Firebase Flutter

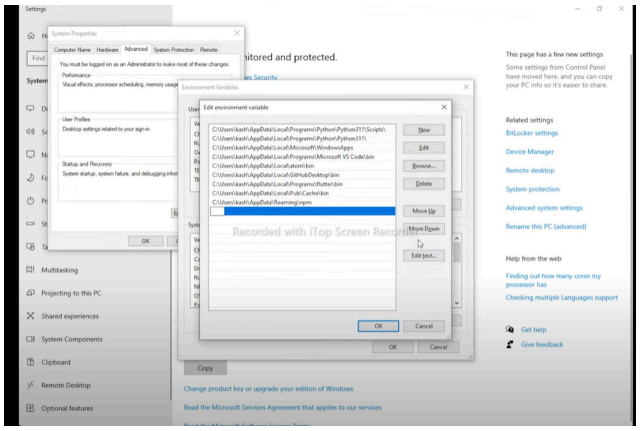
setup
authentication

Part 1 Connection

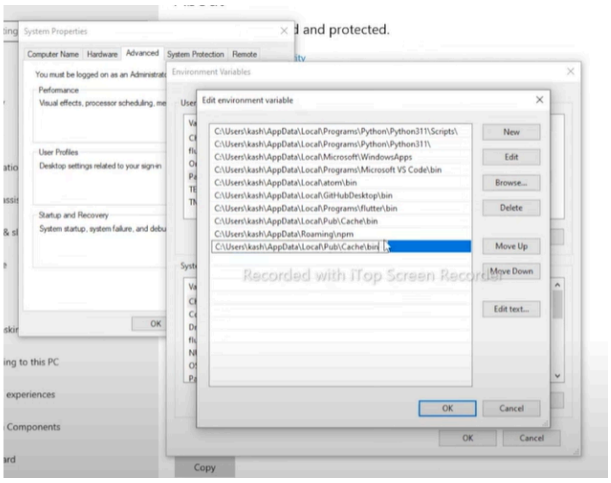
playlist -
🌐 How to connect Setup Flutter Firebase CLI on Window || Flutter Firebase CLI tutorials in ...

Step 1 Install NodeJS

Step 2 Setting up Environment Variables

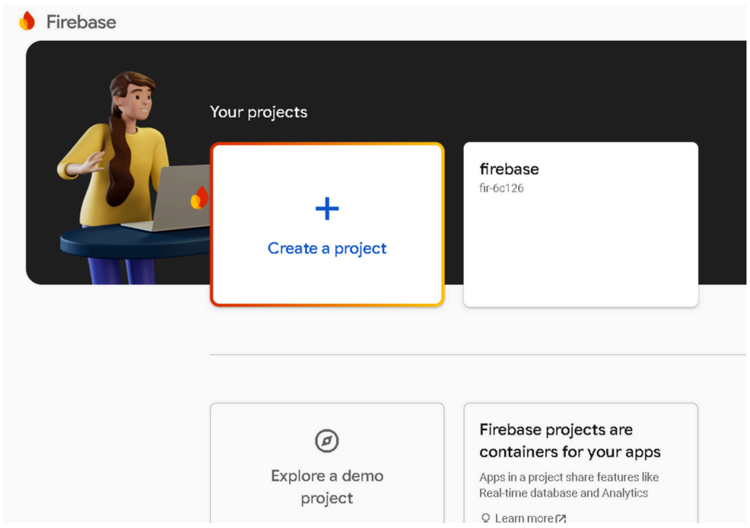


```
C:\Users\kash\AppData\Local\Programs\Python\Python311\Scripts\
C:\Users\kash\AppData\Local\Programs\Python\Python311\
C:\Users\kash\AppData\Local\Microsoft\WindowsApps
C:\Users\kash\AppData\Local\Programs\Microsoft VS Code\bin
C:\Users\kash\AppData\Local\atom\bin
C:\Users\kash\AppData\Local\GitHubDesktop\bin
C:\Users\kash\AppData\Local\Programs\flutter\bin
C:\Users\kash\AppData\Local\Pub\Cache\bin
C:\Users\kash\AppData\Roaming\npm
```



```
C:\Users\kash\AppData\Local\Programs\Python\Python311\Scripts\
C:\Users\kash\AppData\Local\Programs\Python\Python311\
C:\Users\kash\AppData\Local\Microsoft\WindowsApps
C:\Users\kash\AppData\Local\Programs\Microsoft VS Code\bin
C:\Users\kash\AppData\Local\atom\bin
C:\Users\kash\AppData\Local\GitHubDesktop\bin
C:\Users\kash\AppData\Local\Programs\flutter\bin
C:\Users\kash\AppData\Local\Pub\Cache\bin
C:\Users\kash\AppData\Roaming\npm
C:\Users\kash\AppData\Local\Pub\Cache\bin
```

Step 3 firebase console → create a project


Firebase
Your projects
Create a project
firebase
fir-6c126
Explore a demo project
Firebase projects are containers for your apps
Apps in a project share features like Real-time database and Analytics
Learn more

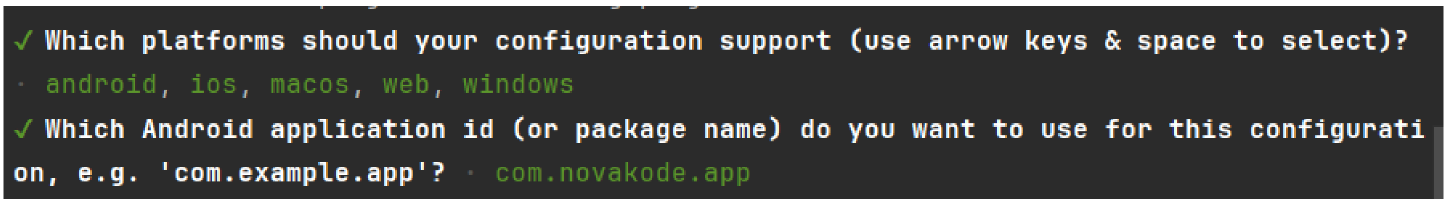
Step 4 CMD

C:\Users\Hayah Ahmed>npm install -g firebase-tools
C:\Users\Hayah Ahmed>firebase login
C:\Users\Hayah Ahmed>firebase projects:list

Step 5

in terminal

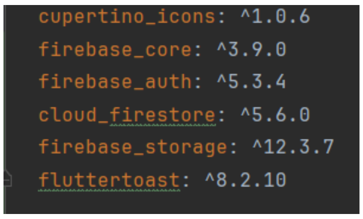
PS C:\Flutter Dev\Projects\firebase> flutterfire configure --project=fir-6c126



```
✓ Which platforms should your configuration support (use arrow keys & space to select)?
· android, ios, macos, web, windows
✓ Which Android application id (or package name) do you want to use for this configurati
on, e.g. 'com.example.app'? · com.novakode.app
```

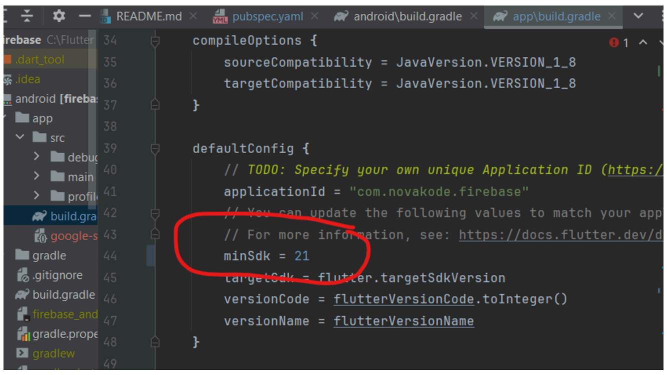
Part 2 & 3 Adding Firebase Dependencies & Running Project

Add the following dependencies



```
cupertino_icons: ^1.0.6
firebase_core: ^3.9.0
firebase_auth: ^5.3.4
cloud_firestore: ^5.6.0
firebase_storage: ^12.3.7
fluttertoast: ^8.2.10
```

app\build.gradle
change minsdk to 21

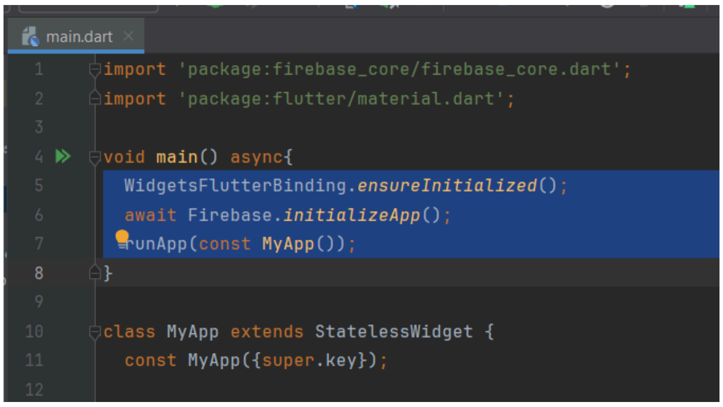


```
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_1_8
        targetCompatibility = JavaVersion.VERSION_1_8
    }

    defaultConfig {
        // TODO: Specify your own unique Application ID (https:/
        applicationId = "com.novakode.firebase"
        // You can update the following values to match your app
        // For more information, see: https://docs.flutter.dev/d
        minSdk = 21
        targetSdk = flutter.targetSdkVersion
        versionCode = flutterVersionCode.toInteger()
        versionName = flutterVersionName
    }
```

setup step for the Flutter engine.

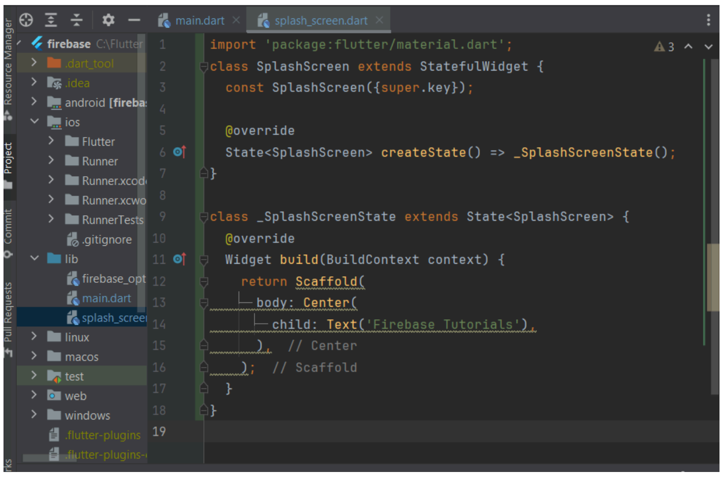
1️⃣ WidgetsFlutterBinding.ensureInitialized()
👉 Think of it as a setup step for the Flutter engine. Without it, your app might crash if you try to access certain services too early.

2️⃣ await Firebase.initializeApp()
👉 We're initializing Firebase here. This connects our app to the Firebase backend—think authentication, database, storage, etc.
🔥 *Pro Tip:* The await means we're waiting for Firebase to be ready before moving forward. That's why the main() is async.



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

void main() async{
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});
```

Part 4 Splash Screen



```
import 'package:flutter/material.dart';
class SplashScreen extends StatefulWidget {
  const SplashScreen({super.key});

  @override
  State<SplashScreen> createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Text('Firebase Tutorials'),
      ), // Center
    ); // Scaffold
  }
}
```

APP STUCK AT LOADING SCREEN AFTER FIREBASE SETUP

watch this video   https://www.youtube.com/watch?v=fM6ebEKPb6E

part 2-5 LOGIN SCREEN, SPLASH SCREEN, ROUND BUTTON, FORM VALIDATION

login_screen.dart

```
import 'package:firebase/widgets/round_button.dart';
import 'package:flutter/material.dart';

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

 @override
 State<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
 final _formKey = GlobalKey<FormState>();
 final emailController = TextEditingController();
 final passwordController = TextEditingController();
 @override
 void dispose() {
  super.dispose();
  emailController.dispose();
  passwordController.dispose();
 }

 Widget build(BuildContext context) {
  return SafeArea(
   child: Scaffold(
    appBar: AppBar(
     backgroundColor: Colors.deepPurple,
     title: const Center(
       child: Text(
      'Login ',
      style: TextStyle(color: Colors.white),
     )),
    ),
    body: Padding(
     padding: const EdgeInsets.all(12),
     child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      crossAxisAlignment: CrossAxisAlignment.center,
      children: [
      Form(
        key: _formKey,
        child: Column(
          children: [
            TextFormField(
              controller: emailController,
              decoration: const InputDecoration(
                  hintText: 'Email',
                  helperText: '  abc@example.com',
                  prefixIcon: Icon(Icons.alternate_email)),
              validator: (value) {
                if (value!.isEmpty) {
                  return 'Enter email';
                }
                return null;
              },
            ),
            TextFormField(
              controller: passwordController,
              obscureText: true,
              decoration: InputDecoration(
                hintText: 'Password',
                prefixIcon: Icon(Icons.lock),
              ),
              validator: (value) {
                if (value!.isEmpty) {
                  return 'Enter password';
                }
                return null;
              },
            ),
          ],
        ),
      ),
      SizedBox(
        height: 20,
      ),
      RoundButton(
        title: 'login',
        onTap: () {
          if (_formKey.currentState!.validate()) {}
        },
      )
    ],
  ),
),
    ),
   );
  }
}
```

round_button.dart

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

class RoundButton extends StatelessWidget {
 final String title;
 final VoidCallback onTap;
 const RoundButton({super.key, required this.title, required this.onTap});

 @override
 Widget build(BuildContext context) {
    return InkWell(
     onTap: onTap,
     child: Container(
      height: 50,
      width: 350


      ,
      decoration: BoxDecoration(
       color: Colors.deepPurple,
       borderRadius: BorderRadius.circular(10),

      ),
        child: Center(child: Text(title, style: TextStyle(color: Colors.white,),),),
       ),
    );
 }
}
```

splash_screen.dart

```
import 'package:firebase/firebase_services/splash_services.dart';
import 'package:flutter/material.dart';
class SplashScreen extends StatefulWidget {
 const SplashScreen({super.key});

 @override
 State<SplashScreen> createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen> {
 SplashServices splashScreen = SplashServices();
@override
void initState(){
 super.initState();
 splashScreen.isLogin(context);
}
 @override
 Widget build(BuildContext context) {
  return const Scaffold(
   body: Center(
    child: Text('Firebase Tutorials',style: TextStyle(fontSize: 30),),
   ),
  );
 }
}
```

splash_services.dart

```
import 'dart:async';

import 'package:firebase/ui/auth/login_screen.dart';
import 'package:flutter/material.dart';

class SplashServices {

 void isLogin(BuildContext context) {
  Timer(const Duration(seconds: 3), () {

Navigator.pushReplacement(context, MaterialPageRoute(builder: (context) => LoginScreen()));
  });
 }
}
```

Form Validation Breakdown 🚀

Step 1: Adding a **Form** and **GlobalKey**

- The Form widget is a container for multiple input fields. It allows us to group them and validate them together.
- We use a **GlobalKey** (_formKey) to uniquely identify the form and call its validation methods.

final _formKey = GlobalKey<FormState>();

The GlobalKey is like a remote control for your form. We'll use it to trigger validation when the login button is tapped.

Step 2: Text Input Fields with Validation

Each field comes with its own **validator** function to check the input.

Email Field

```
TextFormField(

 controller: emailController,

 decoration: const InputDecoration(

    hintText: 'Email',

    helperText: '  abc@example.com',

    prefixIcon: Icon(Icons.alternate_email)),

 validator: (value) {

   if (value!.isEmpty) {

    return 'Enter email';

   }

   return null;

 },

),
```

👉 Validator Logic:

- If the email field is empty (value.isEmpty), it returns an error message: **"Enter email"**.
- If the field isn't empty, it returns null, meaning the input is valid.

Password Field

```
TextFormField(

 controller: passwordController,

 obscureText: true,

 decoration: InputDecoration(

   hintText: 'Password',

   prefixIcon: Icon(Icons.lock),

 ),

 validator: (value) {

   if (value!.isEmpty) {

    return 'Enter password';
    }

    return null;

   },

),
```

👉 **Validator Logic:**

- Similar to the email field. If the password is empty, it returns **"Enter password"**.

Step 3: Validation Trigger

When the **login button** is pressed, we trigger the form validation using:

```
if (_formKey.currentState!.validate()) {

  // If validation passes, you can handle login here!

}
```

How It Works:

1. validate() checks the validator function of each field in the form.
2. If all validators return null, the form is considered valid, and you can proceed with login logic.
3. If any validator returns an error message, that message is displayed below the respective field.

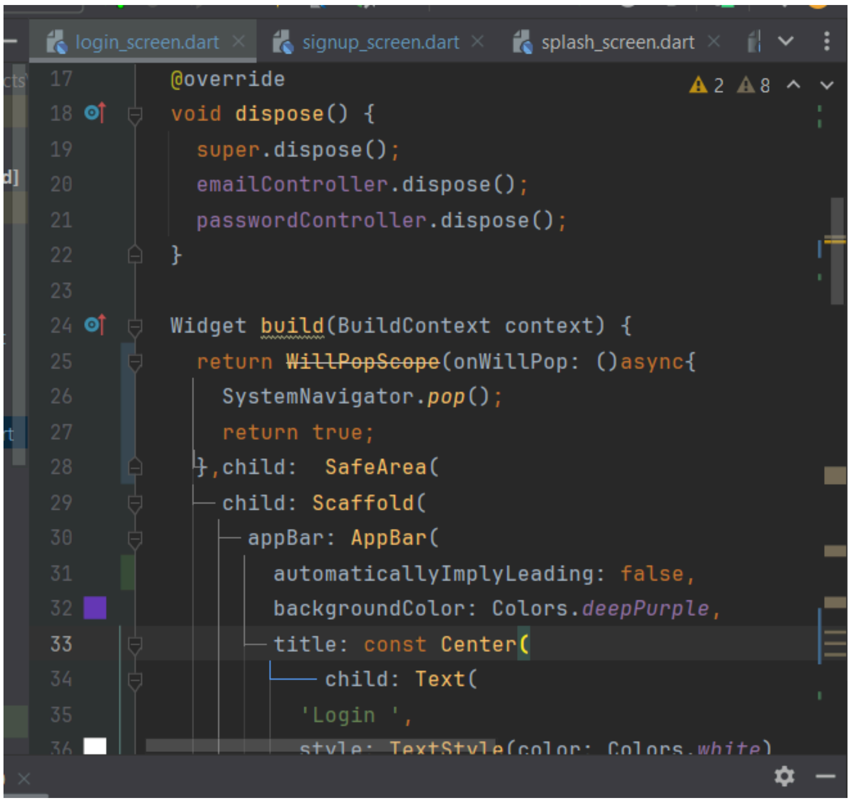
Step 4: Custom Button Action

We’re using a reusable RoundButton widget for the login button. When tapped, it validates the form:

```
RoundButton(

  title: 'login',

  onTap: () {

    if (_formKey.currentState!.validate()) {

      // Login logic goes here!

      print('Form is valid. Proceed with login.');

    }

  },
```

),

part 6 SIGN UP PAGE, WILLPOP


login_screen.dart | signup_screen.dart | splash_screen.dart

```
@override
void dispose() {
  super.dispose();
  emailController.dispose();
  passwordController.dispose();
}

Widget build(BuildContext context) {
  return ~~WillPopScope~~(onWillPop: ()async{
    SystemNavigator.pop();
    return true;
  },child:  SafeArea(
    child: Scaffold(
      appBar: AppBar(
        automaticallyImplyLeading: false,
        backgroundColor: Colors.deepPurple,
        title: const Center(
          child: Text(
        'Login ',
        style: TextStyle(color: Colors.white)
```

You're making a Flutter app, and you've got a screen, right? Now, imagine you want to control what happens when someone tries to **go back** from that screen, like by pressing the back button on Android.

This code snippet is where the **magic** happens, and the star of the show is the **WillPopScope widget**! 🎬

So, what does WillPopScope do?

It's like a **bouncer** for your app's back button. Before Flutter allows the user to leave the current screen, **WillPopScope asks, "Hey, should we actually let this person leave?"**

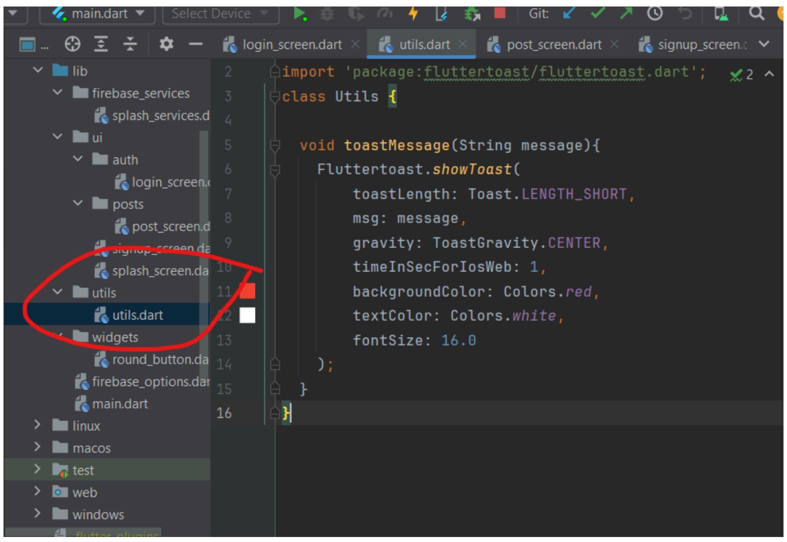
How does it work here?

1. **onWillPop:** This is the callback function where you decide what happens when the back button is pressed.
   - In this case, it's using **SystemNavigator.pop()**. Think of it like saying: 🧙 "Alright system, close the app!"
   - After that, it returns true to confirm that, **yes, the app should exit.**
2. **What's the result?** If the user hits the back button on this screen, **the entire app closes** instead of just navigating back to a previous screen.

part 7 & part 8

CONSOLE SETUP:

🌐 **Part-7 | Flutter How to Create Account/Sign Up with Firebase | Flutter Firebase Tutorials...** (4:30)



```
import 'package:fluttertoast/fluttertoast.dart';
class Utils {

  void toastMessage(String message){
    Fluttertoast.showToast(
        toastLength: Toast.LENGTH_SHORT,
        msg: message,
        gravity: ToastGravity.CENTER,
        timeInSecForIosWeb: 1,
        backgroundColor: Colors.red,
        textColor: Colors.white,
        fontSize: 16.0
    );
  }
}
```

it's our handy-dandy utility toolbox that holds reusable functions for your app

What's going on in utils.dart?

1. **This is a class named Utils.** Think of it like a box where you can put useful tools that you'll use across your entire app.
2. **Inside it, we've got the toastMessage function.** This is your **shortcut** to showing a toast message anywhere in your app.
   💡 A **toast** is a little popup message that shows briefly on the screen, like "Login successful!" or "Error, try again!"

Authentication logic (login)

🔥 Step 1: Setting up FirebaseAuth

First up, you've got the FirebaseAuth instance right here:

final _auth = FirebaseAuth.instance;

This is like the gatekeeper for your Firebase Authentication system. It lets you access all the juicy authentication features like login, signup, and logout.

🚪 Step 3: The Login Method

Here's where the real magic happens!

void login() {

  _auth

    .signInWithEmailAndPassword(

```
    email: emailController.text.toString(),
    password: passwordController.text.toString()
  )
```

- signInWithEmailAndPassword: This is like saying, "Hey Firebase! Here’s my email and password. Let me in!"
- The email and password are pulled directly from the text controllers (emailController and passwordController).

🎉 Step 4: Success Handling

If Firebase gives the green light:

```
.then((value) {
  Utils().toastMessage(value.user!.email.toString());
  Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => PostScreen())
  );
})
```

- The .then block executes when the login is successful.
- Toast Message: You’re displaying the user’s email as a quick popup (using Utils().toastMessage)—a nice touch for user feedback. 🎊
- Navigation: After logging in, the user gets redirected to the PostScreen. Think of this like taking them to the app's main area after they've signed in.

🛑 Step 5: Error Handling

But what if something goes wrong? Firebase doesn’t let them in? 🤔

```
.onError((error, stackTrace) {
  debugPrint(error.toString());
  Utils().toastMessage(error.toString());
  setState(() {
    loading = false;
  });
```

});

- onError: This is where you handle the "Uh-oh!" moments.
- You're printing the error to the console for debugging.
- Toast Message: A popup shows the error to the user, so they know what's wrong (e.g., "Incorrect password" or "User not found").
- Loading State: If you have a loading spinner, you're turning it off with setState.

Part 9 if user is already logged in/ logged out

```
import 'package:firebase/ui/auth/login_screen.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:firebase/utils/utils.dart';

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

  @override
  State<PostScreen> createState() => _PostScreenState();
}

class _PostScreenState extends State<PostScreen> {
  final auth = FirebaseAuth.instance;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        automaticallyImplyLeading: false,
        backgroundColor: Colors.deepPurple,
        title: const Center(
            child: Text(
          'Post Screen ',
          style: TextStyle(color: Colors.white),
        )),
        actions: [
          IconButton(
              onPressed: () {
                auth.signOut().then((value) {
                  Navigator.push(context,
                      MaterialPageRoute(builder: (context) => LoginScreen()));
                }).onError((error,stackTrace){
                  Utils().toastMessage(error.toString());
                });
            },
            icon: Icon(Icons.logout_outlined))
        ],
      ),
    );
  }
}
```

What’s Happening Here?

auth.signOut():

- This is a **FirebaseAuth** method that logs out the currently signed-in user from the Firebase authentication session.
- It’s an **asynchronous operation**, so it returns a **Future**. That’s why we’re using .then(...) to handle what happens after the sign-out is successful.

.then((value) { ... }):

- If the sign-out operation is successful, the code inside the .then block will execute.
- In this case, it navigates the user to the **LoginScreen**:

part 10 login with phone number authentication

consolde/provider setup -

🌐 Part - 10 | Flutter Firebase Phone Authentication || How to Login with Phone Number with Firebase

10:38