Firebase Services

`Firebase Authentication`

Most apps need to know the identity of a user. Knowing a user's identity allows an app to securely save user data in the cloud and provide the same personalized experience across all of the user's devices.

Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more.

Firebase Authentication integrates tightly with other Firebase services, and it leverages industry standards like OAuth 2.0 and OpenID Connect, so it can be easily integrated with your custom backend.

When you upgrade to Firebase Authentication with Identity Platform, you unlock additional features, such as multi-factor authentication, blocking functions, user activity and audit logging, SAML and generic OpenID Connect support, multi-tenancy, and enterprise-level support.

Firebase Authentication with Identity Platform has a different pricing scheme compared to the base product. When upgraded, no-cost (Spark) plan projects will be limited to 3,000 daily active users, and pay-as-you-go (Blaze) plan projects will be charged for usage beyond the free tier of 50,000 monthly active users. Be sure you understand the billing implications before you upgrade.

You can sign in users to your Firebase app either by using FirebaseUI as a complete drop-in auth solution or by using the Firebase Authentication SDK to manually integrate one or several sign-in methods into your app.

- Drop-in authentication solution

  The recommended way to add a complete sign-in system to your app.

  FirebaseUI provides a drop-in auth solution that handles the UI flows for signing in users with email addresses and passwords, phone numbers, and with popular federated identity providers, including Google Sign-In and Facebook Login.

  The FirebaseUI Auth component implements best practices for authentication on mobile devices and websites, which can maximize sign-in and sign-up conversion for your app. It also handles edge cases like account recovery and account linking that can be security sensitive and error-prone to handle correctly.

  FirebaseUI can be easily customized to fit in with the rest of your app's visual style, and it is open source, so you aren't constrained in realizing the user experience you want.

- Firebase SDK Authentication

  Email and password based authentication

  Authenticate users with their email addresses and passwords. The Firebase Authentication SDK provides methods to create and manage users that use their email addresses and passwords to sign in. Firebase Authentication also handles sending password reset emails.

Authentication also handles sending password reset emails.

Federated identity provider integration

Authenticate users by integrating with federated identity providers. The Firebase Authentication SDK provides methods that allow users to sign in with their Google, Facebook, Twitter, and GitHub accounts.

Phone number authentication

Authenticate users by sending SMS messages to their phones.

Custom auth system integration

Connect your app's existing sign-in system to the Firebase Authentication SDK and gain access to Firebase Realtime Database and other Firebase services.

Anonymous auth

Use features that require authentication without requiring users to sign in first by creating temporary anonymous accounts. If the user later chooses to sign up, you can upgrade the anonymous account to a regular account, so the user can continue where they left off.

How does Firebase Authentication work?

To sign a user into your app, you first get authentication credentials from the user. These credentials can be the user's email address and password, or an OAuth token from a federated identity provider. Then, you pass these credentials to the Firebase Authentication SDK. Our backend services will then verify those credentials and return a response to the client.

After a successful sign in, you can access the user's basic profile information, and you can control the user's access to data stored in other Firebase products. You can also use the provided authentication token to verify the identity of users in your own backend services.

Implementation paths

- Using FirebaseUI Auth

  Set up sign-in methods

  For email address and password or phone number sign-in and any federated identity providers you want to support, enable them in the Firebase console and complete any configuration required by the identity provider, such as setting your OAuth redirect URL.

  Customize the sign-in UI

  You can customize the sign-in UI by setting FirebaseUI options, or fork the code on GitHub to customize the sign-in experience further.

Use FirebaseUI to perform the sign-in flow

Import the FirebaseUI library, specify the sign-in methods you want to support, and initiate the FirebaseUI sign-in flow.

- Using the Firebase Authentication SDK

  Set up sign-in methods

  For email address and password or phone number sign-in and any federated identity providers you want to support, enable them in the Firebase console and complete any configuration required by the identity provider, such as setting your OAuth redirect URL.

  Implement UI flows for your sign-in methods

  For email address and password sign-in, implement a flow that prompts users to type their email addresses and passwords. For phone number sign-in, create a flow that prompts users for their phone number, and then for the code from the SMS message they receive. For federated sign-in, implement the flow required by each provider.

  Pass the user's credentials to the Firebase Authentication SDK

  Pass the user's email address and password or the OAuth token that was acquired from the federated identity provider to the Firebase Authentication SDK.

`Firebase Realtime Database`

Store and sync data with our NoSQL cloud database. Data is synced across all clients in realtime, and remains available when your app goes offline.

The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client. When you build cross-platform apps with our Apple platforms, Android, and JavaScript SDKs, all of your clients share one Realtime Database instance and automatically receive updates with the newest data.

Alternatively, consider trying Cloud Firestore for modern applications requiring richer data models, queryability, scalability and higher availability.

Key capabilities

- **Realtime**

  Instead of typical HTTP requests, the Firebase Realtime Database uses data synchronization—every time data changes, any connected device receives that update within milliseconds. Provide collaborative and immersive experiences without thinking about networking code.

- **Offline**

Firebase apps remain responsive even when offline because the Firebase Realtime Database SDK persists your data to disk. Once connectivity is reestablished, the client device receives any changes it missed, synchronizing it with the current server state.

- **Accessible from Client Devices**

  The Firebase Realtime Database can be accessed directly from a mobile device or web browser; there's no need for an application server. Security and data validation are available through the Firebase Realtime Database Security Rules, expression-based rules that are executed when data is read or written.

- **Scale across multiple databases**

  With Firebase Realtime Database on the Blaze pricing plan, you can support your app's data needs at scale by splitting your data across multiple database instances in the same Firebase project. Streamline authentication with Firebase Authentication on your project and authenticate users across your database instances. Control access to the data in each database with custom Firebase Realtime Database Security Rules for each database instance.

How does Firebase Realtime Database work?

The Firebase Realtime Database lets you build rich, collaborative applications by allowing secure access to the database directly from client-side code. Data is persisted locally, and even while offline, realtime events continue to fire, giving the end user a responsive experience. When the device regains connection, the Realtime Database synchronizes the local data changes with the remote updates that occurred while the client was offline, merging any conflicts automatically.

The Realtime Database provides a flexible, expression-based rules language, called Firebase Realtime Database Security Rules, to define how your data should be structured and when data can be read from or written to. When integrated with Firebase Authentication, developers can define who has access to what data, and how they can access it.

The Realtime Database is a NoSQL database and as such has different optimizations and capabilities compared to a relational database. The Realtime Database API is designed to only allow operations that can be executed quickly. This lets you build a great realtime experience that can serve millions of users without compromising on responsiveness. Because of this, it is important to think about how users need to access your data and then structure it accordingly.

Implementation path

1. **Integrate the Firebase Realtime Database SDKs**

   Quickly include clients using Gradle, CocoaPods, or a script include.

2. **Create Realtime Database References**

Reference your JSON data, such as "users/user:1234/phone_number" to set data or subscribe to data changes.

3. **Set Data and Listen for Changes**

   Use these references to write data or subscribe to changes.

4. **Enable Offline Persistence**

   Allow data to be written to the device's local disk so it can be available while offline.

5. **Secure your data**

   Use Firebase Realtime Database Security Rules to secure your data.

Store other types of data

- Cloud Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud.
- Firebase Remote Config stores developer specified key-value pairs to change the behavior and appearance of your app without requiring users to download an update.
- Firebase Hosting hosts the HTML, CSS, and JavaScript for your website as well as other developer-provided assets like graphics, fonts, and icons.
- Cloud Storage stores files such as images, videos, and audio as well as other user-generated content.

`Cloud Firestore`

Cloud Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud. Like Firebase Realtime Database, it keeps your data in sync across client apps through realtime listeners and offers offline support for mobile and web so you can build responsive apps that work regardless of network latency or Internet connectivity. Cloud Firestore also offers seamless integration with other Firebase and Google Cloud products, including Cloud Functions.

Key capabilities

Flexibility

The Cloud Firestore data model supports flexible, hierarchical data structures. Store your data in documents, organized into collections. Documents can contain complex nested objects in addition to subcollections.

Expressive querying

In Cloud Firestore, you can use queries to retrieve individual, specific documents or to retrieve all the documents in a collection that match your query parameters. Your queries can include multiple, chained filters and combine filtering and sorting. They're also indexed by default, so query performance is proportional to the size of your result set, not your data set.

Realtime updates

Like Realtime Database, Cloud Firestore uses data synchronization to update data on any connected device. However, it's also designed to make simple, one-time fetch queries efficiently.

Offline support

Cloud Firestore caches data that your app is actively using, so the app can write, read, listen to, and query data even if the device is offline. When the device comes back online, Cloud Firestore synchronizes any local changes back to Cloud Firestore.

Designed to scale

Cloud Firestore brings you the best of Google Cloud's powerful infrastructure: automatic multi-region data replication, strong consistency guarantees, atomic batch operations, and real transaction support. We've designed Cloud Firestore to handle the toughest database workloads from the world's biggest apps.

How does Cloud Firestore work?

Cloud Firestore is a cloud-hosted, NoSQL database that your Apple, Android, and web apps can access directly via native SDKs. Cloud Firestore is also available in native Node.js, Java, Python, Unity, C++ and Go SDKs, in addition to REST and RPC APIs.

Following Cloud Firestore's NoSQL data model, you store data in documents that contain fields mapping to values. These documents are stored in collections, which are containers for your documents that you can use to organize your data and build queries. Documents support many different data types, from simple strings and numbers, to complex, nested objects. You can also create subcollections within documents and build hierarchical data structures that scale as your database grows. The Cloud Firestore data model supports whatever data structure works best for your app.

Additionally, querying in Cloud Firestore is expressive, efficient, and flexible. Create shallow queries to retrieve data at the document level without needing to retrieve the entire collection, or any nested subcollections. Add sorting, filtering, and limits to your queries or cursors to paginate your results. To keep data in your apps current, without retrieving your entire database each time an update happens, add realtime listeners. Adding realtime listeners to your app notifies you with a data snapshot whenever the data your client apps are listening to changes, retrieving only the new changes.

Protect access to your data in Cloud Firestore with Firebase Authentication and Cloud Firestore Security Rules for Android, Apple platforms, and JavaScript, or Identity and Access Management (IAM) for server-side languages.

Implementation path

1. **Integrate the Cloud Firestore SDKs**

   Quickly include clients via Gradle, CocoaPods, or a script include.

2. **Secure your data**

   Use Cloud Firestore Security Rules or Identity and Access Management (IAM) to secure your data for mobile/web and server development, respectively.

3. **Add Data**

   Create documents and collections in your database.

4. **Get Data**

   Create queries or use realtime listeners to retrieve data from the database.

Cloud Storage for Firebase

Cloud Storage for Firebase is built on fast and secure Google Cloud infrastructure for app developers who need to store and serve user-generated content, such as photos or videos.

Cloud Storage for Firebase is a powerful, simple, and cost-effective object storage service built for Google scale. The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for your Firebase apps, regardless of network quality.

You can use our client SDKs to store images, audio, video, or other user-generated content. On the server, you can use the Firebase Admin SDK to manage buckets and create download URLs, and use Google Cloud Storage APIs to access your files.

Key capabilities

- **Robust operations**

  Firebase SDKs for Cloud Storage perform uploads and downloads regardless of network quality. Uploads and downloads are robust, meaning they restart where they stopped, saving your users time and bandwidth.

- **Strong security**

  Firebase SDKs for Cloud Storage integrate with Firebase Authentication to provide simple and intuitive authentication for developers. You can use our declarative security model to allow access based on filename, size, content type, and other metadata.

- **High scalability**

Cloud Storage is built for exabyte scale when your app goes viral. Effortlessly grow from prototype to production using the same infrastructure that powers Spotify and Google Photos.

How does Cloud Storage work?

Developers use the Firebase SDKs for Cloud Storage to upload and download files directly from clients. If the network connection is poor, the client is able to retry the operation right where it left off, saving your users time and bandwidth.

Cloud Storage for Firebase stores your files in a Google Cloud Storage bucket, making them accessible through both Firebase and Google Cloud. This allows you the flexibility to upload and download files from mobile clients via the Firebase SDKs for Cloud Storage. In addition, you can do server-side processing such as image filtering or video transcoding using the Google Cloud Storage APIs. Cloud Storage scales automatically, meaning that there's no need to migrate to any other provider. Learn more about all the benefits of our integration with Google Cloud.

The Firebase SDKs for Cloud Storage integrate seamlessly with Firebase Authentication to identify users, and we provide a declarative security language that lets you set access controls on individual files or groups of files, so you can make files as public or private as you want.

Implementation path

1. **Integrate the Firebase SDKs for Cloud Storage.**

   Quickly include clients via Gradle, CocoaPods, or a script include.

2. **Create a Reference**

   Reference the path to a file, such as "images/mountains.png", to upload, download, or delete it.

3. **Upload or Download**

   Upload or download to native types in memory or on disk.

4. **Secure your Files**

   Use Firebase Security Rules for Cloud Storage to secure your files.

5. **(Optional) Create and Share Download URLs**

   Use the Firebase Admin SDK to generate shareable URLs to let users download objects.