

EXPERIMENT NO. 6

Aim: To Set Up Firebase with Flutter for iOS and Android Apps

Theory:

Integrating Firebase with your Flutter app unlocks various functionalities like authentication, databases, cloud storage, messaging, and analytics. Here's a theoretical breakdown of the process:

1. Firebase Project and Configuration:

- Create a Firebase project: This establishes your project space in the Firebase console.
- Enable desired services: Choose the specific Firebase services (e.g., Authentication, Firestore, Storage) you want to integrate.
- Configure platform settings: Provide platform-specific configurations like app bundle IDs for iOS and package names for Android.
- Download configuration files: Obtain platform-specific files (GoogleService-Info.plist for iOS, google-services.json for Android) containing API keys and project IDs.

2. Flutter Project Setup:

- Add Firebase plugins: Use the flutter pub add command to install necessary plugins for each Firebase service you're using.
- Configure plugins: Run flutterfire configure command to integrate downloaded configuration files into your Flutter project.
- Initialize Firebase: Add code in your main.dart file to initialize Firebase with the project configuration.

3. Using Firebase Services:

- Authentication: Implement user login/signup flows using Firebase Authentication methods (email/password, social logins, etc.).
- Databases: Store and retrieve data using Firebase Realtime Database or Cloud Firestore (flexible, NoSQL database).
- Cloud Storage: Upload and manage files (images, videos) in Firebase Storage.
- Messaging: Send push notifications to users using Firebase Cloud Messaging.
- Analytics: Track user behavior and app usage with Firebase Analytics.

4. Platform-Specific Considerations:

- Android: Integrate Firebase libraries into your Android project and handle platform-specific permissions.
- iOS: Integrate Firebase SDK with your Xcode project and manage capabilities (e.g., push notifications).

Theoretical Advantages:

- Simplified development: Firebase offers pre-built tools and APIs, reducing development time.
- Cross-platform support: Firebase works seamlessly on both iOS and Android with consistent APIs.
- Scalability and security: Firebase scales automatically and handles security aspects like authentication and data encryption.

Steps to Set Up Firebase:

Prerequisites

To complete this tutorial, you will need:

- A Google account to use Firebase.
- Developing for iOS will require XCode.
- To download and install Flutter.
- To download and install Android Studio and Visual Studio Code.
- It is recommended to install plugins for your code editor

1. Creating a New Flutter Project

This tutorial will require the creation of an example Flutter app.

Once you have your environment set up for Flutter, you can run the following to create a new application:

```
PS C:\Users\Mayuresh\Desktop\Projects\flutter> flutter create newapp
Creating project newapp...
Resolving dependencies in newapp... (1.9s)
Got dependencies in newapp.
Wrote 129 files.

All done!
You can find general documentation for Flutter at: https://docs.flutter.dev/
Detailed API documentation is available at: https://api.flutter.dev/
If you prefer video documentation, consider: https://www.youtube.com/c/flutterdev

In order to run your application, type:

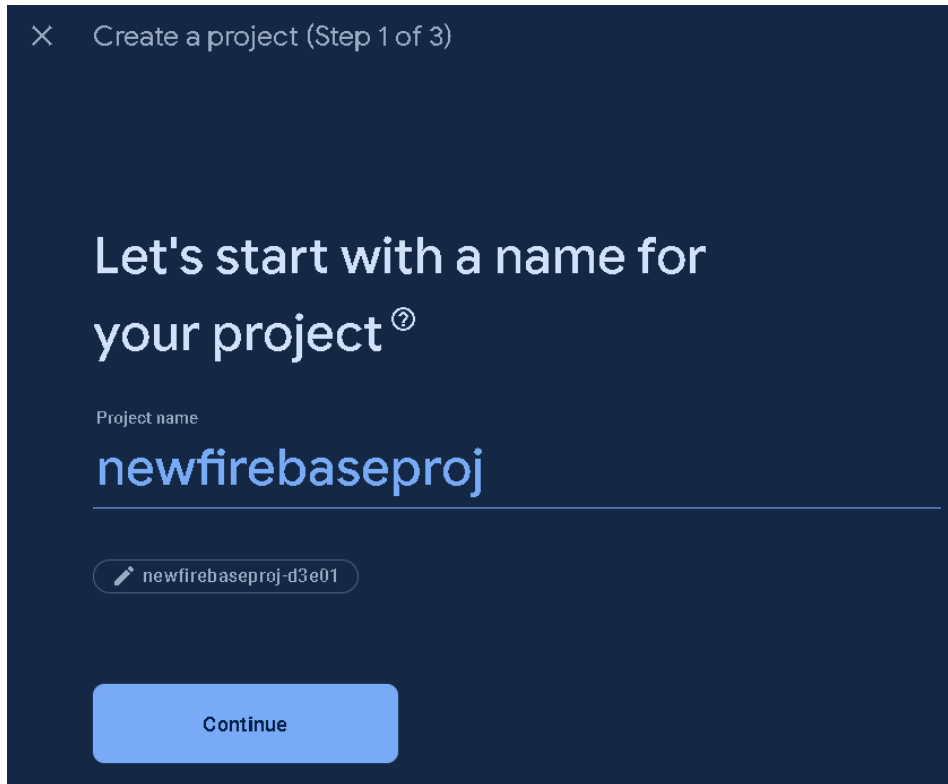
$ cd newapp
$ flutter run

Your application code is in newapp\lib\main.dart.

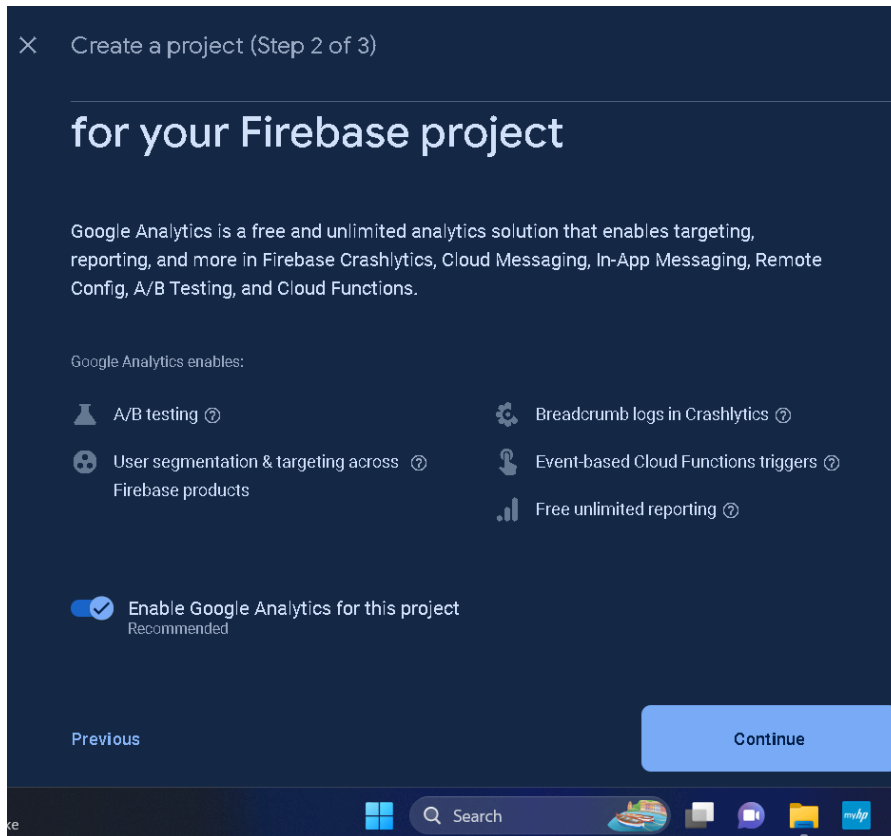
PS C:\Users\Mayuresh\Desktop\Projects\flutter> cd newapp
PS C:\Users\Mayuresh\Desktop\Projects\flutter\newapp>
```

2. Creating a New Firebase Project

First, log in with your Google account to manage your Firebase projects. From within the Firebase dashboard, select the Create new project button and give it a name:

A screenshot of the Firebase 'Create a project' dialog box. The dialog has a dark blue background. At the top left, there is a close button (X) and the text 'Create a project (Step 1 of 3)'. The main heading is 'Let's start with a name for your project' with a help icon. Below this, the label 'Project name' is followed by the input field containing 'newfirebaseproj'. Underneath the input field is a preview bar showing a pencil icon and the text 'newfirebaseproj-d3e01'. At the bottom, there is a blue 'Continue' button.

Next, we're given the option to enable Google Analytics. This tutorial will not require Google Analytics, but you can also choose to add it to your project.



If you choose to use Google Analytics, you will need to review and accept the terms and conditions prior to project creation.

After pressing Continue, your project will be created and resources will be provisioned.

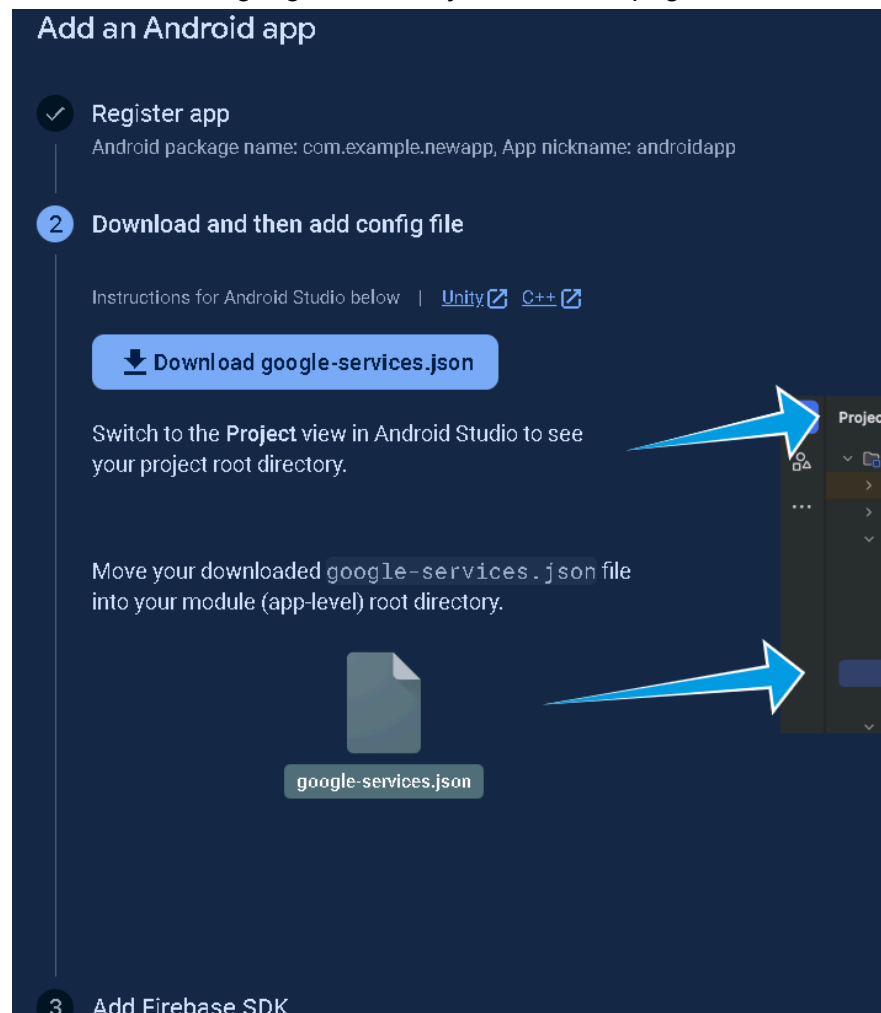
You will then be directed to the dashboard for the new project.

3. Adding Android support

1. Registering the App

In order to add Android support to our Flutter application, select the Android logo from the dashboard. This brings us to the following screen:

Select Download google-services.json from this page:



Next, move the google-services.json file to the android/app directory within the Flutter project.

3. Adding the Firebase SDK

We'll now need to update our Gradle configuration to include the Google Services plugin.

3 Add Firebase SDKInstructions for Gradle | [Unity](#) [C++](#)

★ Are you still using the `buildscript` syntax to manage plugins? [Learn how to add Firebase plugins](#) using that syntax.

1. To make the `google-services.json` config values accessible to Firebase SDKs, you need the Google services Gradle plugin.

☐ Kotlin DSL (`build.gradle.kts`) ☒ Groovy (`build.gradle`)

Add the plugin as a dependency to your **project-level** `build.gradle` file:

Root-level (project-level) Gradle file (`<project>/build.gradle`):

```
plugins {
    // ...

    // Add the dependency for the Google services Gradle plugin
    id 'com.google.gms.google-services' version '4.4.1' apply false
}
```

2. Then, in your **module (app-level)** `build.gradle` file, add both the `google-services` plugin and any Firebase SDKs that you want to use in your app:

Module (app-level) Gradle file (`<project>/<app-module>/build.gradle`):

```
plugins {
    id 'com.android.application'
    // Add the Google services Gradle plugin
    id 'com.google.gms.google-services'
    ...
}

dependencies {
    // Import the Firebase BoM
    implementation platform('com.google.firebase:firebase-bom:32.7.2')

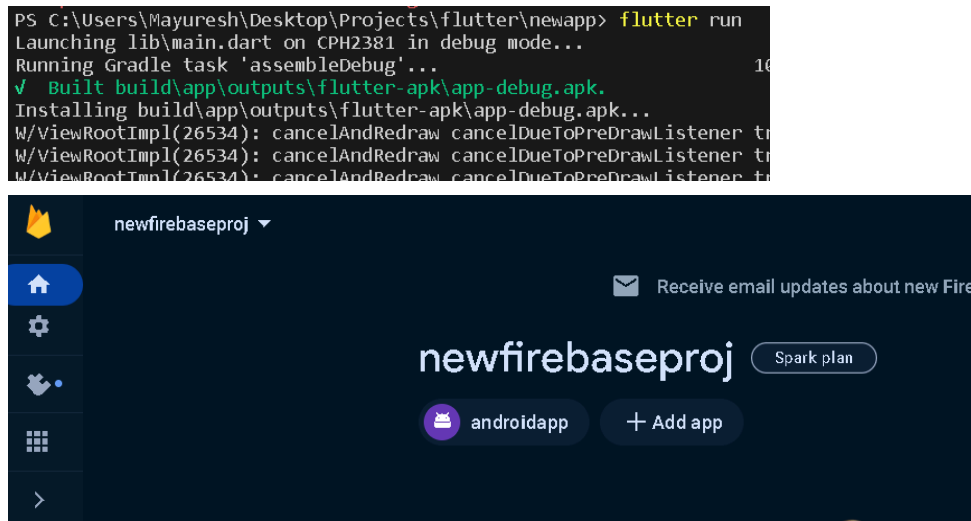
    // TODO: Add the dependencies for Firebase products you want to use
    // When using the BoM, don't specify versions in Firebase dependencies
    implementation 'com.google.firebase:firebase-analytics'

    // Add the dependencies for any other desired Firebase products
    // https://firebase.google.com/docs/android/setup#available-libraries
}
```

By using the Firebase Android BoM, your app will always use compatible Firebase library versions. [Learn more](#)

With this update, we're essentially applying the Google Services plugin as well as looking at how other Flutter Firebase plugins can be activated such as Analytics.

From here, run your application on an Android device or simulator.



Next up, let's add iOS support!

4. Adding iOS Support

In order to add Firebase support for iOS, we have to follow a similar set of instructions. Head back over to the dashboard and select Add app and then iOS icon to be navigated to the setup process.

Registering an App

Once again, we'll need to add an "iOS Bundle ID". It is possible to use the "Android package name" for consistency:

You'll then need to make sure this matches up by opening the iOS project up in Xcode at ios/Runner/Runner.xcodeproj and changing the Bundle identifier under General:

Click Register app to move to the next screen.

Downloading the Config File

In this step, we'll need to download the configuration file and add this to our Xcode project.

Download GoogleService-Info.plist and move this into the root of your Xcode project within Runner:

Be sure to move this file within Xcode to create the proper file references.

There are additional steps for installing the Firebase SDK and adding initialization code, but they are not necessary for this tutorial.

That's it!

Conclusion:

This example showcases basic navigation between two pages using the `Navigator` and `MaterialPageRoute`. The `GestureDetector` is used to respond to user gestures for navigation. Flutter offers more advanced routing options and a wide range of gestures for creating interactive and engaging user interfaces.