

Pigeon Chat – Chatting Application

A PROJECT REPORT

Submitted By

Krishna

University Roll No- 2100290140078

Akanksha Singh

University Roll No- 2100290140011

Prerna Sharma

University Roll No- 2100290140106

**Submitted in partial fulfillment of the
Requirements for the Degree of**

MASTER OF COMPUTER APPLICATIONS

**Under the Supervision of
Mr. ANKIT VERMA
(ASSISTANT PROFESSOR)**



Submitted to

**DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206
(June 2023)**

CERTIFICATE

Certified that **Krishna (University Roll No. 2100290140078)** has carried out the project work for “**Pigeon Chat-Chatting Application**” for Master of Computer Applications from Dr. A.P.J. Abdul Kalam Technical University (AKTU), Technical University, Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself or herself. The contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other university or institution.

Date:

Krishna (University Roll No. 2100290140078)
Akanksha Singh (University Roll No.2100290140011)
Prerna Sharma (University Roll No- 2100290140106)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

Mr. Ankit Verma (Assistant Professor)
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Signature of Internal Examiner

Signature of External Examiner

Dr. Arun Kumar Tripathi
Head, Department of Computer Applications
KIET Group of Institutions, Ghaziabad

ABSTRACT

The purpose of Online Chat Application is to automate the existing manual system by the help of computerized equipment's and full-fledged computer software, fulfilling their requirements, so that their valuable data/information can be stored for a longer period with easy accessing and manipulation of the same. The required software and hardware are easily available and easy to work with

Thus, it will help organization in better utilization of resources. The organization can maintain computerized records without redundant entries. That means that one need not be distracted by information that is not relevant, while being able to reach the information.

Flutter Chat is an innovative online chatting application developed using Flutter, a popular cross-platform framework for building mobile applications.

The primary objective of Flutter Chat is to provide users with a seamless and engaging chatting experience across various platforms, including IOS and Android devices. Leveraging Flutter's robust UI toolkit, Flutter Chat offers a visually appealing and consistent user interface across different devices, ensuring a delightful user experience.

ACKNOWLEDGEMENT

Success in life is never attained alone. My deepest gratitude goes to my project supervisor, **Mr. Ankit Verma** for his guidance, help and encouragement throughout my research work. Their enlightening ideas , comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Arun Kumar Tripathi, Professor and Head, Department of Computer Applications**, for his insightful comments and administrative help on various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical situations.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Krishna (2100290140078)

Akanksha Singh (2100290140011)

Prerna Singh (2100290140106)

List of Chapters

Certificate	i
Abstract	ii
Acknowledgements	iii
Table of Content	iv
List of Abbreviations	vi
List of Figures	vii - viii
Chapter 1 – Introduction	01 - 10
1.1 Project description	01 - 02
1.2 Literature Review	02 - 06
1.3 Hardware / Software used in Project	07
1.4 Functional Requirements	09
1.5 Non- Functional Requirements	10
Chapter 2 Feasibility Study	11 - 12
2.1 Technical feasibility	10
2.2 Operational Feasibility	10
2.3 Behavioral Feasibility	11
2.4 Operational Feasibility	11
Chapter 3 Database Design	13 - 25
3.1 Waterfall Model	12
3.2 Requirement Gathering & Analysis	13
3.3 ER Diagram	14 - 16
3.4 Use Case Diagram	17
3.5 Activity Diagram	18
3.6 Sequential Diagram	19 - 20
3.7 Collaboration Diagram	21
3.8 State Chart Diagram	22
3.9 Component Diagram	23

3.10 Deployment Diagram	25
Chapter 4 Form Design	26 - 31
4.1 Screenshot	26 - 30
Chapter 5 Coding	31 - 79
5.1 Module wise code	31 - 79
Conclusion	80 - 82
Bibliography	82
References	83

CHAPTER 1

INTRODUCTION

1.1 PROJECT DESCRIPTION

This project has been developed to overcome the problems prevailing in the practicing manual system. This application is supported to eliminate and, in some cases, reduce the hardships faced by the existing system. Moreover, this application is designed for moving toward secure digitalization. It will enhance transparency in the work.

No formal knowledge is needed for the user to use this system. Thus, by this all it proves it is user Pigeon. So, this **“Pigeon Chat- Chatting Application”** can lead to an error-free, secure, reliable and fast system.

It provides a lightweight, interactive interface so that it can be used easily on all types of devices.

Customer can choose more than one item to make a Chatting and can view Chatting details before logging off. The Chatting confirmation is sent to the customer. The Chatting is placed in the queue and updated in the database and returned in real time. This system assists the staff to go through the Chatting's in real time and process it efficiently with minimal errors.

Online Chatting Application, as described above, can lead to error free, secure, reliable and fast management system. It can assist the user in concentrating on their other activities rather than concentrate on record keeping. Thus, it will help organization in better utilization of resources. The organization can maintain computerized records without redundant entries.

1.1 LITERATURE REVIEW

1.1.1 Introduction

Chat apps have revolutionized the way people communicate with each other, allowing users to send messages, make voice or video calls, and share files and media with friends, family, and colleagues. As such, chat apps have become an essential part of modern life, providing a flexible and convenient platform for communication that can be accessed from anywhere in the world. This literature review explores the existing research on chat apps, focusing on key areas such as communication theories, user experience, privacy and security, cultural factors, and the impact on social relationships.

1.1.2 Communication Theories

Communication theories provide a framework for understanding how communication works, and can help us to better understand the dynamics of chat app interactions. One widely-cited theory is Social Presence Theory, which suggests that communication technologies vary in their ability to convey social presence, or the sense of being present with others in a given environment.

According to this theory, chat apps can be more effective at conveying social presence than other forms of communication such as email, due to their real-time nature and the ability to see when other users are typing or actively engaging with the conversation. Another theory that has been applied to chat apps is Media Richness Theory, which posits that different communication channels can be used for different types of messages.

1.1.3 Privacy and Security

Privacy and security are important considerations for chat app developers and users alike. Chat apps often involve the transmission and storage of sensitive information, such as personal messages, photos, and videos, which can make users vulnerable to security threats such as hacking and data breaches. Research has identified a number of privacy and security concerns related to chat apps, including the potential for user data to be accessed by third parties, the risk of malware and phishing attacks, and the potential for surveillance by governments or other entities.

One study by Xu and Teo (2017) found that users' perceptions of privacy and security were influenced by factors such as the reputation of the app, the transparency of the app's data collection and storage practices, and the availability of security features such as two-factor authentication and end-to-end encryption. Another study by Xu and Teo (2017) examined user perceptions of privacy and security in the context of workplace chat apps.

They found that users were generally more concerned about privacy and security in the workplace than in their personal lives, and that concerns about privacy and security could influence their decision to use a particular app.

1.1.4 Privacy and Security Concerns

One of the biggest concerns with chat apps is privacy and security. As users share personal information, messages, and media on these platforms, they may be vulnerable to data breaches, hacking, and other forms of cybercrime.

Research has shown that privacy and security concerns are prevalent among chat app users. A study by Xu and Teo (2017) found that many users were concerned about the privacy implications of sharing personal information on chat apps, and were also worried about the risk of being hacked or having their accounts compromised. Similarly, a study by Naylor (2017) found that users were concerned about the potential for chat app data to be intercepted and used for malicious purposes.

To address these concerns, chat app developers have implemented a range of security measures, such as encryption, two-factor authentication, and user verification. However, research suggests that many users are still not aware of these security features or do not use them properly. A study by Zhang et al. (2017) found that many users were not aware of the security features available on their chat apps, and were not taking steps to protect their privacy and security.

1.1.5 User Experience

User experience encompasses a variety of factors, including usability, interface design, and features such as group chats and video calls. One study by He et al. (2016) found that users' perceptions of chat app quality were largely influenced by the perceived usefulness of the app, as well as the quality of the interface and the responsiveness of the app. Another study by Kim et al. (2017) examined user preferences for different chat app features. They found that users preferred apps with features such as group chats, voice and video calls, and the ability to share files and media. Users also expressed a preference for apps that were easy to use and had a simple and intuitive interface.

The user experience and design of chat apps can also have a significant impact on their use and effectiveness. Research has shown that users place a high value on features such as ease of use, speed, and customization options.

A study by Lin et al. (2016) found that users preferred chat apps that were easy to use and customize, and that allowed them to quickly access the features they needed. Similarly, a study by Zhang et al. (2017) found that users were more likely to use chat apps that were designed with a clean, intuitive interface and that offered a range of customization options.

Other important design factors for chat apps include accessibility and compatibility with different devices and platforms. A study by Kang et al. (2017) found that users appreciated chat apps that were accessible on multiple devices and that could be easily synced across different platforms.

1.1.6 Cultural Factors

Chat apps are used around the world and across cultures, which means that cultural factors can influence how users interact with them. For example, some cultures may prioritize politeness and formality in messaging.

Others may be more casual and use abbreviations or emojis. A study by Chen et al. (2016) found that cultural differences can influence the use of chat app features such as voice and video calls. The study found that users in Asian cultures tended to use these features less frequently than users in Western cultures, possibly due to cultural differences in communication styles.

Another cultural factor that can influence chat app use is language. Chat apps that support multiple languages can facilitate communication across linguistic barriers and help to foster cross-cultural connections. However, the use of machine translation or other language processing technologies can also introduce errors or misinterpretations, which can lead to misunderstandings or even offensive messages. A study by Kang et al. (2017) found that users in multilingual chat groups often had to navigate language barriers and develop strategies to ensure clear communication.

1.1.7 Impact on Social Relationships

Chat apps have the potential to both strengthen and weaken social relationships. On the one hand, they can facilitate communication and help to maintain connections with friends and family members who are geographically distant. On the other hand, excessive use of chat apps can lead to social isolation, addiction, and other negative effects.

Research has found that the use of chat apps can have both positive and negative effects on social relationships. A study by Wang et al. (2018) found that frequent use of chat apps was associated with greater social connectedness and satisfaction with social relationships. However, another study by Lin et al. (2016) found that excessive use of chat apps.

1.2.1 Software Used in Project

- Operating System – Windows 11.
- Code Editor – Microsoft Visual Studio Code
- Flutter: for Designing UI
- Dart: for Programming Language
- Firebase Firestore: for Live Database

1.2.2 Hardware Used in Project

- Processor – Ryzen 5 3rd Gen 3500H
- RAM – 16 GB
- Graphic Card – GTX 1650 (4 GB)

1.3 FUNCTIONAL REQUIREMENTS

1.3.1 Chatting System

- Create an Account
- Log into the Website
- Navigate the Restaurant menu.
- Select an item from the menu.
- Customized options for selected item
- Add item to current Chatting.
- Remove items/ remove all items from current Chatting.
- Provide delivery and payment details.

1.3.2 Menu Management System

- Add a new/update/delete vendor to/from the menu.
- Add a new/update/delete Chatting category to/from the menu.
- Add a new/update/delete Chatting item to/from the menu.
- Add a new/update/delete option for a given Chatting item.
- Update price for a given Chatting item.
- Update additional information for a given Chatting item.

1.3.3 Chatting Retrieval System

- Retrieve new Chatting's from the database.
- Display the Chatting's in an easily readable, graphical way.
- Mark a Chatting as having been processed and remove it from the list of active Chatting's.

1.4 NON-FUNCTIONAL REQUIREMENTS

1.4.1 Portability

System running on one platform can easily be converted to run on another platform.

1.4.2 Reliability

The ability of the system to behave consistently in a user-acceptable manner when operating within the environment for which the system was intended.

1.4.3 Availability

The system should be always available, meaning the user can access it using a web browser, only restricted by the down time of the server on which the system runs.

1.4.4 Maintainability

A commercial database is used for maintaining the database and the application server takes care of the site.

1.4.5 Security

The chat app should have robust security measures in place, such as encryption of messages, secure authentication, and protection against hacking and other cyber threats.

CHAPTER 2

FEASIBILITY STUDY

After studying and analyzing all the existing and required functionalities of the system, the next task is to do the feasibility study for the project. The feasibility study includes consideration of all the possible ways to provide a solution to a given problem. The proposed solution should satisfy all the user requirements and should be flexible enough so that future changes can be easily made based on the future upcoming requirements.

2.1 Technical Feasibility

This included the study of function, performance and constraints that may affect the ability to achieve an acceptable system. For this feasibility study, we studied complete functionalities to be provided in the system, as described in the System Requirement Specification (SRS) and checked if everything was possible using different types of front end and backend platform.

2.2 Operational Feasibility

Operational feasibility study is an assessment of whether a proposed project is feasible from an operational standpoint. It evaluates the practicality of implementing the project based on factors such as technology, personnel, infrastructure, and processes. This study aims to identify potential operational issues that may arise during the project's implementation and determine whether the project can be realistically executed within the available resources and constraints. The operational feasibility study is an essential step in project planning, as it helps ensure that the project will be successful and sustainable over the long term.

2.3 Behavioral Feasibility

A behavioral feasibility study is an examination of the potential impacts that a project or initiative may have on human behavior. It assesses whether the project aligns with societal norms and values, and whether it is likely to be embraced by the community it is intended to serve. This type of study explores how the target audience may react to the project and what factors could influence their behavior. It also considers the feasibility of implementing the project, considering the attitudes, beliefs, and cultural values of those who will be affected. The goal of a behavioral feasibility study is to ensure that a project is not only financially feasible, but also socially acceptable and culturally appropriate.

2.4 Economic Feasibility

For the economic feasibility, Economic analysis or cost/benefits analysis is most frequently used technique the effectiveness of a proposed system. It is a procedure to determine the benefits and saving those that are expected from the proposes system and compare them with cost. If the benefits outweigh the costs, a decision is taken to design and implement the system. Other wise, further justification or alternative in proposed system will have to be made if it is to have a chance of being approved this is ongoing effort that improves in accuracy at each phase of a system life cycle.

CHAPTER 3

3.1 Waterfall Model

The waterfall model is a well-known structured methodology for software development. The whole process of system development is divided into distinct phases. The model has been introduced in 1970s. Every phase has a unique output.

It was the first SDLC model to be used widely. So that, sometimes it is referred to Waterfall by SDLC. The waterfall model is used when the system requirements are well known, technology is understood, and the system is a new version of an existing product (Dennis, Wixom and Roth, 2012).

Mainly there are six phases in Waterfall model. If there is a problem faced in any phase of the cycle, the system goes to the previous phase. The phases of Waterfall method is:

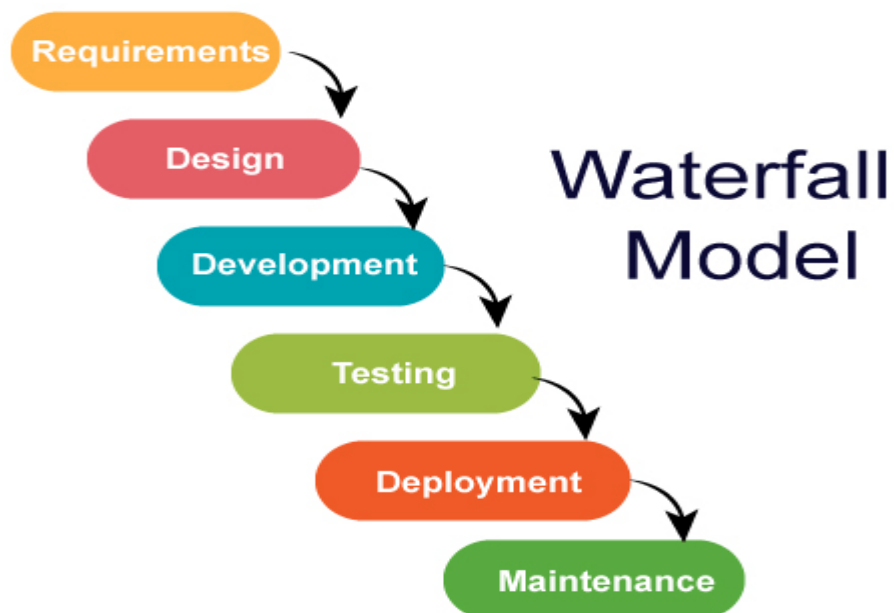


Fig 3.1 Waterfall Model

3.1 REQUIREMENTS GATHERING & ANALYSIS

In this Phase, all possible requirements of the system are captured and documented in a requirement specification doc.

3.1.1 System Design:

The requirements documented in previous phase are studied in this phase and the system design is prepared.

3.1.2 Implementation:

With inputs from system design, the system is developed in several units. Then the units are tested.

3.1.3 Integration & Testing:

The units of the program developed in previous phase are integrated into a system. Then the whole system is tested.

3.1.4 Deployment of the system:

When all kind of testing is done, the product is deployed in the customer environment.

3.1.5 Maintenance:

There are some issues which are found in the client environment. Patches are released to fix those issues.

3.2 ER DIAGRAM

Entities are represented by the rectangle shape. The entity will be our database table of Online Chatting System later.

Attribute is represented by the oval shape. This will be the columns or fields of each table in the Online Chatting System.

Relationship is represented by diamond shape. This will determine the relationships among entities. This is usually in a form of primary key to foreign key connection.

We will follow the 3 basic rules in creating the ER Diagram.

1. Identify all the entities.
2. Identify the relationship between entities and
3. Add meaningful attributes to our entities.

Step 1. Entities

- User
- Product
- Chatting
- Payment
- Admin

Step 2. After we have specified our entities, it is time now to connect or establish a relationship among the entities.

- The user has Chatting.
- Chatting have some products.
- Chatting has some payment.

Step 3. The last part of the ERD process is to add attributes to our entities.

User Entity has following attributes.

- Email
- Password
- FName
- Id

Group Entity has following attributes.

- GID
- GName
- Size
- Image

Chatting Entity has following attributes.

- CName
- Email
- OID
- Size
- Quantity

A Payment Entity has following attributes.

- PID
 - Email
 - Bank
 - Card
 - CCV
-

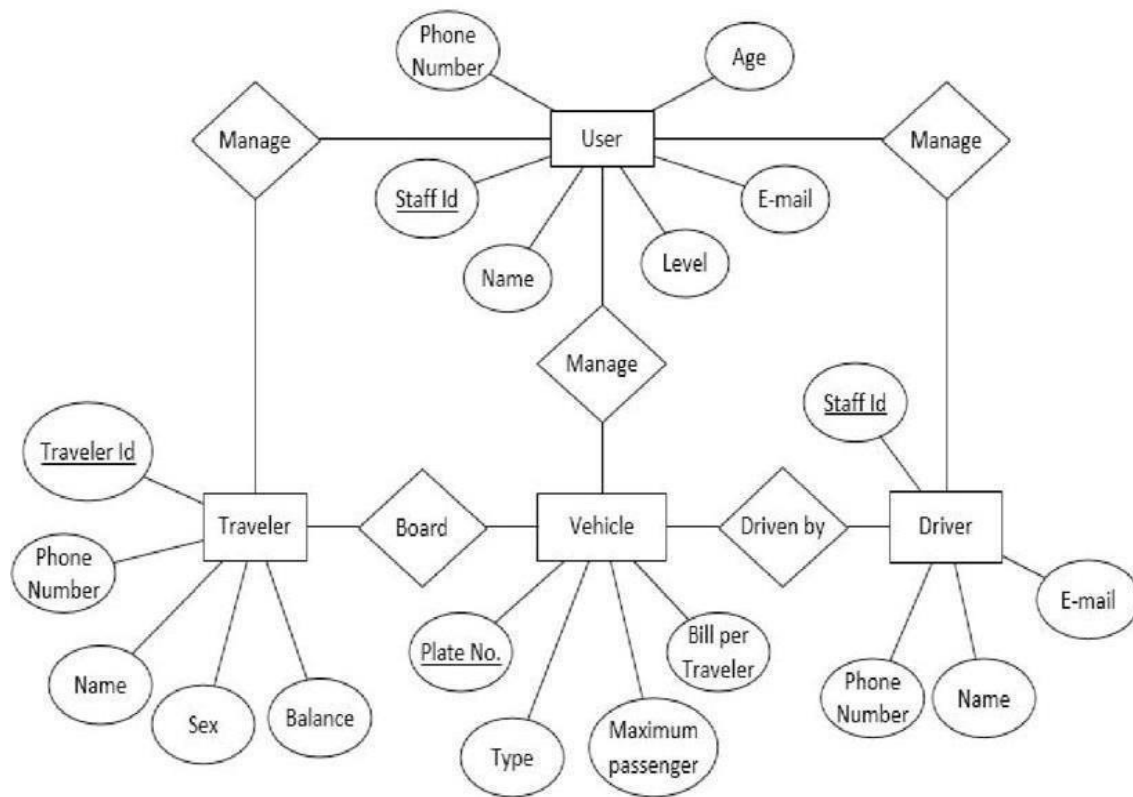


Fig. 3.2 ER Diagram for Firebase Backend

3.3 USE CASE DIAGRAM

Use-case diagrams model the behavior of a system and help to capture the requirements of the system. Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors.

A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and tells how the user handles a system.

Purposes of a use case diagram given below:

1. It gathers the system's needs.
2. It depicts the external view of the system.
3. It recognizes the internal as well as external factors that influence the system.
4. It represents the interaction between the actors.

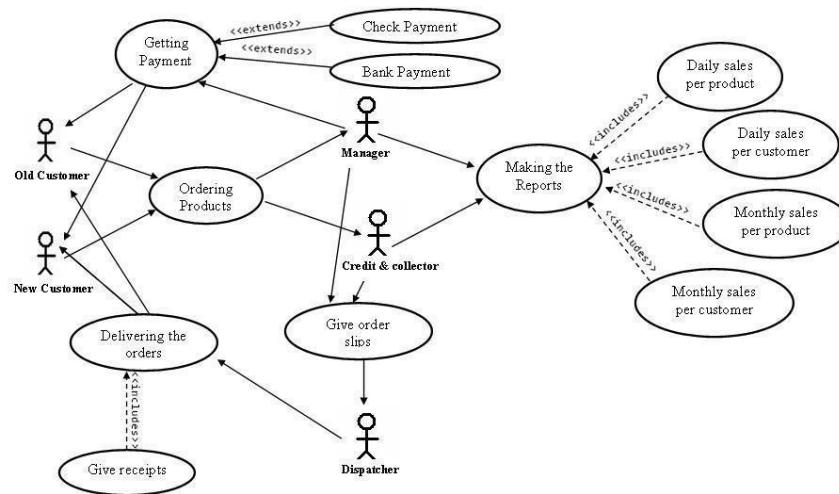


Fig 3.3.USE CASE DIAGRAM for Firebase backend

3.4 ACTIVITY DIAGRAM

An Activity Diagram is a behavioral diagram. It depicts the behaviour of a system. Its primary use is to depict the dynamic aspects of a system. The dynamic aspect of a system specifies how the system operates to attain its function.

It is basically a flowchart to represent the flow from one activity to another activity. Activity Diagrams are not exactly flowcharts as they have some additional capabilities including branching, parallel flow, etc.

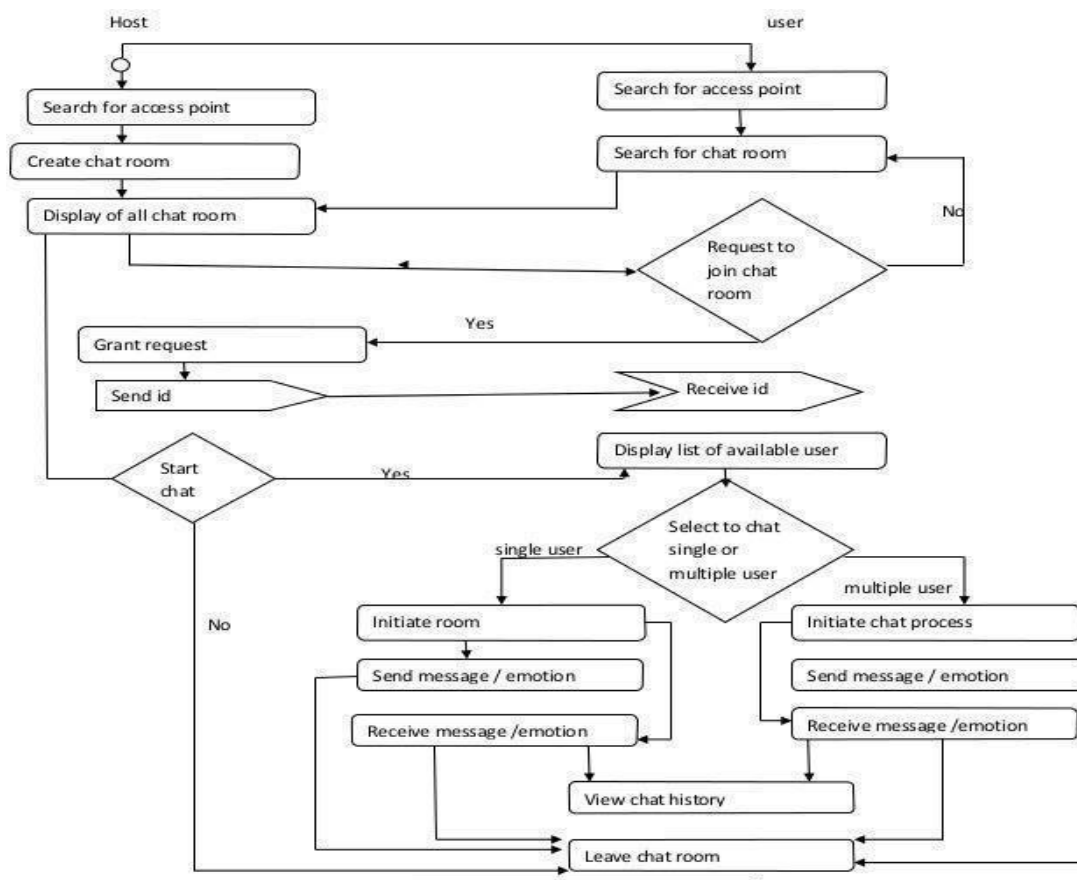


Fig 3.4 Activity Diagram for Firebase backend

3.5 SEQUENCE DIAGRAM

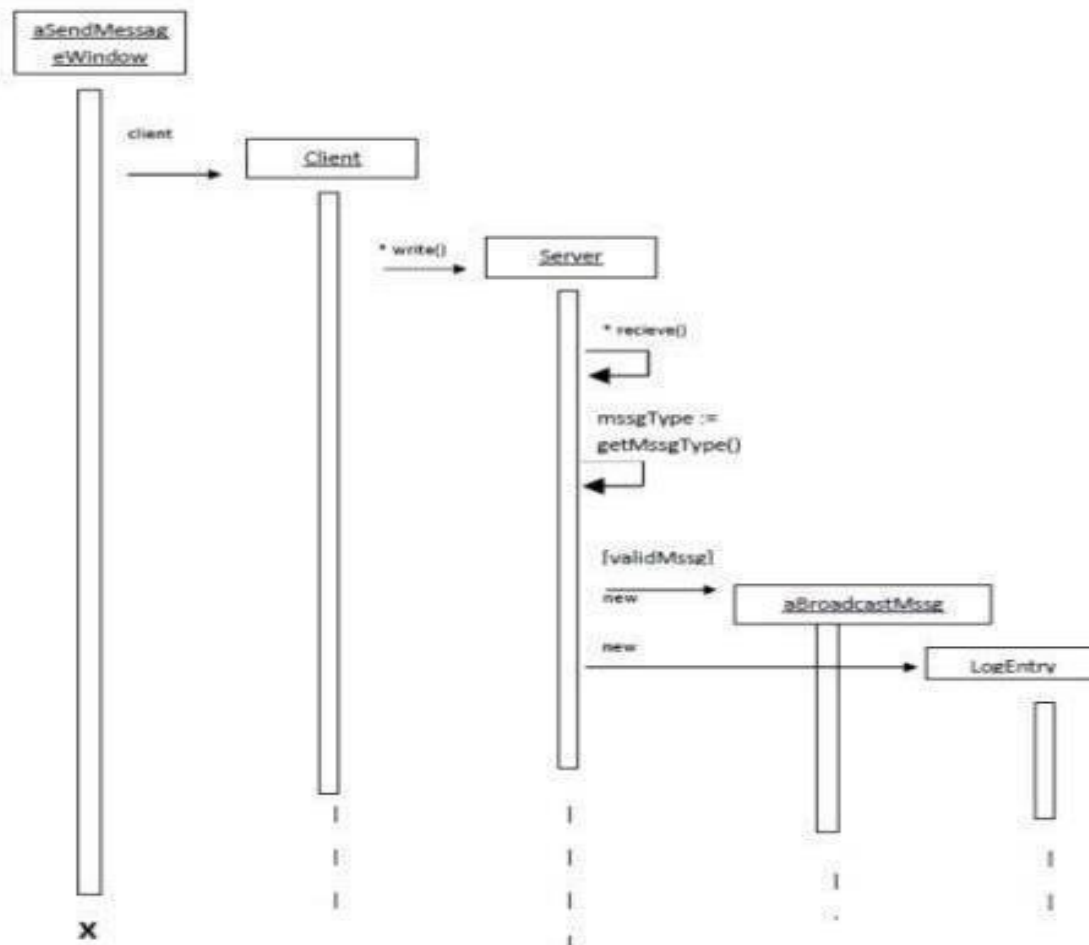


Fig 3.5 Sequence Diagram for Firebase backend

The design shows the detailed illustration of events sequenced and happens in Chatting System. This designed sequence diagram is able to show programmers and readers the sequence of messages between the actor and the objects.

As you can see through the illustration, the conditions and interactions are emphasized. These interactions are essential for the Online Chatting in System development. The series of messages are shown and labeled to guide you in building the System. You can modify the design if you have more ideas. You can also add more features to this design and use it as your project blueprint.

3.6 COLLABORATION DIAGRAM

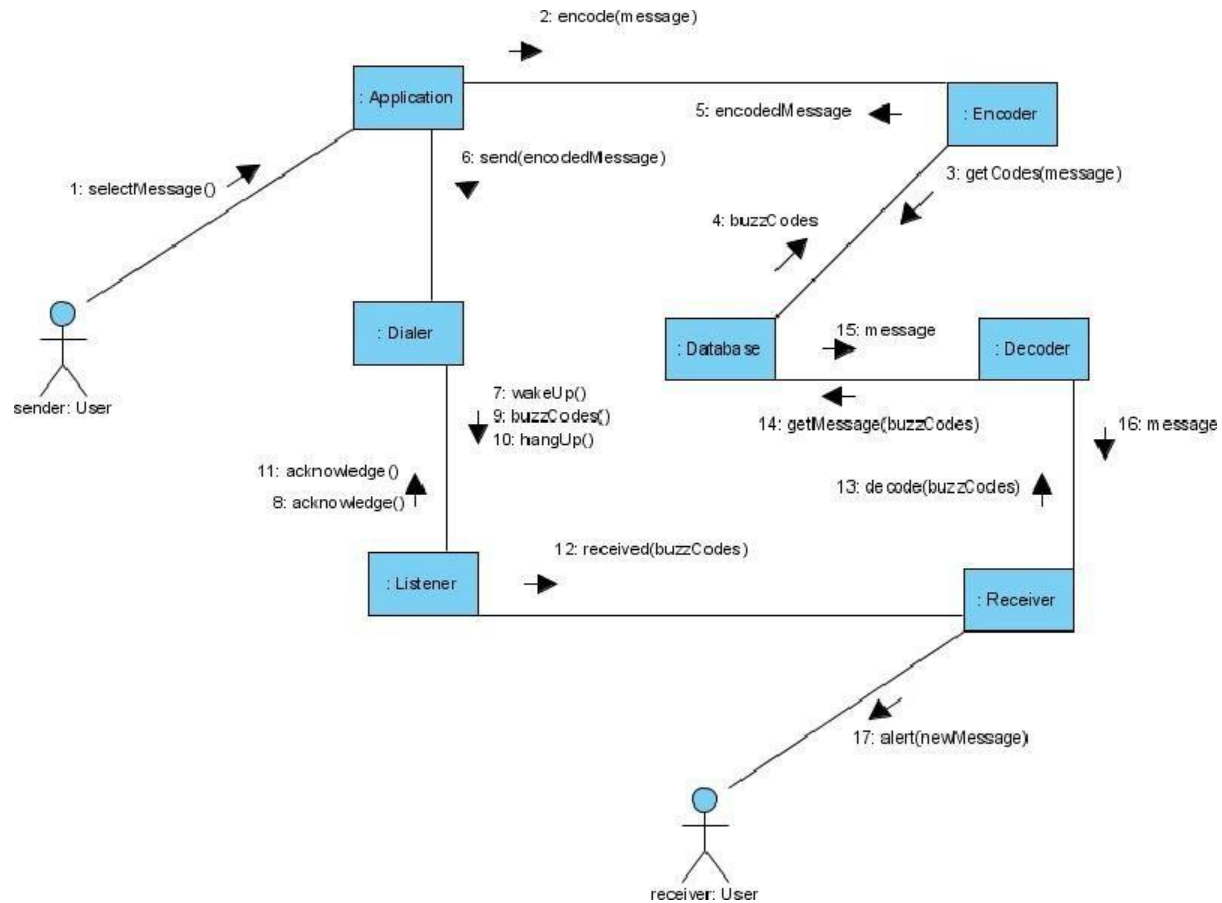


Fig 3.6 Collaboration Diagram for Firebase backend

The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming. An object consists of several features. Multiple objects present in the system are connected to each other. The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system.

3.7 STATE CHART DIAGRAM

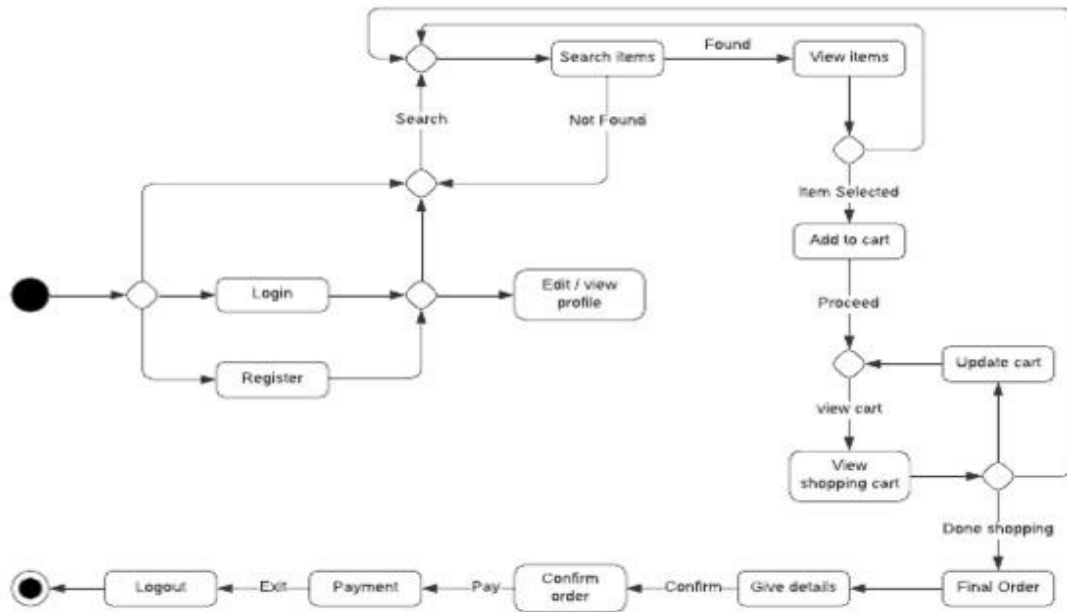


Fig 3.7. State Chart for Firebase backend

The name of the diagram itself clarifies the purpose of the diagram and other details. It describes different states of a component in a system. The states are specific to a component/object of a system.

A State chart diagram describes a state machine. State machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events.

3.8 COMPONENT DIAGRAM

A component diagram is used to break down a large object-oriented system into the smaller components, to make them more manageable. It models the physical view of a system such as executable, files, libraries, etc. that resides within the node.

It visualizes the relationships as well as the organization between the components present in the system. It helps in forming an executable system. A component is a single unit of the system, which is replaceable and executable. The implementation details of a component are hidden, and it necessitates an interface to execute a function. It is like a black box whose behaviour is explained by the provided and required interfaces.

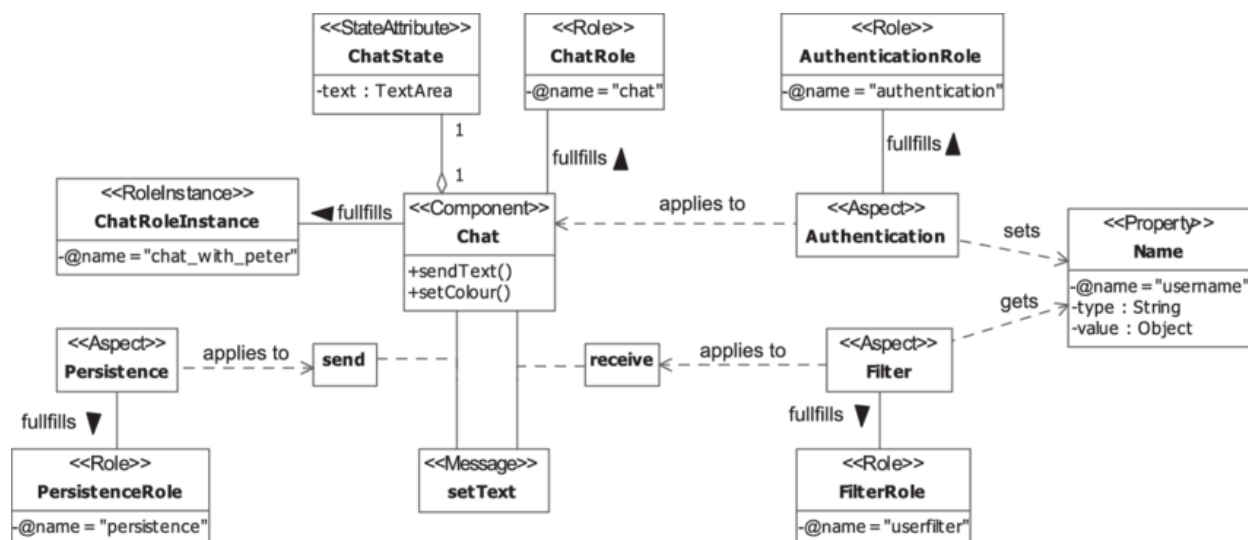


Fig 3.8. USE CASE DIAGRAM for Firebase backend

3.9 DEPLOYMENT DIAGRAM

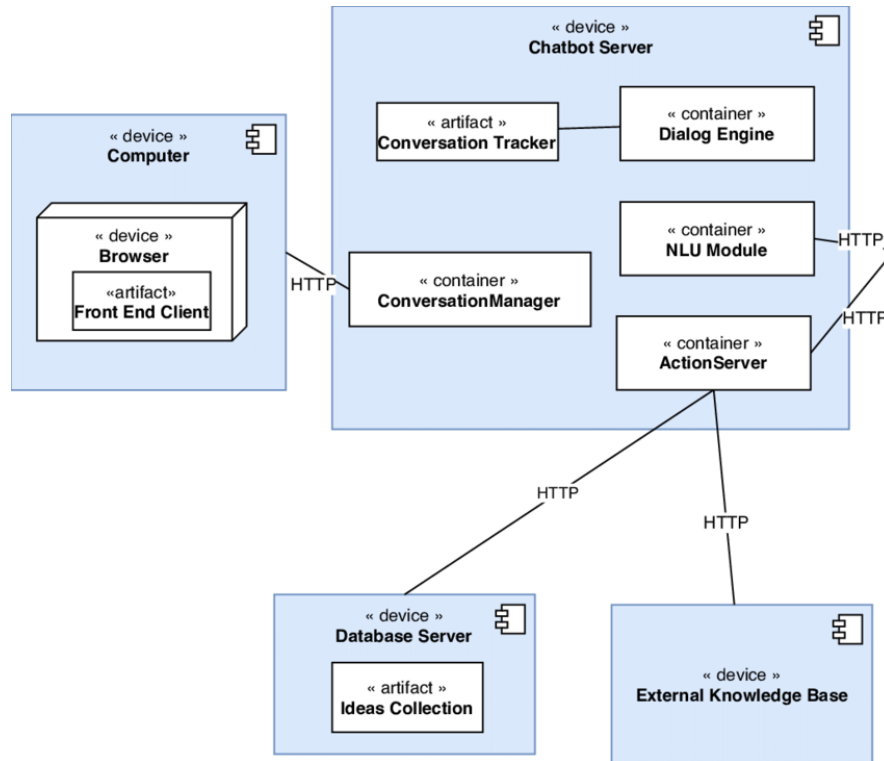


Fig 3.9. Deployment Diagram for Firebase backend

Deployment diagrams are used to visualize the topology of the physical components of a system, where the software components are deployed.

Deployment diagrams are used to describe the static deployment view of a system.

Deployment diagrams consist of nodes and their relationships.

The term Deployment itself describes the purpose of the diagram.

CHAPTER 4

4.1 Screenshot

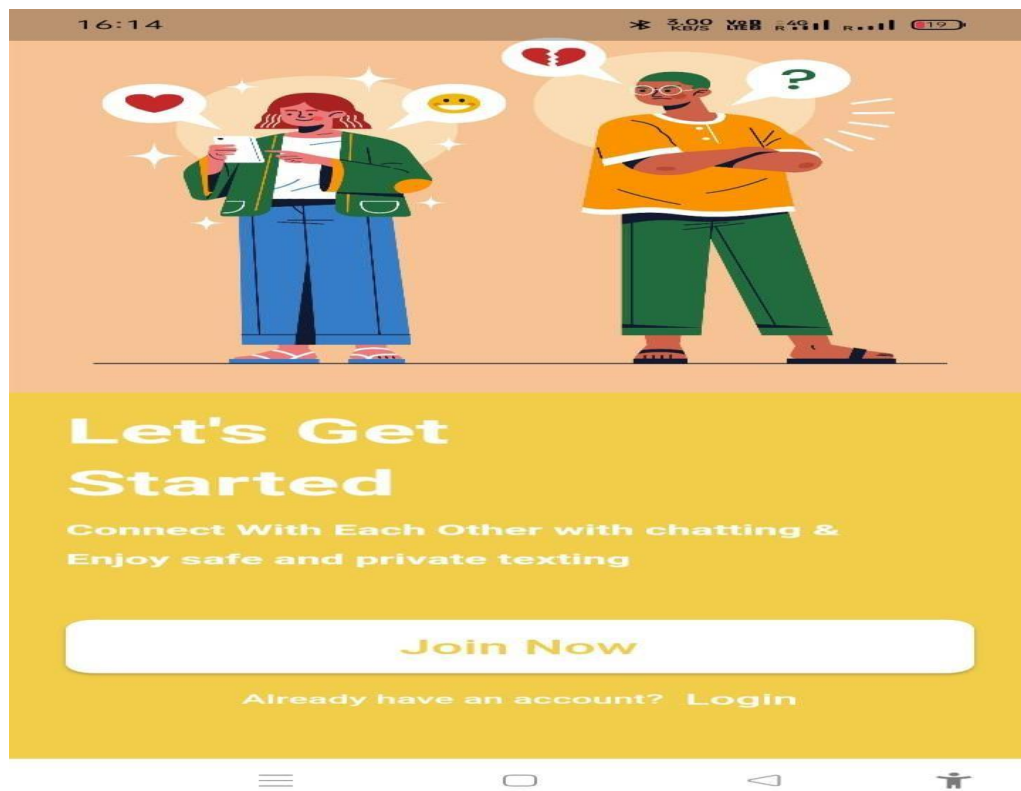





Figure 4.1.1: Login page / Signup page of Application

16:14  26.0 KB/s 4G  

Create Account

Connect with your friends today!


Email Address

Enter your email

Name

Enter your name

Password

Enter your password 

Sign Up

Already have an account? [Login](#)




Fig 4.1.2: Create Account page


16:14 0 9.40 KB/s 4G R 100% 100%

Hi, Welcome Back! 🖐️

Hello again, you've been missed!

Email Address


Password

[Forgot Password](#)

Login

Or Login With



Don't have an account? [Sign Up](#)

☰ □ ◀ ▶ 👤

Fig 4.1.3. Login Page

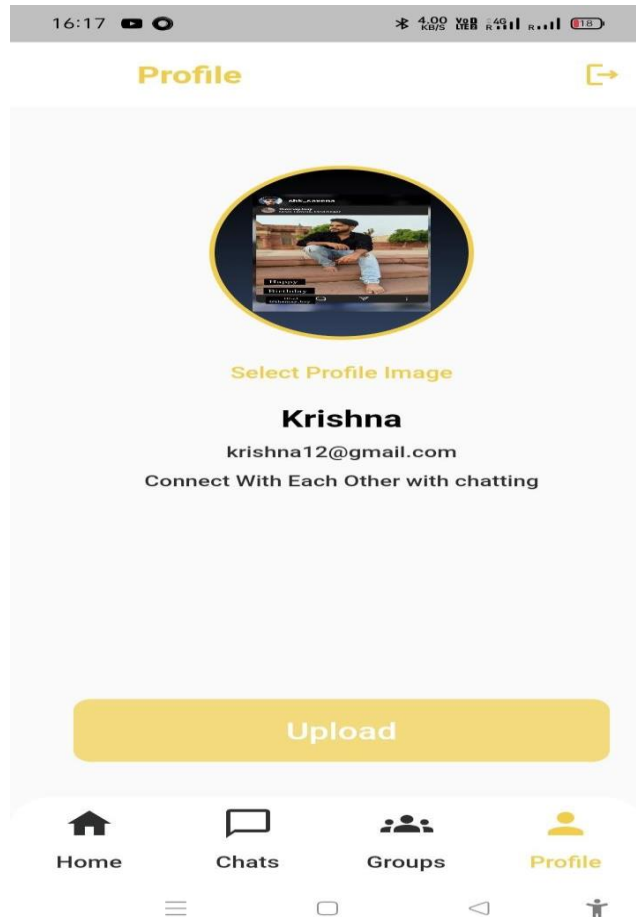


Fig 4.1.5 Profile Page

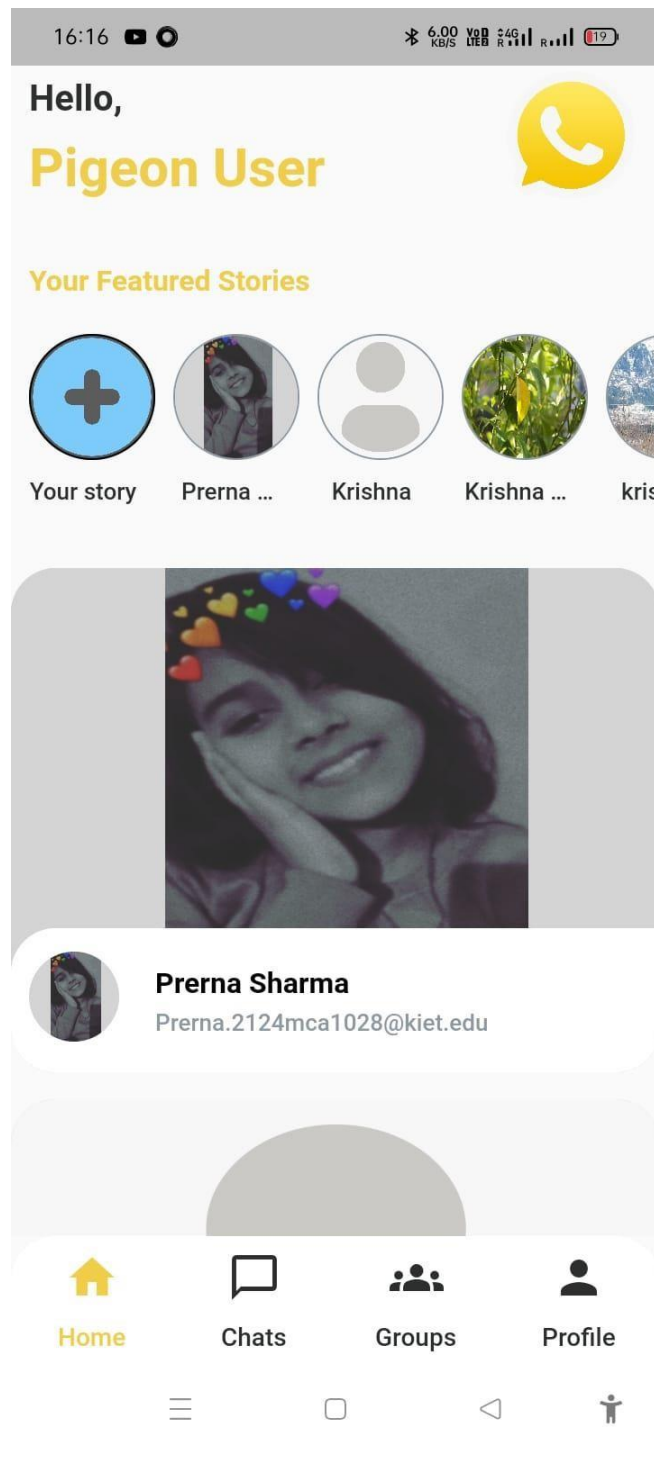


Fig 4.1.6. Home Page

CHAPTER 5

Modules wise code

5.1 Code

5.1.1 Main.dart

```
import
'package:chat_app/helper/helper_function.dart';
import
'package:chat_app/pages/auth/login_page.dart';
import
'package:chat_app/pages/home_page.dart';
import
'package:chat_app/shared/contants.dart';
import
'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'package:flutter/foundation.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();

  if (kIsWeb) {
    //run initilizatio for web
    await
      Firebase.initializeApp
        (options:
          FirebaseOptions(
            apiKey:
              Constants.apiKey,
```

```

void initState() {
  // TODO: implement initState
  getUserLoggedInStatus();
  super.initState();
}

getUserLoggedInStatus() async {
  await HelperFunctions.getUserLoggedInStatus().then((value) {
    if (value != null) {
      setState(() {
        _isSignedIn = value;
      });
    }
  });
}

@override
Widget build(BuildContext context) {
  return MaterialApp(
    debugShowCheckedModeBanner: false,
    theme: ThemeData(
      primaryColor: Constants().primaryColor,
      scaffoldBackgroundColor: Colors.white),
    home: _isSignedIn ? const HomePage() : const LoginPage(),
  );
}
}

```

5.1.2 Register_page.dart

```

import 'package:chat_app/helper/helper_function.dart';
import 'package:chat_app/pages/auth/login_page.dart';
import 'package:chat_app/pages/home_page.dart';
import 'package:chat_app/services/auth_services.dart';
import 'package:chat_app/widgets/widget.dart';
import 'package:flutter/gestures.dart';
import 'package:flutter/material.dart';

```

```

class RegisterPage extends StatefulWidget {
  const RegisterPage({Key? key}) : super(key: key);

  @override
  State<RegisterPage> createState() => _RegisterPageState();
}

class _RegisterPageState extends State<RegisterPage> {
  bool _isLoading = false;
  final formKey = GlobalKey<FormState>();
  String email = "";
  String password = "";
  String fullName = "";
  AuthService authService = AuthService();
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: _isLoading
        ? Center(
            child: CircularProgressIndicator(
              color: Theme.of(context).primaryColor))
        : SingleChildScrollView(
            child: Padding(
              padding:
                const EdgeInsets.symmetric(horizontal: 20, vertical: 80),
              child: Form(
                key: formKey,
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.center,
                  crossAxisAlignment: CrossAxisAlignment.center,
                  children: <Widget>[
                    const Text(
                      "Pigeon Chat",
                      style: TextStyle(
                        fontSize: 40, fontWeight: FontWeight.bold),
                    ),
                    const SizedBox(height: 10),
                    const Text(
                      "Create your account now to chat and explore",
                      style: TextStyle(
                        fontSize: 15, fontWeight: FontWeight.w400)),
                    Image.asset("assets/register.png"),
                    TextFormField(
                      decoration: textInputDecoration.copyWith(
                        labelText: "Full Name",

```

```

        prefixIcon: Icon(
            Icons.person,
            color: Theme.of(context).primaryColor,
        )),
    onChanged: (val) {
        setState(() {
            fullName = val;
        });
    },
    validator: (val) {
        if (val!.isEmpty) {
            return null;
        } else {
            return "Name cannot be empty";
        }
    },
),
const SizedBox(
    height: 15,
),
TextFormField(
    decoration: textInputDecoration.copyWith(
        labelText: "Email",
        prefixIcon: Icon(
            Icons.email,
            color: Theme.of(context).primaryColor,
        )),
    onChanged: (val) {
        setState(() {
            email = val;
        });
    },

    // check the validation
    validator: (val) {
        return RegExp(
            r"^[a-zA-Z0-9.a-zA-Z0-9.!#$%&'*-+
/=^?^_`{ | }~]+@[a-zA-Z0-9]+\.[a-zA-Z]+")
            .hasMatch(val!)
            ? null
            : "Please enter a valid email";
    },
),
const SizedBox(height: 15),
TextFormField(

```

```

        obscureText: true,
        decoration: textInputDecoration.copyWith(
          labelText: "Password",
          prefixIcon: Icon(
            Icons.lock,
            color: Theme.of(context).primaryColor,
          )),
        validator: (val) {
          if (val!.length < 6) {
            return "Password must be at least 6 characters";
          } else {
            return null;
          }
        },
        onChanged: (val) {
          setState(() {
            password = val;
          });
        },
      ),
      const SizedBox(
        height: 20,
      ),
      SizedBox(
        width: double.infinity,
        child: ElevatedButton(
          style: ElevatedButton.styleFrom(
            primary: Theme.of(context).primaryColor,
            elevation: 0,
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(30))),
          child: const Text(
            "Register",
            style:
              TextStyle(color: Colors.white, fontSize: 16),
          ),
          onPressed: () {
            register();
          },
        ),
      ),
      const SizedBox(
        height: 10,
      ),
      Text.rich(TextSpan(

```



```

        text: "Already have an account? ",
        style: const TextStyle(
          color: Colors.black, fontSize: 14),
        children: <TextSpan>[
          TextSpan(
            text: "Login now",
            style: const TextStyle(
              color: Colors.black,
              decoration: TextDecoration.underline),
            recognizer: TapGestureRecognizer()
              ..onTap = () {
                nextScreen(context, const LoginPage());
              },
          ],
        ]),
      ],
    ),
  ),
);
}

register() async {
  if (formKey.currentState!.validate()) {
    setState(() {
      _isLoading = true;
    });
    await authService
      .registerUserWithEmailAndPassword(fullName, email, password)
      .then((value) async {
        if (value == true) {
          // saving the shared preference state
          await HelperFunctions.saveUserLoggedInStatus(true);
          await HelperFunctions.saveUserEmailSF(email);
          await HelperFunctions.saveUserNameSF(fullName);
          nextScreenReplace(context, const HomePage());
        } else {
          showSnackbar(context, Colors.red, value);
          setState(() {
            _isLoading = false;
          });
        }
      });
  }
}
}

```

5.1.3 Login page.dart

```
import 'package:chat_app/helper/helper_function.dart';
import 'package:chat_app/pages/auth/register_page.dart';
import 'package:chat_app/pages/home_page.dart';
import 'package:chat_app/services/auth_services.dart';
import 'package:chat_app/services/database_service.dart';
import 'package:chat_app/widgets/widget.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/gestures.dart';
import 'package:flutter/material.dart';

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

  @override
  State<LoginPage> createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  final formKey = GlobalKey<FormState>();
  String email = "";
  String password = "";
  bool _isLoading = false;
  AuthService authService = AuthService();
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).primaryColor,
      ),
      body: _isLoading
        ? Center(
            child: CircularProgressIndicator(
              color: Theme.of(context).primaryColor,
            )
          )
        : SingleChildScrollView(
```

```

child: Padding(
  padding:
    const EdgeInsets.symmetric(horizontal: 20, vertical: 20),
  child: Form(
    key: formKey,
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      crossAxisAlignment: CrossAxisAlignment.center,
      children: <Widget>[
        const Text(
          "Pigeon Chat",
          style: TextStyle(
            fontSize: 40, fontWeight: FontWeight.bold),
        ),
        const SizedBox(height: 10),
        const Text(
          "Login now to see what they are talking",
          style: TextStyle(
            fontSize: 15, fontWeight: FontWeight.w400),
        ),
        Image.asset("assets/login.png"),
        TextFormField(
          decoration: textInputDecoration.copyWith(
            labelText: "Email",
            prefixIcon: Icon(
              Icons.email,
              color: Theme.of(context).primaryColor,
            ),
          ),
          onChanged: (value) {
            setState(() {
              email = value;
            });
          },
          validator: (val) {
            return RegExp(
              r"^[a-zA-Z0-9.a-zA-Z0-9.!#$%&'*-+
/= ?^_`{ | } ~]+@[a-zA-Z0-9]+\.[a-zA-Z]+")
              .hasMatch(val!)
              ? null
              : "Please enter a valid email";
            },
          ),
        const SizedBox(
          height: 15,

```

```

    ),
    TextFormField(
      obscureText: true,
      decoration: textInputDecoration.copyWith(
        labelText: "Password",
        prefixIcon: Icon(
          Icons.key,
          color: Theme.of(context).primaryColor,
        ),
      ),
    ),
    validator: (value) {
      if (value!.length < 6) {
        return ("Password must be at least 6 characters");
      } else {
        return (null);
      }
    },
    onChanged: (value) {
      setState(() {
        password = value;
      });
    },
  ),
  const SizedBox(
    height: 20,
  ),
  SizedBox(
    width: double.infinity,
    child: ElevatedButton(
      style: ElevatedButton.styleFrom(
        primary: Theme.of(context).primaryColor,
        elevation: 0,
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(30))),
      child: const Text("Sign In",
        style:
          TextStyle(color: Colors.white, fontSize: 16)),
      onPressed: () {
        login();
      },
    ),
  ),
  const SizedBox(
    height: 10,
  ),

```

```

        Text.rich(
          TextSpan(
            text: "Don't have an account? ",
            style: const TextStyle(
              color: Colors.black, fontSize: 14),
            children: <TextSpan>[
              TextSpan(
                text: "Register here",
                style: const TextStyle(
                  color: Colors.black,
                  decoration: TextDecoration.underline),
                recognizer: TapGestureRecognizer()
                  ..onTap = () {
                    nextScreen(context, RegisterPage());
                  },
              ],
            ),
          ),
        ),
      ),
    ),
  ),
);
}

login() async {
  if (formKey.currentState!.validate()) {
    setState(() {
      _isLoading = true;
    });
    await authService
      .loginWithUserNameAndPassword(email, password)
      .then((value) async {
        if (value == true) {
          QuerySnapshot snapshot =
            await DatabaseService(uid: FirebaseAuth.instance.currentUser!.uid)
              .gettingUserData(email);
          // saving the values to our shared preferences
          await HelperFunctions.saveUserLoggedInStatus(true);
          await HelperFunctions.saveUserEmailSF(email);
          await HelperFunctions.saveUserNameSF(snapshot.docs[0]['fullName']);
          nextScreenReplace(context, const HomePage());
        } else {
          showSnackbar(context, Colors.red, value);
        }
      });
  }
}

```

5.1.4 Auth_service.dart

```
// import 'package:chatapp_firebase/helper/helper_function.dart';
// import 'package:chatapp_firebase/service/database_service.dart';
import 'package:chat_app/helper/helper_function.dart';
import 'package:chat_app/services/database_service.dart';
import 'package:firebase_auth/firebase_auth.dart';

class AuthService {
  final FirebaseAuth firebaseAuth = FirebaseAuth.instance;

  // Login
  Future loginWithUserNameAndPassword(String email, String password) async {
    try {
      User user = (await firebaseAuth.signInWithEmailAndPassword(
        email: email, password: password))
        .user!;

      if (user != null) {
        return true;
      }
    } on FirebaseAuthException catch (e) {
      return e.message;
    }
  }

  // register
  Future registerUserWithEmailAndPassword(
    String fullName, String email, String password) async {
```

```

try {
  User user = (await firebaseAuth.createUserWithEmailAndPassword(
    email: email, password: password))
    .user!;

  if (user != null) {
    // call our database service to update the user data.
    await DatabaseService(uid: user.uid).savingUserData(fullName, email);
    return true;
  }
} on FirebaseAuthException catch (e) {
  return e.message;
}

// signout
Future signOut() async {
  try {
    await HelperFunctions.saveUserLoggedInStatus(false);
    await HelperFunctions.saveUserEmailSF("");
    await HelperFunctions.saveUserNameSF("");
    await firebaseAuth.signOut();
  } catch (e) {
    return null;
  }
}
}

```

5.1.5 Database_service.dart

```

import 'package:cloud_firestore/cloud_firestore.dart';

class DatabaseService {
  final String? uid;
  DatabaseService({this.uid});
}

```

```

// reference for our collections
final CollectionReference userCollection =
    FirebaseFirestore.instance.collection("users");
final CollectionReference groupCollection =
    FirebaseFirestore.instance.collection("groups");

// saving the userdata
Future savingUserData(String fullName, String email) async {
    return await userCollection.doc(uid).set({
        "fullName": fullName,
        "email": email,
        "groups": [],
        "profilePic": "",
        "uid": uid,
    });
}

// getting user data
Future gettingUserData(String email) async {
    QuerySnapshot snapshot =
        await userCollection.where("email", isEqualTo: email).get();
    return snapshot;
}

// get user groups
getUserGroups() async {
    return userCollection.doc(uid).snapshots();
}

// creating a group
Future createGroup(String userName, String id, String groupName) async {
    DocumentReference groupDocumentReference = await groupCollection.add({
        "groupName": groupName,
        "groupIcon": "",
        "admin": "${id}_${userName}",
        "members": [],
        "groupId": "",
        "recentMessage": "",
        "recentMessageSender": "",
    });
    // update the members
    await groupDocumentReference.update({
        "members": FieldValue.arrayUnion(["${uid}_${userName}"]),
        "groupId": groupDocumentReference.id,
    });
}

```



```

    DocumentReference userDocumentReference = userCollection.doc(uid);
    return await userDocumentReference.update({
        "groups":
            FieldValue.arrayUnion(["${groupDocumentReference.id}_${groupName}"])
    });
}

// getting the chats
getChats(String groupId) async {
    return groupCollection
        .doc(groupId)
        .collection("messages")
        .orderBy("time")
        .snapshots();
}

Future getGroupAdmin(String groupId) async {
    DocumentReference d = groupCollection.doc(groupId);
    DocumentSnapshot documentSnapshot = await d.get();
    return documentSnapshot['admin'];
}

// get group members
getGroupMembers(groupId) async {
    return groupCollection.doc(groupId).snapshots();
}

// search
searchByName(String groupName) {
    return groupCollection.where("groupName", isEqualTo: groupName).get();
}

// function -> bool
Future<bool> isUserJoined(
    String groupName, String groupId, String userName) async {
    DocumentReference userDocumentReference = userCollection.doc(uid);
    DocumentSnapshot documentSnapshot = await userDocumentReference.get();

    List<dynamic> groups = await documentSnapshot['groups'];
    if (groups.contains("${groupId}_${groupName}")) {
        return true;
    } else {
        return false;
    }
}

```

```

}

// toggling the group join/exit
Future toggleGroupJoin(
    String groupId, String userName, String groupName) async {
    // doc reference
    DocumentReference userDocumentReference = userCollection.doc(uid);
    DocumentReference groupDocumentReference = groupCollection.doc(groupId);

    DocumentSnapshot documentSnapshot = await userDocumentReference.get();
    List<dynamic> groups = await documentSnapshot['groups'];

    // if user has our groups -> then remove then or also in other part re join
    if (groups.contains("${groupId}_${groupName}")) {
        await userDocumentReference.update({
            "groups": FieldValue.arrayRemove(["${groupId}_${groupName}"])
        });
        await groupDocumentReference.update({
            "members": FieldValue.arrayRemove(["${uid}_${userName}"])
        });
    } else {
        await userDocumentReference.update({
            "groups": FieldValue.arrayUnion(["${groupId}_${groupName}"])
        });
        await groupDocumentReference.update({
            "members": FieldValue.arrayUnion(["${uid}_${userName}"])
        });
    }
}

// send message
sendMessage(String groupId, Map<String, dynamic> chatMessageData) async {
    groupCollection.doc(groupId).collection("messages").add(chatMessageData);
    groupCollection.doc(groupId).update({
        "recentMessage": chatMessageData['message'],
        "recentMessageSender": chatMessageData['sender'],
        "recentMessageTime": chatMessageData['time'].toString(),
    });
}
}

```

5.1.5 Widget.dart

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

const InputDecoration = InputDecoration(
  labelStyle: TextStyle(color: Colors.black, fontWeight: FontWeight.w300),
  focusedBorder: OutlineInputBorder(
    borderSide:
      BorderSide(color: Color.fromARGB(255, 137, 215, 251), width: 2)),
  enabledBorder: OutlineInputBorder(
    borderSide:
      BorderSide(color: Color.fromARGB(255, 137, 215, 251), width: 2)),
  errorBorder: OutlineInputBorder(
    borderSide:
      BorderSide(color: Color.fromARGB(255, 137, 215, 251), width: 2)),
);

void nextScreen(context, page) {
  Navigator.push(context, MaterialPageRoute(builder: (context) => page));
}

void nextScreenReplace(context, page) {
  Navigator.pushReplacement(
    context, MaterialPageRoute(builder: (context) => page));
}

void showSnackBar(context, color, message) {
  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(
      content: Text(
        message,
        style: const TextStyle(fontSize: 14),
      ),
      backgroundColor: color,
      duration: const Duration(seconds: 2),
      action: SnackBarAction(
        label: "OK",
        onPressed: () {},
        textColor: Colors.white,
      ),
    ),
  );
}
```

5.1.6 Hepler_function.dart

```
import 'package:shared_preferences/shared_preferences.dart';

class HelperFunctions {
  //keys
  static String userLoggedInKey = "LOGGEDINKEY";
  static String userNameKey = "USERNAMEKEY";
  static String userEmailKey = "USEREMAILKEY";

  // saving the data to SF

  static Future<bool> saveUserLoggedInStatus(bool isUserLoggedIn) async {
    SharedPreferences sf = await SharedPreferences.getInstance();
    return await sf.setBool(userLoggedInKey, isUserLoggedIn);
  }

  static Future<bool> saveUserNameSF(String userName) async {
    SharedPreferences sf = await SharedPreferences.getInstance();
    return await sf.setString(userNameKey, userName);
  }

  static Future<bool> saveUserEmailSF(String userEmail) async {
    SharedPreferences sf = await SharedPreferences.getInstance();
    return await sf.setString(userEmailKey, userEmail);
  }

  // getting the data from SF

  static Future<bool?> getUserLoggedInStatus() async {
    SharedPreferences sf = await SharedPreferences.getInstance();
    return sf.getBool(userLoggedInKey);
  }

  static Future<String?> getUserEmailFromSF() async {
    SharedPreferences sf = await SharedPreferences.getInstance();
    return sf.getString(userEmailKey);
  }

  static Future<String?> getUserNameFromSF() async {
    SharedPreferences sf = await SharedPreferences.getInstance();
    return sf.getString(userNameKey);
  }
}
```

```
}
```

5.1.7 Home_page.dart

```
import 'package:chat_app/helper/helper_function.dart';
import 'package:chat_app/pages/auth/login_page.dart';
import 'package:chat_app/pages/profile_page.dart';
import 'package:chat_app/pages/search_page.dart';
import 'package:chat_app/services/auth_services.dart';
import 'package:chat_app/services/database_service.dart';
import 'package:chat_app/widgets/group_tile.dart';
import 'package:chat_app/widgets/widget.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';

class HomePage extends StatefulWidget {
  const HomePage({Key? key}) : super(key: key);

  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  String userName = "";
  String email = "";
  AuthService authService = AuthService();
  Stream? groups;
  bool _isLoading = false;
  String groupName = "";

  @override
  void initState() {
    super.initState();
    gettingUserData();
  }

  // string manipulation
  String getId(String res) {
    return res.substring(0, res.indexOf("_"));
  }

  String getName(String res) {
    return res.substring(res.indexOf("_") + 1);
  }
}
```

```

        setState(() {
            email = value!;
        });
    });
    await HelperFunctions.getUserNameFromSF().then((val) {
        setState(() {
            userName = val!;
        });
    });
    // getting the list of snapshots in our stream
    await DatabaseService(uid: FirebaseAuth.instance.currentUser!.uid)
        .getUserGroups()
        .then((snapshot) {
            setState(() {
                groups = snapshot;
            });
        });
}

@override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
            actions: [
                IconButton(
                    onPressed: () {
                        nextScreen(context, const SearchPage());
                    },
                    icon: const Icon(
                        Icons.search,
                    ))
            ],
        elevation: 0,
        centerTitle: true,
        backgroundColor: Theme.of(context).primaryColor,
        title: const Text(
            "Groups",
            style: TextStyle(
                color: Colors.white, fontWeight: FontWeight.bold, fontSize: 27),
        ),
        drawer: Drawer(
            child: ListView(
                padding: const EdgeInsets.symmetric(vertical: 50),

```

```

Icon(
  Icons.account_circle,
  size: 150,
  color: Colors.grey[700],
),
const SizedBox(
  height: 15,
),
Text(
  userName,
  textAlign: TextAlign.center,
  style: const TextStyle(fontWeight: FontWeight.bold),
),
const SizedBox(
  height: 30,
),
const Divider(
  height: 2,
),
ListTile(
  onTap: () {},
  selectedColor: Theme.of(context).primaryColor,
  selected: true,
  contentPadding:
    const EdgeInsets.symmetric(horizontal: 20, vertical: 5),
  leading: const Icon(Icons.group),
  title: const Text(
    "Groups",
    style: TextStyle(color: Colors.black),
  ),
),
),
ListTile(
  onTap: () {
    nextScreenReplace(
      context,
      ProfilePage(
        userName: userName,
        email: email,
      ));
  },
  contentPadding:
    const EdgeInsets.symmetric(horizontal: 20, vertical: 5),
  leading: const Icon(Icons.group),
  title: const Text(
    "Profile",

```

```

        style: TextStyle(color: Colors.black),
    ),
),
ListTile(
  onTap: () async {
    showDialog(
      barrierDismissible: false,
      context: context,
      builder: (context) {
        return AlertDialog(
          title: const Text("Logout"),
          content: const Text("Are you sure you want to logout?"),
          actions: [
            IconButton(
              onPressed: () {
                Navigator.pop(context);
              },
              icon: const Icon(
                Icons.cancel,
                color: Colors.red,
              ),
            ),
            IconButton(
              onPressed: () async {
                await authService.signOut();
                Navigator.of(context).pushAndRemoveUntil(
                  MaterialPageRoute(
                    builder: (context) => const LoginPage(),
                    (route) => false);
              },
              icon: const Icon(
                Icons.done,
                color: Colors.green,
              ),
            ),
          ],
        );
      },
    );
  },
  contentPadding:
    const EdgeInsets.symmetric(horizontal: 20, vertical: 5),
  leading: const Icon(Icons.exit_to_app),
  title: const Text(
    "Logout",
    style: TextStyle(color: Colors.black),
  ),
),

```



```

        style: const TextStyle(color: Colors.black),
        decoration: InputDecoration(
          enabledBorder: OutlineInputBorder(
            borderSide: BorderSide(
              color: Theme.of(context).primaryColor,
              borderRadius: BorderRadius.circular(20)),
          errorBorder: OutlineInputBorder(
            borderSide:
              const BorderSide(color: Colors.red),
            borderRadius: BorderRadius.circular(20)),
          focusedBorder: OutlineInputBorder(
            borderSide: BorderSide(
              color: Theme.of(context).primaryColor,
              borderRadius: BorderRadius.circular(20))),
        ),
      ],
    ),
    actions: [
      ElevatedButton(
        onPressed: () {
          Navigator.of(context).pop();
        },
        style: ElevatedButton.styleFrom(
          primary: Theme.of(context).primaryColor,
          child: const Text("CANCEL"),
        ),
      ),
      ElevatedButton(
        onPressed: () async {
          if (groupName != "") {
            setState(() {
              _isLoading = true;
            });
            DatabaseService(
              uid: FirebaseAuth.instance.currentUser!.uid)
              .createGroup(userName,
                FirebaseAuth.instance.currentUser!.uid, groupName)
              .whenComplete(() {
                _isLoading = false;
              });
            Navigator.of(context).pop();
            showSnackBar(
              context, Colors.green, "Group created successfully.");
          }
        },
        style: ElevatedButton.styleFrom(

```

```

        primary: Theme.of(context).primaryColor),
        child: const Text("CREATE"),
      )
    ],
  );
}));
});
}

groupList() {
  return StreamBuilder(
    stream: groups,
    builder: (context, AsyncSnapshot snapshot) {
      // make some checks
      if (snapshot.hasData) {
        if (snapshot.data['groups'] != null) {
          if (snapshot.data['groups'].length != 0) {
            return ListView.builder(
              itemCount: snapshot.data['groups'].length,
              itemBuilder: (context, index) {
                int reverseIndex = snapshot.data['groups'].length - index - 1;
                return GroupTile(
                  groupId: getId(snapshot.data['groups'][reverseIndex]),
                  groupName: getName(snapshot.data['groups'][reverseIndex]),
                  userName: snapshot.data['fullName']);
              },
            );
          } else {
            return noGroupWidget();
          }
        } else {
          return noGroupWidget();
        }
      } else {
        return Center(
          child: CircularProgressIndicator(
            color: Theme.of(context).primaryColor,
          ),
        );
      }
    },
  );
}

noGroupWidget() {
  return Container(

```

```

padding: const EdgeInsets.symmetric(horizontal: 25),
child: Column(
  mainAxisAlignment: MainAxisAlignment.center,
  crossAxisAlignment: CrossAxisAlignment.center,
  children: [
    GestureDetector(
      onTap: () {
        popUpDialog(context);
      },
      child: Icon(
        Icons.add_circle,
        color: Colors.grey[700],
        size: 75,
      ),
    ),
    const SizedBox(
      height: 20,
    ),
    const Text(
      "You've not joined any groups, tap on the add icon to create a group
or also search from top search button.",
      textAlign: TextAlign.center,
    )
  ],
),
);
}

```

5.1.8 Group_tile.dart

```

import 'package:chat_app/pages/chat_page.dart';
import 'package:chat_app/widgets/widget.dart';
import 'package:flutter/material.dart';

class GroupTile extends StatefulWidget {
  final String userName;
  final String groupId;

```

```

final String groupName;
const GroupTile(
  {Key? key,
   required this.groupId,
   required this.groupName,
   required this.userName})
  : super(key: key);

@override
State<GroupTile> createState() => _GroupTileState();
}

class _GroupTileState extends State<GroupTile> {
  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () {
        nextScreen(
          context,
          ChatPage(
            groupId: widget.groupId,
            groupName: widget.groupName,
            userName: widget.userName,
          ));
      },
      child: Container(
        padding: const EdgeInsets.symmetric(vertical: 10, horizontal: 5),
        child: ListTile(
          leading: CircleAvatar(
            radius: 30,
            backgroundColor: Theme.of(context).primaryColor,
            child: Text(
              widget.groupName.substring(0, 1).toUpperCase(),
              textAlign: TextAlign.center,
              style: const TextStyle(
                color: Colors.white, fontWeight: FontWeight.w500),
            ),
          ),
          title: Text(
            widget.groupName,
            style: const TextStyle(fontWeight: FontWeight.bold),
          ),
          subtitle: Text(
            "Join the conversation as ${widget.userName}",
            style: const TextStyle(fontSize: 13),
          ),
        ),
      ),
    );
  }
}

```

```

    ),
  ),
),
);
}
}

```

5.1.9 Message_tile.dart

```

import 'package:flutter/material.dart';

class MessageTile extends StatefulWidget {
  final String message;
  final String sender;
  final bool sentByMe;

  const MessageTile(
    {Key? key,
    required this.message,
    required this.sender,
    required this.sentByMe})
    : super(key: key);

  @override
  State<MessageTile> createState() => _MessageTileState();
}

class _MessageTileState extends State<MessageTile> {
  @override
  Widget build(BuildContext context) {
    return Container(
      padding: EdgeInsets.only(
        top: 4,
        bottom: 4,
        left: widget.sentByMe ? 0 : 24,
        right: widget.sentByMe ? 24 : 0),
      alignment: widget.sentByMe ? Alignment.centerRight : Alignment.centerLeft,
      child: Container(

```

```

margin: widget.sentByMe
  ? const EdgeInsets.only(left: 30)
  : const EdgeInsets.only(right: 30),
padding:
  const EdgeInsets.only(top: 17, bottom: 17, left: 20, right: 20),
decoration: BoxDecoration(
  borderRadius: widget.sentByMe
    ? const BorderRadius.only(
      topLeft: Radius.circular(20),
      topRight: Radius.circular(20),
      bottomLeft: Radius.circular(20),
    )
    : const BorderRadius.only(
      topLeft: Radius.circular(20),
      topRight: Radius.circular(20),
      bottomRight: Radius.circular(20),
    ),
  color: widget.sentByMe
    ? Theme.of(context).primaryColor
    : Colors.grey[700]),
child: Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
    Text(
      widget.sender.toUpperCase(),
      textAlign: TextAlign.start,
      style: const TextStyle(
        fontSize: 13,
        fontWeight: FontWeight.bold,
        color: Colors.white,
        letterSpacing: -0.5),
    ),
    const SizedBox(
      height: 8,
    ),
    Text(widget.message,
      textAlign: TextAlign.start,
      style: const TextStyle(fontSize: 16, color: Colors.white))
  ],
),
),
);
}
}

```

5.1.10 Profile_page.dart

```
import 'package:chat_app/pages/auth/login_page.dart';
import 'package:chat_app/pages/home_page.dart';
import 'package:chat_app/services/auth_services.dart';
import 'package:chat_app/widgets/widget.dart';
import 'package:flutter/material.dart';

class ProfilePage extends StatefulWidget {
  String userName;
  String email;
  ProfilePage({Key? key, required this.email, required this.userName})
    : super(key: key);

  @override
  State<ProfilePage> createState() => _ProfilePageState();
}

class _ProfilePageState extends State<ProfilePage> {
  AuthService authService = AuthService();
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).primaryColor,
        elevation: 0,
        title: const Text(
          "Profile",
          style: TextStyle(
            color: Colors.white, fontSize: 27, fontWeight: FontWeight.bold),
        ),
      ),
      drawer: Drawer(
        child: ListView(
          padding: const EdgeInsets.symmetric(vertical: 50),
          children: <Widget>[
            Icon(
              Icons.account_circle,
              size: 150,
              color: Colors.grey[700],
            ),
            const SizedBox(
              height: 15,
```

```

    ),
    Text(
      widget.userName,
      textAlign: TextAlign.center,
      style: const TextStyle(fontWeight: FontWeight.bold),
    ),
    const SizedBox(
      height: 30,
    ),
    const Divider(
      height: 2,
    ),
    ListTile(
      onTap: () {
        nextScreen(context, const HomePage());
      },
      contentPadding:
        const EdgeInsets.symmetric(horizontal: 20, vertical: 5),
      leading: const Icon(Icons.group),
      title: const Text(
        "Groups",
        style: TextStyle(color: Colors.black),
      ),
    ),
    ListTile(
      onTap: () {},
      selected: true,
      selectedColor: Theme.of(context).primaryColor,
      contentPadding:
        const EdgeInsets.symmetric(horizontal: 20, vertical: 5),
      leading: const Icon(Icons.group),
      title: const Text(
        "Profile",
        style: TextStyle(color: Colors.black),
      ),
    ),
    ListTile(
      onTap: () async {
        showDialog(
          barrierDismissible: false,
          context: context,
          builder: (context) {
            return AlertDialog(
              title: const Text("Logout"),
              content: const Text("Are you sure you want to logout?"),
            );
          },
        );
      },
    ),
  ),
);

```



```

        actions: [
          IconButton(
            onPressed: () {
              Navigator.pop(context);
            },
            icon: const Icon(
              Icons.cancel,
              color: Colors.red,
            ),
          ),
          IconButton(
            onPressed: () async {
              await authService.signOut();
              Navigator.of(context).pushAndRemoveUntil(
                MaterialPageRoute(
                  builder: (context) => const LoginPage(),
                  (route) => false);
            ),
            icon: const Icon(
              Icons.done,
              color: Colors.green,
            ),
          ),
        ],
      );
    });
  },
  contentPadding:
    const EdgeInsets.symmetric(horizontal: 20, vertical: 5),
  leading: const Icon(Icons.exit_to_app),
  title: const Text(
    "Logout",
    style: TextStyle(color: Colors.black),
  ),
)
],
)),
body: Container(
  padding: const EdgeInsets.symmetric(horizontal: 40, vertical: 170),
  child: Column(
    mainAxisAlignment: MainAxisAlignment.start,
    children: [
      Icon(
        Icons.account_circle,
        size: 200,

```

```

        color: Colors.grey[700],
      ),
      const SizedBox(
        height: 15,
      ),
      Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: [
          const Text("Full Name", style: TextStyle(fontSize: 17)),
          Text(widget.userName, style: const TextStyle(fontSize: 17)),
        ],
      ),
      const Divider(
        height: 20,
      ),
      Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: [
          const Text("Email", style: TextStyle(fontSize: 17)),
          Text(widget.email, style: const TextStyle(fontSize: 17)),
        ],
      ),
    ],
  ),
),
);
}
}

```

5.2.1 Search_page.dart

```

import 'package:chat_app/helper/helper_function.dart';
import 'package:chat_app/pages/chat_page.dart';
import 'package:chat_app/services/database_service.dart';
import 'package:chat_app/widgets/widget.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';

```

```

class SearchPage extends StatefulWidget {
  const SearchPage({Key? key}) : super(key: key);

  @override
  State<SearchPage> createState() => _SearchPageState();
}

class _SearchPageState extends State<SearchPage> {
  TextEditingController searchController = TextEditingController();
  bool isLoading = false;
  QuerySnapshot? searchSnapshot;
  bool hasUserSearched = false;
  String userName = "";
  bool isJoined = false;
  User? user;

  @override
  void initState() {
    super.initState();
    getCurrentUserIdandName();
  }

  getCurrentUserIdandName() async {
    await HelperFunctions.getUserNameFromSF().then((value) {
      setState(() {
        userName = value!;
      });
    });
    user = FirebaseAuth.instance.currentUser;
  }

  String getName(String r) {
    return r.substring(r.indexOf("_") + 1);
  }

  String getId(String res) {
    return res.substring(0, res.indexOf("_"));
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        elevation: 0,

```

```

        backgroundColor: Theme.of(context).primaryColor,
        title: const Text(
            "Search",
            style: TextStyle(
                fontSize: 27, fontWeight: FontWeight.bold, color: Colors.white),
        ),
    ),
    body: Column(
        children: [
            Container(
                color: Theme.of(context).primaryColor,
                padding: const EdgeInsets.symmetric(horizontal: 15, vertical: 10),
                child: Row(
                    children: [
                        Expanded(
                            child: TextField(
                                controller: searchController,
                                style: const TextStyle(color: Colors.white),
                                decoration: const InputDecoration(
                                    border: InputBorder.none,
                                    hintText: "Search groups. . .",
                                    hintStyle:
                                        TextStyle(color: Colors.white, fontSize: 16)),
                            ),
                        ),
                        GestureDetector(
                            onTap: () {
                                initiateSearchMethod();
                            },
                            child: Container(
                                width: 40,
                                height: 40,
                                decoration: BoxDecoration(
                                    color: Colors.white.withOpacity(0.1),
                                    borderRadius: BorderRadius.circular(40)),
                                child: const Icon(
                                    Icons.search,
                                    color: Colors.white,
                                ),
                            ),
                        ),
                    ],
                ),
            ),
        ],
    ),
    isLoading

```

```

        ? Center(
            child: CircularProgressIndicator(
                color: Theme.of(context).primaryColor,
            )
        ) : groupList(),
    ],
),
);
}

```

```

initiateSearchMethod() async {
  if (searchController.text.isNotEmpty) {
    setState(() {
      isLoading = true;
    });
    await DatabaseService()
      .searchByName(searchController.text)
      .then((snapshot) {
        setState(() {
          searchSnapshot = snapshot;
          isLoading = false;
          hasUserSearched = true;
        });
      });
  }
}

```

```

groupList() {
  return hasUserSearched
    ? ListView.builder(
        shrinkWrap: true,
        itemCount: searchSnapshot!.docs.length,
        itemBuilder: (context, index) {
          return groupTile(
            userName,
            searchSnapshot!.docs[index]['groupId'],
            searchSnapshot!.docs[index]['groupName'],
            searchSnapshot!.docs[index]['admin'],
          );
        },
      )
    : Container();
}

```

```

joinedOrNot(

```

```

        String userName, String groupId, String groupname, String admin) async {
    await DatabaseService(uid: user!.uid)
        .isUserJoined(groupname, groupId, userName)
        .then((value) {
    setState(() {
        isJoined = value;
    });
    });
}

Widget groupTile(
    String userName, String groupId, String groupName, String admin) {
    // function to check whether user already exists in group
    joinedOrNot(userName, groupId, groupName, admin);
    return ListTile(
        contentPadding: const EdgeInsets.symmetric(horizontal: 10, vertical: 10),
        leading: CircleAvatar(
            radius: 30,
            backgroundColor: Theme.of(context).primaryColor,
            child: Text(
                groupName.substring(0, 1).toUpperCase(),
                style: const TextStyle(color: Colors.white),
            ),
        ),
        title:
            Text(groupName, style: const TextStyle(fontWeight: FontWeight.w600)),
        subtitle: Text("Admin: ${getName(admin)}"),
        trailing: InkWell(
            onTap: () async {
                await DatabaseService(uid: user!.uid)
                    .toggleGroupJoin(groupId, userName, groupName);
                if (isJoined) {
                    setState(() {
                        isJoined = !isJoined;
                    });
                    showSnackBar(context, Colors.green, "Successfully joined he group");
                    Future.delayed(const Duration(seconds: 2), () {
                        nextScreen(
                            context,
                            ChatPage(
                                groupId: groupId,
                                groupName: groupName,
                                userName: userName));
                    });
                } else {

```

```

        setState(() {
          isJoined = !isJoined;
          showSnackBar(context, Colors.red, "Left the group $groupName");
        });
      },
      child: isJoined
        ? Container(
            decoration: BoxDecoration(
              borderRadius: BorderRadius.circular(10),
              color: Colors.black,
              border: Border.all(color: Colors.white, width: 1),
            ),
            padding:
              const EdgeInsets.symmetric(horizontal: 20, vertical: 10),
            child: const Text(
              "Joined",
              style: TextStyle(color: Colors.white),
            ),
          )
        : Container(
            decoration: BoxDecoration(
              borderRadius: BorderRadius.circular(10),
              color: Theme.of(context).primaryColor,
            ),
            padding:
              const EdgeInsets.symmetric(horizontal: 20, vertical: 10),
            child: const Text("Join Now",
              style: TextStyle(color: Colors.white)),
          ),
      ),
    );
  }
}

```

5.2.2 Char_page.dart

```
import 'package:chat_app/pages/group_info.dart';
import 'package:chat_app/services/database_service.dart';
import 'package:chat_app/widgets/message_tile.dart';
import 'package:chat_app/widgets/widget.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';

class ChatPage extends StatefulWidget {
  final String groupId;
  final String groupName;
  final String userName;
  const ChatPage(
    {Key? key,
    required this.groupId,
    required this.groupName,
    required this.userName})
    : super(key: key);

  @override
  State<ChatPage> createState() => _ChatPageState();
}

class _ChatPageState extends State<ChatPage> {
  Stream<QuerySnapshot>? chats;
  TextEditingController messageController = TextEditingController();
  String admin = "";

  @override
  void initState() {
    getChatandAdmin();
    super.initState();
  }

  getChatandAdmin() {
    DatabaseService().getChats(widget.groupId).then((val) {
      setState(() {
        chats = val;
      });
    });
    DatabaseService().getGroupAdmin(widget.groupId).then((val) {
      setState(() {
        admin = val;
      });
    });
  }
}
```



```

    });
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      centerTitle: true,
      elevation: 0,
      title: Text(widget.groupName),
      backgroundColor: Theme.of(context).primaryColor,
      actions: [
        IconButton(
          onPressed: () {
            nextScreen(
              context,
              GroupInfo(
                groupId: widget.groupId,
                groupName: widget.groupName,
                adminName: admin,
              ));
          },
          icon: const Icon(Icons.info))
      ],
    ),
    body: Stack(
      children: <Widget>[
        // chat messages here
        Container(
          width: MediaQuery.of(context).size.width,
          height: MediaQuery.of(context).size.height - 145,
          child: chatMessages(),
        ),
        Container(
          alignment: Alignment.bottomCenter,
          width: MediaQuery.of(context).size.width,
          height: MediaQuery.of(context).size.height,
          child: Container(
            padding: const EdgeInsets.symmetric(horizontal: 10, vertical: 10),
            color: Color.fromRGBO(113, 113, 113, .5),
            child: Row(children: [
              Expanded(
                child: TextFormField(
                  controller: messageController,
                  style: const TextStyle(color: Colors.white),

```

```

        decoration: const InputDecoration(
          hintText: "Send a message...",
          hintStyle: TextStyle(color: Colors.white, fontSize: 16),
          border: InputBorder.none,
        ),
      )),
    const SizedBox(
      width: 12,
    ),
    GestureDetector(
      onTap: () {
        sendMessage();
      },
      child: Container(
        height: 50,
        width: 50,
        decoration: BoxDecoration(
          color: Theme.of(context).primaryColor,
          borderRadius: BorderRadius.circular(30),
        ),
        child: const Center(
          child: Icon(
            Icons.send,
            color: Colors.white,
          ),
        ),
      ),
    ),
  ],
),
],
),
);
}

chatMessages() {
  return StreamBuilder(
    stream: chats,
    builder: (context, AsyncSnapshot snapshot) {
      return snapshot.hasData
        ? ListView.builder(
            itemCount: snapshot.data.docs.length,
            itemBuilder: (context, index) {
              return MessageTile(
                message: snapshot.data.docs[index]['message'],

```

```

        sender: snapshot.data.docs[index]['sender'],
        sentByMe: widget.userName ==
            snapshot.data.docs[index]['sender']);
    },
  )
  : Container();
},
);
}

sendMessage() {
  if (messageController.text.isNotEmpty) {
    Map<String, dynamic> chatMessageMap = {
      "message": messageController.text,
      "sender": widget.userName,
      "time": DateTime.now().millisecondsSinceEpoch,
    };

    DatabaseService().sendMessage(widget.groupId, chatMessageMap);
    setState(() {
      messageController.clear();
    });
  }
}
}

```

5.2.3 Group_info.dart

```

import 'package:chat_app/pages/home_page.dart';
import 'package:chat_app/services/database_service.dart';
import 'package:chat_app/widgets/widget.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';

class GroupInfo extends StatefulWidget {
  final String groupId;
  final String groupName;
  final String adminName;
  const GroupInfo(

```

```

        {Key? key,
        required this.adminName,
        required this.groupName,
        required this.groupId})
        : super(key: key);

    @override
    State<GroupInfo> createState() => _GroupInfoState();
}

class _GroupInfoState extends State<GroupInfo> {
    Stream? members;
    @override
    void initState() {
        getMembers();
        super.initState();
    }

    getMembers() async {
        DatabaseService(uid: FirebaseAuth.instance.currentUser!.uid)
            .getGroupMembers(widget.groupId)
            .then((val) {
                setState(() {
                    members = val;
                });
            });
    }

    String getName(String r) {
        return r.substring(r.indexOf("_") + 1);
    }

    String getId(String res) {
        return res.substring(0, res.indexOf("_"));
    }

    @override
    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(
                centerTitle: true,
                elevation: 0,
                backgroundColor: Theme.of(context).primaryColor,
                title: const Text("Group Info"),
                actions: [

```

```

IconButton(
  onPressed: () {
    showDialog(
      barrierDismissible: false,
      context: context,
      builder: (context) {
        return AlertDialog(
          title: const Text("Exit"),
          content:
            const Text("Are you sure you exit the group? "),
          actions: [
            IconButton(
              onPressed: () {
                Navigator.pop(context);
              },
              icon: const Icon(
                Icons.cancel,
                color: Colors.red,
              ),
            ),
            IconButton(
              onPressed: () async {
                DatabaseService(
                  uid: FirebaseAuth
                    .instance.currentUser!.uid)
                  .toggleGroupJoin(
                    widget.groupId,
                    getName(widget.adminName),
                    widget.groupName)
                  .whenComplete(() {
                    nextScreenReplace(context, const HomePage());
                  });
              },
              icon: const Icon(
                Icons.done,
                color: Colors.green,
              ),
            ),
          ],
        );
      });
  },
  icon: const Icon(Icons.exit_to_app))
],
),

```

```

body: Container(
  padding: const EdgeInsets.symmetric(horizontal: 20, vertical: 20),
  child: Column(
    children: [
      Container(
        padding: const EdgeInsets.all(20),
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(30),
          color: Theme.of(context).primaryColor.withOpacity(0.2)),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.start,
          children: [
            CircleAvatar(
              radius: 30,
              backgroundColor: Theme.of(context).primaryColor,
              child: Text(
                widget.groupName.substring(0, 1).toUpperCase(),
                style: const TextStyle(
                  fontWeight: FontWeight.w500, color: Colors.white),
              ),
            ),
            const SizedBox(
              width: 20,
            ),
            Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(
                  "Group: ${widget.groupName}",
                  style: const TextStyle(fontWeight: FontWeight.w500),
                ),
                const SizedBox(
                  height: 5,
                ),
                Text("Admin: ${getName(widget.adminName)}")
              ],
            ),
          ],
        ),
      ),
      memberList(),
    ],
  ),
);

```

```

}

memberList() {
  return StreamBuilder(
    stream: members,
    builder: (context, AsyncSnapshot snapshot) {
      if (snapshot.hasData) {
        if (snapshot.data['members'] != null) {
          if (snapshot.data['members'].length != 0) {
            return ListView.builder(
              itemCount: snapshot.data['members'].length,
              shrinkWrap: true,
              itemBuilder: (context, index) {
                return Container(
                  padding:
                    const EdgeInsets.symmetric(horizontal: 5, vertical: 10),
                  child: ListTile(
                    leading: CircleAvatar(
                      radius: 30,
                      backgroundColor: Theme.of(context).primaryColor,
                      child: Text(
                        getName(snapshot.data['members'][index])
                          .substring(0, 1)
                          .toUpperCase(),
                        style: const TextStyle(
                          color: Colors.white,
                          fontSize: 15,
                          fontWeight: FontWeight.bold),
                      ),
                    ),
                    title: Text(getName(snapshot.data['members'][index])),
                    subtitle: Text(getId(snapshot.data['members'][index])),
                  ),
                );
              },
            );
          } else {
            return const Center(
              child: Text("NO MEMBERS"),
            );
          }
        } else {
          return const Center(
            child: Text("NO MEMBERS"),
          );
        }
      }
    },
  );
}

```

CHAPTER 6

CONCLUSION

Our project is only a humble venture to satisfy the needs to manage their project work. Several user-Pigeon coding has also adopted. Thaïs's package shall prove to be a powerful package in satisfying all the requirements of the school. The objective of software planning is to provide a frame word that enables the manager to make reasonable estimates made within a limited time frame at the beginning of the software project and should be updated regularly as the project progresses.

At the end it is concluded that we have made effort on following points

- A description of the background and context of the project and its relation to work already done in the area.
- Made a statement of the aims and objectives of the project.
- Description of Purpose, Scope, and applicability.
- We define the problem on which we are working in the project.
- We describe the requirement specifications of the system and the actions that can be done on these things.
- We understand the problem domain and produce a model of the system, which describes operations that can be performed on the system.
- We included features and operations in detail, including screen layouts.
- We designed user interface and security issues related to system.
- Finally, the system is implemented and tested according to test cases.

CHAPTER 7

FUTURE SCOPE OF PROJECT

- We can give more advance software for Online Chat Application including more facilities.
- We will host the platform on online servers to make it accessible worldwide. Integrate multiple load balancers to distribute the loads of the system.
- Include offline chatting within a particular range.
- Implement the backup mechanism for taking backup of codebase and database on regular basis on different servers.

CHAPTER 8

BIBILOGRAPHY

- <https://www.tutorialspoint.com>
- <https://www.javatpoint.com>
- <https://www.w3school.com>
- <https://www.youtube.com>
- <https://www.google.com>
- <https://www.wikipedia.com>
- <https://www.geeksforgeeks.com>
- <https://www.studocu.com>

CHAPTER 9

REFERENCES

- Chen, Y., Huang, C., & Shen, H. (2016). An empirical investigation of WeChat voice and video calls in China: Cultural differences in the use of communication technologies. *International Journal of Human-Computer Interaction*, 32(11), 873-882.
- Kang, C., Chen, Y., & Li, Y. (2017). Multilingual chat groups: Insights into intercultural communication through a case study. *International Journal of Human-Computer Interaction*, 33(11), 861-869.
- Lin, Y., Lu, H. P., & Liu, C. (2016). Examining the relationships among perceived enjoyment, social influence, and trust in the acceptance of hedonic social networking websites. *International Journal of Information Management*, 36(4), 503-514.
- Naylor, R. W. (2017). Online privacy and security risks for social media users. *International Journal of Information Security and Privacy (IJISP)*, 11(2), 1-11.
- Wang, Z., Chen, Y., & Liang, Y. (2018). Social connectedness and WeChat use among college students in China: The mediating role of social capital. *International Journal of Mobile Communications*, 16(2), 171-186.
- Xu, H., & Teo, H. H. (2017). Alleviating privacy concerns in mobile social networking: A critical review and future research directions. *Journal of the Association for Information Science and Technology*, 68(3), 588-600.
- Zhang, X., Ye, Q., & Law, R. (2017). The impact of social media conversations on consumer brand choices. *International Journal of Hospitality Management*, 60, 1-10.