“EZBus Flutter Apps” Documentation

“Flutter Apps”

Table of Contents

A) What is the EZBus App? - [top (#toc)](#toc)

EZBus is a transportation management system that is committed to enabling cities, individuals, and businesses to move anywhere. EzBus is tech-driven, fancy, and convenient.

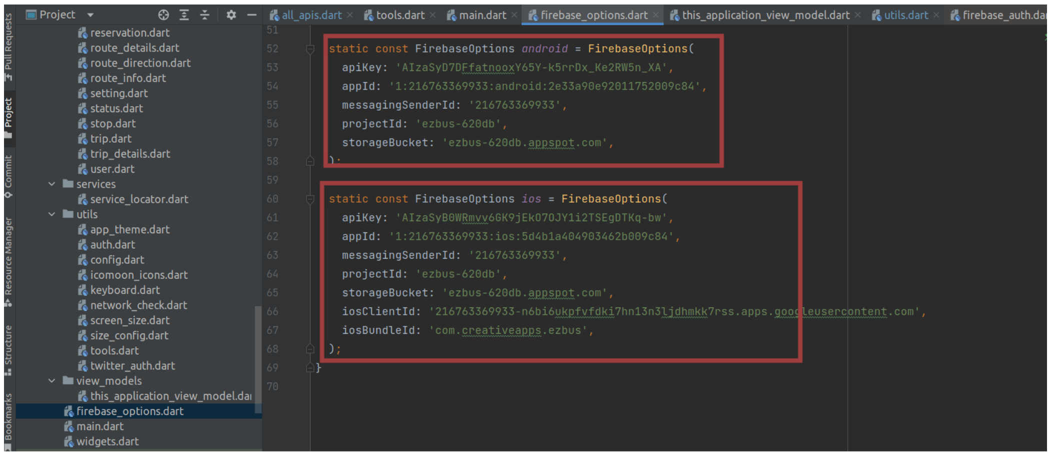
EzBus consists of two Flutter apps for customers and drivers along with an Admin panel with a Laravel API system as a back-end and a Vue.js front-end system as a web interface.

Here we present how you can generate both the **Android and iOS mobile apps** from the purchased source code.

B) Configuration - [top (#toc)](#toc)

1. **Firebase**

   - Go to [Firebase Console (https://console.firebase.google.com/)](https://console.firebase.google.com/) and in Project settings, click on Add App
   - Choose Android and enter the package name of the app. The package name can be found in the file android/app/build.gradle
   - Download the google-services.json file and place it in the path "android/app"
   - Edit the firebase_options file in lib/firebase_options.dart by your changing the FirebaseOptions by yours that can be obtained from the downloaded google-services.json file



```
static const FirebaseOptions android = FirebaseOptions(
  apiKey: 'AIzaSyD7DFfatnooxY65Y-k5rrDx_Ke2RW5n_XA',
  appId: '1:216763369933:android:2e33a90e92011752009c84',
  messagingSenderId: '216763369933',
  projectId: 'ezbus-620db',
  storageBucket: 'ezbus-620db.appspot.com',
);

static const FirebaseOptions ios = FirebaseOptions(
  apiKey: 'AIzaSyB0WRmvv6GK9jEkO7OJY1i2TSEgDTKq-bw',
  appId: '1:216763369933:ios:5d4b1a404903462b009c84',
  messagingSenderId: '216763369933',
  projectId: 'ezbus-620db',
  storageBucket: 'ezbus-620db.appspot.com',
  iosClientId: '216763369933-n6bi6ukpfvfdki7hn13n3ljdhmkk7rss.apps.googleusercontent.com',
  iosBundleId: 'com.creativeapps.ezbus',
);
}
```

   - Repeat the same steps for iOS app by choosing iOS instead of Android in the first step.
   - Please note that some terms may be different than the ones in the google-services.json file. For example, the mobilesdk_app_id in the google-services.json file is called "appId" in the FirebaseOptions class. Another example is the project_number in the google-services.json file is called "messagingSenderId" in the FirebaseOptions class.

2. **Google Maps on Android**

   Edit the manifest file in android/app/src/main/AndroidManifest.xml by adding your Google maps API key in the following line

```
com.google.android.geo.API_KEY
```

3. **Google Maps on IOS**

   Replace YOUR-API-KEY with your Google maps API key in ios/Runner/AppDelegate.swift

   ```
   GMSServices.provideAPIKey("YOUR-API-KEY")
   ```

4. **Google Maps, server url and splash screen configuration in Flutter**

   Edit the file lib/utils/config.dart to configure the Flutter app before building.

   - Backend URL: change the following line with your backend URL

     ```
     static String webUrl = "ezbus.creativeapps.info/backend";
     ```

   - Google Api Key: change the following line with your Google maps API key

     ```
     static String googleApikey = "AIzaSy......";
     ```

   - Socket Server URL and port number: change the following lines

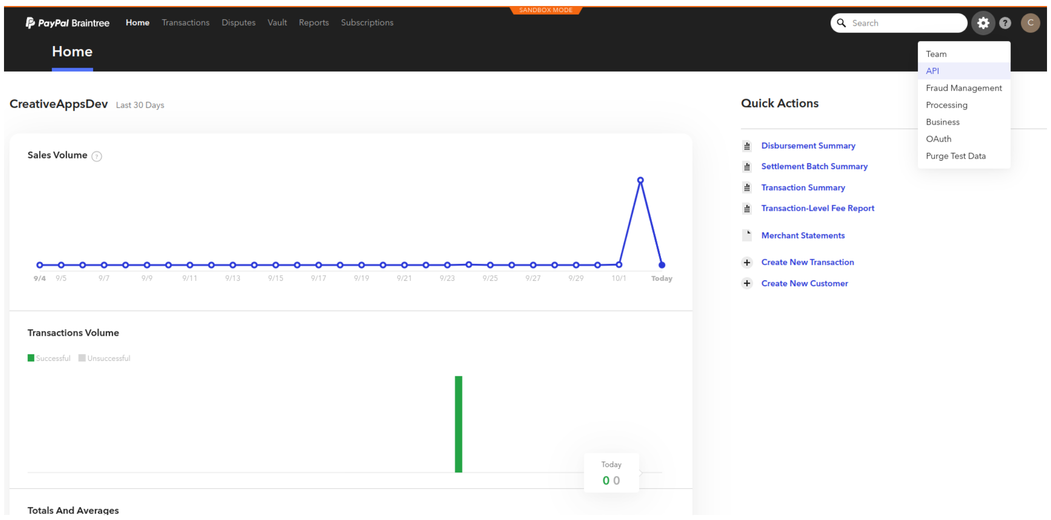
     ```
     static String socketUrl = "YourSocketUrl";
     static String socketPort = 'YourSocketPort';
     ```

5. **Payment gateways configuration in Flutter**

   The system has four types of payments. They are as follows:
   1. Braintree
   2. RazorPay
   3. FlutterWave
   4. PayTabs
   5. No payment (to allow free rides). **If you need this option, skip configuring any payment gateway in the following.**

   1) Braintree

   1. Go to your account in https://www.braintreegateway.com or https://sandbox.braintreegateway.com
   2. Go to "Settings" from the upper right menu, then click on "API"



   3. Click on "Generate New Tokenization Key"
   4. Edit the file lib/utils/config.dart to set the variable braintreeTokenizationKey with the generated tokenization key.

   2) RazorPay

   1. Open your "RazorPay" account dashboard
   2. Choose "Settings" section in the left pane, then click on "API Keys"
   3. Get the "Key Id".
   4. Edit the file lib/utils/config.dart to set the variable razorpayKey with the "Key Id".

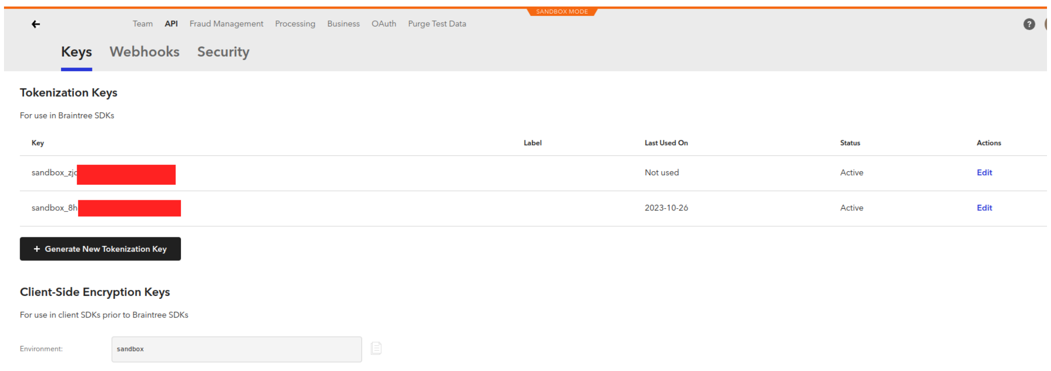
   3) FlutterWave

   1. Open your "Flutterwave" account dashboard
   2. Choose "Settings" section in the left pane, then click on "API"
   3. Get the "Public Key".
   4. Edit the file lib/utils/config.dart to set the variable flutterwaveKey with the "Public Key".

   4) PayTabs

   1. Open your "Paytabs" account dashboard
   2. Choose "Developers" section in the left pane, then click on "Key management"

3. Click on "+" to generate new "API Key". **Select Mobile SDK.**
4. Get the "Server Key" and "Client Key".
5. Obtain your "Profile Id" (the number before your name)
6. Edit the file lib/utils/config.dart to set the variable paytabsProfileId with the "Profile Id" and the variable paytabsServerKey with the "Server Key" and the variable paytabsClientKey with the "Client Key" and



Tokenization Keys

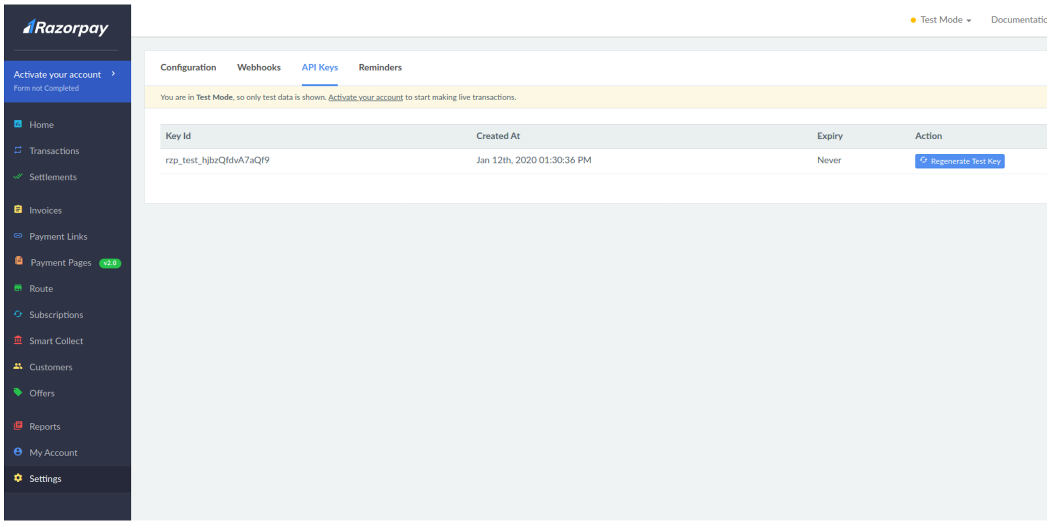
For use in Braintree SDKs

Key	Label	Last Used On	Status	Actions
sandbox_zjc[redacted]		Not used	Active	Edit
sandbox_8h[redacted]		2023-10-26	Active	Edit

Client-Side Encryption Keys

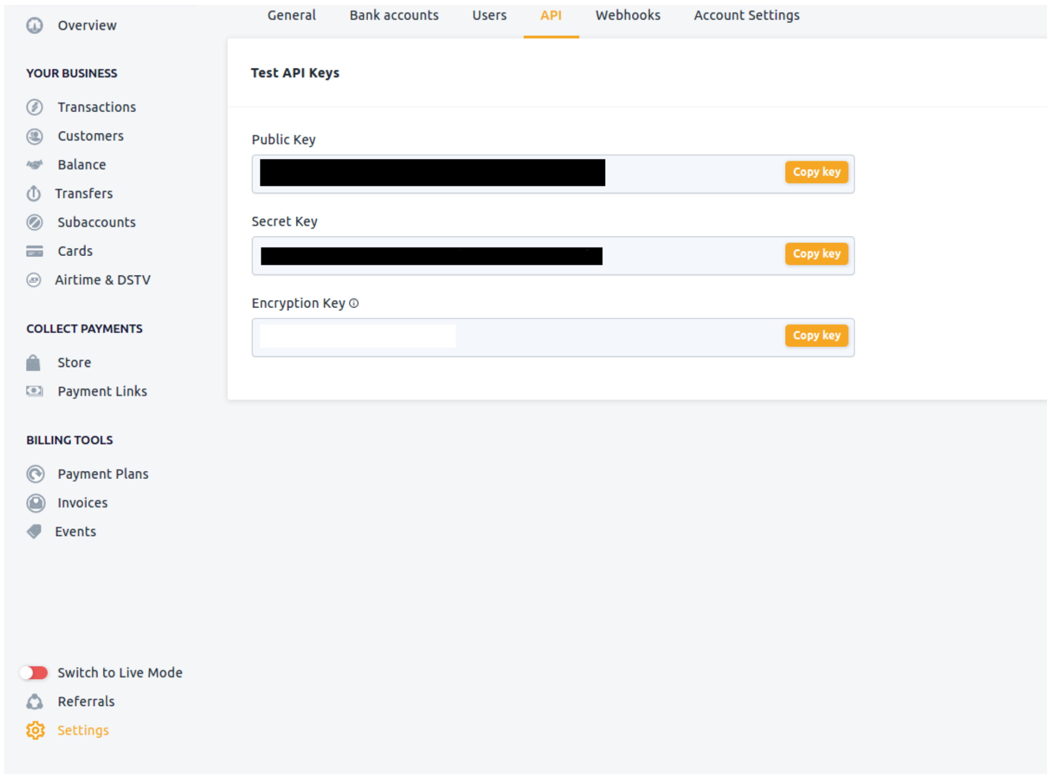
For use in client SDKs prior to Braintree SDKs

Environment: sandbox



You are in **Test Mode**, so only test data is shown. Activate your account to start making live transactions.

Key Id	Created At	Expiry	Action
rzp_test_hjbzQfdvA7aQf9	Jan 12th, 2020 01:30:36 PM	Never	Regenerate Test Key



Test API Keys

Public Key: [redacted]

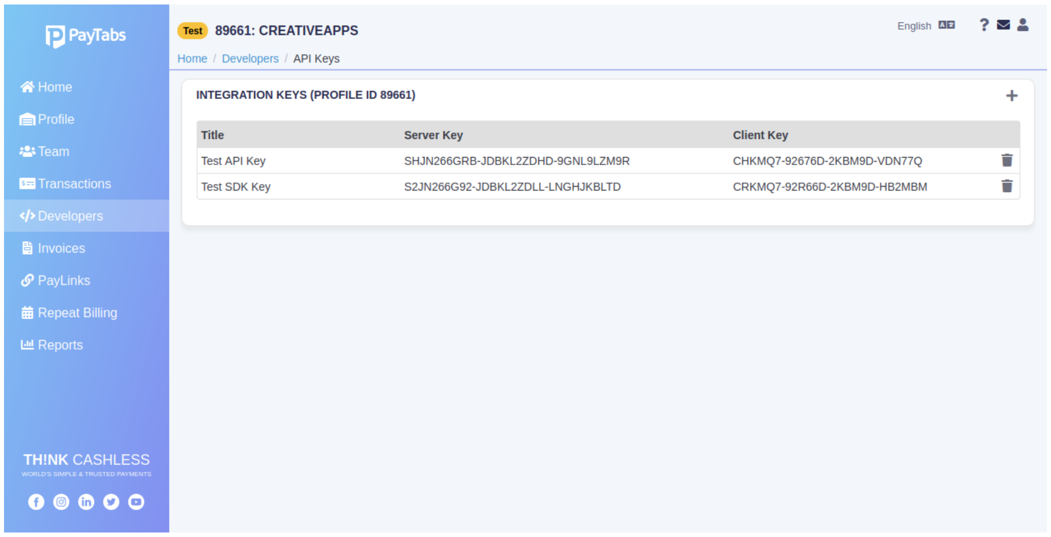
Secret Key: [redacted]

Encryption Key

paytabsMerchantCountryCode with your 2 chars iso country code (as 'US').

6. **Build flutter**

```
flutter pub get
```

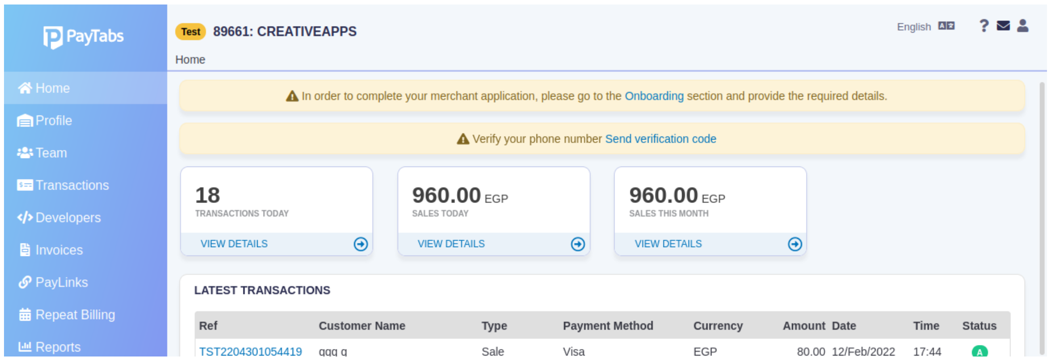


Test 89661: CREATIVEAPPS

Home / Developers / API Keys

INTEGRATION KEYS (PROFILE ID 89661)

Title	Server Key	Client Key
Test API Key	SHJN266GRB-JDBKL2ZDHD-9GNL9LZM9R	CHKMQ7-92676D-2KBM9D-VDN77Q
Test SDK Key	S2JN266G92-JDBKL2ZDLL-LNGHJKBLTD	CRKMQ7-92R66D-2KBM9D-HB2MBM



Test 89661: CREATIVEAPPS

Home

In order to complete your merchant application, please go to the Onboarding section and provide the required details.

Verify your phone number Send verification code

18 TRANSACTIONS TODAY

960.00 EGP SALES TODAY

960.00 EGP SALES THIS MONTH

LATEST TRANSACTIONS

Ref	Customer Name	Type	Payment Method	Currency	Amount	Date	Time	Status
TST2204301054419	qqq q	Sale	Visa	EGP	80.00	12/Feb/2022	17:44	A

C) Android App - top (#toc)

In order to build the android app correctly, please make sure that you have installed the latest **Android Studio IDE. (https://developer.android.com/studio)** Please install Android Studio 3.5 or above.

Prepare for release

Before you can publish your Flutter app to Google Play, the app needs a digital signature.

If you don’t already have a keystore, create one

On Mac, use the following command:

```
keytool -genkey -v -keystore ~/key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias key
```

On Windows, use the following command:

```
keytool -genkey -v -keystore c:/Users/USER_NAME/key.jks -storetype JKS -keyalg RSA -keysize 2048 -validity 10000 -alias key
```

Create a file named **/android/key.properties** that will reference your keystore, it will look like this:

```
storePassword= keyPassword= keyAlias=key
storeFile=/key.jks>
```

Configure signing in Gradle

You will find your Gradle file at **/android/app/build.gradle**. Start editing and follow these steps:

Add the keystore information that we just created before "android{":

```
def keystoreProperties = new Properties()
def keystorePropertiesFile = rootProject.file('key.properties')
if (keystorePropertiesFile.exists()) {
keystoreProperties.load(new FileInputStream(keystorePropertiesFile))
}

android {
```

Then, replace:

```
content_copy
buildTypes {
release {
// TODO: Add your own signing config for the release build.
// Signing with the debug keys for now,
// so `flutter run --release` works.
signingConfig signingConfigs.debug
}
}
```

With the signing configuration info:

```
content_copy
signingConfigs {
release {
keyAlias keystoreProperties['keyAlias']
keyPassword keystoreProperties['keyPassword']
storeFile keystoreProperties['storeFile'] ? file(keystoreProperties['storeFile']) : null
storePassword keystoreProperties['storePassword']
}
}
buildTypes {
release {
signingConfig signingConfigs.release
}
}
```

- Then, go to the **defaultConfig** block.
- Enter a final unique **applicationId**. For example, "com.ezbus.driver"
- Give your app a **versionName** and **versionCode**.

You have now configured your app’s Gradle file and your release builds will be signed automatically.

Build and release the app

Now you’ll build your app’s APK so it can be uploaded to the Google Play Store. Go to your command line and follow these steps:

1. Enter `cd`
2. Run `flutter build apk`

If everything goes well, you will have an APK ready at **/build/app/outputs/apk/release/app.apk**

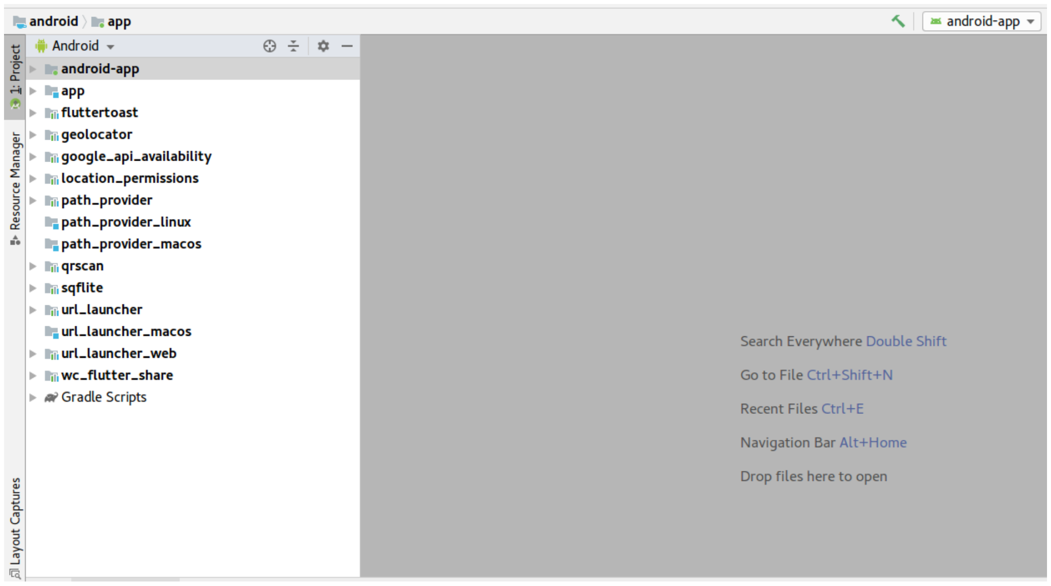
Publish to the Google Play Store

Your app is ready! Next, follow **this link to uploading an APK to the Google Play Store (https://support.google.com/googleplay/android-developer/answer/113469?hl=en)**.

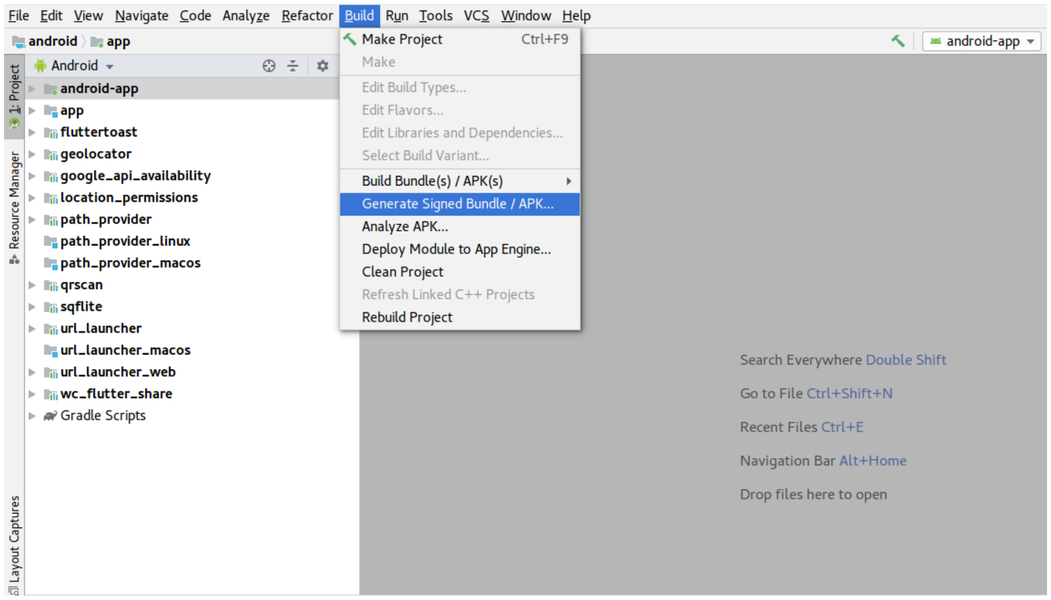
Alternative way: Generate APK from Android Studio

You can also generate the APK directly from the Android Studio by opening the android project inside the purchased folder.

1. Go to file-> open and go to the location "FlutterApp/android" of the purchased "FlutterApp/android" folder



2. Go to build->Generate signed Bundle/APK. Here I will show you how to generate APK.
3. You can use an existing key
4. Or generate a new key by click "Create new" amd filling the following form
5. Then choose the path that you want to save the generated APK in. Make sure that V1 and V2 checkboxes are checked.

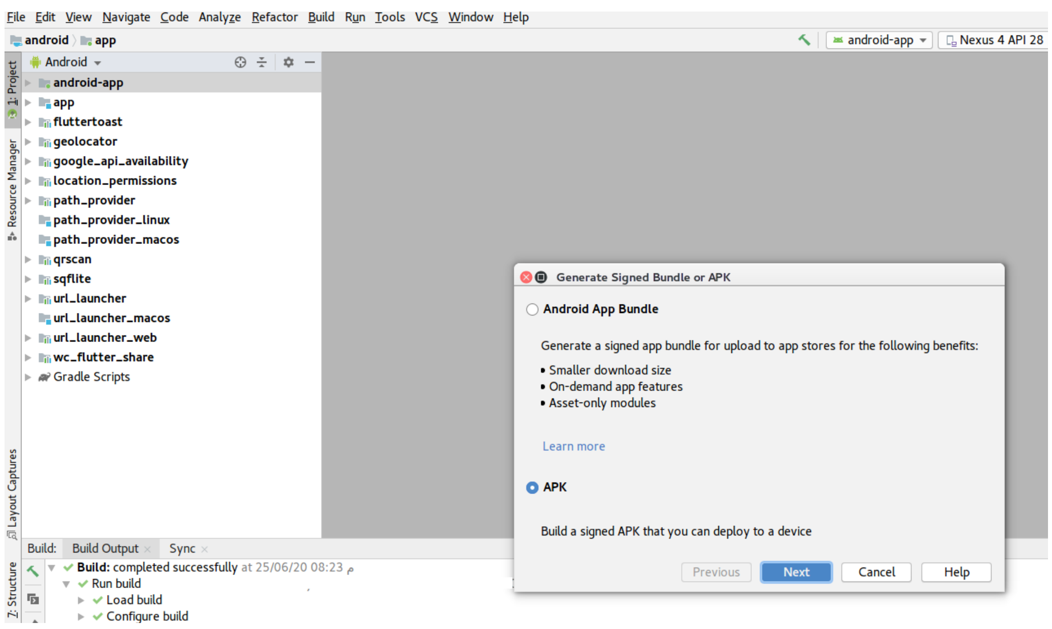


D) iOS App - [top (#toc)](#toc)

In order to build the iOS app please make sure that you have XCode installed. Please install XCode 11.3 or above, then open it and follow the following steps

Prerequisites

- Make sure that you've covered Apple's **guidelines** (https://developer.apple.com/appstore/resources/approval/guidelines.html) for releasing an app on the app store.
- Have your app's icons and launch screens ready.
- Have an Apple Developer account.



Generate Signed Bundle or APK

Android App Bundle

Generate a signed app bundle for upload to app stores for the following benefits:

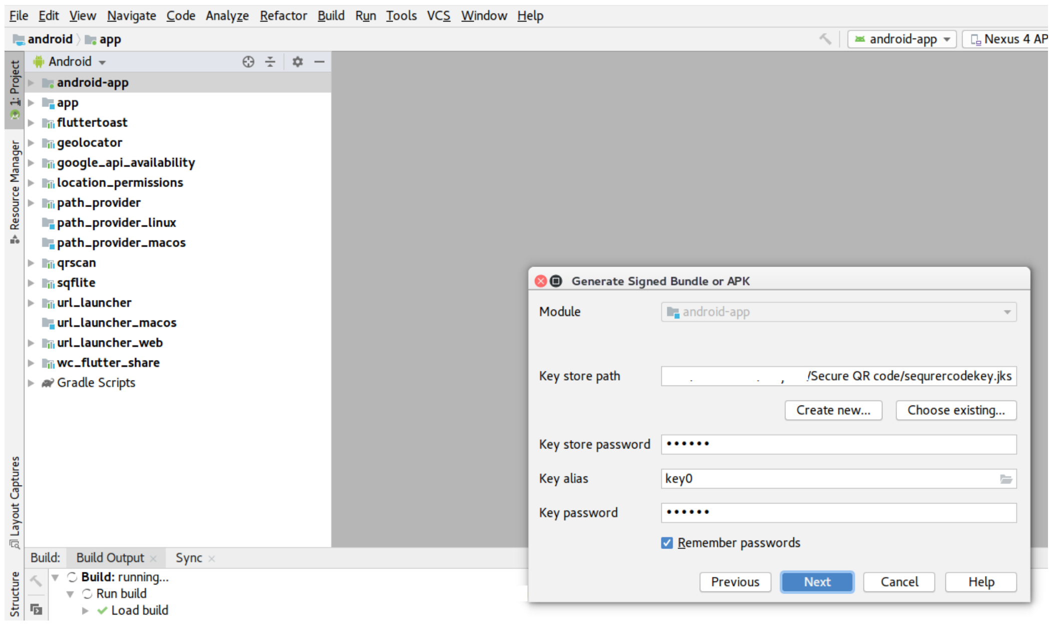
- Smaller download size
- On-demand app features
- Asset-only modules

Learn more

APK

Build a signed APK that you can deploy to a device

```
Build: completed successfully at 25/06/20 08:23
  Run build
    Load build
    Configure build
```



Generate Signed Bundle or APK

Module: android-app

Key store path: /Secure QR code/sequrercodekey.jks

Key store password: ••••••

Key alias: key0

Key password: ••••••

[✓] Remember passwords

```
Build: running...
  Run build
    Load build
```

Prepare for building

Please run this command in the the project directory

```
flutter build ios
```

Before you can build and release your app on the App Store, you need to set up a place for it using App Store Connect. But first, you need to register a unique bundle ID for your app. This can be done by logging into your Apple Developer account and following these steps:

1. Open the **App IDs** page.
2. Click **+** to create a new **Bundle ID**.
3. Fill out the needed information: App Name, and Explicit App ID.
4. If your app needs specific services, select them and click **Continue**.
5. Review the details and click **Register** to finish.

Now that we have a unique bundle ID, it's time to set up a place for your app on the App Store Connect. Log in to the App Store Connect.

1. Select **My Apps**.
2. Click **+** then select **New App**.
3. Fill in your app details and make sure **iOS** is selected, then click **Create**.
4. From the sidebar, select **App Information**.
5. In the General Information section, select the **Bundle ID** that you registered above.

Adjust Xcode project settings for release

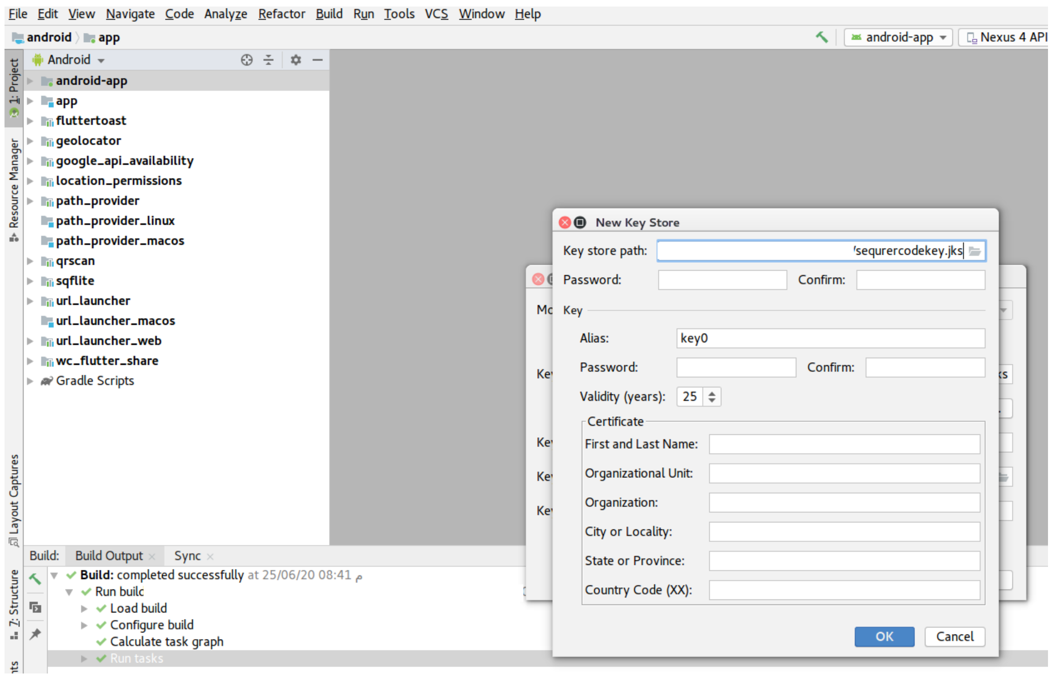
You've set everything up from Apple's side, and next you'll adjust your Xcode project's settings to prepare your app for release. Go ahead and fire up Xcode.

1. Open **Runner.xcworkspace** that is inside your app's iOS folder at "FlutterApp/ios".
2. From the Xcode project navigator, select the **Runner** project.
3. Then, select the **Runner target** in the main view sidebar.
4. Go to the **General** tab.
5. In the **Identity** section, fill out the information and make sure the **Bundle Identifier** is the one registered on App Store Connect.
6. In the **Signing** section, make sure **Automatically manage signing** is checked and select your **team**.
7. Fill out the rest of the information as needed.
8. Next, you'll update your app's icon. This can be done by selecting **Assets.xcassets** in the Runner folder from Xcode's project navigator.

Build and upload your app

At this point, all the settings have been updated for release and there is a placeholder ready on App Store Connect, which means you can build and release.

1. From the command line, run `flutter build ios`
2. Then go back to Xcode and reopen **Runner.xcworkspace**
3. Select **Product** -> **Scheme** -> **Runner**.
4. Select **Product** -> **Destination** -> **Generic iOS Device**.


New Key Store

Key store path: 'sequrercodekey.jks

Password: Confirm:

Key

Alias: key0

Password: Confirm:

Validity (years): 25

Certificate

First and Last Name:

Organizational Unit:

Organization:

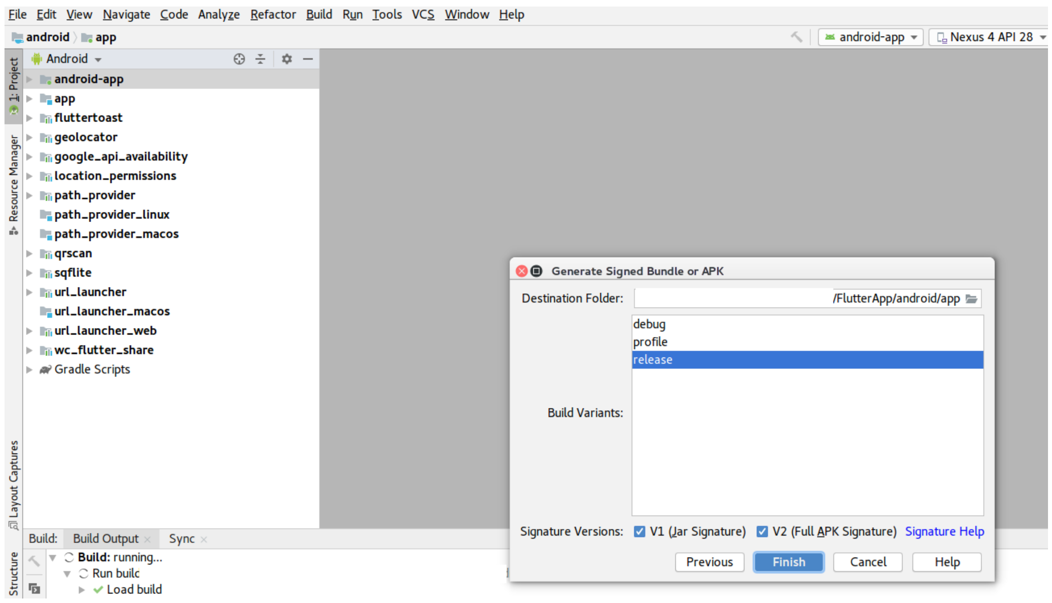
City or Locality:

State or Province:

Country Code (XX):

OK Cancel

```
Build: completed successfully at 25/06/20 08:41
  Run build
    Load build
    Configure build
    Calculate task graph
    Run tasks
```


Generate Signed Bundle or APK

Destination Folder: /FlutterApp/android/app

Build Variants: debug, profile, [✓ release]

Signature Versions: [✓] V1 (Jar Signature) [✓] V2 (Full APK Signature) Signature Help

Previous Finish Cancel Help

```
Build: running...
  Run build
    Load build
```

5. Select **Product** -> **Archive** to produce a build archive.
6. From the Xcode **Organizer** window, select your iOS app from the sidebar, then select the build archive you just produced.
7. Click the **Validate…** button to build.
8. Once the archive is successfully validated, click **Upload to App Store.…**

Back on App Store Connect, check the status of your build from the **Activities** tab. Once it's ready to release:

1. Go to **Pricing and Availability** and fill out the required information.
2. From the sidebar, select the **status**.
3. Select **Prepare for Submission** and complete all required fields.
4. Click **Submit for Review**.

That's it! Your app will now be uploaded to the App Store. Apple will review your app before releasing and keep you updated on the status of your app.

E) FAQ - top (#toc)

1. How to change the theme of the app?

   Edit the app_theme.dart file in the path "FlutterApp/lib/app_theme.dart". The file contains all used colors in the app and font styles of the text.

2. How to change the launcher icon for Android and iOS?

   For Android, go to "FlutterApp/android/app/src/main/res" folder. You should see some folders beginning with mipmap-* which represent different pixel densities. Replace launcher_icon.png file in each folders to use custom icons.

   For iOS, the icon config and files are located inside "FlutterApp/ios/Runner/Assets.xcassets/AppIcon.appiconset". The Contents.json file defines a list of icon images with different sizes and scales. The files are located in the same folder. It's recommended to follow the guideline for designing iOS icon

Once again, thank you so much for purchasing our EZBus App. As I said at the beginning, I'd be glad to help you if you have any questions. I'll do my best to assist. If you have a more general question relating to the items on CodeCanyon, you might consider visiting the forums and asking your question in the "Item

Discussion" section.

CreativeAppsDev Team

Go To Table of Contents (#toc)