

## Lab session 4: CRUD operations with SQLite using Flutter

### Task-2 Firebase for Flutter

In this documentation, we will learn how to add Firebase Authentication to a Flutter app using the **FlutterFire UI package**. With this package, we can add both **email/password auth** and **Google Sign In auth** to a Flutter app. We will also learn how to set up a Firebase project, and use the **FlutterFire CLI** to initialize Firebase in a Flutter app.

#### Create and set up a Firebase project

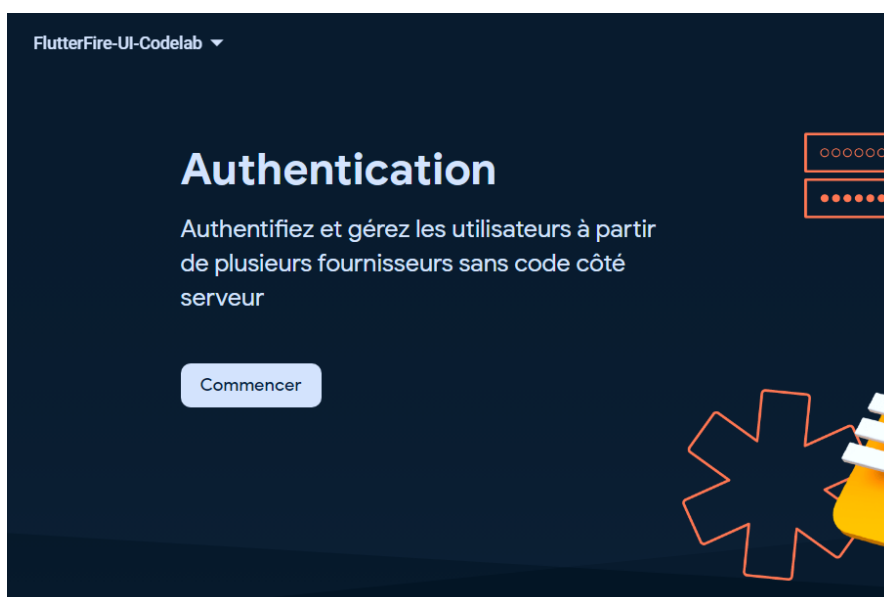
1. Sign in to [Firebase](#).
2. In the Firebase console, click Add Project (or Create a project), and enter a name for a Firebase project (for example, "FlutterFire-UI-Codelab").

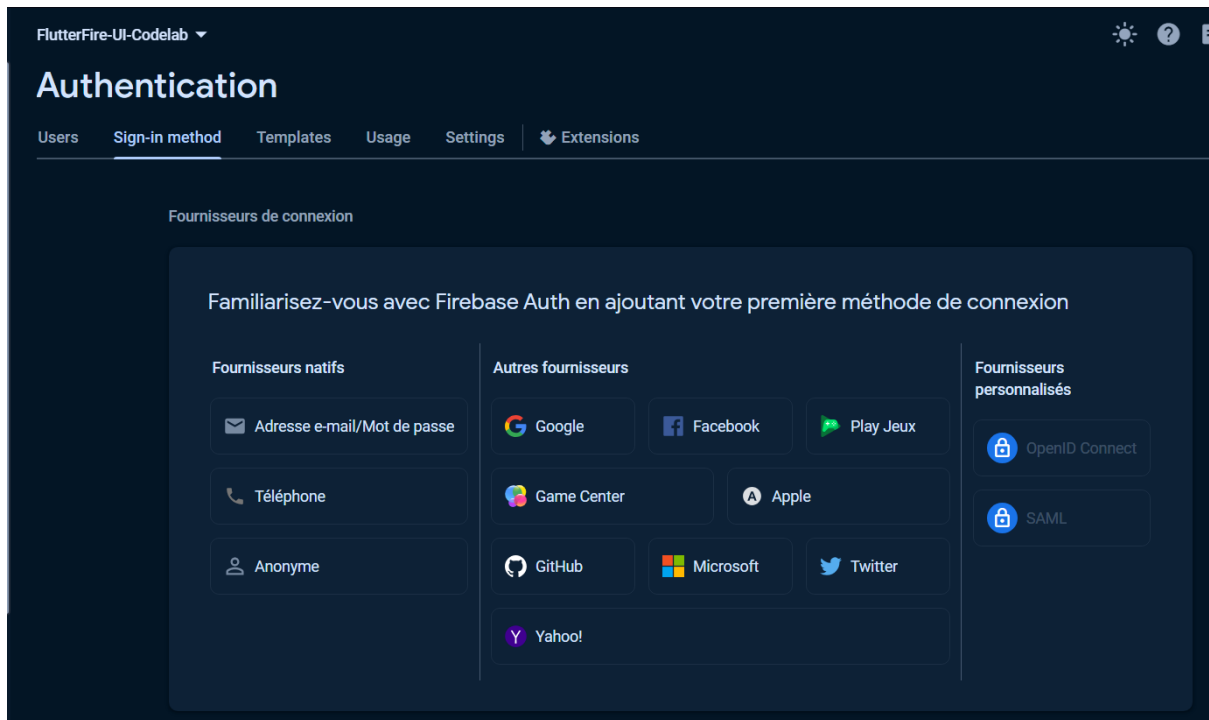


#### Enable email sign-in for Firebase Authentication

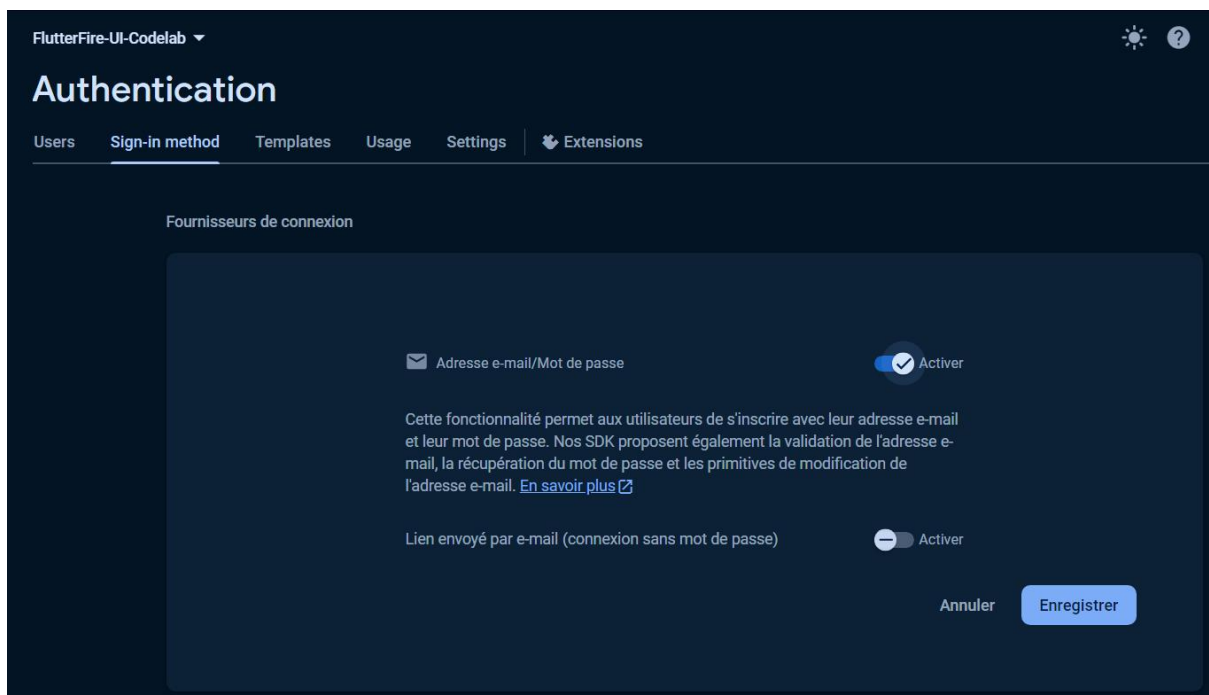
To allow users to sign in to the web app, we will first use the **Email/Password sign-in** method.

1. In the Firebase console, expand the Build menu in the left panel.
2. Click Authentication, and then click the Get Started button, then the Sign-in method tab





3. Click Email/Password in the Sign-in providers list, set the Enable switch to the on position, and then click Save.

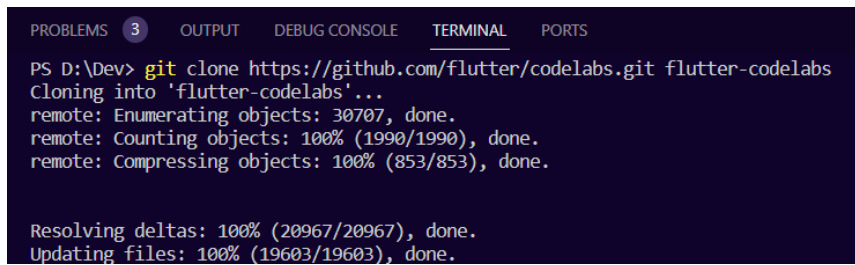


## Set up Flutter app

We will need to download the starter code, and install the **Firebase CLI** before we begin.

### Get the starter code

1. Clone the GitHub repository from the command line:



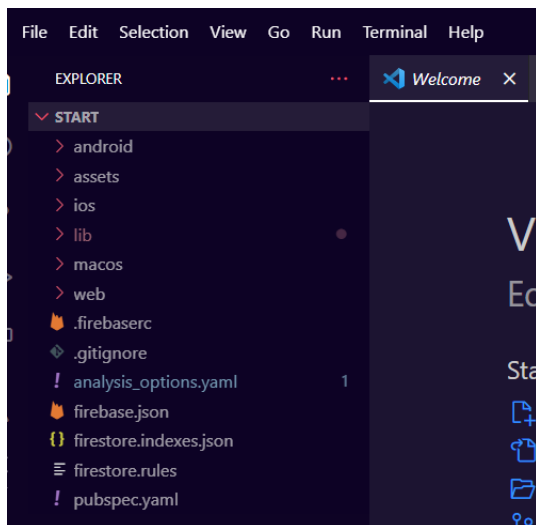
```

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\Dev> git clone https://github.com/flutter/codelabs.git flutter-codelabs
Cloning into 'flutter-codelabs'...
remote: Enumerating objects: 30707, done.
remote: Counting objects: 100% (1990/1990), done.
remote: Compressing objects: 100% (853/853), done.

Resolving deltas: 100% (20967/20967), done.
Updating files: 100% (19603/19603), done.

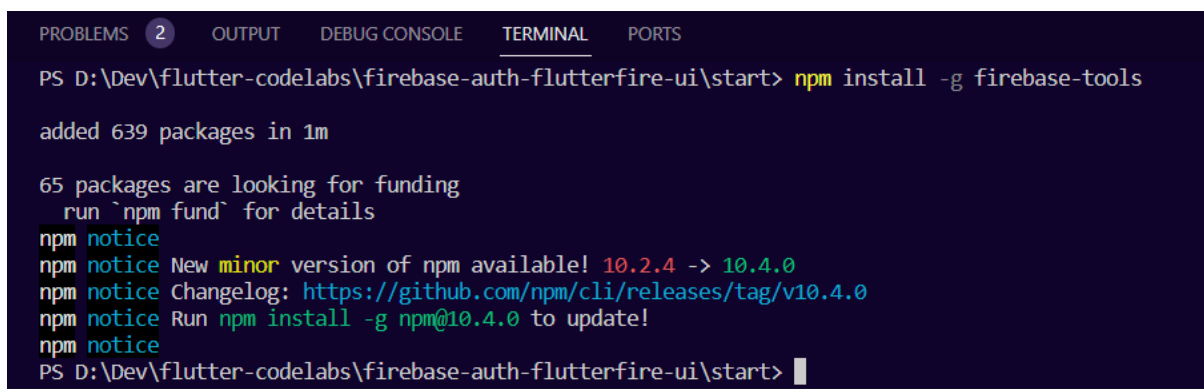
```

The directory `flutter-codelabs/firebase-auth-flutterfire-ui` contains two Flutter projects. One is called **complete** and the other is called **start**. The start directory contains an incomplete project, and it's where we will work.



## Install Firebase CLI

1. The Firebase CLI provides tools for managing Firebase projects. The CLI is required for the FlutterFire CLI, which we will install.



```

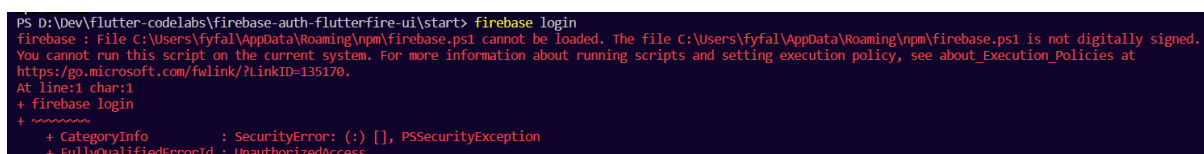
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\Dev\flutter-codelabs\firebase-auth-flutterfire-ui\start> npm install -g firebase-tools

added 639 packages in 1m

65 packages are looking for funding
  run `npm fund` for details
npm notice
npm notice New minor version of npm available! 10.2.4 -> 10.4.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.4.0
npm notice Run npm install -g npm@10.4.0 to update!
npm notice
PS D:\Dev\flutter-codelabs\firebase-auth-flutterfire-ui\start>

```

2. Log into Firebase using a Google account by running the following command:



```

PS D:\Dev\flutter-codelabs\firebase-auth-flutterfire-ui\start> firebase login
firebase: File C:\Users\fyfal\AppData\Roaming\npm\firebase.ps1 cannot be loaded. The file C:\Users\fyfal\AppData\Roaming\npm\firebase.ps1 is not digitally signed.
You cannot run this script on the current system. For more information about running scripts and setting execution policy, see about_Execution_Policies at
https://go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ firebase login
+ ~~~~~
+ CategoryInfo          : SecurityError: (:) [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess

```

We were unable to access it from the VS code terminal but retrying in a new terminal was successful:

```
D:\Dev\flutter-codelabs\firebase-auth-flutterfire-ui\start>firebase login
i Firebase optionally collects CLI and Emulator Suite usage and error reporting information to help improve our product
s. Data is collected in accordance with Google's privacy policy (https://policies.google.com/privacy) and is not used to
identify you.

? Allow Firebase to collect CLI and Emulator Suite usage and error reporting information? Yes
i To change your data collection preference at any time, run 'firebase logout' and log in again.

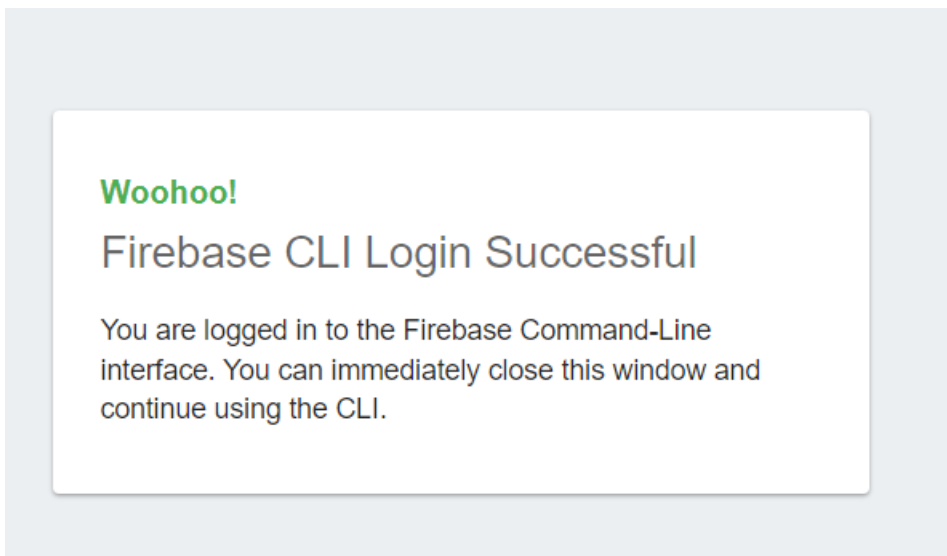
Visit this URL on this device to log in:
https://accounts.google.com/o/oauth2/auth?client_id=563584335869-fgrhgmd47bqnekij5i8b5pr03ho849e6.apps.googleusercontent
.com&scope=email%20openid%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloudplatformprojects.readonly%20https%3A%2F%2Fwww
.googleapis.com%2Fauth%2Ffirebase%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform&response_type=code&state=23
8982721&redirect_uri=http%3A%2F%2Flocalhost%3A9005

Waiting for authentication...

+ Success! Logged in as fyfalinarandrianarijaona@gmail.com
```

This command connects the local machine to Firebase and grants access to Firebase projects.

**Note:** The firebase login command opens a web page that connects to localhost on the user machine



3. Test that the CLI is properly installed and has access to the account by listing all the Firebase projects. Run the following command:

```
D:\Dev\flutter-codelabs\firebase-auth-flutterfire-ui\start>firebase projects:list
✓ Preparing the list of your Firebase projects
```

Project Display Name	Project ID	Project Number	Resource Location ID
AuthWithFirebase-Task2	authwithfirebase-task2	368944361986	[Not specified]
flutterfiresample	flutterfiresample-410d0	555544129034	[Not specified]
My First Project	leafy-garden-369105	951167972352	[Not specified]

```
3 project(s) total.
```

The displayed list should be the same as the Firebase projects listed in the Firebase console.

### Install the FlutterFire CLI

The FlutterFire CLI is a tool that helps ease the installation process of Firebase across all supported platforms in a Flutter app. It's built on top of the Firebase CLI.

1. First, install the CLI:

```
PS D:\Dev\flutter-codelabs\firebase-auth-flutterfire-ui\start> dart pub global activate flutterfire_cli
+ ansi_styles 0.3.2+1s... (2.9s)
+ args 2.4.2
+ async 2.11.0
+ boolean_selector 2.1.1
+ characters 1.3.0
+ ci 0.1.0
+ cli_util 0.3.5 (0.4.1 available)
+ clock 1.1.1
+ collection 1.18.0
+ dart_console 1.2.0
+ deep_pick 0.10.0 (1.0.0 available)
```

2. Add the Firebase project to the Flutter app.

### Configure FlutterFire

We can use FlutterFire to generate the needed Dart code to use Firebase in the Flutter app.

1. Run the command:

#### *flutterfire configure*

When this command is run, we should be prompted to select which Firebase project we want to use, and which platforms we want to set up.

But we can set it by default in the **.firebase** directory in the root of the Flutter project as below:

```
.firebase > ...
1 {
2   "projects": {
3     "default": "flutterfire-ui-codelab-ea225"
4   }
5 }
6
```

So that, the flutterfire configure automatically select the default project:

```
PS D:\Dev\flutter-codelabs\firebase-auth-flutterfire-ui\start> flutterfire configure
i Found 4 Firebase projects. Selecting project flutterfire-ui-codelab-ea225.
✓ Which platforms should your configuration support (use arrow keys & space to select)? · ios, macos, web, android
i Firebase android app com.example.complete is not registered on Firebase project flutterfire-ui-codelab-ea225.
i Registered a new Firebase android app on Firebase project flutterfire-ui-codelab-ea225.
i Firebase ios app com.example.complete is not registered on Firebase project flutterfire-ui-codelab-ea225.
i Registered a new Firebase ios app on Firebase project flutterfire-ui-codelab-ea225.
i Firebase macos app com.example.complete.RunnerTests is not registered on Firebase project flutterfire-ui-codelab-ea225.
i Registered a new Firebase macos app on Firebase project flutterfire-ui-codelab-ea225.
```

2. Select which platforms we want to use. In this codelab, there are steps to configure Firebase Authentication for Flutter for web, iOS, and Android, but we can set up your Firebase project to use alloptions.

```
i Found 9 Firebase projects.
✓ Select a Firebase project to configure your Flutter application with · flutter-firebase-c
? Which platforms should your configuration support (use arrow keys & space to select)? >
✓ android
✓ ios
✓ macos
✓ web
```

This screenshot shows the output at the end of the process.

```
PS D:\Dev\flutter-codelabs\firebase-auth-flutterfire-ui\start> flutterfire configure
i Found 4 Firebase projects. Selecting project flutterfire-ui-codelab.
FirebaseProjectNotFoundException: Firebase project id "flutterfire-ui-codelab" could not be found on this Firebase account.
PS D:\Dev\flutter-codelabs\firebase-auth-flutterfire-ui\start> flutterfire configure
i Found 4 Firebase projects. Selecting project flutterfire-ui-codelab-ea225.
✓ Which platforms should your configuration support (use arrow keys & space to select)? · ios, macos, web, android
i Firebase android app com.example.complete is not registered on Firebase project flutterfire-ui-codelab-ea225.
i Registered a new Firebase android app on Firebase project flutterfire-ui-codelab-ea225.
i Firebase ios app com.example.complete is not registered on Firebase project flutterfire-ui-codelab-ea225.
i Registered a new Firebase ios app on Firebase project flutterfire-ui-codelab-ea225.
i Firebase macos app com.example.complete.RunnerTests is not registered on Firebase project flutterfire-ui-codelab-ea225.
i Registered a new Firebase macos app on Firebase project flutterfire-ui-codelab-ea225.
i Firebase web app complete (web) is not registered on Firebase project flutterfire-ui-codelab-ea225.
i Registered a new Firebase web app on Firebase project flutterfire-ui-codelab-ea225.
✓ Generated FirebaseOptions file lib\firebase_options.dart already exists, do you want to override it? · yes

Firebase configuration file lib\firebase_options.dart generated successfully with the following Firebase apps:

Platform  Firebase App Id
web        1:43272927608:web:934f0913fc1d9e4839708d
android    1:43272927608:android:b89d3772cb13b19939708d
ios        1:43272927608:ios:e634578ef14a671039708d
macos      1:43272927608:ios:c8ab5a7fea08d90d39708d

Learn more about using this file and next steps from the documentation:
> https://firebase.google.com/docs/flutter/setup
```

Now, if we take a look at the Flutter app in the text editor, FlutterFire CLI has generated a new file called **firebase\_options.dart**. This file contains a class called **FirebaseOptions**, which has static variables that hold the Firebase configuration needed for each platform.

We selected all the platforms when running **flutterfire configure**, that's why we can see static values named web, android, ios, and macos.

```
lib > firebase_options.dart > DefaultFirebaseOptions > currentPlatform

1 // File generated by FlutterFire CLI.
2 // ignore_for_file: lines_longer_than_80_chars, avoid_classes_with_only_static_members
3 import 'package:firebase_core/firebase_core.dart' show FirebaseOptions;
4 import 'package:flutter/foundation.dart'
5   show defaultTargetPlatform, kIsWeb, TargetPlatform;
6
7 /// Default [FirebaseOptions] for use with your Firebase apps.
8 ///
9 /// Example:
10 /// ```dart
11 /// import 'firebase_options.dart';
12 /// // ...
13 /// await Firebase.initializeApp(
14 ///   options: DefaultFirebaseOptions.currentPlatform,
15 /// );
16 /// ```
17 class DefaultFirebaseOptions {
18   static FirebaseOptions get currentPlatform {
19     if (kIsWeb) {
20       return web;
21     }
22     switch (defaultTargetPlatform) {
23       case TargetPlatform.android:
24         return android;
25       case TargetPlatform.iOS:
26         return ios;
27       case TargetPlatform.macOS:
```

### Add Firebase packages to Flutter app

**Flutter pub get** should normally install all the necessary dependencies but in case it fails to do so: Then, run the three following commands:

```
flutter pub add firebase_core
```

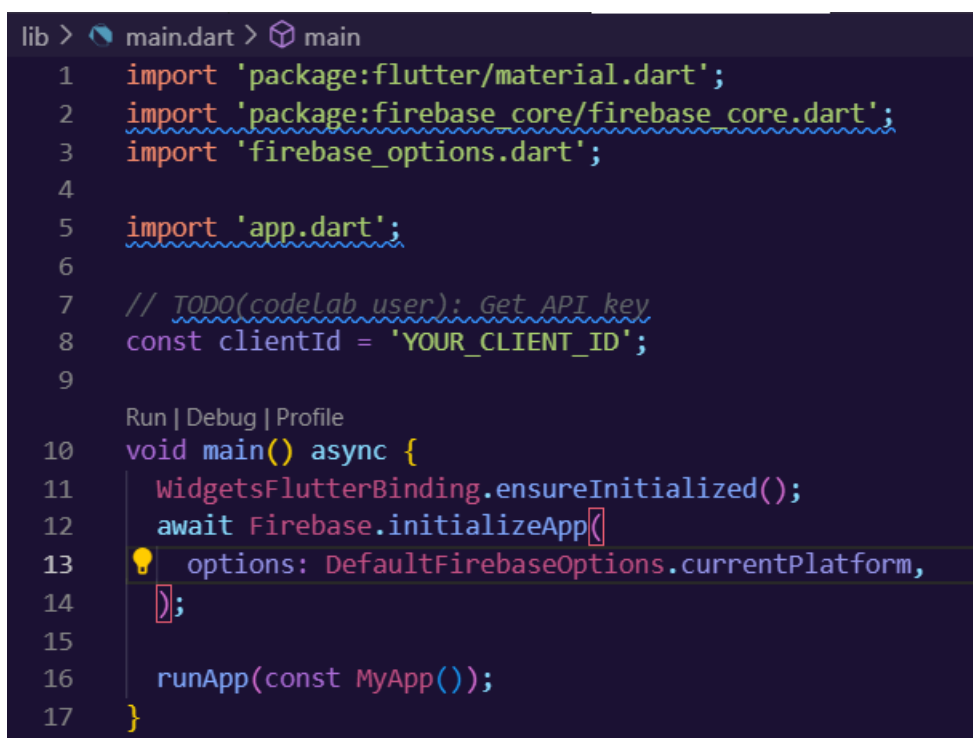
```
flutter pub add firebase_auth
```

```
flutter pub add firebase_ui_auth
```

These are the only packages we need at this point.

### Initialize Firebase

In order to use the packages added, and the **DefaultFirebaseOptions.currentPlatform**, update the code in the main function in the main.dart file.



```
lib > main.dart > main
1  import 'package:flutter/material.dart';
2  import 'package:firebase_core/firebase_core.dart';
3  import 'firebase_options.dart';
4
5  import 'app.dart';
6
7  // TODO(codeLab user): Get API key
8  const clientId = 'YOUR_CLIENT_ID';
9
10 void main() async {
11   WidgetsFlutterBinding.ensureInitialized();
12   await Firebase.initializeApp(
13     options: DefaultFirebaseOptions.currentPlatform,
14   );
15
16   runApp(const MyApp());
17 }
```

This code does two things:

- **WidgetsFlutterBinding.ensureInitialized()** tells Flutter not to start running the application widget code until the Flutter framework is completely booted. Firebase uses native platform channels, which require the framework to be running.
- **Firebase.initializeApp** sets up a connection between your Flutter app and your Firebase project. The **DefaultFirebaseOptions.currentPlatform** is imported from our generated **firebase\_options.dart** file. This static value detects which platform we are running on, and passes in the corresponding Firebase keys.

### Add initial Firebase UI Auth page

Firebase UI for Auth provides widgets that represent entire screens in the application. These screens handle different authentication flows throughout the application, such as *Sign In*, *Registration*, *Forgot Password*, *User Profile*, and more.

To get started, we will add a landing page to the app that acts as an authentication guard to the main application.

### Material or Cupertino App

FlutterFire UI requires that the application is wrapped in either a **MaterialApp** or **CupertinoApp**. Depending on the choice, the UI will automatically reflect the differences of Material or Cupertino widgets.

For this example, we use **MaterialApp**, which is already added to the app in `app.dart`.

```
lib > app.dart > ...
1  import 'package:flutter/material.dart';
2
3  import 'auth_gate.dart';
4
5  class MyApp extends StatelessWidget {
6    const MyApp({super.key});
7    @override
8    Widget build(BuildContext context) {
9      return MaterialApp(
10         theme: ThemeData(
11           colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
12           useMaterial3: true,
13         ), // ThemeData
14         home: const AuthGate(),
15       ); // MaterialApp
16     }
17 }
```

### Check authentication state

Before we can display a sign-in screen, we need to determine whether the user is currently authenticated. The most common way to check for this is to listen to **FirebaseAuth's** **authStateChanges** using the Firebase Auth plugin.

In the code sample above, the **MaterialApp** is building an **AuthGate** widget in its build method. (This is a custom widget, not provided by FlutterFire UI.)

That widget needs to be updated to include the **authStateChanges** stream.

The **authStateChanges** API returns a Stream with either the current user (if they are signed in), or null if they are not. To subscribe to this state in our application, we can use Flutter's **StreamBuilder** widget and pass the stream to it.

**StreamBuilder** is a widget that builds itself based on the latest snapshot of data from a Stream that is passed to it. It automatically rebuilds when the Stream emits a new snapshot.

1. Update the code in `auth_gate.dart`:



```

lib > auth_gate.dart > AuthGate > build
1  import 'package:firebase_auth/firebase_auth.dart' hide EmailAuthProvider;
2  import 'package:firebase_ui_auth/firebase_ui_auth.dart';
3  import 'package:flutter/material.dart';
4
5  import 'home.dart';
6
7  class AuthGate extends StatelessWidget {
8    const AuthGate({super.key});
9
10   @override
11   widget build(BuildContext context) {
12     return StreamBuilder<User?>(
13       stream: FirebaseAuth.instance.authStateChanges(),
14       builder: (context, snapshot) {
15         if (!snapshot.hasData) {
16           return SignInScreen(
17             providers: [],
18           ); // SignInScreen
19         }
20
21         return const HomeScreen();
22       },
23     ); // StreamBuilder
24   }
25 }

```

**StreamBuilder.stream** is being passed **FirebaseAuth.instance.authStateChanges**, the aforementioned stream, which will return a Firebase User object if the user has authenticated. (Otherwise it will return null.)

Next, the code is using **snapshot.hasData** to check if the value from the stream contains the User object.

If there isn't, it'll return a **SignInScreen** widget. Currently, that screen won't do anything. This will be updated in the next step.

Otherwise, it returns a **HomeScreen**, which is the main part of the application that only authenticated users can access.

### Sign-In screen

The **SignInScreen** widget, provided by FlutterFire UI, adds the following functionality:

1. Allows users to sign in
2. If users forgot their password, they can tap "Forgot password?" and be taken to a form to reset their password
3. If a user isn't yet registered, they can tap "Register", and be taken to another form that allows them to sign up.

Again, this requires only a couple lines of code. Recall the code in the **AuthGate** widget:

```

if (!snapshot.hasData) {
  return SignInScreen(
    providers: [
      EmailAuthProvider(),
    ],
  ); // SignInScreen
}

```

The `SignInScreen` widget, and its `providers` argument, is the only code required to get all the aforementioned functionality.

### Customize the sign-in Screen

#### Header Builder

Using the **`SignInScreen.headerBuilder`** argument, we can add whatever widgets we want above the sign-in form. Update the `auth_gate.dart` file with this code:

```

lib > auth_gate.dart > AuthGate > build
16  return SignInScreen(
17    providers: [
18      EmailAuthProvider(),
19    ],
20    headerBuilder: (context, constraints, shrinkOffset) {
21      return Padding(
22        padding: const EdgeInsets.all(20),
23        child: AspectRatio(
24          aspectRatio: 1,
25          child: Image.asset('assets/flutterfire_300x.png'),
26        ), // AspectRatio
27      ); // Padding
28    },
29  ); // SignInScreen
30  }
31
32  return const HomeScreen();
33  },
34 ); // StreamBuilder
35 }
36 }

```

Because it's a callback, it exposes values we could use, such as the `BuildContext` and `BoxConstraints`, and requires to return a widget. Whichever widget we return is displayed at the top of the screen.

In this example, the new code adds an image to the top of the screen. Your application should now look like this.

#### Subtitle Builder

The sign-in screen exposes three additional parameters that allow to customize the screen: `subtitleBuilder`, `footerBuilder`, and `sideBuilder`.

The **`subtitleBuilder`** is slightly different in that the callback arguments include an action, which is of type **`AuthAction`**. `AuthAction` is an **enum** that we can use to detect if the screen the user is on is the "sign in" screen or the "register" screen.

Update the code in `auth_gate.dart` to use the `subtitleBuilder`.

```
lib > auth_gate.dart > AuthGate > build
24         child: Image.asset('assets/flutterfire_300x.png'),
25       ), // AspectRatio
26     ); // Padding
27   },
28   subtitleBuilder: (context, action) {
29     return Padding(
30       padding: const EdgeInsets.symmetric(vertical: 8.0),
31       child: action == AuthAction.signIn
32         ? const Text('Welcome to FlutterFire, please sign in!')
33         : const Text('Welcome to Flutterfire, please sign up!'),
34     ); // Padding
35   },
36 ]; // SignInScreen
37 }
38
39
40 return const HomeScreen();
41 },
42 ); // StreamBuilder
43 }
44 }
```

#### Footer builder

The **footerBuilder** argument is the same as the **subtitleBuilder**. It doesn't expose **BoxConstraints** or **shrinkOffset**, as it's intended for text rather than images.

Add a footer to the sign-in screen with this code.

```
lib > auth_gate.dart > AuthGate > build
34         ); // Padding
35     ); // Padding
36   },
37   footerBuilder: (context, action) {
38     return const Padding(
39       padding: EdgeInsets.only(top: 16),
40       child: Text(
41         'By signing in, you agree to our terms and conditions.',
42         style: TextStyle(color: Colors.grey),
43       ), // Text
44     ); // Padding
45   },
46 ]; // SignInScreen
47 }
48
49 return const HomeScreen();
50 },
51 ); // StreamBuilder
52 }
53 }
```

#### Side Builder

The **SignInScreen.sidebuilder** argument accepts a callback, and this time the arguments to that callback are **BuildContext** and double **shrinkOffset**. The widget that **sideBuilder** returns will be

displayed to the left of the sign in form, and only on wide screens. Effectively that means the widget will only be displayed on desktop and web apps.

Internally, FlutterFire UI uses a breakpoint to determine if the header content should be shown (on tall screens, like mobile) or the side content should be shown (on wide screens, desktop or web). Specifically, if a screen is more than 800 pixels wide, the side builder content is shown, and the header content is not. If the screen is less than 800 pixels wide, the opposite is true.

Update the code in `auth_gate.dart` to add `sideBuilder` widgets.

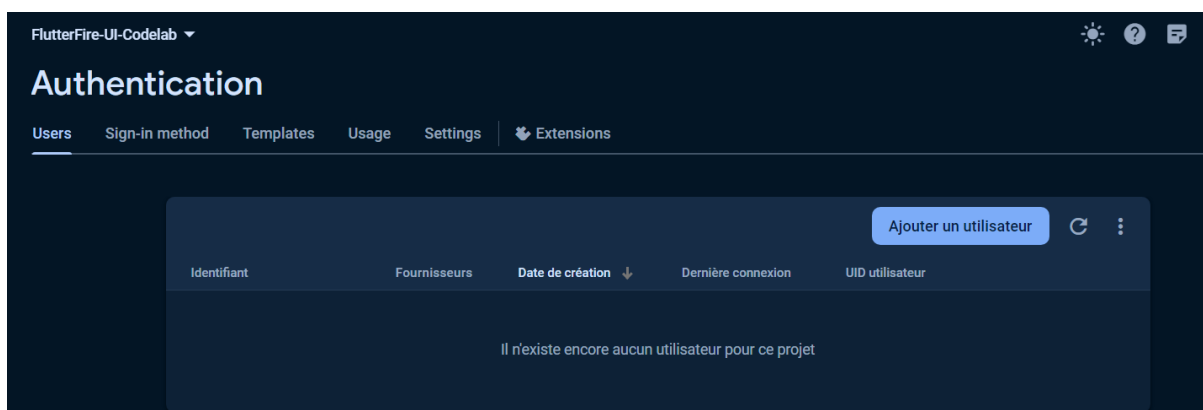
```
lib > auth_gate.dart > AuthGate > build
44         ), // Padding
45     },
46     sideBuilder: (context, shrinkOffset) {
47         return Padding(
48             padding: const EdgeInsets.all(20),
49             child: AspectRatio(
50                 aspectRatio: 1,
51                 child: Image.asset('flutterfire_300x.png'),
52             ), // AspectRatio
53         ); // Padding
54     },
55 ); // SignInScreen
56 }
57
58 return const HomeScreen();
59 },
60 ); // StreamBuilder
61 }
62 }
```

### Create a user

At this point, all of the code for the screen is done. Before we can sign-in, we need to create a User. We can do this with the "Register" screen, or we can create a user in the Firebase console.

To use the console:

1. Go to the "Users" table in the Firebase console.
2. Select 'flutterfire-ui-codelab'. We will see this table:



- Click the "Add user" button.

Ajouter un utilisateur

Identifiant Fournisseurs Date de création ↓ Dernière connexion UID utilisateur

Ajouter un utilisateur avec un e-mail/mot de passe

E-mail Mot de passe

Annuler Ajouter un utilisateur

- Enter an email address and password for the new user.

This can be a fake email and password, but the "Forgot password" functionality will not work if we use a fake email address.

FlutterFire-UI-Codelab

# Authentication

Users Sign-in method Templates Usage Settings Extensions

Ajouter un utilisateur

Identifiant Fournisseurs Date de création ↓ Dernière connexion UID utilisateur

Ajouter un utilisateur avec un e-mail/mot de passe

E-mail Mot de passe

fyfalinar@gmail.com @041517

Annuler Ajouter un utilisateur

Il n'existe encore aucun utilisateur pour ce projet

- Click "Add user"

FlutterFire-UI-Codelab

# Authentication

Users Sign-in method Templates Usage Settings Extensions

Recherchez par adresse e-mail, numéro de téléphone ou ID utilisateur

Ajouter un utilisateur

Identifiant Fournisseurs Date de création ↓ Dernière connexion UID utilisateur

fyfalinar@gmail.com	✉	30 janv. 2024		a4BxIMGRSGQSJiX04gECaml...
---------------------	---	---------------	--	----------------------------

Lignes par page : 50 1 - 1 of 1

## Profile Screen

FlutterFire UI also provides a `ProfileScreen` widget, which again, gives a lot of functionality in a few lines of code.

### Add `ProfileScreen` widget

1. Navigate to the `home.dart` file in your text editor. Update it with this code:

```
lib > home.dart > HomeScreen > build
1  import 'package:firebase_ui_auth/firebase_ui_auth.dart';
2  import 'package:flutter/material.dart';
3
4  class HomeScreen extends StatelessWidget {
5    const HomeScreen({super.key});
6
7    @override
8    Widget build(BuildContext context) {
9      return Scaffold(
10        appBar: AppBar(
11          actions: [
12            IconButton(
13              icon: const Icon(Icons.person),
14              onPressed: () {
15                Navigator.push(
16                  context,
17                  MaterialPageRoute<ProfileScreen>(
18                    builder: (context) => const ProfileScreen(),
19                  ), // MaterialPageRoute
20                );
21              },
22            ) // IconButton
23          ],
24          automaticallyImplyLeading: false,
25        ), // AppBar
26        body: Center(
```

The new code is the callback passed to the `IconButton.isPressed` method. When that `IconButton` is pressed, the application creates a new anonymous route and navigates to it. That route will display the `ProfileScreen` widget, which is returned from the `MaterialPageRoute.builder` callback.

## Signing Out

Now, if we press the "Sign out" button, the app will not change. It will sign the user out, but it will not be navigated back to the `AuthGate` widget. To implement this, use the `ProfileScreen.actions` parameter.

First, update the code in `home.dart`.

```

lib > home.dart > HomeScreen > build
14      onPressed: () {
15          Navigator.push(
16              context,
17              MaterialPageRoute<ProfileScreen>(
18                  builder: (context) => const ProfileScreen(
19                      actions: [
20                          SignedOutAction((context) {
21                              Navigator.of(context).pop();
22                          }) // SignedOutAction
23                      ],
24                      ), // ProfileScreen

```

Now, when we create an instance of **ProfileScreen**, we also pass it a list of actions to the **ProfileScreen.actions** argument. These actions are of the type **FlutterFireUiAction**.

There are many different classes that are subtypes of **FlutterFireUiAction**, and in general we use them to tell the app to react to different auth state changes. The **SignedOutAction** calls a callback function that we give it when the Firebase auth state changes to the **currentUser** being null.

By adding a callback that calls **Navigator.of(context).pop()** when **SignedOutAction** triggers, the app will navigate to the previous page. In this example app, there is only one permanent route, which shows the sign in page if there isn't a user signed in, and the home page if there is a user. Because this happens when the user signs out, the app will display the Sign In page.

### Customize the Profile Page

Similar to the Sign In page, the profile page is customizable. First, our current page has no way of navigating back to the home page once a user is on the profile page. Fix this by giving the **ProfileScreen** widget an **AppBar**.

```

lib > home.dart > HomeScreen > build
18      builder: (context) => ProfileScreen(
19          appBar: AppBar(
20              title: const Text('User Profile'),
21          ), // AppBar
22          actions: [
23              SignedOutAction((context) {
24                  Navigator.of(context).pop();
25              }) // SignedOutAction
26          ],
27      ), // ProfileScreen
28  ], // MaterialPageRoute
29  );
30  },
31  ) // IconButton
32  ],
33  automaticallyImplyLeading: false,

```

The `ProfileScreen.appBar` argument accepts an `AppBar` widget from the Flutter Material package, so it can be treated like any other `AppBar` built and passed to a `Scaffold`. In this example, the default functionality of automatically adding a "back" button is kept, and the screen now has a title.

### Add Children to the Profile Screen

The `ProfileScreen` widget also has an optional argument named `children`. This argument accepts a list of widgets, and those widgets will be placed vertically inside of a `Column` widget that's already used internally to build the `ProfileScreen`. This `Column` widget in the `ProfileScreen` build method will place the children we pass it above the "Sign out" button.

Update the code in `home.dart` to show the company logo, similar to the sign in screen.

```
lib > home.dart > HomeScreen > build
18 builder: (context) => ProfileScreen(
19   appBar: AppBar(
20     title: const Text('User Profile'),
21   ), // AppBar
22   actions: [
23     SignedOutAction((context) {
24       Navigator.of(context).pop();
25     }) // SignedOutAction
26   ],
27   children: [
28     const Divider(),
29     Padding(
30       padding: const EdgeInsets.all(2),
31       child: AspectRatio(
32         aspectRatio: 1,
33         child: Image.asset('flutterfire_300x.png'),
34       ), // AspectRatio
35     ), // Padding
36   ],
37 ), // ProfileScreen
```

### Multiplatform Google Auth Sign In

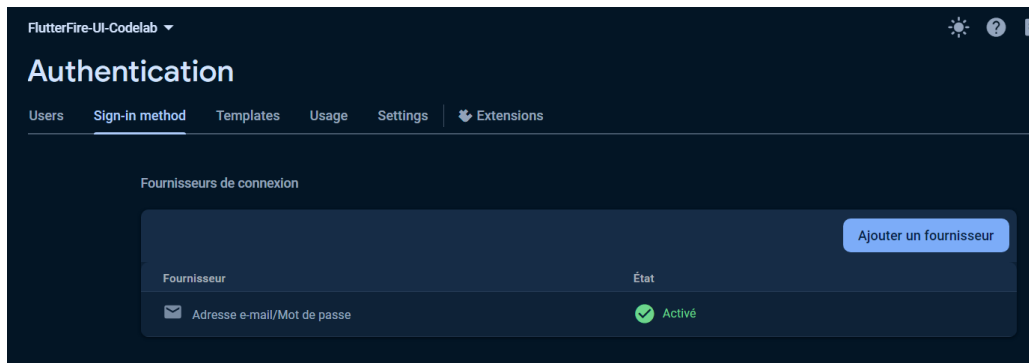
FlutterFire UI also provides widgets and functionality for authenticating with 3rd party providers, such as Google, Twitter, Facebook, Apple, and Github.

1. To integrate with Google authentication, install the official **firebase\_ui\_oauth\_google** plugin and its dependencies, which will handle the native authentication flow. In the terminal, navigate to the root of your flutter project and enter the following command:

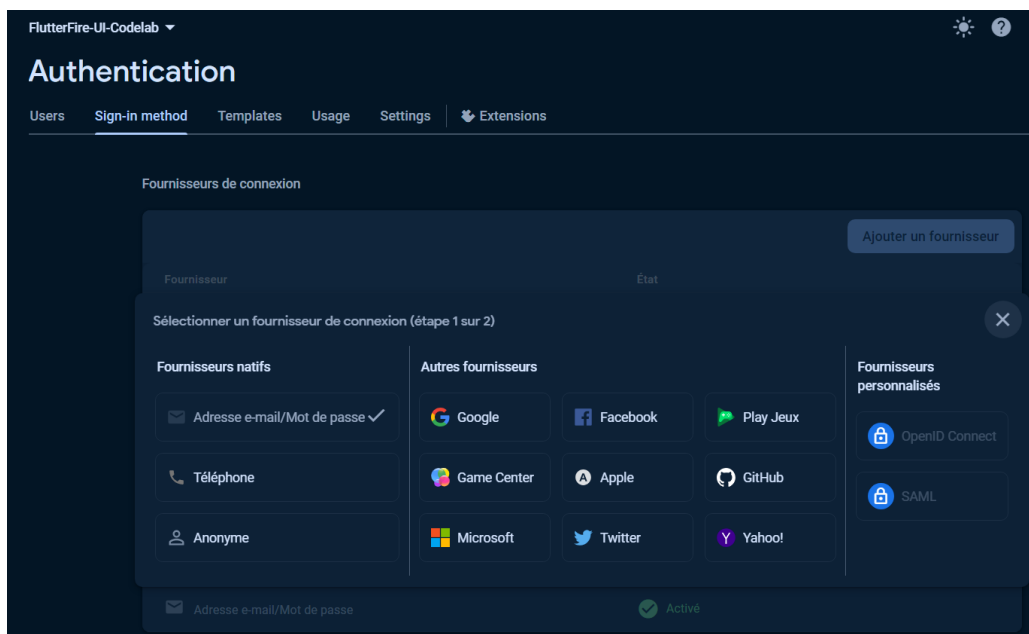
#### *Enable Google Sign-in Provider*

2. Next, enable the Google provider in the Firebase Console:
  - Navigate to the Authentication sign-in providers screen in the console.
  - Click "Add new provider".

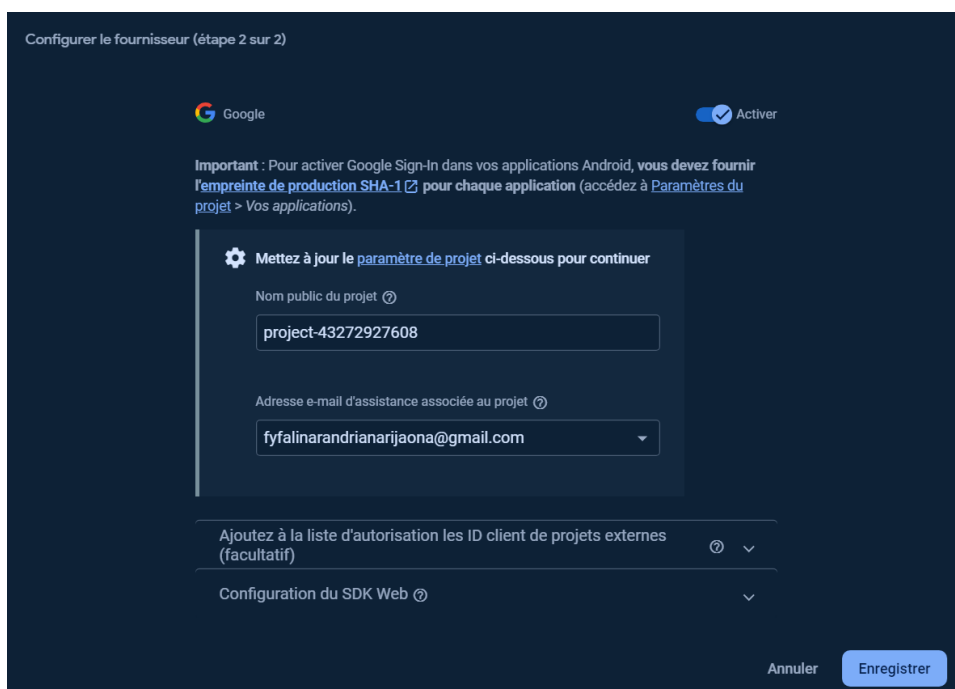




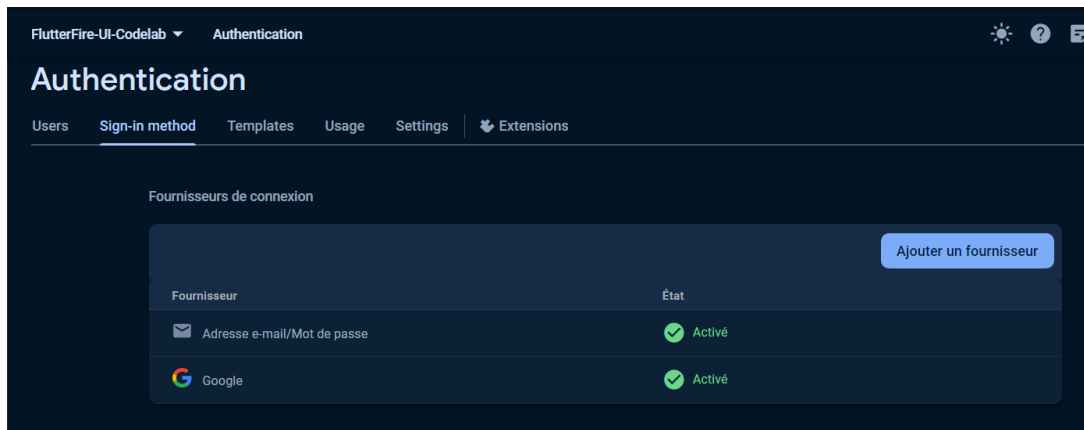
- Select "Google".



- Toggle the switch labeled "Enable", and press "Save"



- Confirm that the Google sign-in provider has been added.



### Add Google sign-in button

With Google sign-in enabled, add the widget needed to display a stylized Google sign-in button to the sign in page. Navigate to `auth_gate.dart` file and update the code to the following:

```
lib > auth_gate.dart > AuthGate > build
14   stream: FirebaseAuth.instance.authStateChanges(),
15   builder: (context, snapshot) {
16     if (!snapshot.hasData) {
17       return SignInScreen(
18         providers: [
19           EmailAuthProvider(),
20           GoogleProvider(clientId: "YOUR_WEBCLIENT_ID"),
21         ],

```

The only new code here is the addition of `GoogleProvider(clientId: "YOUR_WEBCLIENT_ID")` to the `SignInScreen` widget configuration.

### Configure sign-in button

The button does not work without additional configuration. If we are developing with Flutter Web, this is the only step we have to add for this to work. Other platforms require additional steps, which are discussed in a bit.

1. Navigate to the Authentication providers page in the Firebase Console.
2. Click on the Google provider.
3. Click on the "Web SDK configuration" expansion-panel.
4. Copy the value from 'Web client ID'

Google Activer

**Important :** Pour activer Google Sign-In dans vos applications Android, vous devez fournir l'empreinte de production SHA-1 pour chaque application (accédez à Paramètres du projet > Vos applications).

**Mettez à jour le paramètre de projet ci-dessous pour continuer**

Nom public du projet ?

Adresse e-mail d'assistance associée au projet ?

Ajoutez à la liste d'autorisation les ID client de projets externes (facultatif) ?

Configuration du SDK Web ?

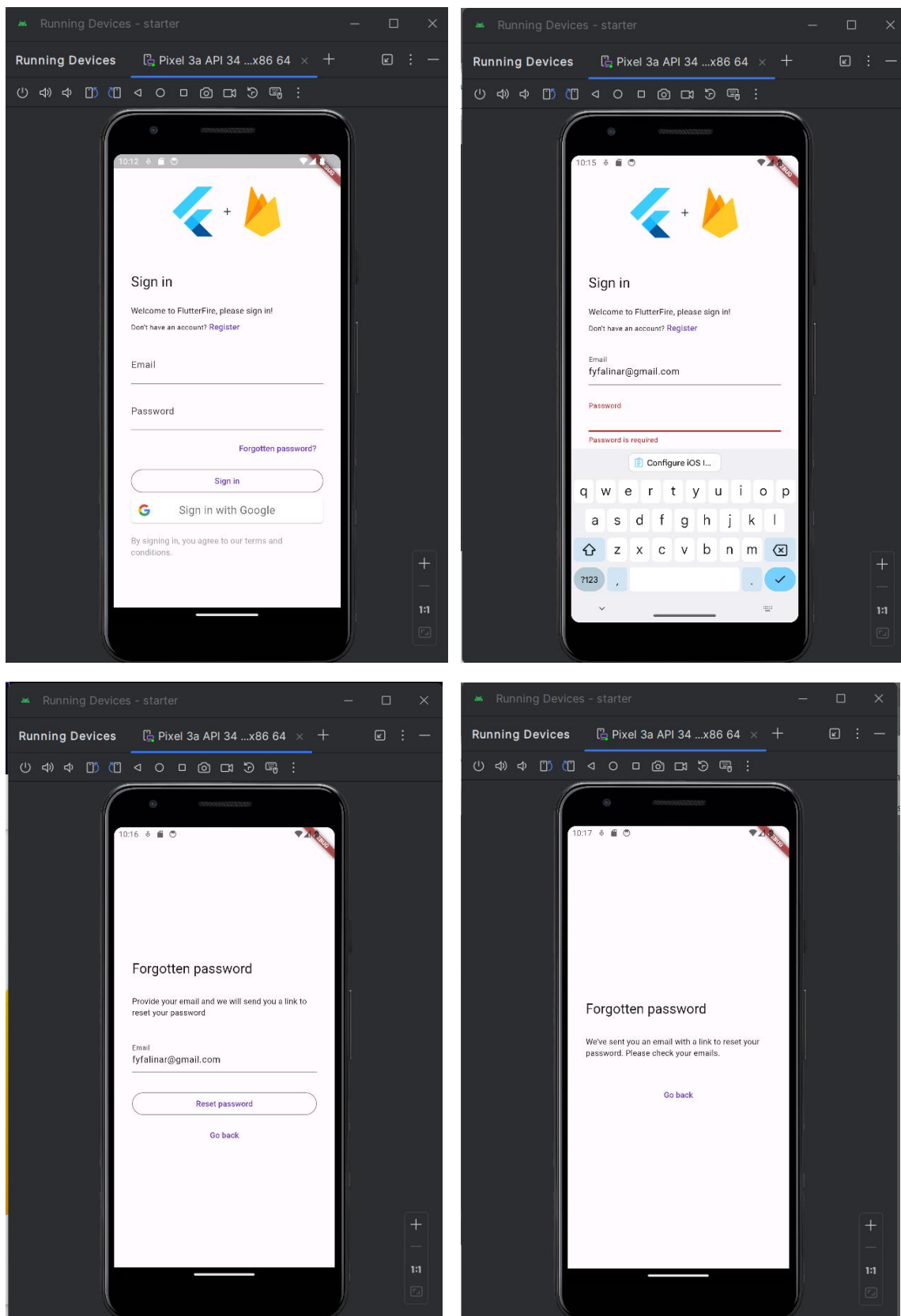
ID client Web

- Return to your text editor, and update the instance of GoogleProvider in the file auth\_gate.dart by passing this ID to the clientId named parameter.

```
lib > auth_gate.dart > AuthGate > build
11  @override
12  Widget build(BuildContext context) {
13    return StreamBuilder<User?>(  
14      stream: FirebaseAuth.instance.authStateChanges(),  
15      builder: (context, snapshot) {  
16        if (!snapshot.hasData) {  
17          return SignInScreen(  
18            providers: [  
19              EmailAuthProvider(),  
20              GoogleProvider(clientId: '43272927608-cl2kv2lk5273pt0sus2cmp4viosvlne4.apps.goo  
21            ],
```

## Testings

After all these steps, our app should be like this:



We can see here that the “Forgot Password” actually sends the mail to the specified email address:

