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.

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

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.

`Cloud Functions for Firebase`

Cloud Functions for Firebase is a serverless framework that lets you automatically run backend code in response to events triggered by background events, HTTPS requests, the Admin SDK, or Cloud Scheduler jobs. Your JavaScript, TypeScript or Python code is stored on Google Cloud infrastructure and runs in a managed environment. There's no need to manage and scale your own servers.

Key capabilities

- **Integrates Firebase features and connects Firebase with Google Cloud**

  The functions you write can respond to events generated by various Firebase and Google Cloud features, from Firebase Authentication triggers to Cloud Storage Triggers.

  Integrate across Firebase features using the Admin SDK together with Cloud Functions, and integrate with third-party services by writing your own webhooks. Cloud Functions minimizes boilerplate code, making it easier to use Firebase and Google Cloud inside your function.

- **Zero maintenance**

  Deploy your JavaScript, TypeScript, or Python code to our servers with one command from the command line. After that, Firebase automatically scales up computing resources to match the usage patterns of your users. You never worry about credentials, server configuration, provisioning new servers, or decommissioning old ones.

- **Keeps your logic private and secure**

In many cases, developers prefer to control application logic on the server to avoid tampering on the client side. Also, sometimes it's not desirable to allow that code to be reverse engineered. Cloud Functions is fully insulated from the client, so you can be sure it is private and always does exactly what you want.

How does Cloud Functions work?

After you write and deploy a function, Google's servers begin to manage the function immediately. You can fire the function directly with an HTTP request, the Admin SDK, or a scheduled job, or, in the case of background functions, Google's servers listen for events and run the function when it is triggered.

As the load increases or decreases, Google responds by rapidly scaling the number of virtual server instances needed to run your function. Each function runs in isolation, in its own environment with its own configuration.

Lifecycle of a background function

1. You write code for a new function, selecting an event provider (such as Cloud Firestore), and defining the conditions under which the function should execute.

2. When you deploy your function:The Firebase CLI creates a .zip archive of the function code, which is then uploaded to a Cloud Storage bucket (prefixed with gcf-sources) before Cloud Functions creates an Artifact Registry repository (named gcf-artifacts) in your project.

   The Firebase CLI creates a .zip archive of the function code, which is then uploaded to a Cloud Storage bucket (prefixed with gcf-sources) before Cloud Functions creates an Artifact Registry repository (named gcf-artifacts) in your project.

   Cloud Build retrieves the function code and builds the function source. You can view Cloud Build logs in the Google Cloud Console.

   The container image for the built functions code is uploaded to a private Artifact Registry repository in your project (named gcf-artifacts), and your new function is rolled out.

3. When the event provider generates an event that matches the function's conditions, the code is invoked.

4. If the function is busy handling many events, Google creates more instances to handle work faster. If the function is idle, instances are cleaned up.

5. When you update the function by deploying updated code, instances for older versions are cleaned up along with build artifacts in Artifact Registry, and replaced by new instances.

6. When you delete the function, all instances and zip archives are cleaned up, along with related build artifacts in Artifact Registry. The connection between the function and the event provider is removed.

In addition to listening for events with a background function, you can call functions directly with an HTTP request or a call from the client. You can also trigger functions on a fixed schedule or enqueue task functions via the Admin SDK.

Implementation path

1. **Set up Cloud Functions**

   Install the Firebase CLI and initialize Cloud Functions in your Firebase project.

2. **Write functions**

   Write JavaScript code, TypeScript code, or Python code to handle events from Firebase services, Google Cloud services, or other event providers.

3. **Test functions**

   Use the local emulator to test your functions.

4. **Deploy and monitor**

   Enable billing for your project and deploy your functions using the Firebase CLI. You can use the Google Cloud Console to view and search through your logs.

Firebase Hosting

Firebase Hosting provides fast and secure hosting for your web app, static and dynamic content, and microservices.

Firebase Hosting is production-grade web content hosting for developers. With a single command, you can quickly deploy web apps and serve both static and dynamic content to a global CDN (content delivery network). You can also pair Firebase Hosting with Cloud Functions or Cloud Run to build and host microservices on Firebase.

Key capabilities

- **Serve content over a secure connection**

  The modern web is secure. Zero-configuration SSL is built into Firebase Hosting, so content is always delivered securely.

- **Host static and dynamic content plus microservices**

  Firebase Hosting supports all kinds of content for hosting, from your CSS and HTML files to your Express.js microservices or APIs.

- **Deliver content fast**

  Each file that you upload is cached on SSDs at CDN edges around the world and served as gzip or Brotli. We auto-select the best compression method for your content. No matter where your users are, the content is delivered fast.

- **Emulate and even share your changes before going live**

  View and test your changes on a locally hosted URL and interact with an emulated backend.

  Share your changes with teammates using temporary preview URLs. Hosting also provides a GitHub integration for easy iterations of your previewed content.

- **Deploy new versions with one command**

  Using the Firebase CLI, you can get your app up and running in seconds. Command line tools make it easy to add deployment targets into your build process.

And if you need to undo the deploy, Hosting provides one-click rollbacks.

How does Firebase Hosting work?

Firebase Hosting is built for the modern web developer. Websites and apps are more powerful than ever with the rise of front-end JavaScript frameworks like Angular and static generator tools like Jekyll. Whether you are deploying a simple app landing page or a complex Progressive Web App (PWA), Hosting gives you the infrastructure, features, and tooling tailored to deploying and managing websites and apps.

Using the Firebase CLI, you deploy files from local directories on your computer to our Hosting servers. Beyond serving static content, you can use Cloud Functions for Firebase or Cloud Run to serve dynamic content and host microservices on your sites. All content is served over an SSL connection from the closest edge server on our global CDN.

You can also view and test your changes before going live. Using the Firebase Local Emulator Suite, you can emulate your app and backend resources at a locally hosted URL. You can also share your changes at a temporary preview URL and set up a GitHub integration for easy iterations during development.

Firebase Hosting has lightweight hosting configuration options for you to build sophisticated PWAs. You can easily rewrite URLs for client-side routing, set up custom headers, and even serve localized content.

For serving your content, Firebase offers several domain and subdomain options:

- By default, every Firebase project has subdomains at no cost on the web.app and firebaseapp.com domains. These two sites serve the same deployed content and configuration.
- You can create multiple sites if you have related sites and apps that serve different content but still share the same Firebase project resources (for example if you have a blog, admin panel, and public app).
- You can connect your own domain name to a Firebase-hosted site.

Firebase automatically provisions SSL certificates for all your domains so that all your content is served securely.

Implementation path

1. **Install the Firebase CLI**

   The Firebase CLI makes it easy to set up a new Hosting project, run a local development server, and deploy content.

2. **Set up a project directory**

   Add your static assets to a local project directory, then run firebase init to connect the directory to a Firebase project.

   In your local project directory, you can also set up Cloud Functions or Cloud Run for your dynamic content and microservices.

3. **View, test, and share your changes before going live *(optional)***

   Run firebase emulators: start to emulate Hosting and your backend project resources at a locally hosted URL.

   To view and share your changes at a temporary preview URL, run firebase hosting:channel:deploy to create and deploy to a preview channel. Set up the GitHub integration for easy iterations of your previewed content.

4. **Deploy your site**

   When things are looking good, run firebase deploy to upload the latest snapshot to our servers. If you need to undo the deploy, you can roll back with just one click in the Firebase console.

5. **Link to a Firebase Web App *(optional)***

   By linking your site to a Firebase Web App, you can use Google Analytics to collect usage and behavior data for your app and use Firebase Performance Monitoring to gain insight into the performance characteristics of your app.

Firebase Cloud Messaging

Firebase Cloud Messaging (FCM) is a cross-platform messaging solution that lets you reliably send messages at no cost.

Using FCM, you can notify a client app that new email or other data is available to sync. You can send notification messages to drive user re-engagement and retention. For use cases such as instant messaging, a message can transfer a payload of up to 4000 bytes to a client app.

Key capabilities

- **Send notification messages or data messages**

  Send notification messages that are displayed to your user. Or send data messages and determine completely what happens in your application code.

- **Versatile message targeting**

  Distribute messages to your client app in any of 3 ways—to single devices, to groups of devices, or to devices subscribed to topics.

- **Send messages from client apps**

  Send acknowledgments, chats, and other messages from devices back to your server over FCM's reliable and battery-efficient connection channel.

How does FCM work?

An FCM implementation includes two main components for sending and receiving:

1. A trusted environment such as Cloud Functions for Firebase or an app server on which to build, target, and send messages.
2. An Apple, Android, or web (JavaScript) client app that receives messages via the corresponding platform-specific transport service.

You can send messages via the Firebase Admin SDK or the FCM server protocol. You can use the Notifications composer for testing and to send marketing or engagement

messages using powerful built-in targeting and analytics or custom imported segments.

Implementation path

- **Set up the FCM SDK**

  Set up Firebase and FCM on your app according to the setup instructions for your platform.

- **Develop your client app**

  Add message handling, topic subscription logic, or other optional features to your client app. During the development, you can easily send test messages from the Notifications composer.

- **Develop your app server**

  Decide whether you want to use the Firebase Admin SDK or the server protocol to create your sending logic—logic to authenticate, build send requests, handle responses, and so on. Then build out the logic in your trusted environment.

`Google Analytics`

Google Analytics is an app measurement solution, available at no charge, that provides insight on app usage and user engagement.

At the heart of Firebase is Google Analytics, an unlimited analytics solution available at no charge. Analytics integrates across Firebase features and provides you with unlimited reporting for up to 500 distinct events that you can define using the Firebase SDK. Analytics reports help you understand clearly how your users behave, which enables you to make informed decisions regarding app marketing and performance optimizations.

Key capabilities

- **Unlimited Reporting**

  Analytics provides unlimited reporting on up to 500 distinct events.

- **Audience Segmentation**

  Custom audiences can be defined in the Firebase console based on device data, custom events, or user properties. These audiences can be used with other Firebase features when targeting new features or notification messages.

How does Google Analytics work?

Google Analytics helps you understand how people use your web, Apple, or Android app. The SDK automatically captures a number of events and user properties and also allows you to define your own custom events to measure the things that uniquely matter to your business. Once the data is captured, it's available in a dashboard through the Firebase console. This dashboard provides detailed insights about your data — from summary data such as active users and demographics, to more detailed data such as identifying your most purchased items.

Analytics also integrates with a number of other Firebase features. For example, it automatically logs events that correspond to notification messages sent via the Notifications composer and provides reporting on the impact of each campaign.

Analytics helps you understand how your users behave, so you can make informed decisions about how to market your app. See the performance of your campaigns across organic and paid channels to understand which methods are most effective at driving high-value users. If you need to perform custom analysis or join your data with other sources you can link your Analytics data to BigQuery, which allows for more complex analysis like querying large data sets and joining multiple data sources.

Integrations with other services

- **BigQuery**

  Link your Firebase app to BigQuery where you can perform custom analysis on your entire Analytics dataset and import other data sources.

- **Crashlytics**

Analytics logs events for each crash so you can get a sense of the rate of crashes for different versions or regions, allowing you to gain insight into which users are impacted. You can also create audiences for users who have experienced multiple crashes and respond with notification messages directed at that audience.

- **FCM**

  Analytics automatically logs events that correspond to notification messages sent via the Notifications composer and supports reporting on the impact of each campaign.

- **Firebase Remote Config**

  Use Analytics audience definitions to change the behavior and appearance of your app for different audiences without distributing multiple versions of your app.

- **Google Tag Manager**

  Integrating Google Tag Manager alongside Google Analytics enables you to manage your Analytics implementation remotely from a web interface after your app has been distributed.

Implementation path

1. **Connect your app to Firebase**

   Getting started with Analytics is easy. Just add the Firebase SDK to your new or existing app, and data collection begins automatically. You can view analytics data in the Firebase console within hours.

2. **Log custom data**

   You can use Analytics to log custom events that make sense for your app, like E-Commerce purchases or achievements.

3. **Create audiences**

You can define the audiences that matter to you in the Firebase console.

4. **Target audiences**

   Use your custom audiences to target messages, promotions, or new app features using other Firebase features, such as FCM, and Remote Config.

Firebase Crashlytics

Get clear, actionable insight into app issues with this powerful crash reporting solution for Apple, Android, Flutter, and Unity.

Firebase Crashlytics is a lightweight, realtime crash reporter that helps you track, prioritize, and fix stability issues that erode your app quality. Crashlytics saves you troubleshooting time by intelligently grouping crashes and highlighting the circumstances that lead up to them.

Find out if a particular crash is impacting a lot of users. Get alerts when an issue suddenly increases in severity. Figure out which lines of code are causing crashes.

Key capabilities

- **Curated crash reports**

  Crashlytics synthesizes an avalanche of crashes into a manageable list of issues, provides contextual information, and highlights the severity and prevalence of crashes so you can pinpoint the root cause faster.

- **Cures for the common crash**

  Crashlytics offers Crash Insights, helpful tips that highlight common stability problems and provide resources that make them easier to troubleshoot, triage, and resolve.

- **Integrated with Analytics**

Crashlytics can capture your app's errors as app_exception events in Analytics. The events simplify debugging by giving you access a list of other events leading up to each crash, and provide audience insights by letting you pull Analytics reports for users with crashes.

- **Realtime alerts**

  Get realtime alerts for new issues, regressed issues, and growing issues that might require immediate attention.

Implementation path

1. **Connect your app**

   Start by adding Firebase to your app in the Firebase console.

2. **Integrate the SDK**

   Add the Crashlytics SDK via CocoaPods, Gradle, or Pub, and Crashlytics starts collecting reports.

3. **Check reports in the Firebase console**

   Visit the Firebase console to track, prioritize, and fix issues in your app.

How does Crashlytics analyze your crashes for easier debugging?

To provide metrics and reports about your app, Crashlytics collects and analyzes crashes, non-fatal exceptions, and other event types from your app. We use the mapping information for your app's build to create human-readable crash reports to help you understand the events (for example, we use the debug symbol (dSYM) files for Apple platform apps).

When Crashlytics receives events, it uses an analysis engine to group related events into ***issues***. The analysis engine looks at the frames in the stack trace, exception message, error code, and other platform or error type characteristics to group events into issues. In an issue, all events have a common point of failure. As more incoming events match to an issue, the issue rises towards the top of your app's *Issues* table in the Crashlytics dashboard. This grouping and ranking helps you identify and fix the most impactful problems faster.

Yet even within this group of events, the stack traces leading to the point-of-failure might be different. And a different stack trace could mean a different root cause. To represent this possible difference within an issue, Crashlytics creates ***variants*** within issues — each variant is a sub-group of events in an issue that have the same failure point *and* a similar stack trace. With variants, you can debug the most common stack traces within an issue and determine if different root causes are leading to the failure.