



**SANTO
TOMÁS**
INSTITUTO PROFESIONAL



**SANTO[®]
TOMÁS**

INSTITUTO PROFESIONAL

PROGRAMACIÓN ANDROID



CRUD FIRESTORE



Implementación de un CRUD en Android con Firebase



Conexión a Firebase

× Crear un proyecto

Comencemos con el nombre
de tu proyecto[®]

Ingresa el nombre de tu proyecto

my-awesome-project-id

¿Ya tienes un proyecto de Google Cloud?
[Agregar Firebase al proyecto de Google Cloud](#)

Continuar



Firebase gives you the tools and infrastructure from Google to help you develop, grow and earn money from your app. [Learn more](#)

> Analytics

Measure user activity and engagement with free, easy, and unlimited analytics. [More info](#)

> Authentication

Sign in and manage users with ease using popular login providers like Google, Facebook, and others. You can even use a custom authentication system. [More info](#)

✓ Realtime Database

Store and sync data with this cloud-hosted NoSQL database. Data is synced across all clients in realtime and remains available when your app goes offline. [More info](#)

▶ [Get started with Realtime Database](#)

▶ [Get started with Realtime Database \[Java\]](#)

← Firebase > Realtime Database

Get started with Realtime Database [Java]

The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client.

[Launch in browser](#)

1 Connect your app to Firebase

✓ Connected

2 Add the Realtime Database to your app

[Add the Realtime Database SDK to your app](#)

NOTE: After adding the SDK, here are some other helpful configurations to consider:

- Do you want an easier way to manage library versions?
You can use the [Firebase Android BOM](#) to manage your Firebase library versions and ensure that your app is always using compatible library versions.

To use the Realtime Database, you need to create a database instance in the [Firebase console](#).

3 Configure Realtime Database Rules

The Realtime Database provides a declarative rules language that allows you to define how your data should be structured, how it should be indexed, and when your data can be read from and written to.

By default, read and write access to your database is restricted so only authenticated users can read or write data. To get started without setting up authentication, you can [configure your rules for public access](#). This does make your database open to anyone, even people not using your app, so be sure to restrict your database again when you set up authentication.

4 Write to your database

Retrieve an instance of your database using `getReference()` and reference the location you want to write to.

To get a reference to a database other than a `co-central` default database, you must pass the database URL to `getReference()`. For a `co-central` default database, you can call `getReference()` without arguments. [Learn more about database locations.](#)

```
// Create a reference to the database
FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference rootRef = database.getReference();
// Write some data to the database
rootRef.child("name").setValue("John");
```

You can save a range of data types to the database this way, including Java objects. When you have an object the responses from any getters will be saved as children of this location.

5 Read from your database

To make your app data update in realtime, you should add a `ValueEventListener` to the reference you just created.

The `onDataChange()` method in this class is triggered once when the listener is attached and again every time the data changes, including the children.

```
// Read from the database
rootRef.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DatabaseSnapshot snapshot) {
        // This method is called once with the complete value and snapshot
        // Retrieve the data from the snapshot as a String
        String value = snapshot.child("name").getValue(String.class);
        Log.d(TAG, "Name is: " + value);
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {
        // Failed to read value
        Log.d(TAG, "Failed to read value.", error.toDatabaseError());
    }
});
```

6 Optional: Configure ProGuard

When using Firebase Realtime Database in your app along with ProGuard, you need to consider how your model objects will be serialized and deserialized after obfuscation. If you use `Parcelable` or `Serializable` interfaces, you must ensure that your data class implements these interfaces to be deserialized in your app. For more information, see [ProGuard](#).



Realtime Database

Almacena y sincroniza datos en tiempo
real



Organización del Proyecto

```
> activities
> models
> ui
```

- Estructura de carpetas:
 - `ui` : Contiene los fragmentos (`BuscarFragment` , `CrearFragment` , `ListarFragment`).
 - `models` : Contiene la clase `Contacto` .
 - `activities` : Contiene la `MainActivity` para la navegación.
- Clase `Contacto` :
 - Atributos: `id` , `nombre` , `edad` .
 - Métodos para obtener cada atributo (`getId()` , `getNombre()` , `getEdad()`).



Guardar Contacto en Firebase

```
databaseReference.child(id).setValue(contacto)
    .addOnSuccessListener(aVoid -> {
        Toast.makeText(getApplicationContext(), text: "Contacto guardado correctamente", Toast.LENGTH_SHORT).show();
        limpiarFormulario();
    })
    .addOnFailureListener(e -> {
        Toast.makeText(getApplicationContext(), text: "Error al guardar el contacto", Toast.LENGTH_SHORT).show();
    });
```



Listar Contactos desde Firebase

```
public class ListarFragment extends Fragment {

    private ListView listViewContactos; 2 usages
    private ArrayAdapter<String> adapter; 3 usages
    private ArrayList<String> listaContactos; 5 usages
    private DatabaseReference databaseReference; 2 usages

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container,
                             @Nullable Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_listar, container, attachToRoot: false);

        // Inicializar la vista y la lista
        listViewContactos = view.findViewById(R.id.list_view_contactos);
        listaContactos = new ArrayList<>();
        adapter = new ArrayAdapter<>(getContext(), android.R.layout.simple_list_item_1, listaContactos);
        listViewContactos.setAdapter(adapter);

        // Inicializar la referencia a la base de datos
        databaseReference = FirebaseDatabase.getInstance().getReference(path: "contacto");

        // Obtener los contactos desde Firebase
        obtenerContactos();

        return view;
    }

    private void obtenerContactos() { 1 usage
        databaseReference.addValueEventListener(new ValueEventListener() {
            @Override 2 usages
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                listaContactos.clear();
                for (DataSnapshot dataSnapshot : snapshot.getChildren()) {
                    Contacto contacto = dataSnapshot.getValue(Contacto.class);
                    if (contacto != null) {
                        // Agregar el contacto a la lista
                        listaContactos.add(contacto.getNombre() + " - Edad: " + contacto.getEdad());
                        Log.d(tag: "ListarFragment", msg: "Contacto encontrado: " + contacto.getNombre());
                    }
                }
                adapter.notifyDataSetChanged();
                Log.d(tag: "ListarFragment", msg: "Número de contactos: " + listaContactos.size());
            }
        });

        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Toast.makeText(getContext(), text: "Error al cargar los contactos: " + error.getMessage(), Toast.LENGTH_SHORT).show();
            Log.e(tag: "ListarFragment", msg: "Error al cargar los contactos", error.toException());
        }
    }
}
```



Buscar y Actualizar Contacto

```
private void buscarContacto() { 1 usage
    String nombre = etBuscarNombre.getText().toString();

    if (TextUtils.isEmpty(nombre)) {
        Toast.makeText(getContext(), text: "Por favor, ingrese un nombre", Toast.LENGTH_SHORT).show();
        return;
    }

    databaseReference.orderByChild( path: "nombre").equalTo(nombre)
        .addListenerForSingleValueEvent(new ValueEventListener() {
            @Override 2 usages
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                if (snapshot.exists()) {
                    for (DataSnapshot dataSnapshot : snapshot.getChildren()) {
                        Contacto contacto = dataSnapshot.getValue(Contacto.class);
                        if (contacto != null) {
                            contactoId = contacto.getId();
                            etNuevoNombre.setText(contacto.getNombre());
                            etNuevaEdad.setText(String.valueOf(contacto.getEdad()));
                            tvResultado.setText("Contacto encontrado: " + contacto.getNombre());
                        }
                    }
                } else {
                    tvResultado.setText("No se encontró el contacto.");
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                Toast.makeText(getContext(), text: "Error al buscar el contacto", Toast.LENGTH_SHORT).show();
            }
        });
}
```

```
private void actualizarContacto() { 1 usage
    String nuevoNombre = etNuevoNombre.getText().toString();
    String nuevaEdadStr = etNuevaEdad.getText().toString();

    if (TextUtils.isEmpty(nuevoNombre) || TextUtils.isEmpty(nuevaEdadStr)) {
        Toast.makeText(getContext(), text: "Por favor, complete todos los campos", Toast.LENGTH_SHORT).show();
        return;
    }

    int nuevaEdad = Integer.parseInt(nuevaEdadStr);

    if (contactoId != null) {
        Contacto contactoActualizado = new Contacto(contactoId, nuevoNombre, nuevaEdad);
        databaseReference.child(contactoId).setValue(contactoActualizado)
            .addOnSuccessListener(aVoid ->
                Toast.makeText(getContext(), text: "Contacto actualizado", Toast.LENGTH_SHORT).show())
            .addOnFailureListener(e ->
                Toast.makeText(getContext(), text: "Error al actualizar el contacto", Toast.LENGTH_SHORT).show());
    } else {
        Toast.makeText(getContext(), text: "Primero busque un contacto para actualizar", Toast.LENGTH_SHORT).show();
    }
}
```



Eliminar Contacto

```
private void eliminarContacto() { 1 usage
    if (contactoId != null) {
        databaseReference.child(contactoId).removeValue()
            .addOnSuccessListener(aVoid ->
                Toast.makeText(getContext(), text: "Contacto eliminado", Toast.LENGTH_SHORT).show())
            .addOnFailureListener(e ->
                Toast.makeText(getContext(), text: "Error al eliminar el contacto", Toast.LENGTH_SHORT).show());

        // Limpiar los campos después de eliminar
        etNuevoNombre.setText("");
        etNuevaEdad.setText("");
        etBuscarNombre.setText("");
        tvResultado.setText("Contacto eliminado.");
        contactoId = null;
    } else {
        Toast.makeText(getContext(), text: "Primero busque un contacto para eliminar", Toast.LENGTH_SHORT).show();
    }
}
```




SANTO
TOMÁS[®]
INSTITUTO PROFESIONAL



SANTO[®]
TOMÁS

INSTITUTO PROFESIONAL

Fin

