

**Name: Ameya M. Angne**  
**Class: D15B**

**Roll no.: 01**

### **MAD Experiment 6**

Aim : To connect Flutter UI with Firebase.

Theory :

Google Firebase is a set of cloud-based development tools that helps mobile app developers build, deploy and scale their apps.

Firebase provides a variety of features, including the following:

- Authentication. Firebase provides a secure and easy way for users to sign into their app. Developers can use Firebase Authentication to support email and password login, Google Sign-In, Facebook Login and more.
- Realtime Database. The Firebase Realtime Database is a cloud-hosted NoSQL database that lets organizations store and sync data in real time across all of their users' devices. This makes it easy to build apps that are always up to date, even when users are offline.
- Cloud Messaging. Firebase Cloud Messaging (FCM) is a service that lets businesses send messages to their users' devices, even if they're not using the app. Developers can use FCM to send push notifications, update app content, and more.
- Crashlytics. Firebase Crashlytics is a service that helps organizations track and fix crashes in their app. Crashlytics provides detailed reports on crashes, so they can quickly identify the root cause and fix the problem.
- Performance Monitoring. Firebase Performance Monitoring provides insights into the performance of their app. Organizations can use Performance Monitoring to track metrics like CPU usage, memory usage and network traffic.
- Test Lab. Firebase Test Lab is a cloud-based service that lets developers test their app on a variety of devices and configurations. This helps them ensure the app works well on a variety of devices and in different network conditions.

Execution Steps:

## How To Set Up Firebase with Flutter for Android Apps:

Firebase is a great backend solution for anyone that wants to use authentication, databases, cloud functions, ads, and countless other features within an app.

In this article, you will create a Firebase project for Android platforms using Flutter.

Prerequisites:

To complete this tutorial, you will need:

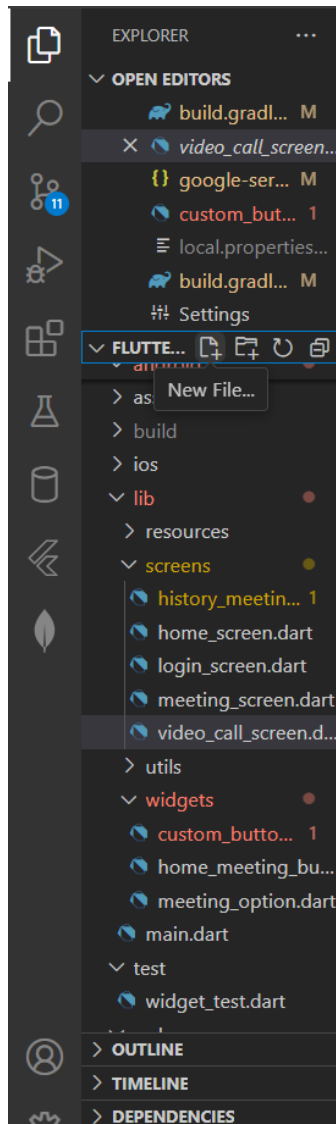
- A Google account to use Firebase.
- Developing for iOS will require XCode.
- To download and install Flutter.
- To download and install Android Studio and Visual Studio Code.
- It is recommended to install plugins for your code editor:
  - Flutter and Dart plugins installed for Android Studio.
  -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

This tutorial was verified with Flutter v2.0.6, Android SDK v31.0.2, and Android Studio v4.1.

## Creating a New Flutter Project:

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

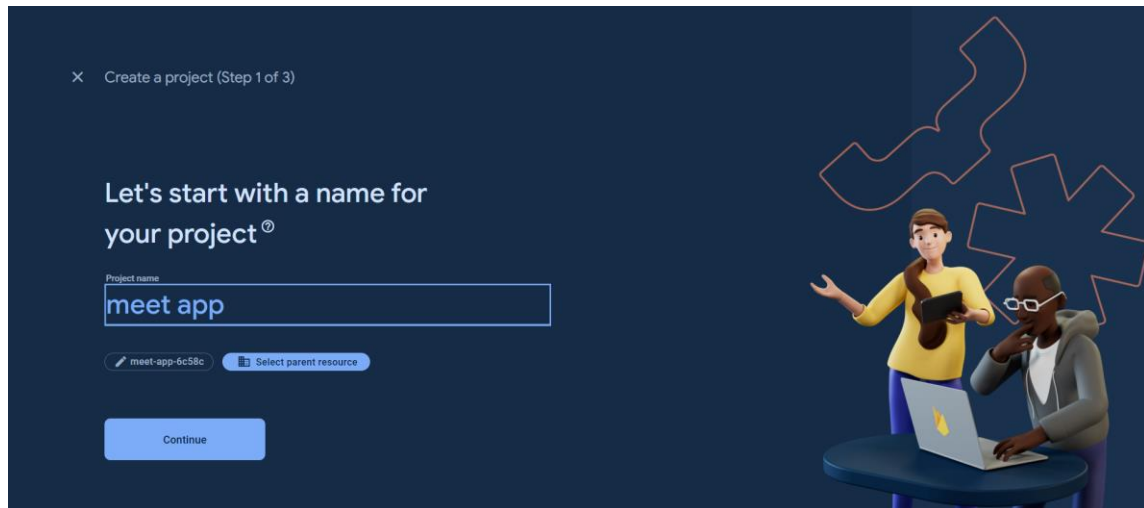
Using `flutter create` will produce a demo application that will display the number of times a button is clicked.



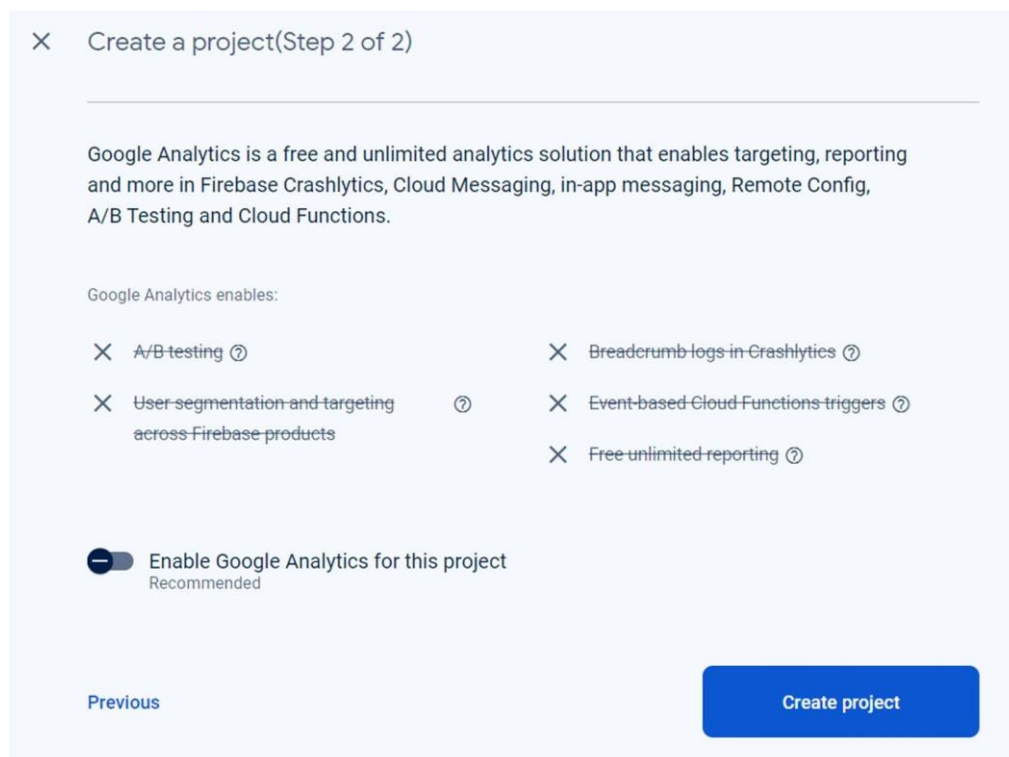
Now that we've got a Flutter project up and running, we can add Firebase.

## 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:



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.

Adding Android support

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:

The most important thing here is to match up the Android package name that you choose here with the one inside of our application.

The structure consists of at least two segments. A common pattern is to use a domain name, a company name, and the application name:

“com.example.recipe\_app”

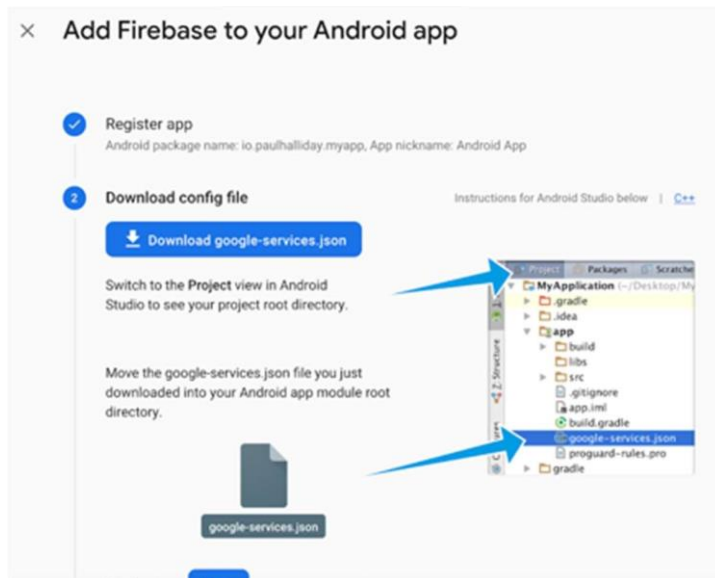
Once you’ve decided on a name, open `android/app/build.gradle` in your code editor and update the `applicationId` to match the Android package name:

```
android/app/build.gradle
defaultConfig {
    // TODO: Specify your own unique Application ID
    (https://developer.android.com/studio/build/application-id.html).
    applicationId "com.example.recipe_app"
    // You can update the following values to match your application needs.
```

You can skip the app nickname and debug signing keys at this stage. Select Register app to continue.

Downloading the Config File:

The next step is to add the Firebase configuration file into our Flutter project. This is important as it contains the API keys and other critical information for Firebase to use. Select Download `google-services.json` from this page:



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

Adding the Firebase SDK:

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

Open `android/build.gradle` in your code editor and modify it to include the following:

```

                                android/build.gradle
buildscript {
    ext.kotlin_version =
'1.7.10'    repositories {
    google()
        mavenCentral()
    }

    dependencies {        classpath "org.jetbrains.kotlin:kotlin-
gradle-plugin:$kotlin_version"                                classpath
'com.google.gms:google-services:4.3.15'
    }
}

```

Finally, update the app level `build.gradle` file at `android/app/build.gradle` to include the following:

```

                                android/app/build.gradle
                                android/app/build.gradle
plugins {    id
"com.android.application"    id

```

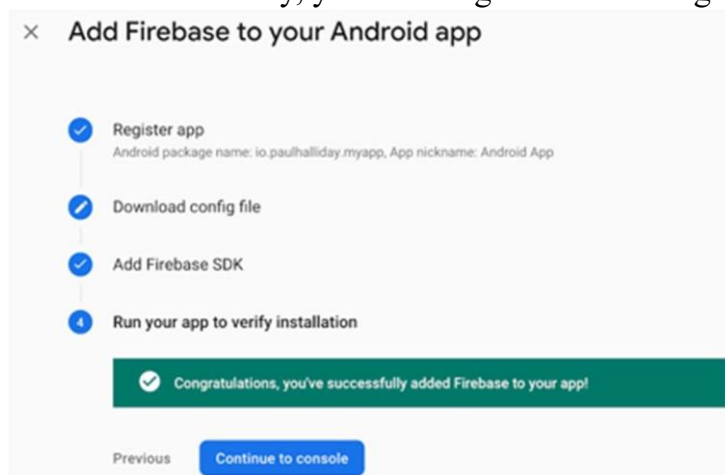
```
"kotlin-android"    id
"dev.flutter.flutter-gradle-plugin"
id 'com.google.gms.google-services'
}
```

...

```
dependencies {    implementation
platform('com.google.firebase:firebase-bom:32.7.3')
}
```

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. If everything has worked correctly, you should get the following message in the dashboard:



Login/ SignUp using firebase authentication:

We make a wrapper file in which it checks whether user is login or not.

wrapper.dart :

```
import
'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:recipe_app/home.dart';
import 'package:recipe_app/login.dart';
class Wrapper extends StatefulWidget {
  const Wrapper({super.key});
```

@override

```

    State<Wrapper> createState() => _WrapperState();
  }

class _WrapperState extends State<Wrapper> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body:
      FirebaseAuth.instance.authStateChanges(),
      StreamBuilder(stream:
      (context,snapshot){
        if (snapshot.hasData){
          return Home();
        } else{
          return Login();
        }
      },
      builder:
    );
  }
}

```

login.dart :

```

import
'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:get/get.dart'; import
'package:recipe_app/signup.dart';

class Login extends StatefulWidget {
  const Login({super.key});

  @override
  State<Login> createState() => _LoginState();
}

```

```

class _LoginState extends State<Login> {

  TextEditingController email=TextEditingController();
  TextEditingController password=TextEditingController();

  signIn() async {
    await
    FirebaseAuth.instance.signInWithEmailAndPassword(email: email.text,
    password:

```



```
password.text);
}
```

```
@override
Widget build(BuildContext context) { return
Scaffold( appBar: AppBar(title: Text("Login")),
body:Padding( padding: const
EdgeInsets.all(20.0), child: Column(
children: [ TextField(
controller:email, decoration:
InputDecoration(hintText: 'Enter email'),
),
TextField(
controller:password,
decoration: InputDecoration(hintText: 'Enter password'),
),
ElevatedButton(onPressed: ()=>signIn()), child: Text("Login")),
SizedBox(height: 30,),
ElevatedButton(onPressed: ()=>Get.to(SignUp())), child: Text("Register
Now"))
],),
)
);
}
}
```

signup.dart :

```
import
'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:get/get.dart'; import
'package:recipe_app/wrapper.dart'; class
SignUp extends StatefulWidget {
const SignUp({super.key});

@override
State<SignUp> createState() => _SignUpState();
}
class _SignUpState extends State<SignUp> {
```

```
TextEditingController email=TextEditingController();
```

```
TextEditingController password=TextEditingController();
```

```
    signUp() async {
        await
        FirebaseAuth.instance.createUserWithEmailAndPassword(email:
        email.text, password: password.text);    Get.offAll(()=>Wrapper());
    }
    @override
    Widget build(BuildContext context) { return
    Scaffold(    appBar: AppBar(title: Text("Sign
    Up")),    body:Padding(    padding: const
    EdgeInsets.all(20.0),    child: Column(
    children: [    TextField(
    controller:email,    decoration:
    InputDecoration(hintText: 'Enter email'),
        ),
        TextField(    controller:password,
    decoration: InputDecoration(hintText: 'Enter password'),
        ),
        ElevatedButton(onPressed: ()=>signUp()), child: Text("Sign Up"))
    ],),
    )
    );
    }
    }
```

UI :



# Sign In

[Forgot password?](#)

[Help](#) 

After login, we route to home page.

After sign up/ register in our app the authentication adds the user in firebase authentication services:

Conclusion : From this experiment, first of all we studied how to setup firebase. Then we created a firebase project, added multiple dependencies and packages to our flutter project so as to integrate our flutter project with firebase. Next step we authenticate the input fields from our app like email and password using Firebase and stored it.