
```

Es decir encierra el elemento HTML 'h1' por el elemento 'ng-template', el cual define un enlace de propiedad al encerrarlo entre corchetes.

Si disponemos de else la transformación es:

```
<ng-template [ngIf]="estado" [ngIfElse]="fueradeservicio">
<h1>Bienvenido al sitio</h1>
</ng-template>
<ng-template #fueradeservicio>
<h1>Sitio fuera de servicio</h1>
</ng-template>
```

Recordar que normalmente se utiliza la sintaxis `*ngIf` para implementar esta directiva estructural, el conocer la transformación es importante para cuando creamos nuestras propias directivas.

Directivas estructurales *ngSwitchCase, *ngSwitchDefault y ngSwitch

Para implementar la sintaxis de un switch similar a Javascript, Angular requiere dos directivas estructurales: *ngSwitchCase, *ngSwitchDefault y una directiva de atributo que deben trabajar en conjunto.

Veamos con un ejemplo como actúan en conjunto estas directivas.

Implementar una aplicación que permita cargar dos valores numéricos y mediante un control select indicar si queremos sumar, restar, multiplicar o dividir. Actualizar el resultado inmediatamente se seleccione la operación del control select.

```
ng new proyecto045
```

Como utilizaremos la directiva ngModel debemos importar la clase 'FormsModule' en el archivo 'app.module.ts':

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

En la clase principal 'AppComponent' definimos las variables que luego en la vista se asocian a los distintos controles de formulario:

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  valor1: number;
  valor2: number;
  operacion: string = "ninguna";
  resultado: number;
}
```

En la vista definimos los tres controles de formulario y mediante estas nuevas directivas reaccionamos para mostrar alguno de los cinco valores que puede tomar el control 'select':

```
<p>Ingrese primer valor
  <input [(ngModel)]="valor1" type="number">
</p>
<p>Ingrese segundo valor
  <input [(ngModel)]="valor2" type="number">
</p>
<p>
  <select [(ngModel)]="operacion">
    <option value="ninguna">Elija una opción</option>
    <option value="sumar">Sumar</option>
    <option value="restar">Restar</option>
```



```

<option value="multiplicar">multiplicar</option>
<option value="dividir">Dividir</option>
</select>
</p>
<p>Resultado:
<span [ngSwitch]="operacion">
  <span *ngSwitchCase="sumar">{{valor1+valor2}}</span>
  <span *ngSwitchCase="restar">{{valor1-valor2}}</span>
  <span *ngSwitchCase="multiplicar">{{valor1*valor2}}</span>
  <span *ngSwitchCase="dividir">{{valor1/valor2}}</span>
  <span *ngSwitchDefault>(No eligió operación)</span>
</span>
</p>

```

Disponemos una etiqueta 'span' y le asociamos una directiva de atributo indicando la variable 'operación':

```
<span [ngSwitch]="operacion">
```

Según el valor que almacena la variable 'operación' se ejecutará el *ngSwitchCase que coincide exactamente con el string que almacena. En el caso que tenga almacenado la cadena 'ninguna' se verifica verdadera la directiva *ngSwitchDefault.

Las comillas simples dentro de las dobles son importantes ya que la variable operación es de tipo string:

```
*ngSwitchCase="sumar"
```

Podemos probar esta aplicación en la web [aquí](#).

Microsintaxis

La microsintaxis de angular le permite configurar una directiva en una cadena compacta y amigable. El analizador de microsintaxis traduce esa cadena en atributos de una etiqueta <ng-template>. La microsintaxis se aplica también para las directivas *ngSwitchCase y *ngSwitchDefault.

Por ejemplo la sintaxis del problema anterior:

```
<span [ngSwitch]="operacion">
```

```
<span *ngSwitchCase="sumar">{{valor1+valor2}}</span>
<span *ngSwitchCase="restar">{{valor1-valor2}}</span>
<span *ngSwitchCase="multiplicar">{{valor1*valor2}}</span>
<span *ngSwitchCase="dividir">{{valor1/valor2}}</span>
<span *ngSwitchDefault>(No eligió operación)</span>
</span>
```

El transpilador de Angular procede a modificar las directivas estructurales `*ngSwitchCase` y `*ngSwitchDefault` por elementos `<ng-template>`:

```
<span [ngSwitch]="operacion">
  <ng-template [ngSwitchCase]="sumar">
    <span>{{valor1+valor2}}</span>
  </ng-template>
  <ng-template [ngSwitchCase]="restar">
    <span>{{valor1-valor2}}</span>
  </ng-template>
  <ng-template [ngSwitchCase]="multiplicar">
    <span>{{valor1*valor2}}</span>
  </ng-template>
  <ng-template [ngSwitchCase]="dividir">
    <span>{{valor1/valor2}}</span>
  </ng-template>
  <ng-template ngSwitchDefault>
    <span>(No eligió operación)</span>
  </ng-template>
</span>
```

Normalmente utilizamos el prefijo `*` para crear las directivas estructurales, pero cuando veamos la creación de nuestras propias directivas estructurales necesitamos conocer como Angular las trata y modifica.

IV. PROCEDIMIENTO

Creación Crud Angular – Firebase

1. Crear un nuevo

```
ng new crudAngularFirebase
```

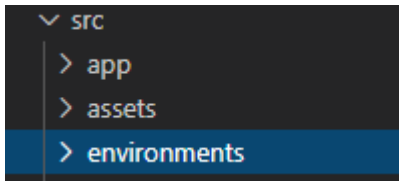
2. Instalar Firebase al proyecto y el paquete que integra angular con firebase

```
npm install firebase @angular/fire
```

3. Pueden verificar la instalación en la terminal

```
+ angularfire2@5.4.2
+ firebase@7.19.1
```

4. Verificar la carpeta **environments**, donde se tendrá la configuración firebase



5. Instalar tres componentes, una clase, un servicio y una lib para notificaciones toastr

ng g c components/products	*(componente principal)
ng g c components/products/product-list	*(para listar todos los registros)
ng g c components/products/product	*(para mostrar un registro)
ng g cl models/product	*(clase producto, para almacenar registros)
ng g s services/product	*(servicio que se conectara con Firebase)
npm install ngx-toastr --save	*(Librería para notificaciones en Angular)

6. Agregamos la configuración de **toastr** a nuestro proyecto, al archivo **angular.json**

```
"styles": [
  "src/styles.css",
  "node_modules/ngx-toastr/toastr.css"
],
```

7. El siguiente paso es compilar el proyecto para verificar que todo está bien.

```
ng serve -o
```

8. Buscamos el archivo de configuración **app.module.ts** , para agregar las configuraciones que hacen falta.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';

// firebase
import { environment } from '../environments/environment';
import { AngularFireModule } from '@angular/fire';
import { AngularFireDatabaseModule } from '@angular/fire/database';
//import { AngularFireStoreModule } from '@angular/fire/firestore';
//import { AngularFireStorageModule } from '@angular/fire/storage';
//import { AngularFireAuthModule } from '@angular/fire/auth';

// components
import { ProductsComponent } from './components/products/products.component';
import { ProductListComponent } from './components/products/product-list/product-list.component';
import { ProductComponent } from './components/products/product/product.component';

// service
import { ProductService } from './services/product.service';

// Toastr, para notificaciones en angular
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { ToastrModule } from 'ngx-toastr';

@NgModule({
  declarations: [
    AppComponent,
    ProductsComponent,
    ProductListComponent,
    ProductComponent
  ],
  imports: [ // Se importan todas las dependencias, para utilizarlos en todo el proyecto.
    BrowserModule,
    AngularFireModule.initializeApp(environment.firebase),
    AngularFireDatabaseModule,
    FormsModule,
    ToastrModule.forRoot(),
    BrowserAnimationsModule
  ],
})
```

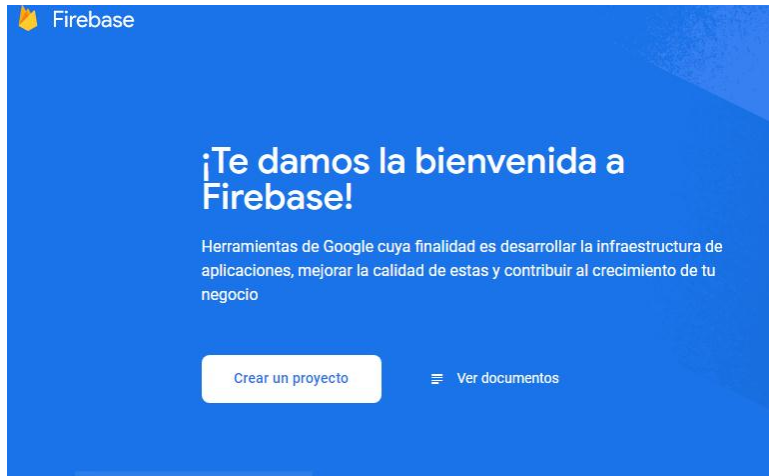
```
providers: [ // El servicio se configura global, para utilizarlos en todo el proyecto.
  ProductService
],
bootstrap: [AppComponent]
}))
export class AppModule { }
```

- En el proyecto de angular buscamos el archivo de configuración **index.html** y agregamos bootstrap 4.0.0y fontawesome v5.0.6

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>CrudAngularFirebase</title>
  <base href="/">
  <!-- bootstrap 4.0.0 -->
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
    integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">
  <!-- fontawesome v5.0.6 -->
  <script defer src="https://use.fontawesome.com/releases/v5.0.6/js/all.js"></script>
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

- Los recursos están en el repositorio : (los archivos están comentados)
<https://github.com/AlexanderSiguenza/crudAngularFirebase>
- Crear o ingresar a la cuenta personal de Firebase <https://firebase.google.com/> , Dentro Firebase hacer los siguientes pasos

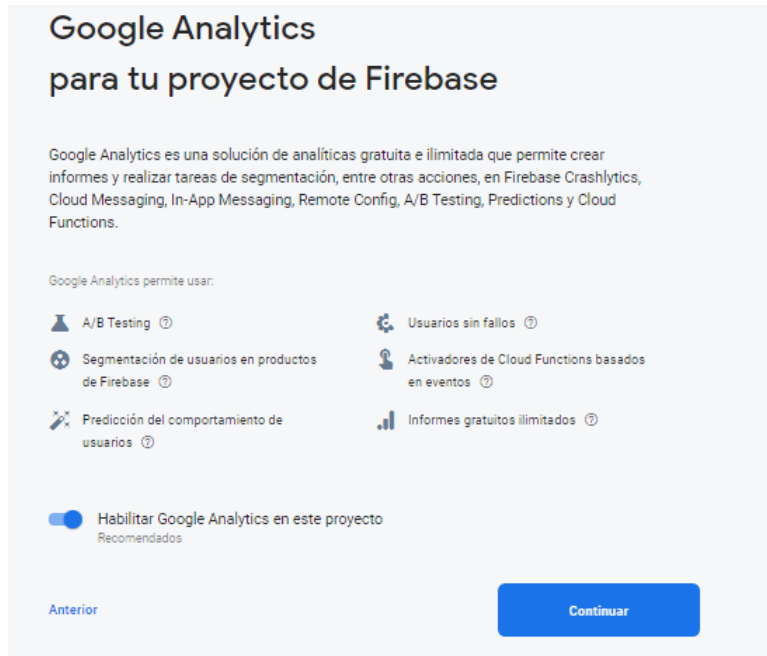
12. Crear un nuevo proyecto



13. Al presionar el botón crear un proyecto, colocar el nombre y presionar continuar.

The image shows the Firebase project creation form. The heading is "Empieza por ponerle un nombre al proyecto". Below it, the label "Nombre del proyecto" is followed by the text "crudAngularFirebase". A text box below shows the default project ID "crudangularfirebase-8ccb8" with an edit icon. At the bottom, there is a checked checkbox next to the text "Acepto los términos de Firebase." and a blue "Continuar" button.

14. Agregar Google Analytic al proyecto



15. Para finalizar seleccionar Estados Unidos y aceptar los términos

X Crear un proyecto(paso 3 de 3)

Configurar Google Analytics

Ubicación de Analytics ⓘ

Estados Unidos ▼

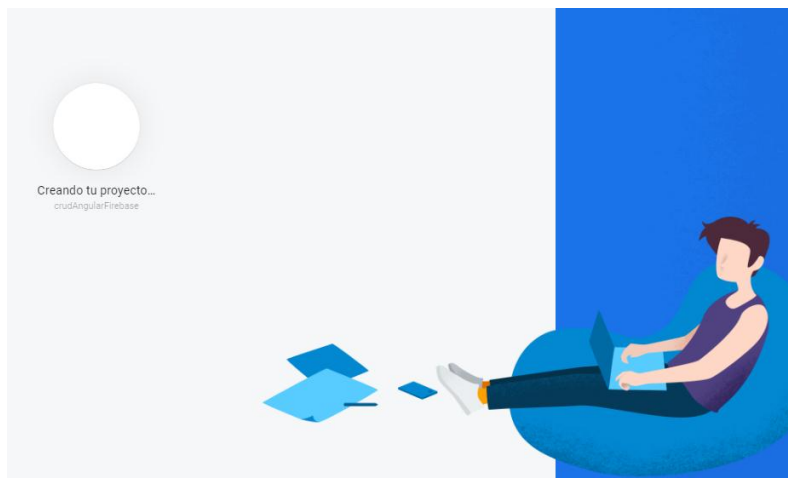
Términos de Google Analytics y configuración de uso compartido de datos

- ☒ Usar la configuración predeterminada para compartir datos de Google Analytics. [Learn more](#)
 - ✓ Comparte tus datos de Analytics con nosotros para ayudarnos a mejorar nuestros productos y servicios.
 - ✓ Comparte tus datos de Analytics con nosotros para habilitar las comparativas.
 - ✓ Comparte tus datos de Analytics con Google para habilitar el servicio de asistencia técnica.
 - ✓ Comparte tus datos de Analytics con los especialistas en cuentas de Google.
- ☒ Acepto los [términos de protección de datos entre responsables del tratamiento de datos y responsables del tratamiento de datos de mediciones](#), y reconozco que estoy sujeto a la [política de consentimiento de usuarios finales de la Unión Europea](#). Es obligatorio marcar esta casilla si vas a compartir tus datos de Analytics para mejorar los productos y servicios de Google. [Más información](#)
- ☒ Acepto los [términos de Google Analytics](#).

Al crear el proyecto, también se creará una propiedad de Google Analytics que se asociará a tu proyecto de Firebase. De esta forma, se habilitará el flujo de datos entre los productos. Los datos que se exportan de tu propiedad de Google Analytics a Firebase están sujetos a los términos del servicio de Firebase, mientras que los datos que se importan de Firebase a Google Analytics se rigen por los términos del servicio de Google Analytics. [Más información](#)

[Anterior](#) [Crear proyecto](#)

16. Esperar que el proyecto se cree



17. Al finalizar se mostrará la pantalla



18. Para empezar, conectarse a la aplicación web



19. Añadir SDK de Firebase

2
Añadir SDK de Firebase

Antes de utilizar cualquier servicio de Firebase, copia y pega estas secuencias de comandos en la parte inferior de la etiqueta <body>:

```

<!-- The core Firebase JS SDK is always required and must be listed first -->
<script src="//firebase/7.19.1/firebase-app.js"></script>

<!-- TODO: Add SDKs for Firebase products that you want to use
https://firebase.google.com/docs/web/setup#available-libraries -->
<script src="//firebase/7.19.1/firebase-analytics.js"></script>

<!-- Initialize Firebase -->
<script src="//firebase/init.js"></script>

```

Más información sobre Firebase para aplicaciones web: [Primeros pasos](#), [Referencia de APIs del SDK web](#) y [Muestras](#).

20. Instalar CLI de Firebase

3
Instalar CLI de Firebase

Para alojar tu sitio web con Firebase Hosting, necesitas la herramienta de línea de comandos (CLI) de Firebase.

Ejecuta el siguiente comando [npm](#) para instalar la CLI o actualizarla a la versión más reciente.

```
$ npm install -g firebase-tools
```

¿No funciona? Consulta la [referencia de la CLI de Firebase](#) o cambia los [permisos npm](#).

Siguiente

21. Desplegar en Firebase Hosting

4

Desplegar en Firebase Hosting

Puedes desplegarlo ahora o [más adelante](#). Para hacerlo ahora, abre una ventana de terminal y accede al directorio "root" de la aplicación web (si no tienes uno, tendrás que crearlo).
Inicia sesión en Google

```
$ firebase login
```

Inicia tu proyecto
Ejecuta este comando desde el directorio "root" de tu aplicación:

```
$ firebase init
```

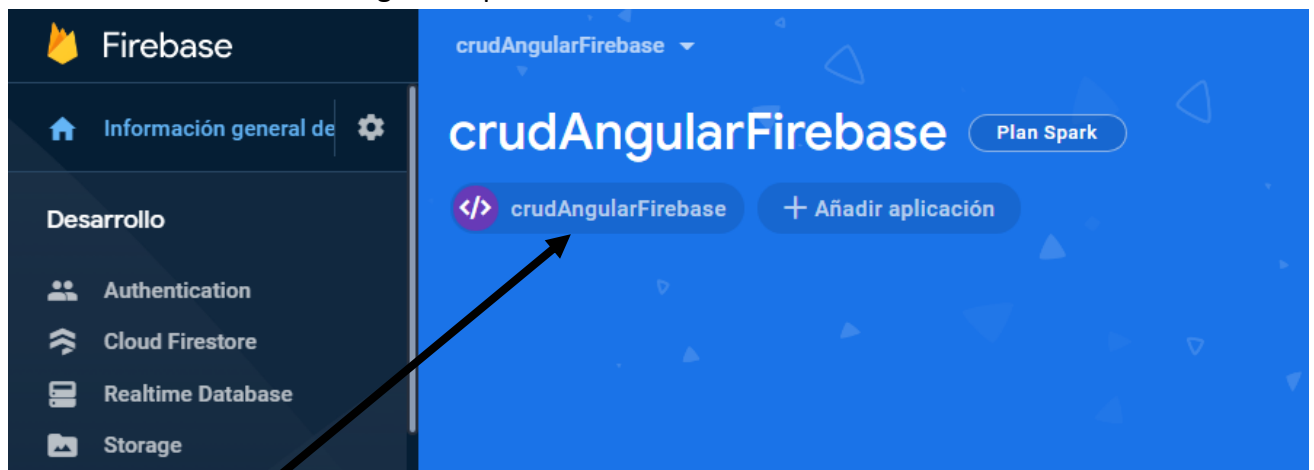
Cuando tengas todo listo, despliega tu aplicación web
Incluye los archivos estáticos (p. ej., HTML, CSS o JS) en el directorio de despliegue de tu aplicación, que se configura como "public" de forma predeterminada. Luego, ejecuta el siguiente comando desde el directorio "root" de la aplicación:

```
$ firebase deploy
```

Después del despliegue, consulta tu aplicación en [crudangularfirebase-faa00.web.app](#)
¿Necesitas ayuda? Consulta la [documentación de Hosting](#)

[Ir a la consola](#)

22. El resultado final tendrá la siguiente pantalla



23. Ir a la opción

24. Luego al engranaje



25. Ir a Opción **general**, ir al final de la página y copiar la configuración para conectarse desde Angular a Firebase

Firebase SDK snippet

☐ Automático  ☐ Red CDN  ☒ Configuración 

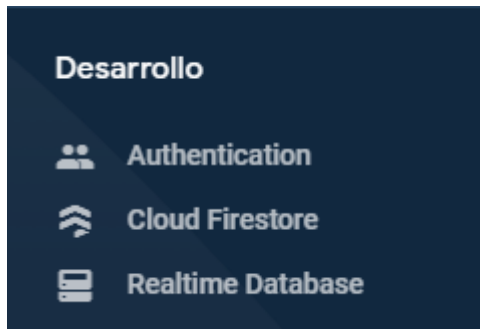
Antes de utilizar cualquier servicio de Firebase, copia y pega estas secuencias de comandos en la parte inferior de la etiqueta <body>:

```
const firebaseConfig = {
  apiKey: "AIzaSyDX9GW-4LgfkVK7jbxWDPqKgqZW6x22rI",
  authDomain: "crudangularfirebase-faa00.firebaseio.com",
  databaseURL: "https://crudangularfirebase-faa00.firebaseio.com",
  projectId: "crudangularfirebase-faa00",
  storageBucket: "crudangularfirebase-faa00.appspot.com",
  messagingSenderId: "1016092009193",
  appId: "1:1016092009193:web:2026a6968ee8e9b7d58678",
  measurementId: "G-C54Y3N5GG4"
};
```

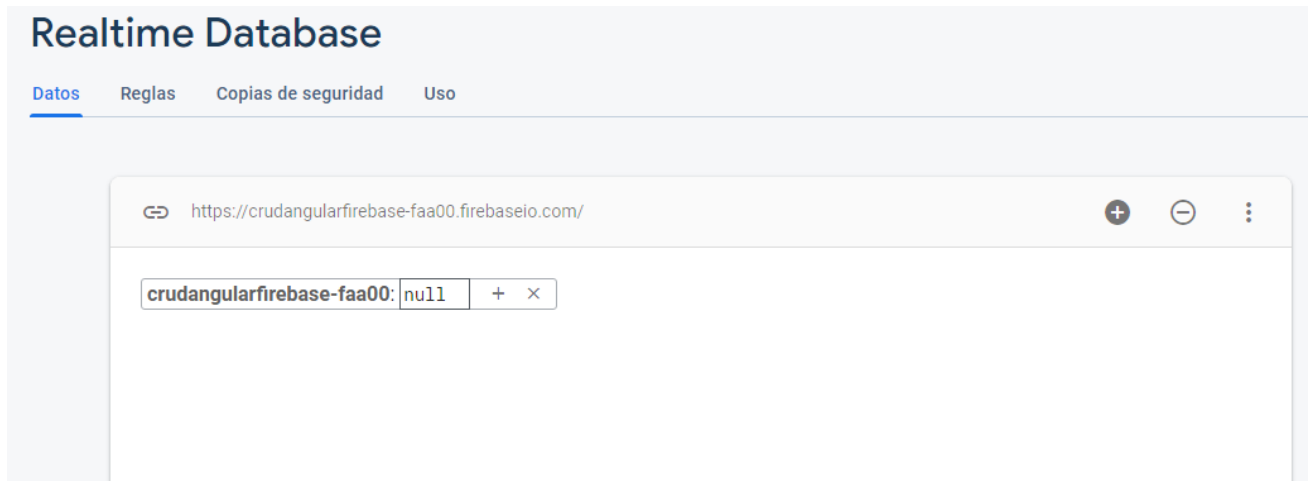
26. Copiar las siguientes líneas, para llevarlas al proyecto angular al archivo configuración app/environments/environment.ts

```
export const environment = {
  production: false,
  firebase : {
    apiKey: "AIzaSyDX9GW-4LgfkVK7jbxWDPqKgqZWo6x22rI",
    authDomain: "crudangularfirebase-faa00.firebaseio.com",
    databaseURL: "https://crudangularfirebase-faa00.firebaseio.com",
    projectId: "crudangularfirebase-faa00",
    storageBucket: "crudangularfirebase-faa00.appspot.com",
    messagingSenderId: "1016092009193",
    appId: "1:1016092009193:web:2026a6968ee8e9b7d58678",
    measurementId: "G-C54Y3N5GG4"
  }
};
```

27. Regresamos a firebase , para crear la base de datos / Realtime Database



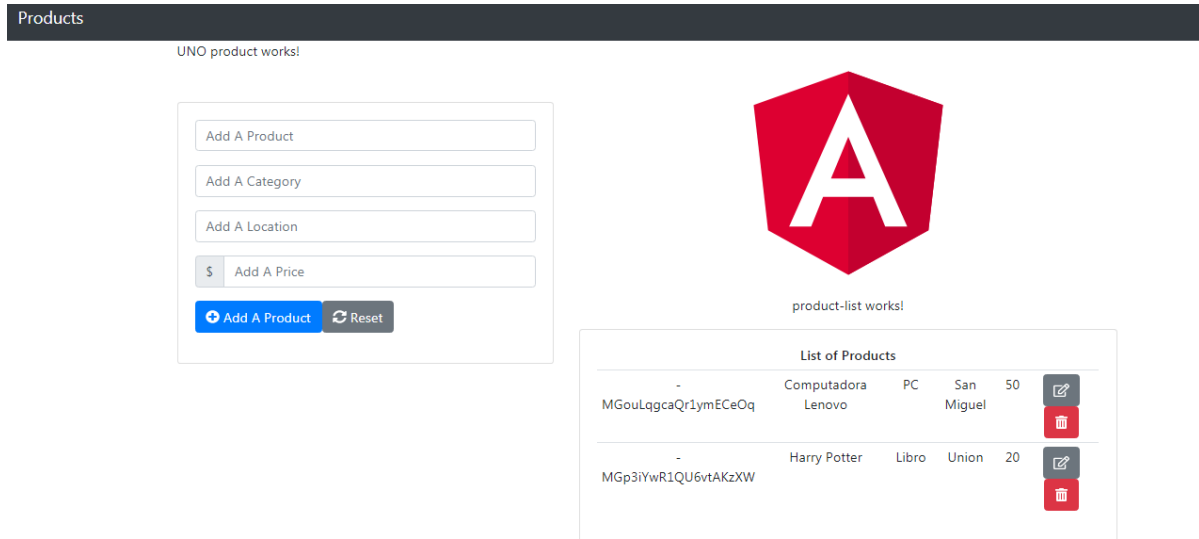
28. Mostrará una pantalla similar, la base la dejaremos vacía:



29. Vamos a la opción reglas y configuramos / read y write = true y luego presionar **publicar**



30. El resultado final será una pantalla similar:



V. DISCUSION DE RESULTADOS

1. Este mismo ejercicio de **crudAngularFirebase** de producto, se va a replicar para el ejemplo uno de la Guia#04, del formulario **alumno**, además se debe de subir al **Hosting de Firebase**.

CRUD ANGULAR LOCAL CON ARREGLO

1	- Alex - Campos - 35
2	- Maria - Lopez - 20
3	- Juan - Castro - 25

Agregar Nuevo Alumno

VII. BIBLIOGRAFIA

- Flanagan, David. JavaScript La Guía Definitiva. 1ra Edición. Editorial ANAYA Multimedia. 2007. Madrid, España.
- Tom Negrito / Dori Smith. JavaScript y AJAX para diseño web. Editorial Pearson Prentice Hall. Madrid, España. 2007.