

Guía de ejercicios - Firebase (I)



¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

La siguiente guía de estudio tiene como objetivo practicar y ejercitar los contenidos que hemos visto en clase.

¡Vamos con todo!



Tabla de contenidos

Actividad guiada: Agregando y eliminando datos en Firebase.	2
Configuración del Front para extraer los datos de Firebase	7
Analicemos el código, pieza por pieza	11
¡Manos a la obra! - Practica lo aprendido	13



¡Comencemos!



Actividad guiada: Agregando y eliminando datos en Firebase.

A continuación, realizaremos un ejercicio con Vue JS y Firebase donde realizaremos la construcción de un CRUD. Esta aplicación está destinada a almacenar información de colaboradores en el ámbito de la programación y desarrollo web.

Recordemos que un CRUD consiste en un conjunto de operaciones que podemos realizar, entre las cuales están la creación, lectura, actualización y eliminación de datos. **De momento solo nos enfocaremos en las operaciones de leer, agregar y eliminar datos.**

Dicho esto, vayamos entonces con los pasos de implementación. Para ello, dividiremos el desarrollo en dos capas: los datos y el Front que traerá esos datos para mostrarlos.



Nota: Enfocaremos este desarrollo únicamente en las acciones de Create, Read, Delete.

- **Paso 1:** Iniciamos con la creación de un nuevo proyecto en Firebase. Para ello nos dirigimos a su consola y damos clic en “Agregar proyecto”.



Imagen 1. Agregar proyecto
Fuente: Firebase

- **Paso 2:** Seguimos los pasos de creación donde primero definimos el nombre del proyecto, lo llamaremos crud-colaboradores.

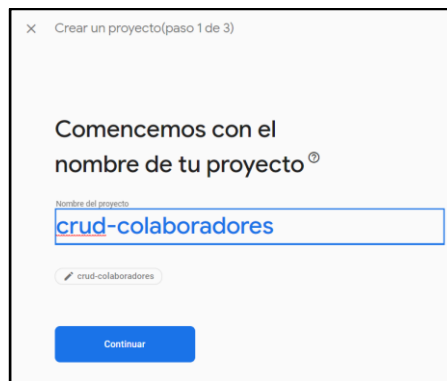


Imagen 2. Definiendo el nombre del proyecto
Fuente: Firebase

- **Paso 3:** Desactivamos la opción de Google Analytics y damos clic en “Crear proyecto”.

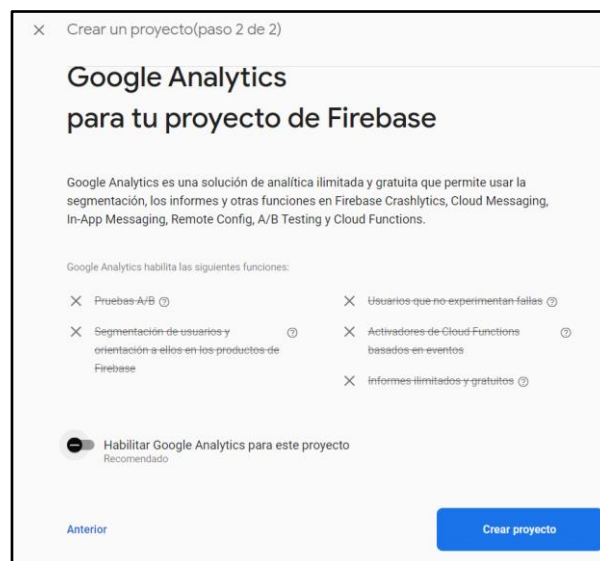


Imagen 3. Deshabilitar Google Analytics.
Fuente: Firebase

Esperamos unos segundos, recordemos que el proceso de creación depende también de nuestra velocidad de internet. Una vez creado, veremos el siguiente mensaje:

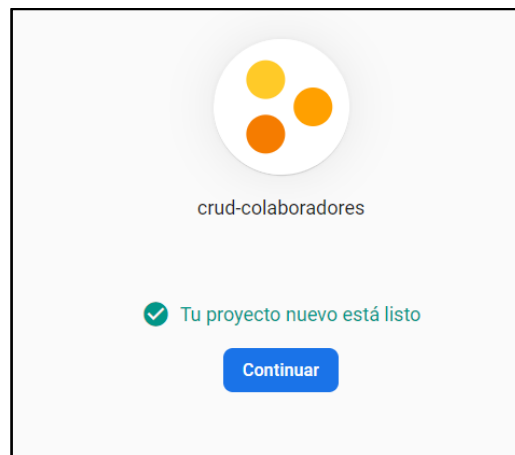


Imagen 4. Proyecto creado satisfactoriamente
Fuente: Firebase

- **Paso 4:** Ahora nos dirigimos a la sección de Compilación y seleccionamos "Firestore Database".

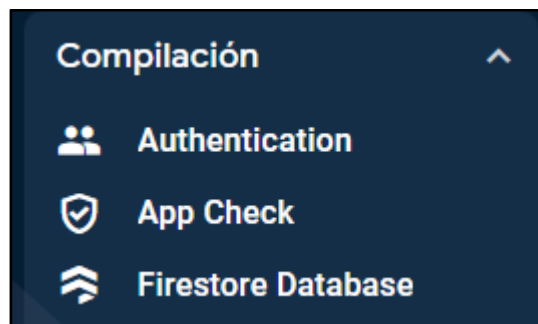


Imagen 5. Firestore Database
Fuente: Firebase

- **Paso 5:** Seguidamente, creamos nuestra base de datos haciendo clic en el botón "Crear base de datos".



Imagen 6. Crear una base de datos.
Fuente: Firebase

- **Paso 6:** Definimos las reglas de seguridad y seleccionamos el "Modo prueba".

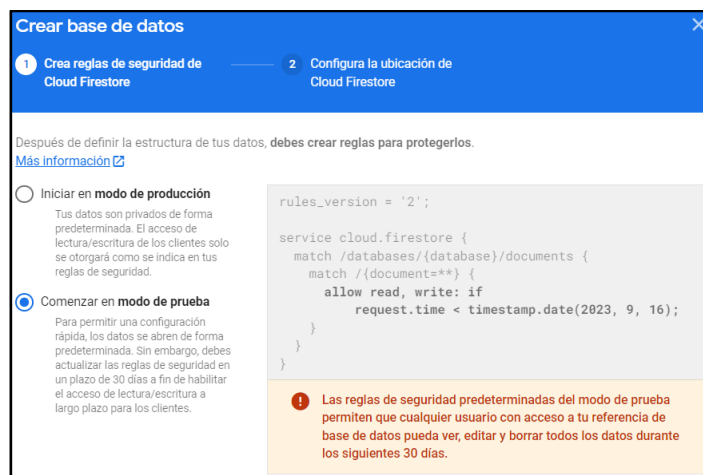


Imagen 7. Modo prueba
Fuente: Firebase



¡Importante! El modo prueba tiene algunas restricciones en cuanto al funcionamiento de las bases de datos en Firebase. Una de estas restricciones consiste en que las reglas de seguridad para cada proyecto en modo prueba, tendrá una duración de 30 días.

Estas reglas son modificables y se pueden cambiar a través de la siguiente [documentación](#). De momento, dejaremos el modo prueba para los efectos académicos de esta guía.

- **Paso 7:** Ahora, habilitamos la aplicación dando click en "Habilitar"

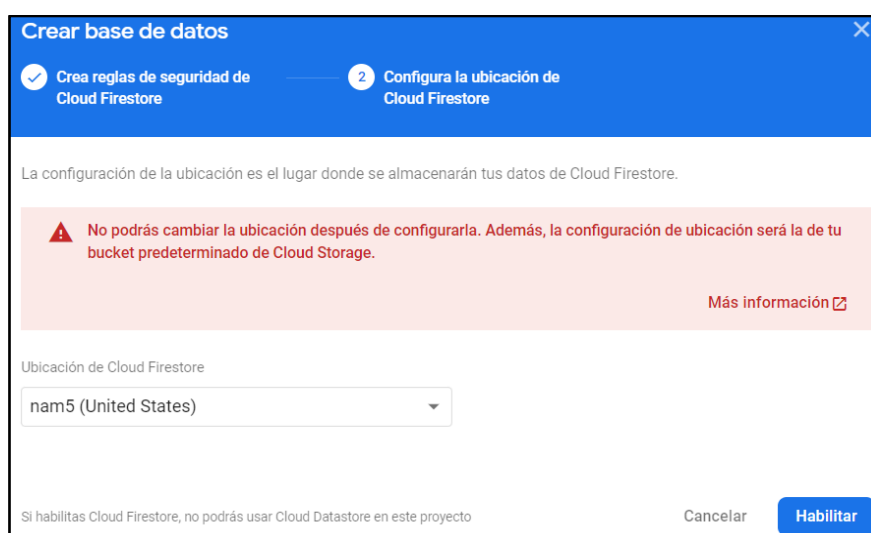


Imagen 8. Habilitar la base de datos
Fuente: Firebase

- **Paso 8:** Ya tenemos nuestra base de datos habilitada, por lo que ahora vamos a crear una nueva colección de datos. Esta colección servirá para definir los campos que tendrá nuestra base de datos de colaboradores. De tal manera, podríamos suponer que un colaborador tiene un campo definido como nombre.

Accedemos entonces al panel de Firestore Database y creemos esta colección.



Imagen 9 Panel de Firestore Database

Fuente: Firebase

- **8.1:** Definimos el ID de la colección y le asignamos el nombre de colaboradores.

The image shows a dialog box titled 'Inicia una colección'. It has two steps: '1. Asignar un ID a la colección' and '2. Agregar el primer documento'. The first step is active. It shows a 'Ruta superior' (Superior route) as '/'. Below it, there's a label 'ID de la colección' and a text input field containing 'colaboradores'. At the bottom right, there are 'Cancelar' and 'Siguiente' buttons.

Imagen 10. ID de la colección

Fuente: Firebase

- **8.2:** Definimos los campos de nuestra colección de colaboradores

The image shows a dialog box titled 'ID de documento'. It has a text input field containing 'TOJt3Y8sa2D6LoVuuvdR'. Below this, there's a table with three columns: 'Campo' (Field), 'Tipo' (Type), and 'Valor' (Value). The first row has 'nombre' in the 'Campo' column, '=' in the 'Tipo' column, and an empty text input field in the 'Valor' column. There's a minus sign button at the bottom right.

Imagen 11. Campo nombre de la colección de colaboradores

Fuente: Firebase



Nota: el primer campo "ID de documento" puede ser autogenerado dando click en "ID automático"

Imagen 12. ID automático en la colección de colaboradores.

Fuente: Firebase

Al realizar estos sub pasos para generar la colección de colaboradores, veremos en la consola de Firestore Database la siguiente información:

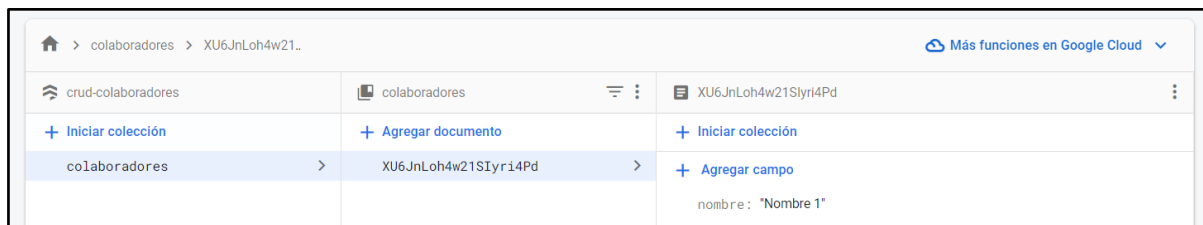


Imagen 13. Colección de colaboradores creada con sus campos.

Fuente: Firebase

Hasta este punto, ya hemos logrado configurar nuestra base de datos en Firebase. Ahora debemos enlazar este servicio con el Front, para ello sigamos los siguientes pasos con Vue JS.

Configuración del Front para extraer los datos de Firebase

- **Paso 9:** Creamos la aplicación con el CLI de Vue JS. En este caso, selecciona dentro de sus presets la presencia de Vuex y Babel. Una vez terminada la creación, procedemos a abrir con VS Code.
- **Paso 10:** En Firebase nos dirigimos al apartado de Descripción General y damos clic al botón Web `</>`.

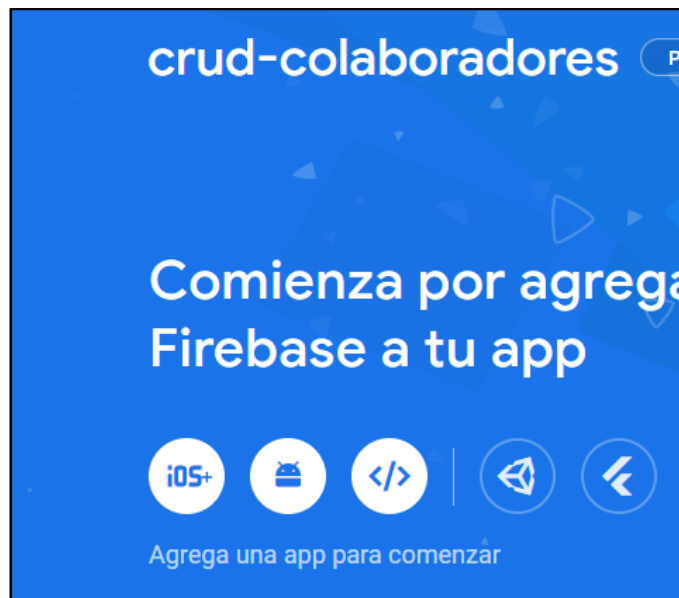


Imagen 14. Enlace para la Web
Fuente: Firebase

- **Paso 11:** Definimos el nombre de la aplicación web que hará uso de esta base de datos y damos click en el botón Registrar app.

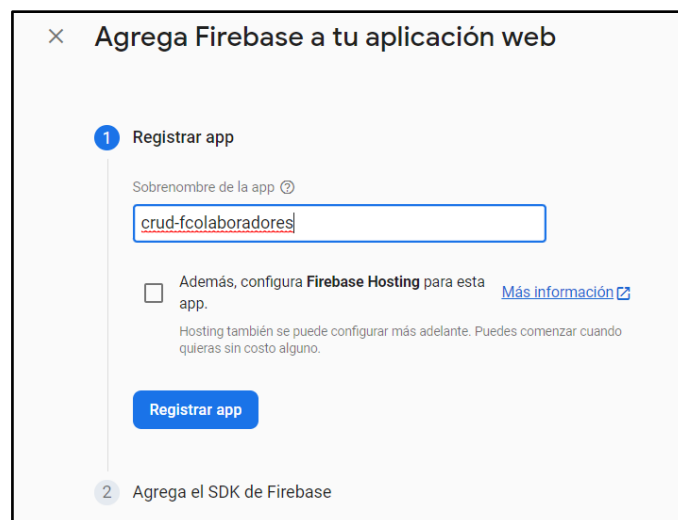


Imagen 15. Registro de la app para el enlace web
Fuente: Firebase

- **Paso 12:** Luego de esto, tendremos el SDK para configurar nuestra aplicación de Vue JS con Firebase.

2 Agrega el SDK de Firebase

☒ Usar npm ☐ Usar una etiqueta <script>

Si ya usas [npm](#) y un agrupador de módulos como [Webpack](#) o [Rollup](#), puedes ejecutar el siguiente comando para instalar la versión más reciente del SDK ([más información](#)):

```
$ npm install firebase
```

Luego, inicializa Firebase y comienza a usar los SDK de los productos que quieres utilizar.

```
// 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: "AIzaSyBl3nowIfNw4fF5idjpquYinLAGCRvn3zg",
  authDomain: "crud-colaboradores.firebaseio.com",
  projectId: "crud-colaboradores",
  storageBucket: "crud-colaboradores.appspot.com",
  messagingSenderId: "500651570035",
  appId: "1:500651570035:web:ecfcdce5161a8ede524b06"
};

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

Nota: Esta opción utiliza el [SDK de JavaScript modular](#), que proporciona un tamaño reducido del SDK.

Obtén más información sobre Firebase para la Web: [primeros pasos](#), [referencia de la API del SDK web](#) y [muestras](#)

Imagen 16. SDK generado
Fuente: Firebase



Nota: Recuerda que el SDK que se muestra en la imagen anterior, es propio y único durante la creación de cada proyecto. En este sentido, en la medida que vayas siguiendo los pasos, la información de tu SDK será distinta a la que muestra la imagen.

- **Paso 13:** Ejecutamos la instalación de Firebase en el proyecto de Vue JS, para ello utilizamos el comando `npm install firebase`.
- **Paso 14:** Nos dirigimos a VS Code y creamos un archivo llamado `firebaseconfig.js`, en el copiaremos el SDK que nos generó Firebase. Este archivo estará en la carpeta `/src`.
- **Paso 15:** Creamos un componente llamado `Colaboradores.vue`. En él definiremos el siguiente código para el template:

```
<template>
  <div>
    <form @submit.prevent="addColaborador">
      <input v-model="nuevoColaborador" placeholder="Nuevo colaborador" />
      <button type="submit">Agregar</button>
    </form>
    <ul>
      <li v-for="colaborador in colaboradores" :key="colaborador.id">
        {{ colaborador.id }} - {{ colaborador.nombre }}
        <button @click="deleteColaborador(colaborador.id)">Eliminar</button>
      </li>
    </ul>
  </div>
</template>
```

- **15.1:** Ahora, en el bloque del script, definimos la lógica para generar este CRUD, además de conectarnos a la base de datos y guardar en un objeto data() la información que será reactiva.

```
<script>
import { getFirestore, collection, onSnapshot, addDoc, doc, deleteDoc } from
'firebase/firestore';
import firebaseApp from '../firebaseconfig';

export default {
  data() {
    return {
      nuevoColaborador: '',
      colaboradores: []
    };
  },
  mounted() {
    // Cargar tareas existentes al montar el componente
    const db = getFirestore(firebaseApp);
    const colaboradoresRef = collection(db, 'colaboradores'); // "todos" es
    el nombre de la colección en Firestore
    onSnapshot(colaboradoresRef, (snapshot) => {
      this.colaboradores = snapshot.docs.map((doc) => ({ id: doc.id,
      ...doc.data() }));
    });
  },
  methods: {
    async addColaborador() {
      if (this.nuevoColaborador.trim() === '') return;
      const db = getFirestore(firebaseApp);
```

```
const colaboradoresRef = collection(db, 'colaboradores');
await addDoc(colaboradoresRef, { nombre: this.nuevoColaborador });

this.nuevoColaborador = ''; // Limpiar el campo después de agregar
},
async deleteColaborador(colaboradorId) {
  const db = getFirestore(firebaseApp);
  const colaboradorRef = doc(db, 'colaboradores', colaboradorId);
  await deleteDoc(colaboradorRef);
}
}
};
</script>
```

Analicemos el código, pieza por pieza

1. En primer lugar, tenemos la importación de un conjunto de funciones que dispone firebase para lograr implementar nuestro CRUD. Entre ellas tenemos:

```
import { getFirestore, collection, onSnapshot, addDoc, doc, deleteDoc } from
'firebase/firestore';
import firebaseApp from '../firebaseconfig';
```

- a. **getFirestore:** nos trae la instancia de Firebase inicializada en el archivo que creamos llamado firebaseconfig.js.
 - b. **collection:** Permite conectarnos a la colección de datos que tenemos almacenado en nuestro Firebase.
 - c. **onSnapshot:** Es un escuchador de eventos, en este caso la información será cargada cuando el evento del montaje del componente sea ejecutado.
 - d. **addDoc:** Es una función que nos permite agregar un documentos nuestra colección de referencia y además incorpora el campo ID de manera automática. Recordemos que a nuestra colección de colaboradores le definimos que el ID fuera generado automáticamente.
 - e. **doc:** Es una función que nos permite traer el conjunto de documentos o datos que se encuentran en nuestra colección de colaboradores.
 - f. **deleteDoc:** Es una función que permite borrar un documento de nuestra colección de datos, este toma como referencia el ID del elemento.
2. En segundo lugar, tenemos el objeto data con la información que será dinámica y reactiva en nuestro componente.

```
export default {  
  data() {  
    return {  
      nuevoColaborador: '',  
      colaboradores: []  
    };  
  },  
}
```

3. Luego, tenemos el mounted()

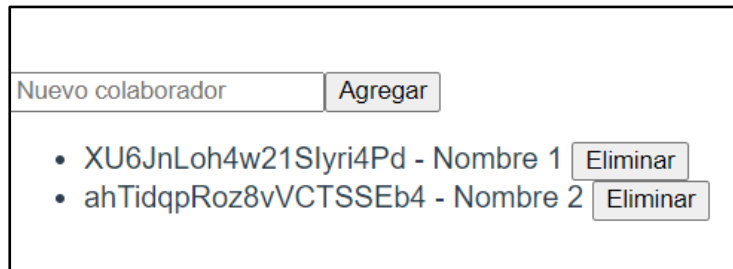
```
mounted() {  
  // Cargar tareas existentes al montar el componente  
  const db = getFirestore(firebaseApp);  
  const colaboradoresRef = collection(db, 'colaboradores'); // "todos" es  
  el nombre de la colección en Firestore  
  onSnapshot(colaboradoresRef, (snapshot) => {  
    this.colaboradores = snapshot.docs.map((doc) => ({ id: doc.id,  
    ...doc.data() }));  
  });  
},
```

En este, estamos definiendo que la información sea mostrada una vez que el componente esté cargado.

4. Por último, los métodos que estarán ejecutados desde el formulario de nuestro componente, en los cuales se realizan las operaciones de borrar y agregar un nuevo colaborador.

```
methods: {  
  async addColaborador() {  
    if (this.nuevoColaborador.trim() === '') return;  
    const db = getFirestore(firebaseApp);  
    const colaboradoresRef = collection(db, 'colaboradores');  
    await addDoc(colaboradoresRef, { nombre: this.nuevoColaborador });  
  
    this.nuevoColaborador = ''; // Esta instrucción hace un reset al input  
  },  
  async deleteColaborador(colaboradorId) {  
    const db = getFirestore(firebaseApp);  
    const colaboradorRef = doc(db, 'colaboradores', colaboradorId);  
    await deleteDoc(colaboradorRef);  
  }  
}
```

Hasta este punto, y al ejecutar la App, veremos el siguiente resultado en el navegador:



The screenshot shows a web application interface. At the top, there is a form with a text input labeled "Nuevo colaborador" and a button labeled "Agregar". Below the form, there is a list of two items, each consisting of a long alphanumeric string followed by "- Nombre 1" and "- Nombre 2" respectively. Each item has a button labeled "Eliminar" next to it.

Nuevo colaborador		Agregar
• XU6JnLoh4w21Slyri4Pd - Nombre 1	Eliminar	
• ahTidqpRoz8vVCTSSEb4 - Nombre 2	Eliminar	

Imagen 17. Resultado parcial de la aplicación
Fuente: Desafío Latam



¡Manos a la obra! - Practica lo aprendido

A continuación, deberás desarrollar una aplicación para una farmacia en el cual de momento solo se te solicita que se pueda almacenar el nombre del medicamento. Para lograrlo, implementa Firebase y Vue JS.