

Nombre del instructor	Adolfo Centeno Tellez		
Área temática	Ciencia de datos		
Rol	Data Scientist		
Competencia	Full Stack		
Subcompetencia	Frontend		
Objetivo Módulo Conocimiento			
Módulo Conocimiento (5h en total)			
	Duración	Objetivos	Subtemas
Tema 3 - Concepto de Service y consumo de una API en Angular	1 hora	Definir el concepto de Service en Angular, crear una Service básico que consuma una base de datos en la nube para realizar las operaciones básicas de lectura de una base de datos no relacional.	1.- Concepto y creación de Services en Angular 2.- Creación de Services que permitan la conectividad a una base de datos de Google Firestore

Introducción al tema

En los temas anteriores se creó el ambiente de desarrollo para Angular, formado por git, nodejs 20.0 y Angular 18.0. Además se creó un proyecto en Angular llamado mycv, se crearon los componentes necesarios para crear nuestro propio CV al estilo de Elon Musk

Al finalizar este módulo crearás una base de datos no relacional llamada Firestore, crearas las estructuras de datos para cada uno de los componentes del proyecto. Entenderás el concepto de Servicio, serás capaz de crear los

servicios usando angular CLI, codificarlas cada servicio para recuperar los datos para el llenado de cada componente, modificaras los componentes para inyectar los servicios y visualizar la información.

SUBTEMA 1: Concepto y creación de Services en Angular

Angular es un framework basado en componentes. Pero existen otros tipos de artefactos que podemos usar para organizar el código de nuestras aplicaciones de una manera más lógica y fácil de mantener.

Los servicios son de igual importancia a los componentes, en esta sección iniciamos con la descripción de los servicios o "services", son una de las piezas fundamentales en el desarrollo de aplicaciones Angular. Veremos qué es un servicio y cómo dar nuestros primeros pasos en su creación y utilización en un proyecto.

Básicamente un servicio lo podemos definir como un proveedor de datos, que mantiene lógica de acceso a ellos, los servicios son consumidos por los componentes, que delegan en ellos la responsabilidad de acceder a la información y la realización de operaciones con los datos.

Para crear un servicio usamos el Angular CLI con el comando

- `ng generate service <service-name>`

Para nuestro proyecto, crearemos servicios que nos permitirán conectarnos a una base de datos de Firestore.

Nos aseguramos que estemos en la raíz de nuestro proyecto (debemos tener a la vista `node_modules` y `package.json`) :

```
$ ls -la
```

```
[→ mycv git:(master) ✘ ls -la
total 1016
drwxr-xr-x  17 adsoft  staff      544 Nov 18 18:03 .
drwxr-x---+ 79 adsoft  staff     2528 Dec 19 13:52 ..
drwxr-xr-x   3 adsoft  staff       96 Nov 18 18:03 .angular
-rw-r--r--   1 adsoft  staff      274 Nov 18 16:24 .editorconfig
drwxr-xr-x  12 adsoft  staff     384 Nov 18 16:26 .git
-rw-r--r--   1 adsoft  staff      587 Nov 18 16:24 .gitignore
drwxr-xr-x   5 adsoft  staff     160 Nov 18 16:24 .vscode
-rw-r--r--   1 adsoft  staff     1065 Nov 18 16:24 README.md
-rw-r--r--   1 adsoft  staff     2582 Nov 18 16:24 angular.json
drwxr-xr-x  567 adsoft  staff   18144 Nov 18 16:26 node_modules
-rw-r--r--   1 adsoft  staff    484245 Nov 18 16:26 package-lock.json
-rw-r--r--   1 adsoft  staff     1035 Nov 18 16:24 package.json
drwxr-xr-x   3 adsoft  staff       96 Nov 18 16:24 public
drwxr-xr-x   6 adsoft  staff     192 Nov 18 16:24 src
-rw-r--r--   1 adsoft  staff      424 Nov 18 16:24 tsconfig.app.json
-rw-r--r--   1 adsoft  staff     1021 Nov 18 16:24 tsconfig.json
-rw-r--r--   1 adsoft  staff      434 Nov 18 16:24 tsconfig.spec.json
```

Ahora nos movemos a la carpeta src/app, ejecutamos el comando **ls -la** para verificar que estemos en la carpeta del código de la aplicación.

```
$ cd src/app
```

```
$ ls -la
```

```
[→ mycv git:(master) ✘ cd src/app
[→ app git:(master) ✘ ls -la
total 40
drwxr-xr-x 15 adsoft  staff  480 Nov 19 16:54 .
drwxr-xr-x   6 adsoft  staff  192 Nov 18 16:24 ..
-rw-r--r--   1 adsoft  staff    0 Nov 18 16:24 app.component.css
-rw-r--r--   1 adsoft  staff  420 Nov 19 16:54 app.component.html
-rw-r--r--   1 adsoft  staff  910 Nov 18 16:24 app.component.spec.ts
-rw-r--r--   1 adsoft  staff  957 Nov 19 14:08 app.component.ts
-rw-r--r--   1 adsoft  staff  310 Nov 18 16:24 app.config.ts
-rw-r--r--   1 adsoft  staff   77 Nov 18 16:24 app.routes.ts
drwxr-xr-x   6 adsoft  staff  192 Nov 18 23:39 certificates
drwxr-xr-x   6 adsoft  staff  192 Nov 18 23:34 education
drwxr-xr-x   6 adsoft  staff  192 Nov 18 23:22 header
drwxr-xr-x   6 adsoft  staff  192 Nov 18 23:42 interests
drwxr-xr-x   6 adsoft  staff  192 Nov 18 23:41 languages
drwxr-xr-x   6 adsoft  staff  192 Nov 18 23:36 skills
drwxr-xr-x   6 adsoft  staff  192 Nov 18 23:31 work-experience
[→ app git:(master) ✘ ]
```

Ahora, creamos una carpeta especial para nuestros servicios y nos movemos dentro de la carpeta.

```
$ mkdir services
```

```
$ cd services
```

```
[→ app git:(master) ✘ mkdir services
[→ app git:(master) ✘ cd services
[→ services git:(master) ✘ ls -la
total 0
drwxr-xr-x  2 adsoft  staff  64 Dec 19 14:07 .
drwxr-xr-x 16 adsoft  staff  512 Dec 19 14:07 ..
→ services git:(master) ✘ █
```

Una vez dentro services, creamos una carpeta para cada uno de los servicios que crearemos. Para efectos didácticos crearemos un servicio para cada sección de nuestro CV, así en la siguiente sección programaremos cada servicio para recuperar información específica para cada componente de la aplicación.

```
$ mkdir header-service
```

```
$ mkdir work-experience-service
```

```
$ mkdir education-service
```

```
$ mkdir skills-service
```

```
$ mkdir certificates-service
```

```
$ mkdir languages-service
```

```
$ mkdir interests-service
```

```

→ services git:(master) ✘ mkdir header-service
→ services git:(master) ✘ mkdir work-experience-service
→ services git:(master) ✘ mkdir education-service
→ services git:(master) ✘ mkdir skills-service
→ services git:(master) ✘ mkdir certificates-service
→ services git:(master) ✘ mkdir languages-service
→ services git:(master) ✘ mkdir interests-service
→ services git:(master) ✘ ls -la
total 0
drwxr-xr-x  9 adsoft  staff  288 Dec 19 14:25 .
drwxr-xr-x 16 adsoft  staff  512 Dec 19 14:07 ..
drwxr-xr-x  2 adsoft  staff   64 Dec 19 14:25 certificates-service
drwxr-xr-x  2 adsoft  staff   64 Dec 19 14:25 education-service
drwxr-xr-x  2 adsoft  staff   64 Dec 19 14:24 header-service
drwxr-xr-x  2 adsoft  staff   64 Dec 19 14:25 interests-service
drwxr-xr-x  2 adsoft  staff   64 Dec 19 14:25 languages-service
drwxr-xr-x  2 adsoft  staff   64 Dec 19 14:25 skills-service
drwxr-xr-x  2 adsoft  staff   64 Dec 19 14:25 work-experience-service
→ services git:(master) ✘ pwd
/Users/adsoft/mycv/src/app/services
→ services git:(master) ✘

```

Hasta ahora tenemos la estructura para cada servicio dentro de services, procedemos a crear cada servicio dentro de la carpeta correspondiente.

\$ ng generate service header-service/header

```

→ services git:(master) ✘ ng generate service header-service/header
CREATE src/app/services/header-service/header.service.spec.ts (357 bytes)
CREATE src/app/services/header-service/header.service.ts (135 bytes)
→ services git:(master) ✘

```

Nos creará 2 archivos dentro de la carpeta header-service: **header-services.ts** tipo Typescript, podemos visualizarlo con:

\$ cat header-service/header.service.ts

```
[→ services git:(master) ✘ cat header-service/header.service.ts
import { Injectable } from '@angular/core';

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

  constructor() { }
}
```

Además crea su correspondiente archivo para generar sus pruebas unitarias (que se analizará en el tema 4) **header.service.spec.ts**.

\$ cat header-service/header-service.spec.ts

```
[→ services git:(master) ✘ cat header-service/header.service.spec.ts
import { TestBed } from '@angular/core/testing';

import { HeaderService } from './header.service';

describe('HeaderService', () => {
  let service: HeaderService;

  beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(HeaderService);
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });
});
```

Hemos creado el service para header, repetimos el proceso para el resto de los componentes.

\$ ng generate service work-experience-service/work-experience

```
→ services git:(master) ✘ ng generate service work-experience-service/work-experience
CREATE src/app/services/work-experience-service/work-experience.service.spec.ts (398 bytes)
CREATE src/app/services/work-experience-service/work-experience.service.ts (143 bytes)
→ services git:(master) ✘
```

\$ ng generate service education-service/education

```
→ services git:(master) ✘ ng generate service education-service/education
CREATE src/app/services/education-service/education.service.spec.ts (372 bytes)
CREATE src/app/services/education-service/education.service.ts (138 bytes)
→ services git:(master) ✘
```

\$ ng generate service skills-service/skills

```
→ services git:(master) ✘ ng generate service skills-service/skills
CREATE src/app/services/skills-service/skills.service.spec.ts (357 bytes)
CREATE src/app/services/skills-service/skills.service.ts (135 bytes)
→ services git:(master) ✘
```

\$ ng generate service certificates-service/certificates

```
→ services git:(master) ✘ ng generate service certificates-service/certificates
CREATE src/app/services/certificates-service/certificates.service.spec.ts (387 bytes)
CREATE src/app/services/certificates-service/certificates.service.ts (141 bytes)
→ services git:(master) ✘
```

\$ ng generate service languages-service/languages

```
→ services git:(master) ✘ ng generate service languages-service/languages
CREATE src/app/services/languages-service/languages.service.spec.ts (372 bytes)
CREATE src/app/services/languages-service/languages.service.ts (138 bytes)
→ services git:(master) ✘
```

\$ ng generate service interests-service/interests

```
→ services git:(master) ✘ ng generate service interests-service/interests
CREATE src/app/services/interests-service/interests.service.spec.ts (372 bytes)
CREATE src/app/services/interests-service/interests.service.ts (138 bytes)
→ services git:(master) ✘
```

Inyección de dependencias.

Ahora, analizaremos Angular y la inyección de dependencias, que vamos a usar para poder disponer del servicio en un componente.

Como en otros frameworks, en Angular la inyección de dependencias se realiza por medio del constructor. En el constructor del componente, que hasta ahora habíamos dejado siempre vacío, podemos declarar cualquiera de los servicios

que vamos a usar y que Angular se encargará de proporcionar, sin que tengamos que realizar nosotros ningún trabajo adicional.

Esto es tan sencillo como declarar como parámetro la dependencia en el constructor del componente

Ahora, inyectamos HeaderService en el componente Header, empezaremos actualizando header-service.ts, agregamos la línea 7, con una variable **accesoHeader**, que después vamos a visualizar en el componente Header.

\$ nano header-service/header.service.ts

```

1 import { Injectable } from '@angular/core';
2
3 @Injectable({
4   providedIn: 'root'
5 })
6 export class HeaderService {
7   accesoHeader = 'header service running...';
8
9   constructor() { }
10 }
```

Actualizamos el componente **Header** con el código necesario para injectar el service. Empezamos importando el servicio.

```
2 import { HeaderService } from '../services/header-service/header.service';
```

Crearemos una **constructor** para crear una instancia pública del service.

```

11   constructor(public headerService: HeaderService)
12   {
13     console.log(this.headerService);
14 }
```

Para implementar la inyección del servicio, regresamos un nivel para estar en src/app con cd ..

```
$ cd ..
```

```
$ nano header/header.component.ts
```

```

1 import { Component } from '@angular/core';
2 import { HeaderService } from '../services/header-service/header.service';
3
4 @Component({
5   selector: 'app-header',
6   templateUrl: './header.component.html',
7   styleUrls: ['./header.component.css']
8 })
9 export class HeaderComponent {
10
11   constructor(public headerService: HeaderService)
12   {
13     console.log(this.headerService);
14   }
15 }
```

Para comprobar, imprimimos la variable **accesoHeader**, en el HTML de header usando la directiva de angular **{} {}**, que permite visualizar cualquier variable u objeto TypeScript dentro del HTML.zw

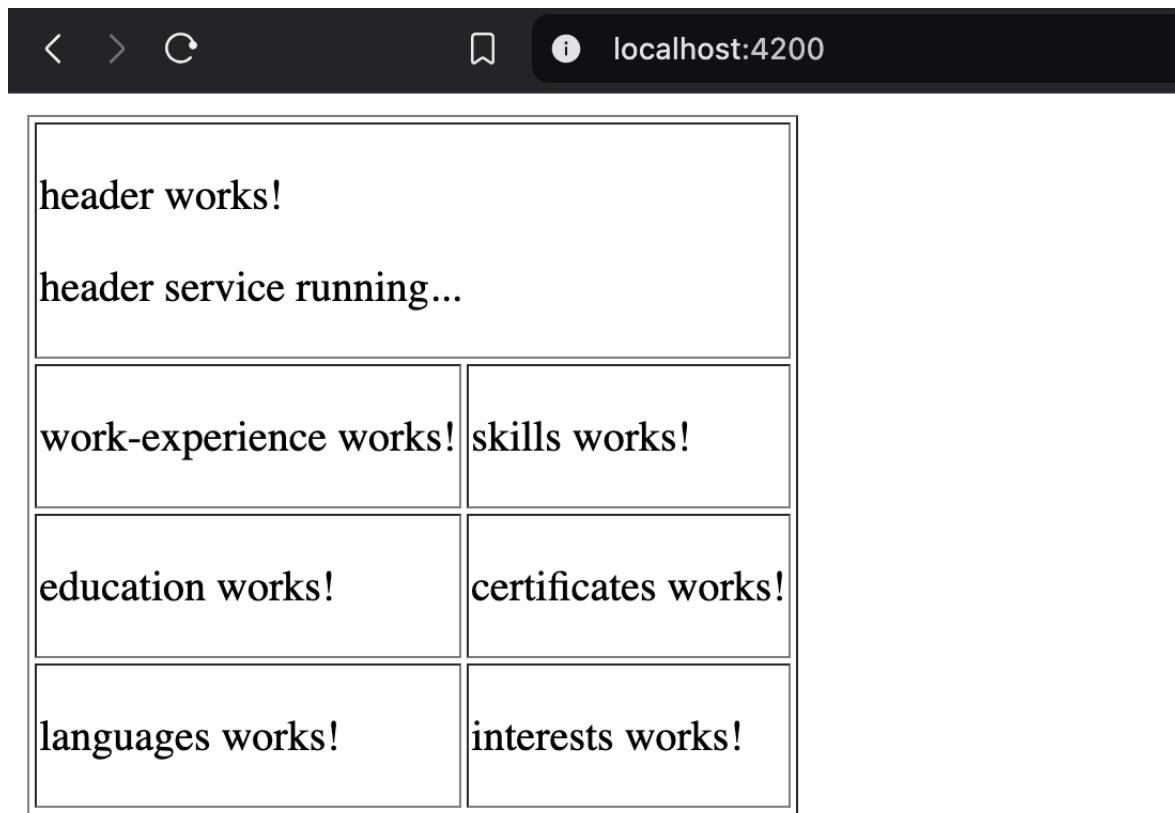
```
$ nano header/header.component.html
```

```

1 <p>header works!</p>
2 <p>
3 {{headerService.accesoHeader}}
4 </p>
```

Usando Angular CLI, arrancamos en modo frontend usando el comando ng serve, visualizamos la aplicación en la dirección: <http://localhost:4200>, así validamos que

el servicio está insertado en el componente Header, visualizando la variable **accesHeader** en el HTML.



Repetimos el proceso para el servicio work-experience, nos aseguramos de estar en la carpeta services.

```
→ services git:(master) ✘ pwd
/Users/adsoft/mycv/src/app/services
→ services git:(master) ✘ ls -la
total 0
drwxr-xr-x  9 adsoft  staff  288 Dec 19 14:25 .
drwxr-xr-x 16 adsoft  staff  512 Dec 19 14:07 ..
drwxr-xr-x  4 adsoft  staff  128 Dec 20 00:20 certificates-service
drwxr-xr-x  4 adsoft  staff  128 Dec 20 00:19 education-service
drwxr-xr-x  4 adsoft  staff  128 Dec 20 01:19 header-service
drwxr-xr-x  4 adsoft  staff  128 Dec 20 00:21 interests-service
drwxr-xr-x  4 adsoft  staff  128 Dec 20 00:20 languages-service
drwxr-xr-x  4 adsoft  staff  128 Dec 20 00:19 skills-service
drwxr-xr-x  4 adsoft  staff  128 Dec 20 00:18 work-experience-service
→ services git:(master) ✘
```

\$ nano work-experience-service/work-experience.service.ts

```

1 import { Injectable } from '@angular/core';
2
3 @Injectable({
4   providedIn: 'root'
5 })
6 export class WorkExperienceService {
7
8   accesoWorkExperience = "work experience running...";
9
10  constructor() { }
11 }
```

Ahora modificamos el componente **WorkExperience**

```
$ cd ..
$ nano work-experience/work-experience.component.ts
```

```

1 import { Component } from '@angular/core';
2 import { WorkExperienceService } from '../services/work-experience-service/workexperience.service';
3
4 @Component({
5   selector: 'app-work-experience',
6   templateUrl: './work-experience.component.html',
7   styleUrls: ['./work-experience.component.css']
8 })
9 export class WorkExperienceComponent {
10
11  constructor(public workExperienceService: WorkExperienceService)
12  {
13    console.log(this.workExperienceService);
14  }
15 }
```

Visualizamos la variable **workExperienceService** en el html del componente.

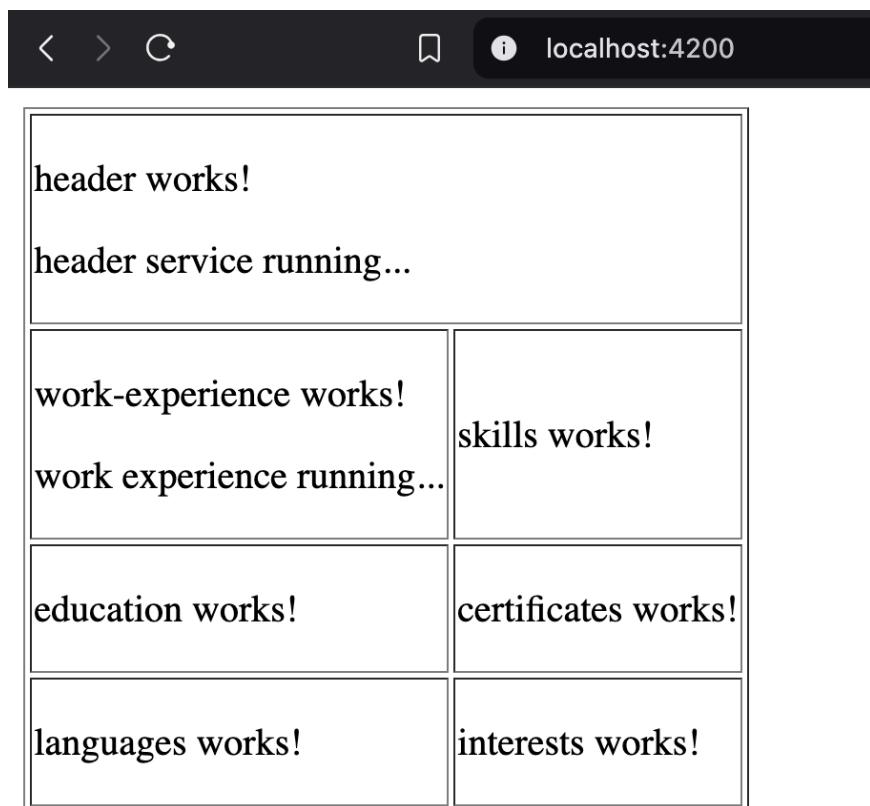
```
$ nano work-experience/work-experience.component.html
```

```

1 <p>work-experience works!</p>
2 <p>
3   {{workExperienceService.accesoWorkExperience}}
4 </p>
```

Probamos nuevamente la aplicación con:

```
$ ng serve
```



A screenshot of a web browser window showing a grid of cards. The browser header includes icons for back, forward, and refresh, and the URL 'localhost:4200'. The grid has four columns and four rows. The first row contains two cards: 'header works!' and 'header service running...'. The second row contains two cards: 'work-experience works!' and 'skills works!'. The third row contains two cards: 'education works!' and 'certificates works!'. The fourth row contains two cards: 'languages works!' and 'interests works!'. All text in the cards is bold.

header works!	header service running...
work-experience works!	skills works!
education works!	certificates works!
languages works!	interests works!

Hasta ahora, hemos trabajado con services, hemos creado servicios para los componente header y work-experience. Además hemos inyectado los servicios en sus respectivos componentes y visualizado en el HTML del componente.

Sala de inspiración

El conocimiento que has adquirido hasta ahora, te permite construir e inyectar servicios en Angular.

- Como tarea adicional crea los servicios para las secciones de: education, skills, certificates, languages e interests
- Una vez creado los servicios inyecta cada servicio en su respectivo componente
- Modifica el html de cada componente para visualizar la variable del servicio
- Comprueba si puedes visualizar la aplicación de forma local, verifica con: ng serve

Sala de pruebas

Plantilla Nivel de Conocimiento

Lee cada una de las preguntas y selecciona la respuesta que consideres correcta de acuerdo a tu experiencia hasta ahora.

1. ¿ Con qué comando de Angular CLI generamos un service llamado **data-service** ?

Opción a: ng g c data-service	Incorrecta: <i>La sintaxis correcta es:</i> \$ ng generate service data-service
Opción b: ng generate data-service	Incorrecta: <i>La sintaxis correcta es:</i> \$ ng generate service data-service
Opción c: ng generate component data-service	Incorrecta: <i>La sintaxis correcta es:</i> \$ ng generate service data-service
Opción d: ng generate service data-service	CORRECTA: <i>La sintaxis correcta es:</i> \$ ng generate service data-service

2. ¿ Qué archivos generaría el comando anterior ?

Opción a: data-service.spec.ts data-service.ts	Incorrecta El comando ng generate service data-service creará los archivos: data-service.service.spec.ts data-service.service.ts
---	--

Plantilla Nivel de Conocimiento

Opción b: data-service.spec.ts data-service.service.ts	Incorrecta El comando <i>ng generate service data-service</i> creará los archivos: data-service.service.spec.ts data-service.service.ts
Opción c: data-service.service.spec.ts data-service.service.ts	CORRECTA El comando <i>ng generate service data-service</i> creará los archivos: data-service.service.spec.ts data-service.service.ts
Opción d: data-service.component.spec.ts data-service.component.ts	Incorrecta: El comando <i>ng generate service data-service</i> creará los archivos: data-service.service.spec.ts data-service.service.ts

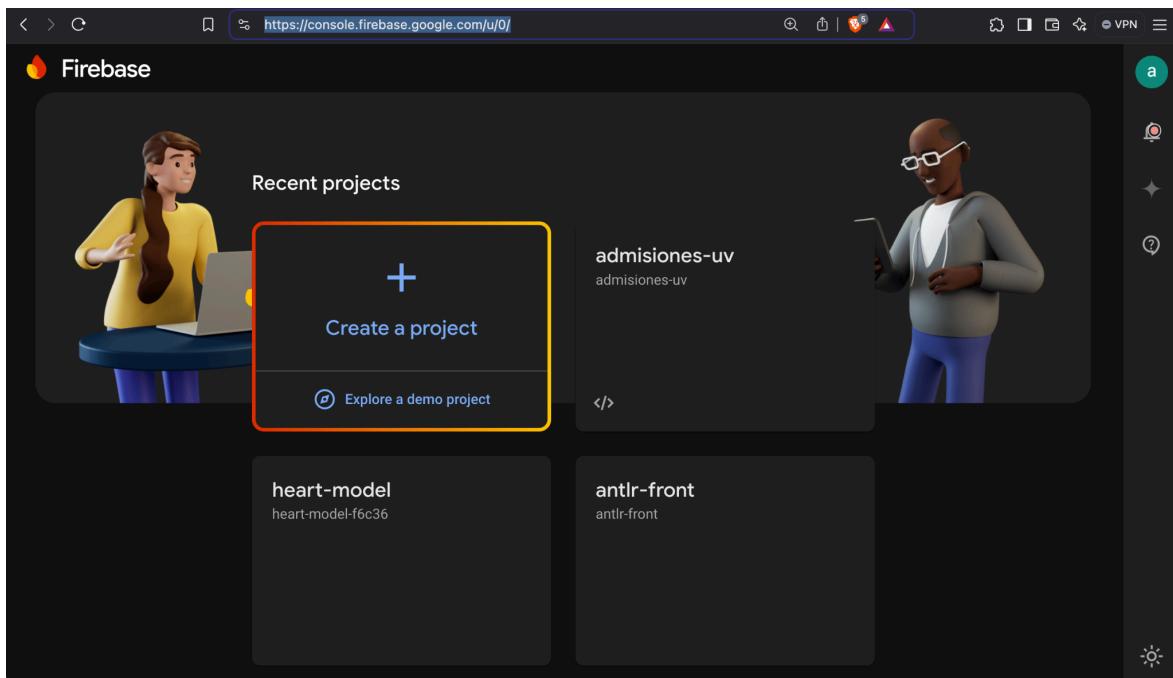
3.- ¿ Cómo imprimes en el html de un componente la instancia del servicio llamada: **dataService.hello** ?

Opción a: { dataService.hello }	Incorrecta: La sintaxis correcta es: {{ dataService.hello }}
Opción b: console.log(dataService.hello)	Incorrecta: La sintaxis correcta es: {{ dataService.hello }}

Opción c: <code>{{ dataService.hello }}</code>	CORRECTA La sintaxis correcta es: <code>{{ dataService.hello }}</code>
Opción d: <code><h3> dataService.hello </h3></code>	Incorrecta: La sintaxis correcta es: <code>{{ dataService.hello }}</code>

SUBTEMA 2: Creación de Services que permitan la conectividad a una base de datos de Google Firestore

En este tema crearemos servicios para conectar nuestra aplicación a una base de datos no relacional Firestore que es parte de la Plataforma como Servicio (PaaS) llamada Google firebase disponible en <https://console.firebaseio.google.com/>



Firebase de Google es una plataforma en la nube para el **desarrollo de aplicaciones web y móviles**, y se encuentra actualmente disponible para distintas plataformas como iOS, Android, web, flutter y Unity.

Firebase fue creada en 2011 pero pasó a ser parte de Google en 2014, comenzando como una base de datos en tiempo real. Sin embargo, se añadieron más y más funciones que, en parte, permitieron agrupar los SDK de productos de Google con distintos fines, facilitando su uso.

Firebase es la forma más sencilla de crear aplicaciones web y móviles, procurando que el trabajo sea más rápido, pero sin renunciar a la calidad requerida y que es especialmente interesante para que los desarrolladores no dediquen tanto tiempo al backend, tanto en cuestiones de desarrollo como de mantenimiento.

Asimismo, la plataforma Firebase está compuesta por una suite de productos:

1. **Real Time**.- Firebase Realtime es una base de datos NoSQL alojada en la nube que te permite almacenar y sincronizar datos en tiempo real. La sincronización en tiempo real permite que los usuarios accedan a sus datos desde cualquier dispositivo, web o móvil, con facilidad, y los ayuda a trabajar en conjunto. Si un usuario pierde conectividad de internet, los SDK de Realtime Database usan la caché local del dispositivo para publicar y almacenar cambios y cuando el dispositivo se conecta, los datos locales se sincronizan de manera automática.

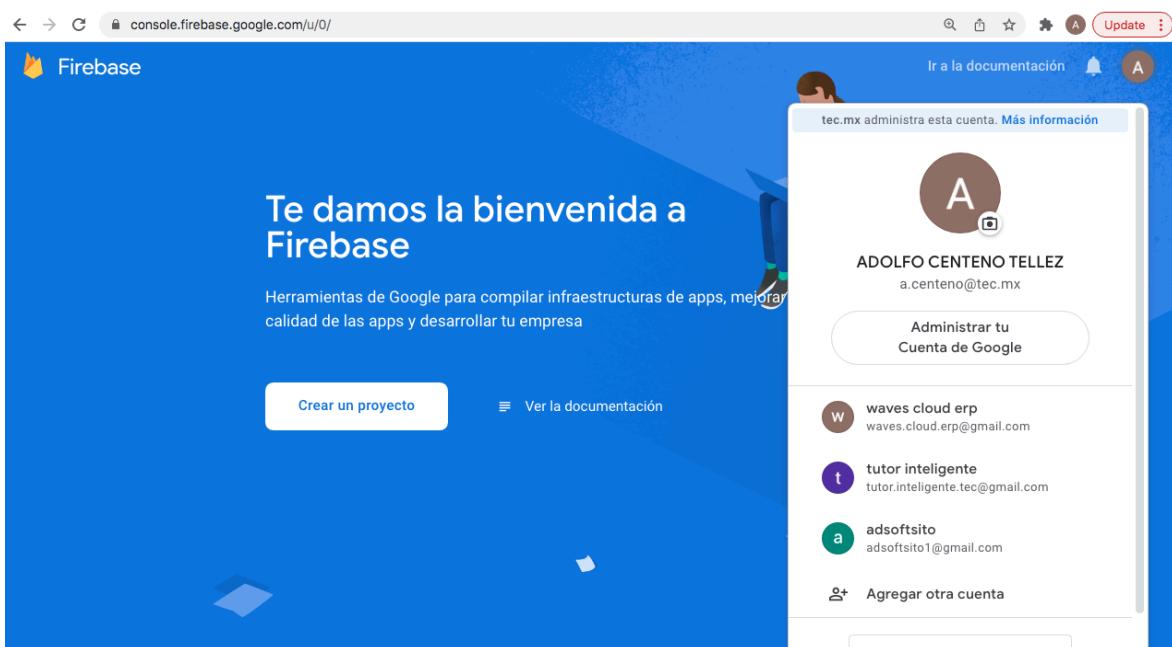
2. **Cloud Firestore.**- Cloud Firestore te permite almacenar, sincronizar y realizar búsquedas inteligentes de datos en una arquitectura altamente escalable, Firestore se analizará a profundidad en este módulo ya que es el producto de firebase ideal para nuestra aplicación web.
3. **Firebase ML.**- Firebase ML es un producto que incorpora un conjunto de APIs de inteligencia artificial para los casos de uso más comunes como: reconocer texto, etiquetar imágenes y reconocer puntos de referencia. A diferencia de otras API integradas en el dispositivo, estas aprovechan la potencia de la tecnología de aprendizaje automático de Google Cloud para brindarte un nivel alto de precisión. Simplemente se transfieren los datos a la API y se creará una solicitud sin interrupciones para los modelos que se ejecutan en Google Cloud y recibirás la información que necesitas con pocas líneas de código
4. **Cloud functions.**- Cloud Functions te permite crear funciones de backend en JavaScript para resolver problemas específicos, las cuales se ejecutan en un entorno de Node.js seguro y administrado y solo se ejecutan cuando se emite un evento específico bajo observación. Cloud functions se activan con productos de Firebase, como cambios en los datos de Realtime Database, el registro de usuarios nuevos mediante Auth o los eventos de conversión en Analytics
5. **Hosting.**- Con Firebase Hosting, puedes publicar un sitio web completo sin complicaciones solo contando con la versión de producción en HTML, CSS y Javascript, sin importar el framework con el que fueron creados como Jquery, Angular, React o Vue. El contenido se publica rápidamente, sin importar la ubicación del usuario, los archivos implementados en Firebase Hosting se almacenan en servidores en la nube de Google.
6. **Cloud Storage.**- Cloud Storage es un servicio de almacenamiento de objetos como imágenes, videos, audios, pdf, xml, entre otros. Es potente, simple y rentable, este servicio agrega seguridad a las operaciones de carga y descarga de archivos de las aplicaciones de Firebase, sin importar la latencia de la red.

Al igual que Firebase Realtime, Firestore mantiene tus datos sincronizados entre apps cliente a través de objetos que escuchan en tiempo real y ofrece soporte sin conexión para dispositivos móviles y la Web, por lo que puedes compilar apps con capacidad de respuesta que funcionan sin importar la latencia de la red ni la conectividad a Internet. Explicas que Firestore también ofrece una integración sin interrupciones con otros productos de Firebase y Google Cloud, incluido el Storage y Cloud Functions.

Crear una cuenta en Firebase

Como primer paso ir a la consola de administración de firebase :
<https://console.firebaseio.google.com/>

NOTA: Si ya te encuentras logueado con alguna cuenta de gmail o institucional, Firebase tomará esta cuenta como tu cuenta por default para Firebase.

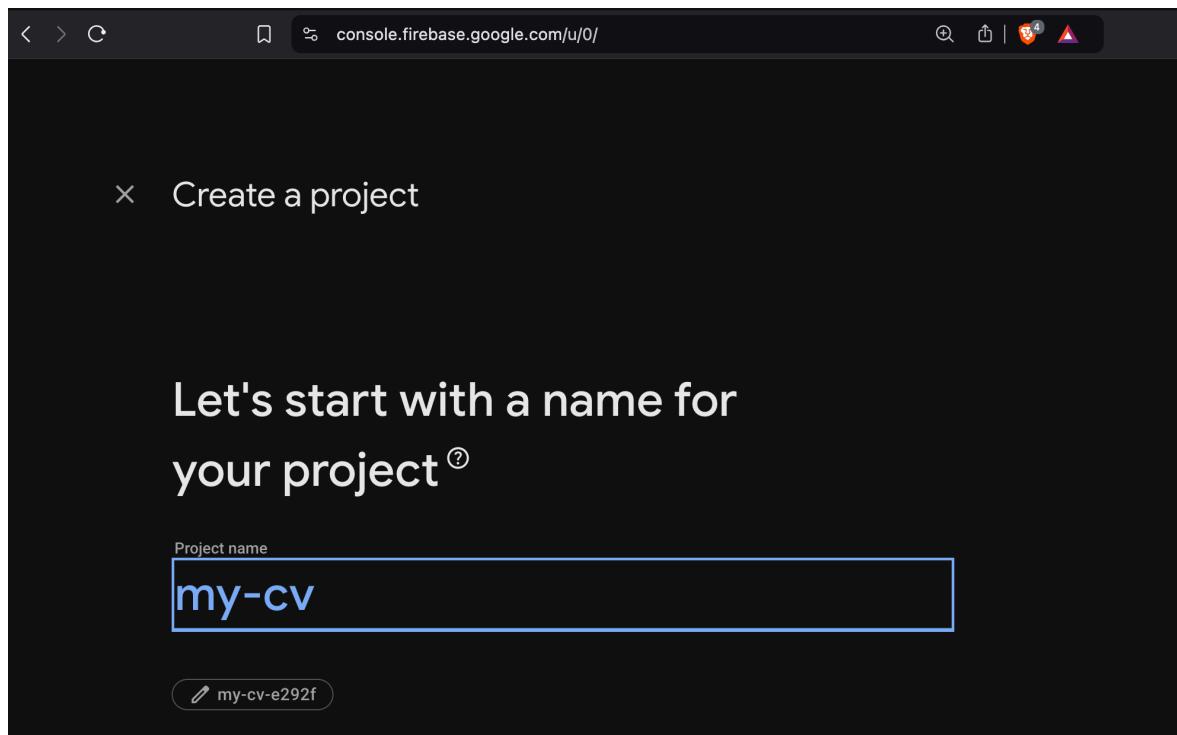


En caso contrario, te pedirá que te registres con alguna cuenta de correo electrónico, la recomendación sería usar la misma cuenta que usaste para Github.

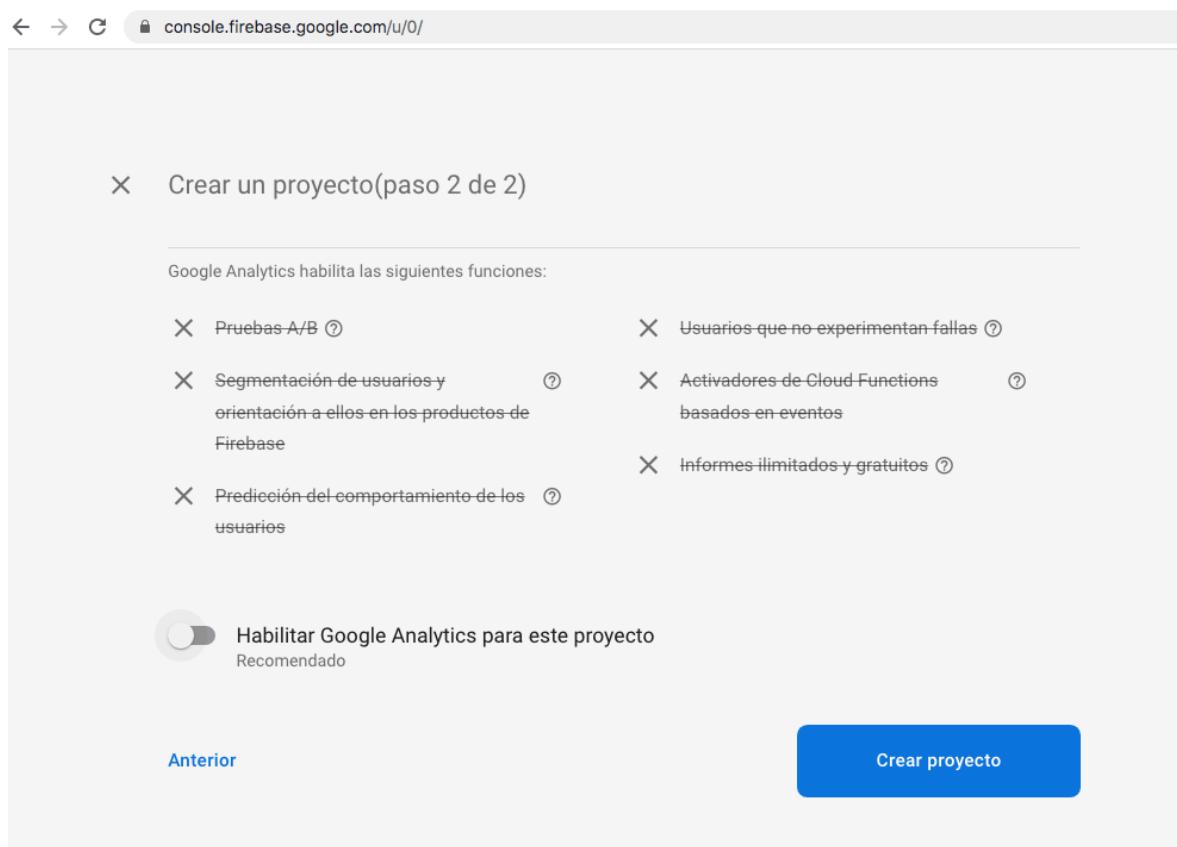
Crear un proyecto

Como primer paso, crearemos un proyecto haciendo click en el botón “Crear un proyecto”, después pondremos un nombre y firebase generará un ID único.

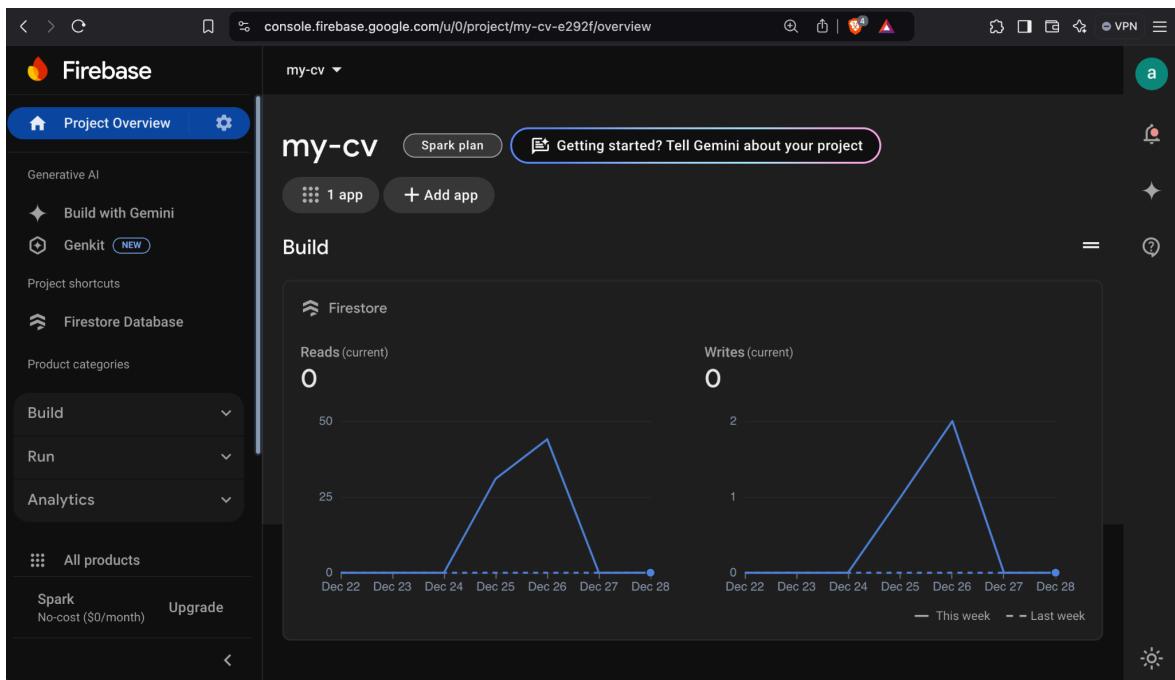
Para nuestro ejemplo, crearemos un proyecto que almacene la información para cada sección de nuestro proyecto de CV.



Posteriormente **deshabilitamos Google Analytics** ya que esta funcionalidad no es requerida para este proyecto, y hacemos clic en “Crear proyecto”



Click en Crear proyecto, posteriormente clic en Continuar y nos aparecerá la consola de administración de nuestro proyecto



Crear una base de datos

Antes de crear nuestra base de datos en Firestore, analizaremos algunos conceptos del modelo de datos de Firebase.

Cloud Firestore es una base de datos NoSQL orientada a los documentos. A diferencia de una base de datos SQL, no hay tablas ni filas; En su lugar, almacena los datos en *documentos*, que se organizan en *colecciones*.

Cada *documento* contiene un conjunto de pares clave-valor. Cloud Firestore está optimizado para almacenar grandes colecciones de documentos pequeños.

Todos los documentos se deben almacenar en colecciones, y pueden contener *subcolecciones* y objetos anidados. Además, ambos pueden incluir campos primitivos, como strings, o tipos de objetos complejos, como listas.

Documentos

En Cloud Firestore, la unidad de almacenamiento es el documento. Un documento es un registro que usa pocos recursos y contiene campos con valores

asignados. Cada documento se identifica con un nombre específico o bien un ID generado automáticamente.

Por ejemplo un documento que representa a un registro de **names (con los atributos index, nombre y sexo de una persona)** puede tener el siguiente aspecto:

```
index: 314
name: "Priscilla"
sex: "F"
```

Cloud Firestore es compatible con diversos tipos de datos para los valores, como **booleanos, números, strings, puntos geográficos, BLOB binarios y TimeStamp**. Además, puedes usar arreglos u objetos anidados, llamados **mapas**, para estructurar datos dentro de un documento.

Un documento en Firestore es muy similar a un JSON; de hecho, básicamente son JSON. Existen algunas ligeras diferencias (como que los documentos en Firebase admiten tipos de datos adicionales y su tamaño se limita a 1 MB), pero regularmente se puede considerar los documentos como registros JSON.

Colecciones

Los **documentos** viven en **colecciones**, que simplemente son contenedores de documentos. Por ejemplo, podrías tener una colección llamada **names** con diversos registros de nombres con un ID único, en la que haya un documento que represente a cada uno:

+ Iniciar colección	+ Agregar documento	+ Iniciar colección
names >	+ Agregar documento 07ftJR9xwUUchYjlxIvE 09YEpSre6Ka3Pzr6VeYW 0DUm7LTx8CeLkUjMCHuB > 0M7kuYnEMmQFuSQBIppu 0MAfiMal410QzFgo3EXI 0W9apk3G7Un7r94nb8Lm 15E5JYwqDdMImx3oROyn 16LnmXoe4sDyyMwthRZN 1I9n2XX7AiKb5PLJ0mEJ 1Mg2A99Ta6CjqmuKK6L4 1hLiqVGrS8FKGdX7vcqd 1nM7DC3jM5UcTWadDEew 1pHessPP5MB52C2KhoOp	+ Iniciar colección + Agregar campo index: 74 name: "Caroline" sex: "F"

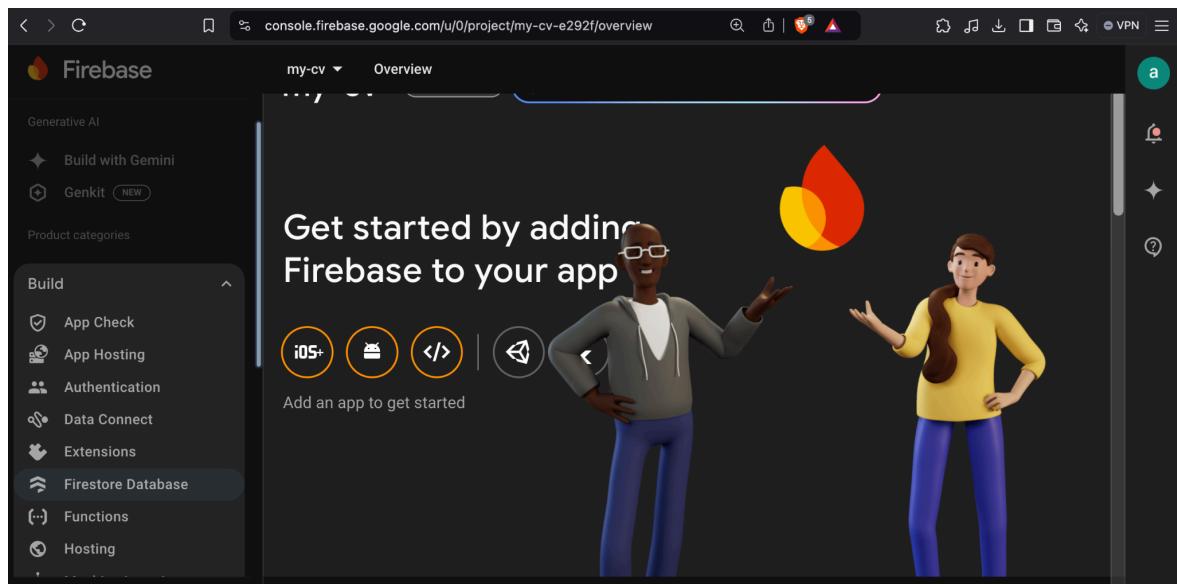
En Cloud Firestore tienes libertad total sobre los campos que pones en cada documento y los tipos de datos que almacenas en esos campos.

NOTA: Los documentos dentro de una misma colección pueden contener campos diferentes o almacenar distintos tipos de datos en esos campos. Sin embargo, se recomienda usar los ***mismos campos y tipos de datos en varios documentos***, de manera que puedas consultarlos con mayor facilidad.

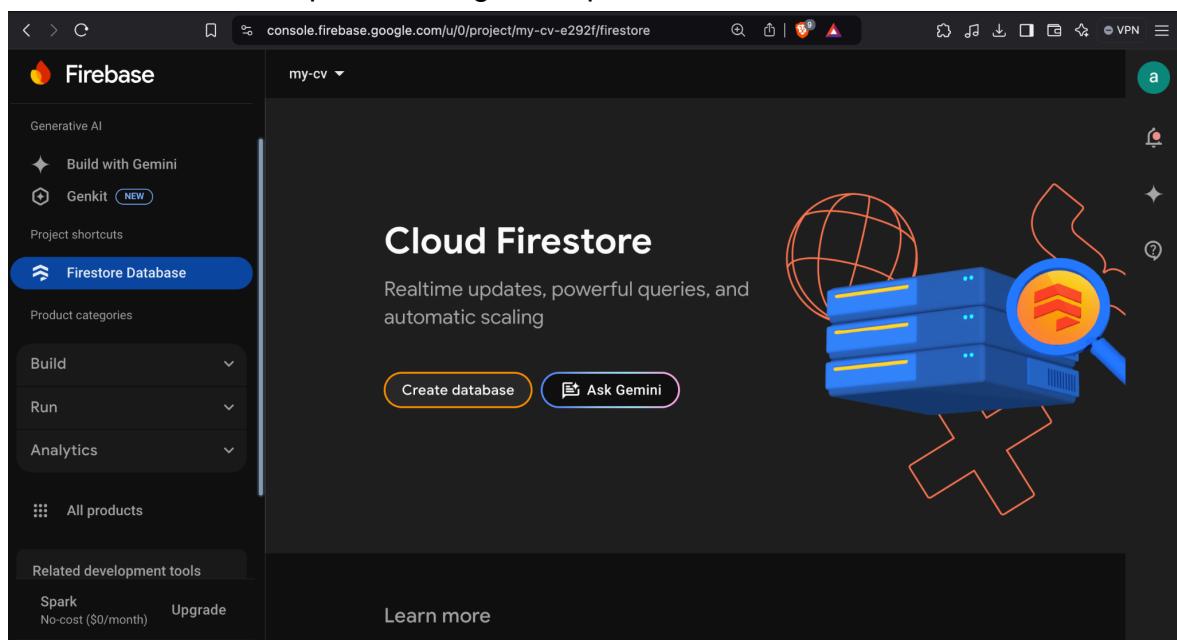
Los nombres de documentos dentro de una colección son únicos. Puedes proporcionar tus propias claves, como los ID de usuario, o puedes dejar que Cloud Firestore cree ***ID aleatorios*** de forma automática.

No es necesario crear ni borrar las colecciones; Cuando se crea el primer documento de una colección, esta se crea automáticamente y por el contrario si borras todos los documentos de una colección, esta deja de existir.

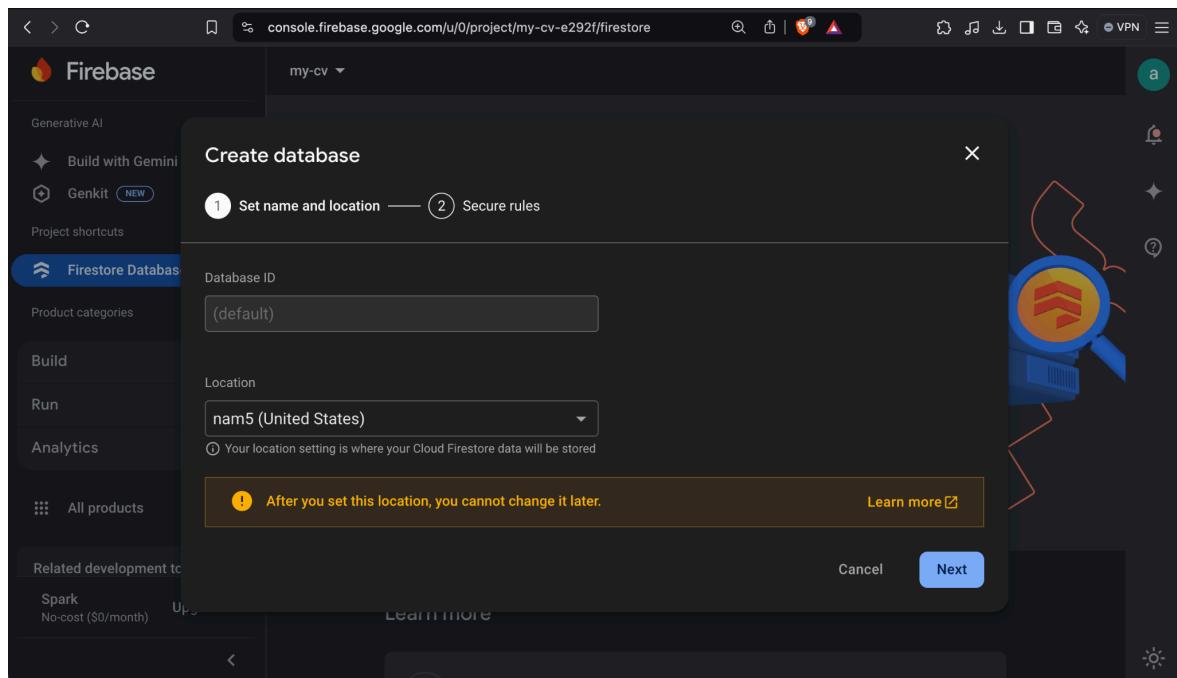
Una vez revisado estos conceptos básicos sobre base de datos en firebase, creamos nuestra primera base de datos en Cloud Firestore. Seleccionamos del menú del lado izquierdo **Firestore Database**.



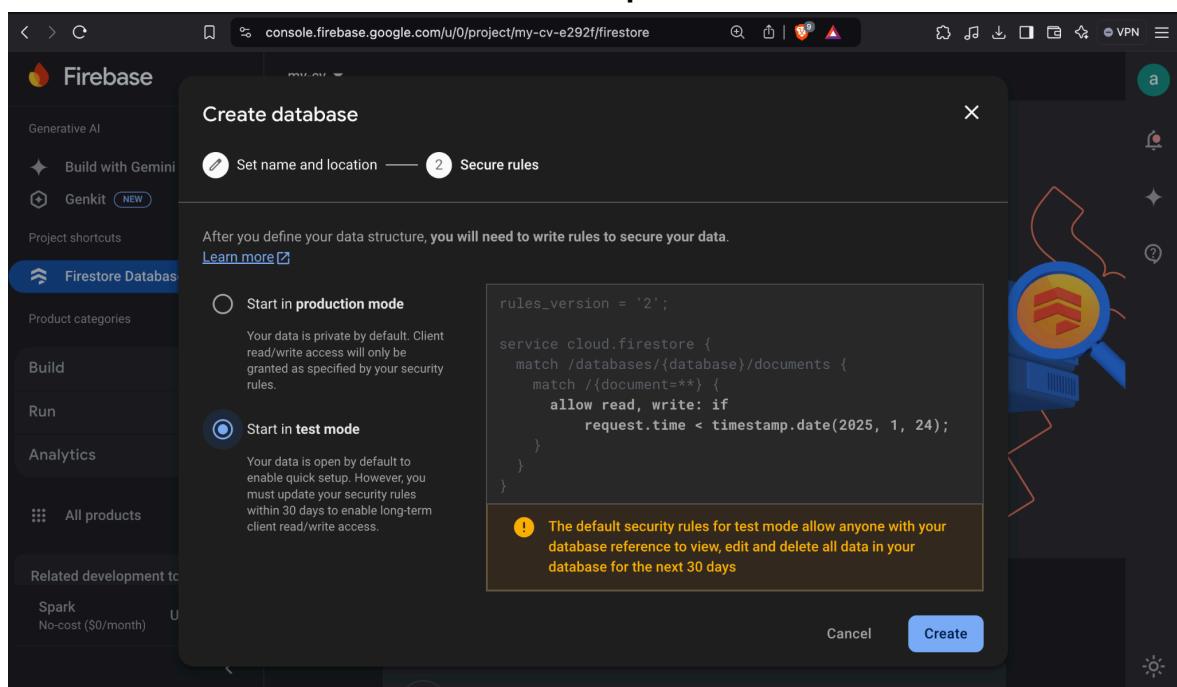
Posteriormente nos aparece la siguiente pantalla, clic en **Crear base de datos**.



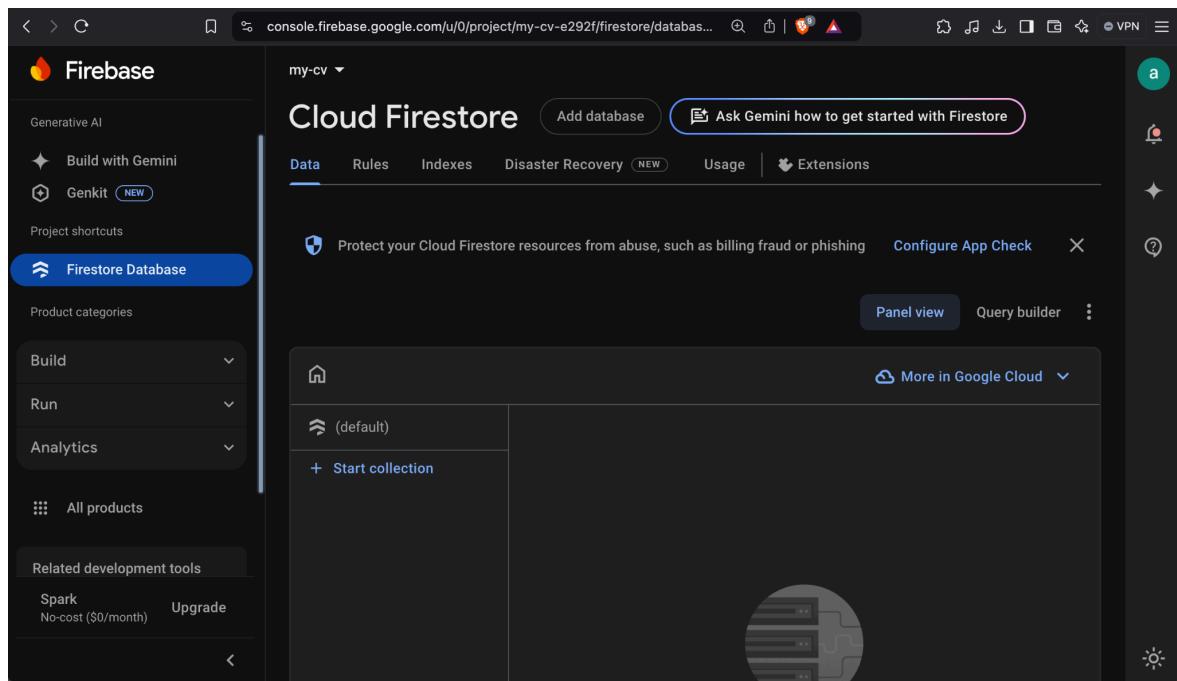
Ahora nos pedirá el datacenter de Google Cloud, deseamos crear nuestra base de datos, se recomienda seleccionar la región más cercana al mercado potencial donde están los clientes del proyecto.



Aparecerá un cuadro de diálogo, crearemos las reglas de seguridad para nuestra base de datos. Seleccionamos el **modo de pruebas**.



Una vez creada la base de datos de Firestore, nos aparecerá la siguiente pantalla, lista para crear las colecciones y documentos.



Crear las estructuras de de la base de datos

En la siguiente tabla, se enumeran los tipos de datos que admite Cloud Firestore :

1. Boolean values
2. Integer and floating-point values, sorted in numerical order
3. Date values
4. Text string values
5. Byte values
6. Cloud Firestore references
7. Geographical point values
8. Array values
9. Map values

En esta sección crearemos las Colecciones de Firestore, para almacenar documentos de **nuestro proyecto**. Crearemos una colección para cada llenar cada uno de los componentes, por tanto tendremos las siguientes colecciones:

1. **header**
2. **work-experience**
3. **education**
4. **skills**
5. **certificates**
6. **languages**

7. interests

Empezaremos por Header, recordamos los requerimientos para el componente Header.



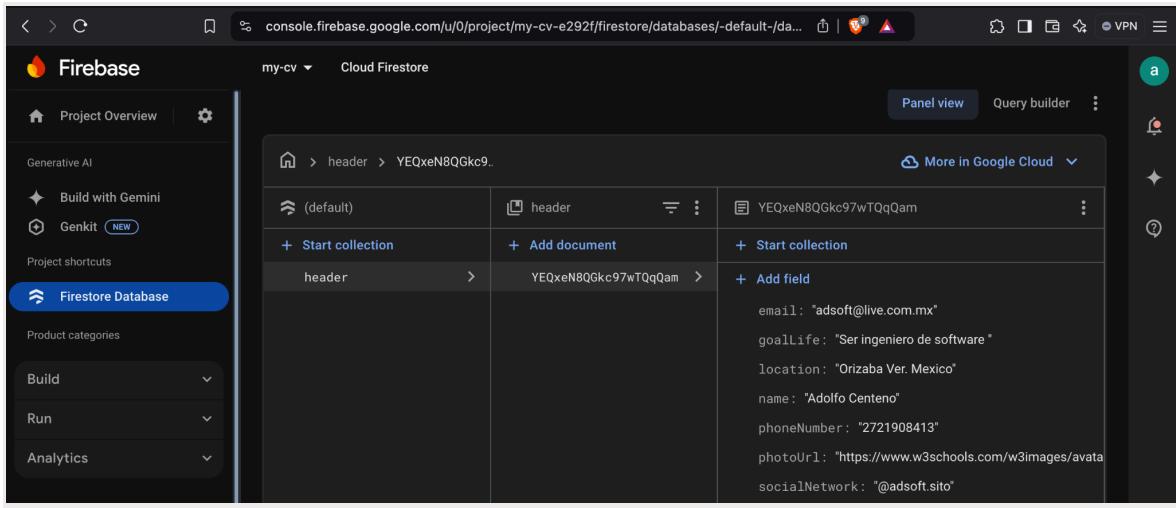
Crearemos un documento con la estructura de names, dar clic en generar ID automático, ingresar los 3 campos de la estructura names:

1. **name** tipo *string*
2. **goalLife** tipo *string*
3. **photoUrl** tipo *string*
4. **email** tipo *string*
5. **phoneNumber** tipo *string*
6. **location** tipo *string*
7. **socialNetwork** tipo *string*

Dar click en next, generar un ID automáticamente, luego ingresar cada atributo de la colección.

Field	Type	Value
name	string	Adolfo Centeno
goalLife	string	Ser ingeniero de
photoUrl	string	https://www.w3s
email	string	adsoft@live.com
phoneNumber	string	2721908413
location	string	Orizaba Ver. Mex
socialNetwork	string	@adsoft.sito

Nuestra colección **header** con el nuevo documento luce de la siguiente manera:



The screenshot shows the Firebase Firestore interface. On the left, there's a sidebar with 'Project Overview', 'Generative AI' (with 'Build with Gemini' and 'Genkit (NEW)'), 'Project shortcuts', and 'Firestore Database' (which is selected and highlighted in blue). The main area shows a document structure under 'header'. The path is 'header > YEQxeN8QGkc9..'. The document itself has fields like 'email', 'goallife', 'location', 'name', 'phoneNumber', 'photoUrl', and 'socialNetwork'. A tooltip 'More in Google Cloud' is visible above the document details.

(default)	header	...	YEQxeN8QGkc97wTQqQam	...
+ Start collection	+ Add document		+ Start collection	
header	YEQxeN8QGkc97wTQqQam	>	+ Add field	
			email: "adsoft@live.com.mx"	
			goallife: "Ser ingeniero de software"	
			location: "Orizaba Ver. Mexico"	
			name: "Adolfo Centeno"	
			phoneNumber: "2721908413"	
			photoUrl: "https://www.w3schools.com/w3images/avatar2.jpg"	
			socialNetwork: "@adsoft.sito"	

NOTA: Solo necesitamos un registro de la colección Header, porque de acuerdo al requerimiento solo se necesita un registro para llenar el componente Header.

Repetimos el proceso para la colección work-experience.

Para recordar el componente work-experience, debe guardar la siguiente información:

Work Experience

06/2006 - Present

Chairman
 SolarCity

San Mateo, USA

Accomplishments

Created a collaboration between SolarCity and Tesla to use electric vehicle batteries to smooth the impact of rooftop solar on the power grid.

Provided the initial concept and financial capital.

02/2004 - Present

CEO and Product Architect
 Tesla Motors

Palo Alto, USA

Accomplishments

Currently oversee the company's product strategy -- including the design, engineering and manufacturing of more and more affordable electric vehicles for mainstream consumers.

Insisted on using carbon fiber composite materials in the hull to minimize weight, developed the battery module and even some elements of design, like the headlights.

Received Global Green 2006 product design award for Tesla Roadster design.

06/2002 - Present

CEO and CTO
 SpaceX

Hawthorne, USA

Accomplishments

Plans to reduce space transportation costs to enable people to colonize Mars.

Oversee the development of rockets and spacecraft for missions to Earth orbit and ultimately to other planets.

Developed the Falcon 9 spacecraft which replaced the space shuttle when it retired in 2011.

03/1999 - 10/2002

CEO
 X.com and PayPal

San Jose, USA

Accomplishments

Involved in the development of new business models, conducted a successful viral marketing campaign, which led to a rapid increase in the number of customers.

Created a method of securely transferring money using a recipient's e-mail address.

01/1995 - 02/1999

Co-founder
 Zip2

California, USA

Accomplishments

Created a platform where newspapers – including credible ones as New York Times – could offer their customers some additional commercial services.

Crearemos un documento con la estructura de work-experience, dar clic en generar ID automático, ingresar los campos necesarios:

8. **startDate** tipo *string*
9. **endDate** tipo *string*
10. **location** tipo *string*
11. **position** tipo *string*
12. **company** tipo *string*
13. **accomplishments** tipo *string*

Plantilla Nivel de Conocimiento

Start a collection

Give the collection an ID

Add its first document

Parent path

/

Collection ID

work-experience

Cancel Next

Document parent path

/work-experience

Document ID

nxv8CStESIAm7jaBr4ra

Field	Type	Value
startDate	string	ene-2020
endDate	string	oct-2023
location	string	Orizaba, Mexico
position	string	Frontend Developer
company	string	Waves Lab
accomplishment	string	web platform, a

Add field

Cancel Save

Ahora agregamos un documento nuevo de tipo work-experience.

NOTA: esta colección representa los *n* puestos de trabajo que una persona ha tenido pueden ser más de 1 documento.

Field	Type	Value
startDate	string	nov-2023
endDate	string	nov-2024
location	string	Puebla, Mexico
position	string	Backend Developer
company	string	kubeet SA de CV
accomplishment	string	graphql api, REST

Contamos por ahora con 2 documentos tipo work-experience

Document ID	Fields
AW3bCFWx7A8SIDGxKGao	accomplishment: "graphql api, REST", company: "kubeet SA de CV", endDate: "nov-2024", location: "Puebla, Mexico", position: "Backend Developer", startDate: "nov-2023"
nxv8CStESIAm7jaBr4ra	

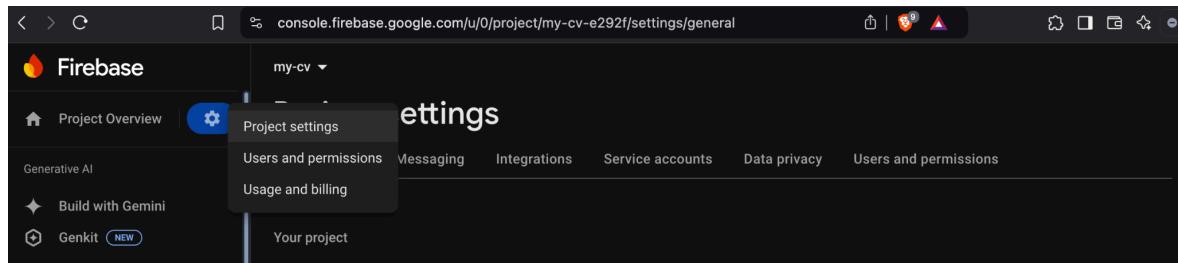
Como reto, crear las colecciones de firestore para las secciones:

1. education
2. skills

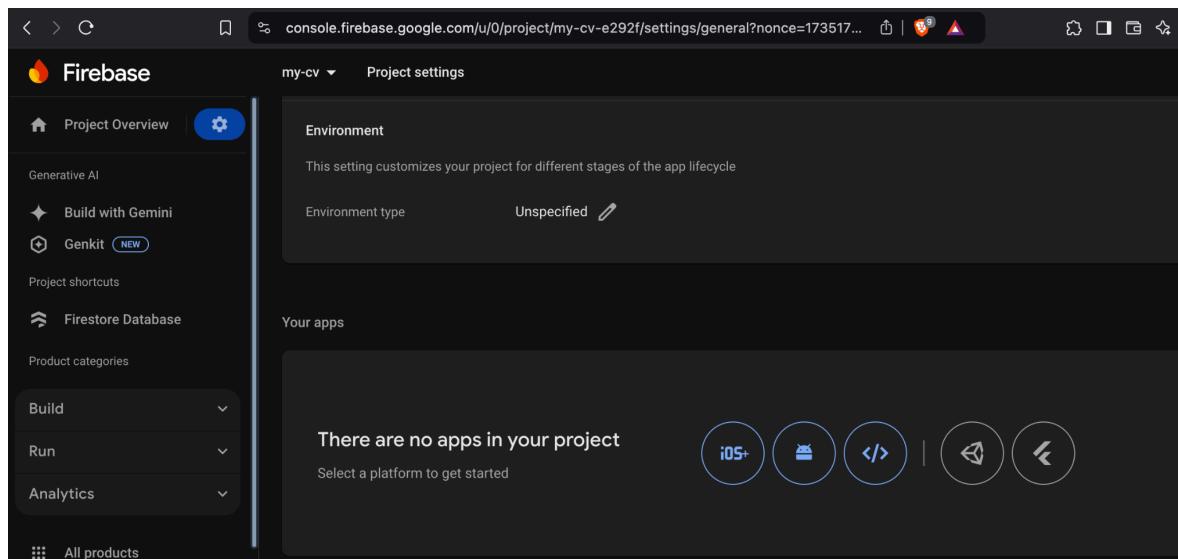
3. certificates
4. languages
5. interests

Conecitar Angular con Firebase.

Es necesario, descargar las llaves de acceso a **firebase**, Click en Project Overview -> Project Settings.



Posteriormente crear un Aplicación tipo Web, Click en el icono con el símbolo </>



Ahora ingresa un nombre para tu app, teclea: **my-cv**

The screenshot shows the Firebase console interface. At the top, it says "Add Firebase to your web app". Below that, there's a list of steps:

- 1 Register app (completed)
- 2 Add Firebase SDK (selected)

Under "Add Firebase SDK", there are two options:

- Use npm
- Use a <script> tag

A note below says: "If you're already using npm and a module bundler such as webpack or Rollup, you can run the following command to install the latest SDK (Learn more):"

```
$ npm install firebase
```

Then, initialize Firebase and begin using the SDKs for the products you'd like to use.

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AIzaSyDVb_5lgLQJP_30MAGJ-FL4BBCSqkLV50c",
  authDomain: "my-cv-e292f.firebaseio.com",
  projectId: "my-cv-e292f",
  storageBucket: "my-cv-e292f.firebaseiostorage.app",
  messagingSenderId: "18704728562",
  appId: "1:18704728562:web:a61c4eba2ffc95a2deeb6b"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
```

Note: This option uses the modular JavaScript SDK, which provides reduced SDK size.

Learn more about Firebase for web: [Get Started](#), [Web SDK API Reference](#), [Samples](#)

[Continue to console](#)

El asistente nos genera keys para accesar a Firebase desde Angular, Copiar el Script de código como sigue, ya que las ocuparemos posteriormente.

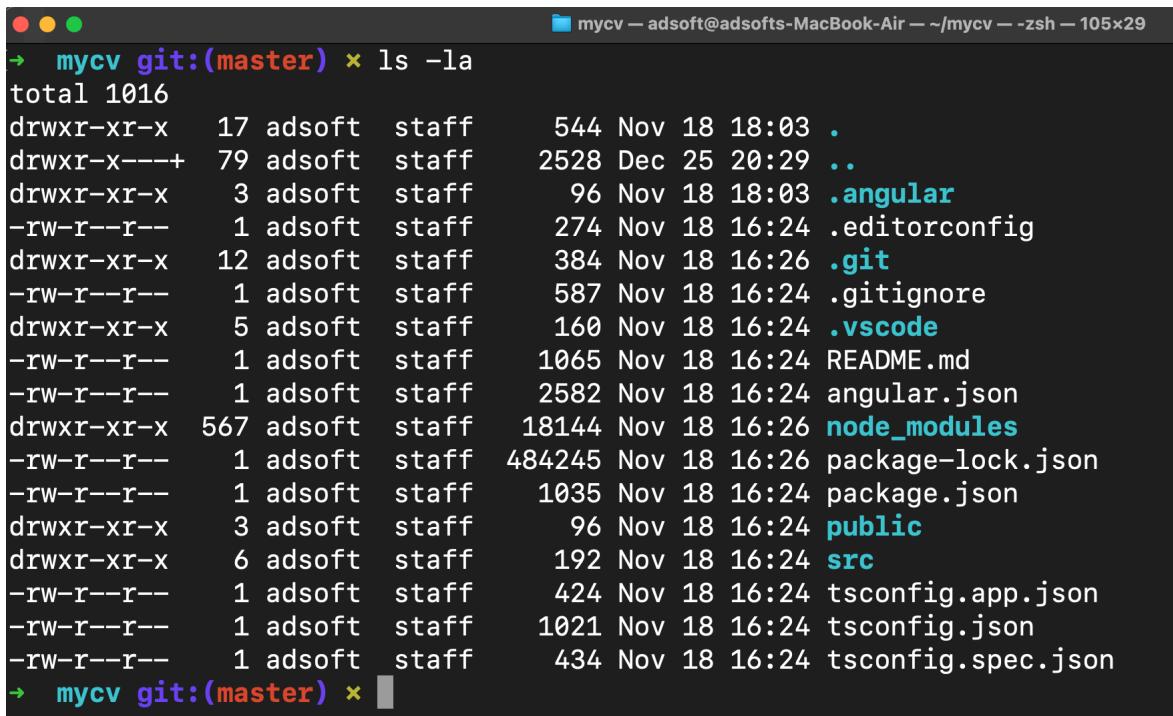
```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AIzaSyDVb_5lgLQJP_30MAGJ-FL4BBCSqkLV50c",
  authDomain: "my-cv-e292f.firebaseio.com",
  projectId: "my-cv-e292f",
  storageBucket: "my-cv-e292f.firebaseiostorage.app",
  messagingSenderId: "18704728562",
  appId: "1:18704728562:web:a61c4eba2ffc95a2deeb6b"
```

```
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
```

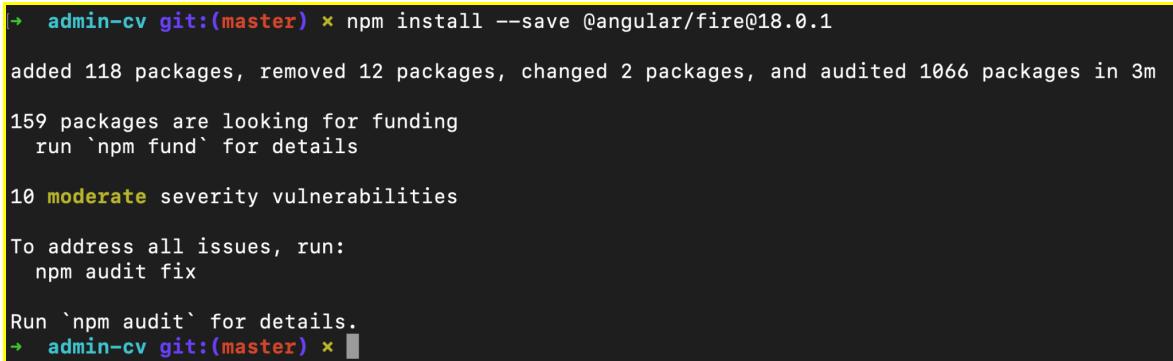
Posteriormente, regresamos al proyecto Angular, y nos colocamos en el root del proyecto.



```
mycv git:(master) ✘ ls -la
total 1016
drwxr-xr-x  17 adsoft  staff   544 Nov 18 18:03 .
drwxr-x---+ 79 adsoft  staff  2528 Dec 25 20:29 ..
drwxr-xr-x  3 adsoft  staff   96 Nov 18 18:03 .angular
-rw-r--r--  1 adsoft  staff  274 Nov 18 16:24 .editorconfig
drwxr-xr-x  12 adsoft  staff  384 Nov 18 16:26 .git
-rw-r--r--  1 adsoft  staff  587 Nov 18 16:24 .gitignore
drwxr-xr-x  5 adsoft  staff  160 Nov 18 16:24 .vscode
-rw-r--r--  1 adsoft  staff  1065 Nov 18 16:24 README.md
-rw-r--r--  1 adsoft  staff  2582 Nov 18 16:24 angular.json
drwxr-xr-x  567 adsoft  staff  18144 Nov 18 16:26 node_modules
-rw-r--r--  1 adsoft  staff  484245 Nov 18 16:26 package-lock.json
-rw-r--r--  1 adsoft  staff  1035 Nov 18 16:24 package.json
drwxr-xr-x  3 adsoft  staff   96 Nov 18 16:24 public
drwxr-xr-x  6 adsoft  staff  192 Nov 18 16:24 src
-rw-r--r--  1 adsoft  staff  424 Nov 18 16:24 tsconfig.app.json
-rw-r--r--  1 adsoft  staff  1021 Nov 18 16:24 tsconfig.json
-rw-r--r--  1 adsoft  staff  434 Nov 18 16:24 tsconfig.spec.json
→ mycv git:(master) ✘
```

Instalamos la biblioteca de firebase en nuestro proyecto.

```
$ npm install --save @angular/fire@18.0.1
```



```
admin-cv git:(master) ✘ npm install --save @angular/fire@18.0.1
added 118 packages, removed 12 packages, changed 2 packages, and audited 1066 packages in 3m
159 packages are looking for funding
  run `npm fund` for details

10 moderate severity vulnerabilities

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
→ admin-cv git:(master) ✘
```

Ahora creamos los ambientes de desarrollo (develop y production) de nuestro proyecto donde colocaremos las keys de firebase.

\$ ng generate environments

```
→ mycv git:(master) ✘ ng generate environments
CREATE src/environments/environment.ts (31 bytes)
CREATE src/environments/environment.development.ts (31 bytes)
UPDATE angular.json (2807 bytes)
→ mycv git:(master) ✘
```

Verificamos

\$ ls -la src/environments

```
→ mycv git:(master) ✘ ls -la src/environments

total 16
drwxr-xr-x  4 adsoft  staff  128 Dec 25 20:42 .
drwxr-xr-x  7 adsoft  staff  224 Dec 25 20:42 ..
-rw-r--r--  1 adsoft  staff   31 Dec 25 20:42 environment.development.ts
-rw-r--r--  1 adsoft  staff   31 Dec 25 20:42 environment.ts
→ mycv git:(master) ✘
```

Modificamos environment.ts con las llaves de tu proyecto firebase de la siguiente manera:

\$ nano src/environments/environment.ts

```
1 export const environment = {
2   production: false,
3   firebase: {
4     apiKey: "AIzaSyDVb_5lgLQJP_30MAGJ-FL4BBCSqkLV50c",
5     authDomain: "my-cv-e292f.firebaseio.com",
6     projectId: "my-cv-e292f",
7     storageBucket: "my-cv-e292f.firebaseio.storage.app",
8     messagingSenderId: "18704728562",
9     appId: "1:18704728562:web:a61c4eba2ffc95a2deeb6b"
10  }
11 };
```

\$ nano src/environments/environment.development.ts

```

1 export const environment = {
2   production: false,
3   firebase: {
4     apiKey: "AIzaSyDVb_5lgLQJP_30MAGJ-FL4BBCSvkLV50c",
5     authDomain: "my-cv-e292f.firebaseio.com",
6     projectId: "my-cv-e292f",
7     storageBucket: "my-cv-e292f.firebaseiostorage.app",
8     messagingSenderId: "18704728562",
9     appId: "1:18704728562:web:a61c4eba2ffc95a2deeb6b"
10  }
11 };
```

Después, debemos configurar el archivo **src/app/app.module.ts** para soportar firebase en nuestro proyecto

\$ nano src/app/app.module.ts

NOTA: se agrega línea 14, 15 para importar la biblioteca de firebase y el archivo del ambiente de desarrollo con las keys. En la línea 31 se inicializa el módulo firebase con las keys especificadas.

```

10 import { CertificatesComponent } from './certificates/certificates.component';
11 import { LanguagesComponent } from './languages/languages.component';
12 import { InterestsComponent } from './interests/interests.component';
13
14 import { AngularFireModule } from '@angular/fire/compat';
15 import { environment } from '../environments/environment';
16
17 @NgModule({
18   declarations: [
19     AppComponent,
20     HeaderComponent,
21     WorkExperienceComponent,
22     EducationComponent,
23     SkillsComponent,
24     CertificatesComponent,
25     LanguagesComponent,
26     InterestsComponent
27   ],
28   imports: [
29     BrowserModule,
30     AppRoutingModule,
31     AngularFireModule.initializeApp(environment.firebaseioConfig),
32   ],
33   providers: [],
34   bootstrap: [AppComponent]
35 })
36 export class AppModule {}
```

Consumo del servicio header-service

Para consumir de forma adecuada las colecciones de Firebase debemos crear clases llamadas modelos, que contengan la misma estructura que las colecciones de Firebase, así podemos crear variables o arrays en Angular del modelo apropiado para almacenar las colecciones.

Creamos una carpeta models en src/app

```
$ mkdir src/app/models
```

```
[→ mycv git:(master) ✘ mkdir src/app/models
[→ mycv git:(master) ✘ ls -la src/app
total 40
drwxr-xr-x 17 adsoft staff 544 Dec 25 21:12 .
drwxr-xr-x  7 adsoft staff 224 Dec 25 20:42 ..
-rw-r--r--  1 adsoft staff  0 Nov 18 16:24 app.component.css
-rw-r--r--  1 adsoft staff 420 Nov 19 16:54 app.component.html
-rw-r--r--  1 adsoft staff 910 Nov 18 16:24 app.component.spec.ts
-rw-r--r--  1 adsoft staff 957 Nov 19 14:08 app.component.ts
-rw-r--r--  1 adsoft staff 310 Nov 18 16:24 app.config.ts
-rw-r--r--  1 adsoft staff  77 Nov 18 16:24 app.routes.ts
drwxr-xr-x  6 adsoft staff 192 Nov 18 23:39 certificates
drwxr-xr-x  6 adsoft staff 192 Nov 18 23:34 education
drwxr-xr-x  6 adsoft staff 192 Dec 20 13:40 header
drwxr-xr-x  6 adsoft staff 192 Nov 18 23:42 interests
drwxr-xr-x  6 adsoft staff 192 Nov 18 23:41 languages
drwxr-xr-x  2 adsoft staff  64 Dec 25 21:12 models
drwxr-xr-x  9 adsoft staff 288 Dec 19 14:25 services
drwxr-xr-x  6 adsoft staff 192 Nov 18 23:36 skills
drwxr-xr-x  6 adsoft staff 192 Dec 20 15:26 work-experience
```

Creamos una carpeta header dentro de models

```
$ mkdir src/app/models/header
```

```
[→ mycv git:(master) ✘ mkdir src/app/models/header
[→ mycv git:(master) ✘ ls -la src/app/models/header
total 0
drwxr-xr-x  2 adsoft staff  64 Dec 25 21:14 .
drwxr-xr-x  3 adsoft staff  96 Dec 25 21:14 ..
[→ mycv git:(master) ✘ ]
```

Creamos el modelo para Header: **header.model.ts**

```
$ nano src/app/models/header/header.model.ts
```

```

1 export class Header {
2   id?: string;
3   name?: string = 'name';
4   goalLife?: string = 'goal';
5   photoUrl?: string = 'photo';
6   email?: string = 'email@domain.com';
7   phoneNumber?: string = '999-999-9999';
8   location?: string = 'city, country';
9   socialNetwork?: string = '@facebook';
10 }

```

Ahora modificamos el service **header.service.ts**.

Importamos los módulos de firebase (línea 2), importamos el modelo header.model (línea 3).

```

2 import { AngularFirestore, AngularFirestoreCollection} from '@angular/fire/compat/firestore';
3 import { Header } from '../../../../../models/header/header.model';

```

Definimos el nombre de la colección a leer (línea 11), declaramos una variable para almacenar la colección Header (línea 13).

```

11   private dbPath = '/header';
12
13   headerRef: AngularFirestoreCollection<Header>;

```

Creamos un constructor (líneas 15 a 17), con un parámetro que inyecte una instancia de Firestore (línea 15), leemos la colección /header y la almacenamos en **this.headerRef** (línea 16).

```

15   constructor(private db: AngularFirestore) {
16     this.headerRef = db.collection(this.dbPath);
17   }

```

Creamos un método para retornar la colección recuperada de Firestore y almacenada en **this.headerRef** (líneas 19 y 21)

```

19     getHeader(): AngularFirestoreCollection<Header> {
20         return this.headerRef;
21     }

```

Podemos analizar **header.service.ts** completo con la siguiente instrucción.

```
$ nano src/app/services/header-service/header.service.ts
```

```

1 import { Injectable } from '@angular/core';
2 import { AngularFirestore, AngularFirestoreCollection} from '@angular/fire/compat.firebaseio';
3 import { Header } from '../../../../../models/header/header.model';
4
5 @Injectable({
6     providedIn: 'root'
7 })
8 export class HeaderService {
9     accesoHeader = 'header service running...';
10
11     private dbPath = '/header';
12
13     headerRef: AngularFirestoreCollection<Header>;
14
15     constructor(private db: AngularFirestore) {
16         this.headerRef = db.collection(this.dbPath);
17     }
18
19     getHeader(): AngularFirestoreCollection<Header> {
20         return this.headerRef;
21     }
22 }

```

Ahora debemos modificar el componente Header, para consumir los datos de firebase que lee el service.

Empezamos con **header.component.ts**

Importamos el modelo header.model (línea 3) y el operador map necesario para la lectura de las colecciones (línea 4)

```

3 import { Header } from '../../../../../models/header/header.model';
4 import { map } from 'rxjs/operators';

```

Creamos una instancia del Modelo Header llamada **header**, que almacenará el resultado de la lectura de Firestore (línea 13).

```

13     header: Header = new Header();

```

Actualizamos el constructor para llamar al método **getHeader** del service (línea 17 a 26), en la línea 24 se almacena en la variable **header** el resultado de la lectura de la colección Header de Firestore.

NOTA: Como mencionamos anteriormente la colección Header solo guarda un documento, por tanto la variable header solo debe almacenar el primer documento: **this.header = data[0];**

```

17      this.headerService.getHeader().snapshotChanges().pipe(
18        map(changes =>
19          changes.map(c =>
20            ({ id: c.payload.doc.id, ...c.payload.doc.data() })
21          )
22        )
23      ).subscribe(data => {
24        this.header = data[0];
25        console.log(this.header);
26      });

```

Podemos visualizar el código completo de header.component.ts con la siguiente instrucción:

\$ nano src/app/header/header.component.ts

```

1 import { Component } from '@angular/core';
2 import { HeaderService } from '../services/header-service/header.service';
3 import { Header } from '../models/header/header.model';
4 import { map } from 'rxjs/operators';
5
6 @Component({
7   selector: 'app-header',
8   templateUrl: './header.component.html',
9   styleUrls: ['./header.component.css']
10 })
11 export class HeaderComponent {
12
13   header: Header = new Header();
14   constructor(public headerService: HeaderService)
15   {
16     console.log(this.headerService);
17     this.headerService.getHeader().snapshotChanges().pipe(
18       map(changes =>
19         changes.map(c =>
20           ({ id: c.payload.doc.id, ...c.payload.doc.data() })
21         )
22       )
23     ).subscribe(data => {
24       this.header = data[0];
25       console.log(this.header);
26     });
27   }
28 }

```

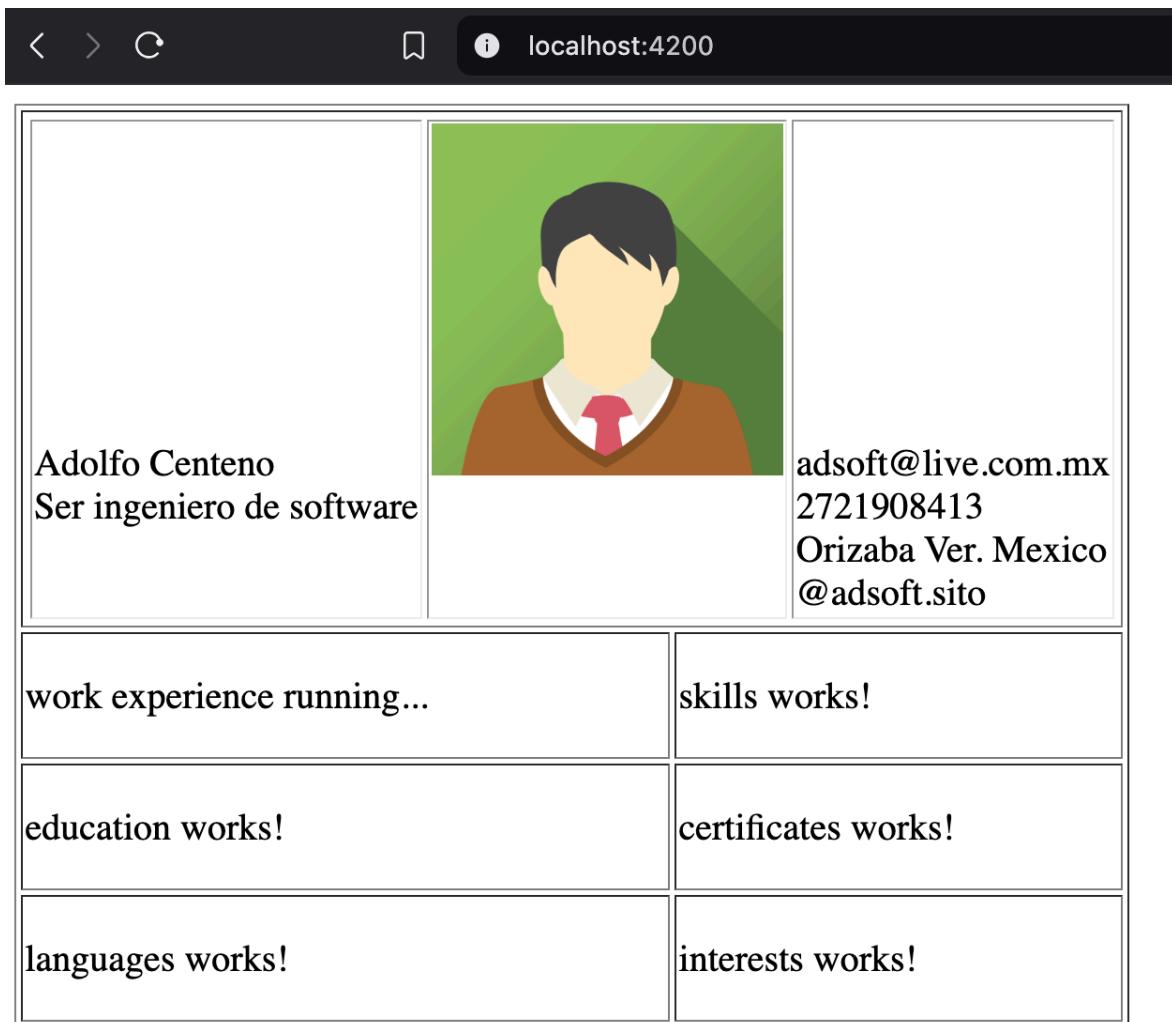
Continuamos con header.component.html, lo modificaremos para visualizar los cada uno de los atributos del documento de Header.

```
$ nano src/app/header/header.component.html
```

```
1 <tr>
2   <td>
3     {{header.name}} <br>
4     {{header.goalLife}}
5   </td>
6   <td>
7     
8   </td>
9   <td>
10    {{header.email}}<br>
11    {{header.phoneNumber}}<br>
12    {{header.location}}<br>
13    {{header.socialNetwork}}
14  </td>
15 </tr>
```

Podemos probar que el componente Header ya esté lleno con los atributos de la colección:

```
$ ng serve
```



The screenshot shows a resume template with the following details:

- Name:** Adolfo Centeno
- Role:** Ser ingeniero de software
- Contact Information:**
 - adsoft@live.com.mx
 - 2721908413
 - Orizaba Ver. Mexico
 - @adsoft.sito
- Work Experience:** work experience running...
- Skills:** skills works!
- Education:** education works!
- Certificates:** certificates works!
- Languages:** languages works!
- Interests:** interests works!

Consumo del servicio work-experience-service

Para consumir la colección work-experience de Firestore, creamos una carpeta header dentro de models

```
$ mkdir src/app/models/work-experience
```

```
→ mycv git:(master) ✘ mkdir src/app/models/work-experience
```

Ahora, creamos el modelo para work-experience (con la misma estructura que la colección en Firestore): **work-experience.model.ts**

```
$ nano src/app/models/header/work-experience.model.ts
```

```

1 export class WorkExperience {
2   id?: string;
3   startDate?: string = 'mmm-yyyy';
4   endDate?: string = 'mmm-yyyy';
5   location?: string = 'city, country';
6   position?: string = 'my position';
7   company?: string = 'my company';
8   accomplishments?: string = 'item1, item2, item n';
9 }
10

```

Ahora modificamos el service de work-experience, con las mismas instrucciones que Header, solo cambiando el **modelo** y el nombre de la **colección** en Firestore.

Podemos visualizar el servicio completo a continuación.

\$ nano src/app/services/work-experience-service/work-experience.service.ts

```

1 import { Injectable } from '@angular/core';
2 import { AngularFirestore, AngularFirestoreCollection } from '@angular/fire/compat/firestore';
3 import { WorkExperience } from '../../../../../models/work-experience/work-experience.model';
4
5
6 @Injectable({
7   providedIn: 'root'
8 })
9 export class WorkExperienceService {
10
11   accesoWorkExperience = "work experience running...";
12   private dbPath = '/work-experience';
13
14   workExperienceRef: AngularFirestoreCollection<WorkExperience>;
15
16   constructor(private db: AngularFirestore) {
17     this.workExperienceRef = db.collection(this.dbPath);
18   }
19
20   getWorkExperience(): AngularFirestoreCollection<WorkExperience> {
21     return this.workExperienceRef;
22   }
23 }
24
25
26

```

Ahora debemos modificar el componente work-experience, para consumir los atributos de la colección de Firestore con los elementos que representan la experiencia laboral de una persona.

Empezamos con work-experience.component.ts

En este caso se define un array de objetos de tipo WorkExperience en la línea 13.

```
13     workExperience: WorkExperience[] = [];
```

Cuando se recupera la colección, esta contiene n elementos que se asignan por completo al array workExperiece, en la línea 25

```
24         ).subscribe(data => {
25             this.workExperience = data;
26             console.log(this.workExperience);
```

Podemos consultar el código completo a continuación:

```
$ nano src/app/work-experience/work-experience.component.ts
```

```
1 import { Component } from '@angular/core';
2 import { WorkExperienceService } from './services/work-experience-service/work-experience.service';
3 import { WorkExperience } from './models/work-experience/work-experience.model';
4 import { map } from 'rxjs/operators';
5
6 @Component({
7   selector: 'app-work-experience',
8   templateUrl: './work-experience.component.html',
9   styleUrls: ['./work-experience.component.css']
10 })
11 export class WorkExperienceComponent {
12
13   workExperience: WorkExperience[] = [];
14
15   constructor(public workExperienceService: WorkExperienceService)
16   {
17     console.log(this.workExperienceService);
18     this.workExperienceService.getWorkExperience().snapshotChanges().pipe(
19       map(changes =>
20         changes.map(c =>
21           ({ id: c.payload.doc.id, ...c.payload.doc.data() })
22         )
23       )
24     ).subscribe(data => {
25       this.workExperience = data;
26       console.log(this.workExperience);
27     });
28   }
29 }
30
```

Continuamos con header.component.html para visualizar los atributos de los n documentos de la colección (en este caso 2).

NOTA: en esta caso utilizaremos la directiva ***ngFor** para recorrer cada elemento de la colección e imprimir cada empleo en una lista no ordenada (..)

```
$ nano src/app/header/header.component.html
```

```
1 <ul *ngFor="let job of workExperience;">
2   <li>
3     {{ job.startDate }} - {{ job.endDate }}    <i>{{job.location}} </i> <br>
4     <b> {{ job.position }} </b> <br>
5     {{ job.company }} <br>
6     Logros: <br>
7     {{ job.accomplishments }} <br>
8   </li>
9 </ul>
```

Podemos comprobar que se visualicen correctamente los atributos de la colección work-experience con: \$ ng serve

\$ ng serve

Adolfo Centeno Ser ingeniero de software		adsoft@live.com.mx 2721908413 Orizaba Ver. Mexico @adsoft.sito
<ul style="list-style-type: none"> nov-2023 - nov-2024 <i>Puebla, Mexico</i> Backend Developer kubeet SA de CV Logros: graphql api, REST api ene-2020 - oct-2023 <i>Orizaba, Mexico</i> Frontend Developer Waves Lab Logros: web platform, android app, ios app 	skills works!	
education works!	certificates works!	
languages works!	interests works!	

Hasta ahora ya hemos consumido 2 servicios y los hemos implementado en su respectivo componente. Podrías consumir ahora los servicios: education, languages, skills, certificates e interests ?

Sala de inspiración

Todas las plataformas...

Sala de pruebas

Lee cada una de las preguntas y selecciona la respuesta que consideres correcta

1.- *Con que instrucción instalamos las dependencias para soportar Firestore en nuestro proyecto Angular 18 ?*

Opción a: <code>npm --save @angular/fire</code>	Incorrecta: <i>La sintaxis correcta para instalar las dependencias de firebase son:</i> <code>npm install --save @angular/fire</code>
Opción b: <code>npm install --save @fire</code>	Incorrecta: <i>La sintaxis correcta para instalar las dependencias de firebase son:</i> <code>npm install --save @angular/fire</code>
Opción c: <code>npm install --save @angular/fire</code>	CORRECTA: <i>La sintaxis correcta para instalar las dependencias de firebase son:</i> <code>npm install --save @angular/fire</code>
Opción d: <code>npm install @angular.firebaseio</code>	Incorrecta: <i>La sintaxis correcta para instalar las dependencias de firebase son:</i>

	<i>npm install --save @angular/fire</i>
--	---

2.- *Instrucción de Angular CLI para crear los archivos de configuración para los ambientes de desarrollo.*

Opción a: <i>ng create environments</i>	Incorrecta: La sintaxis correcta de Angular CLI es: <i>ng generate environments</i>
Opción b: <i>ng g environment</i>	Incorrecta: La sintaxis correcta de Angular CLI es: <i>ng generate environments</i>
Opción c: <i>ng generate environment</i>	Incorrecta: La sintaxis correcta de Angular CLI es: <i>ng generate environments</i>
Opción d: <i>ng generate environments</i>	CORRECTA: La sintaxis correcta de Angular CLI es: <i>ng generate environments</i>

3.- Archivo de nuestro proyecto, donde configuramos nuestro proyecto para soportar firebase

Opción a: <i>src/app/app.module.ts</i>	CORRECTA El archivo correcto es:
--	--

	src/app/app.module.ts
Opción b: package.json	Incorrecta: El archivo correcto es: src/app/app.module.ts
Opción c: src/app/app-routing.module.ts	Incorrecta: El archivo correcto es: src/app/app.module.ts
Opción d: src/main.ts	Incorrecta: El archivo correcto es: src/app/app.module.ts

Contenido (NUTRE /SIGNIFICA)-

Hasta esta sección ya hemos estudiado ..

Pruébate (de los 3 temas)

Este es el momento ideal para realizar algunos ejercicios adicionales con angular y su enfoque basado en componentes..

Al finalizar el tema 2, dejamos este reto: Podrías consumir ahora los servicios: education, languages, skills, certificates e interests ?

Lo podrás lograr con la siguiente secuencia de pasos

- 1.- Crea las colecciones en Firestore de cada colección
- 2.- Ingresa al menos 2 documentos de cada colección de Firestore.
- 3.- Crea modelos con los mismos atributos de las colecciones Firestore.
- 4.- Actualiza cada servicio para leer su respectiva colección
- 5.- Actualiza cada componente para inyectar el servicio
- 6.- Modifica el archivo HTML de cada componente para visualizar la información
- 7.- Realiza pruebas periódicamente con :ng serve

Ideas para llevar

Actualmente la mayoría de las plataformas web consumen servicios de diferentes fuentes como: APIs de google maps, youtube, spotify, instagram, facebook, whatsapp, incluso del SAT para timbrar las facturas electrónicas.

- ¿Cómo integrarias a tu proyecto alguna de estas APIs ?

El siguiente paso es investigar cómo crear servicios en Angular para diferentes APIs en la nube, o bien para otras fuentes de datos como: SQL server, MySQL, MongoDB, PostgreSql entre muchas otras.

Referencias

- Gechev, Minko (2024-05-23). "[Meet Angular v19](#)". Medium. Retrieved 2024-06-02.
["https://angular.dev/guide/di"](https://angular.dev/guide/di). angular.dev. Retrieved 2024-12-20.
["https://angular.dev/guide/templates/binding"](https://angular.dev/guide/templates/binding). angular.dev. Retrieved 2024-12-21.
["https://angular.dev/guide/routing"](https://angular.dev/guide/routing). angular.dev. Retrieved 2024-12-27.

Material de consulta

- <https://v17.angular.io/guide/architecture-services>
<https://medium.com/@aqeelabbas3972/services-in-angular-b125a5b5690e>
<https://www.c-sharpcorner.com/article/create-services-in-angular-application/>