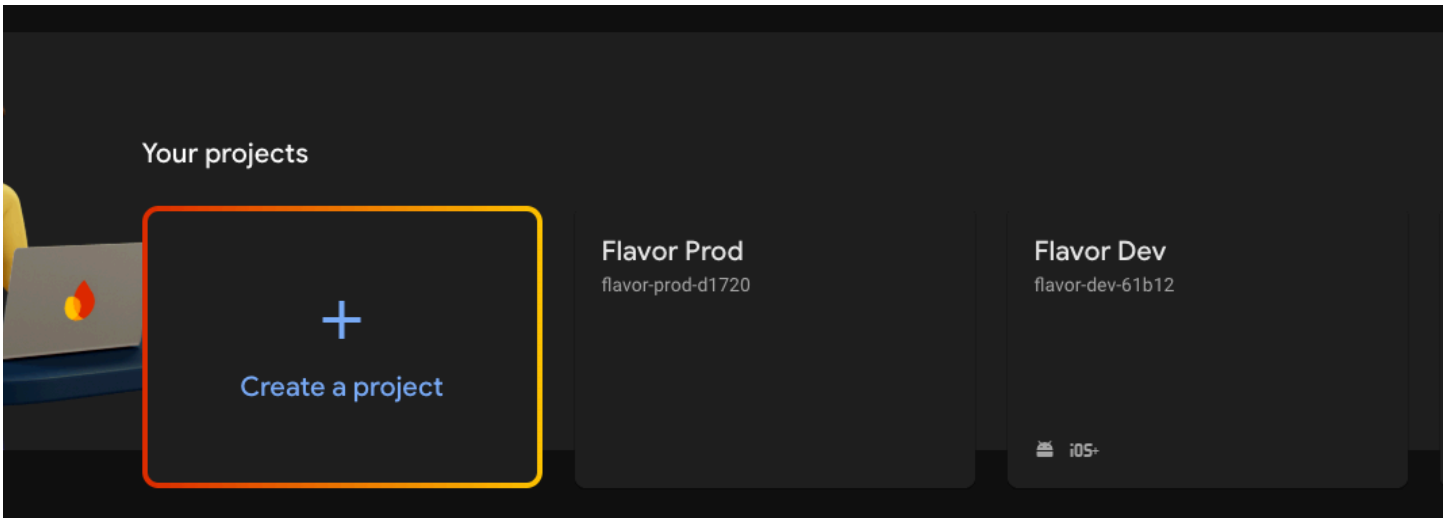
Flutter Flavor Setup

Step 1: Create a new project.

Step 2: Configure the app to connect to Firebase

Create the app for iOS and Android in the Firebase console and download the `GoogleService-Info.plist` and `google-services.json` respectively.

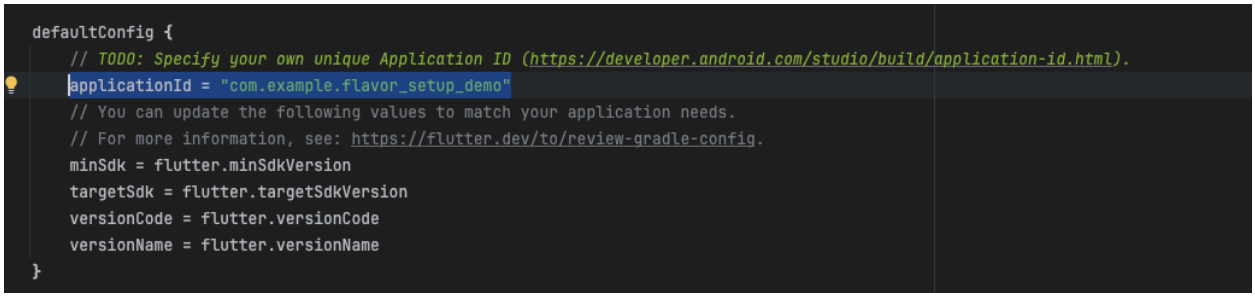


Step 3: Setup Build Flavor for Android

1. **Adding build flavors to Android**

Add flavors to `app/build.gradle` and it looks something like this

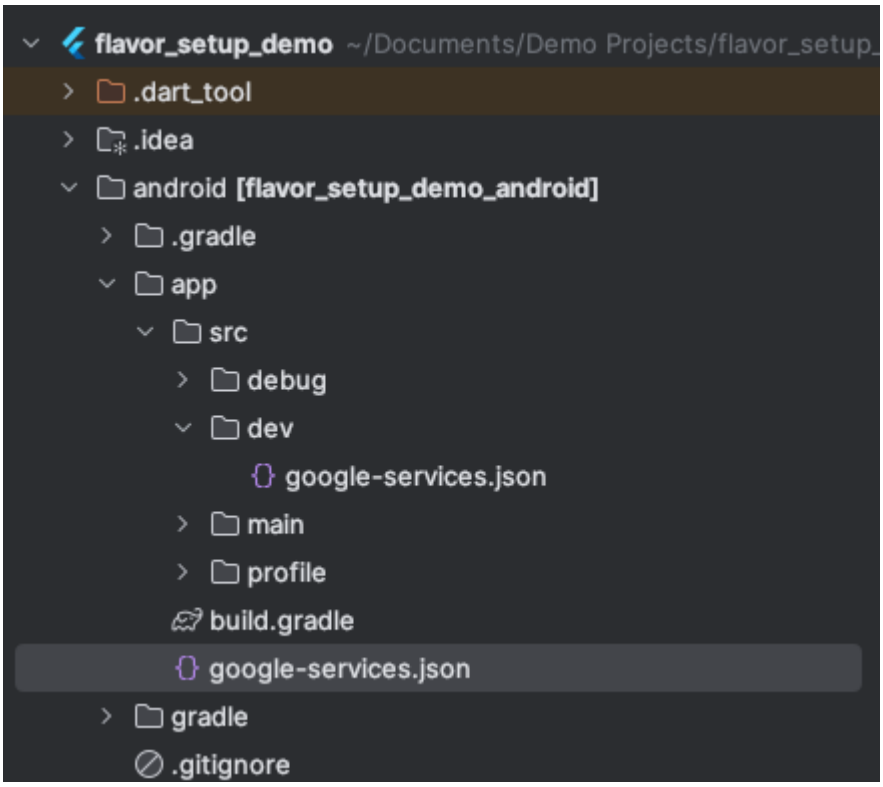
```
flavorDimensions "default"
productFlavors {
    prod {
        dimension "default"
        resValue "string", "app_name", "Flavor Demo"
    }
    dev {
        dimension "default"
        applicationIdSuffix ".dev"
        resValue "string", "app_name", "Dev Flavor Demo"
        versionNameSuffix ".dev"
    }
}
```



As you see, my applicationId is `com.example.flavor_setup_demo` so when my flavor is dev then the `applicationIdSuffix` will add at the end i.e `"com.example.flavor_setup_demo.dev"`

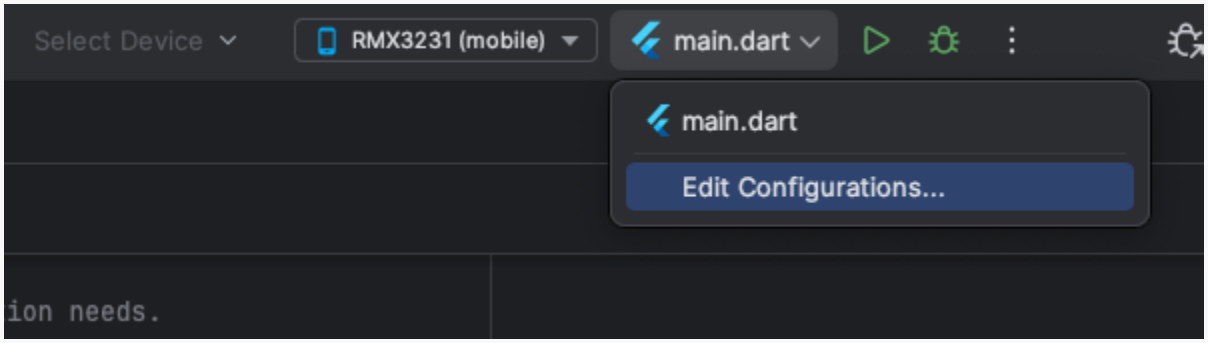
2. Organizing Firebase Config for Dev and Prod in Android

Create a folder **'dev'** in your project android/app/src, and add your android dev firebase project's `google-service.json` file. And add your prod `google-service.json` file in the android/app folder.

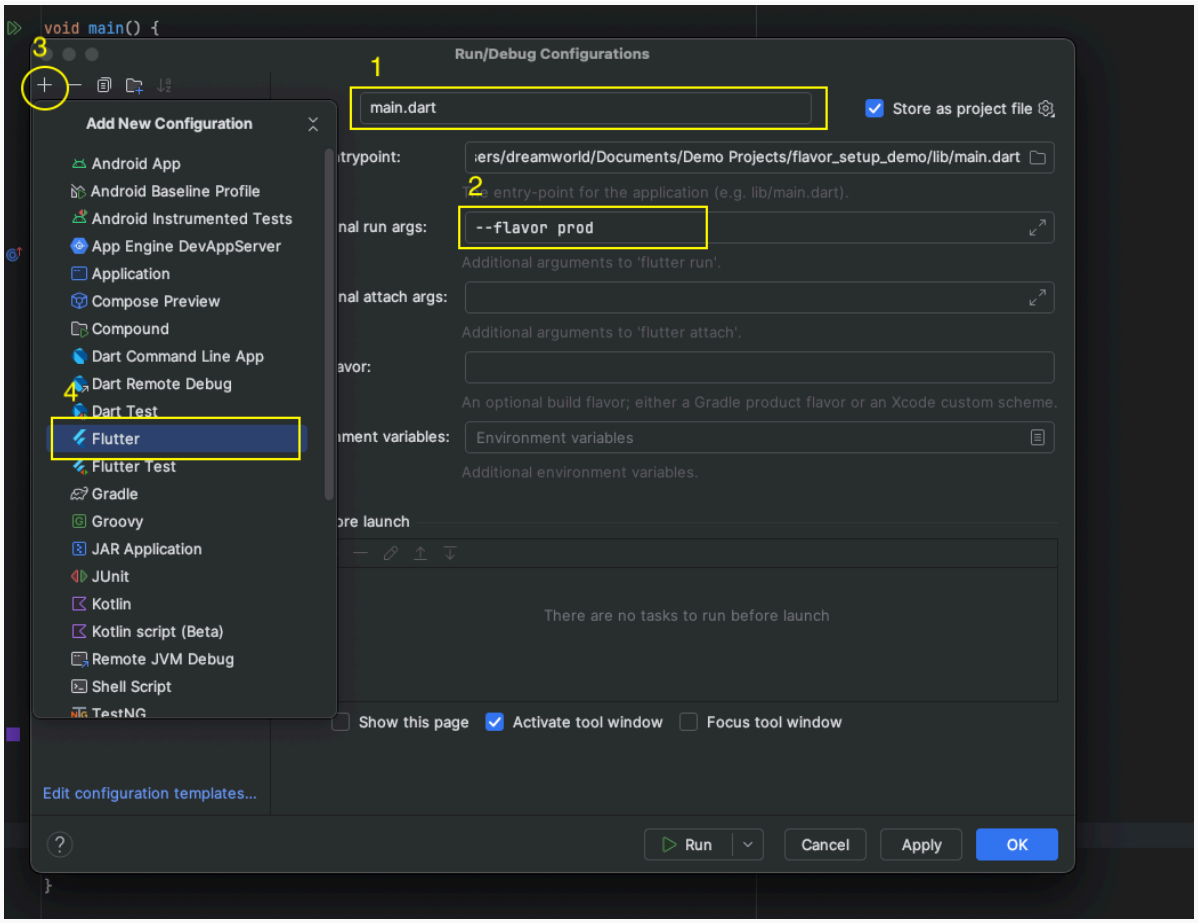


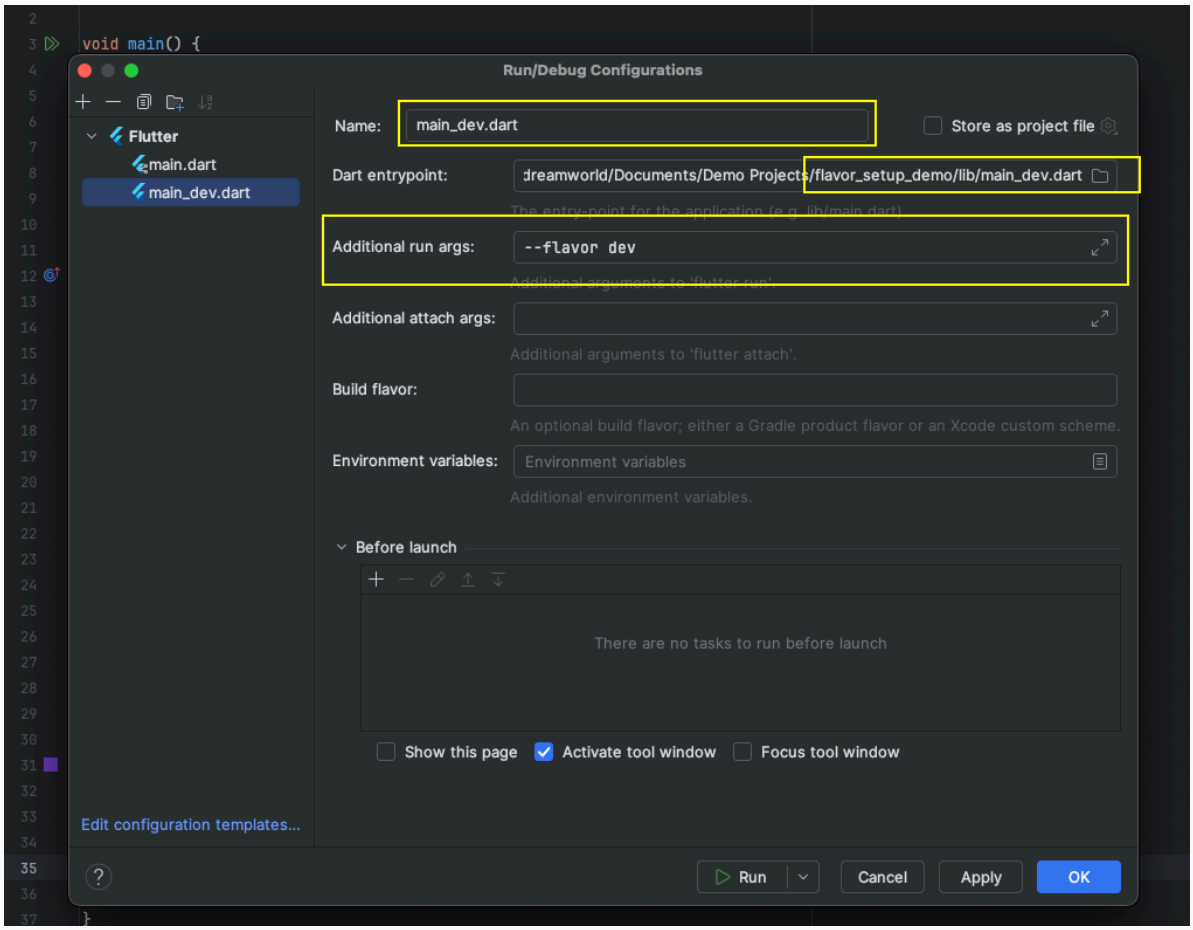
3. Setting Up a New Flutter Run Configuration for Development

Create a **main_dev.dart** file in your lib folder. Go to **Edit Configuration** -> Add New Configuration (+) -> Select Flutter -> Give name as main_dev.dart -> Select the path of main_dev.dart.



For main.dart : --flavor prod
For main_dev.dart : --flavor dev





Then Apply → Ok.

Step 4: Create AppConfig File

Create a class `AppConfig` this singleton class is useful for storing flavor based configurations

```
import 'package:flutter/material.dart';

enum Flavor { prod, dev }

class AppConfig {
 String appName = "";
 String baseUrl = "";
 MaterialColor primaryColor = Colors.blue;
 Flavor flavor = Flavor.dev;

 static AppConfig shared = AppConfig.create();

 factory AppConfig.create({
   String appName = "",
   String baseUrl = "",
   MaterialColor primaryColor = Colors.blue,
   Flavor flavor = Flavor.dev,
 }) {
   return shared = AppConfig(appName, baseUrl, primaryColor, flavor);
 }

 AppConfig(this.appName, this.baseUrl, this.primaryColor, this.flavor);
}
```

Step 5: Create Entry Point Each Flavor

Entry point for Prod

In `lib/main.dart` file, you will define the flavor type and give the app a specific name, base url, primary color etc for the production environment.

```
void main() async {
 AppConfig.create(
   appName: "Prod Flavor Example",
   baseUrl: "https://apiurl.com",
   primaryColor: Colors.yellow,
   flavor: Flavor.prod,
 );

 runApp(const MyApp());
}
```

Entry Point for Dev

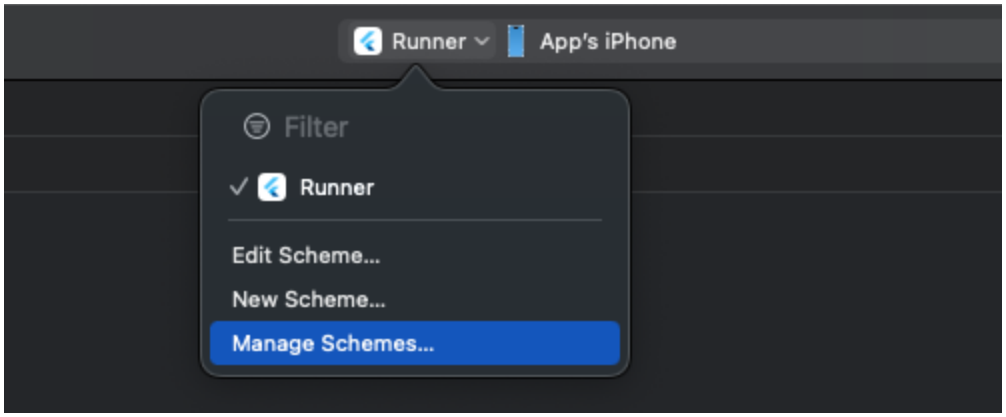
In `lib/main_dev.dart` file, you will define the flavor type and give the app a specific name, base url, primary color etc for the staging environment.

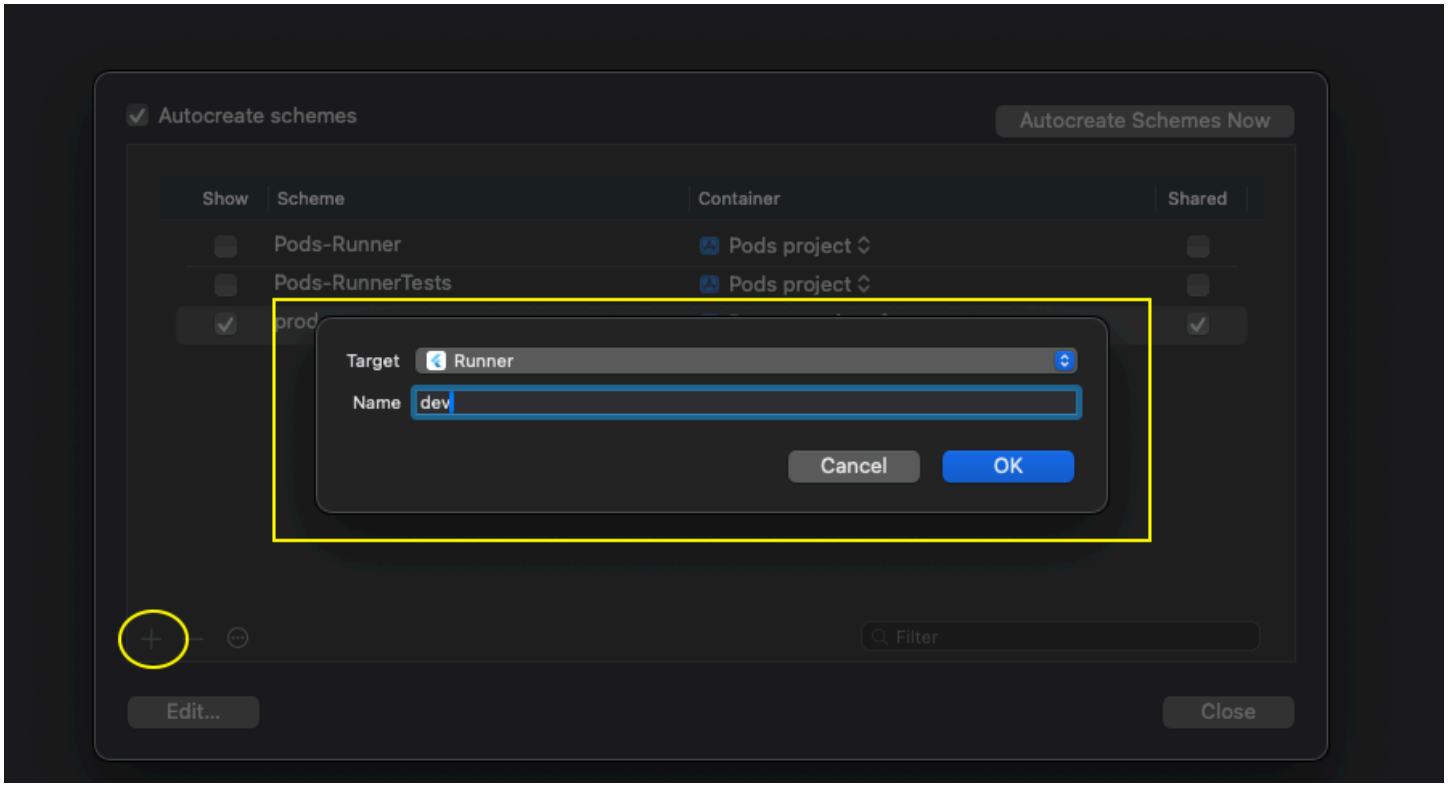
```
void main() async {
```

```
  AppConfig.create(
    appName: "Dev Flavor Example",
    baseUrl: "https://dev.apiurl.com",
    primaryColor: Colors.blue,
    flavor: Flavor.dev,
  );


  runApp(const MyApp());
}
```

Step 6: Setup Build Flavor for iOS

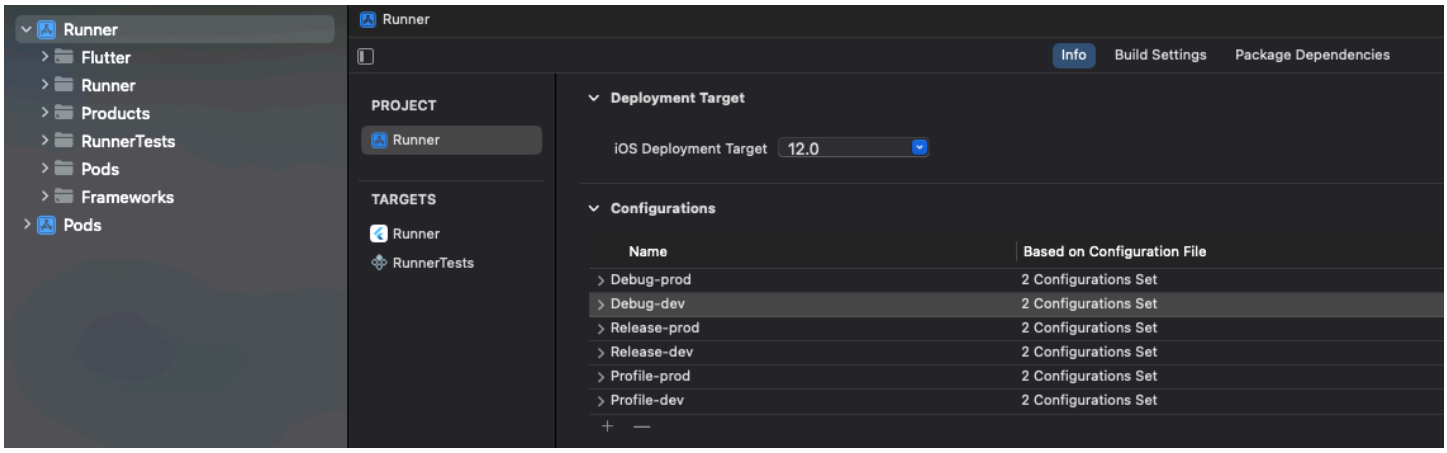
1. **Open your project in Xcode.**
2. **Go to Manage Schemes** → Change the name **Runner** to **prod** → **(+) Add** a new Scheme → select Target **Runner** → Name as **'dev'.**



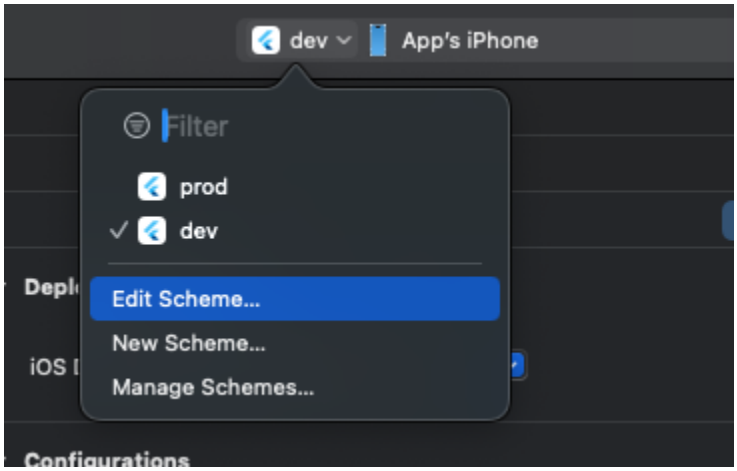


3. **Duplicating Build Configurations with '-dev' Suffix**

Go to Runner -> Info -> Configurations -> Add the duplicate of Debug, Release, Profile and set the suffix -dev to all like `Debug-dev`

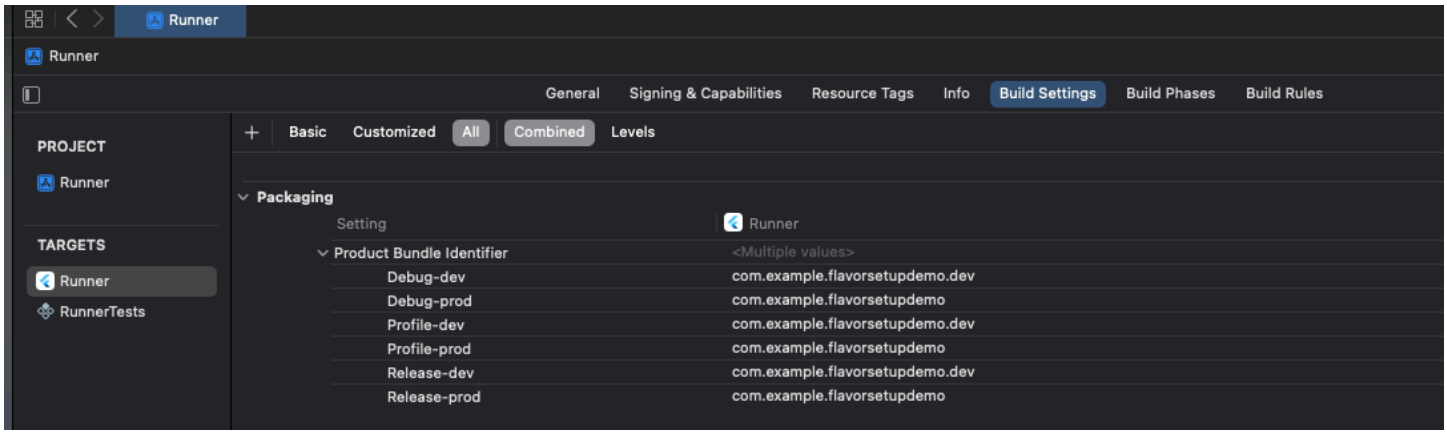


4. **Change the configurations for prod and dev by Edit Configuration.**



5. **Set the Product Bundle Identifier**

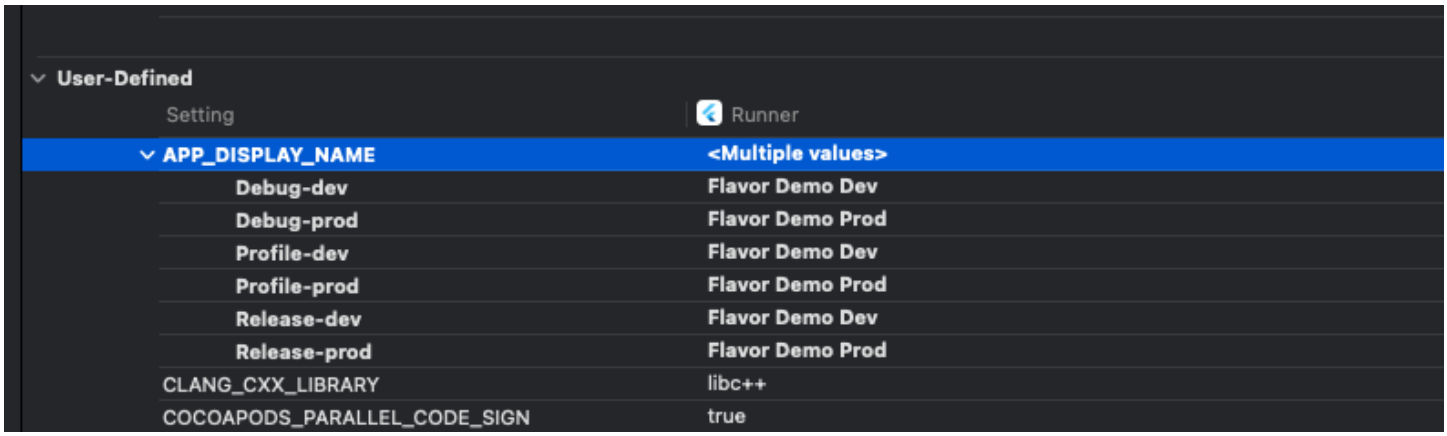
Go to Target Runner -> Build Setting -> Search the **Product Bundle Identifier** → Set the bundle name for dev and prod.



6. **Set App Display Name for different flavor**

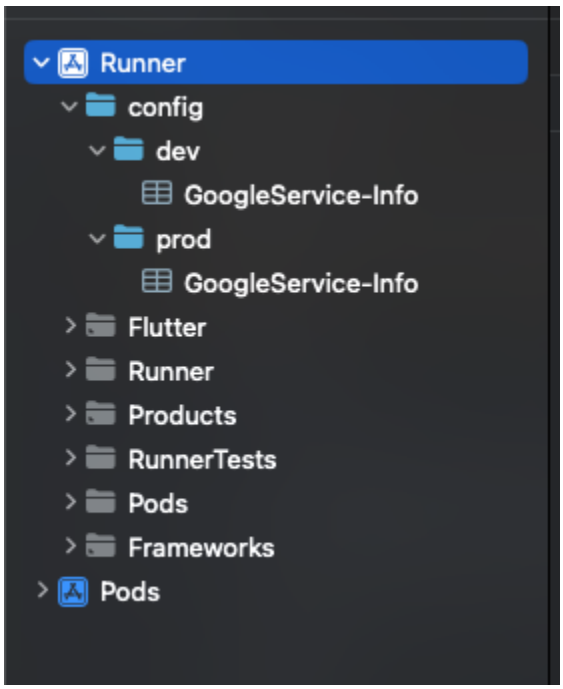
Now on **Build Setting** → Create **User Defined Setting** → Give the name as `**APP_DISPLAY_NAME**` and add the name of your project that you want to show to the user. And then set **$(APP_DISPLAY_NAME)** to info.plist file as:

```
<key>CFBundleDisplayName</key>
<string>$(APP_DISPLAY_NAME)</string>
```



7. Setting Up Firebase Config for iOS with Dev and Prod Flavors

Now make a folder inside the IOS folder as config -> then create 2 folders inside that as prod and dev -> and add your GoogleService-Info.plist file for particular flavor.

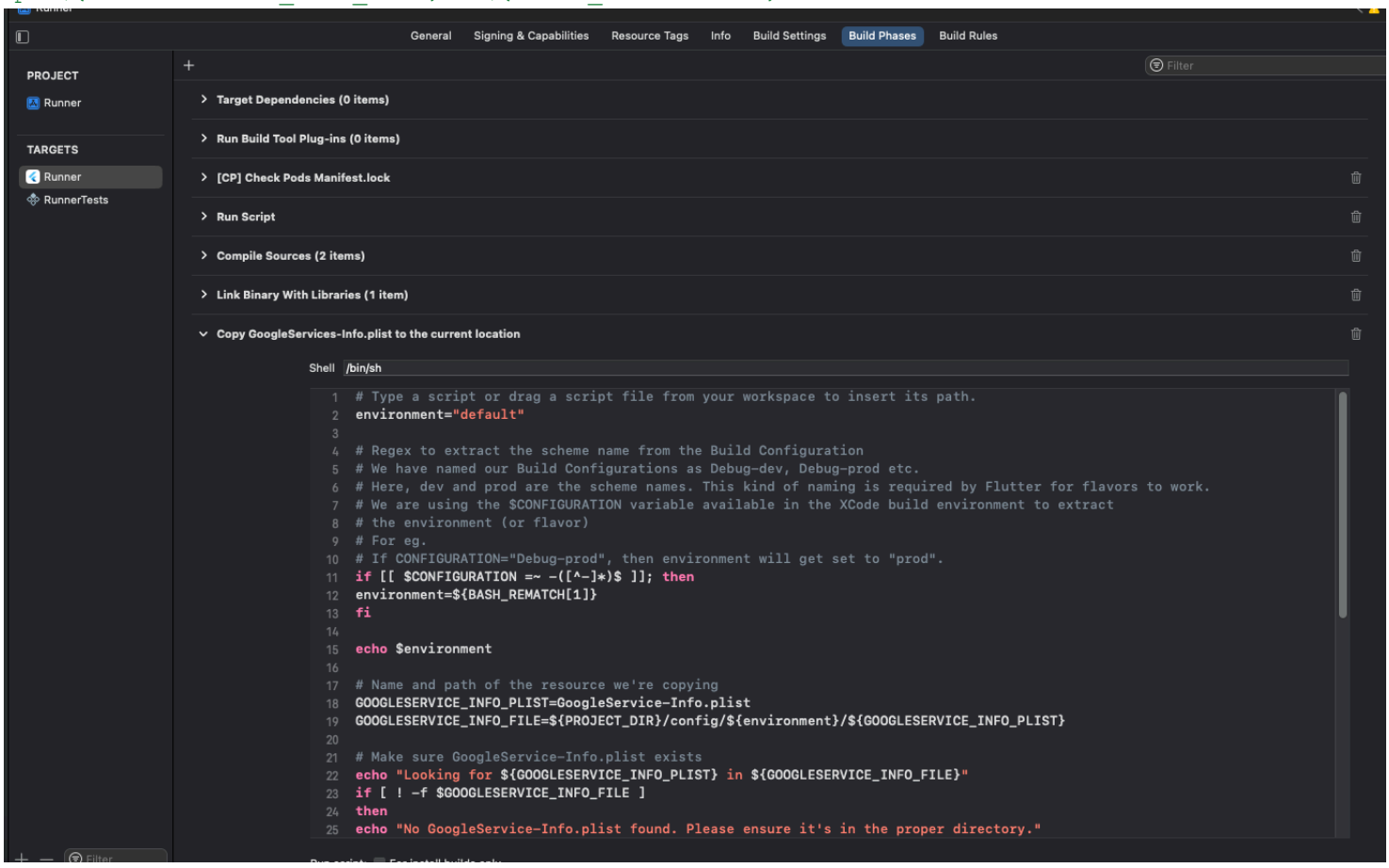


8. Adding a Run Script to Copy GoogleServices-Info.plist in iOS

Go to Build Phases → Create New Run Script name as `Copy GoogleServices-Info.plist to the current location` → This Run script should be next to **Link Binary With Libraries.**
Add this script.

```
environment="default"

# Regex to extract the scheme name from the Build Configuration
# We have named our Build Configurations as Debug-dev, Debug-prod etc.
```

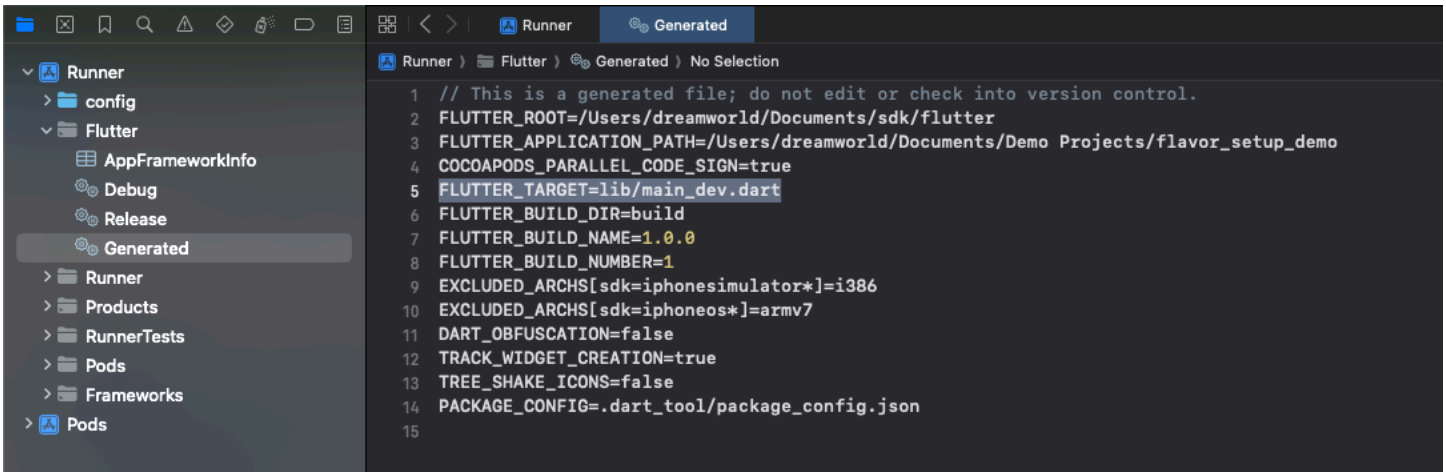
```
# Here, dev and prod are the scheme names. This kind of naming is required by Flutter for
flavors to work.
# We are using the $CONFIGURATION variable available in the XCode build environment to
extract
# the environment (or flavor)
# For eg.
# If CONFIGURATION="Debug-prod", then environment will get set to "prod".
if [[ $CONFIGURATION =~ -([^-]*)$ ]]; then
environment=${BASH_REMATCH[1]}
fi

echo $environment

# Name and path of the resource we're copying
GOOGLESERVICE_INFO_PLIST=GoogleService-Info.plist
GOOGLESERVICE_INFO_FILE=${PROJECT_DIR}/config/${environment}/${GOOGLESERVICE_INFO_PLIST}

# Make sure GoogleService-Info.plist exists
echo "Looking for ${GOOGLESERVICE_INFO_PLIST} in ${GOOGLESERVICE_INFO_FILE}"
if [ ! -f $GOOGLESERVICE_INFO_FILE ]
then
echo "No GoogleService-Info.plist found. Please ensure it's in the proper directory."
exit 1
fi

# Get a reference to the destination location for the GoogleService-Info.plist
# This is the default location where Firebase init code expects to find
GoogleServices-Info.plist file
PLIST_DESTINATION=${BUILT_PRODUCTS_DIR}/${PRODUCT_NAME}.app
echo "Will copy ${GOOGLESERVICE_INFO_PLIST} to final destination: ${PLIST_DESTINATION}"

# Copy over the prod GoogleService-Info.plist for Release builds
cp "${GOOGLESERVICE_INFO_FILE}" "${PLIST_DESTINATION}"
```



9. Manually Setting the FLUTTER_TARGET for iOS Configuration

In Android, selecting main_dev.dart ensures the dev configuration runs, while selecting main.dart sets the production data. However, in iOS, you must manually specify the main file in **Runner** → **Flutter** → **Generated** by setting:

FLUTTER_TARGET = lib/{main_dev.dart for dev, main.dart for prod}



Run command

- debug dev `flutter run -t lib/main_dev.dart --flavor=dev`

- release dev `flutter run -t lib/main_dev.dart --release --flavor=dev`

- debug prod `flutter run -t lib/main.dart --flavor=prod`

- release prod `flutter run -t lib/main.dart --release --flavor=prod`

Build command

- Android
  - dev `flutter build apk -t lib/main_dev.dart --flavor=dev`

  - prod `flutter build apk -t lib/main.dart --flavor=prod`

- iOS
  - dev `flutter build ios -t lib/main_dev.dart --flavor=dev`

  - prod `flutter build ios -t lib/main.dart --flavor=prod`

Source code Link

https://drive.google.com/file/d/1IMAR6_nJ7md7oBiUQqc4pBv2rTyW5SPY/view?usp=sharing