Software Engineering Project

(Chat Application)

SUBMITTED BY : SPARSH

ROLL NUMBER :- 2019UCP1400

TABLE OF CONTENTS

Software requirement specification:

Purpose:

The main purpose of our chat Application is to connect users so that they can Share Information with each other.

Scope:

This will allow them to share their messages and images through our platform and create groups and interact with multiple people at a time and storing details in the Firebase Firestore Database.

User requirements:

Functional Requirements:

- Authentication of Admin whenever required
- All the users will have to be registered on the platform to enter with a valid and unique email id.
- A search mechanism will have to be provided so that users can search other users on the platform
- Users can chat with other users by creating a chat room which will allow to transfer messages and Images to each other
- Users can build group chatrooms and can send messages to each other, the admin of the group will have access to remove other participants from the group.
- Users will have access to change their display pictures
- users will have access to share their valuable feedback with the creator
- in Case a bug arrives, users will be able to share the bug reports.

Non-functional Requirements:

Security: This system is a critical system as it has important information regarding flights, employees and optimal measures must be taken to ensure data is safe from unauthorized users.

Availability: The system must always be on as otherwise users will not be able to transfer information.

Portability: The users access the software from various platforms and by different stakeholders. The app must be portable to all systems and the user experience must be optimal.

Interface requirements:

1. User-Interface:-

- Frontend software: Flutter and Dart
- Backend software: Firebase

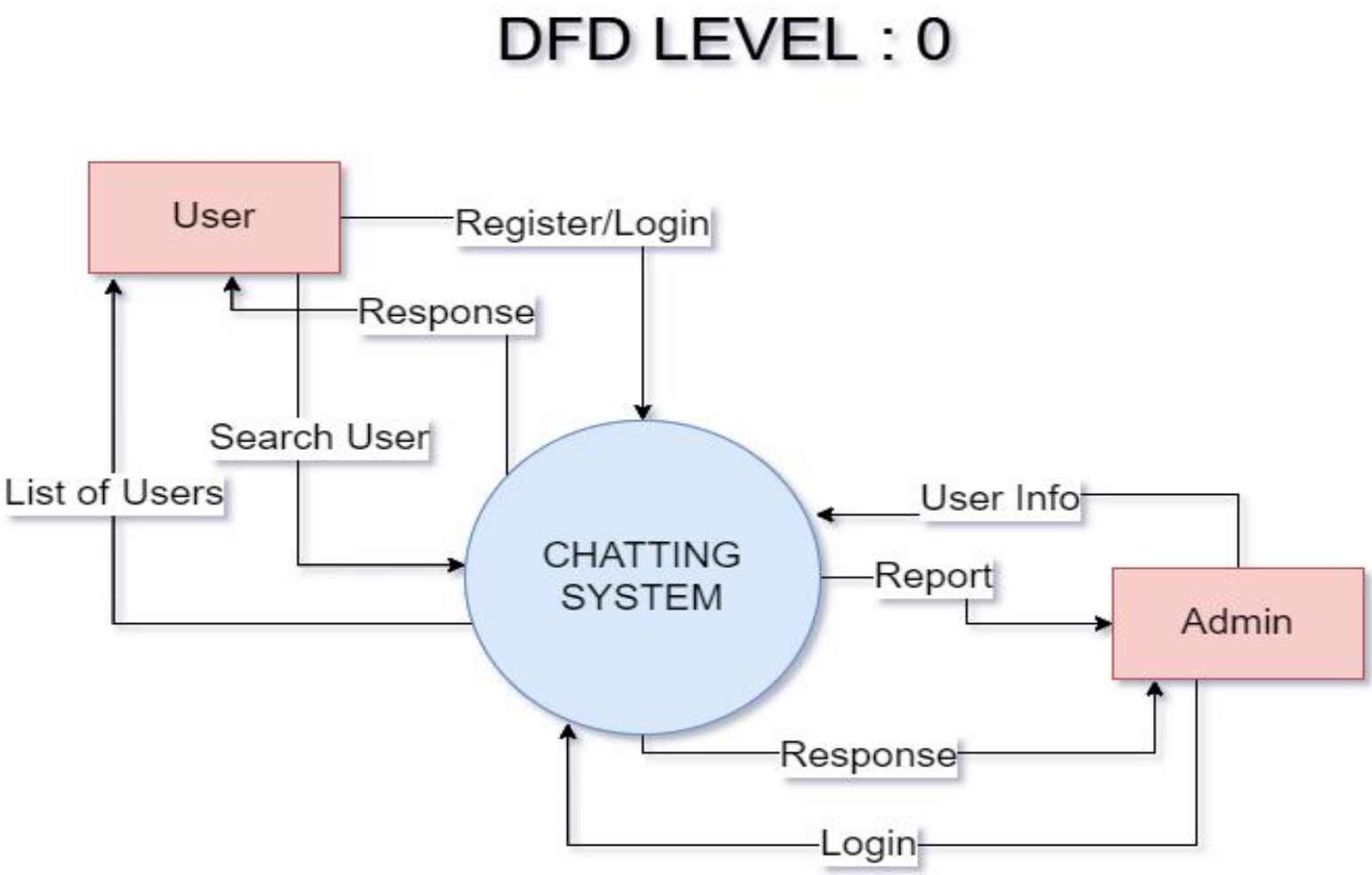
2. Software Requirments:-

- Windows /mac for running vscode and Emulator for running app
- Database:- to save records, firebase is used.

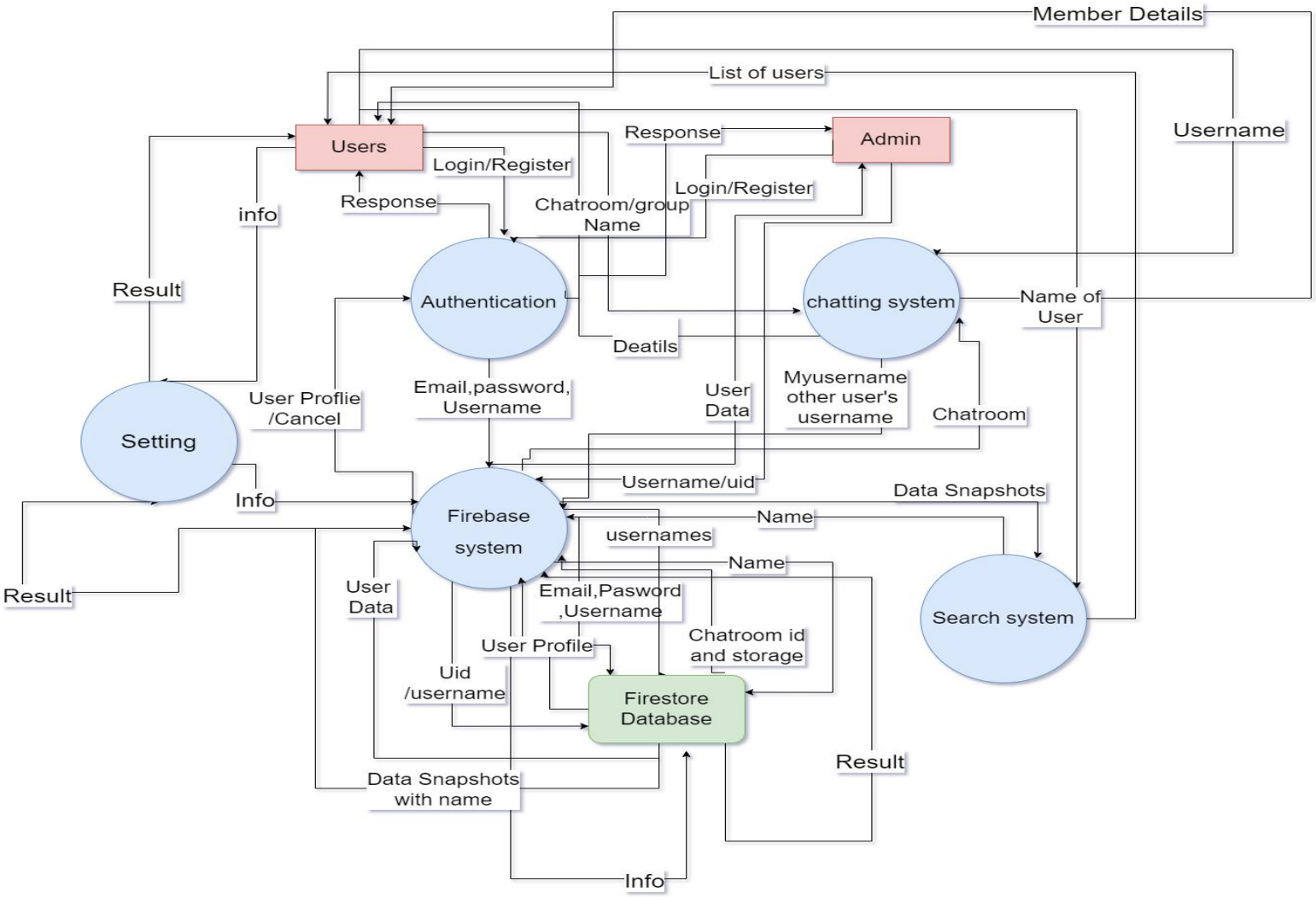
Feasibility Analysis:

Designing this type of Chat Application system is totally feasible from all aspects of economic and technical points

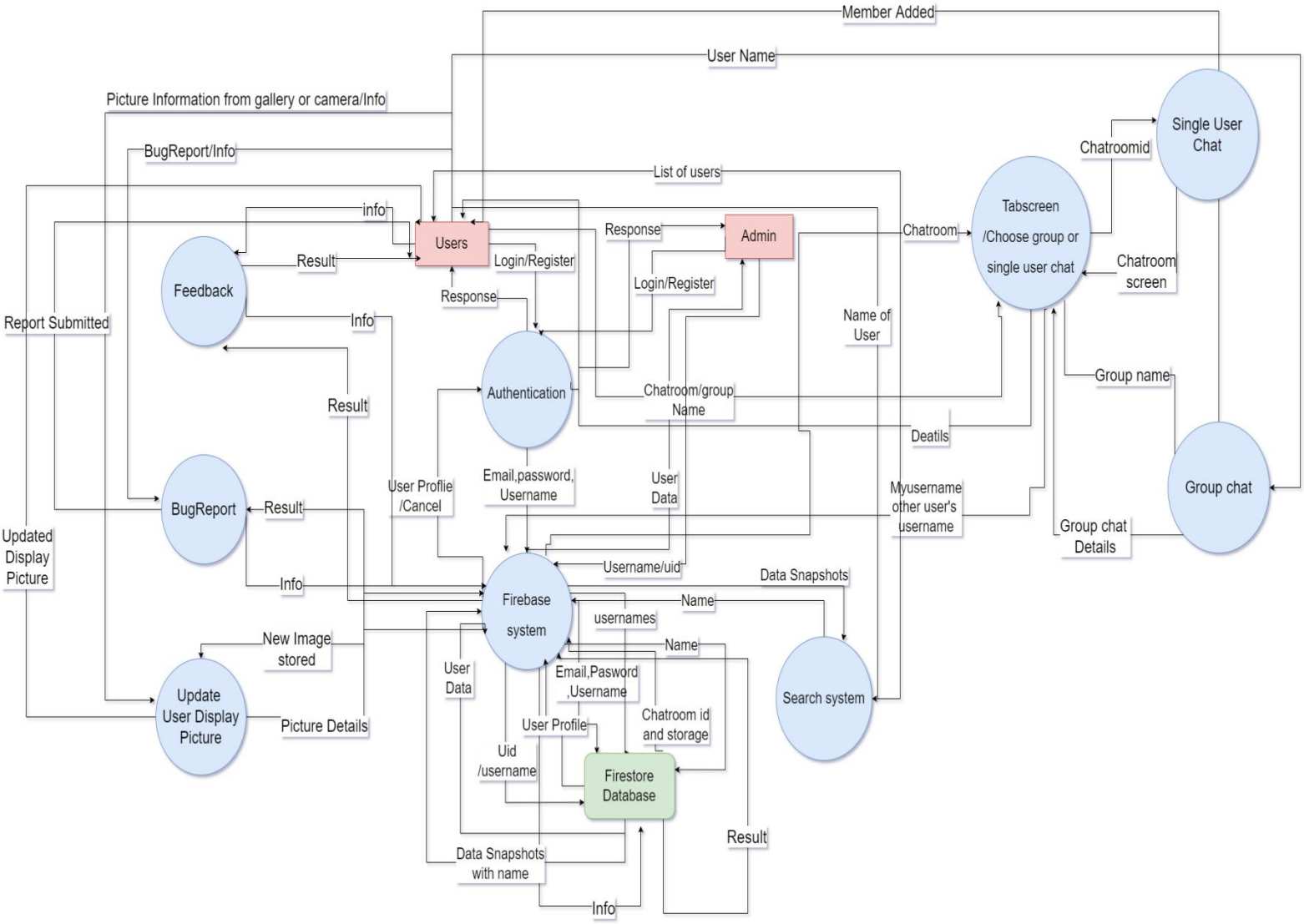
Data flow DIAGRAM:



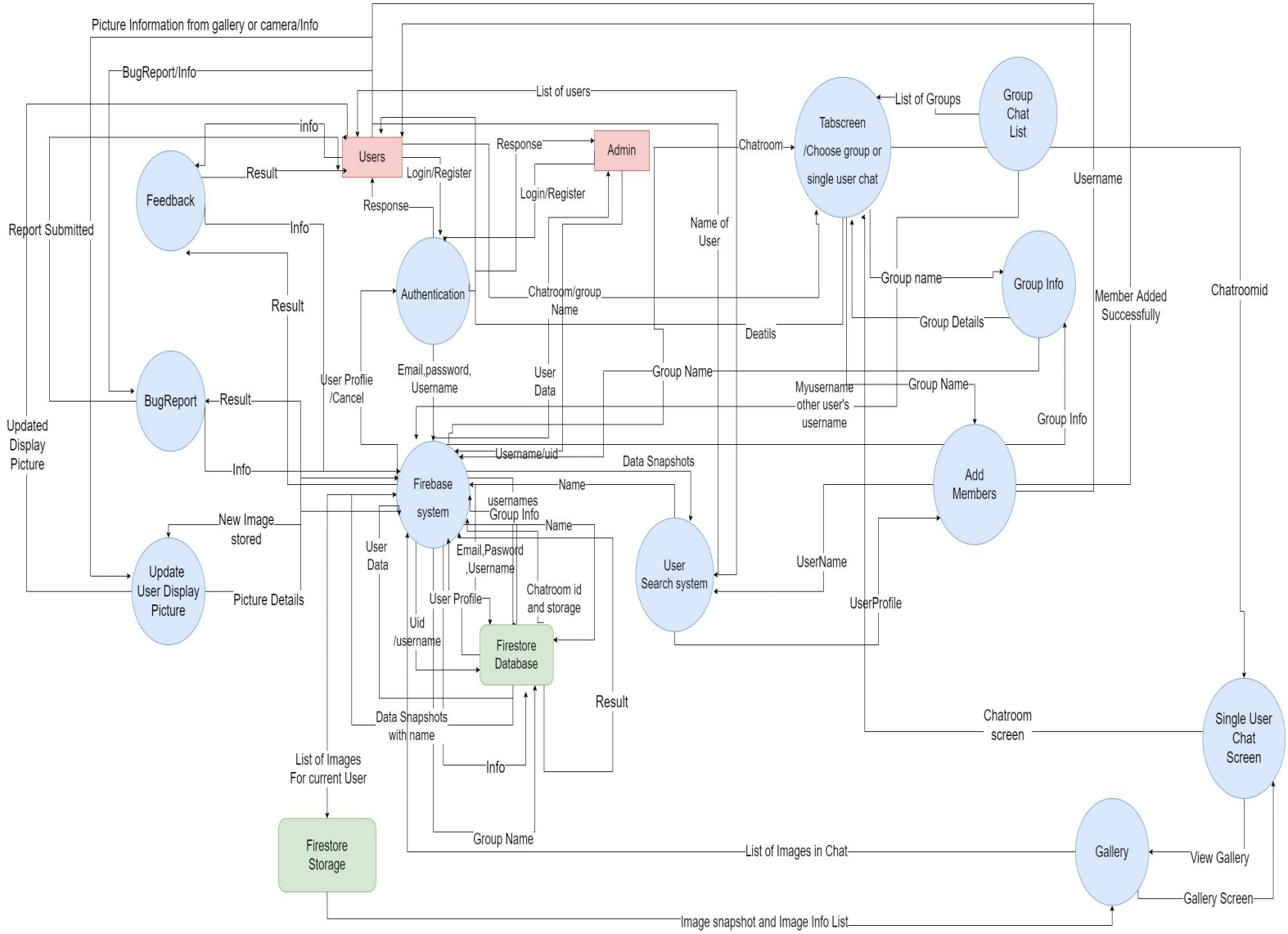
DFD LEVEL: 1


Member Details
List of users
Username
Users
Response
Admin
Login/Register
Login/Register
info
Response
Chatroom/group Name
Result
Authentication
chatting system
Name of User
Deatils
Email,password, Username
User Data
Myusername other user's username
User Proflie /Cancel
Chatroom
Setting
Info
Username/uid
Data Snapshots
Firebase system
Name
usernames
Name
Result
User Data
Email,Pasword ,Username
Search system
User Profile
Chatroom id and storage
Uid /username
Firestore Database
Result
Data Snapshots with name
Info

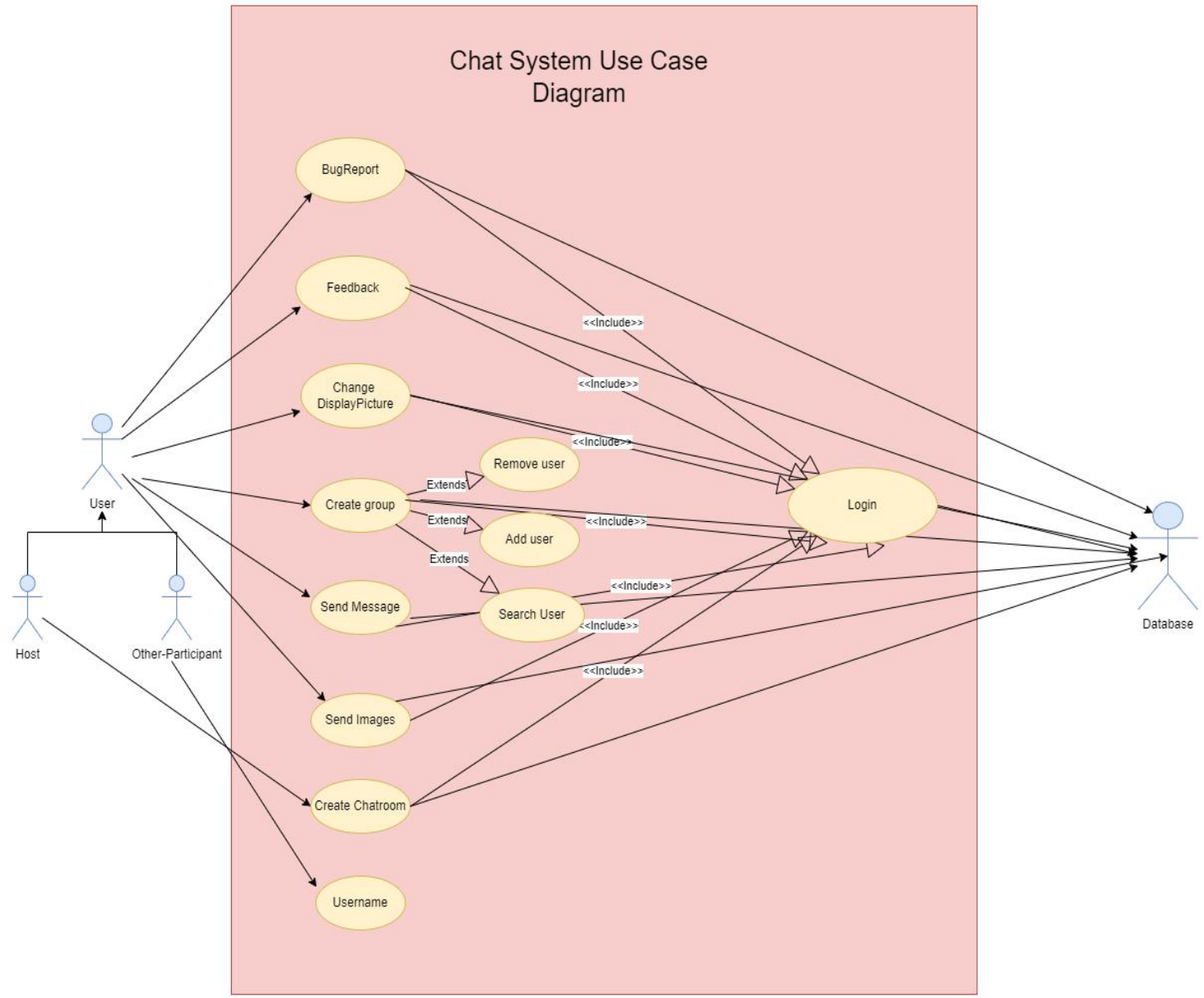
DFD LEVEL: 2


Member Added
User Name
Picture Information from gallery or camera/Info
BugReport/Info
List of users
Single User Chat
Chatroomid
info
Users
Response
Admin
Chatroom
Tabscreen /Choose group or single user chat
Result
Login/Register
Feedback
Login/Register
Chatroom screen
Response
Report Submitted
Info
Name of User
Authentication
Chatroom/group Name
Group name
Result
Deatils
Group chat
User Profile /Cancel
Email,password, Username
User Data
Myusername other user's username
BugReport
Result
Updated Display Picture
Group chat Details
Username/uid
Data Snapshots
Info
Firebase system
Name
usernames
New Image stored
Name
User Data
Email,Pasword ,Username
Search system
Update User Display Picture
Picture Details
User Profile
Chatroom id and storage
Uid /username
Firestore Database
Result
Data Snapshots with name
Info

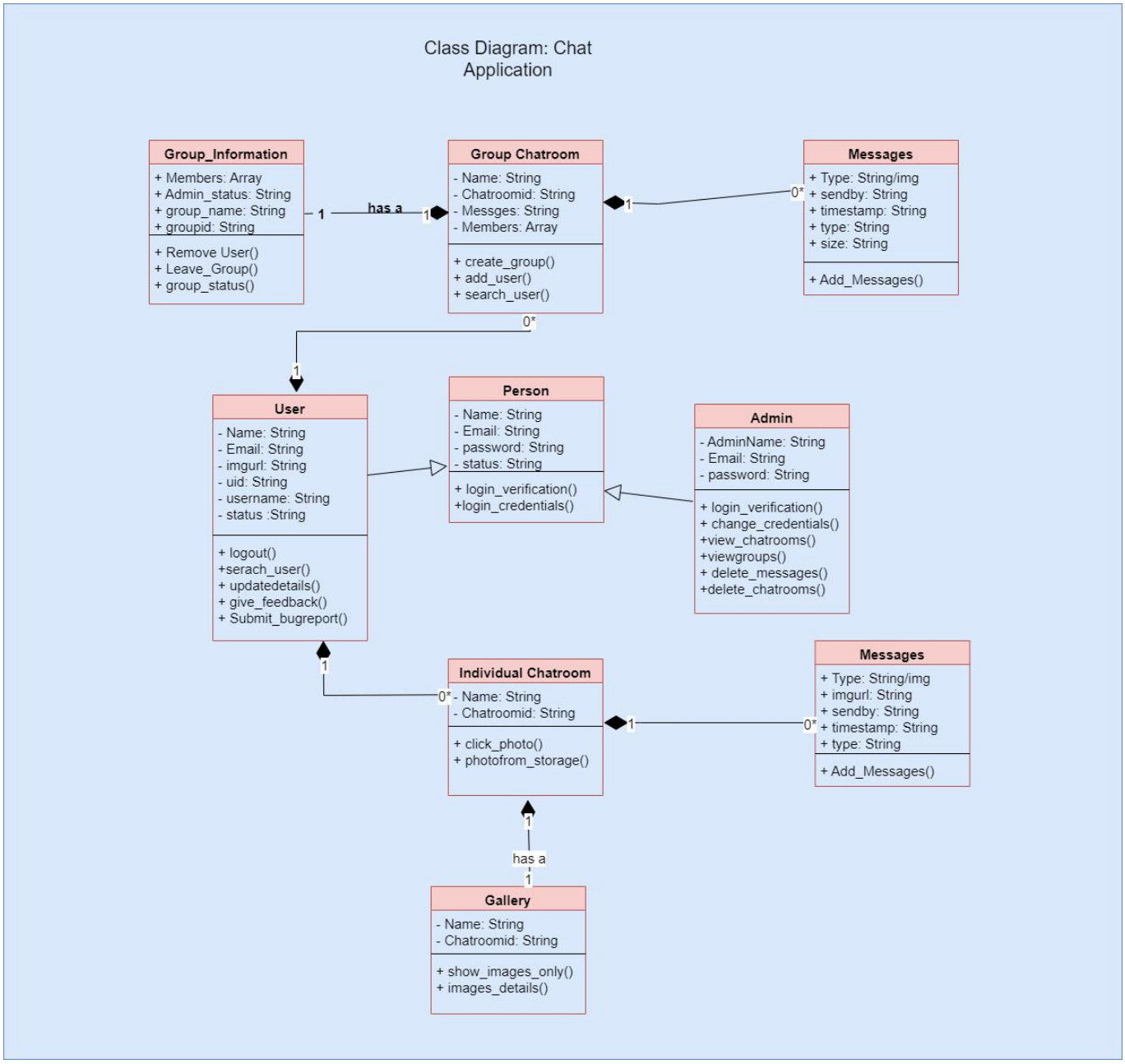
DFD LEVEL: 3


Picture Information from gallery or camera/Info
BugReport/Info
List of users
List of Groups
Group Chat List
info
Users
Response
Admin
Chatroom
Tabscreen /Choose group or single user chat
Feedback
Result
Login/Register
Login/Register
Response
Username
Report Submitted
Info
Name of User
Authentication
Group name
Group Info
Chatroomid
Result
Chatroom/group Name
Member Added Successfully
Group Details
Deatils
BugReport
Result
User Profile /Cancel
Email,password, Username
User Data
Group Name
Myusername other user's username
Group Name
Updated Display Picture
Group Info
Username/uid
Info
Data Snapshots
Add Members
Firebase system
Name
usernames
Group Info
New Image stored
Name
User Data
Email,Pasword ,Username
User Search system
UserName
Update User Display Picture
Picture Details
User Profile
Chatroom id and storage
UserProfile
Uid /username
Firestore Database
Result
Data Snapshots with name
Chatroom screen
Single User Chat Screen
List of Images For current User
Info
Firestore Storage
Group Name
Gallery
List of Images in Chat
View Gallery
Gallery Screen
Image snapshot and Image Info List

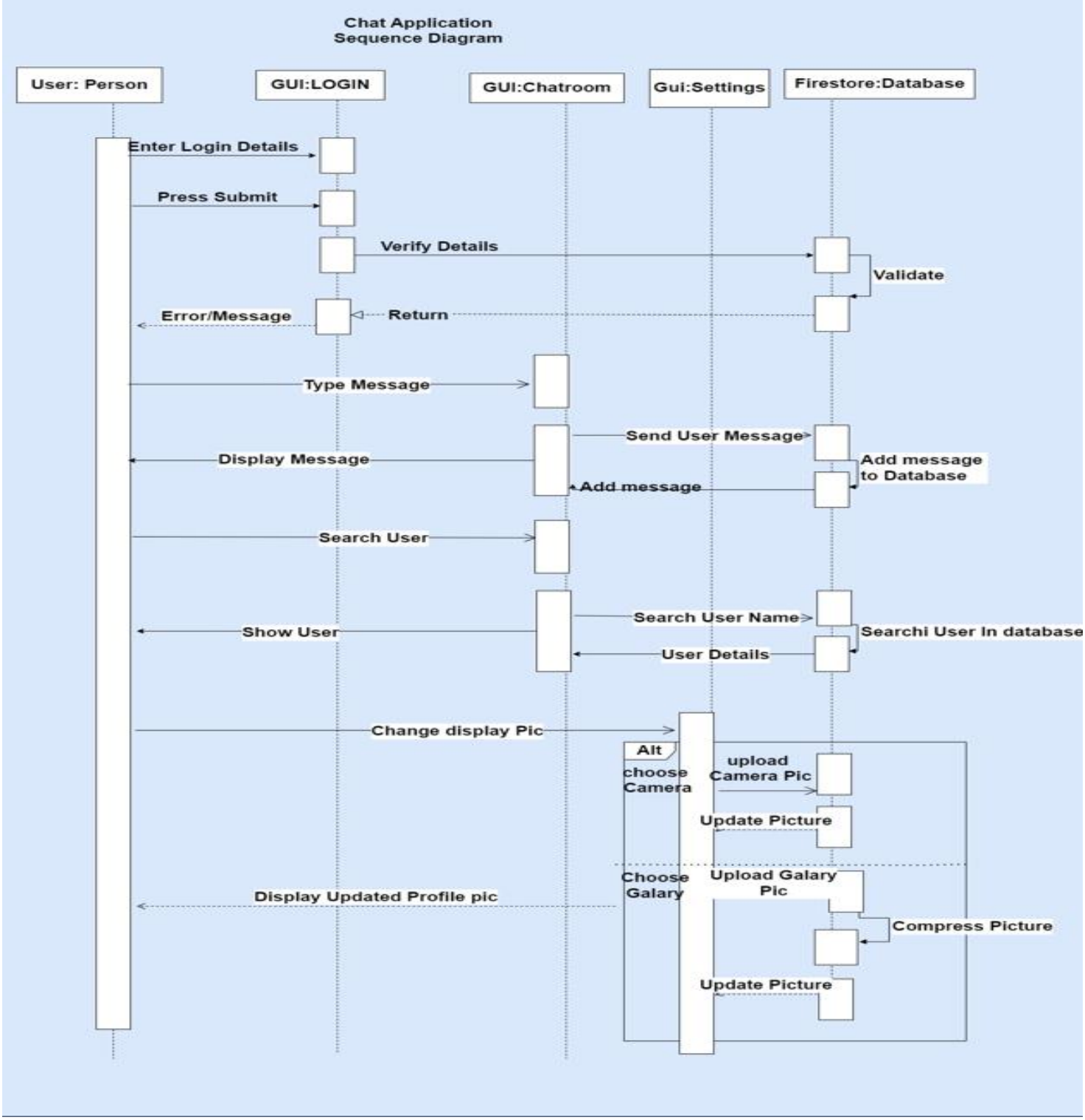
Use Case Diagram:-



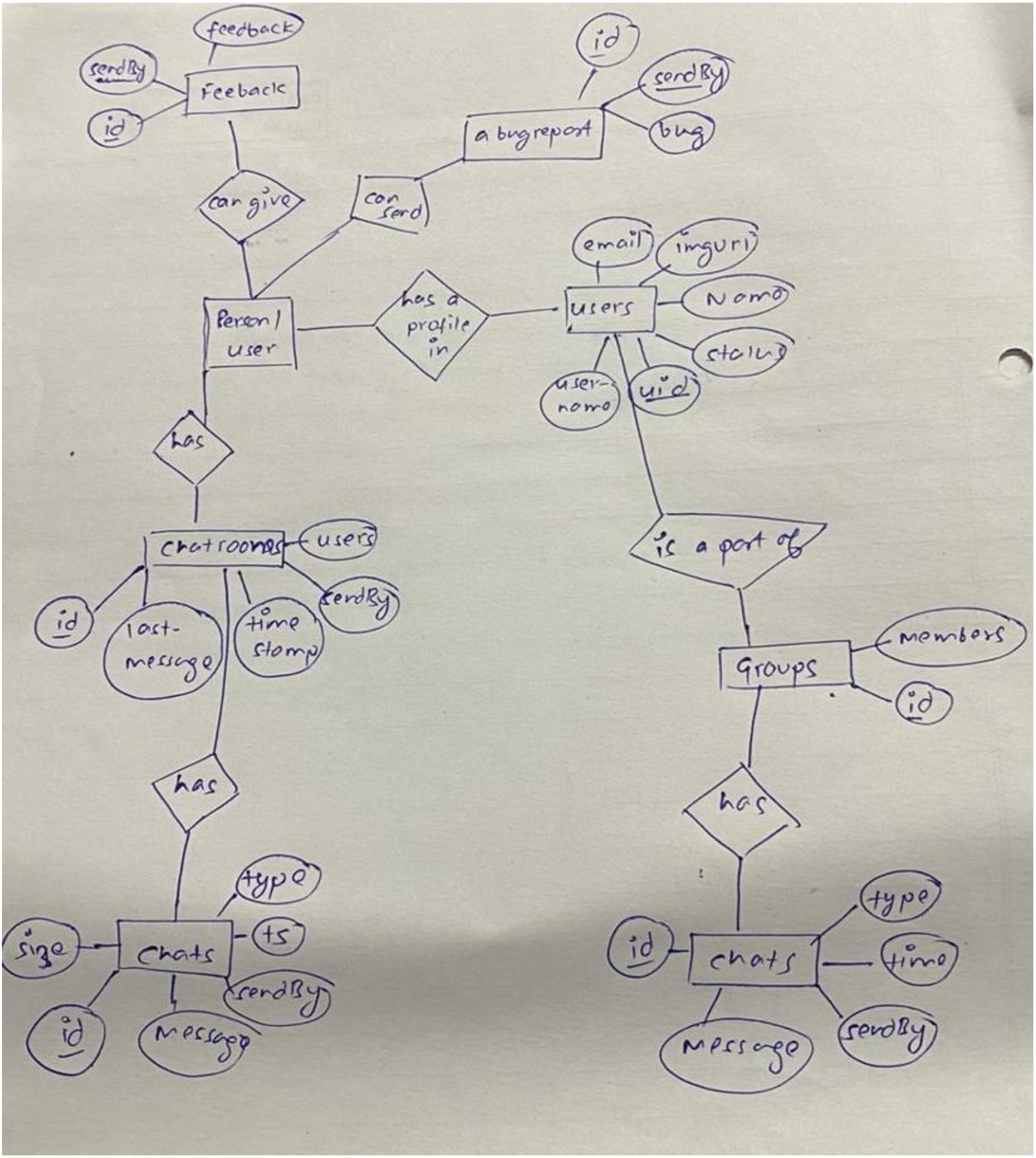
CLASS DIAGRAM:-



Sequence Diagram:-



ER Table:-


feedback
sendBy
Feeback
id
can give
id
sendBy
bug
a bugreport
send
email
imguri
Nama
status
users
has a profile in
Person/ user
user-name
uid
has
chatrooms
users
id
last-message
time stamp
sendBy
is a part of
members
Groups
id
has
has
type
size
chats
ts
sendBy
id
Message
type
id
chats
time
sendBy
Message

DATA DICTIONARY:-

USERS:-

Column Status	Attribute	DataType	Size	Description
NOT_NULL	email	VARCHAR	255	Email address of the user, is unique
NOT_NULL	imgUrl	VARCHAR	255	Display image address for the user
NOT_NULL	name	VARCHAR	255	FULL name of the user
NOT_NULL	status	VARCHAR	7	Status of the user ,can be ONLINE/OFFLINE
NOT_NULL PRIMARY_KEY	uid	VARCHAR	30	User identification number, unique number
NOT_NULL	username	VARCHAR	255	Username associated with the user

FEEDBACK:-

Column Status	Attribute	DataType	Size	Description
NOT_NULL PRIMARY_KEY	Feedback_id	VARCHAR	255	The feedback id ,unique
NOT-NULL	feedback	INT	0-5	Feedback message
NOT-NULL	Sendby	VARCHAR	255	Person ‘s name

BUGS:-

Column Status	Attribute	DataType	Size	Description
NOT_NULL PRIMARY_KEY	bug_id	VARCHAR	255	The bugreport-id ,unique
NOT-NULL	bugreport	varchar	0-5	Bug report sent by user
NOT-NULL	Sendby	VARCHAR	255	Person ‘s name

CHATROOM:-

Column Status	Attribute	DataType	Size	Description
NOT_NULL PRIMARY_KEY	Chatroomid	Varchar	255	Unique id given to every chatroom found on the database,combination of both users username
NOT_NULL	lastmessage	Varchar	255	Lastmessage send in the chatroom
NOT_NULL	lastmessagets	varchar	255	Time stamp of the period when th elast message was sent
NOT_NULL	users	Array	2	Array of elements containing the usernames of the people associated with the chat room
NOT_NULL	chats	Map	NO LIMIT	Maps of the messages and images exchanged between users
NOT_NULL	messagesendby	VARCHAR	255	The name of the person who sent the last message in the chatroom

Algorithms:-

1) SignUp:-

1) Enter Email which includes @ and .com

2) Enter a username

3) Enter a password which has length > 6 characters

4) If(email already in database) => Error

5) If(email not in database)

Then create userdata in database and signin as well.

---

2) SignIN:-

6) Enter Email which includes @ and .com

7) Enter a password which has length > 6 characters

8) If(email already in database) => SignIn

9) If(email not in database)

Then Print error

---

3) Creating and displaying a chatroom:-

10) Enter the username of the other user

11) From the list of user click on the desired user Image

12)If chatroom of the previous user exits with the current user, then enter chatroom

13) If chatroom with the user does not exist then create one in collection("chatrooms")

With the name user1name_user2name where user1name is lexicographical greater than user2name

14) Display the messages and images of the user.

TESTING:-

Unit Testing :-

Code:-

```dart
import 'package:chattingapp/views/signup.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:flutter/material.dart';

void main() {
  bool showIgnIn = true;
  void toggleView() {
    showIgnIn = !showIgnIn;
  }
  test('title', () {
    //setup//run//verigy
  });
  test("empty email returns error string", () {
    var result = EmailFieldValidator.validate('');
    expect(result, "Please provide a valid UserId");
  });
  test("non-empty email returns null", () {
    var result = EmailFieldValidator.validate('sp@sp.com');
    expect(result, null);
  });
  test("empty password returns error string", () {
    var result = PasswordFieldValidator.validate('');
    expect(result, "provide password with 6 charachter");
  });
  test("non-empty password returns error string", () {
    var result = PasswordFieldValidator.validate('asasasas');
    expect(result, null);
  });
  test("empty username returns error string", () {
    var result = UsernameFieldValidator.validate('');
    expect(result, "please provide UserName");
  });
  test("non-empty username returns error string", () {
    var result = UsernameFieldValidator.validate('abcdef');
    expect(result, null);
  });
}
```

Testing of different units of signin module with the help of their validators.

For emailvalidator:-

Testcase 1:-

Input: - “ “,

Expected :- Please provide a valid userid;

Result => passed

Testcase 2:-

Input :- “sp@sp.com”

Expected:- null,

Result => passed

For password validator :-

Input:- “ “,

Expected:- provide password with 6 charachter,

Result:- passed,

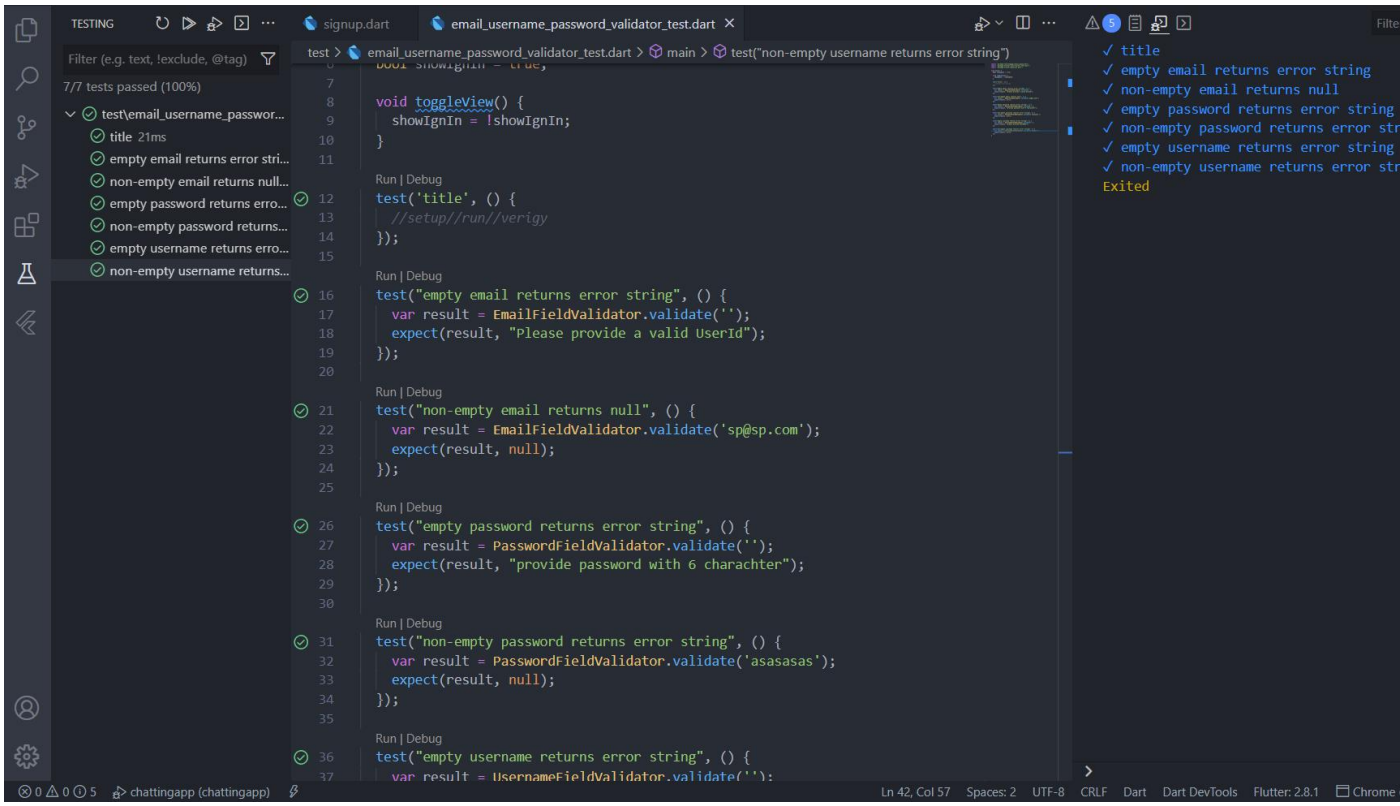
Testcase 2:-

Input :- “assasas”

Expected :- null,

Result:- pass

==========



Integration Testing:-

Testing the Full SignIn module :-

```dart
import 'package:chattingapp/services/auth.dart';
import 'package:chattingapp/services/authprovider.dart';
import 'package:chattingapp/views/signin.dart';
import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';

import 'package:mockito/mockito.dart';
class MockAuth extends Mock implements BaseAuth {}
void main() {
  Widget makeTestableWidget({required Widget child, required BaseAuth auth}) {
    return AuthProvider(
      auth: auth,
      child: MaterialApp(
        home: child,
      ),
    );
  }
  bool showIgnIn = true;
  void toggleView() {
    showIgnIn = !showIgnIn;
  }
  testWidgets('email or password is empty, does not sign in',
      (WidgetTester tester) async {
    MockAuth mockAuth = MockAuth();
    bool didSignIn = false;
    Signin page = Signin(toggleView);
    //LoginPage(onSignedIn: () => didSignIn = true);
    await tester.pumpWidget(makeTestableWidget(child: page, auth: mockAuth));
    await tester.tap(find.byKey(const Key('signIn')));
    verifyNever(mockAuth.signInwithEmailAndPassword('', ''));
    expect(showIgnIn, false);
  });
}
```

Testing the Whole SIgnin Module for widget Testing:-

Testcase 1:-

Input :- “ ”, “ “

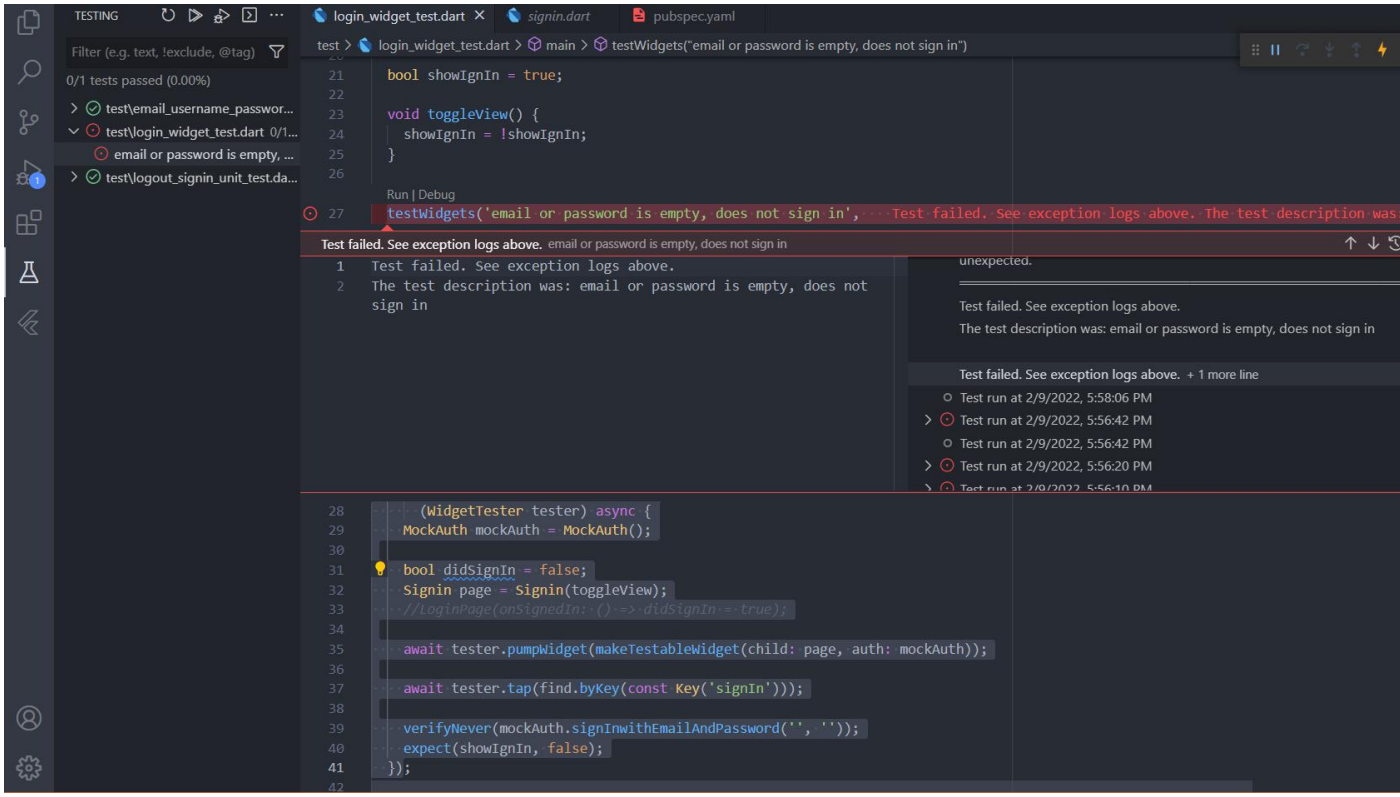
Expected :- showIgnIn = false,

Result:- showIgnIn = false,

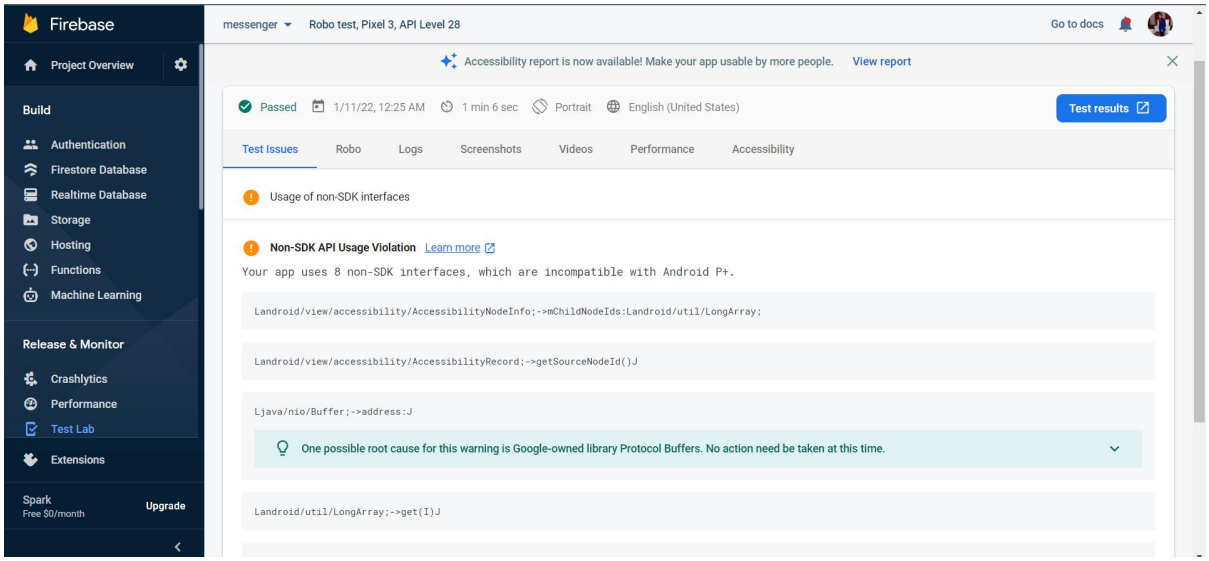
Testcase Failed;

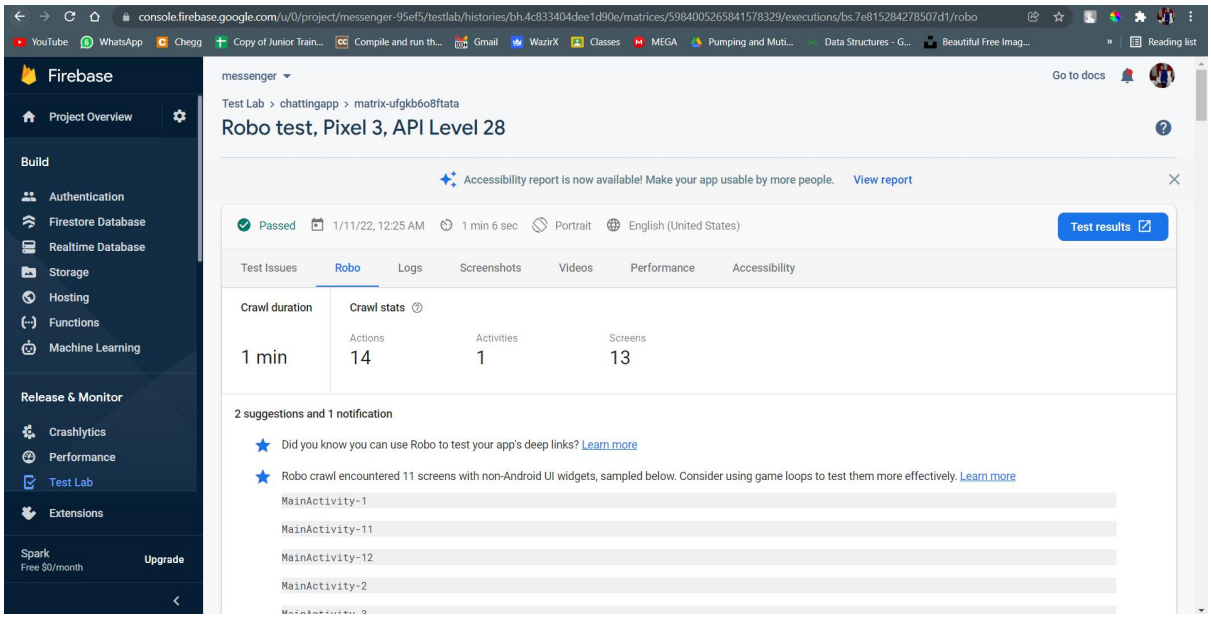
==========

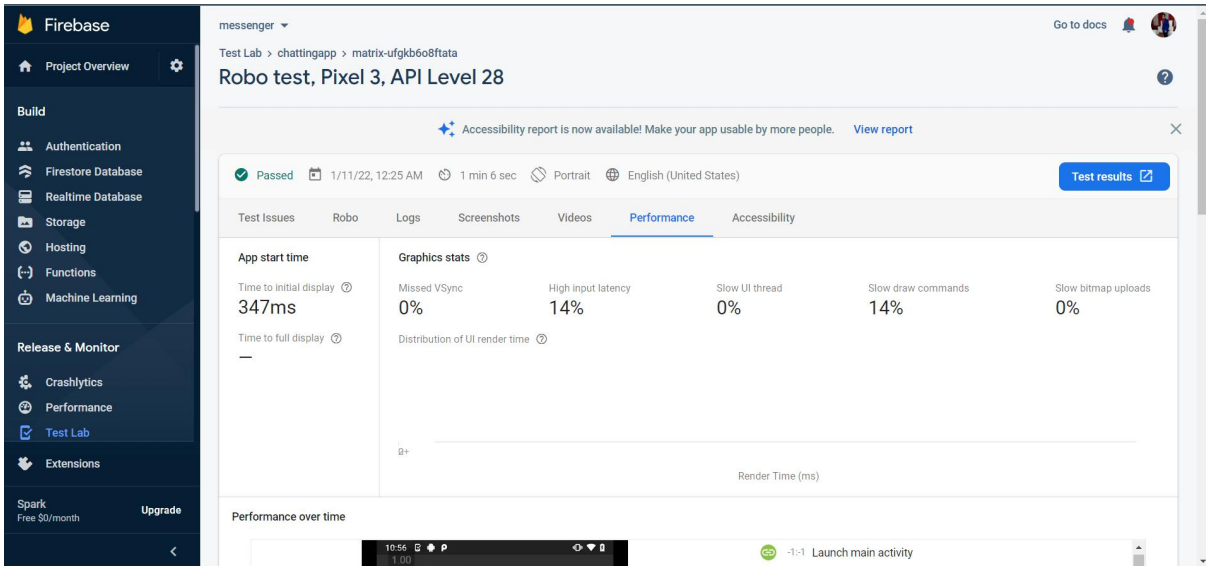
Screenshot:-

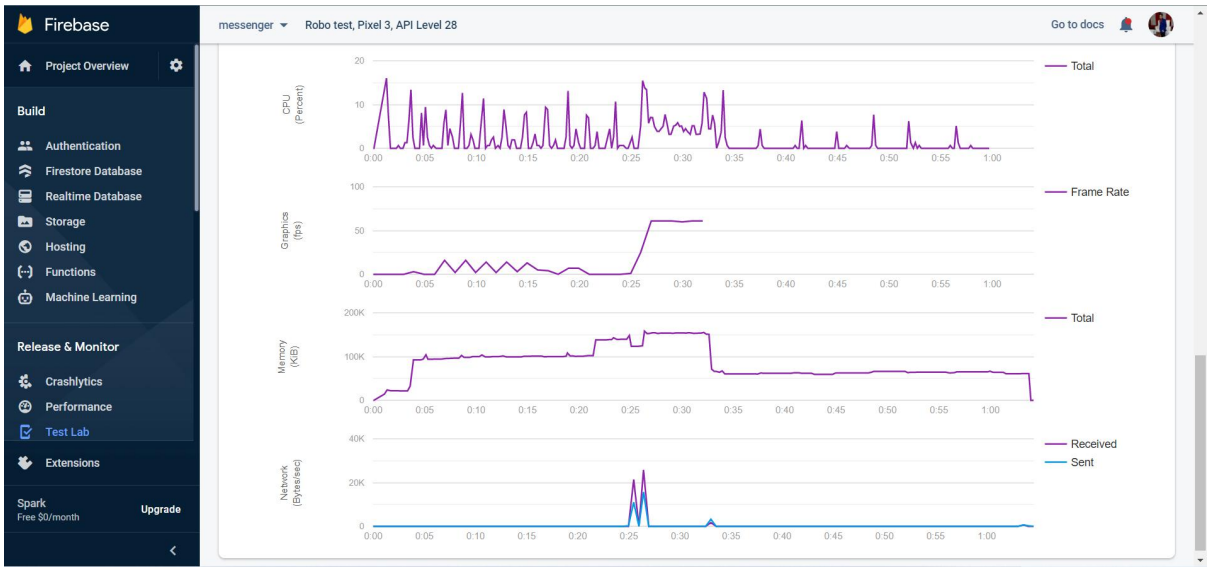

TESTING
Filter (e.g. text, !exclude, @tag)
0/1 tests passed (0.00%)
test\email_username_passwor...
test\login_widget_test.dart 0/1...
email or password is empty, ...
test\logout_signin_unit_test.da...
login_widget_test.dart
signin.dart
pubspec.yaml
test > login_widget_test.dart > main > testWidgets("email or password is empty, does not sign in")
bool showIgnIn = true;
void toggleView() {
showIgnIn = !showIgnIn;
}
Run | Debug
testWidgets('email or password is empty, does not sign in', Test failed. See exception logs above. The test description was
Test failed. See exception logs above. email or password is empty, does not sign in
Test failed. See exception logs above.
The test description was: email or password is empty, does not sign in
Test failed. See exception logs above.
The test description was: email or password is empty, does not sign in
Test failed. See exception logs above. + 1 more line
Test run at 2/9/2022, 5:58:06 PM
Test run at 2/9/2022, 5:56:42 PM
Test run at 2/9/2022, 5:56:42 PM
Test run at 2/9/2022, 5:56:20 PM
(WidgetTester tester) async {
MockAuth mockAuth = MockAuth();
bool didSignIn = false;
Signin page = Signin(toggleView);
//LoginPage(onSignedIn: () => didSignIn = true);
await tester.pumpWidget(makeTestableWidget(child: page, auth: mockAuth));
await tester.tap(find.byKey(const Key('signIn')));
verifyNever(mockAuth.signInwithEmailAndPassword('', ''));
expect(showIgnIn, false);
});

Function Testing :-






Firebase
messenger
Go to docs
Project Overview
Build
Authentication
Firestore Database
Realtime Database
Storage
Hosting
Functions
Machine Learning
Release & Monitor
Crashlytics
Performance
Test Lab
Extensions
Spark
Free $0/month
Upgrade
Test Lab › chattingapp › matrix-ufgkb6o8ftata
Robo test, Pixel 3, API Level 28
Accessibility report is now available! Make your app usable by more people.
View report
Passed
1/11/22, 12:25 AM
1 min 6 sec
Portrait
English (United States)
Test results
Test Issues
Robo
Logs
Screenshots
Videos
Performance
Accessibility
App start time
Time to initial display
347ms
Time to full display
—
Graphics stats
Missed VSync
0%
High input latency
14%
Slow UI thread
0%
Slow draw commands
14%
Slow bitmap uploads
0%
Distribution of UI render time
Render Time (ms)
Performance over time
Launch main activity


Firebase
messenger
Robo test, Pixel 3, API Level 28
Go to docs
Project Overview
Build
Authentication
Firestore Database
Realtime Database
Storage
Hosting
Functions
Machine Learning
Release & Monitor
Crashlytics
Performance
Test Lab
Extensions
Spark
Free $0/month
Upgrade
CPU (Percent)
Total
Graphics (fps)
Frame Rate
Memory (KiB)
Total
Network (Bytes/sec)
Received
Sent