Struggling with authStateChanges in Flutter

Asked 3 months ago Active today Viewed 1k times

3

1

Whenever the user closes the app, they have to re-log back in. I looked and tried to implement authStateChanges. But yet my app is still forcing users to log back in after they close the app. In the App class, you can see that I tried to do exactly the authStateChange but nothing seems to be happening, unfortunately.

```
Future<void> main() async {


WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(App());
}

// Firebase Auth Instance
FirebaseAuth auth = FirebaseAuth.instance;

class MyApp extends StatelessWidget {
  final Future<FirebaseApp> _initialization = Firebase.initializeApp();



// This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
  title: 'Tanbo',
  theme: ThemeData(
    primarySwatch: Colors.blue,
  ),
  home: LoginPage(),
);


}
```

```
@override
  Widget build(BuildContext context) {
    return FutureBuilder(
      // Initialize FlutterFire
      future: _initialization,


builder: (context, snapshot) {
    final user = FirebaseAuth.instance.authStateChanges().listen((User user) {
      if (user == null) {
        print('User signed out');
      } else {
        print('User signed in');
      }
    });

    // Check for errors
    if (snapshot.hasError) {
      return ErrorHandler();
    }

    // Once complete, show your application
    if (snapshot.connectionState == ConnectionState.done) {
      // If the user isn't logged in, we will be sent to sign up page.
      if (user != null) {
        return MyApp();
      } else {
        // If the user is logged in, TabHandler will be shown.
        return TabHandler();
      }
    }

    // Otherwise, show something whilst waiting for initialization to complete
    return LoadingHandler();
  },
);


 }
}
```


Doug Stevenson
227k 25 256 285


Jazil Zaim
41 7

You have to respond to changes in state of the user **inside** the callback that you pass to listen(), not immediately after you set up the listener. The user is only effectively signed in again after some time has passed after you add the listener, and only your listener will be aware of that change. – Doug Stevenson Oct 25 '20 at 5:11

3 Answers

Active | Oldest | Votes

9

The problem is you have explicitly made your login page as your homepage. When the app opens, it will automatically read through the `main.dart` file and see login page as the assigned Homepage.

This is how you fix it. And also make an authentication system where you can get the logged in user's ID at any point in the app.

What you need:

1. Provider dependency of whichever version. Preferably the latest
2. A smile- Stop frowning. You're about to fix your problem
3. Firebase Auth dependency. At whichever version. I'm going to give you a fix for the newest version, and for decidedly older versions.

Step 0: Add the needed dependencies and run flutter pub get. This one is a no brainer.

Step1: Make an `auth_services` class: Code is below

For older versions of `firebase auth` :

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:google_sign_in/google_sign_in.dart';
```

```
      _firebaseAuth.onAuthStateChanged.map(
            (FirebaseUser user) => user?.uid,
      );

  // GET UID
  Future<String> getCurrentUID() async {
    return (await _firebaseAuth.currentUser()).uid;
  }
}
```

For newer versions of firebase auth dependency:

```
class AuthService {
  final FirebaseAuth _firebaseAuth = FirebaseAuth.instance;
  final GoogleSignIn _googleSignIn = GoogleSignIn();

  Stream<User> get onAuthStateChanged => _firebaseAuth.authStateChanges();

  // GET UID
  Future<String> getCurrentUID() async {
    return _firebaseAuth.currentUser.uid;
  }
}
```

Explanation : I have made this class to handle all auth functions. I am making a function called onAuthStateChanged that returns a stream of type User (ids for older versions of firebase auth) and i am going to listen to this stream to find out whether there is a user logged in or not.

Step 2: Create the auth provider that is going to wrap the entire app and make it possible to get our user id anywhere in the app.

Create a file called auth_provider.dart. The code is below.

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

class Provider extends InheritedWidget {
  final AuthService auth;
```

```
  @override
  bool updateShouldNotify(InheritedWidget oldWiddget) {
    return true;
  }

  static Provider of(BuildContext context) =>
      (context.dependOnInheritedWidgetOfExactType<Provider>());
}
```

The next step is to wrap the entire app in this provider widget and set the home controller as the homepage.

Step 3: On the main.dart file, or anywhere really, make a class called HomeController which will handle the logged in state, and set the home Controller as the assigned homepage.

NB: I have set a container of color black to be shown as the app is loading to determine if a user is logged in or not. It is a fairly fast process but if you want to, you can set it to be container of the theme color of your app. You can even set it to be a splash screen. (Please note. This containeris shown for about 1 second at most. From my experience)

Code is below: Import all the necessary things

```
void main() {
 //WidgetsFlutterBinding.ensureInitialized();
 // await Firebase.initializeApp();
//only add these if you're on the latest firebase
  runApp(MyApp());
}

 class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Provider(
      auth: AuthService(),
      child: MaterialApp(
            title: 'Dreamora',
            theme: ThemeData(
              // fontFamily: "Montserrat",
              brightness: Brightness.light,
```

```
              ),
              primarySwatch: Colors.purple,
              visualDensity: VisualDensity.adaptivePlatformDensity,
            ),
            home: HomeController(),
          );
        },
      ),
    ),}
//(I think i have messed up the brackets here, but, you get the
//gist)

class HomeController extends StatelessWidget {
  const HomeController({Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final AuthService auth = Provider.of(context).auth;

    return StreamBuilder(
      stream: auth.onAuthStateChanged,
      builder: (context, AsyncSnapshot<String> snapshot) {
        if (snapshot.connectionState == ConnectionState.active) {
          final bool signedIn = snapshot.hasData;
          return signedIn ? DashBoard() : FirstView();
        }
        return Container(
          color: Colors.black,
        );
      },
    );
  }
}
```

Explanation: The home controller is just a steam builder that listens to the steeam of auth changes we made. If there is any stuff, the user is logged in. If not, he is logged out. Fairly simple. There is like a 1 second lag though in between determining a user's logged in state. Soo, i am returning a black container in that time. The user will see the screen turn black for about a second after opening the app and then boom. Homepage

IMPORTANT: You should wrap your entire app with provider. This is important. Do not forget it.

There you go. Enjoy

Share Improve this answer Follow

edited Oct 25 '20 at 6:12

answered Oct 25 '20 at 6:05

Simeon

198 8

0

I believe it does not work because of this issue - https://github.com/FirebaseExtended/flutterfire/issues/3356

Solved by downgrading firebase-app.js and firebase-auth.js versions to 7.20.0

and replacing authStateChanges() method with userChanges()

```
FirebaseAuth.instance
    .userChanges()
    .listen((User user) {
        if (user == null) {
            print('User is currently signed out!');
        } else {
            print('User is signed in!');
        }
});
```

Share Improve this answer Follow

answered 8 hours ago

Roman Chervotkin

1 1

New contributor

Use

```
      print('User is signed in!');
    }
  });
```

Share Improve this answer Follow

edited Jan 30 at 19:37

β.εηοιτ.βε
14.6k 11 39 52

answered Jan 30 at 17:51

Richie
385 4 7

3 While this code may provide a solution to the question, it's better to add context as to why/how it works. This can help future users learn, and apply that knowledge to their own code. You are also likely to have positive feedback from users in the form of upvotes, when the code is explained. – Amit Verma Jan 30 at 18:33