Expo Docs

Search

User interface

Development builds

Config plugins

Debugging

Celebrating the best apps built with Expo

Create amazing apps that run everywhere

Build one JavaScript/TypeScript project that runs natively on all your users' devices.

Quick Start

Copy

-

```
npx create-expo-app@latest
```

Then continue setting up your environment.

Create a universal Android, iOS, and web app

Launch to app stores

Ship apps with zero config or no prior experience. Launch easily guides you through the technical stuff, directly from GitHub. No config or prior knowledge needed.

Deploy from CLI

Deploy your apps using command-line tools for iOS and web platforms.

Deploy to TestFlight

Copy

-

```
npx testflight
```

This is an iOS-only command that will upload your app to TestFlight.

Deploy your web app

Copy

-

```
npx eas-cli deploy
```

For prerequisites and complete instructions, see our guide.

Discover more

Try out Expo in minutes and learn how to get the most out of Expo.

Speed up your development with Expo Application Services

Discover the benefits of file-based routing with Expo Router

Try Expo in your browser

Expo's Snack lets you try Expo with zero local setup.

Chat with the community

Join over 50,000 other developers
on the Expo Community Discord.

Explore APIs

Expo supplies a vast array of SDK modules. You can also create your own.

Explore examples

Explore a variety of example projects showcasing how to use Expo and seamlessly integrate it with popular services.

Watch our latest talks

Explore our team's presentations. Stay informed and gain expertise.

Charlie Cheever, Jon Samp

App.js Conf 2025

Evan Bacon

App.js Conf 2025

Keith Kurak

App.js Conf 2025

Gabriel Donadel

App.js Conf 2024

Join the community

See the source code, connect with others, and get connected.

Discord and Forums

Join our Discord to chat, ask questions or attend events.

GitHub

View SDK and docs code, submit a PR, or report an issue.

YouTube

Follow our channel to explore tutorials and other content.

LinkedIn

Follow Expo on LinkedIn for news and updates.

Bluesky

Follow Expo on Bluesky for news and updates.

X

Follow Expo on X for news and updates.

Reddit

Get the latest on r/expo.

Canny

Give us a feedback or request a feature.

Was this doc helpful?

Share your feedback

Share your feedback

Sign up for the Expo Newsletter

Unsubscribe at any time. Read our privacy policy.

Supabase Docs

Start

Products

Build

Manage

Reference

Resources

Search
docs...

Start

Products

Build

Manage

Reference

Resources

Search docs...

Supabase Documentation

Learn how to get up and running with Supabase through tutorials, APIs and platform resources.

Getting Started

Set up and connect a database in just a few minutes.

Products

Database

Supabase provides a full Postgres database for every project with Realtime functionality, database backups, extensions, and more.

Database

Auth

Add and manage email and password, passwordless, OAuth, and mobile logins to your project through a suite of identity providers and APIs.

Auth

Storage

Store, organize, transform, and serve large files—fully integrated with your Postgres database with Row Level Security access policies.

Storage

Realtime

Listen to database changes, store and sync user states across clients, broadcast data to clients subscribed to a channel, and more.

Realtime

Edge Functions

Globally distributed, server-side functions to execute your code closest to your users for the lowest latency.

Edge Functions

Postgres Modules

Client Libraries

Migrate to Supabase

Bring your existing data, auth and storage to Supabase following our migration guides.

Amazon RDS

Auth0

Firebase Auth

Firebase Storage

Firestore Data

Heroku

MSSQL

MySQL

Neon

Postgres

Render

Vercel Postgres

Additional resources

Management API

Manage your Supabase projects and organizations.

Management API

Supabase CLI

Use the CLI to develop, manage and deploy your projects.

Supabase CLI

Platform Guides

Learn more about the tools and services powering Supabase.

Platform Guides

Integrations

Explore a variety of integrations from Supabase partners.

Integrations

Supabase UI

A collection of pre-built Supabase components to speed up your project.

Supabase UI

Self-Hosting

Get started with self-hosting Supabase.

Auth

Realtime

Storage

Analytics

Need some help?

Contact support

Need some help?

Latest product updates?

See Changelog

Latest product updates?

Something's not right?

Check system status

Something's not right?

Stripe React Native SDK

Accept a payment

Securely accept payments online.

Build a payment form or use a prebuilt checkout page to start accepting online payments.

This integration combines all of the steps required to pay—collecting payment details and confirming the payment—into a single sheet that displays on top of your app.

Set up Stripe
Server-side
Client-side

First, you need a Stripe account. Register now.

Server-side

This integration requires endpoints on your server that talk to the Stripe API. Use the official libraries for access to the Stripe API from your server:

```
# Available as a gem
sudo gem install stripe
```

```
# Available as a gem
sudo gem install stripe
```

```
# If you use bundler, you can add this line to your Gemfile
gem 'stripe'
```

```
# If you use bundler, you can add this line to your Gemfile
gem 'stripe'
```

Client-side

The React Native SDK is open source and fully documented. Internally, it uses the native iOS and Android SDKs. To install Stripe's React Native SDK, run one of the following commands in your project's directory (depending on which package manager you use):

```
yarn add @stripe/stripe-react-native
```

```
yarn add @stripe/stripe-react-native
```

Next, install some other necessary dependencies:

For iOS, go to the ios directory and run pod install to ensure that you also install the required native dependencies.

```
pod install
```

For Android, there are no more dependencies to install.

We recommend following the official TypeScript guide to add TypeScript support.

Stripe initialization

To initialize Stripe in your React Native app, either wrap your payment screen with the StripeProvider component, or use the initStripe initialization method. Only the API publishable key in publishableKey is required. The following example shows how to initialize Stripe using the StripeProvider component.

StripeProvider

initStripe

publishableKey

StripeProvider

```
import { useState, useEffect } from 'react';
import { StripeProvider } from '@stripe/stripe-react-native';

function App() {
  const [publishableKey, setPublishableKey] = useState('');

  const fetchPublishableKey = async () => {
    const key = await fetchKey(); // fetch key from your server here
    setPublishableKey(key);
  };

  useEffect(() => {
    fetchPublishableKey();
  }, []);

  return (
    <StripeProvider
      publishableKey={publishableKey}
      merchantIdentifier="merchant.identifier" // required for Apple Pay
      urlScheme="your-url-scheme" // required for 3D Secure and bank redirects
    >
      {/* Your app code here */}
    </StripeProvider>
  );
}
```

```
import { useState, useEffect } from 'react';
import { StripeProvider } from '@stripe/stripe-react-native';

function App() {
  const [publishableKey, setPublishableKey] = useState('');

  const fetchPublishableKey = async () => {
    const key = await fetchKey(); // fetch key from your server here
    setPublishableKey(key);
  };

  useEffect(() => {
    fetchPublishableKey();
  }, []);

  return (
    <StripeProvider
      publishableKey={publishableKey}
      merchantIdentifier="merchant.identifier" // required for Apple Pay
      urlScheme="your-url-scheme" // required for 3D Secure and bank redirects
    >
      {/* Your app code here */}
    </StripeProvider>
  );
}
```

Use your API test keys while you test and develop, and your live mode keys when you publish your app.

Enable payment methods

View your payment methods settings and enable the payment methods you want to support. You need at least one payment method enabled to create a PaymentIntent.

By default, Stripe enables cards and other prevalent payment methods that can help you reach more customers, but we recommend turning on additional payment methods that are relevant for your business and customers. See Payment method support for product and payment method support, and our pricing page for fees.

Add an endpoint
Server-side

To display the PaymentSheet before you create a PaymentIntent, see Collect payment details before creating an Intent.

This integration uses three Stripe API objects:

PaymentIntent: Stripe uses this to represent your intent to collect payment from a customer, tracking your charge attempts and payment state changes throughout the process.

PaymentIntent: Stripe uses this to represent your intent to collect payment from a customer, tracking your charge attempts and payment state changes throughout the process.

(Optional) Customer: To set up a payment method for future payments, you must attach it to a Customer. Create a Customer object when your customer creates an account with your business. If your customer is making a payment as a guest, you can create a Customer object before payment and associate it with your own internal representation of the customer’s account later.

(Optional) Customer: To set up a payment method for future payments, you must attach it to a Customer. Create a Customer object when your customer creates an account with your business. If your customer is making a payment as a guest, you can create a Customer object before payment and associate it with your own internal representation of the customer’s account later.

(Optional) Customer Ephemeral Key: Information on the Customer object is sensitive, and can’t be retrieved directly from an app. An Ephemeral Key grants the SDK temporary access to the Customer.

(Optional) Customer Ephemeral Key: Information on the Customer object is sensitive, and can’t be retrieved directly from an app. An Ephemeral Key grants the SDK temporary access to the Customer.

If you never save cards to a Customer and don’t allow returning Customers to reuse saved cards, you can omit the Customer and Customer Ephemeral Key objects from your integration.

For security reasons, your app can’t create these objects. Instead, add an endpoint on your server that:

Retrieves the Customer, or creates a new one.

Creates an Ephemeral Key for the Customer.

Creates a PaymentIntent with the amount, currency, and customer. You can also optionally include the automatic_payment_methods parameter. Stripe enables its functionality by default in the latest version of the API.

automatic_payment_methods

Returns the Payment Intent’s client secret, the Ephemeral Key’s secret, the Customer’s id, and your publishable key to your app.

secret

The payment methods shown to customers during the checkout process are also included on the PaymentIntent. You can let Stripe pull payment methods from your Dashboard settings or you can list them manually. Regardless of the option you choose, know that the currency passed in the PaymentIntent filters the payment methods shown to the customer. For example, if you pass eur on the PaymentIntent and have OXXO enabled in the Dashboard, OXXO won't be shown to the customer because OXXO doesn't support eur payments.

eur

eur

Unless your integration requires a code-based option for offering payment methods, Stripe recommends the automated option. This is because Stripe evaluates the currency, payment method restrictions, and other parameters to determine the list of supported payment methods. Payment methods that increase conversion and that are most relevant to the currency and customer's location are prioritized.

You can fork and deploy an implementation of this endpoint on CodeSandbox for testing.

You can manage payment methods from the Dashboard. Stripe handles the return of eligible payment methods based on factors such as the transaction's amount, currency, and payment flow. The PaymentIntent is created using the payment methods you configured in the Dashboard. If you don't want to use the Dashboard or if you want to specify payment methods manually, you can list them using the payment_method_types attribute.

payment_method_types

```
# Create a Customer (use an existing Customer ID if this is a returning customer)
curl https://api.stripe.com/v1/customers \
  -u sk_test_CsnggH3iChIYjrFoue5y6M98: \
  -X "POST" \
  -H "Stripe-Account: {{CONNECTED_ACCOUNT_ID}}"

# Create an Ephemeral Key for the Customer
curl https://api.stripe.com/v1/ephemeral_keys \
  -u sk_test_CsnggH3iChIYjrFoue5y6M98: \
  -H "Stripe-Version: 2025-09-30.clover" \
  -H "Stripe-Account: 2025-09-30.clover" \
  -X "POST" \
  -d "customer"="{{CUSTOMER_ID}}" \

# Create a PaymentIntent
curl https://api.stripe.com/v1/payment_intents \
  -u sk_test_CsnggH3iChIYjrFoue5y6M98: \
  -X "POST" \
  -d "customer"="{{CUSTOMER_ID}}" \
  -d "amount"=1099 \
  -d "currency"="eur" \
  # In the latest version of the API, specifying the `automatic_payment_methods` parameter
  # is optional because Stripe enables its functionality by default.
  -d "automatic_payment_methods[enabled]"=true \

# Create a Customer (use an existing Customer ID if this is a returning customer)
curl https://api.stripe.com/v1/customers \
  -u sk_test_CsnggH3iChIYjrFoue5y6M98: \
  -X "POST" \
  -H "Stripe-Account: {{CONNECTED_ACCOUNT_ID}}"

# Create an Ephemeral Key for the Customer
curl https://api.stripe.com/v1/ephemeral_keys \
  -u sk_test_CsnggH3iChIYjrFoue5y6M98: \
  -H "Stripe-Version: 2025-09-30.clover" \
  -H "Stripe-Account: 2025-09-30.clover" \
  -X "POST" \
  -d "customer"="{{CUSTOMER_ID}}" \
# Create a PaymentIntent
curl https://api.stripe.com/v1/payment_intents \
  -u sk_test_CsnggH3iChIYjrFoue5y6M98: \
  -X "POST" \
  -d "customer"="{{CUSTOMER_ID}}" \
  -d "amount"=1099 \
  -d "currency"="eur" \
  # In the latest version of the API, specifying the `automatic_payment_methods` parameter
  # is optional because Stripe enables its functionality by default.
  -d "automatic_payment_methods[enabled]"=true \
```

Collect payment details
Client-side

Before displaying the mobile Payment Element, your checkout page should:

Show the products being purchased and the total amount

Collect any required shipping information

Include a checkout button to present Stripe’s UI

In the checkout of your app, make a network request to the backend endpoint you created in the previous step and call initPaymentSheet from the useStripe hook.

initPaymentSheet

useStripe

```
export default function CheckoutScreen() {
  const { initPaymentSheet, presentPaymentSheet } = useStripe();
  const [loading, setLoading] = useState(false);

  const fetchPaymentSheetParams = async () => {
    const response = await fetch(`${API_URL}/payment-sheet`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
    });
    const { paymentIntent, ephemeralKey, customer } = await response.json();

    return {
      paymentIntent,
      ephemeralKey,
      customer,
    };
  };

  const initializePaymentSheet = async () => {
    const {
      paymentIntent,
      ephemeralKey,
      customer,
    } = await fetchPaymentSheetParams();

    const { error } = await initPaymentSheet({
      merchantDisplayName: "Example, Inc.",
      customerId: customer,
      customerEphemeralKeySecret: ephemeralKey,
      paymentIntentClientSecret: paymentIntent,
      // Set `allowsDelayedPaymentMethods` to true if your business can handle payment
      //methods that complete payment after a delay, like SEPA Debit and Sofort.
      allowsDelayedPaymentMethods: true,
      defaultBillingDetails: {
        name: 'Jane Doe',
      }
    });
    if (!error) {
      setLoading(true);
    }
  };

  const openPaymentSheet = async () => {
    // see below
  };

  useEffect(() => {
    initializePaymentSheet();
  }, []);

  return (
    <Screen>
      <Button
        variant="primary"
        disabled={!loading}
        title="Checkout"
        onPress={openPaymentSheet}
      />
    </Screen>
  );
}

export default function CheckoutScreen() {
  const { initPaymentSheet, presentPaymentSheet } = useStripe();
  const [loading, setLoading] = useState(false);

  const fetchPaymentSheetParams = async () => {
    const response = await fetch(`${API_URL}/payment-sheet`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
    });
    const { paymentIntent, ephemeralKey, customer } = await response.json();

    return {
      paymentIntent,
      ephemeralKey,
      customer,
    };
  };

  const initializePaymentSheet = async () => {
    const {
      paymentIntent,
      ephemeralKey,
      customer,
    } = await fetchPaymentSheetParams();

    const { error } = await initPaymentSheet({
      merchantDisplayName: "Example, Inc.",
      customerId: customer,
      customerEphemeralKeySecret: ephemeralKey,
      paymentIntentClientSecret: paymentIntent,
      // Set `allowsDelayedPaymentMethods` to true if your business can handle payment
      //methods that complete payment after a delay, like SEPA Debit and Sofort.
      allowsDelayedPaymentMethods: true,
      defaultBillingDetails: {
        name: 'Jane Doe',
      }
    });
    if (!error) {
      setLoading(true);
    }
  };

  const openPaymentSheet = async () => {
    // see below
  };

  useEffect(() => {
    initializePaymentSheet();
  }, []);

  return (
    <Screen>
      <Button
        variant="primary"
        disabled={!loading}
        title="Checkout"
        onPress={openPaymentSheet}
      />
    </Screen>
  );
}
```

When your customer taps the Checkout button, call presentPaymentSheet() to open the sheet. After the customer completes the payment, the sheet is dismissed and the promise resolves with an optional StripeError<PaymentSheetError>.

presentPaymentSheet()

StripeError<PaymentSheetError>

```
export default function CheckoutScreen() {
  // continued from above

  const openPaymentSheet = async () => {
    const { error } = await presentPaymentSheet();

    if (error) {
      Alert.alert(`Error code: ${error.code}`, error.message);
    } else {
      Alert.alert('Success', 'Your order is confirmed!');
    }
  };

  return (
    <Screen>
      <Button
        variant="primary"
        disabled={!loading}
        title="Checkout"
        onPress={openPaymentSheet}
      />
    </Screen>
  );
}

export default function CheckoutScreen() {
  // continued from above

  const openPaymentSheet = async () => {
    const { error } = await presentPaymentSheet();

    if (error) {
      Alert.alert(`Error code: ${error.code}`, error.message);
    } else {
      Alert.alert('Success', 'Your order is confirmed!');
    }
  };

  return (
    <Screen>
      <Button
        variant="primary"
        disabled={!loading}
        title="Checkout"
        onPress={openPaymentSheet}
      />
    </Screen>
  );
}
```

If there is no error, inform the user they're done (for example, by displaying an order confirmation screen).

Setting allowsDelayedPaymentMethods to true allows delayed notification payment methods like US bank accounts. For these payment methods, the final payment status isn't known when the PaymentSheet completes, and instead succeeds or fails later. If you support these types of payment methods, inform the customer their order is confirmed and only fulfill their order (for example, ship their product) when the payment is successful.

allowsDelayedPaymentMethods

PaymentSheet

Set up a return URL (iOS only)
Client-side

When a customer exits your app (for example to authenticate in Safari or their banking app), provide a way for them to automatically return to your app. Many payment method types require a return URL. If you don't provide one, we can't present payment methods that require a return URL to your users, even if you've enabled them.

To provide a return URL:

Register a custom URL. Universal links aren't supported.

Configure your custom URL.

Set up your root component to forward the URL to the Stripe SDK as shown below.

If you're using Expo, set your scheme in the app.json file.

app.json

```
import { useEffect, useCallback } from 'react';
import { Linking } from 'react-native';
import { useStripe } from '@stripe/stripe-react-native';

export default function MyApp() {
  const { handleURLCallback } = useStripe();

  const handleDeepLink = useCallback(
    async (url: string | null) => {
      if (url) {
        const stripeHandled = await handleURLCallback(url);
        if (stripeHandled) {
          // This was a Stripe URL - you can return or add extra handling here as you see fit
        } else {
          // This was NOT a Stripe URL – handle as you normally would
        }
      }
    },
    [handleURLCallback]
  );

  useEffect(() => {
    const getUrlAsync = async () => {
      const initialUrl = await Linking.getInitialURL();
      handleDeepLink(initialUrl);
    };

    getUrlAsync();

    const deepLinkListener = Linking.addEventListener(
      'url',
      (event: { url: string }) => {
        handleDeepLink(event.url);
      }
    );

    return () => deepLinkListener.remove();
  }, [handleDeepLink]);

  return (
    <View>
      <AwesomeAppComponent />
    </View>
  );
}
```

```
import { useEffect, useCallback } from 'react';
import { Linking } from 'react-native';
import { useStripe } from '@stripe/stripe-react-native';

export default function MyApp() {
  const { handleURLCallback } = useStripe();

  const handleDeepLink = useCallback(
    async (url: string | null) => {
      if (url) {
        const stripeHandled = await handleURLCallback(url);
        if (stripeHandled) {
          // This was a Stripe URL - you can return or add extra handling here as you see fit
        } else {
```

```
        // This was NOT a Stripe URL – handle as you normally would
      }
    }
  },
  [handleURLCallback]
);

useEffect(() => {
  const getUrlAsync = async () => {
    const initialUrl = await Linking.getInitialURL();
    handleDeepLink(initialUrl);
  };

  getUrlAsync();

  const deepLinkListener = Linking.addEventListener(
    'url',
    (event: { url: string }) => {
      handleDeepLink(event.url);
    }
  );

  return () => deepLinkListener.remove();
}, [handleDeepLink]);

return (
  <View>
    <AwesomeAppComponent />
  </View>
);
}
```

Additionally, set the returnURL when you call the initPaymentSheet method:

returnURL

initPaymentSheet

```
await initPaymentSheet({
  ...
  returnURL: 'your-app://stripe-redirect',
  ...
});
```

```
await initPaymentSheet({
  ...
  returnURL: 'your-app://stripe-redirect',
  ...
});
```

For more information on native URL schemes, refer to the Android and iOS docs.

Handle post-payment events

Stripe sends a payment_intent.succeeded event when the payment completes. Use the Dashboard webhook tool or follow the webhook guide to receive these events and run actions, such as sending an order confirmation email to your customer, logging the sale in a database, or starting a shipping workflow.

Listen for these events rather than waiting on a callback from the client. On the client, the customer could close the browser window or quit the app before the callback executes, and malicious clients could manipulate the response. Setting up your integration to listen for asynchronous events is what enables you to accept different types of payment methods with a single integration.

In addition to handling the payment_intent.succeeded event, we recommend handling these other events when collecting payments with the Payment Element:

payment_intent.succeeded

payment_intent.succeeded

payment_intent.payment_failed

processing

payment_failed

Test the integration

insufficient_funds

See Testing for additional information to test your integration.

Optional
Enable Link

Optional
Enable Apple Pay

Optional
Enable Google Pay

Optional
Enable card scanning (iOS only)
Client-side

Optional
Customize the sheet
Client-side

Optional
Handle user logout

Optional
Complete payment in your UI

Need help? Contact Support.

Check out our changelog.

Questions? Contact Sales.

LLM? Read llms.txt.

Powered by Markdoc

Welcome to the Stripe Shell!

Stripe Shell is a browser-based shell with the Stripe CLI pre-installed. Log in to your
Stripe account and press Control + Backtick (`) on your keyboard to start managing your Stripe
resources in test mode.

- View supported Stripe commands:

stripe help %¶þ

- Find webhook events:
stripe trigger %¶þ [event]

- Listen for webhook events:
stripe listen %¶

- Call Stripe APIs: stripe [api resource] [operation] (e.g.,
stripe customers list %¶þ
)

Welcome to the Stripe Shell!

Stripe Shell is a browser-based shell with the Stripe CLI pre-installed. Log in to your
Stripe account and press Control + Backtick (`) on your keyboard to start managing your Stripe
resources in test mode.

- View supported Stripe commands:
stripe help %¶þ

- Find webhook events:
stripe trigger %¶þ [event]

- Listen for webhook events:
stripe listen %¶

- Call Stripe APIs: stripe [api resource] [operation] (e.g.,
stripe customers list %¶þ
)

stripe help %¶þ

stripe trigger %¶þ [event]

stripe listen %¶

stripe customers list %¶þ

$

$

Firebase Cloud Messaging

Overview

Fundamentals

AI

Build

Run

Reference

Samples

Build

Get to market quickly and securely with products that can scale globally

Go to Build

Build Products

App Check

App Hosting

Authentication

Cloud Functions

Cloud Storage

Data Connect

Extensions

Firebase ML

Firestore

Genkit

Hosting

Realtime Database

Firebase AI Logic client SDKs

Generative AI

Run

Run your app with confidence and deliver the best experience for your users

Go to Run

Run Products

N-e‡ – ~AšÔ

eåg,Šž

Õ\-mÅ´

Documentation

FCM

Add Firebase - Apple platforms (iOS+)

Add Firebase - Android

Add Firebase - Web

Add Firebase - Flutter

Add Firebase - C++

Add Firebase - Unity

Add Firebase - Server environments

Manage Firebase projects

Supported platforms & frameworks

Use Emulator Suite

AI-assisted development

Develop with AI assistance

Overview

Firebase Studio

Build and ship full-stack AI-infused apps right from your browser.

Gemini in Firebase

Streamline development with an AI-powered assistant in Firebase interfaces and tools.

MCP, Gemini CLI, & agents

Access agentive development tools, like our MCP server and Gemini CLI extension.

Build AI-powered apps

Firebase AI Logic

Build AI-powered mobile and web apps and features with the Gemini and Imagen models using Firebase

AI Logic.

Genkit

Build full-stack AI-powered applications using this open-source framework.

Emulator Suite

Authentication

App Check

Data Connect

Firestore

Realtime Database

Storage

Security Rules

App Hosting

Hosting

Cloud Functions

Extensions

ML

Test Lab

App Distribution

Crashlytics

Performance Monitoring

Remote Config

A/B Testing

Analytics

Cloud Messaging

In-App Messaging

Dynamic Links

Google AdMob

Google Ads

Build

More

More

Run

More

More

Solutions

Pricing

Docs

More

Overview

Fundamentals

More

AI

More

Build

More

Run

More

Reference

Samples

More

Overview

Fundamentals

More

More

AI

More

More

Build

More

More

Run

More

More

Reference

Samples

Community

More

More

Support

Blog

Studio

Log events

Set user properties

Introduction

FCM Architectural Overview

Get started

Sending Messages to Devices

Setting Up Your Server Environment

Sending a Message

Using FCM v1 API

Using the Admin SDK

Using the Firebase Console

Receiving Messages

Customizing Message Behavior

Set message type

Set message priority

Set message lifespan

Non-collapsible and collapsible message types

Modify message behavior

Customize messages

Google AdMob

Introduction

Get started for iOS

Get started for Android

Use Analytics and Firebase with AdMob apps

Use AdMob in a game

C++
    Get startedAdMob Migration Guide

Get started

AdMob Migration Guide

Unity
    Get started

Get started

Solutions

Test adoption of new ad formats
    Solution overviewSolution tutorial

Solution overview

Solution tutorial

Optimize ad frequency
    Solution overviewSolution tutorial

Solution overview

Solution tutorial

Optimize hybrid monetization
    Solution tutorial

Solution tutorial

Google Ads

Introduction

Solutions

Measure iOS Ads conversions
    Using iOS on-device conversion measurement solutionsFirst-party solution tutorialEvent-driven solution tutorial

Using iOS on-device conversion measurement solutions

Build

Go to Build

Build Products

App Check

App Hosting

Authentication

Cloud Functions

Cloud Storage

Data Connect

Extensions

Firebase ML

Firestore

Genkit

Hosting

Realtime Database

Firebase AI Logic client SDKs

Generative AI

Run

Go to Run

Run Products

A/B Testing

App Distribution

Cloud Messaging

Crashlytics

Google Analytics

In-App Messaging

Performance Monitoring

Remote Config

Test Lab

Overview

Fundamentals

AI

Build

Run

Reference

Samples

Learn

Events

Stories

Add Firebase - Apple platforms (iOS+)

Add Firebase - Android

Add Firebase - Web

Add Firebase - Flutter

Add Firebase - C++

Add Firebase - Unity

Add Firebase - Server environments

Manage Firebase projects

Supported platforms & frameworks

Use Emulator Suite

AI-assisted development

Develop with AI assistance

Overview

Firebase Studio

Gemini in Firebase

MCP, Gemini CLI, & agents

Build AI-powered apps

Firebase AI Logic

Genkit

Emulator Suite

Authentication

App Check

Key capabilities

How does it work?

Firebase Cloud Messaging
bookmark_border

Firebase Cloud Messaging (FCM) is a cross-platform messaging solution that lets you reliably send messages.

Ready to get started? Choose your platform:

iOS+ Android Web Flutter

Unity C++

Key capabilities

How does it work?

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

A trusted environment such as Cloud Functions for Firebase or an app server on which to build, target, and send messages.

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.

See the architectural overview for more detail and important information about the components of FCM.

Implementation path

Next steps

Follow the Get started guide to set up your client apps and learn to send messages with FCM.

Follow the Get started guide to set up your client apps and learn to send messages with FCM.

Run the Android or iOS Quickstart sample.

Run the Android or iOS Quickstart sample.

Learn how to receive messages in your client app.

Learn how to receive messages in your client app.

Set up your server environment to build and send message requests. You can write sending logic using the Admin SDK or the FCM v1 API.

Set up your server environment to build and send message requests. You can write sending logic using the Admin SDK or the FCM v1 API.

Explore advanced features, such as targeting groups with topic messaging, and learn how to understand message delivery with the FCM Data API and BigQuery export.

Explore advanced features, such as targeting groups with topic messaging, and learn how to understand message delivery with the FCM Data API and BigQuery export.

Learn more about FCM in the architecture overview and review best practices for sending messages at scale and managing registration tokens.

Learn more about FCM in the architecture overview and review best practices for sending messages at scale and managing registration tokens.

Last updated 2025-10-14 UTC.

Learn
Developer guides
SDK & API reference
Samples
Libraries
GitHub

Learn

Developer guides

SDK & API reference

Samples

Libraries

GitHub

Stay connected
Check out the blog
Find us on Reddit
Follow on X
Subscribe on YouTube
Attend an event

Stay connected

Check out the blog

Find us on Reddit

Follow on X

Subscribe on YouTube

Attend an event

Support
Contact support
Stack Overflow
Slack community
Google group
Release notes
Brand guidelines
FAQs

Support

Contact support

Stack Overflow

Slack community

Google group

Release notes

Brand guidelines

FAQs

Android

Chrome

Firebase

Google Cloud Platform

All products

Terms

Privacy

Manage cookies

English

Deutsch

Español – América Latina

Français

Indonesia

Italiano

Polski

Português – Brasil

Tiê0 ær Viê27@

Türkçe

B CD AC¤8C•

^%Ñ^…Ù^

bvDc–1b†Je)

d 'c 3lÀ

“™?•)&”

šÉ¾˜)²›à

â 2âž2äN â

N-e‡ – {€OS

N-e‡ – ~AšÔ

eåg,Šž

Õ\-mÅ´

ZKTeco Biometric Web API Guide

Oops, the page got lost!