IOS and android App publish

https://medium.com/@uristrimber/how-to-submit-your-flutter-app-to-the-app-store-in-2022-c9e7b59fa3ad

1. android

<https://medium.com/@uristrimber/how-to-submit-your-flutter-app-to-the-app-store-in-2022-c9e7b59fa3ad>

step 1:

create the Launcher icon for app

<https://easyappicon.com/>

Once you have downloaded the icon package, unzip the .zip and place the contents of the android folder in your project, inside android/app/src/main/res.

<application…android:icon=”@mipmap/ic\_launcher”android:roundIcon=”@mipmap/ic\_launcher\_round”…>

</application>

Application id is unique so we can findl application id in app > build

defaultConfig **{** // *TODO: Specify your own unique Application ID (https://developer.android.com/studio/build/application-id.html).* applicationId "com.example.firebaseflutterproject"  
 // You can update the following values to match your application needs.  
 // For more information, see: https://docs.flutter.dev/deployment/android#reviewing-the-gradle-build-configuration.  
 minSdkVersion 21  
 targetSdkVersion 33  
 versionCode flutterVersionCode.toInteger()  
 versionName flutterVersionName  
**}**

# Google Play Store

To create a Java keystore in the Google Play Store, which serves as repositories of certificates and private keys, you will use Key and Certificate Management Tool.

As developers, we need to create the upload key while Google handles the app signing key for distribution. Essentially, the keystore is a simple file with a really large block of encrypted data. This file can be stored anywhere on your computer.

To create the keystore, run the following command on your OS terminal:

In Mac/Linux, use:

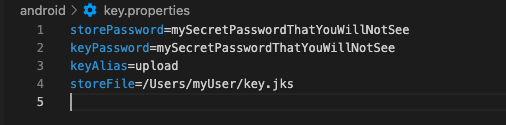
keytool -genkey -v -keystore ~/upload-keystore.jks -keyalg RSA -keysize 2048 -validity 10000 -alias upload

In Windows, use:

keytool -genkey -v -keystore c:\Users\USER\_NAME\upload-keystore.jks -storetype JKS -keyalg RSA -keysize 2048 -validity 10000 -alias upload

Once you have created the *keystore*, you will need to refer to it from the app. To do this, create a *key.properties* file inside the Android folder in your project and configure the file with these lines of code:

storePassword=<password from previous step>keyPassword=<password from previous step>keyAlias=uploadstoreFile=<location of the key store file, such as /Users/<user name>/upload-keystore.jks>



Your next step is to configure the signing in gradle, so the app can handle your upload key in release mode. To do this, go to the build.gradle file located in [project]/android/app/build.gradle.

First add the keystore information from your properties file before the Android block:

def keystoreProperties = new Properties()def keystorePropertiesFile = rootProject.file(‘key.properties’)if (keystorePropertiesFile.exists()) {keystoreProperties.load(new   
 FileInputStream(keystorePropertiesFile))}android {…}

Secondly, find the *buildType* block inside the same file:

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}}

And replace it with the following signing configuration info:

signingConfigs {release {keyAlias keystoreProperties[‘keyAlias’]keyPassword keystoreProperties[‘keyPassword’]storeFile keystoreProperties[‘storeFile’] ?  
 file(keystoreProperties[‘storeFile’]) : nullstorePassword keystoreProperties[‘storePassword’]}}buildTypes {release {signingConfig signingConfigs.release}}

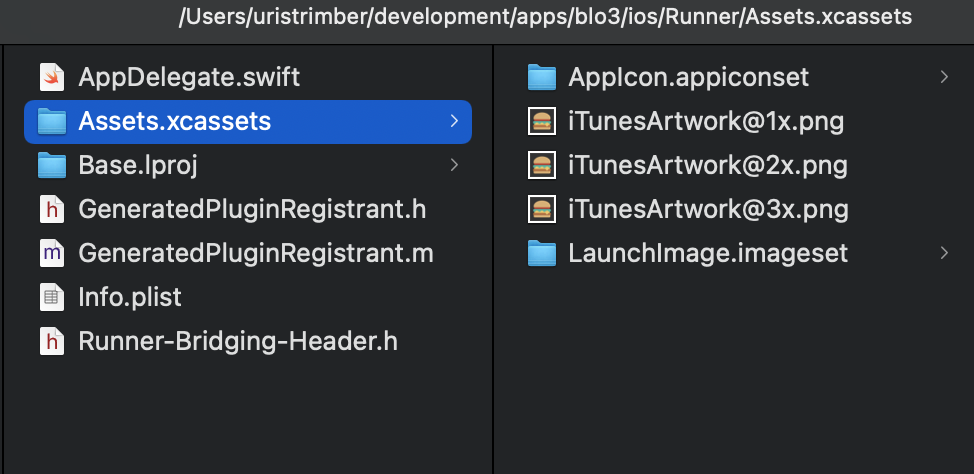
With this last change, release builds of your app will now be signed automatically. From here, you’re almost ready to start the process of uploading the app to Google Play Store, just remember to run a flutter clean command before doing so, due to the fact that you changed the gradle file, and it can mess up the signing process if you have any cached builds in your project.

1. IOS app

step 1:

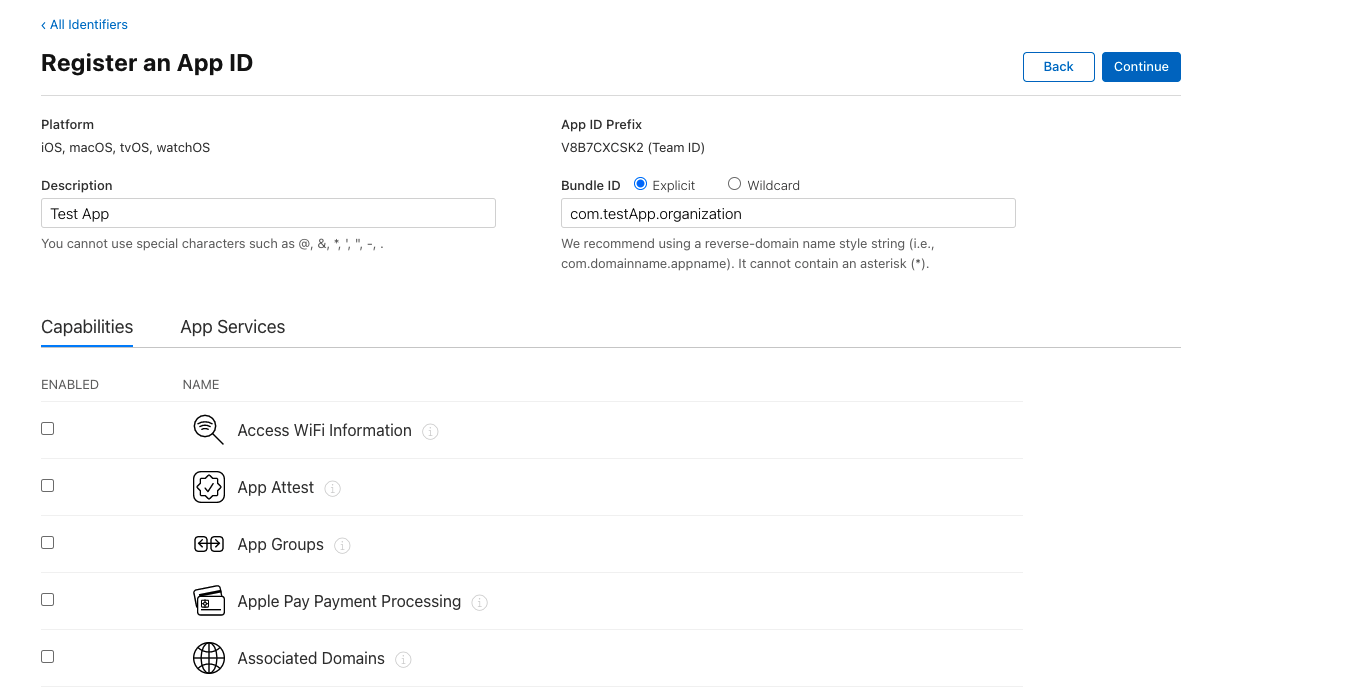
create the Launcher icon for app

For iOS, the process is even simpler: simply place the files from the .zip iOS folder in your project, in the ios/Runner/Assets.xcassets folder.



o begin with, you will need to register a new [identifier](https://developer.apple.com/account/resources/identifiers/add/bundleId" \t "_blank) and create an **App ID.**Create the description that best suits you and an **Explicit**type Bundle ID. The Bundle ID must be unique to the app store. If you select an existing one, a pop up notice will ask you to change it.

Select the capabilities and services that match the ones in your app.



Next: Xcode. To open the project in Xcode from the iOS folder level, head to Runner, select the General tab and change the Display name and Bundle Identifier to the name you prefer.

**App signing and Apple Certificates**

To upload an app to both app stores, you will need to provide it with a digital signature.

To protect your app you will need to have a generated certificate and digital “key” which provides a unique, encrypted and reasonably unhackable signature. This proves that the app came from you, not some other suspicious source.

# Apple Store

Apple signing is quite different from Google’s app signing. It’s based on certificates and distribution profiles, which are handled in Xcode and created at [developer.apple.com](https://developer.apple.com/). These certificates are also needed to test the app on iPhones.

First you will need to create a certificate signing request. To do this, simply open **Keychain Access** located in Utilities in your Mac and select the option to request a certificate from a certificate Authority.

