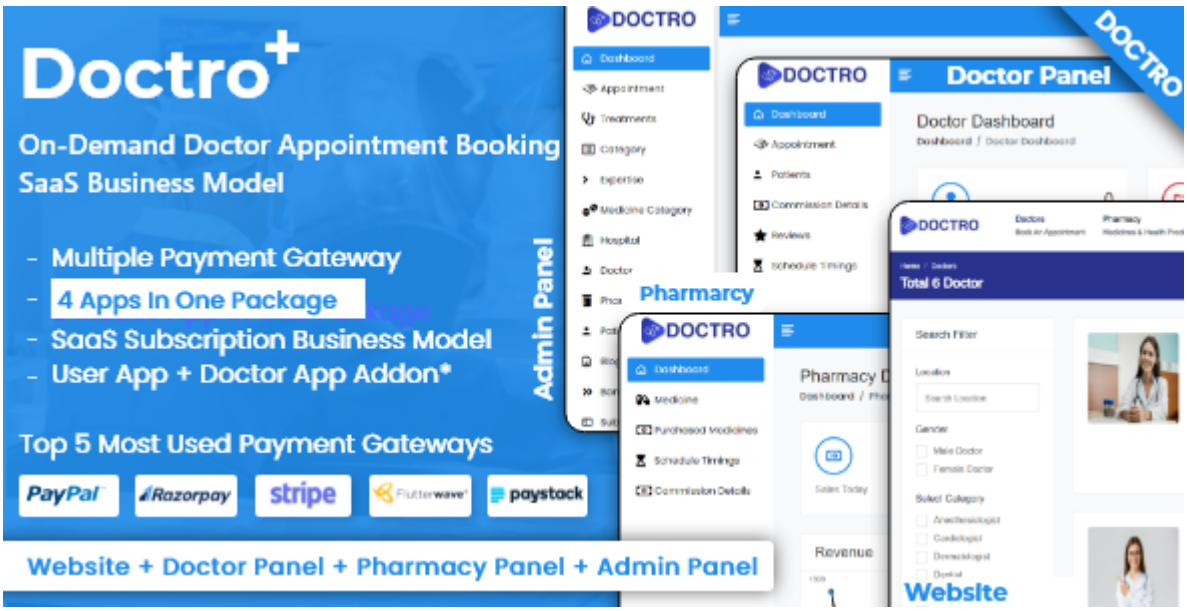
# Doctro - On Demand Doctor Appointment Mobile Apps

# Welcome to Doctro




We are more then happy to assist with any queries you have. this documentation is for basic overview and about how to generate builds using this. Read this document thoroughly if you are experiencing any difficulties.

# Mobile Application Setup

# Flutter installation

To install and run Flutter, your development environment must meet these minimum requirements:

## System requirements

- **Operating Systems**: macOS (64-bit)
- **Disk Space**: 2.8 GB (does not include disk space for IDE/tools).
- **Tools**: Flutter uses `git` for installation and upgrade. We recommend installing Xcode, which includes `git`, but you can also install `git` separately.

## Get the Flutter SDK

1- Download the following installation bundle to get the **3.16.4 stable release** of the Flutter SDK:

```
$ cd ~/development$ unzip ~/Downloads/flutter_macos_v3.16.4-stable.zip
```

2- Extract the file in the desired location, for example:

```
$ export PATH="$PATH:`pwd`/flutter/bin"
```

3- Add the `flutter` tool to your path:

This command sets your `PATH` variable for the *current* terminal window only. To permanently add Flutter to your path, see Update your path.

You are now ready to run Flutter commands!

### Run flutter doctor

Run the following command to see if there are any dependencies you need to install to complete the setup (for verbose output, add the `-v` flag):

This command checks your environment and displays a report to the terminal window. The Dart SDK is bundled with Flutter; it is not necessary to install Dart separately. Check the output carefully for other software you might need to install or further tasks to perform (shown in **bold** text).

For example:

```
[-] Android toolchain - develop for Android devices    • Android SDK at /
```

The following sections describe how to perform these tasks and finish the setup process.

Once you have installed any missing dependencies, run the `flutter doctor` command again to verify that you've set everything up correctly.

## Downloading straight from GitHub instead of using an archive

*This is only suggested for advanced use cases.*

You can also use git directly instead of downloading the prepared archive. For example, to download the stable branch:

```
$ git clone https://github.com/flutter/flutter.git -b stable
```

Update Your Path, and run `flutter doctor`. That will let you know if there are other dependencies you need to install to use Flutter (e.g. the Android SDK).

If you did not use the archive, Flutter will download necessary development binaries as they are needed (if you used the archive, they are included in the download). You may wish to pre-download these development binaries (for example, you may wish to do this when setting up hermetic build environments, or

if you only have intermittent network availability). To do so, run the following command:

For additional download options, see `flutter help precache`.

## Update your path

You can update your PATH variable for the current session at the command line, as shown in Get The Flutter SDK. You'll probably want to update this variable permanently, so you can run `flutter` commands in any terminal session.

The steps for modifying this variable permanently for all terminal sessions are machine-specific. Typically you add a line to a file that is executed whenever you open a new window. For example:

1. Determine the path of your clone of the Flutter SDK. You need this in Step 3.
2. Open (or create) the `rc` file for your shell. Typing `echo $SHELL` in your Terminal tells you which shell you're using. If you're using Bash, edit `$HOME/.bash_profile` or `$HOME/.bashrc`. If you're using Z shell, edit `$HOME/.zshrc`. If you're using a different shell, the file path and filename will be different on your machine.
3. Add the following line and change `[PATH_OF_FLUTTER_GIT_DIRECTORY]` to be the path of your clone of the Flutter git repo:

   ```
   $ export PATH="$PATH:[PATH_OF_FLUTTER_GIT_DIRECTORY]/bin"
   ```

4. Run `source $HOME/.` to refresh the current window, or open a new terminal window to automatically source the file.
5. Verify that the `flutter/bin` directory is now in your PATH by running:

   Verify that the `flutter` command is available by running:

# Platform setup

macOS supports developing Flutter apps in iOS, Android, and the web (technical preview release). Complete at least one of the platform setup steps now, to be able

to build and run your first Flutter app.

# iOS setup

## Install Xcode

To develop Flutter apps for iOS, you need a Mac with XCode installed.

1. Install the latest stable version of XCode (using web download or the Mac App Store).
2. Configure the XCode command-line tools to use the newly-installed version of XCode by running the following from the command line:

   ```
   $ sudo xcode-select --switch /Applications/Xcode.app/Contents/Develope
   ```

   This is the correct path for most cases, when you want to use the latest version of XCode. If you need to use a different version, specify that path instead.
3. Make sure the XCode license agreement is signed by either opening XCode once and confirming or running `sudo xcodebuild -license` from the command line.

Versions older than the latest stable version may still work, but are not recommended for Flutter development. Using old versions of XCode to target bit code is not supported, and is likely not to work.

With XCode, you'll be able to run Flutter apps on an iOS device or on the simulator.

## Set up the iOS simulator

To prepare to run and test your Flutter app on the iOS simulator, follow these steps:

1. On your Mac, find the Simulator via Spotlight or by using the following command:
2. Make sure your simulator is using a 64-bit device (iPhone 5s or later) by checking the settings in the simulator's **Hardware > Device** menu.

3. Depending on your development machine's screen size, simulated high-screen-density iOS devices might overflow your screen. Grab the corner of the simulator and drag it to change the scale. You can also use the **Window > Physical Size** or **Window > Pixel Accurate** options if your computer's resolution is high enough.
    - If you are using a version of XCode older than 9.1, you should instead set the device scale in the **Window > Scale** menu.

## Create and run a simple Flutter app

To create your first Flutter app and test your setup, follow these steps:

1. Create a new Flutter app by running the following from the command line:
2. A `my_app` directory is created, containing Flutter's starter app. Enter this directory:
3. To launch the app in the Simulator, ensure that the Simulator is running and enter:

## Deploy to iOS devices

To deploy your Flutter app to a physical iOS device you'll need to set up physical device deployment in XCode and an Apple Developer account. If your app is using Flutter plugins, you will also need the third-party Cocoa Pods dependency manager.

1. You can skip this step if your apps do not depend on Flutter plugins with native iOS code. Install and set up Cocoa Pods by running the following commands:

    ```
    $ sudo gem install cocoapods
    ```

2. Follow the XCode signing flow to provision your project:
    a. Open the default XCode workspace in your project by running `open ios/Runner.xcworkspace` in a terminal window from your Flutter project directory.
    b. Select the device you intend to deploy to in the device drop-down menu next to the run button.

c. Select the `Runner` project in the left navigation panel.

d. In the `Runner` target settings page, make sure your Development Team is selected under **Signing & Capabilities > Team**.

   When you select a team, XCode creates and downloads a Development Certificate, registers your device with your account, and creates and downloads a provisioning profile (if needed).

   - To start your first iOS development project, you might need to sign into Xcode with your Apple ID. Development and testing is supported for any Apple ID. Enrolling in the Apple Developer Program is required to distribute your app to the App Store. For details about membership types, see Choosing a Membership.
   - The first time you use an attached physical device for iOS development, you need to trust both your Mac and the Development Certificate on that device. Select `Trust` in the dialog prompt when first connecting the iOS device to your Mac.

     Then, go to the Settings app on the iOS device, select **General > Device Management** and trust your Certificate. For first time users, you may need to select **General > Profiles > Device Management** instead.
   - If automatic signing fails in XCode, verify that the project's **General > Identity > Bundle Identifier** value is unique.

3. Start your app by running `flutter run` or clicking the Run button in XCode Android setup.

# Android setup

## Install Android Studio

1. Download and install Android Studio.
2. Start Android Studio, and go through the 'Android Studio Setup Wizard'. This installs the latest Android SDK, Android SDK Command-line Tools, and Android SDK Build-Tools, which are required by Flutter when developing for Android.
3. Run `flutter doctor` to confirm that Flutter has located your installation of Android Studio. If Flutter cannot locate it, run `flutter config --android-studio-dir` to set the directory that Android Studio is installed to.

## Set up your Android device

To prepare to run and test your Flutter app on an Android device, you need an Android device running Android 5 (API level 21) or higher.

1. Enable **Developer options** and **USB debugging** on your device. Detailed instructions are available in the Android documentation.
2. Windows-only: Install the Google USB Driver.
3. Using a USB cable, plug your phone into your computer. If prompted on your device, authorize your computer to access your device.
4. In the terminal, run the `flutter devices` command to verify that Flutter recognizes your connected Android device. By default, Flutter uses the version of the Android SDK where your `adb` tool is based. If you want Flutter to use a different installation of the Android SDK, you must set the `ANDROID_SDK_ROOT` environment variable to that installation directory.

## Set up the Android emulator

To prepare to run and test your Flutter app on the Android emulator, follow these steps:

1. Enable VM acceleration on your machine.
2. Launch **Android Studio**, click the **AVD Manager** icon, and select **Create Virtual Device...**
   - In older versions of Android Studio, you should instead launch **Android Studio > Tools > Android > AVD Manager** and select **Create Virtual Device...**. (The **Android** submenu is only present when inside an Android project.)
   - If you do not have a project open, you can choose **Configure > AVD Manager** and select **Create Virtual Device...**
3. Choose a device definition and select **Next**.
4. Select one or more system images for the Android versions you want to emulate, and select **Next**. An *x86* or *x86_64* image is recommended.

5. Under Emulated Performance, select **Hardware - GLES 2.0** to enable hardware acceleration.
6. Verify the AVD configuration is correct, and select **Finish**.

   For details on the above steps, see Managing AVDs.
7. In Android Virtual Device Manager, click **Run** in the toolbar. The emulator starts up and displays the default canvas for your selected OS version and device.

## Agree to Android Licenses

Before you can use Flutter, you must agree to the licenses of the Android SDK platform. This step should be done after you have installed the tools listed above.

1. Make sure that you have a version of Java 8 installed and that your `JAVA_HOME` environment variable is set to the JDK's folder.

   Android Studio versions 2.2 and higher come with a JDK, so this should already be done.
2. Open an elevated console window and run the following command to begin signing licenses.

   ```
   $ flutter doctor --android-licenses
   ```
3. Review the terms of each license carefully before agreeing to them.
4. Once you are done agreeing with licenses, run `flutter doctor` again to confirm that you are ready to use Flutter.

# Setup An Editor With Flutter

**Install the Flutter and Dart plugins**

The installation instructions vary by platform.

## Mac

Use the following instructions for macOS:

1. Start Android Studio.
2. Open plugin preferences (**Preferences > Plugins** as of v3.6.3.0 or later).
3. Select the Flutter plugin and click **Install**.
4. Click **Yes** when prompted to install the Dart plugin.
5. Click **Restart** when prompted.

## Linux or Windows

Use the following instructions for Linux or Windows:

1. Open plugin preferences (**File > Settings > Plugins**).
2. Select **Marketplace**, select the Flutter plugin and click **Install**.

# Install VS Code

VS Code is a lightweight editor with Flutter app execution and debug support.

- VS Code, latest stable version

# Install the Flutter and Dart plugins

1. Start VS Code.
2. Invoke **View > Command Palette...**.
3. Type “install”, and select **Extensions: Install Extensions**.
4. Type “flutter” in the extensions search field, select **Flutter** in the list, and click **Install**. This also installs the required Dart plugin.

# Validate your setup with the Flutter Doctor

1. Invoke **View > Command Palette...**.
2. Type “doctor”, and select the **Flutter: Run Flutter Doctor**.
3. Review the output in the **OUTPUT** pane for any issues. Make sure to select Flutter from the dropdown in the different Output Options.

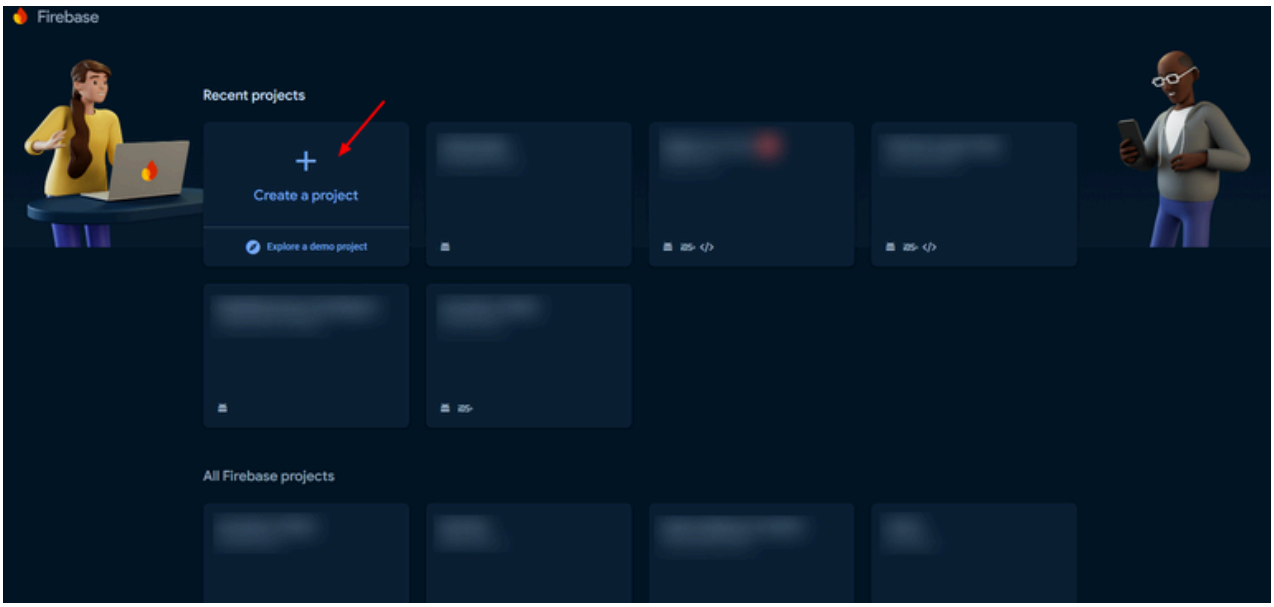
# Adding Firebase

INITIAL CHECKLIST

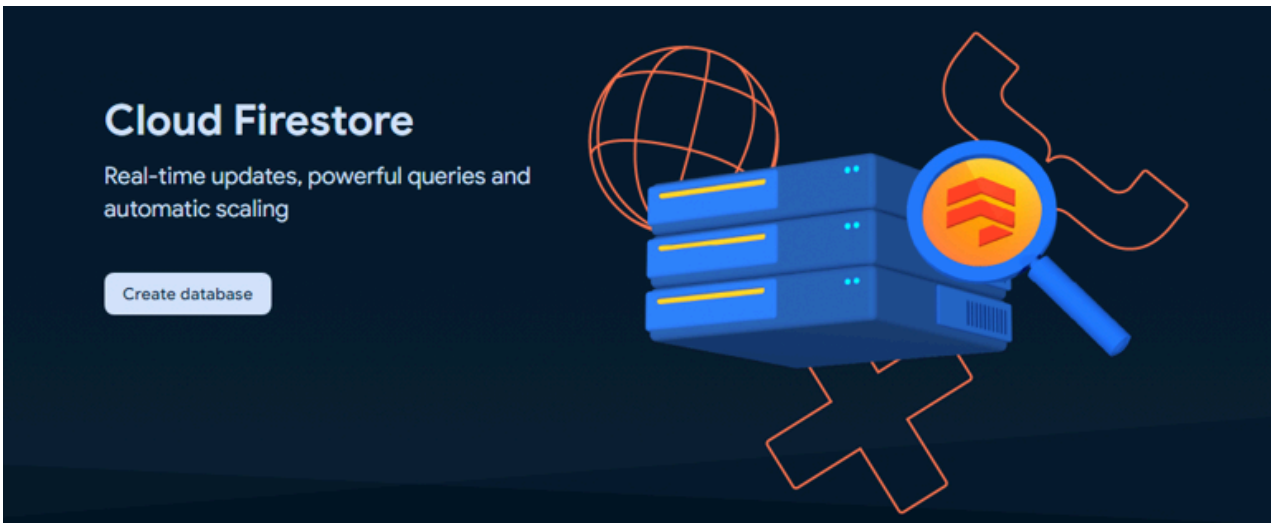
- Before we proceed, Make sure your App's BundleID is not containing an underscore _ character.
  For example, this won't work: `com.example.xyz_app`
- `flutter doctor -v` command is accessible from the system-wide terminal (not just the project folder)

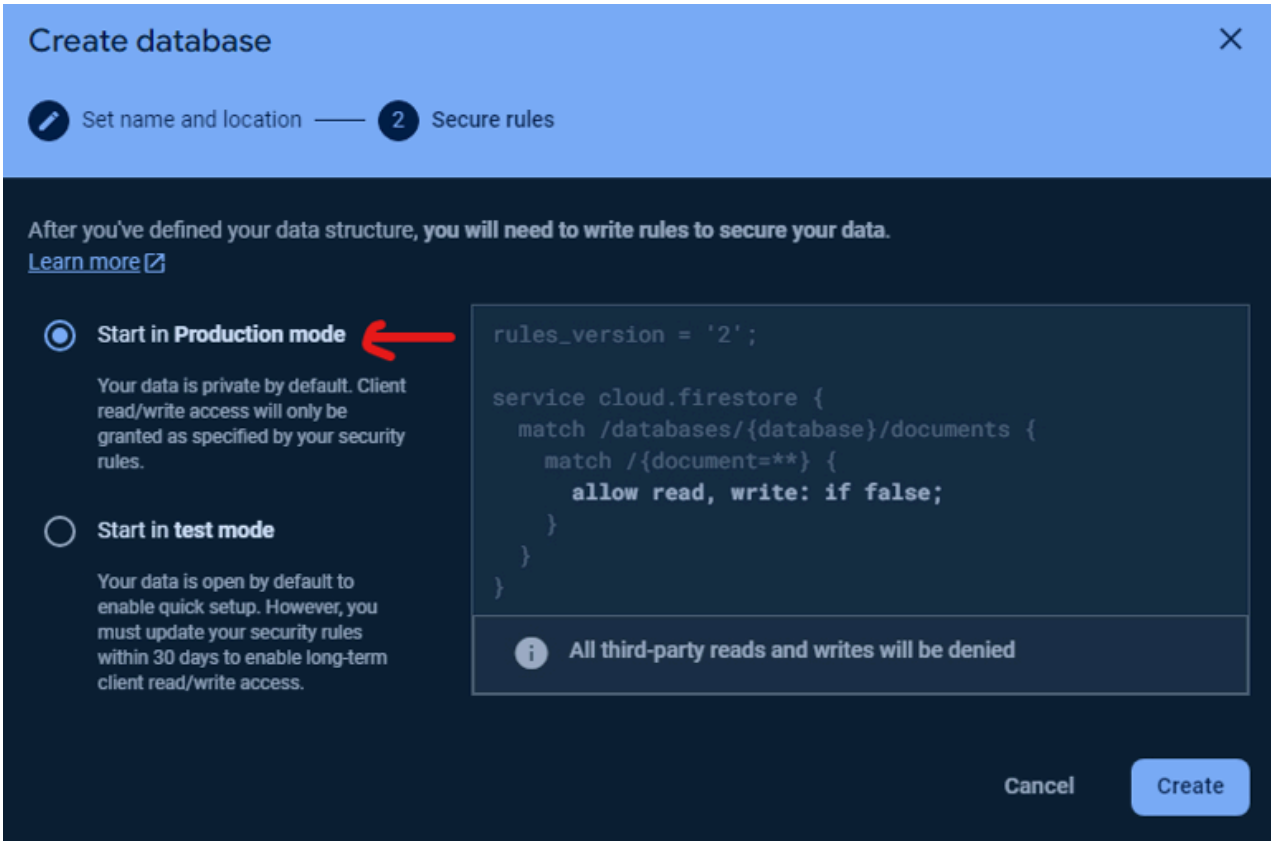
## Step 1: Creating Firebase Project

https://console.firebase.google.com/
console.firebase.google.com

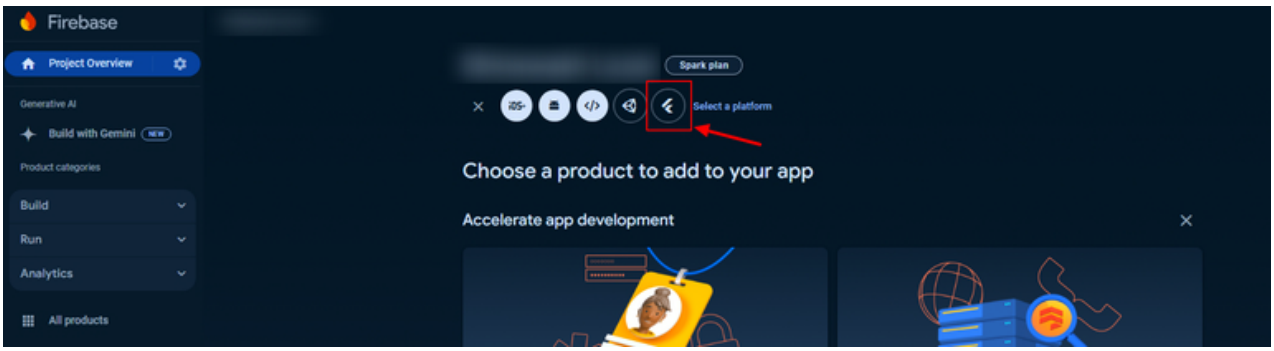



## Create Firestore Database



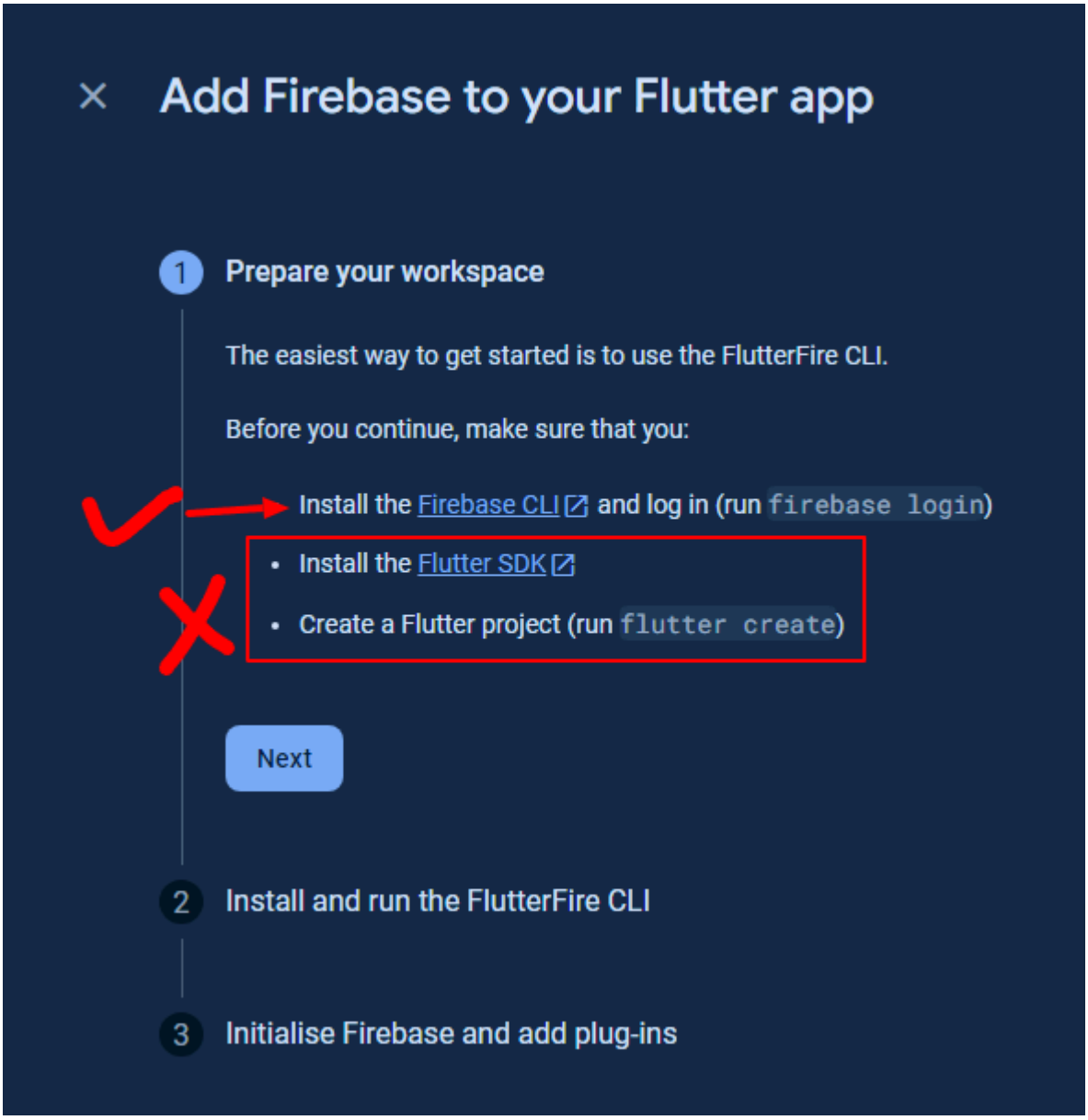




## Let's create flutter app configs inside the Firebase Project



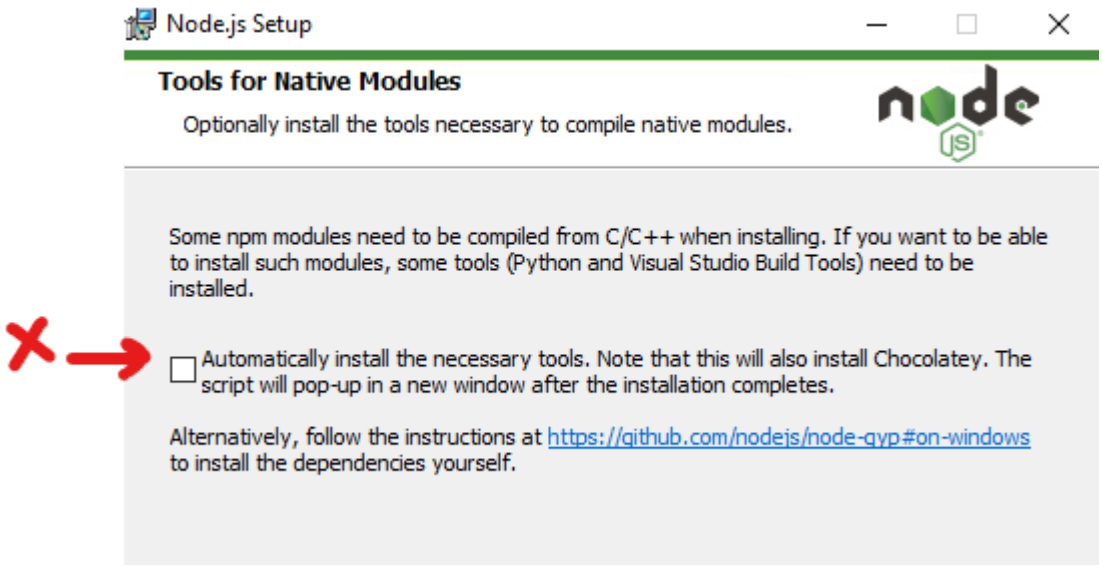

Now install Firebase CLI. We recommend taking the Node(NPM) route which will add Firebase CLI globally, which means it can be run from any terminal system-wide.

- 



NOTES:

- **Do not opt** for `Chocolatey` while installing the node. You won't need this, plus it will install a WHOLE lot of unwanted things.

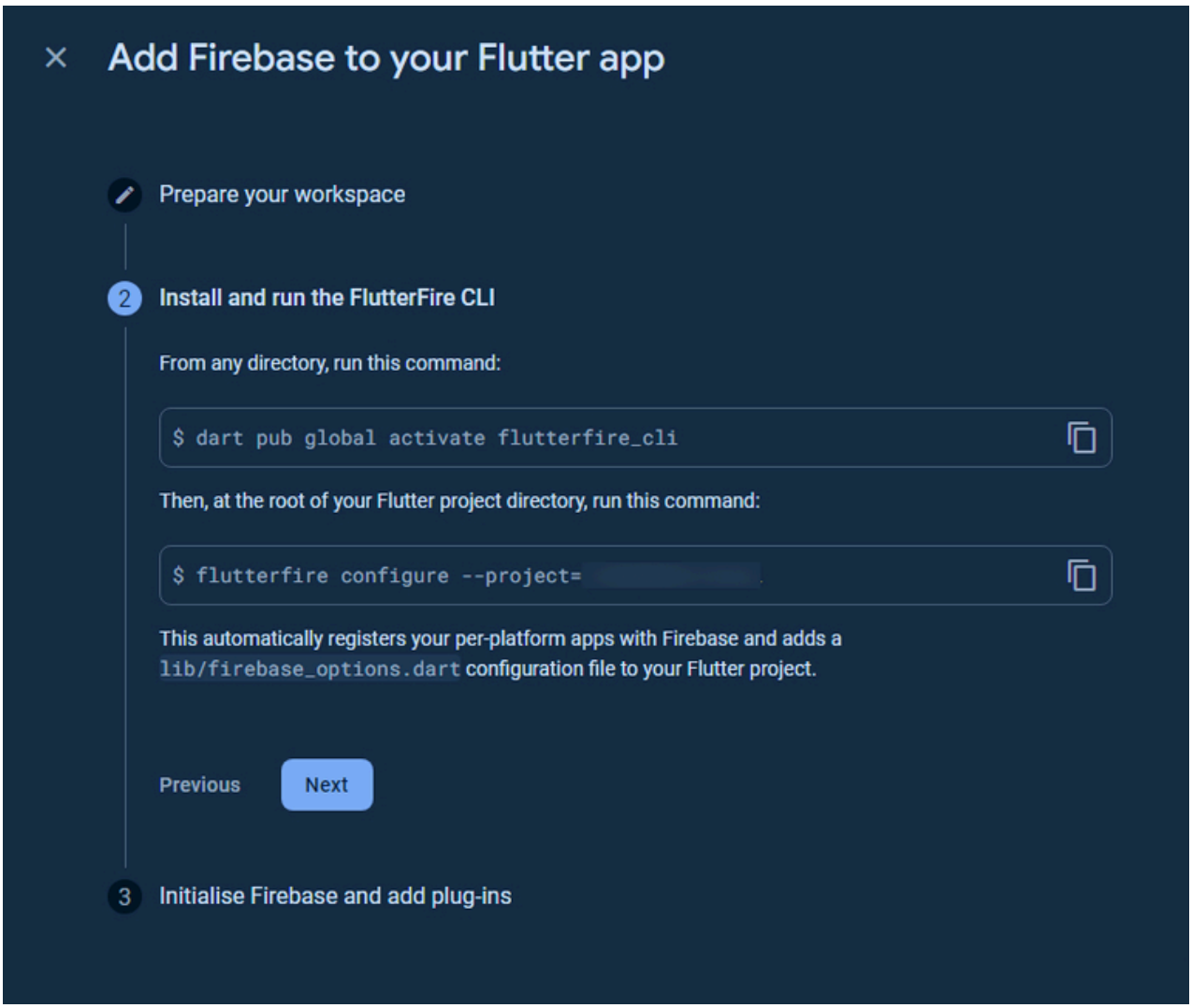



- If you have added/modified the system environment variables path during the process, a system reboot is a **must**.

Then log in to Firebase via CLI Wizard. or run the command: `firebase login` from system terminal

## Open the App's Project in your IDE

Open the terminal and run the following commands one by one at a project-level terminal




When CLI Wizard asks you for the platform, DO NOT CHOOSE any other than Android or iOS

this CLI command will create & fetch configurations automatically in your project,

It will create the following files:

```
android/app/google-services.json
lib/firebase_options.dart
firebase.json
```

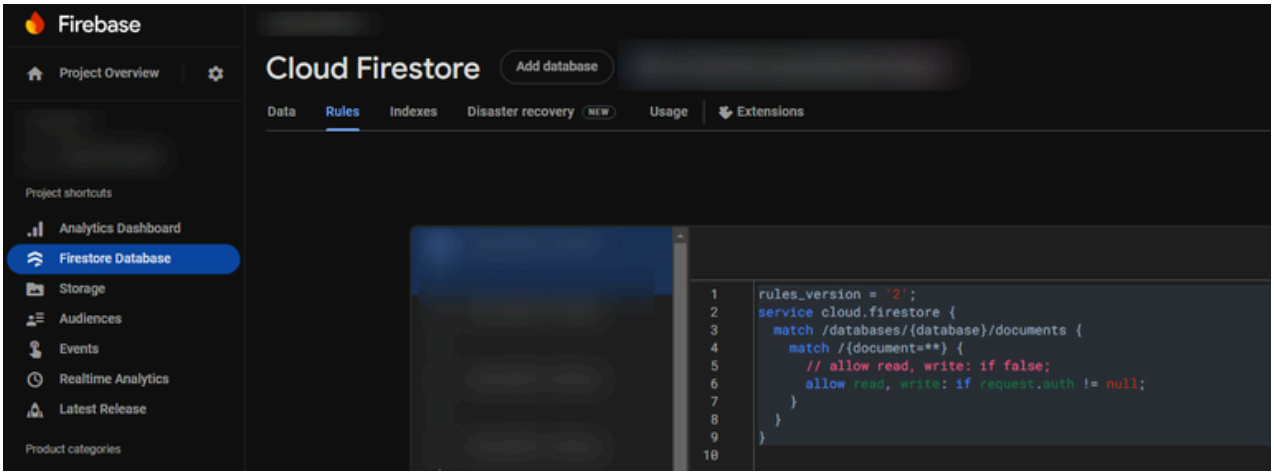
## Enable Cloud Firestore Service.

1. Open this Firebase Project Console > Cloud Firestore > Add Database > Choose the server closest to your user base

2. Goto `Rules` Tab

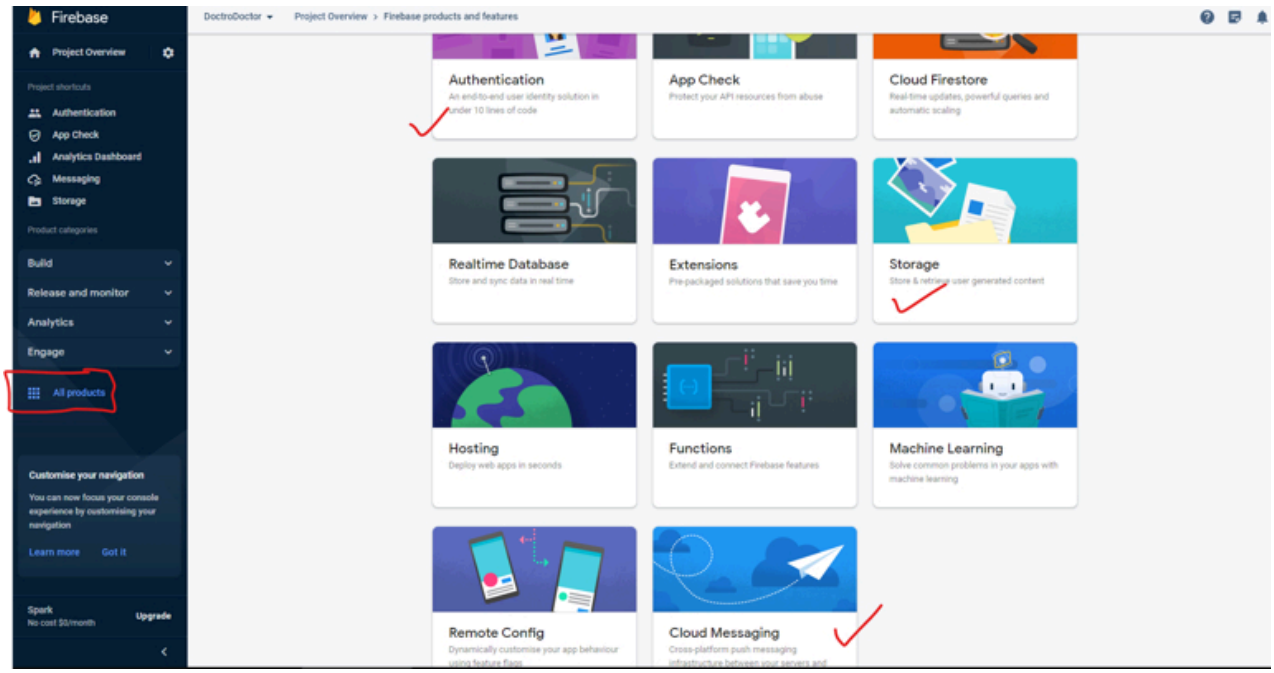
and set the below-given rules content:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      // allow read, write: if false;
      allow read, write: if request.auth != null;
    }
  }
}
```




After Connecting with Firebase, **DO NOT MAKE CHANGES TO YOUR LARAVEL DATABASE USER CREDENTIALS DIRECTLY, IT COULD MESS UP THE SYNC BETWEEN Firebase & Laravel**

# Enable Firebase Services

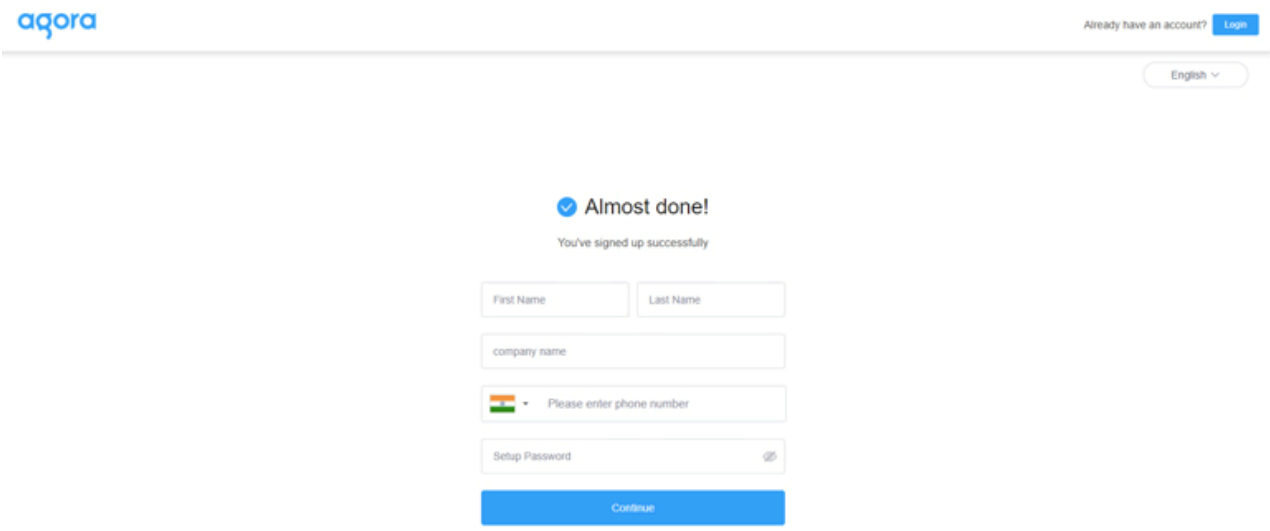



Note: Use Email/Password as Authentication Method

Authentication → To manage emails and passwords
Storage → When patient send any images in chat, it need storage
Messaging → To save messages

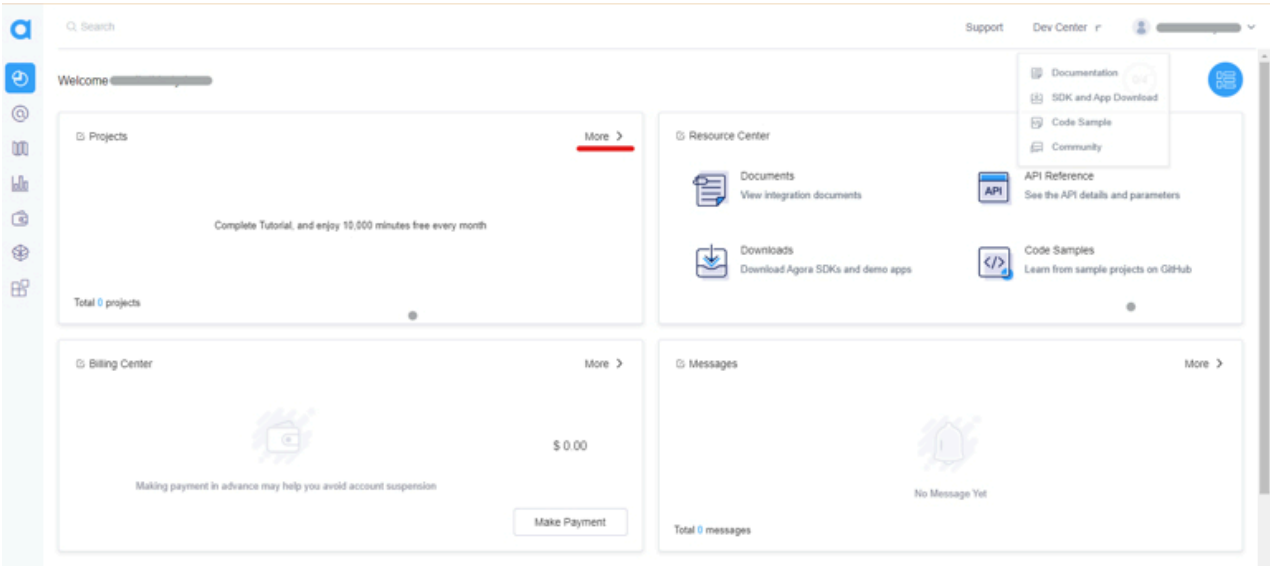
# Setup of Agora video call

1. Create an account on Agora.

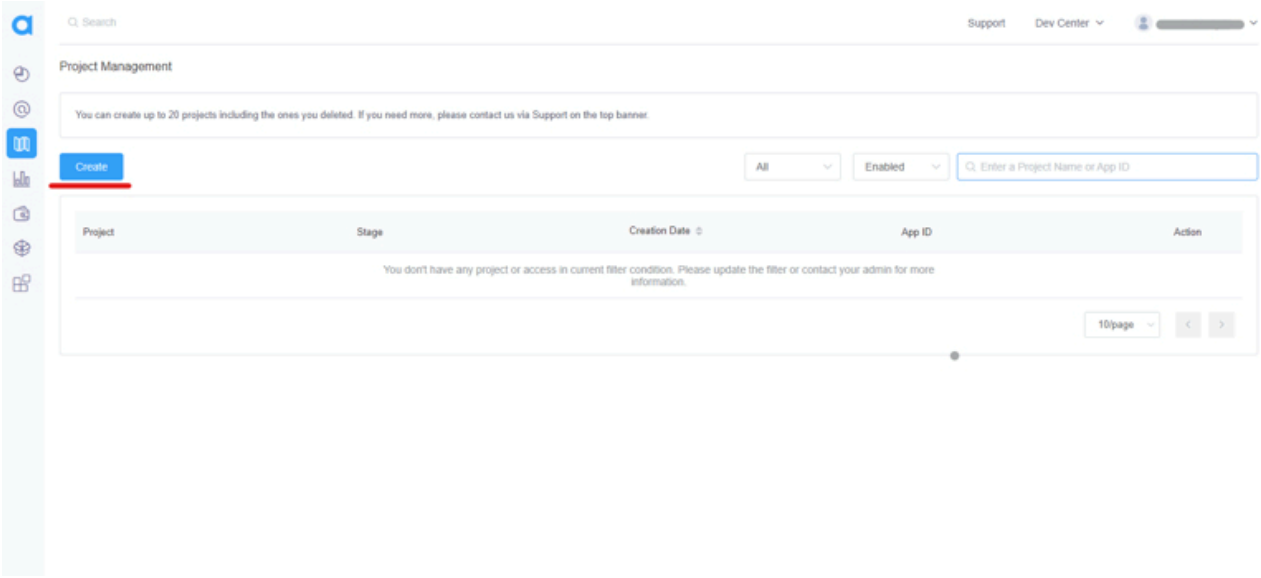



2. After completing the registration process, you will be redirected to the dashboard page.

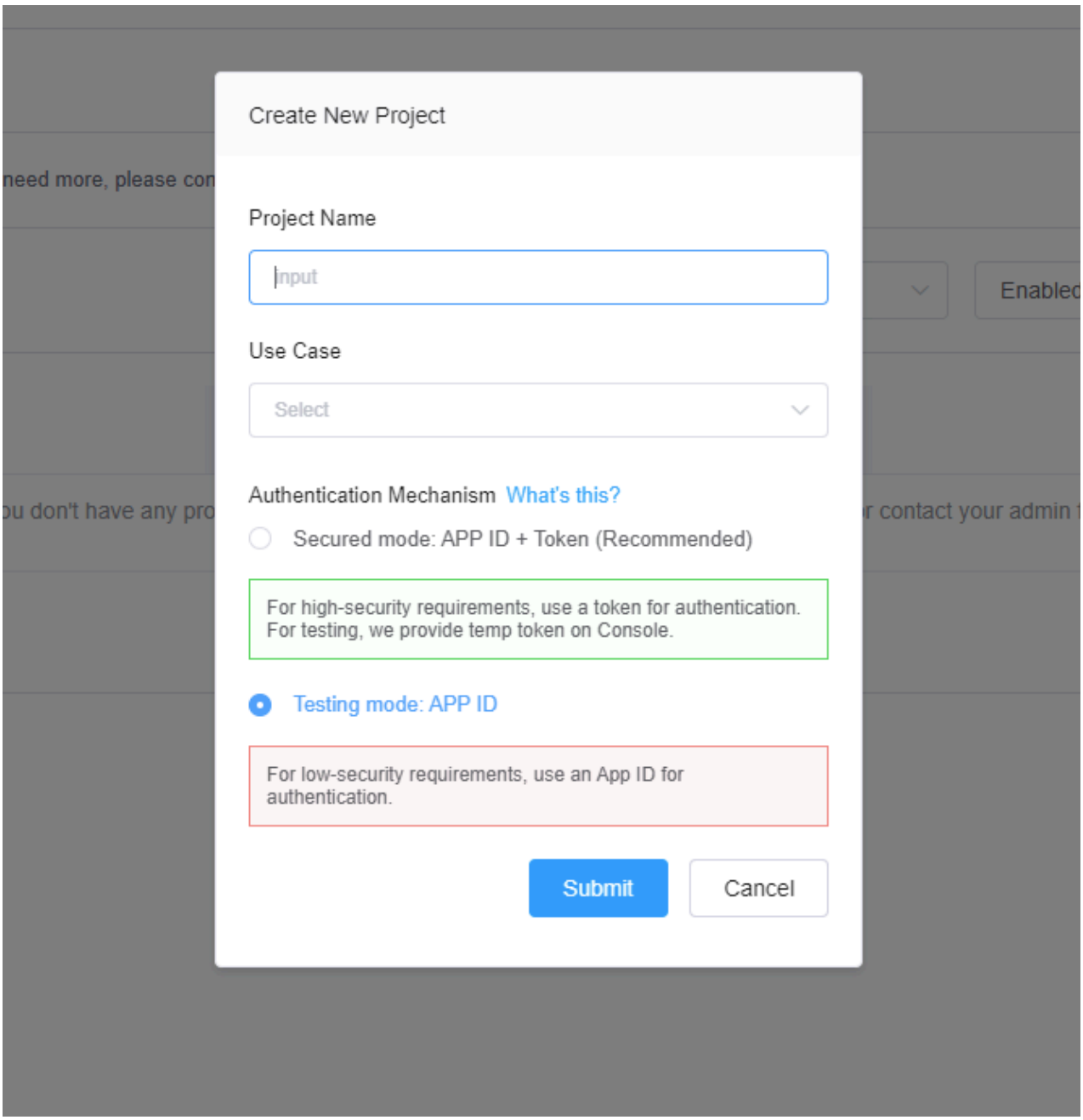
3. Navigate to the Projects management from left hand side panel of the dashboard.



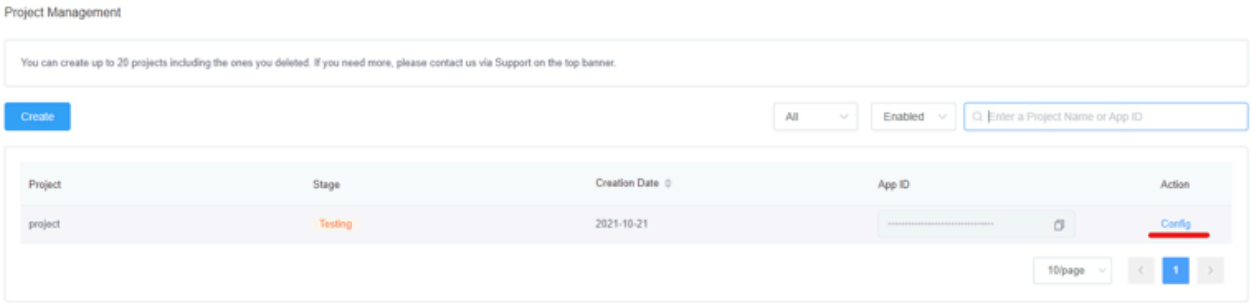

4. After that, showing Project Management screen, in that click on create button.



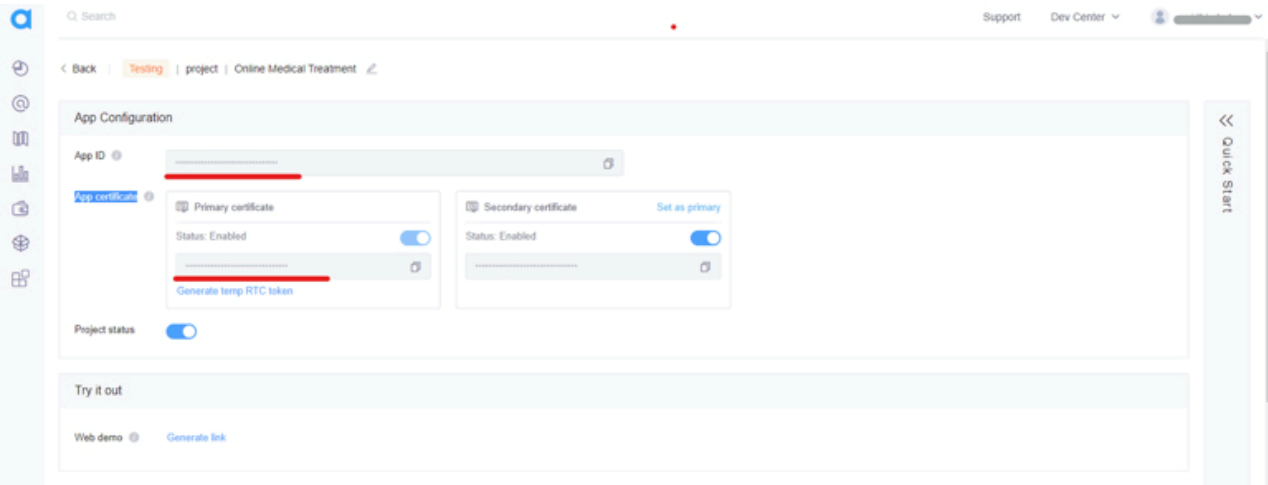

5. Create New Project dialog will appear, fill all project related details.



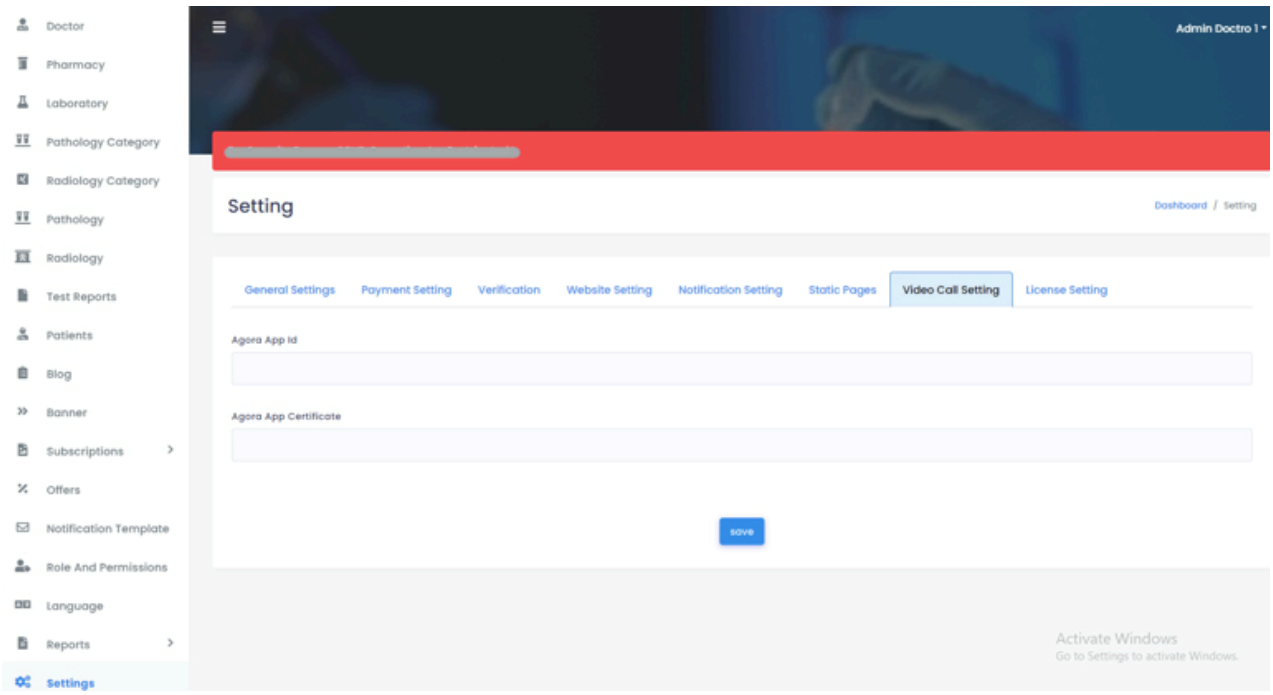

6. your Project created successfully, after that, right side shows action column, in that column click on config button.




7. project detail page will open, copy the App Id & App Certificate codes and set it in the admin panel of your project.




8. In admin panel, click on Settings button from menu panel, under video call setting option set your App Id & App Certificate and click on save button.

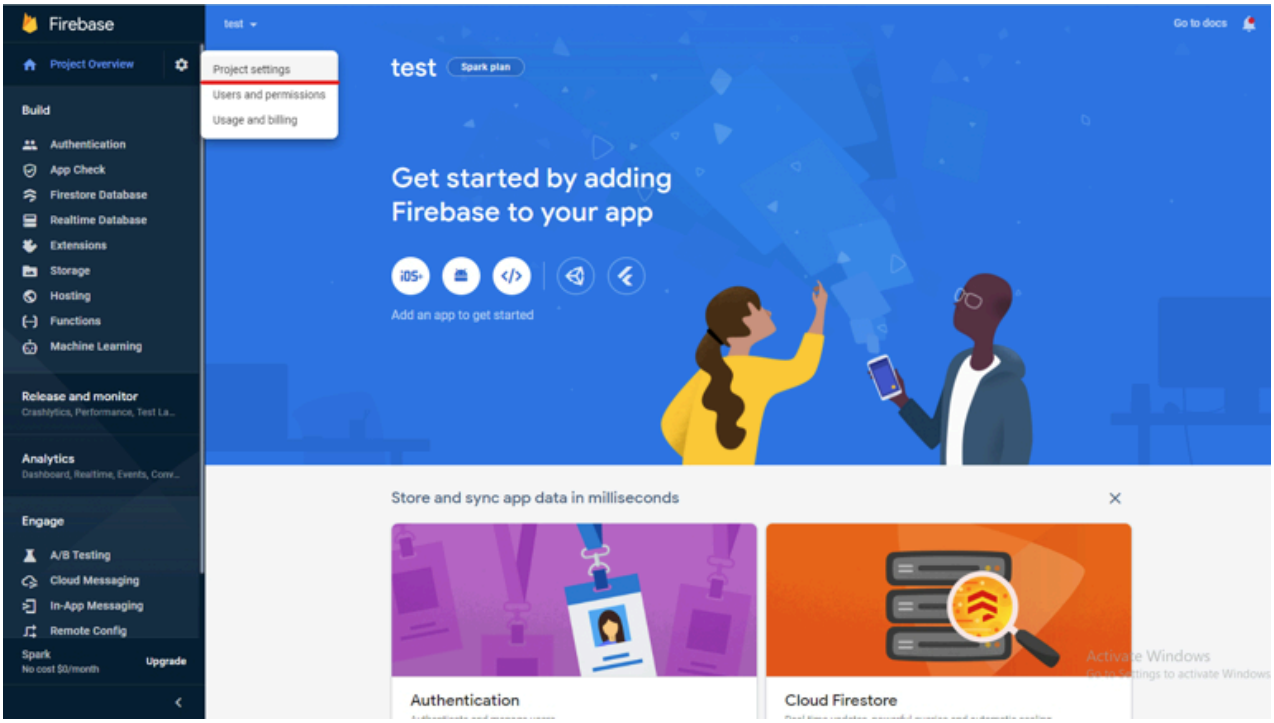



That's all for video call setup.

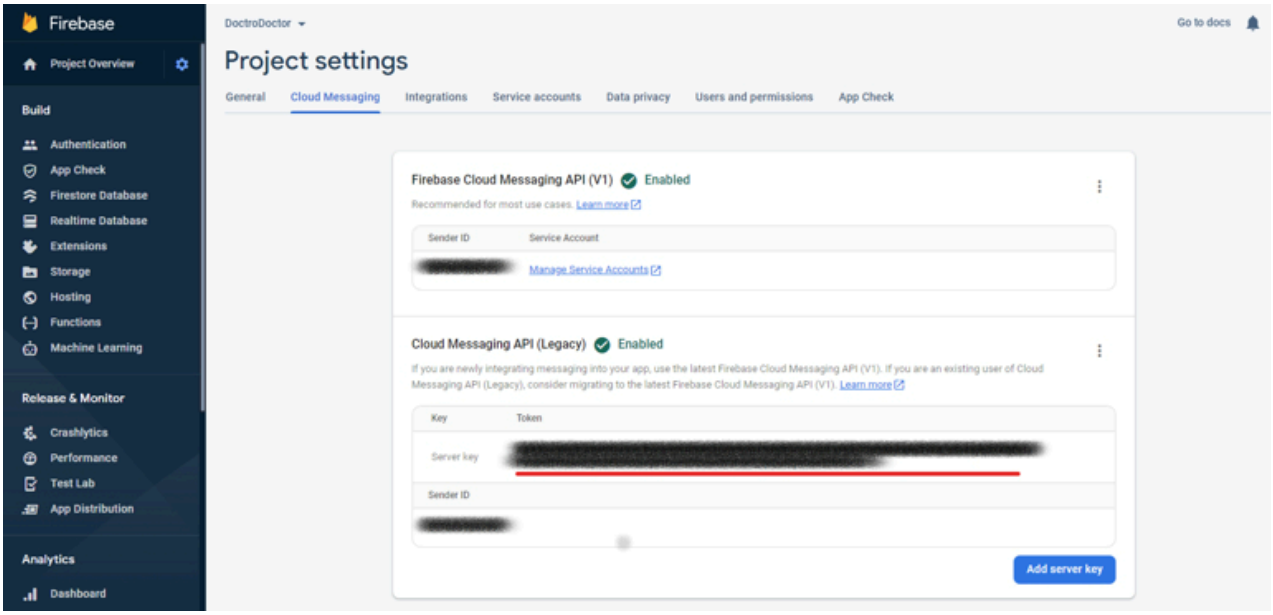
NOTE: Video Call Notification will not show up if OneSignal API Credentials are not provided in the *Notification Settings* Tab

# Setup to Firebase Chat Notification

1. After successfully creating Firebase database, Click the Settings icon next to Project Overview on the left side of the Dashboard, after 3 options will opened in which click on Project Settings.



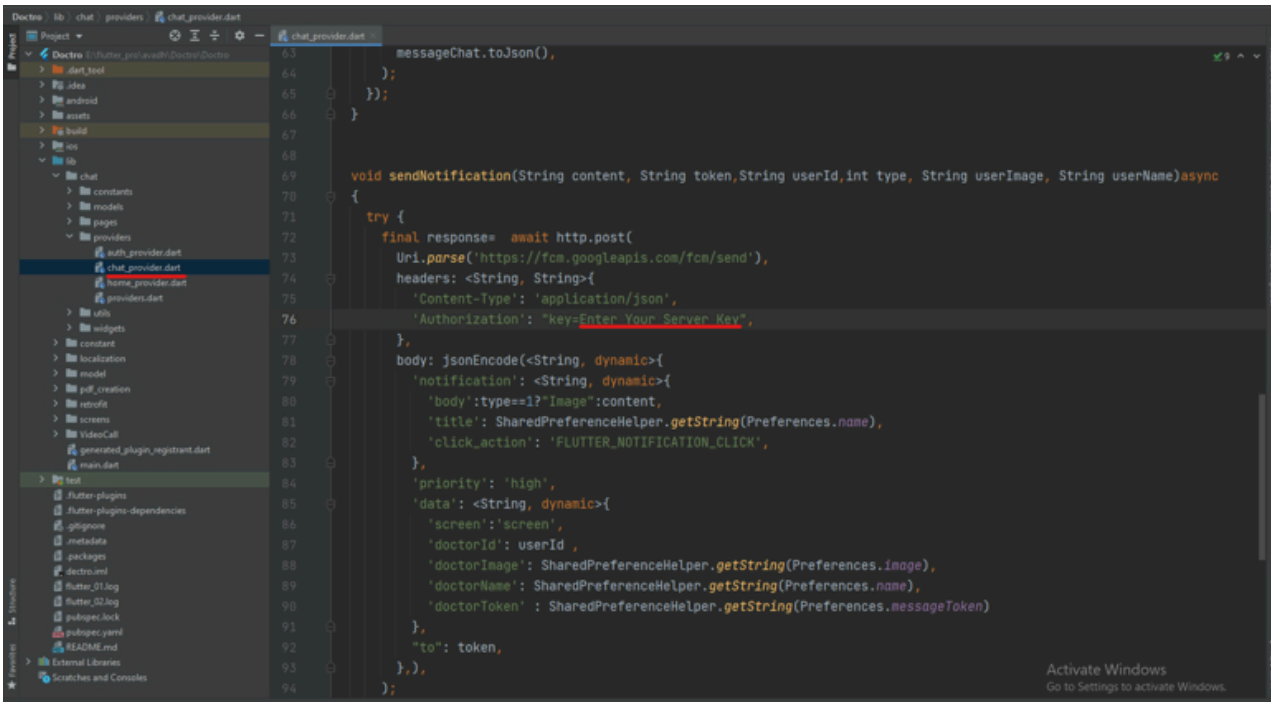

2. Then project settings screen will open, in that click on Cloud Messaging, in which copy the "**server key**".



3. Then put that server key in the Doctro project at the location below.

- **Doctro Doctor**

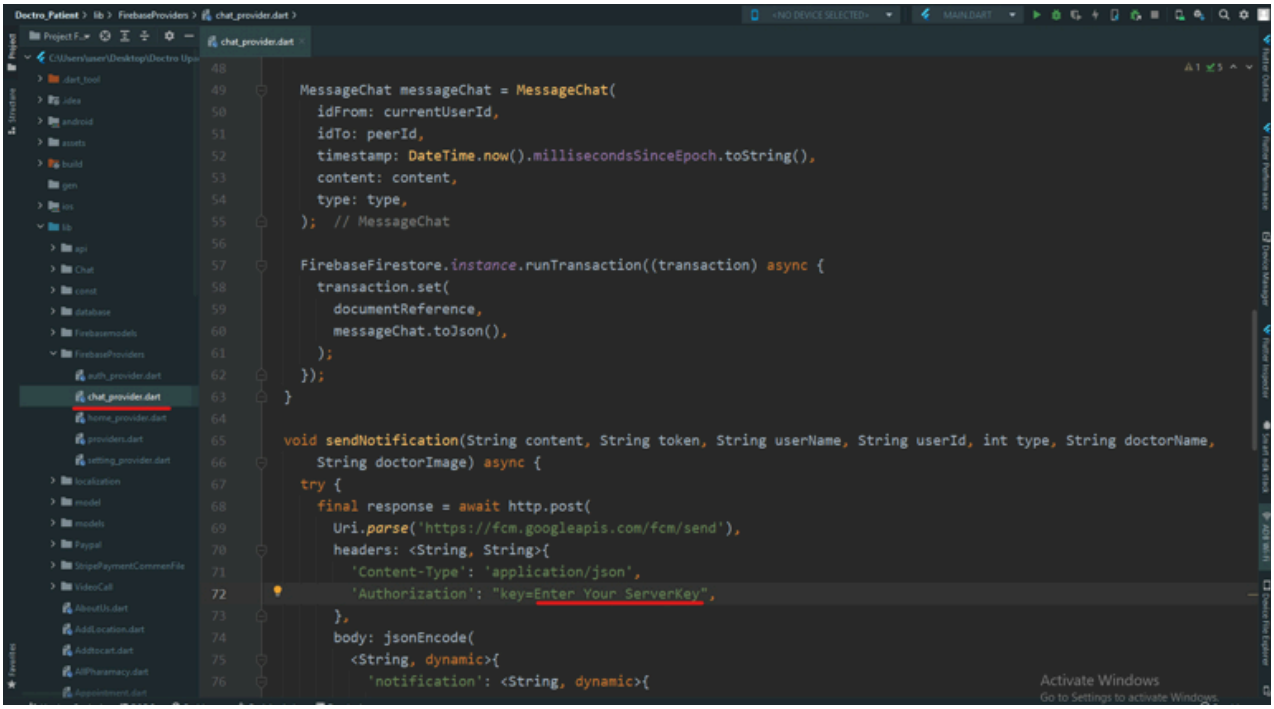
File Location is : - **project/lib/chat/providers/chat_provider.dart**.



```
        messageChat.toJson(),
      );
    });
  }


  void sendNotification(String content, String token,String userId,int type, String userImage, String userName)async
  {
    try {
      final response=  await http.post(
        Uri.parse('https://fcm.googleapis.com/fcm/send'),
        headers: <String, String>{
          'Content-Type': 'application/json',
          'Authorization': "key=Enter Your Server Key",
        },
        body: jsonEncode(<String, dynamic>{
          'notification': <String, dynamic>{
            'body':type==1?"Image":content,
            'title': SharedPreferenceHelper.getString(Preferences.name),
            'click_action': 'FLUTTER_NOTIFICATION_CLICK',
          },
          'priority': 'high',
          'data': <String, dynamic>{
            'screen':'screen',
            'doctorId': userId ,
            'doctorImage': SharedPreferenceHelper.getString(Preferences.image),
            'doctorName': SharedPreferenceHelper.getString(Preferences.name),
            'doctorToken' : SharedPreferenceHelper.getString(Preferences.messageToken)
          },
          "to": token,
        },),
      );
```

- **Doctro Patient**

File Location is : - **project/lib/FirebaseProviders/chat_provider.dart**.



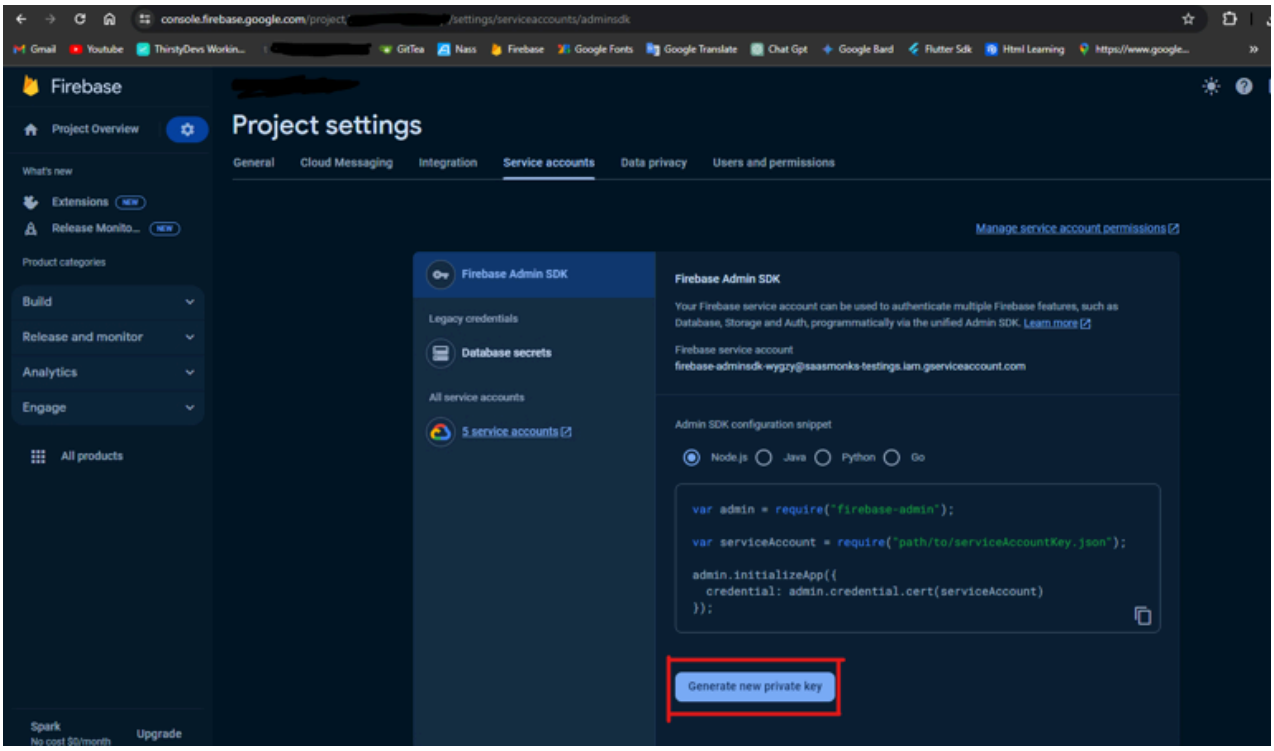
```
  MessageChat messageChat = MessageChat(
    idFrom: currentUserId,
    idTo: peerId,
    timestamp: DateTime.now().millisecondsSinceEpoch.toString(),
    content: content,
    type: type,
  );  // MessageChat

  FirebaseFirestore.instance.runTransaction((transaction) async {
    transaction.set(
      documentReference,
      messageChat.toJson(),
    );
  });
}

void sendNotification(String content, String token, String userName, String userId, int type, String doctorName,
    String doctorImage) async {
  try {
    final response = await http.post(
      Uri.parse('https://fcm.googleapis.com/fcm/send'),
      headers: <String, String>{
        'Content-Type': 'application/json',
        'Authorization': "key=Enter Your ServerKey",
      },
      body: jsonEncode(
        <String, dynamic>{
          'notification': <String, dynamic>{
```

# OneSignal

## OneSignal Configuration on Backend (Admin)

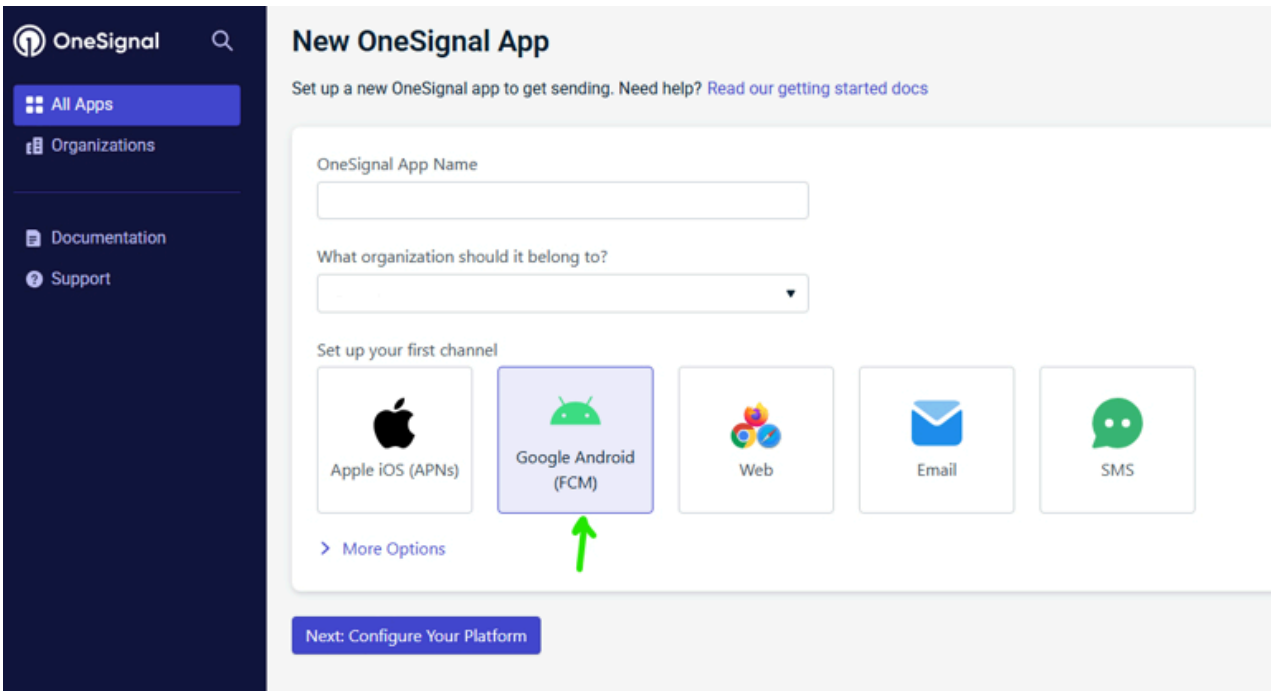
### STEP 1:

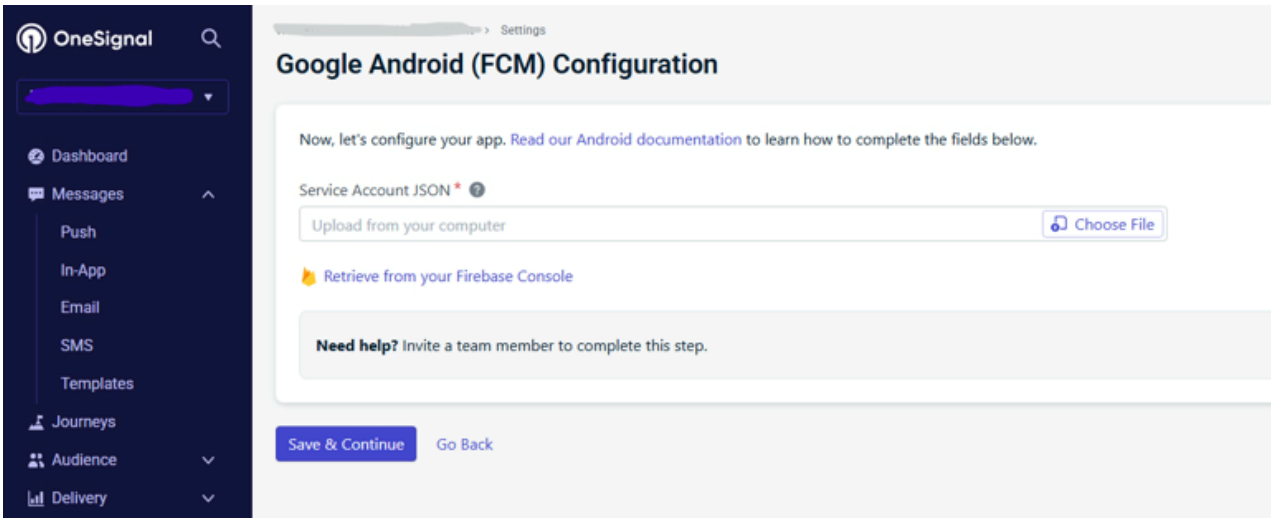
This one JSON file will work for all the apps which are within this One Firebase Project




### STEP 2:

Go to the OneSignal official site and add the specific platform SDK implementation.
Refer to: OneSignal Firebase Credentials Documentation




Next, Choose ***Flutter*** when onesignal asks for Platform/SDK choices.

## STEP 3:

Copy and paste the OneSignal App ID into the Admin Panel > Notification/OneSignal Settings > App ID input box.

Normally, API Key & Auth Key will be the same if you are using the same account of OneSignal for all the apps, Still, contact **OneSignal Support** if you are not sure what to do in *their platform*.

## STEP 4:

Make sure you create two different apps in one signal—so 2 App IDs you will have.

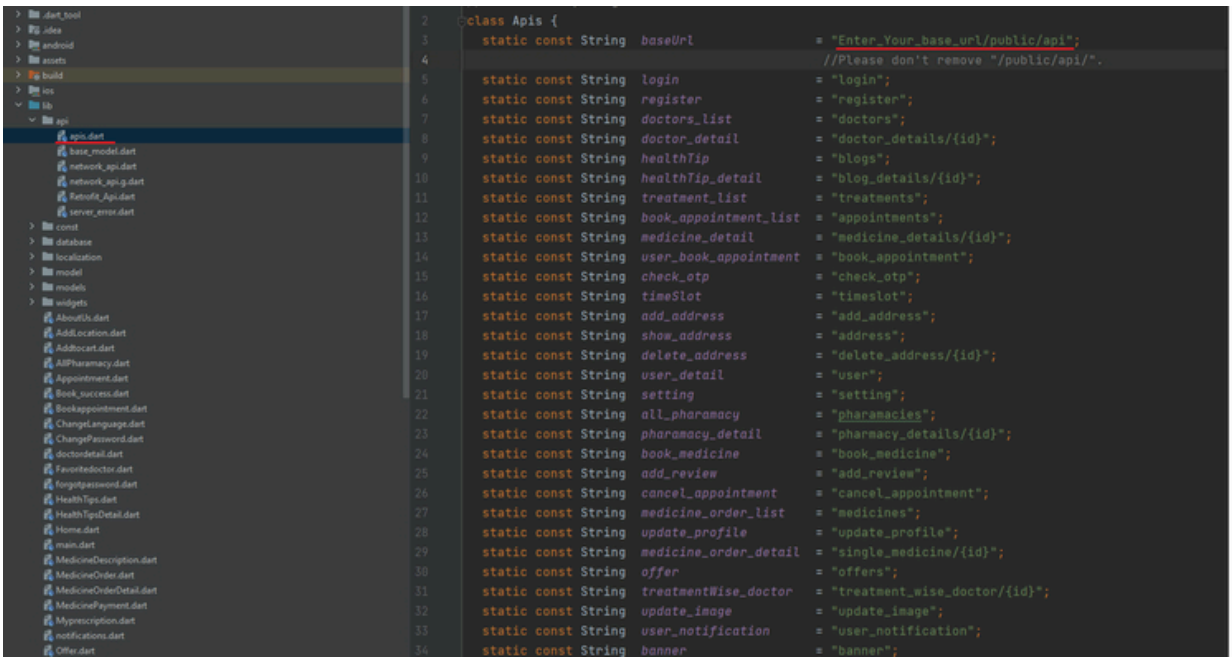
**NOTE:** No matter if it’s Android or iOS, the APP ID will be the same on both platforms. Mobile Apps will automatically use this via our backend APIs.

**NOTE:** Please note these App IDs to be entered in the Individual Mobile App Configuration steps.

# How to Change BASE_URL(for User, Doctor)

## For User Project

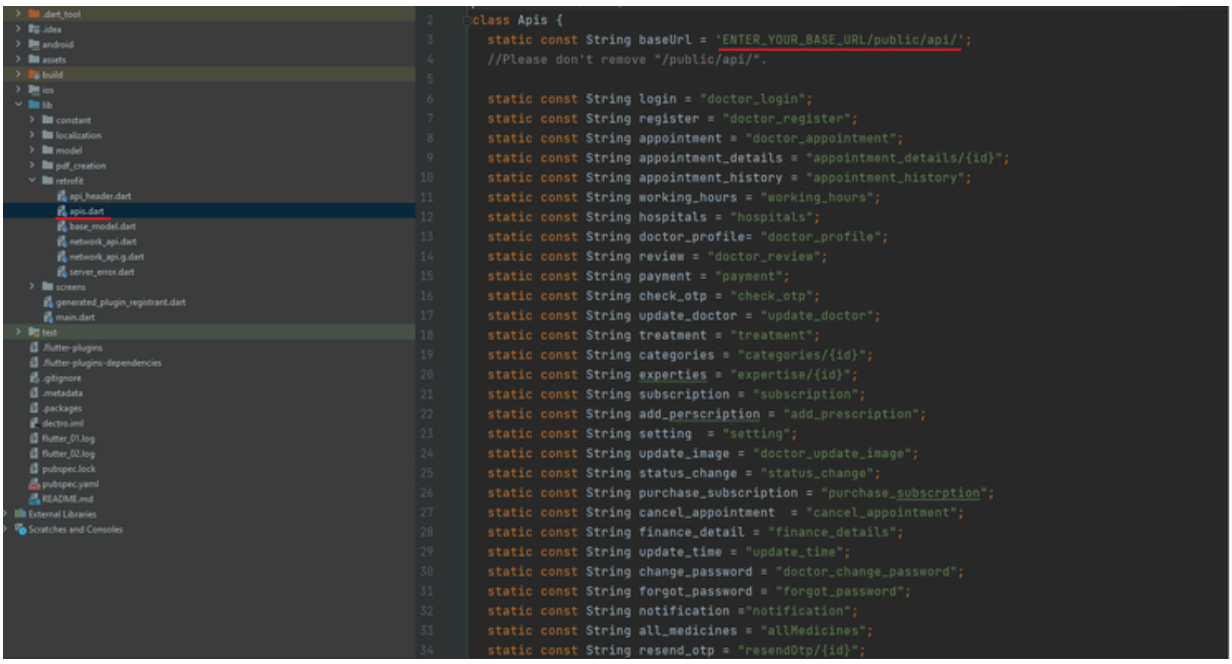
1. Open project code with android studio
2. From left pane : Select Project →
3. here you can see all project folder
4. Select lib folder.
5. Open apis.dart file.
6. Select project → lib → api → apis.dart File.
7. in this file you can see baseUrl: "Enter_Your_Base_Url/public/api/" : replace your Server URL with it.
8. After replace URL.
9. at the bottom of android studio : TODO, Dart Analysis, Terminal buttons available.
10. click on Terminal button.
11. execute command : - **flutter pub get && flutter pub run build_runner build --delete-conflicting-outputs** wait for complete of the process.
12. Now follow steps of How to Generate APK file.



```
class Apis {
  static const String baseUrl             = "Enter_Your_base_url/public/api";
                                           //Please don't remove "/public/api/".
  static const String login               = "login";
  static const String register            = "register";
  static const String doctors_list        = "doctors";
  static const String doctor_detail       = "doctor_details/{id}";
  static const String healthTip           = "blogs";
  static const String healthTip_detail    = "blog_details/{id}";
  static const String treatment_list      = "treatments";
  static const String book_appointment_list = "appointments";
  static const String medicine_detail     = "medicine_details/{id}";
  static const String user_book_appointment = "book_appointment";
  static const String check_otp           = "check_otp";
  static const String timeSlot            = "timeslot";
  static const String add_address         = "add_address";
  static const String show_address        = "address";
  static const String delete_address      = "delete_address/{id}";
  static const String user_detail         = "user";
  static const String setting             = "setting";
  static const String all_pharamacy       = "pharamacies";
  static const String pharamacy_detail    = "pharmacy_details/{id}";
  static const String book_medicine       = "book_medicine";
  static const String add_review          = "add_review";
  static const String cancel_appointment  = "cancel_appointment";
  static const String medicine_order_list = "medicines";
  static const String update_profile      = "update_profile";
  static const String medicine_order_detail = "single_medicine/{id}";
  static const String offer               = "offers";
  static const String treatmentWise_doctor = "treatment_wise_doctor/{id}";
  static const String update_image        = "update_image";
  static const String user_notification   = "user_notification";
  static const String banner              = "banner";
```

# For Doctor Project

1. Open project code with android studio
2. From left pane : Select Project →
3. here you can see all project folder
4. Select lib folder.
5. Open apis.dart file.
6. Select project → lib → retrofit → apis.dart File.
7. in this file you can see baseUrl: "Enter_Your_Base_Url/public/api/" : replace your Server URL with it.
8. After replace URL.
9. at the bottom of android studio : TODO, Dart Analysis, Terminal buttons available.
10. click on Terminal button.
11. execute command : - **flutter pub get && flutter pub run build_runner build --delete-conflicting-outputs** wait for complete of the process.
12. Now follow steps of How to Generate APK file.




NOTE : Base URL for app is : Enter_Your_BaseUrl/public/api/

# Change App Stuff(for User)

Changes in Application for name, Package name, Application Color, Splash screen, Application Icon.

# For Change Application name(User)

## For Android

- Go to below path and change the name `android:label="Your_App_Name"`
- File Location is : - **project/android/app/src/main/AndroidManifest.xml**.

```
<application
    android:label="Your_App_Name"        //Change this line
    android:usesCleartextTraffic="true"
    android:icon="@mipmap/ic_launcher">
```

## For iOS

- Go to below path and change app name which seems like this `<string>Your_App_Name</string>`
- File Location is : - **project/ios/Runner/Info.plist.**

```
<key>CFBundleName</key>
<string>Your_App_Name</string>
```

# For Change Application package name (User,Doctor)

Execute below commands in project/terminal.

1. `flutter pub add change_app_package_name`

2. `flutter pub get`
3. `flutter pub run change_app_package_name:main com.new.package.name`

**Note** : - (replace your package name with this "com.new.package.name").

# For User App :-

## 1. For Change Application color

go to lib/const/Palette.dart file and change color according to your need.

## 2. For Change Application splash screen

- Delete '**splash.png**' file from **images** folder.
- Put your splash screen image in **images** folder**.**
- rename your splash screen image to '**splash.png**' this name.
- **Note** - Execute the command in the project terminal '**flutter pub run flutter_native_splash:create**'.

## 3. For Change Application icon

- Delete '**appicon.png**' file from **images** folder.
- Put your app icon in i**mages** folder**.**
- rename your app icon to '**appicon.png**' this name.
- **Note** - Execute the command in the project terminal '**flutter pub run flutter_launcher_icons:main**'.

# For Doctor App :-

## 1. For Change Application color

go to lib/constant/color_constant.dart file and change color according to your need.

## 2. For Change Application splash screen

- Delete '**splash.png**' file from **images** folder.
- Put your splash screen image in **images** folder**.**
- rename your splash screen image to '**splash.png**' this name.
- **Note** - Execute the command in the project terminal '**flutter pub run flutter_native_splash:create**'.

## 3. For Change Application icon

- Delete '**appLogo.png**' file from **images** folder.
- Put your app icon in i**mages** folder**.**
- rename your app icon to '**appLogo.png**' this name.
- **Note** - Execute the command in the project terminal '**flutter pub run flutter_launcher_icons:main**'.

# Add google map key(for User)

First of all getting google map keys for both Android and iOS.

## Android :-

For Android, follow the instructions at Maps SDK for Android — Get API Key. Once you have your API key, change it to your Flutter app in the application manifest (`android/app/src/main/AndroidManifest.xml`), change this ENTER_YOUR_GOOGLE_ANDROID_KEY.

```
android:value="ENTER_YOUR_GOOGLE_ANDROID_KEY"/>
```

## iOS :-

For iOS, follow the instructions at Maps SDK for iOS — Get API Key. Once you have this key, add it to your Flutter app in the application delegate (`ios/Runner/AppDelegate.m`),change ENTER_YOUR_GOOGLE_iOS_KEY.

```
#import "AppDelegate.h"#import "GeneratedPluginRegistrant.h"#import "Goog
```

## Add your google map key in constant file

constant file path is - project/lib/const/prefConstatnt.dart.

change google map key here.

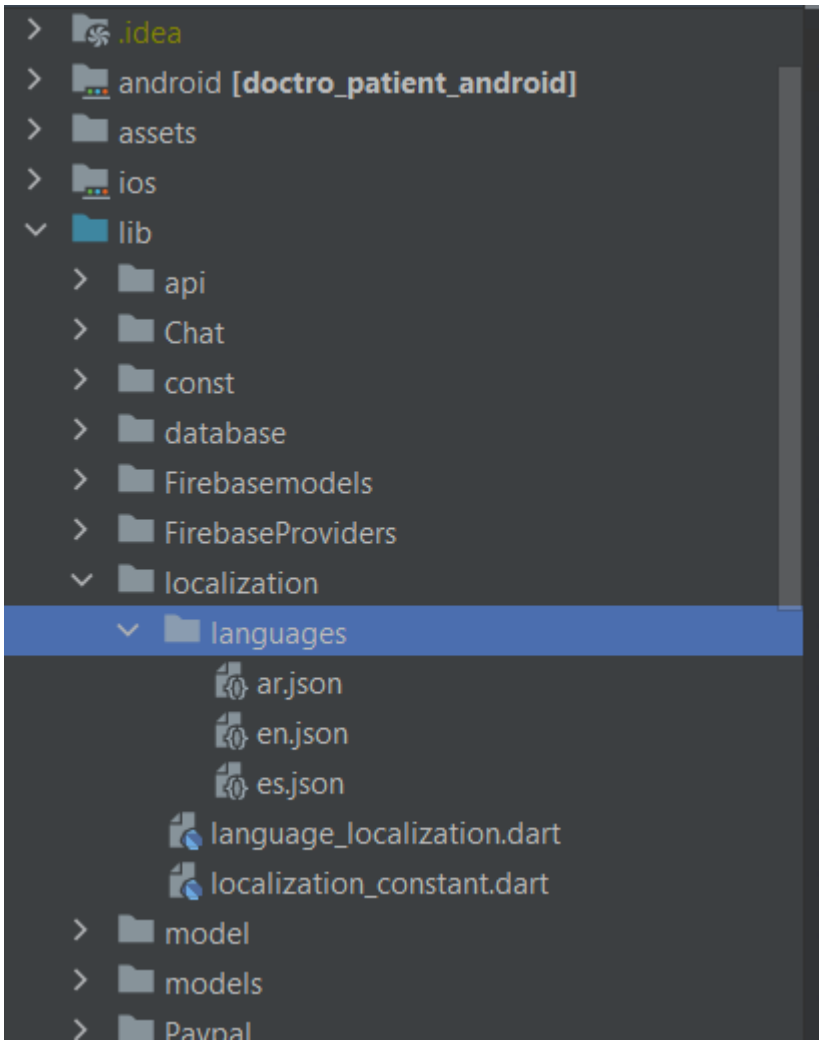
```
static const String map_key = "Enter_Your_Map_Key";
```

# Localization/Translation

## Patient App

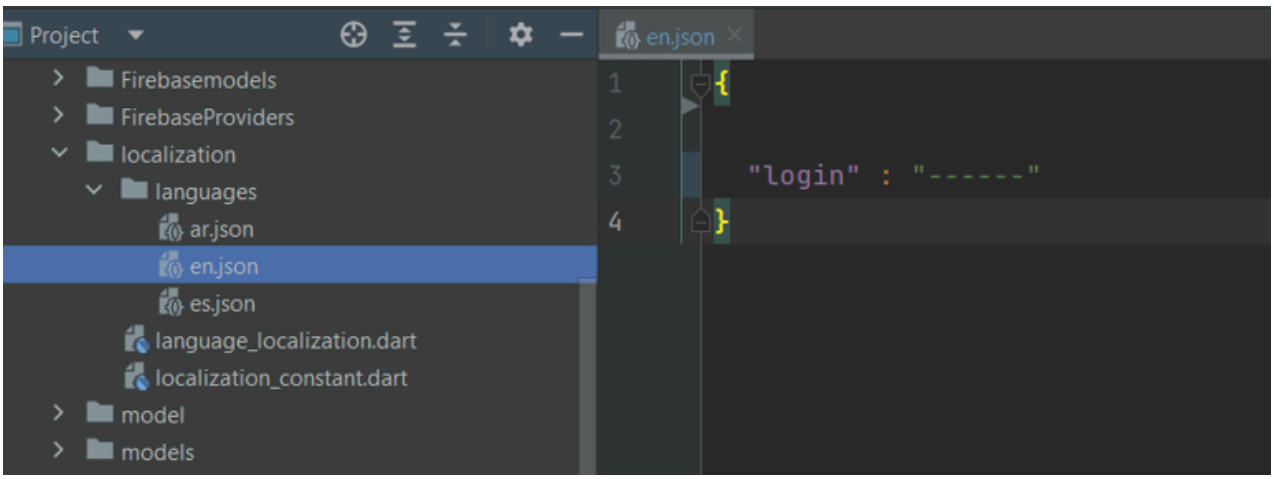
### Step 1:

- Create the JSON file of the language to be added in the following folder:
  - Duplicate `en.json` and rename it based on your language.
  - Folder path: `lib -> localization -> languages`



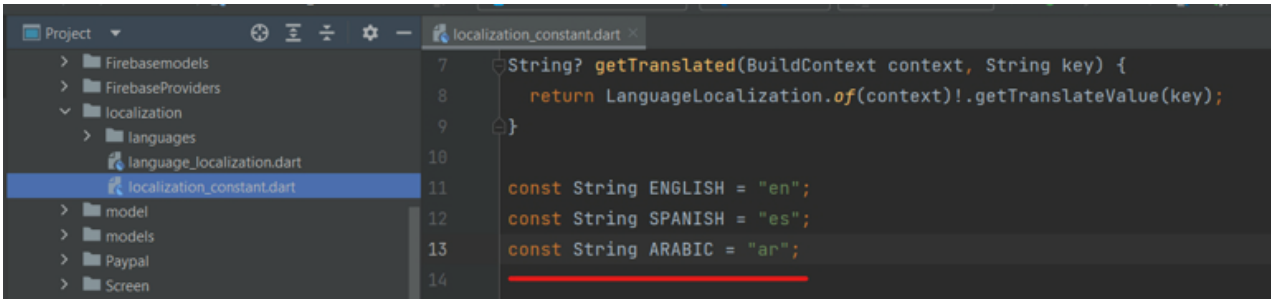

### Step 2:

- Replace all English values with translated values. Change only values, not keys.



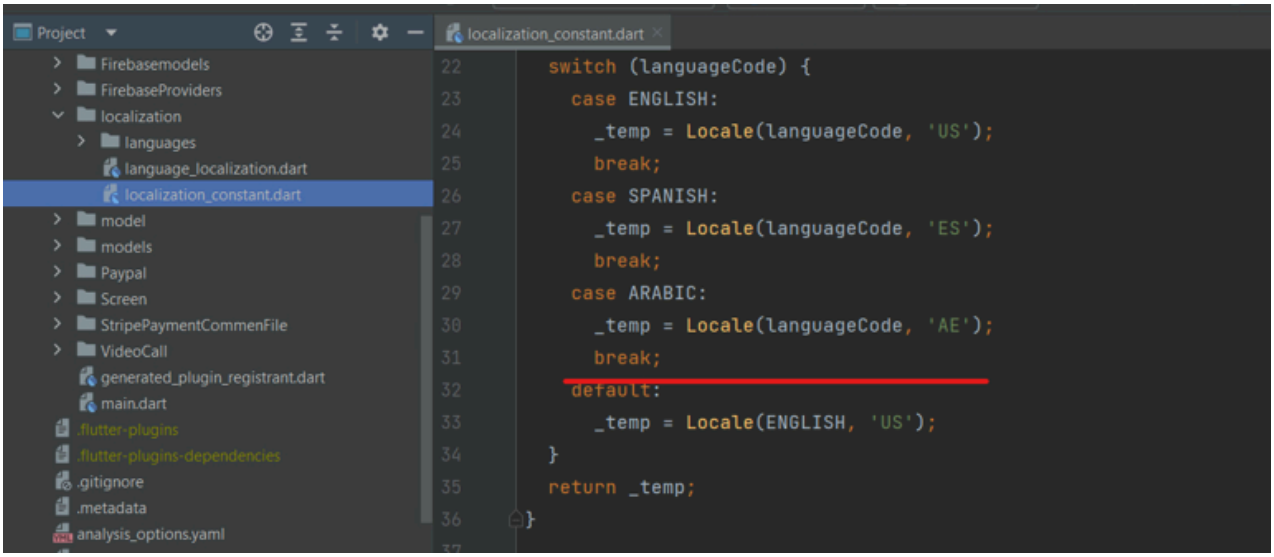

## Step 3:

- Add your language constant variable in `localization_constant.dart`:
  - File path: `lib -> localization -> localization_constant.dart`



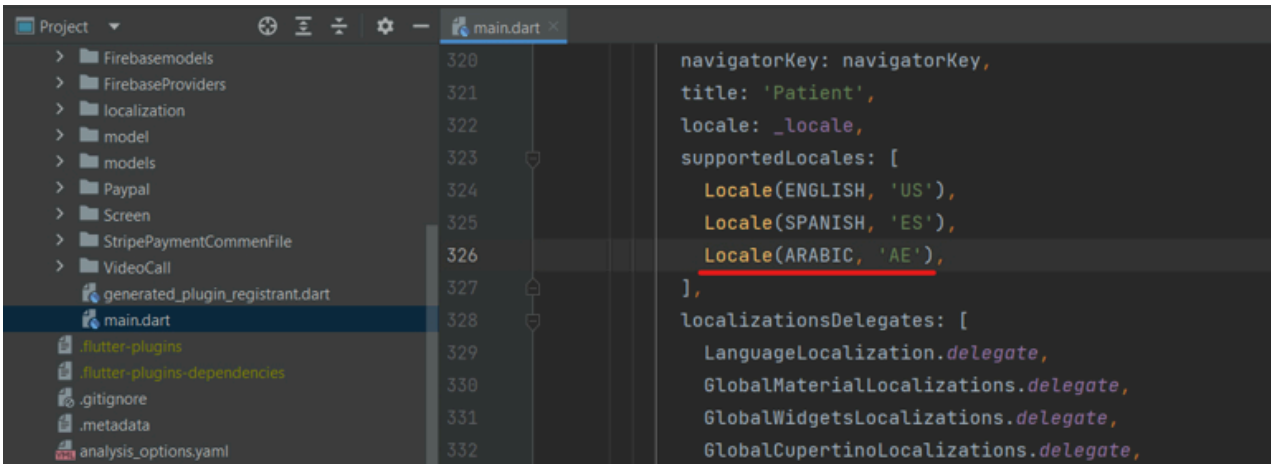

## Step 4:

- Add language-related code to `localization_constant.dart`.



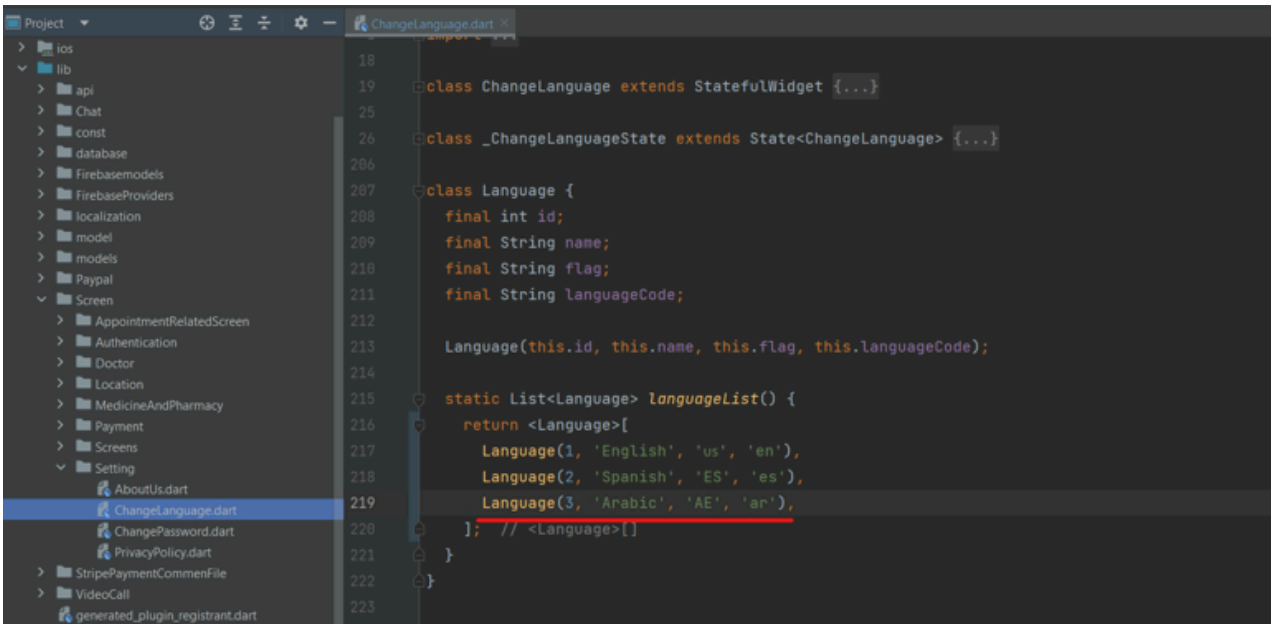
## Step 5:

- Open `main.dart` file and add a new language code with the country code.




## Step 6:

- Add a new language number with a numeric value in `ChangeLanguage.dart`.
  - File path: `lib -> Screen -> Setting -> ChangeLanguage.dart`
  - Example: `Language(4, 'Language Name', 'Country Code', 'Language Code')`

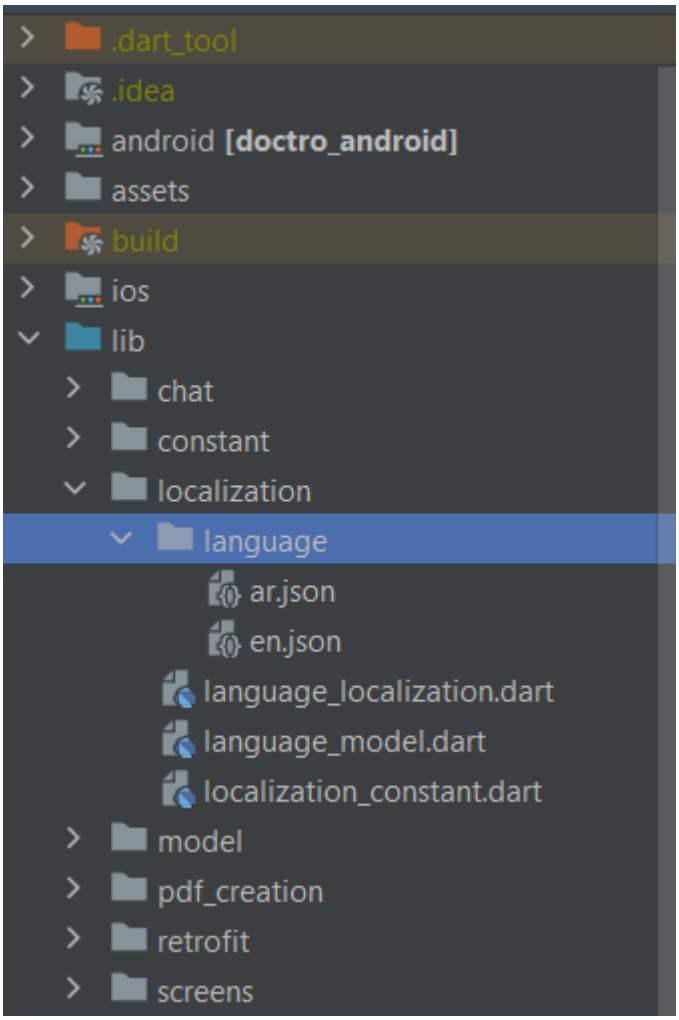



Done for Patient App.

# Doctor App

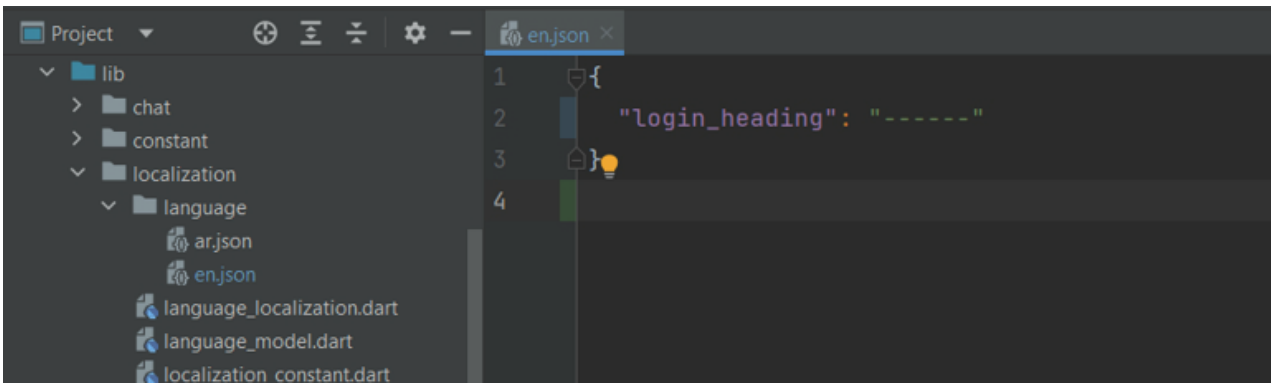
## Step 1:

- Create the JSON file of the language to be added in the following folder:
  - Duplicate `en.json` and rename it based on your language.
  - Folder path: `lib -> localization -> language`



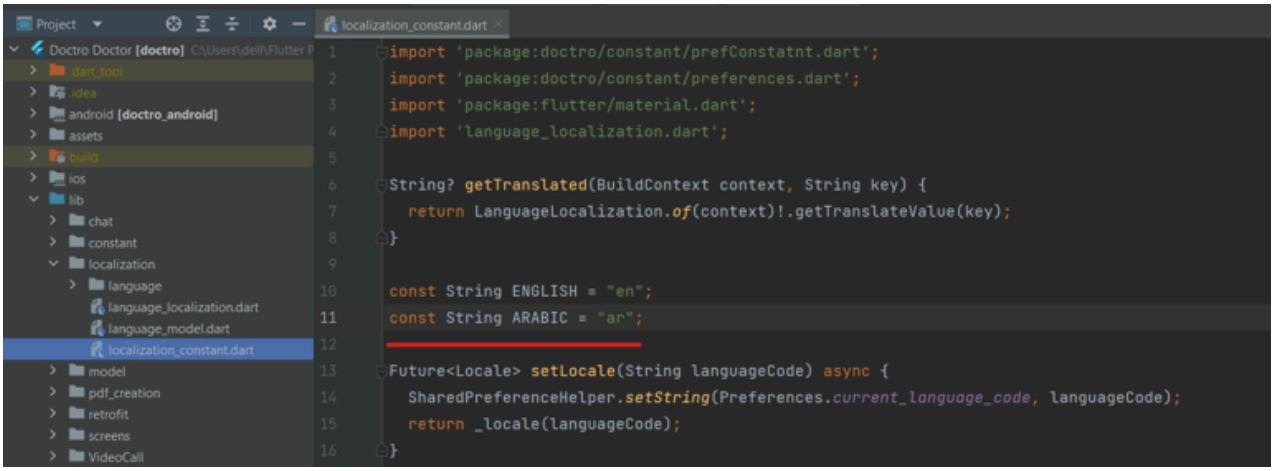

## Step 2:

- Replace all English values with translated values. Change only values, not keys.



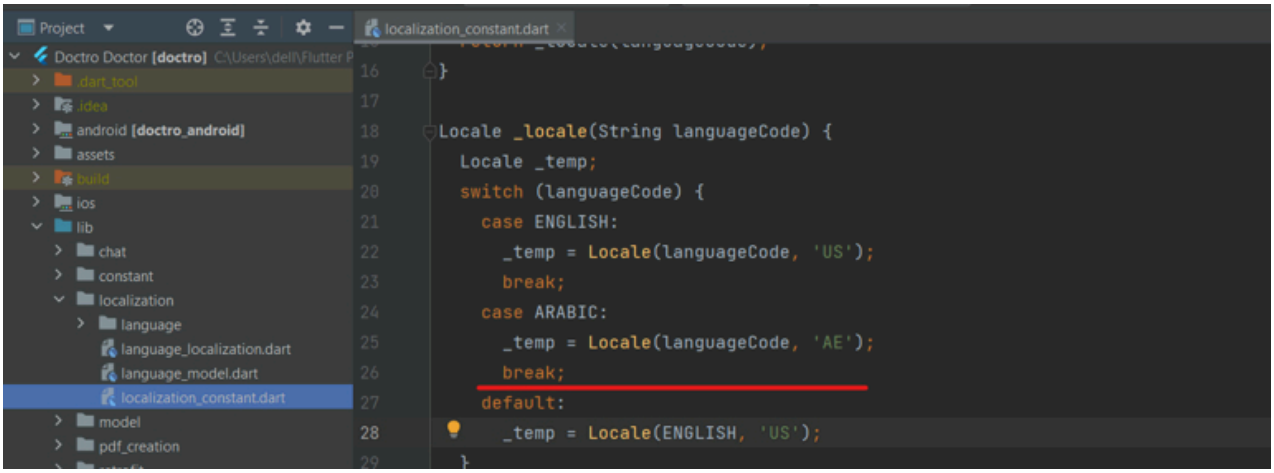

## Step 3:

- Add your language constant variable in `localization_constant.dart`.
  - File path: `lib -> localization -> localization_constant.dart`



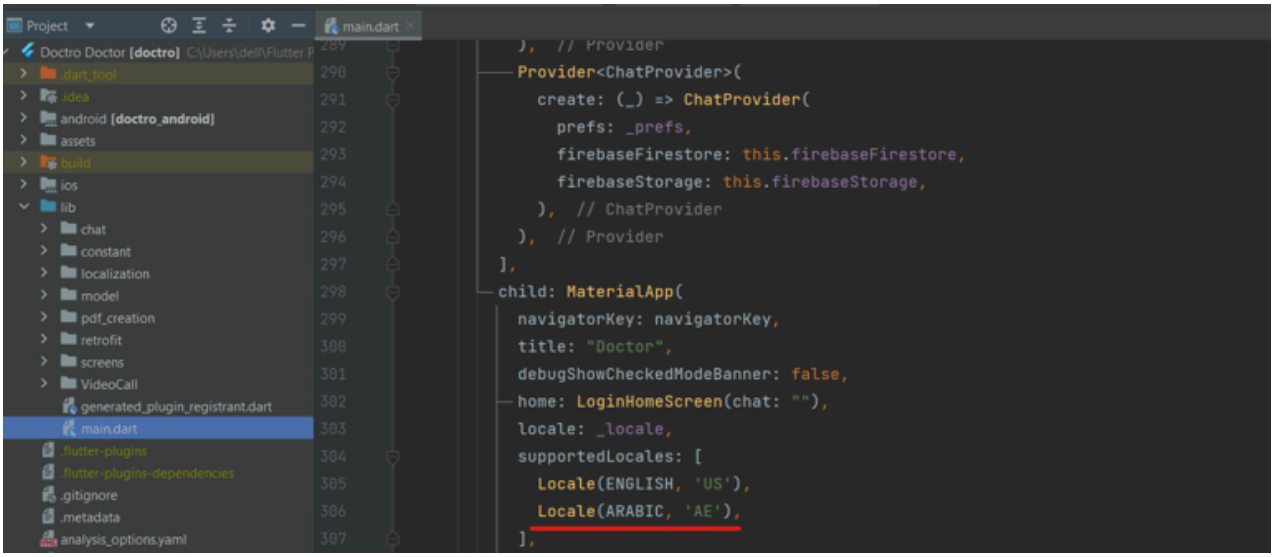

## Step 4:

- Add language-related code to `localization_constant.dart`.



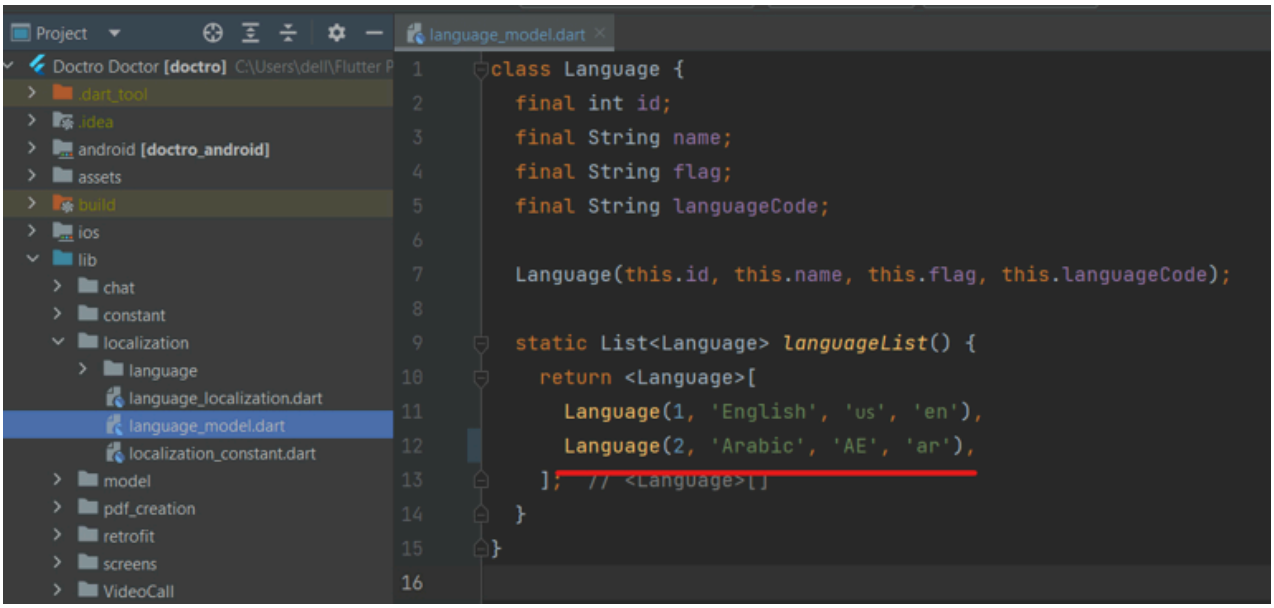
## Step 5:

- Open `main.dart` file and add a new language code with the country code.




## Step 6:

- Add a new language number with a numeric value in `language_model.dart`.
  - File path: `lib -> localization -> language_model.dart`
  - Example: `Language(3, 'Language Name', 'Country Code', 'Language Code')`



Done.

# Run The App

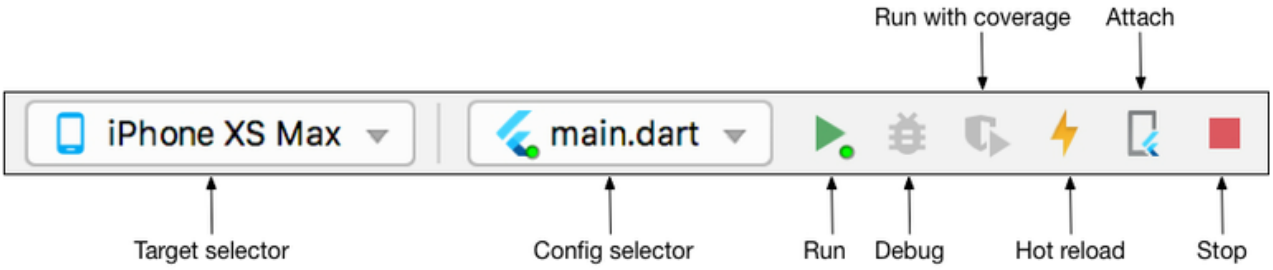
## Using Android Studio

### Open the app files

Select File from top list menu, and chose open folder then select the project folder.

### Run the app

1- Locate the main Android Studio toolbar:




2- In the **target selector**, select an Android device for running the app. If none are listed as available, select **Tools> Android > AVD Manager** and create one there. For details, see Managing AVDs.

3- Click the run icon in the toolbar, or invoke the menu item **Run > Run**.

After the app build completes, you'll see the app on your device.

Starter app

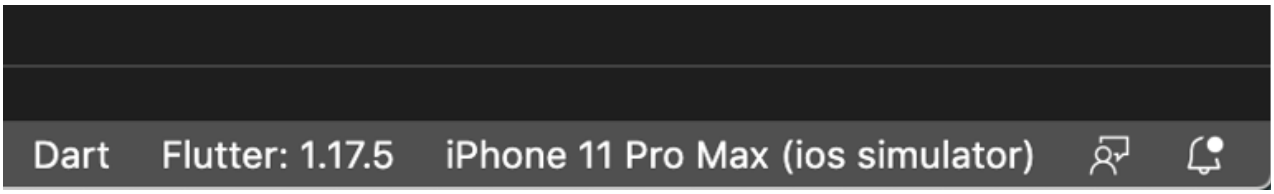
## Using VS Code Studio

## Open the app files

Select File from top list menu, and chose open folder then select the project folder.

## Run the app

1- Locate the VS Code status bar (the blue bar at the bottom of the window):




2- Select a device from the **Device Selector** area. For details, see Quickly switching between Flutter devices.

- If no device is available and you want to use a device simulator, click **No Devices** and launch a simulator.

**Warning:** You may not see **Start iOS Simulator** option when you click **No Devices** in VS Code. If you are on Mac then you may have to run following command in terminal to launch a simulator.

In Android it is not possible to launch iOS simulator.

- To setup a real device, follow the device-specific instructions on the Install page for your OS.

3- Invoke **Run > Start Debugging** or press F5.

4- Wait for the app to launch — progress is printed in the **Debug Console** view.

After the app build completes, you'll see the app on your device.

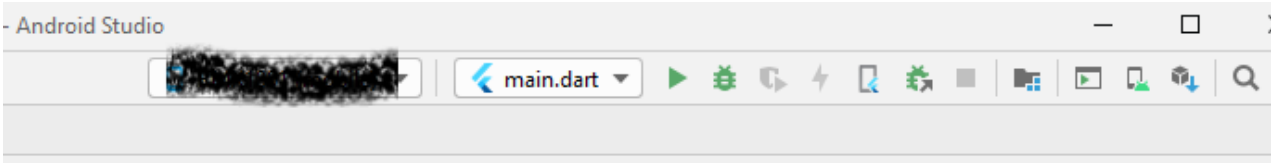
Starter app

# Install App To Your Android Device

## Install app to your android device

Steps for: How to install application in android device

1. Connect your android device with OTG supported cable.
2. Open setting and turn on Developer option and turn on USB Debugging.
3. In android studio you can see your device connected at right hand side of screen.
4. then you need to just click on Green color play button to run it. it will take 2-4 minute as per your system configuration.



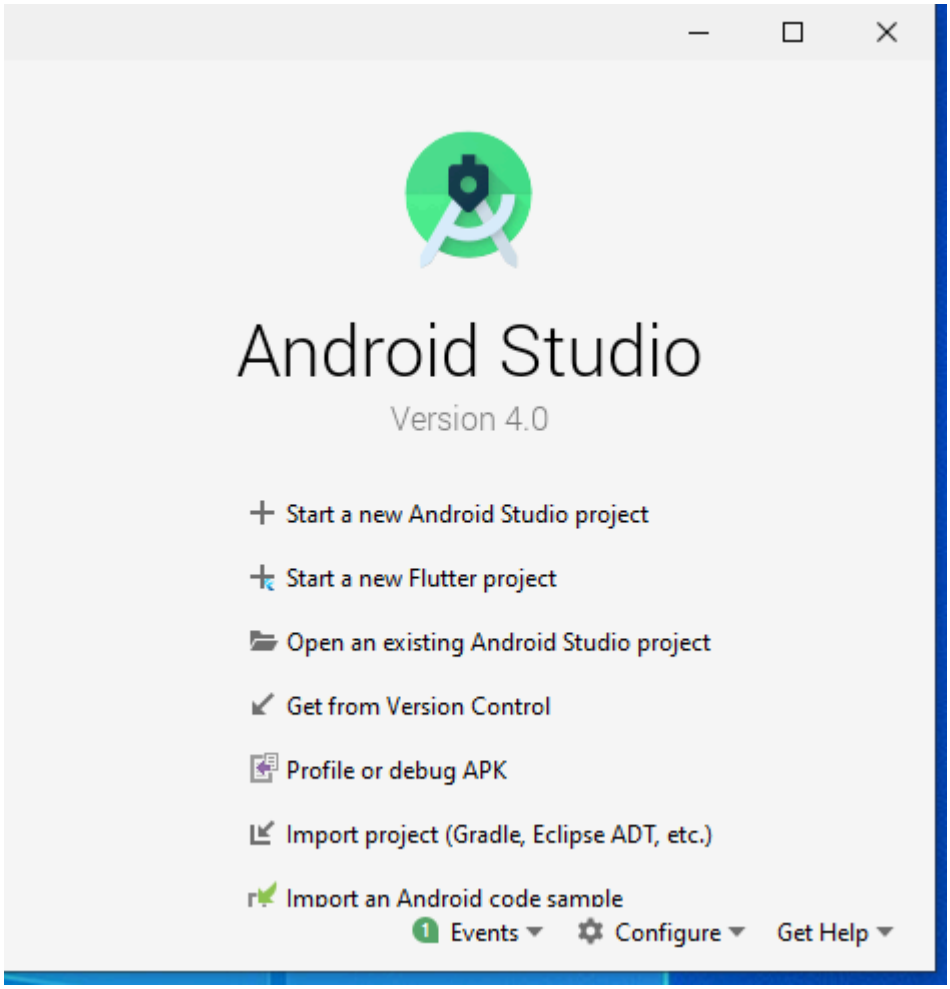
# How to Generate APK file

In the project folder there will have 2 folders located 1st for user and 2nd for driver, to generate both(user, driver) apk follow these steps.

1. at bottom of android studio : TODO, Dart Analysis, Terminal buttons available
2. click on Terminal button
3. execute command : - flutter build apk --target-platform android-arm,android-arm64 --split-per-abi
4. after 3 to 5 minute you can get APK file.

# How to Generate Signed APK

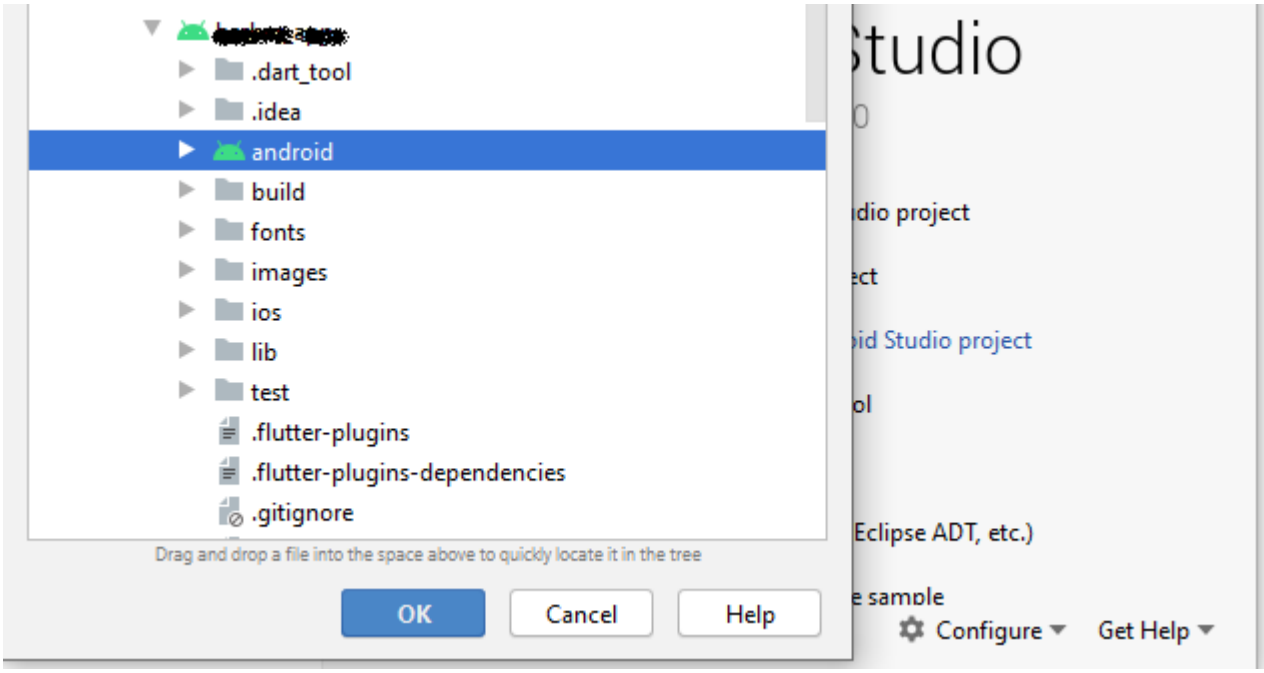
1. Open Android studio




2. Open an existing Android Studio project

3. Go to specific path of your code : Doctro(Patient) → android

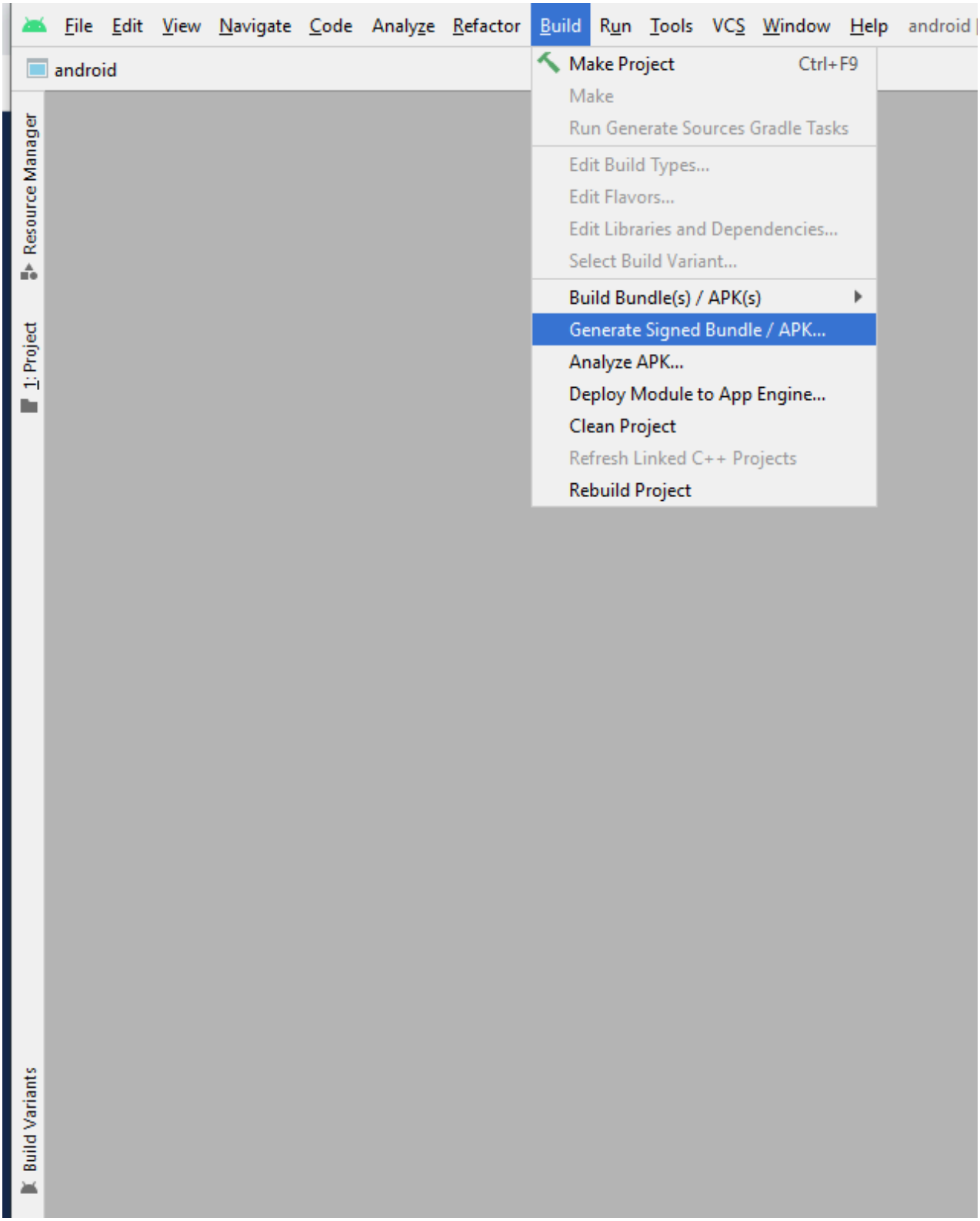
4. Select android



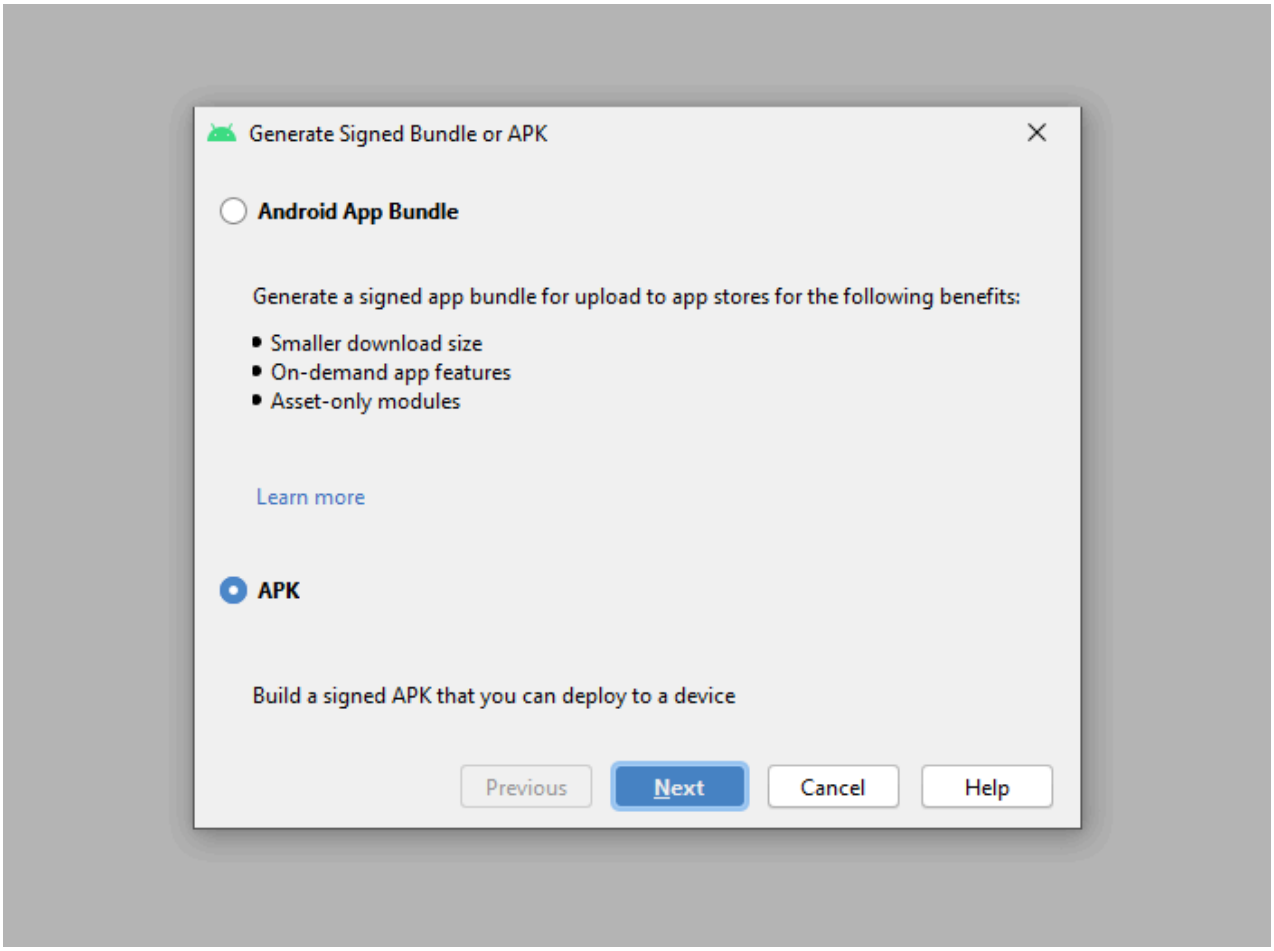

Click OK

Wait while build finish successfully.

After build finish :



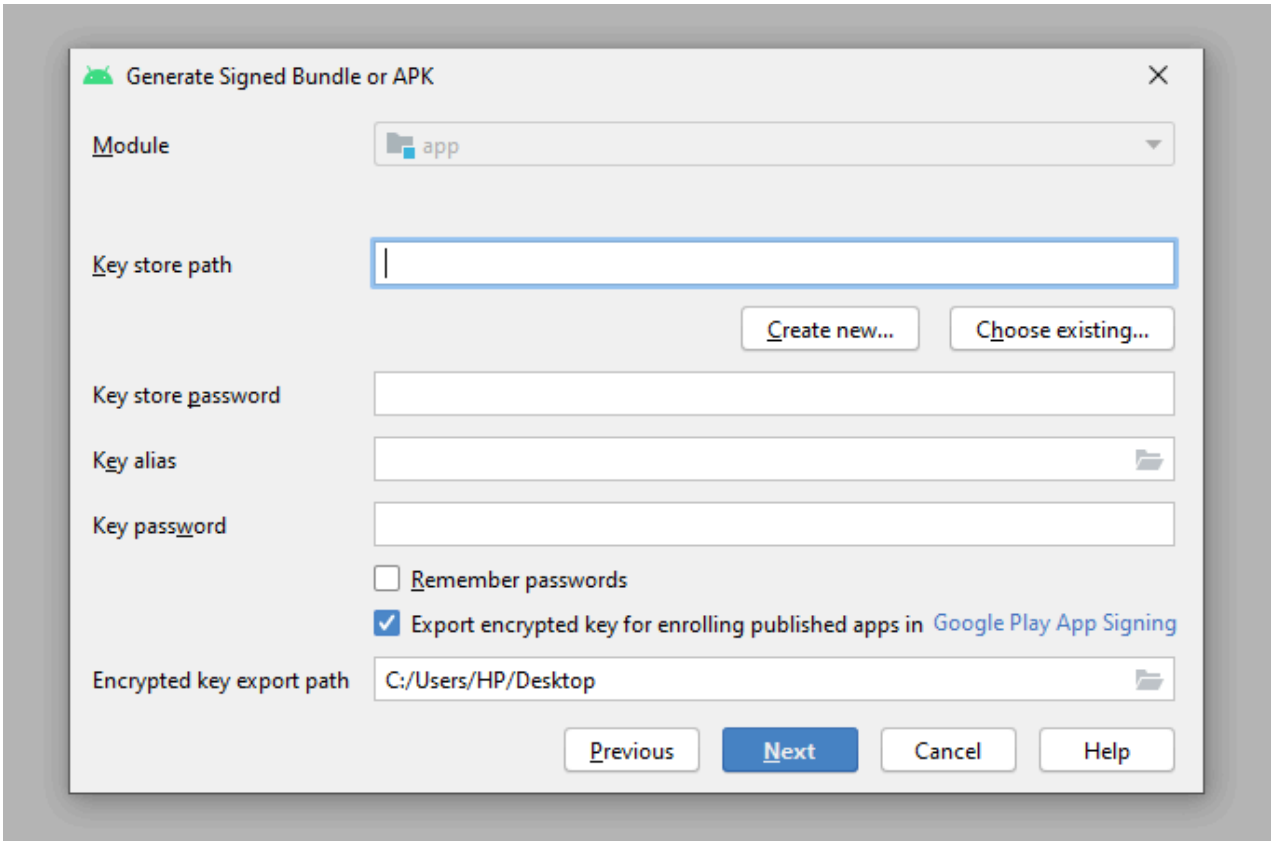

Select Build Menu → Generate Signed Bundle/APK..




After Selecting Generate Signed Bundle/APK.. You can see above dialog box.

Here you have 2 option

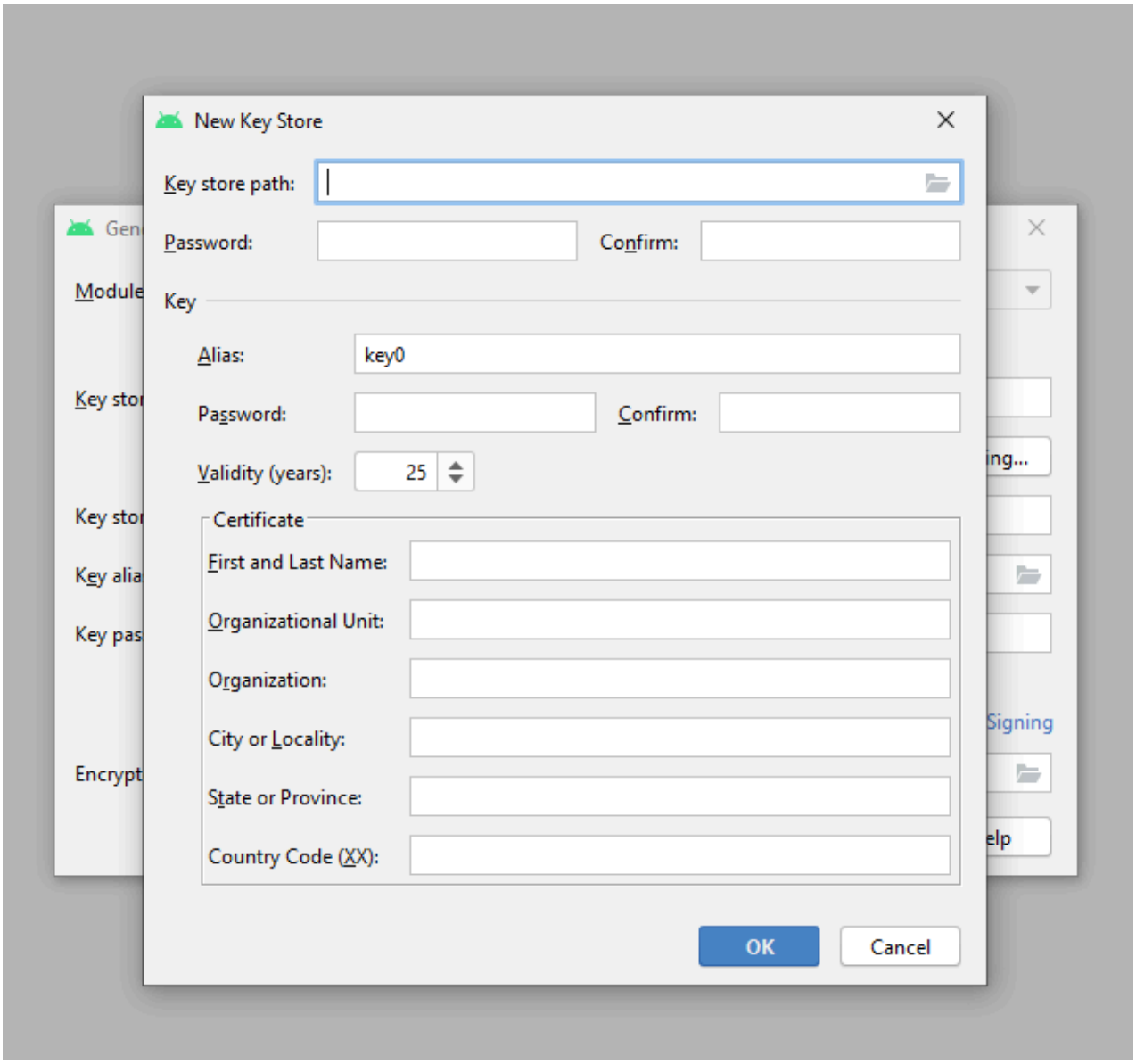
1. Android App Bundle
2. APK
3. Select App Bundle to Generate Signed APK.




You can see above dialog box :

If you are Generate first time signed apk at that time you need CREATE NEW KEYSTORE PATH

so that first click on Create new... After click create new... , You can see below dialog box




Fill Necessary Details : Key store path , Keystore Password Keystore confirm Password, KEY Alias, Alias Password, Alias confirm password, validity(years), Certificate: First & Last Name

OTHER Details are optional.

Select key store path :

NOTE : (Select a specific path where you want to store KEY STORE File. This file will use while every time when you want to publish apk or Update application to google play store. So that keep it safe location and save it.)

Set Password and Confirm Password

Key----------

Alias: Set Alias Name

Alias Password: Set Alias Password

Alias Confirm Password: Set Alias Confirm Password

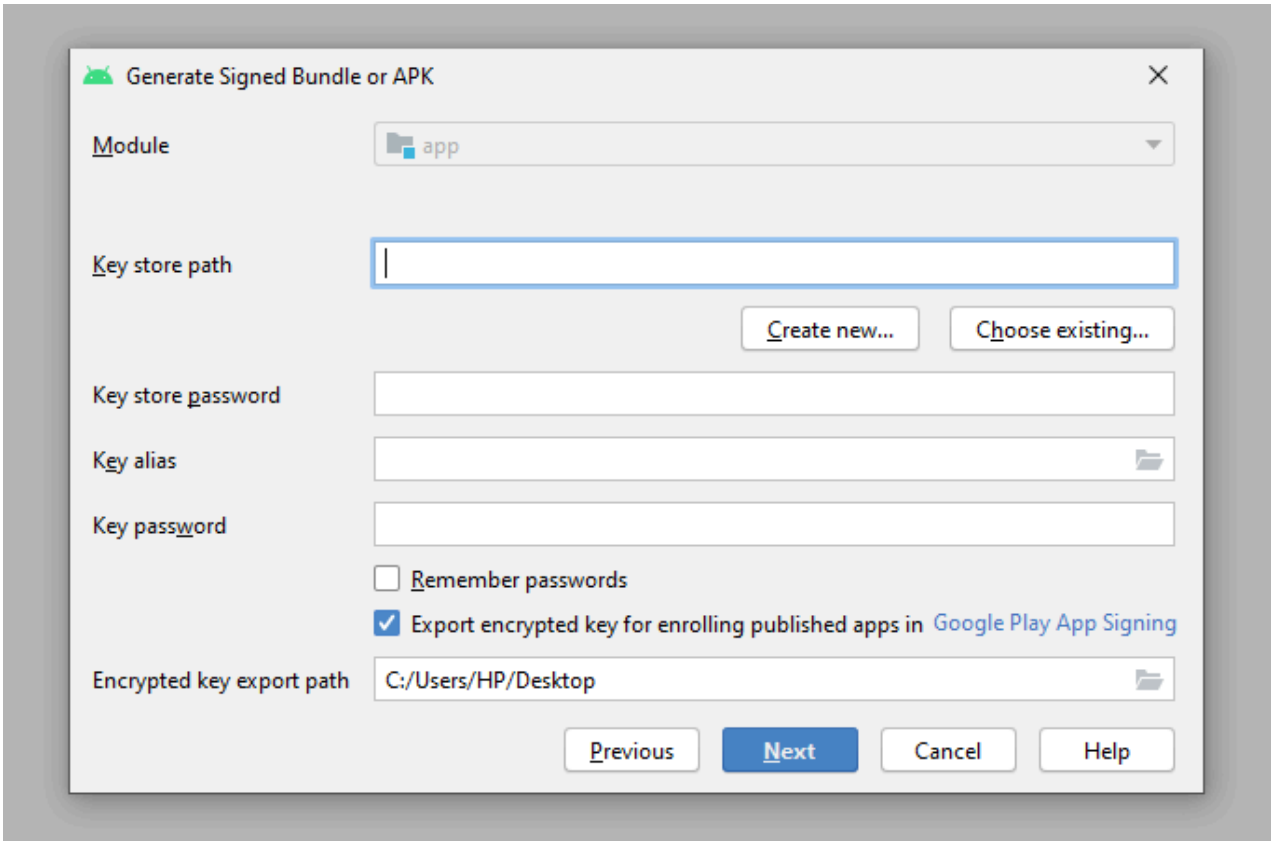
Validity : in years

Certificate :

First name and Last Name:

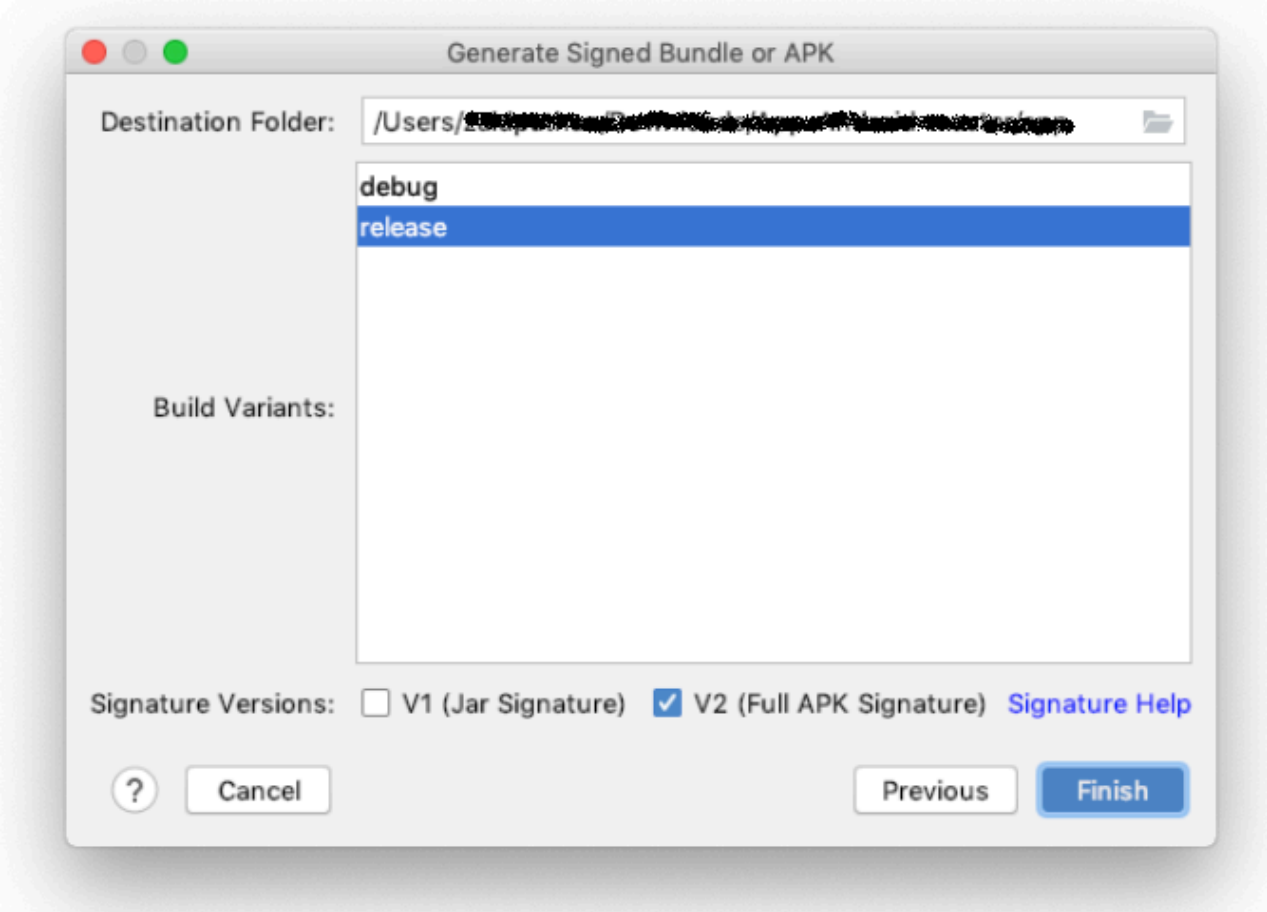
AFTER Fill UP All Details Click OK

Then you can see previous dialog box with keystore path filled as your given data.




Enter Keystore password , key alias , and key Password as you given while creating keystore path.

Click NEXT




Here you need to select build Variants : release

Signature versions : V1 and V2

You can select Signature version for secure apk file from reverse engineering.

Click Finish.

Generating Signed apk take 3-4 minute to Generate Signed APK.

# How to publish APK to Google play store

## How to publish APK to Google play store

Here steps for Android application published on the Google play store:

To publish your application on the google play store you need to have a Developer Account to publish your android application on the google play store.

Create a Developer account using the below link:

https://play.google.com/apps/publish

The First Step to publish APK to google play store, We need to generate Signed APK. (Regular Build APK can not be published on the google play store.)

If you have knowledge of how to generate a signed APK then you can go throw the below link

# Upload APK/App bundle to Google play store

After Successfully Generation of Signed APK, you can publish it on Google play store.

As pervious mentioned you must have Google developer / google play store account to complete the process of it. Create and set up your app - Play Console Help Starting August 2021, new apps will be required to publish with the Android App Bundle on Google Play.

New apps larger than 150MB can use either Play Asset Delivery or Play Feature Delivery. Reasupport.google.com

**Thankyou**

PreviousHow to publish APK to Google play store