

MINI PROJECT

(2021 - 2022)

A Chat App

(App Development)

MID-TERM REPORT

Department of Computer Engineering &

Application Institute of Engineering



Technology

TEAM MEMBERS

Deepak Awasthi

(181500202)

Maneesh Kumar

(181500363)

Anoop Kumar

(181500104)

Mohit kumar Jalan

(181500390)

SUPERVISED BY:

Mr. Neeraj Khanna

(Technical Trainer)

Contents

Abstract

1. Introduction

- a. General Introduction to the topic
- b. Area of Computer Science
- c. Hardware and Software Requirements

2. Objectives

3. Implementation Details

4. Progress till Date & The Remaining work

5. Some Screenshots

6. References

Abstract

Chat refers to the process of communicating, interacting and/or exchanging messages over the Internet. It involves two or more individuals that communicate through a chat-enabled service or software. Chat may be delivered through text, audio or video communication via the Internet.

A chat application has basic two components, viz server and client. A server is a computer program or a device that provides functionality for other programs or devices. Clients who want to chat with each other connect to the server. The chat application we are going to make will be more like a chat room, rather than a peer to peer chat. So this means that multiple users can connect to the chat server and send their messages. Every message is broadcasted to every connected chat user.

Introduction

1.1 General Introduction of the topic

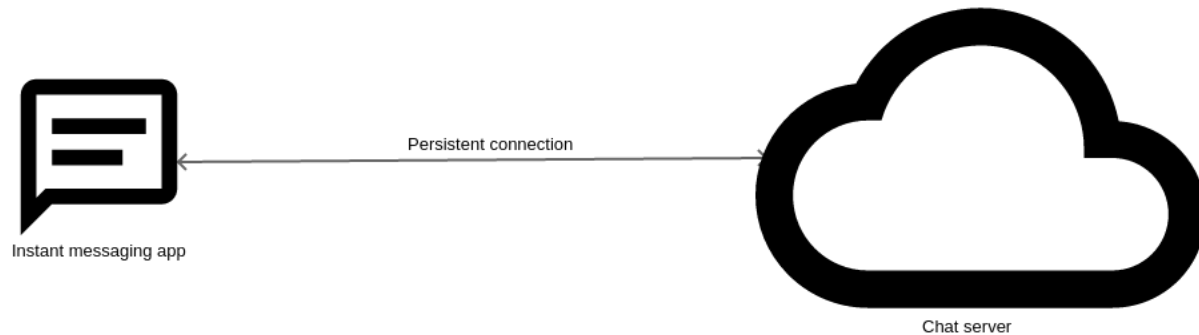
Chatting system is a two-way communication that is used by the users for the purpose of exchanging text messages and files between the system's users. It is more common to say as a peer-to-peer system which supports two way communications. The user of the system is defined as client-server. Chatting system is a distributed programming which consists of two distributed components, chat server and chat client. Chat client supports all communication including requesting chat server location information from a location server and displaying received chat messages. Chat server will conduct chat sessions and manage all chat clients. Basically the chat client starts the chat session by requesting the communication parameter (server name and port number). There are two types of communication between client-servers which are control messages (used to join and leave chat sessions, create chat room and switch to chat room) and chat message (supports only public chat messages).

About Chat Application

Chat App is an application used to communicate with our known people. Messaging apps ("social messaging" or "chat applications") are apps and platforms that enable instant messaging. Many such apps have developed into broad platforms enabling status updates, chatbots, payments and conversational commerce (e-commerce via chat). They are normally centralised networks run by the servers of the platform's operators, unlike peer-to-peer protocols. Some examples of popular messaging apps include WhatsApp, Facebook Messenger, China's WeChat and QQ Messenger, Telegram, Viber, Line, and Snapchat.

How Chat Application Works-

A chat application has the following components: a messaging application, a server and a persistent connection.



The messaging application is the part that you see. This is the part of the system that resides on your phone, laptop or personal computer as a small app. There is a text box where you type messages and another text box where all the previous messages in the conversation are shown along with the timelines. There is a button to send the messages and on the desktop / laptop, pressing the Enter / Return key will do the job. There is a trigger to open a set of emoticons that you can use. You can achieve the same using a combination of symbols that look like the same emoticon when looked sideways. Finally, for more sophisticated apps, there will be voice / video calls.

Before being ready for use, the messaging app connects to a central server. It is only because of this connection that you are able to send messages to others who are connected to the same server and others are able to see you online and send messages to you. To establish a connection, the user must first authenticate with his/her credentials.

On the server side of the equation, there is a server-side software that listens for connections from the instant messaging client. The server maintains a map of which connections belongs to which user, so that messages can be reliably relayed to the correct recipient and marked as being sent by a certain sender.



Finally, the third component is a persistent connection. This jargonistic word simply means that the instant messaging app must remain connected to the server ALL the time. A user is seen online and is able to send and receive messages ONLY as long as the connection is held stably. Internet failures, unreliable Internet connection, company firewall rules and Internet provider restrictions will often cause the connection to fail and the instant messaging app either does not work or suffers dropped messages.

What is Flutter

Flutter is an open-source UI software development kit created by Google. It is used to develop applications for Android, iOS, Linux, Mac, Windows, Google Fuchsia, and the web from a single codebase.

The first version of Flutter was known as codename "Sky" and ran on the Android operating system. It was unveiled at the 2015 Dart developer summit, with the stated intent of being able to render consistently at 120 frames per second. During the keynote of Google Developer Days in Shanghai, Google announced Flutter Release Preview 2, which is the last big release before Flutter 1.0. On December 4, 2018, Flutter 1.0 was released at the Flutter Live event, denoting the first "stable" version of the Framework. On December 11, 2019, Flutter 1.12 was released at the Flutter Interactive event. On May 6, 2020, the Dart SDK in version 2.8 and the Flutter in version 1.17.0 were released, where support was added to the Metal API, improving performance on iOS devices (approximately 50%), new Material widgets, and new network tracking tools.

What is Dart

Dart is an open-source general-purpose programming language. It was originally developed by Google and later approved as a standard by ECMA. Dart is a new programming language meant for the server as well as the browser. Introduced by Google, the Dart SDK ships with its compiler - the Dart VM. The SDK also includes a utility -dart2js, a transpiler that generates JavaScript equivalent of a Dart Script. This tutorial provides a basic level understanding of the Dart programming language.

The first compiler to generate JavaScript from Dart code was dartc, but it was deprecated. The second Dart-to-JavaScript compiler was Frog. It was written in Dart, but never implemented the full semantics of the language. The third Dart-to-JavaScript compiler was dart2js. An evolution of earlier compilers, dart2js is written in Dart and intended to implement the full Dart language specification and semantics.

Area of Computer Science

Our Project falls under the Category of App Development of Computer science. App development means the process of designing, creating, testing and finally launching an app that is meant to satisfy the needs of many users. However, App development does not refer strictly to smartphone mobile applications. It can broadly go to installing the App in the Machine for a specific purpose like ATM machines

An app is the common slang term for a software application or software program that can be run on a computer device to accomplish a task easier and more efficiently than we could do it ourselves as mere mortals. If you have a smartphone or computer tablet, you probably have used some game apps, news apps or even map apps to help you find the local coffee shop. Application development is the name of the profession that employs people who design, develop and deploy these computer applications.

Hardware Requirements

In hardware requirement we require all those components which will provide us the platform for the development of the project. The minimum hardware required for the development of this project is as follows--

- A Laptop or a PC for app development.
- Android device for testing/debugging.

Software Requirements

Software can be defined as programs which run on our computer .It acts as petrol in the vehicle. It provides the relationship between the human and a computer. It is very important to run software to function the computer. Various software are needed in this project for its development. Which are as follows--

- VS Code Editor for coding purpose
- Android Studio for compilation of the code
- Software Development Kit(SDK)

Objective

The main objective of this project is to build a user friendly application which can be used by users to communicate with each other. We are developing this application to provide a peer to peer connection based facility to the users. With this application, we aim to provide an amazing experience to the users. This application provides users the ability to sign in, sign out, search for users, and send messages. We are using firebase for realtime messaging whenever a user sends a message to the other user.

The users will be able to chat with each other, most likely only from user to user, no group chatting will be developed, unless there is time to do so.

Implementation Details

Part1: Design User Interface for the Chat app. It involved creating different screens like Sign Up, Sign In, etc.

Part 2: Implementing the backend algorithm and connecting the app to firebase for the purpose of database and authentication.

Part 3: Designing the home screen, chat screen, friend screen, etc.

Part 4: Implementing the backend like uploading the chat to the firebase, fetching the chat from firebase, logics for the interchanging of the data between different screens.

Progress

1.) Part 1 is completed

- Created SignIn screen.
- Created SignUp screen.
- Created various files like auth, authentication, user, database, etc.

2.) Part 2 is completed

- Implementing SignIn with firebase.
- Implementing SignUp with firebase.



3.) Part 3 is remaining






4.) Part 4 is remaining

Screenshots




Start chatting with people from
all over the world.







76% 15:29

Let's Chat
with friends








Login




Sign up






Email

Password



Login



76% 15:28




Let's Chat with friends






[Login](#)[Sign up](#)

Email

Password

[Login](#)



76% 15:28

Let's Chat with friends

Login

Sign up

Username

Enter user name

Email

Enter your email



Password






Enter password

Confirm password

Enter password again

Sign up



76% 15:30

Let's Chat with friends


Login

Sign up

Email

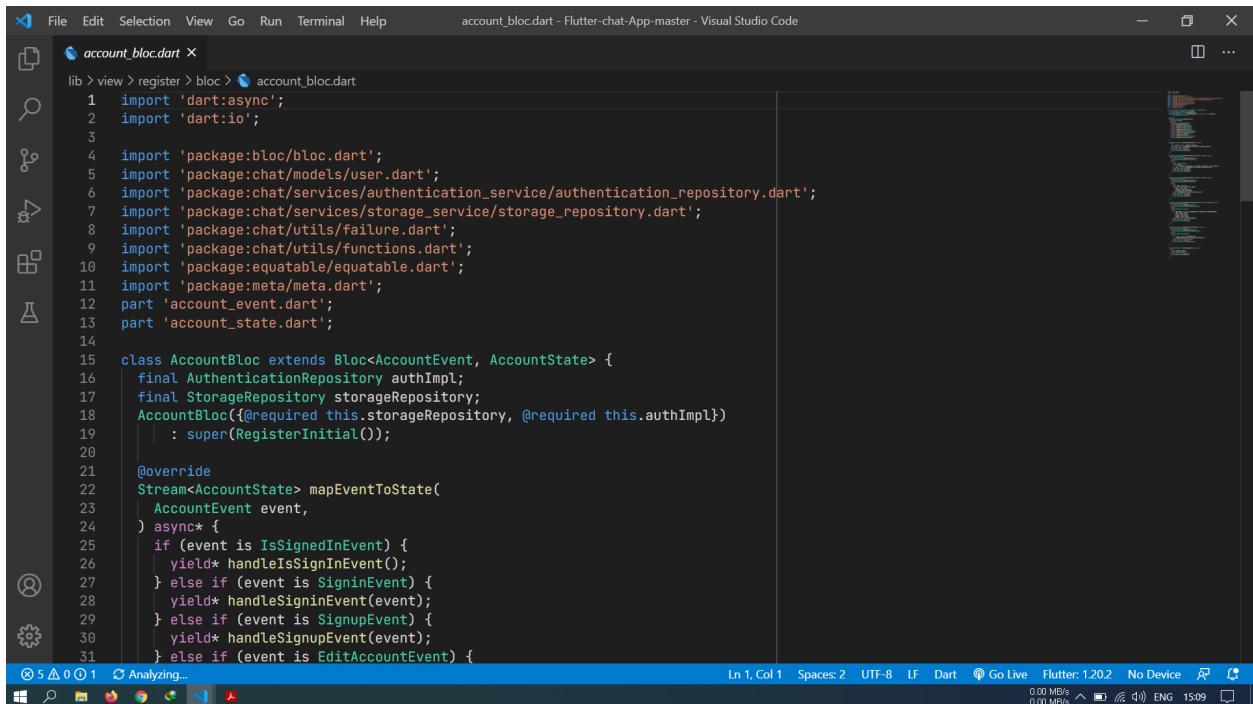
maneeshkr.123@gmail.com

Password

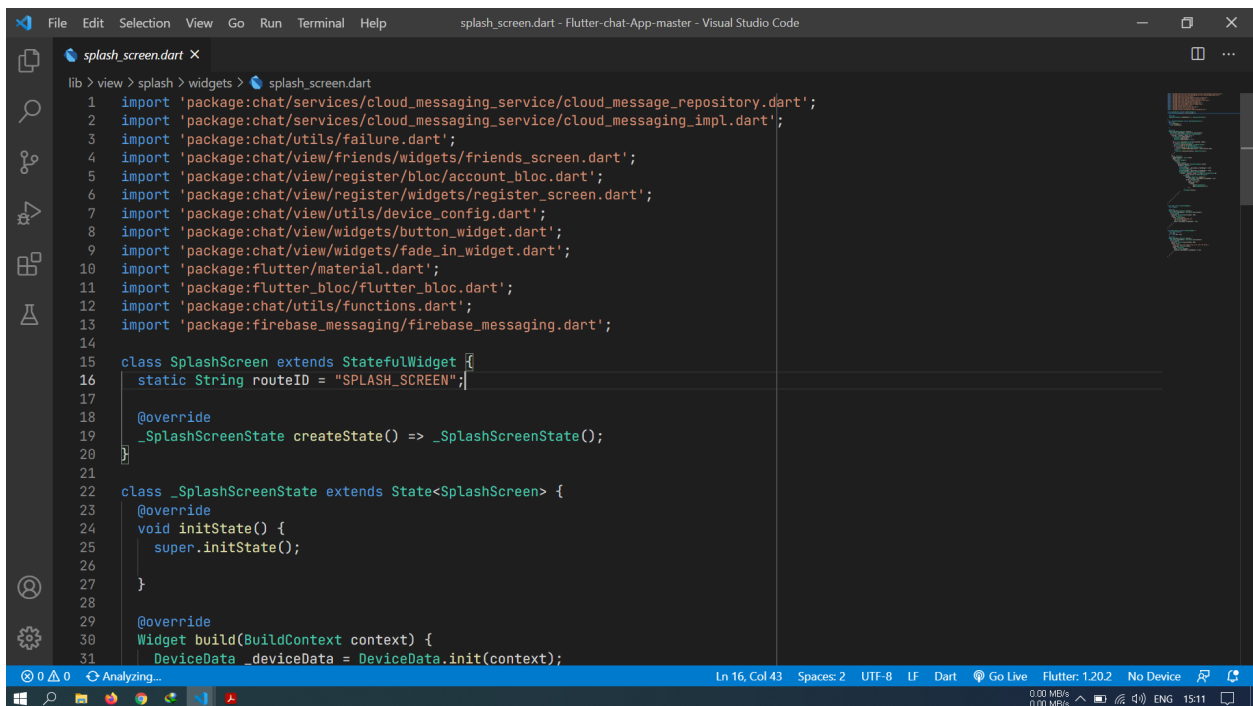
.....

Login

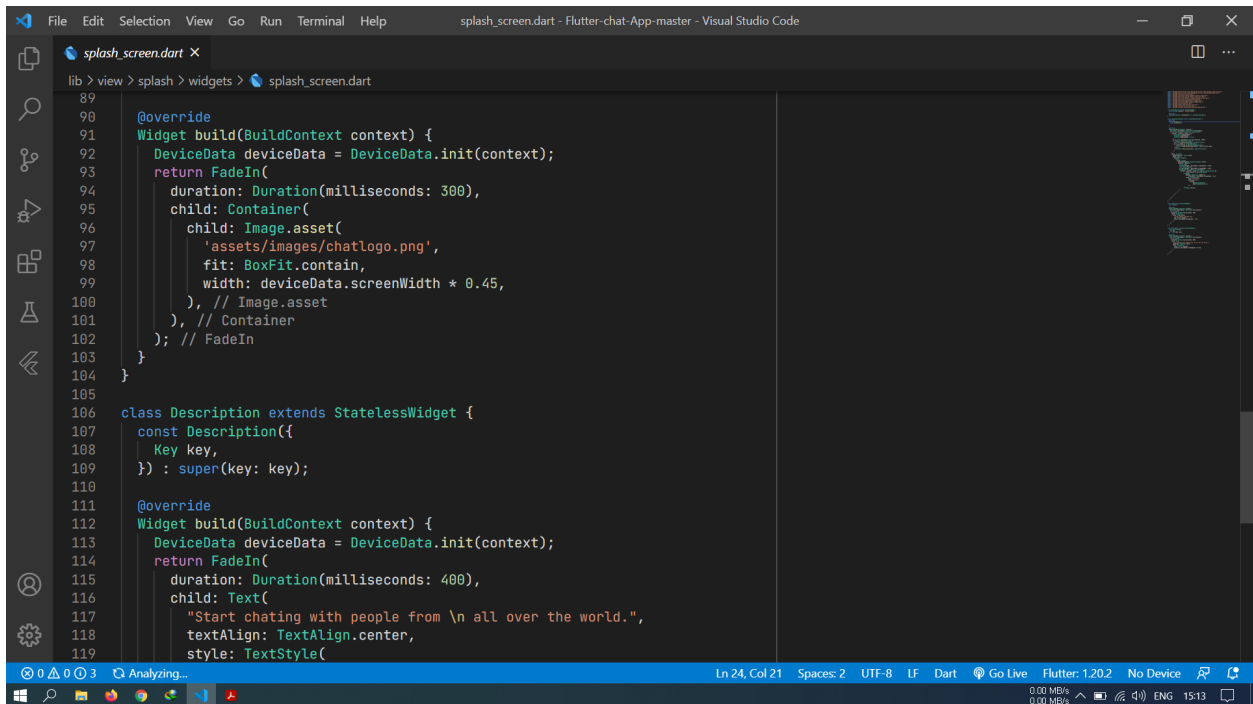
Signing in ...



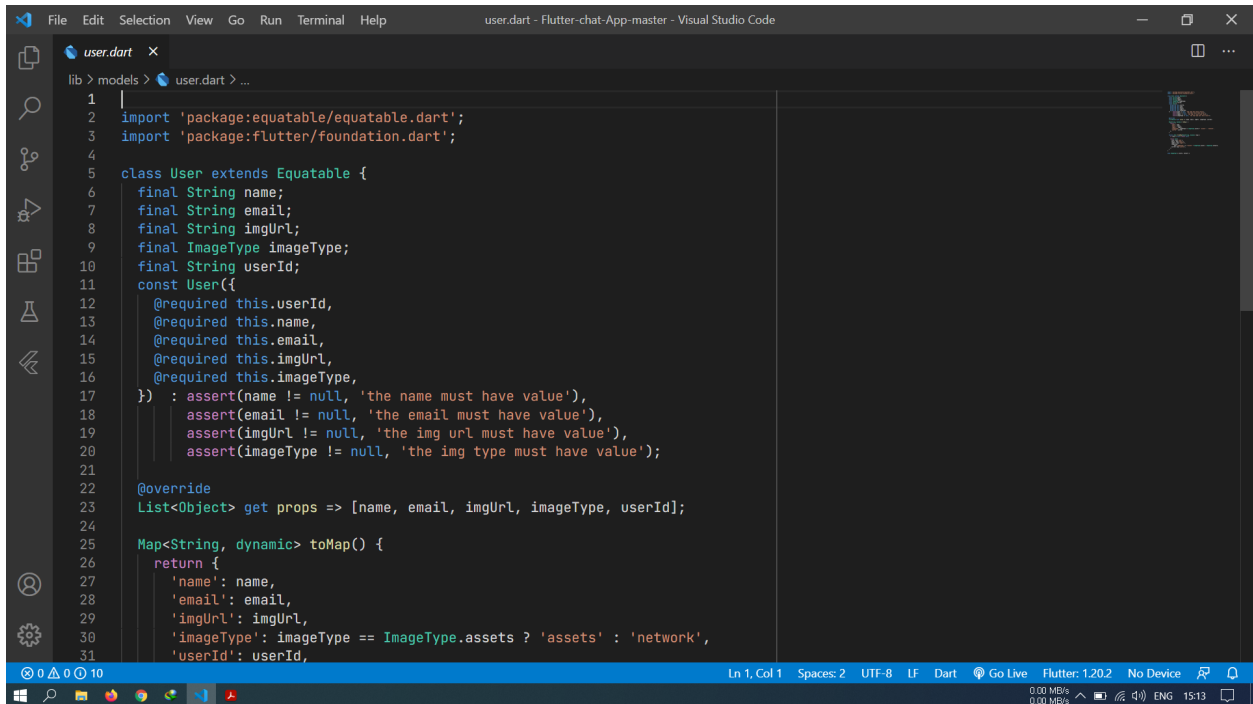
```
lib > view > register > bloc > account_bloc.dart
1 import 'dart:async';
2 import 'dart:io';
3
4 import 'package:bloc/bloc.dart';
5 import 'package:chat/models/user.dart';
6 import 'package:chat/services/authentication_service/authentication_repository.dart';
7 import 'package:chat/services/storage_service/storage_repository.dart';
8 import 'package:chat/utils/failure.dart';
9 import 'package:chat/utils/functions.dart';
10 import 'package:equatable/equatable.dart';
11 import 'package:meta/meta.dart';
12 part 'account_event.dart';
13 part 'account_state.dart';
14
15 class AccountBloc extends Bloc<AccountEvent, AccountState> {
16   final AuthenticationRepository authImpl;
17   final StorageRepository storageRepository;
18   AccountBloc({@required this.storageRepository, @required this.authImpl})
19     : super(RegisterInitial());
20
21   @override
22   Stream<AccountState> mapEventToState(
23     AccountEvent event,
24   ) async* {
25     if (event is IsSignedInEvent) {
26       yield* handleIsSignInEvent();
27     } else if (event is SignInEvent) {
28       yield* handleSignInEvent(event);
29     } else if (event is SignUpEvent) {
30       yield* handleSignUpEvent(event);
31     } else if (event is EditAccountEvent) {
```



```
lib > view > splash > widgets > splash_screen.dart
1 import 'package:chat/services/cloud_messaging_service/cloud_message_repository.dart';
2 import 'package:chat/services/cloud_messaging_service/cloud_messaging_impl.dart';
3 import 'package:chat/utils/failure.dart';
4 import 'package:chat/view/friends/widgets/friends_screen.dart';
5 import 'package:chat/view/register/bloc/account_bloc.dart';
6 import 'package:chat/view/register/widgets/register_screen.dart';
7 import 'package:chat/view/utils/device_config.dart';
8 import 'package:chat/view/widgets/button_widget.dart';
9 import 'package:chat/view/widgets/fade_in_widget.dart';
10 import 'package:flutter/material.dart';
11 import 'package:flutter_bloc/flutter_bloc.dart';
12 import 'package:chat/utils/functions.dart';
13 import 'package:firebase_messaging/firebase_messaging.dart';
14
15 class SplashScreen extends StatefulWidget {
16   static String routeID = "SPLASH_SCREEN";
17
18   @override
19   _SplashScreenState createState() => _SplashScreenState();
20 }
21
22 class _SplashScreenState extends State<SplashScreen> {
23   @override
24   void initState() {
25     super.initState();
26   }
27
28   @override
29   Widget build(BuildContext context) {
30     DeviceData _deviceData = DeviceData.init(context);
```



```
lib > view > splash > widgets > splash_screen.dart
89
90 @override
91 Widget build(BuildContext context) {
92   DeviceData deviceData = DeviceData.init(context);
93   return FadeIn(
94     duration: Duration(milliseconds: 300),
95     child: Container(
96       child: Image.asset(
97         'assets/images/chatlogo.png',
98         fit: BoxFit.contain,
99         width: deviceData.screenWidth * 0.45,
100       ), // Image.asset
101     ), // Container
102   ); // FadeIn
103 }
104
105
106 class Description extends StatelessWidget {
107   const Description({
108     Key key,
109   }) : super(key: key);
110
111   @override
112   Widget build(BuildContext context) {
113     DeviceData deviceData = DeviceData.init(context);
114     return FadeIn(
115       duration: Duration(milliseconds: 400),
116       child: Text(
117         "Start chatting with people from \n all over the world.",
118         textAlign: TextAlign.center,
119         style: TextStyle(
```



```
lib > models > user.dart > ...
1
2 import 'package:equatable/equatable.dart';
3 import 'package:flutter/foundation.dart';
4
5 class User extends Equatable {
6   final String name;
7   final String email;
8   final String imgUrl;
9   final ImageType imageType;
10  final String userId;
11  const User({
12    @required this.userId,
13    @required this.name,
14    @required this.email,
15    @required this.imgUrl,
16    @required this.imageType,
17  }) : assert(name != null, 'the name must have value'),
18      assert(email != null, 'the email must have value'),
19      assert(imgUrl != null, 'the img url must have value'),
20      assert(imageType != null, 'the img type must have value');
21
22  @override
23  List<Object> get props => [name, email, imgUrl, imageType, userId];
24
25  Map<String, dynamic> toMap() {
26    return {
27      'name': name,
28      'email': email,
29      'imgUrl': imgUrl,
30      'imageType': imageType == ImageType.assets ? 'assets' : 'network',
31      'userId': userId,
```

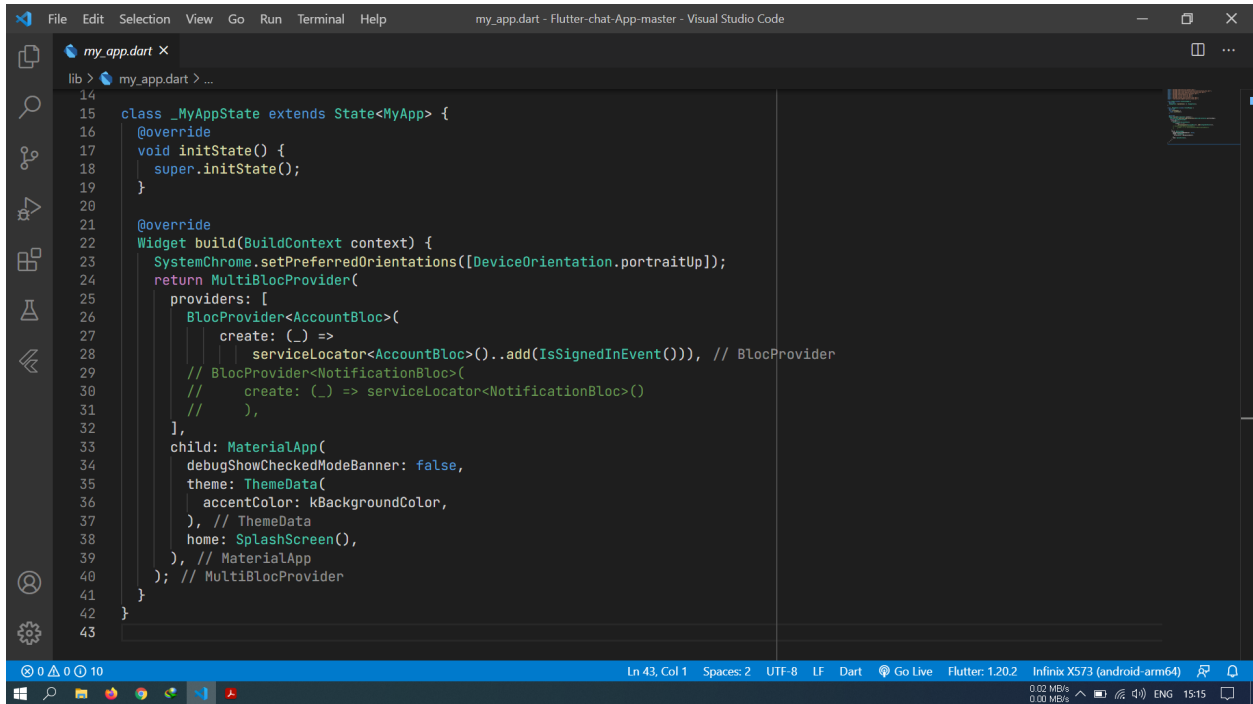
```
lib > main.dart > ...
1 import 'package:chat/bloc_observer.dart';
2 import 'package:flutter_bloc/flutter_bloc.dart';
3 import 'my_app.dart';
4 import 'package:flutter/material.dart';
5 import 'package:chat/service_locator.dart';
6
7 void main() {
8   serviceLocatorSetup();
9   Bloc.observer = SimpleBlocObserver();
10  runApp(MyApp());
11 }
12
```

Ln 1, Col 1 Spaces: 2 UTF-8 LF Dart Go Live Flutter: 1.20.2 No Device

```
lib > my_app.dart > ...
4 import 'package:cnat/view/utils/constants.dart';
5 import 'package:flutter/material.dart';
6 import 'package:flutter/services.dart';
7 import 'package:flutter_bloc/flutter_bloc.dart';
8 import 'view/splash/widgets/splash_screen.dart';
9
10 class MyApp extends StatefulWidget {
11   @override
12   _MyAppState createState() => _MyAppState();
13 }
14
15 class _MyAppState extends State<MyApp> {
16   @override
17   void initState() {
18     super.initState();
19   }
20
21   @override
22   Widget build(BuildContext context) {
23     SystemChrome.setPreferredOrientations([DeviceOrientation.portraitUp]);
24     return MultiBlocProvider(
25       providers: [
26         BlocProvider<AccountBloc>(
27           create: (_) =>
28             serviceLocator<AccountBloc>()..add(IsSignedInEvent()), // BlocProvider
29           // BlocProvider<NotificationBloc>(
30             // create: (_) => serviceLocator<NotificationBloc>()
31             // ),
32         ],
33       child: MaterialApp(
34         debugShowCheckedModeBanner: false,

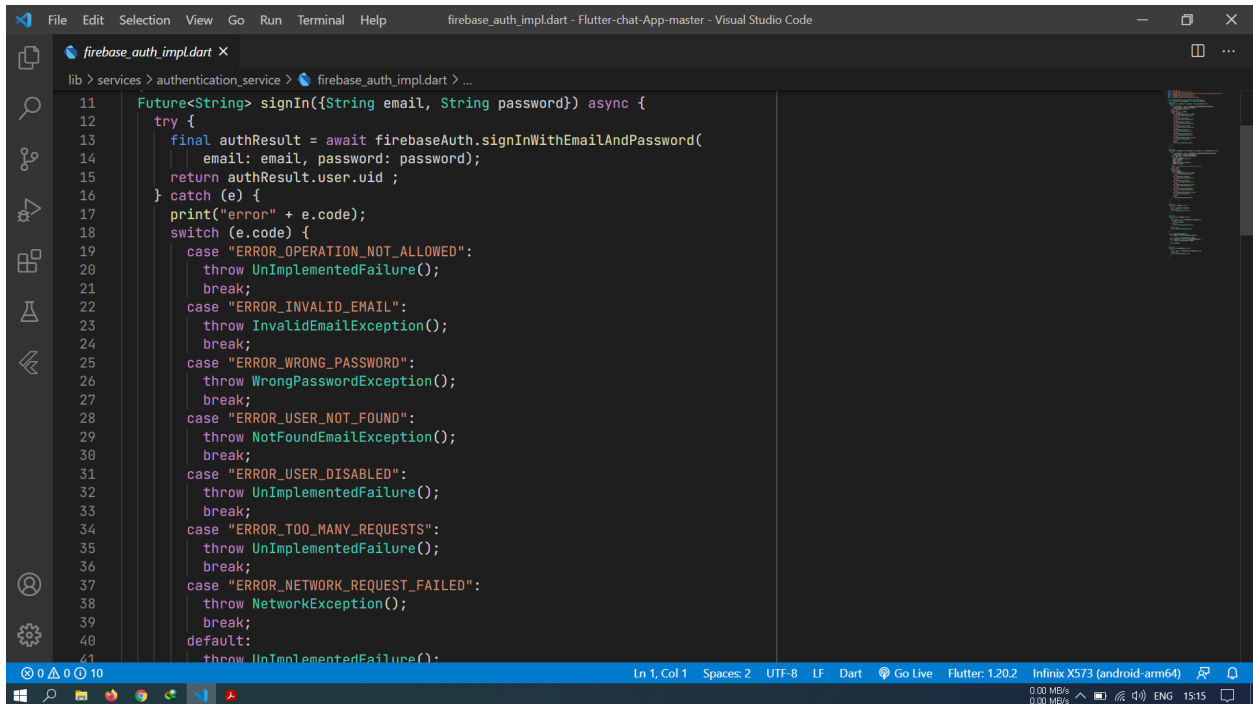
```

Ln 1, Col 1 Spaces: 2 UTF-8 LF Dart Go Live Flutter: 1.20.2 Infinix X573 (android-arm64)



```
lib > my_app.dart > ...
14
15 class _MyAppState extends State<MyApp> {
16   @override
17   void initState() {
18     super.initState();
19   }
20
21   @override
22   Widget build(BuildContext context) {
23     SystemChrome.setPreferredOrientations([DeviceOrientation.portraitUp]);
24     return MultiBlocProvider(
25       providers: [
26         BlocProvider<AccountBloc>(
27           create: (_) =>
28             serviceLocator<AccountBloc>().add(IsSignedInEvent()), // BlocProvider
29           // BlocProvider<NotificationBloc>(
30           //   create: (_) => serviceLocator<NotificationBloc>()
31           // ),
32         ],
33       child: MaterialApp(
34         debugShowCheckedModeBanner: false,
35         theme: ThemeData(
36           accentColor: kBackgroundColor,
37         ), // ThemeData
38         home: SplashScreen(),
39       ), // MaterialApp
40     ); // MultiBlocProvider
41   }
42 }
43
```

Ln 43, Col 1 Spaces: 2 UTF-8 LF Dart Go Live Flutter: 1.20.2 Infinix X573 (android-arm64) 0.02 MB/s 0.00 MB/s ENG 15:15



```
lib > services > authentication_service > firebase_auth_impl.dart > ...
11 Future<String> signIn({String email, String password}) async {
12   try {
13     final authResult = await firebaseAuth.signInWithEmailAndPassword(
14       email: email, password: password);
15     return authResult.user.uid ;
16   } catch (e) {
17     print("error" + e.code);
18     switch (e.code) {
19       case "ERROR_OPERATION_NOT_ALLOWED":
20         throw UnimplementedFailure();
21       break;
22       case "ERROR_INVALID_EMAIL":
23         throw InvalidEmailException();
24       break;
25       case "ERROR_WRONG_PASSWORD":
26         throw WrongPasswordException();
27       break;
28       case "ERROR_USER_NOT_FOUND":
29         throw NotFoundEmailException();
30       break;
31       case "ERROR_USER_DISABLED":
32         throw UnimplementedFailure();
33       break;
34       case "ERROR_TOO_MANY_REQUESTS":
35         throw UnimplementedFailure();
36       break;
37       case "ERROR_NETWORK_REQUEST_FAILED":
38         throw NetworkException();
39       break;
40       default:
41         throw UnimplementedFailure();
42     }
43   }
44 }
```

Ln 1, Col 1 Spaces: 2 UTF-8 LF Dart Go Live Flutter: 1.20.2 Infinix X573 (android-arm64) 0.00 MB/s 0.00 MB/s ENG 15:15

```
File Edit Selection View Go Run Terminal Help firebase_auth_impl.dart - Flutter-chat-App-master - Visual Studio Code

firebase_auth_impl.dart X
lib > services > authentication_service > firebase_auth_impl.dart > ...

47 Future<User> signUp({String username, String email, String password}) async {
48   try {
49     final authResult = await firebaseAuth.createUserWithEmailAndPassword(
50       email: email, password: password);
51     String photoUrl = _generateRandomPath();
52     final user = User(
53       userId: authResult.user.uid,
54       name: username,
55       email: email,
56       imageType: ImageType.assets,
57       imgUrl: photoUrl,
58     );
59     //await storageRepository.saveProfileUser(user);
60     return user;
61   } catch (e) {
62     print(e.code);
63     switch (e.code) {
64       case "ERROR_OPERATION_NOT_ALLOWED":
65         throw UnimplementedFailure();
66         break;
67       case "ERROR_WEAK_PASSWORD":
68         throw WeakPasswordException();
69         break;
70       case "ERROR_INVALID_EMAIL":
71         throw InvalidEmailException();
72         break;
73       case "ERROR_EMAIL_ALREADY_IN_USE":
74         throw EmailInUseException();
75         break;
76       case "ERROR_INVALID_CREDENTIAL":
77         throw InvalidEmailException();
78     }
79   }
80 }
```

```
File Edit Selection View Go Run Terminal Help authentication_repository.dart - Flutter-chat-App-master - Visual Studio Code

authentication_repository.dart X
lib > services > authentication_service > authentication_repository.dart > ...

1 import 'package:chat/models/user.dart';
2 import 'package:chat/service_locator.dart';
3 import 'package:chat/services/storage_service/storage_repository.dart';
4 import 'package:flutter/foundation.dart';
5
6 abstract class AuthenticationRepository {
7   // final StorageRepository storageRepository = serviceLocator<StorageRepository>();
8   Future<String> signIn({@required String email, @required String password});
9   Future<User> signUp({
10     @required String username,
11     @required String email,
12     @required String password,
13   });
14   Future<void> logout();
15   Future<String> isSignIn();
16   Future<String> getUserID();
17 }
18
```