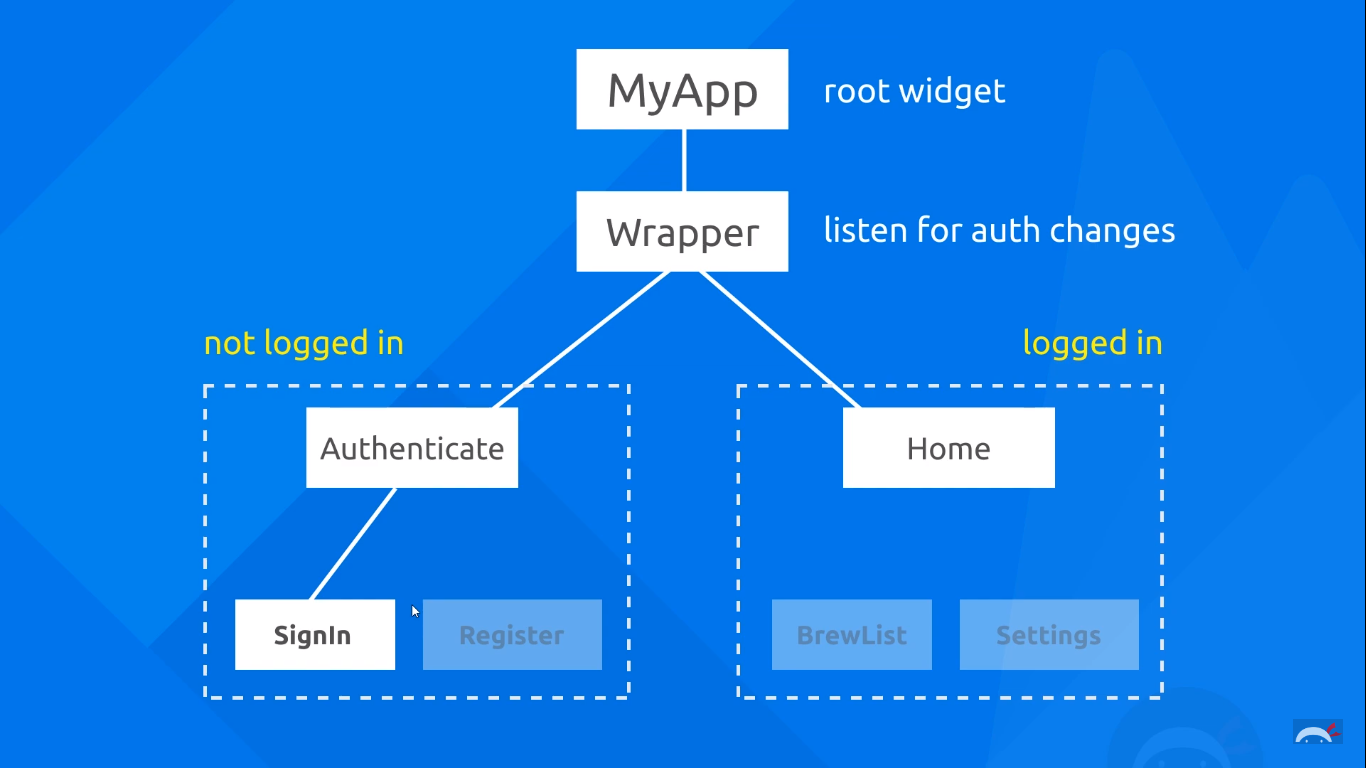
Streams:

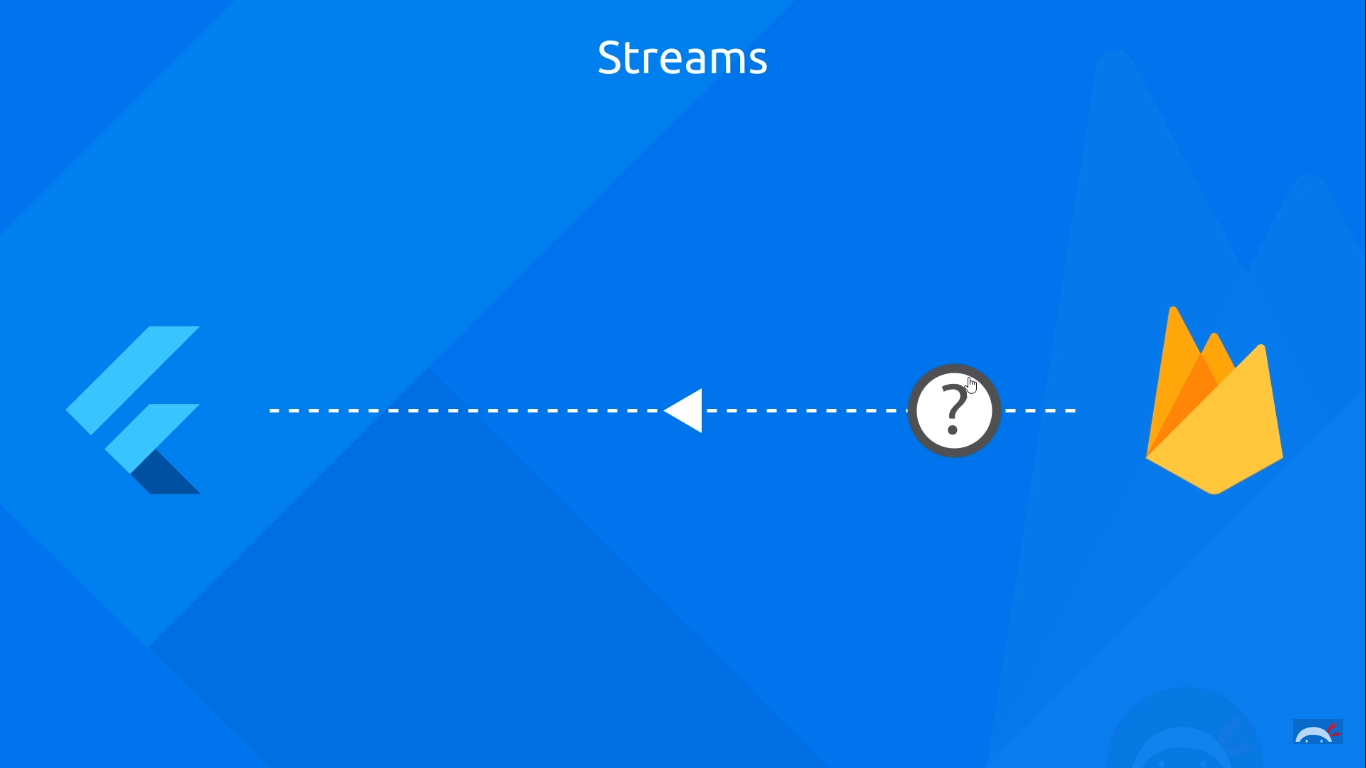
Till now we have developed the app till this stage. We can now sign in anonymously, and in return we get access to the user information in the form of a FirebaseUser object which we then converted into a custom ‘User’ object.



Now what we want to happen is to identify this authentication change, and then display the home screen to the user when he logs in. Moreover, if in the future, we could also listen to THAT authentication change, and display the login screen again to the user. Thus, essentially we are protecting our home screen from unauthenticated uses.

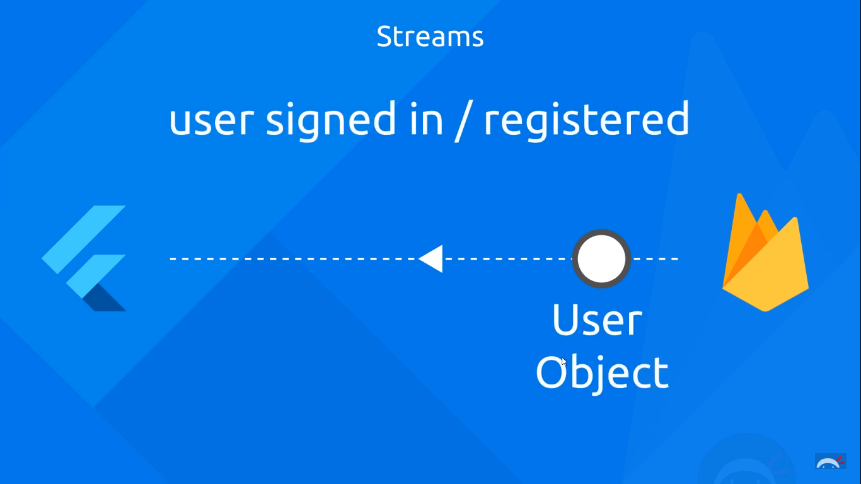
Remember, that in our ‘authenticate.dart’ file, we are supposed to be listening to these authentication changes. We want the app to switch between the home screen and login screen dynamically. In order to do so, we’ll be using a ‘stream’.

Remember what we discussed about Futures? (If not, get a recap [here](https://youtu.be/OTS-ap9_aXc).) We said that a Future was nothing but a box that the calling function receives. When the asynchronous task completes, it adds processed data to this box, and then opens it. The future then returns this data to the caller. Similarly, if the async task ends with an error, the future returns an error to the calling function.



Streams are nothing but Futures, but unlike futures which can only process single data, a ‘stream’ processes a stream of data. This means that it can execute async functions and tasks on data ***AS IT ARRIVES ON THE EVENT LOOP.*** Our auth changes are nothing but streams. It is just piece of data which is generated as the user logs in or out, and we need processing logic to deal with it AS IT HAPPENS—ergo **STREAMS!** A stream has a listen method by which we can listen to the stream data which is returned by a function call (like a file access on a disk). A stream has 2 ends: The ‘Stream’ end – where data arrives, and there is the ‘Sink’ end – where data is inserted.

Learn more in depth about streams [here](https://youtu.be/nQBpOIHE4eE).

The FirebaseAuth service already has a stream that we can listen to. We can do this by invoking the onAuthChanged function. We shall do this in our auth.dart file which has the ‘AuthService’ class, and also has all the methods for signing-in, signing-out,etc.

Now, in order to create a stream we shall use the get keyword to access the Stream end. (Here we are not creating a stream from scratch, but we are essentially hijacking a stream). The syntax for that is:

Stream <Return\_Type> get Stream\_Name {

//code;

//return statement;

}

We shall therefore invoke the onAuthStateChanged() function in order to get the stream. We shall then return this stream. Our stream therefore looks like:

Stream<FirebaseUser> get user{

    return \_auth.onAuthStateChanged;

}

Now, this is absolutely fine. But as we said earlier, we don’t want to work with the FirebaseUser object, but instead, we want to work with our custom user object which just contains the uid as of now. Thus, what we could do is to *map* this stream of FirebaseUser objects to objects of our custom ‘User’ class. We take the help of the ‘map’ function for that.

The map method takes in a function as the parameter, returns the result of this function to its caller.

Stream<FirebaseUser> get user{

    return \_auth.onAuthStateChanged.map((){});

}

Our function will take a FirebaseUser instance as the input parameter, and will return a ‘User’ object as the output. And we already have a function which does that, don’t we? Yes! It’s the userFromFirebaseUser(FirebaseUser user) function.

One more thing, upon doing so, this stream will no longer be a FirebaseUser stream, but instead it will be a ‘User’ stream.

//auth changed user stream

  Stream<User> get user{

    return \_auth.onAuthStateChanged.map((FirebaseUser user)

    {

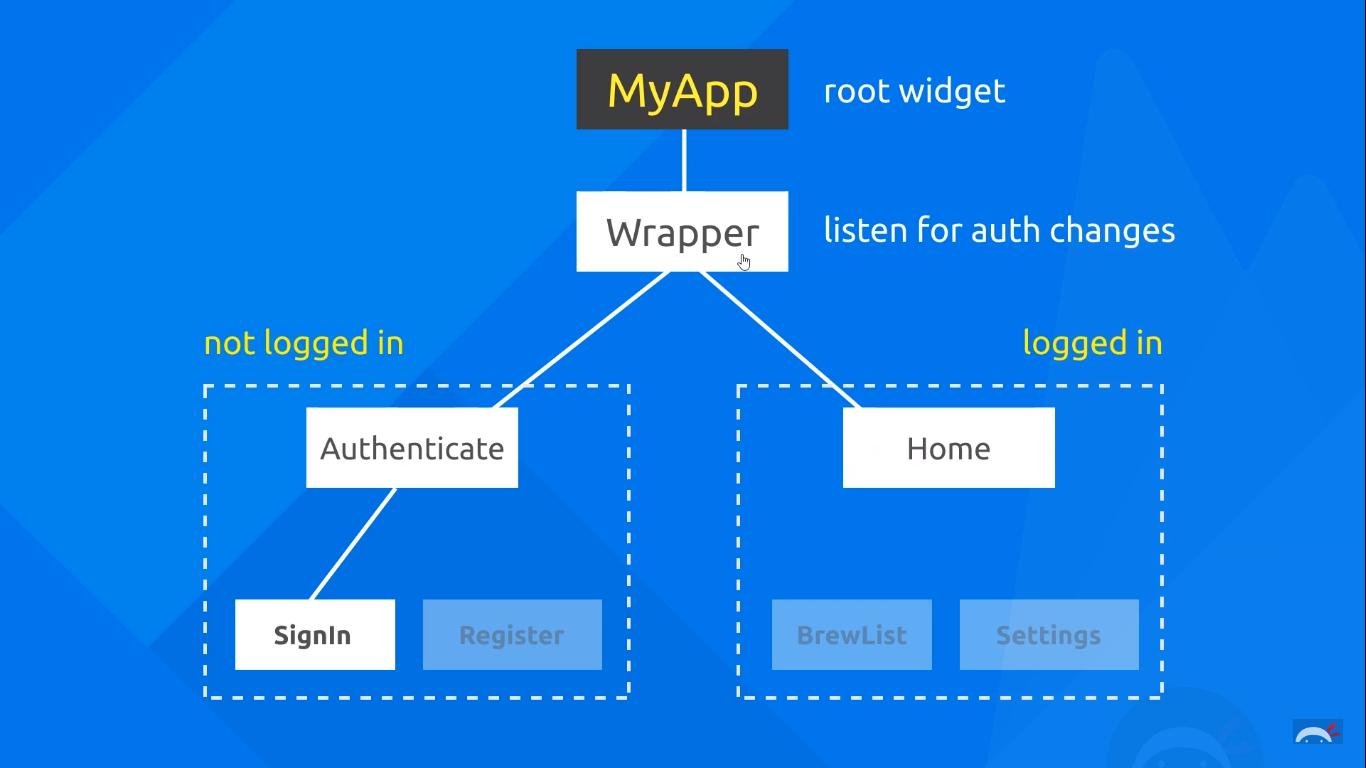
      return \_userFromFirebaseUser(user);

    });

}

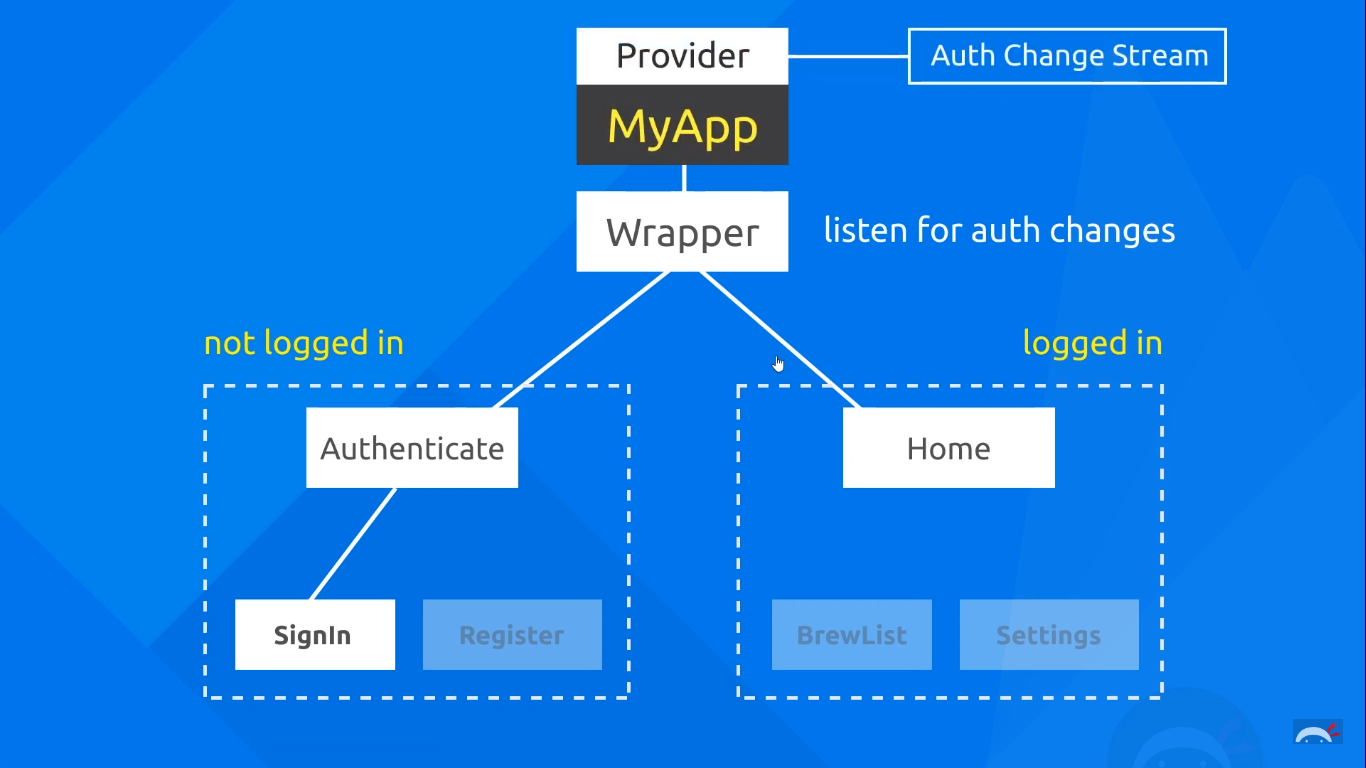
The provider package:

Now we have our ‘auth’ stream set up in our authService class. We just have to find a way to use it in our wrapper class.



As discussed earlier we need the wrapper widget to continuously listen to auth changes. If the object returned by the stream is a ‘null’ value, it implies that the user has logged out, and we need to display the Authenticate() widget. On the other hand, if the authStream returns a valid User object, then we need to display the Home() widget which represents the home screen.

We need a way to access the stream and pass the stream data downwards to all children of the Widget tree. The way we do exactly this is by a package called ‘Provider’.



The ‘provider’ package is a Google recommended solution for state management in flutter. We wrap the root widget with a provider, and pass a stream to it. Whenever we receive data inside the stream, the provider package passes it down the widget tree. Get the provider package [here](https://pub.dev/packages/provider).

Now we head on to the main.dart file and wrap the root widget (i.e. MyApp) with the provider widget. In particular we’ll be using a ‘StreamProvider’ we shall be using a specific method called ‘StreamProvider.value’. This takes in a property called ‘valu’ wherein we specify the stream. In order to specify the stream, we create an anonymous instance of the AuthService class, and access the ‘user’ stream that we just created. We also need to specify what kind of data we’ll be listening to from the stream. This will be data of the type ‘User’. Thus, we’ll specify this to the StreamProvider within angular brackets. The code looks like:

import 'package:cupped\_lightning/models/user.dart';

import 'package:cupped\_lightning/screens/wrapper.dart';

import 'package:cupped\_lightning/services/auth.dart';

import 'package:flutter/material.dart';

import 'package:provider/provider.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  // This widget is the root of your application.

  @override

  Widget build(BuildContext context) {

//StreamProvider listening to ‘user’ stream of datatype ‘User’

    return StreamProvider<User>.value(

        value: AuthService().user,

          child: MaterialApp(

        home: Wrapper(),

        theme: ThemeData(

    // Define the default brightness and colors.

    brightness: Brightness.dark,

    primaryColor: Colors.pink,

    accentColor: Colors.pinkAccent,

    textSelectionHandleColor: Colors.pinkAccent

    ),

  )

);

  }

}

In order to access the provider data inside the wrapper.dart file, we need to user the Provider.of() method. We’ll invoke this inside the build method as we need the context object.

final user=Provider.of<User>(context);

Since we need to dynamically decide between the Home() and Authenticate widget, we place in a few ‘if’ statements, and our wrapper.dart file looks like:

import 'package:cupped\_lightning/models/user.dart';

import 'package:cupped\_lightning/screens/authenticate/authenticate.dart';

import 'package:cupped\_lightning/screens/home/home.dart';

import 'package:flutter/material.dart';

import 'package:provider/provider.dart';

class Wrapper extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    final user=Provider.of<User>(context);

    print(user);

    //Return either Home or Authenticate widgets

    if(user==null)

    {

      return Authenticate();

    }

    else {

       return Home();

    }

  }

}

Signing Out:

Let’s now add a sign out button to our home screen! We’ll add our sign out button to the AppBar using the actions property which takes in a list of widgets. Currently we only have one button in the list. Flushing out the UI code, our home.dart looks like this:

import 'package:cupped\_lightning/services/auth.dart';

import 'package:flutter/material.dart';

class Home extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      backgroundColor: Colors.grey[900],

      appBar: AppBar(

      title: Text('Cupped Lightning'),

      backgroundColor: Colors.grey[850],

      elevation: 0.0,

      actions: <Widget>[

        FlatButton.icon(

          onPressed: () {},

        icon: Icon(

          Icons.person,

          color: Colors.grey[200]

        ),

         label: Text(

           'Log Out',

           style: TextStyle(

             color: Colors.grey[200]

           ),

           ),

         )

      ],

      ),

    );

  }

}

Now, inside the AuthService class in auth.dart we’ll need to define the signOut method. Here we shall call the signOut method of the FirebaseAuth object. This method, unlike the signIn method doesn’t return any value. The sign out method looks like:

Future signOut() async{

    try{

      return await \_auth.signOut();

    }

    catch(e)

    {

      print(e.toString());

      return null;

    }

  }

}

In the home.dart file, we’ll need an instance of the AuthService class in order to access this signOut method. So, that’s what we’ll do:

AuthService \_auth=AuthService();

We’ll also need to make an asynchronous call to this signOut method in the onPressed property of the signout button. It looks like:

FlatButton.icon(

          onPressed: () async{

           await \_auth.signOut();

        },

        icon: Icon(

          Icons.person,

          color: Colors.grey[200]

        ),

         label: Text(

           'Log Out',

           style: TextStyle(

             color: Colors.grey[200]

           ),

           ),

         )

Our entire home.dart file looks like:

import 'package:cupped\_lightning/services/auth.dart';

import 'package:flutter/material.dart';

class Home extends StatelessWidget {

  AuthService \_auth=AuthService();

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      backgroundColor: Colors.grey[900],

      appBar: AppBar(

      title: Text('Cupped Lightning'),

      backgroundColor: Colors.grey[850],

      elevation: 0.0,

      actions: <Widget>[

        FlatButton.icon(

          onPressed: () async{

           await \_auth.signOut();

        },

        icon: Icon(

          Icons.person,

          color: Colors.grey[200]

        ),

         label: Text(

           'Log Out',

           style: TextStyle(

             color: Colors.grey[200]

           ),

           ),

         )

      ],

      ),

    );

  }

}