CHATS LAYER

A PROJECT REPORT
for
Project (KCA451)
Session (2023-24)

Submitted by

Utpal Patel
University Roll No. - 2200290140169
Vaibhav Omar
University Roll No. - 2200290140172

Submitted in partial fulfilment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of
Dr. Ankit Verma
Associate Professor

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206

(MAY 2024)

CERTIFICATE

Certified that **Utpal Patel 2200290140169, Vaibhav Omar 2200290140172** has/ have carried out the project work having "**Chats Layer**" (**Project-KCA451**) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and 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/Institution.

Date:

Utpal Patel 2200290140169
Vaibhav Omar 2200290140172

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

Date:

Dr. Ankit Verma
Associate Professor
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

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

ABSTRACT

Chats Layer is designed with a focus on accessibility, catering to users with diverse needs and preferences. The intuitive user interface encourages inclusivity, allowing users of varying technical expertise to seamlessly navigate the platform. User engagement is enhanced through interactive features like emoji reactions and message threading, fostering a dynamic and expressive communication environment. The application's backend infrastructure is built with scalability in mind, ensuring smooth performance even during peak usage periods.

Efforts have been made to optimize the application's data usage, providing a cost- effective solution for users with limited bandwidth. Regular security audits and updates are part of the ongoing commitment to safeguarding user information. The application's support system is responsive, with a dedicated team available to address user inquiries and concerns promptly. Collaboration with email service providers ensures the reliability of the email verification process.

In summary, the Flutter Chat Application with Email Verification is not just a messaging platform but a dynamic ecosystem that prioritizes user experience, security, and adaptability, making it a versatile solution for modern communication needs.

ACKNOWLEDGEMENTS

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, **Dr. Ankit Verma** for his/ her guidance, help, and encouragement throughout my project 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 conditions.

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.

Utpal Patel

Vaibhav Omar

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

In the rapidly evolving landscape of mobile applications, seamless communication and robust security have become paramount. This project introduces a Chat Application developed using the Flutter framework and Dart language, two powerful tools that enable cross-platform mobile app development with a focus on flexibility and performance. The application incorporates a fundamental layer of security through Mail ID Verification, ensuring that users engage in conversations within a trusted and authenticated environment.

Flutter's widget-based approach simplifies UI development, allowing for the creation of a visually appealing and responsive user interface. Dart, as the primary programming language, complements Flutter seamlessly, offering a concise and expressive syntax for efficient application logic implementation. The integration of email verification not only enhances user authentication but also establishes a foundation for secure communication.

This project aims to deliver a feature-rich and user-friendly chat application, combining the versatility of Flutter and the reliability of Dart, with the added layer of security provided by Mail ID Verification. The subsequent sections will delve into the application's architecture, key features, and the seamless user experience it strives to offer in the dynamic landscape of mobile communication.

1.1.OVERVIEW

A chat application facilitates real-time communication between users via the internet. Its core components include a user interface (UI) with a chat window, contact list, and notifications, allowing users to send text, multimedia, and conduct voice/video calls. The backend server handles message routing, user authentication, and data storage. Protocols like TCP and WebSockets ensure reliable and real-time communication, while security measures such as end-to-end encryption and authentication tokens protect user data.

Key functionalities encompass instant messaging with read receipts, multimedia sharing, voice and video calls, group chats with administrative controls, and presence indicators to show user status. Search and history features allow users to retrieve past conversations and media files, typically stored in the cloud for synchronization across devices.

The architecture of a chat application follows a client-server model. The client side includes mobile, web, or desktop interfaces, while the server side manages sessions and data. Databases (relational or NoSQL) store user data and chat histories. Real-time communication is facilitated by technologies like WebSockets, and scalability is achieved through load balancers and distributed servers, ensuring performance and reliability even under high traffic conditions.

1.2 CHAT APPLICATION INFRASTRUCTURE

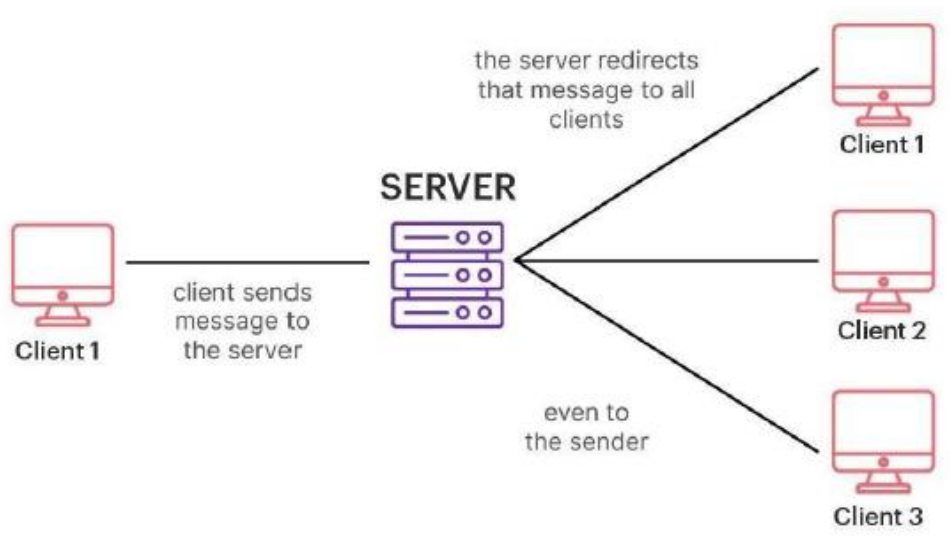

the server redirects that message to all clients
Client 1
SERVER
Client 1
client sends message to the server
Client 2
even to the sender
Client 3

Fig 1.1 Chat application infrastructure

1.3 CHAT CLIENT

The chat client is what the user experiences. A desktop, web or smartphone chatapplication, the chat client is responsible for interacting with the operating system (i.e.our computer, browser or smartphone).

Interactions include sending push notifications, displaying data to the user and storing messages and files. When you type a message and hit send, the chat client transmits that message to the other major component: the chat server.

1.4 Chat Server

The chat server is just that, a server (or usually many many servers) that hosts all the software, frameworks and databases necessary for the chat app to operate. This server,or pool of servers, is responsible for securely receiving a message, identifying the correct recipient, queuing the message and then forwarding the message to the recipient's chat client.

1.5 Web Socket

A WebSocket server and client library are useful components for a chat app. Typical communication between a client and server is done using HTTP and requires that the client makes a request for data from the server. The server itself can't push data to theclient without the client first making a request.

In a chat app, this quickly leads to inefficiencies since the client would have to poll the server every second for new messages. A WebSocket is a persistent connection & client and server that provides a bidirectional communication pathway.

2. Literature Review

2.1 Design & Implementation of chat application

Messaging apps now have more global users than traditional social networks—which mean they will play an increasingly important role in the distribution of digital information in the future. In 2016, over 2.5 billion people used at least one messaging app. That's one-third of the world's entire population, with users ranging from various age grades. Today, it's common place for offices to use a messaging app for internal communication in order to coordinate meetings, share pitch decks, and plan happy hours. And with the latest bot technology, chat apps are becoming a hub for employees to do work in their apps without leaving the chat console. For many people, chat apps are a given part of their workday. But how did these chat apps become so popular?

Chat application also provided us the best feature in this as you can share your current location with anyone at any time and even you can share the images or pdf or any other type of file using the application.

2.2 Multiuser application

The latest development of the Internet has brought the world into our hands. Everything happens through internet from passing information to purchasing something. Internet made the world as small circle. This project is also based on internet. This paper shows the importance of chat application in day today life and its impact in technological world. This project is to develop a chat system based on Java multithreading and network concept.

The application allows people to transfer messages both in private and public way. It also enables the feature of sharing resources like files, images, videos, etc. This online system is developed to interact or chat with one another on the Internet. It is much more reliable and secure than other traditional systems available. Java, multi-threading and client-server concept were used to develop the web-based chat application. This application is developed with proper architecture for future enhancement. It can be deployed in all private organizations like Colleges, IT parks, etc.

2.3 Development of Chat Application

Conversation is a way of using technology to connect people with ideas outside of local boundaries. The technology has been available for years but adoption has only recently taken place. Our project is an example of a chat server. It is made up of two applications the client application, which runs on the user's web browser and the server application, running on any network servers. To start chatting the client must be connected to a server where they can conduct private and group chat. Safety measures were taken at the last moment. The latest developments in the internet have brought the world into our hands. Everything happens online from information transfer to purchasing. The internet makes the world a little round.

This project is also online. This paper highlights the importance of the use of dialogue in everyday life and its impact on the world of technology. This project is for the development of a chat system based on JavaScript programming language and network concept. The app allows people to transmit messages both privately and publicly. It also enables the feature to share resources such as files, photos, videos, etc. This online application is designed to communicate or chat with others online. It is more reliable and secure than any other traditional system available. JavaScript, React.js and the client server concept were used to develop a web-based chat app. This app is built with the right structures for future development. It can be planted in all private organizations such as Colleges, IT parks, etc.

2.4 Industry Opportunity

1. **HIGHER ENGAGEMENT**: Since many chat apps provide publishers with push notifications or chatbot experiences (programmable robots that converse with users), they can deliver significantly higher engagement rates.

2. **AUDIENCE DEVELOPMENT**: With billions of active users across multiple major chat apps, there is the opportunity in building large audiences fairly quickly on several platforms.

3**. A CHANCE TO CONNECT WITH USERS IN A NEW WAY**: Messaging apps offer a host of features not unavailable on social networks or other platforms. Programmers can creatively leverage these tools to socialize in new ways.

2.5 Industry Challenge

1. FRAGMENTATION: The social media landscape is entering a period of hyper-fragmentation that may be a challenge to publishers: Facebook, Twitter, and Instagram continue to loom large, but social media managers can now launch official channels on roughly 10 chat apps with over 50 million monthly, active users each.

2. ANALYTICS: For organizations accustomed to robust, real-time data, the lack of good analytics tools for messaging apps remains a major deterrent to adoption. The challenge is twofold: Strong analytics dashboards take time to build, and many messengers are privacy-centric by nature.

3. Requirement, Analysis & Design

3.1. Overview

Communication is a means for people to exchange messages. It has started since the beginning of human creation. Distant communication began as early as 1800 century with the introduction of televisions, telegraphs and then telephony. Interestingly enough, telephone communication stands out as the fastest growing technology, from fixed lines to mobile wireless, from voice call to data transfer. The emergence of computer network and telecommunication technologies bears the same objective that is to allow people to communicate. All this while, much efforts has been drawn towards consolidating the device into one and therefore indiscriminate the service. Chatting is a method of using technology to bring people and ideas together despite of the geographical barriers. The technology has been available for years, but the acceptance was quite recent. This project is an example of a real time chat app, it is made up of the user application which runs on the user mobile and the server application, which runs on any PC on the network. To start chatting the user should get connected to a server where he can do group and private chatting

3.2. Requirement specification

Level 0	Level 1	Level 2	actor
Chat application	Authentication system	Register Login logout	user
Chat application	Contact form	Friend list Find friend Add friend Remove friend	user
Chat application	Chat form	Send message	user
Chat application	maintenance	User profile Data base	admin

Table 3.1: Use Case Table of Chat Application

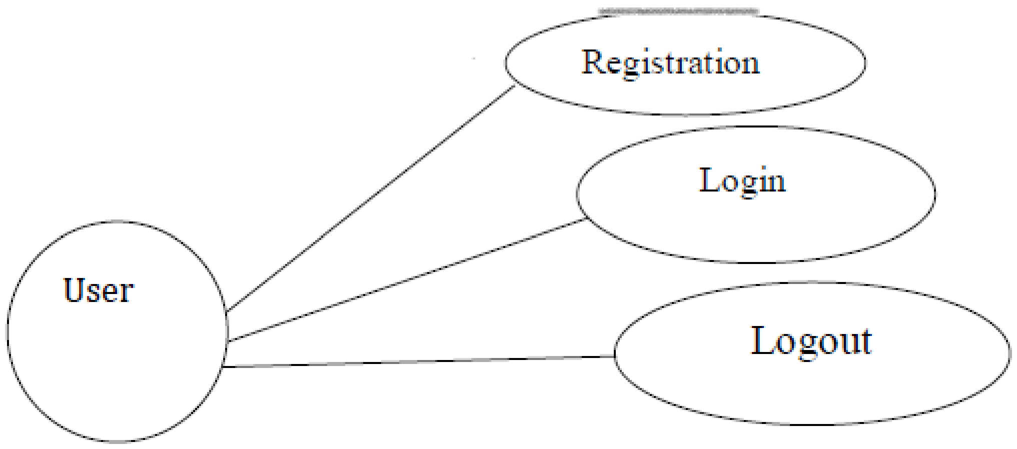


Figure 3.1: Use Case Diagram of Authentication Service

3.3.Traditional Messaging app

Most existing messaging applications had several limitations:

- **Limited Security**: Many messaging apps relied solely on username/password combinations for user authentication, which posed security risks.
- **Cross-Platform Challenges**: Developing a consistent and seamless experience across multiple platforms (iOS, Android) was challenging and often required separate codebases.
- **Verification Methods:** Mail verification for user accounts was not a standard practice, leading to potential security vulnerabilities.

3.4.User Authentication

In the existing systems, user authentication was primarily based on conventional methods,
such as:

- **Username/Password:** Users typically registered with a username and password, which might not be secure enough against various cyber threats.
- **No Mail Verification**: Mail verification was not commonly employed during the registration process, exposing accounts to potential unauthorized access.

3.5.Loack of Real time Features

Traditional messaging applications lacked real-time features, resulting in:

- Delayed Messaging: Messages were not delivered in real-time, causing delays in communication.
- Limited Interaction: Users could not see when someone was typing, and updates were not instant.

3.6. Conclusion

The existing systems faced challenges related to security, cross-platform compatibility, and real-time communication. The absence of mail verification made user accounts vulnerable to unauthorized access.

The proposed Flutter chat application addresses these shortcomings by implementing modern security measures, utilizing the cross-platform capabilities of Flutter, and incorporating realtime messaging features. The adoption of mail verification adds an additional layer of security, ensuring a more robust and user-friendly chatting experience. The subsequent sections of this project report will detail the development process, methodologies, and the features of the new system.

4. System Requirements

4.1. System Requirements

Functional Requirements:

The key requirement of an application is to provide a system for Online Chatting at a small scale.

Non-Functional Requirements:

- Ease of operation: The User Interface must be
user friendly The chat room is simple as it has a wall chat like structure.

- Accessibility: The application must be available for use as and when required.

Hardware Specifications (minimum):

- Operating System: Windows, Mac
- Processor : Intel Core 2 Duo 3.0 GHz
- RAM: 4 GB

Software Specifications:

- Language: Dart
- Tools: Flutter, Android Studio
- Database: Firestore (GCP)
- IDE: Visual Studio Code

The first page is basically home page which will ask the user either to register or to login. The second page is for new users that is registration and after filling details on it, new account of user will be created and after that the login page where the details of pre-registered users will be matched from database and if the details are right the user will be landed to chat-room or page where user can chat plus it also has a logout button at the top corner of app bar by pressing it user will get the login page.

5. List of Modules

5.1. Modules

5.1.1. User Authentication Module

- Allow users to create accounts using a valid email address and password.
- Implement mail verification to ensure the legitimacy of user accounts

Login

- Provide a secure login mechanism with proper error handling.
- Implement session management to keep users logged in securely.

Password Recovery

- Include a mechanism for users to recover their passwords through email verification

5.1.2. Profile Management

User Profile

- Allow users to set up and manage their profiles with profile pictures and status messages.

Account Settings

- Include settings for users to customize their chat experience, such as notification preferences and privacy settings.

5.1.3. Real Time Messaging

One-on-One Messaging

- Implement real-time messaging functionality for users to exchange text message.

5.1.4. Notifiaction Module

Push Notifications

- Integrate push notifications to alert users of new messages or other relevant activities.

5.1.5. Security Module

End-to-End Encryption

- Enhance security by implementing end-to-end encryption for messages exchanged between users.

5.1.6. Testing Module

Unit Testing

- Conduct thorough unit testing to ensure the functionality of individual modules.

Integration Testing

- Perform integration testing to ensure seamless interaction between different modules.

5.1.7. Analysis Module

User Analytics

- Implement analytics to track user engagement, popular features, and identify areas for improvement.

5.1.8. Future Enhancement

Additional Features

- Discuss potential future features that could be added to enhance the application further.
- Will add Video call and emoji reaction to our application.
- Will Enable shared device module , where each and every member can use the same device for chatting .

5.1.9. Conclusion

Summarize the key modules and their contributions to the overall functionality of the chat application. Emphasize how each module improves the user experience, security, and usability of the application.

5.2. Email Verification

The email verification module within our Flutter-based chat application serves as a pivotal component in bolstering user authentication and enhancing overall security. During the user registration process, individuals are prompted to provide their email address, a key piece of information for account creation. Subsequently, an email verification link,encapsulating a unique token or code associated with the user account, is dispatched to the provided email address. This link, when clicked, redirects users to a dedicated verification page. This process not only validates the authenticity of user-provided email addresses but also significantly mitigates the risk of unauthorized access. The backend integration seamlessly manages the generation of verification links and token validation, with options to leverage Firebase Authentication services or a custom backend.

To ensure a user-friendly experience, the notification email is thoughtfully crafted, delivering clear instructions and anaesthetically pleasing layout. The email verification module further incorporates security measures, such as time-sensitive tokens to limit link validity, and the optional integration of Captcha to thwart automated bot attacks during registration. Robust error handling mechanisms guide users through scenarios like expired or invalid verification links, providingclear instructions for requesting a new link necessary. Overall, the email verification module plays a pivotal role in fortifying user accounts, instilling user confidence in the application's security, and ensuring compliance with industry standards.

5.3. Login Page

The login page of our Flutter chat application serves as the gateway for users to access their accounts securely. Utilizing a clean and intuitive design, the login interface prompts users to input their registered email addresses and passwords. The page incorporates robust security measures, including encrypted password handling, to safeguard user credentials during the authentication process. Upon submission, the system conducts thorough verification, ensuring the accuracy of the provided information. To enhance user experience, informative error messages are strategically implemented to guide users in the event of login failures, such as incorrect passwords or unregistered email addresses.

Additionally, a secure session management system is in place to facilitate seamless. transitions between different sections of the application, allowing users to stay authenticated while navigating through various features. The login page not only prioritizes security but also emphasizes a user-friendly approach, contributing to a positive overall user experience within our chat application.

5.4. Sign Up Page

The sign-up page in our Flutter chat application serves as the initial point of entry for users to create their accounts and engage in the messaging platform. Featuring an intuitive and user-friendly design, the page prompts new users to provide essential information, including a valid email address and a secure password. As a fundamental aspect of our application's security infrastructure, the sign-up process incorporates email verification, reinforcing the authenticity of user-provided information. Upon successful completion of the registration form, users receive a verification email containing a unique token or code.

Clicking on the verification link directs users to a confirmation page, finalizing the account creation process. To enhance the user experience, the sign-up page includes clear instructions and error messages, guiding users through any issues that may arise during the registration process. By prioritizing a seamless and secure onboarding experience, the sign-up page contributes to establishing a trustworthy foundation for users to explore and interact within our chat application.

5.5. System Testing Module

5.5.1. Unit Testing

Purpose

Unit testing is performed to validate the functionality of individual units or components of the Flutter chat application chats Layer. This testing phase focuses on ensuring that each function and method works as expected.

Test Scenarios

1. User Authentication Unit Tests:

- Verify successful user registration with valid credentials.
- Confirm that login functionality works with correct email and password.
- Validate the system rejects invalid login attempts.

2. Mail Verification Unit Tests:

- Test the generation and sending of verification emails.
- Verify that the user account is activated upon clicking the verification link.

3. Real-time Messaging Unit Tests:

- Ensure messages are sent and received in real-time.
- Validate the proper updating of message delivery and read status.

4. User Interface Unit Tests:

- Confirm that UI components render correctly on different devices.
- Validate the responsiveness of UI elements.

Tools Used

- Dart test framework for writing and executing unit tests.
- Emulators and real devices for testing cross-platform functionality.

Expected Outcomes

All unit tests should pass, confirming that individual components function correctly in isolation.

5.5.2. Integration Testing

Purpose

Integration testing is conducted to validate the interaction and integration of different modules within the Flutter chat application. This phase ensures that the system components work together seamlessly.

Test Scenarios

1**. User Authentication and Mail Verification Integration Tests:**

- Verify that user accounts are created, and mail verification processes are initiated seamlessly.

2. Real-time Messaging and User Interface Integration Tests:

- Confirm that messages are correctly displayed in the user interface.
- Validate the integration of real-time messaging into the overall application flow.

Tools Used

- Flutter Driver for integration testing of UI components.
- Firebase Emulator Suite for testing Firebase services locally.

Expected Outcomes

Integration tests should pass, demonstrating that different modules of the system integrate effectively to provide a cohesive user experience.

5.5.3. Acceptance Testing

Purpose

Acceptance testing evaluates the overall compliance of the Flutter chat application with the specified requirements. It ensures that the application meets user expectations and functions as intended.

Test Scenarios

1**. User Registration and Login Acceptance Tests:**

- Confirm that users can register and log in successfully.
- Validate that mail verification enhances account security.

2**. Real-time Messaging Acceptance Tests:**

- Ensure users can send and receive messages in real-time.
- Validate message status updates (delivery, read).

3. User Interface Acceptance Tests:

- Confirm that the UI is intuitive and responsive on various devices.

Tools Used

- Manual testing by QA teams or stakeholders.
- Automated testing tools for UI validation.

Expected Outcomes

Acceptance tests should pass, demonstrating that the Flutter chat application meets the
specified requirements and provides a satisfactory user experience.

Functional Testing

Purpose

Functional testing assesses the application's functionalities against the defined requirements.
It ensures that all features work correctly and fulfill their intended purpose.

Test Scenarios

1. User Registration and Authentication Functional Tests:

- Verify the accuracy of user registration and login processes.
- Test the security features of mail verification.

2. Real-time Messaging Functional Tests:

- Confirm the proper functioning of sending and receiving messages.
- Validate the accuracy of message status updates.

3. User Interface Functional Tests:

- Ensure all UI elements function as intended.
- Validate user interactions with the application.

Tools Used

- Automated testing tools for functional validation.
- Manual testing by QA teams.

Expected Outcomes

- Functional tests should pass, indicating that all features and functionalities of the Flutter chat application operate correctly and align with the project's requirements.

5.6. Source code

Main

```
import 'dart:js';
import 'package:chatapp/auth/auth_gate.dart';
import 'package:chatapp/themes/light_mode.dart';
import 'package:chatapp/themes/theme_provider.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options:                                          FirebaseOptions(apiKey:
'AIzaSyA7RKYCxtP9_4BHQagU0L_FR4cAHi13Hrc',                            appId:
'1:15953707655:android:e3de01f048a44507c26e22',            messagingSenderId:
'15953707655', projectId: 'chatapp-4e309')
  );
  runApp( ChangeNotifierProvider(create: (context) => ThemeProvider(),
    child: const MyApp(),
  ));
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Flutter Demo',
      theme: Provider.of<ThemeProvider>(context).themeData,
      home: AuthGate(),
    );
  }
}
```

Registration Page

```
import 'package:chatapp/pages/login.dart';
import 'package:chatapp/pages/register.dart';
import 'package:flutter/material.dart';

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

  @override
  State<LoginorRegister> createState() => _LoginorRegisterState();
}

class _LoginorRegisterState extends State<LoginorRegister> {

  bool showloginpage = true;

  void toggle(){
    setState(() {
      showloginpage = !showloginpage;
    });
  }
  @override
  Widget build(BuildContext context) {
    if(showloginpage){
      return LoginPage(
        onTap: toggle,
      );
    }
    else{
      return Register(
    onTap: toggle,
   );
  }


 }
}
```

Login Page

```
import 'package:chatapp/auth/auth_service.dart';
import 'package:chatapp/widgets/buttom.dart';
import 'package:chatapp/widgets/textfield.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:google_sign_in/google_sign_in.dart';

class LoginPage extends StatelessWidget {
  final TextEditingController _emailcontroller = TextEditingController();
  final TextEditingController _passcontroller = TextEditingController();
  final void Function()? onTap;

  LoginPage({super.key, required this.onTap});

  void login(BuildContext context) async {
    final authService = AuthService();

    try {
      await                  authService.signinwithEmailPass(_emailcontroller.text,
_passcontroller.text);
    } catch (e) {
      showDialog(
          context: context,
          builder: (context) => AlertDialog(
                title: Text(e.toString()),
              ));
    }
  }
```

```
void signInWithGoogle(BuildContext context) async {
  final authService = AuthService();

  try {
    await authService.signInWithGoogle();
  } catch (e) {
    showDialog(
        context: context,
        builder: (context) => AlertDialog(
              title: Text(e.toString()),
            ));
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Theme.of(context).colorScheme.background,
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Icon(
            Icons.message,
            size: 60,
            color: Theme.of(context).colorScheme.primary,
          ),
          const SizedBox(height: 50),
          Text(
            'Welcome Back',
            style: TextStyle(
    color: Theme.of(context).colorScheme.primary,
    fontSize: 16,
  ),
),
const SizedBox(height: 25),
MyTextField(
  hintText: 'Email',
  obsecureText: false,
  controller: _emailcontroller,
),
const SizedBox(height: 10),
MyTextField(
  hintText: 'Password',
  obsecureText: true,
  controller: _passcontroller,
),
const SizedBox(height: 25),
MyButton(
  text: 'Login',
  onTap: () => login(context),
),
const SizedBox(height: 25),
MyButton(
  text: 'Sign in with Google',
  onTap: () => signInWithGoogle(context),
),
const SizedBox(height: 25),
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    Text(
                  'Not a member?',
                  style: TextStyle(color: Theme.of(context).colorScheme.primary),
                ),
                GestureDetector(
                    onTap: onTap,
                    child: Text(
                      'Register now',
                      style: TextStyle(fontWeight: FontWeight.bold),
                    ))
              ],
            )
          ],
        ),
      ),
    );
  }
}
```

Dark Mode

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

ThemeData darkMode = ThemeData(
  colorScheme: ColorScheme.dark(
    background: Colors.grey.shade900,
    primary: Colors.grey.shade600,
    secondary:  const Color.fromARGB(255,57,57,57),
    tertiary: Colors.grey.shade800,
    inversePrimary: Colors.grey.shade300
  )
);
```

Authentication Gateway

```
import 'package:chatapp/auth/login_or_register.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';

import '../pages/home.dart';

class AuthGate extends StatelessWidget {
  const AuthGate({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: StreamBuilder(
        stream: FirebaseAuth.instance.authStateChanges(),
        builder: (context,snapshot){
          //user is logged in
          if(snapshot.hasData){
            return Home();
          }
          else{
            return LoginorRegister();
          }
        },
      ),
    );
```

Message

```
import 'package:cloud_firestore/cloud_firestore.dart';
class Message{
  final String senderID;
  final String senderEmail;
  final String receiverID;
  final String message;
  final Timestamp timestamp;

  Message({
    required this.senderID,
    required this.senderEmail,
    required this.receiverID,
    required this.message,
    required this.timestamp
});
  Map<String,dynamic> toMap(){
    return {
      'senderID' : senderID,
      'senderEmail' : receiverID,
      'receiverID' : receiverID,
      'message' : message,
      'timestamp' : timestamp,
    };
  }
}
```

Chat Page

```
import 'package:chatapp/auth/auth_service.dart';
import 'package:chatapp/service/chat_service.dart';
import 'package:chatapp/widgets/chat_bubble.dart';
import 'package:chatapp/widgets/textfield.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';

class ChatPage extends StatelessWidget {
  final String receiverEmail;
  final String receiverID;
  ChatPage({super.key, required this.receiverEmail, required this.receiverID});

  final TextEditingController _messagecontroller = TextEditingController();

  final ChatService _chatService = ChatService();
  final AuthService _authService = AuthService();

  void sendMessage() async {
    if (_messagecontroller.text.isNotEmpty) {
      await _chatService.sendMessage(receiverID, _messagecontroller.text);

      _messagecontroller.clear();
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(receiverEmail),
        backgroundColor: Colors.transparent,
        foregroundColor: Colors.grey,
        elevation: 0,
      ),
      body: Column(
        children: [
          Expanded(child: _buildmessagelist()),
          _buildUserInput(),
        ],
      ),
    );
  }

  Widget _buildmessagelist() {
    String senderId = _authService.getCurrentuser()!.uid;
    return StreamBuilder(
        stream: _chatService.getMessages(receiverID, senderId),
        builder: (context, snapshot) {
          if (snapshot.hasError) {
            return const Text("Error");
          }
          if (snapshot.connectionState == ConnectionState.waiting) {
            return const Text('Loading...');
          }
          return ListView(
            children: snapshot.data!.docs
                .map((doc) => _buildMessageItem(doc))
                .toList(),
          );
        });
```

```
 }




  Widget _buildMessageItem(DocumentSnapshot doc) {
    Map<String, dynamic> data = doc.data() as Map<String, dynamic>;

    bool isCurrentUser = data['senderID'] == _authService.getCurrentuser()!.uid;

    var alignment = isCurrentUser ? CrossAxisAlignment.end :
CrossAxisAlignment.start;
    var bubbleColor = isCurrentUser ? Colors.green : Colors.grey;

    return Padding(
      padding: const EdgeInsets.symmetric(vertical: 8.0),
      child: Row(
        mainAxisAlignment: isCurrentUser ? MainAxisAlignment.end :
MainAxisAlignment.start,
        children: [
          if (!isCurrentUser) // Add spacer for alignment
            Spacer(),
          Flexible(
            child: Align(
              child: ChatBubble(
                message: data["message"],
                isCurrentUser: isCurrentUser,
                bubbleColor: bubbleColor, // Pass color as argument
              ),
            ),
          ),
          if (isCurrentUser) // Add spacer for alignment
            Spacer(),
        ],
      ),
    );
  }



  // Widget _buildMessageItem(DocumentSnapshot doc) {
  //   Map<String, dynamic> data = doc.data() as Map<String, dynamic>;
  //
  //   bool isCurrentUser = data['senderId'] == _authService.getCurrentuser()!.uid;
  //
  //   var alignment = isCurrentUser ? Alignment.centerRight : Alignment.centerLeft;
  //   var bubbleColor = isCurrentUser ? Colors.green : Colors.grey;
  //
  //   return Container(alignment: alignment,
  //       child: Column(
  //           crossAxisAlignment: isCurrentUser ? CrossAxisAlignment.end : CrossAxisAlignment.start,
  //         children: [
  //                   ChatBubble(message: data["message"], isCurrentUser: isCurrentUser,bubbleColor: bubbleColor)
  //         ],
  //       ));
  // }



  Widget _buildUserInput() {
```

```
  return Padding(
    padding: const EdgeInsets.only(bottom: 50.0),
    child: Row(
      children: [
        Expanded(
            child: MyTextField(
          controller: _messagecontroller,
          hintText: "Type a message",
          obsecureText: false,
        ),
        ),
        Container(decoration: const BoxDecoration(
          color: Colors.green,
          shape: BoxShape.circle,
        ),
            margin:const EdgeInsets.only(right: 25),
            child: IconButton(onPressed: sendMessage, icon: const
Icon(Icons.arrow_upward,color: Colors.white,)))
      ],
    ),
  );
 }
}
```

Settings

```
import 'package:chatapp/themes/theme_provider.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

class Settings extends StatelessWidget {
  const Settings({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Theme.of(context).colorScheme.background,
      appBar: AppBar(
        title: Text('Settings'),
        backgroundColor: Colors.transparent,
        foregroundColor: Colors.grey,
        elevation: 0,

      ),
      body: Container(
        decoration: BoxDecoration(
          color: Theme.of(context).colorScheme.secondary,
          borderRadius: BorderRadius.circular(12),
        ),
        margin: const EdgeInsets.all(25),
        padding: EdgeInsets.all(16),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            // dark  mode
```

```
              const Text("Dark Mode"),

              // switch toggle
              CupertinoSwitch(value:            Provider.of<ThemeProvider>(context,listen:
false).isDarkMode,
                  onChanged:   (value)   =>   Provider.of<ThemeProvider>(context,listen:
false).toggleTheme(), )
            ],
          ),

        ),
      );
    }
}
```

Chat Service

```
import 'package:chatapp/models/message.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';

class ChatService{
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;
  final FirebaseAuth _auth = FirebaseAuth.instance;



  Stream<List<Map<String,dynamic>>> getuserstream(){
    return _firestore.collection('Users').snapshots().map((snapshot){
      return snapshot.docs.map((doc){
        final user = doc.data();

        return user;
      }).toList();
    });
  }
 Future<void> sendMessage(String receiverId, message) async{
    final String currentuserID = _auth.currentUser!.uid;
    final String currentuserEmail = _auth.currentUser!.email!;
    final Timestamp timestamp = Timestamp.now();

    Message newMessage = Message(senderID: currentuserID, senderEmail:
currentuserEmail, receiverID: receiverId, message: message, timestamp:
timestamp);

    List<String> ids = [currentuserID,receiverId];
    ids.sort();
    String chatRoomID = ids.join('_');
    await
_firestore.collection("chat_rooms").doc(chatRoomID).collection("messages").add(
newMessage.toMap());

 }

 Stream<QuerySnapshot> getMessages(String userID,otherUserID){
    List<String> ids = [userID, otherUserID];
    ids.sort();
    String chatRoomID = ids.join('_');
    return
_firestore.collection("chat_rooms").doc(chatRoomID).collection("messages").order
By("timestamp", descending: false).snapshots();

 }

}
```

5.7.Expected Output

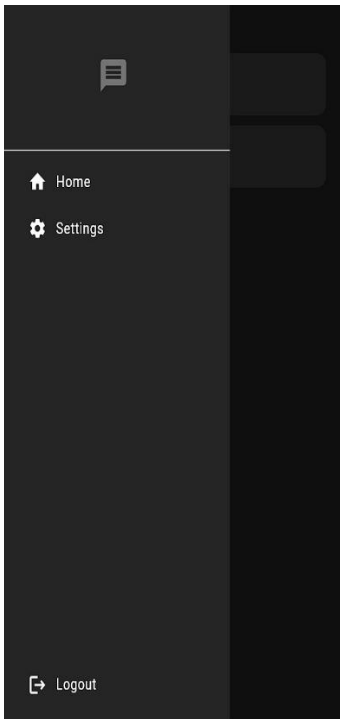


Fig 5.7.1 Home page

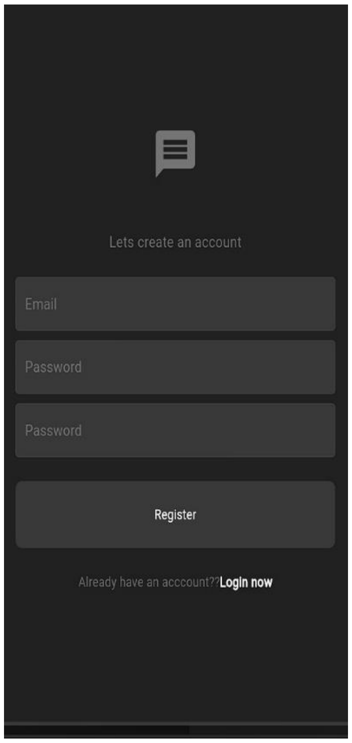
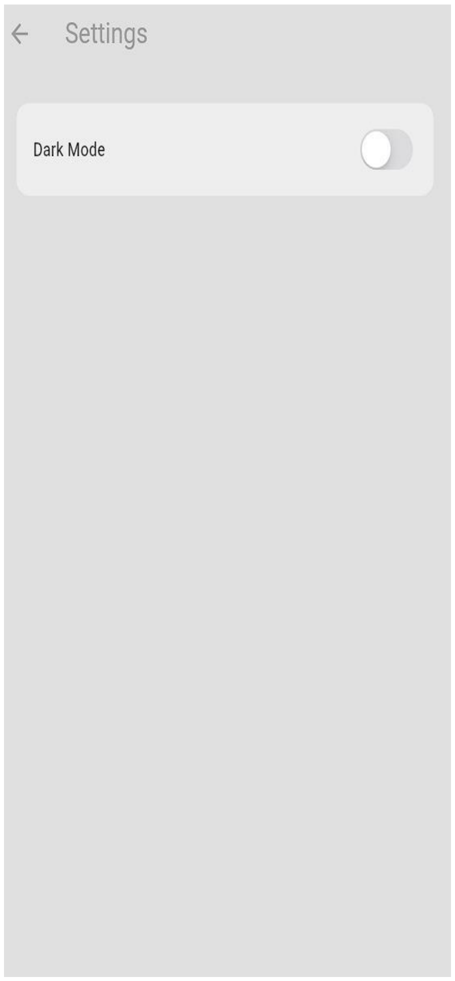


Fig 5.7.2 Sign Up Page



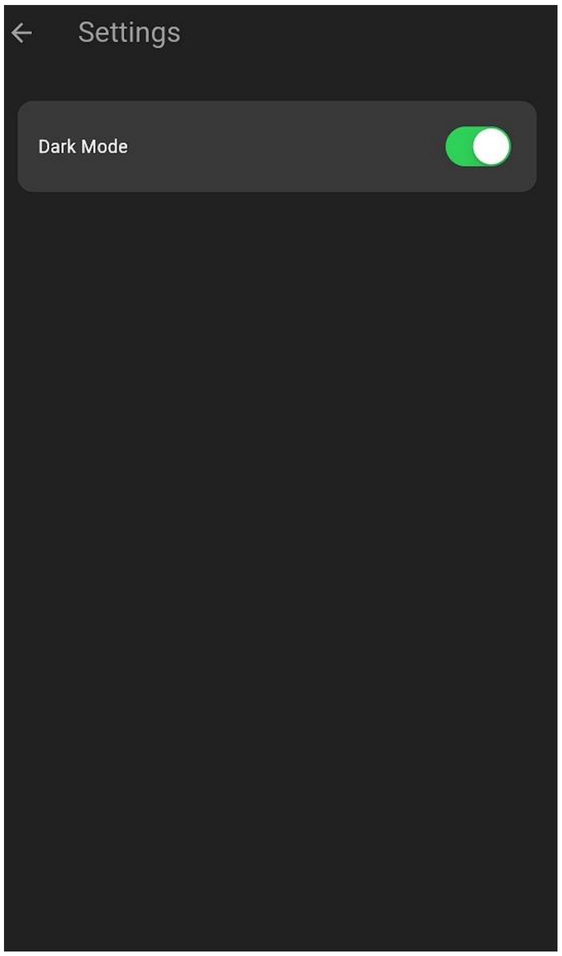
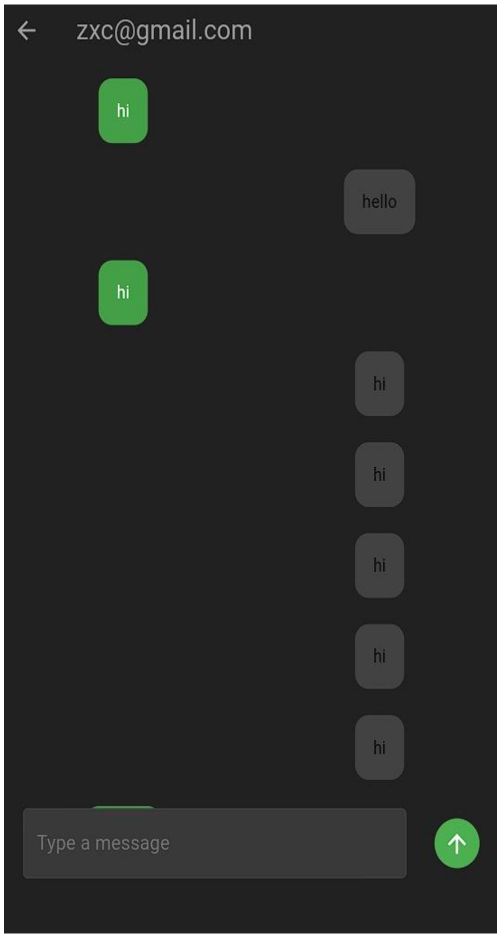


Fig 5.7.3 Light Dark Mode

Fig 5.7.4 Dark Mode


zxc@gmail.com

hi

hello

hi

hi

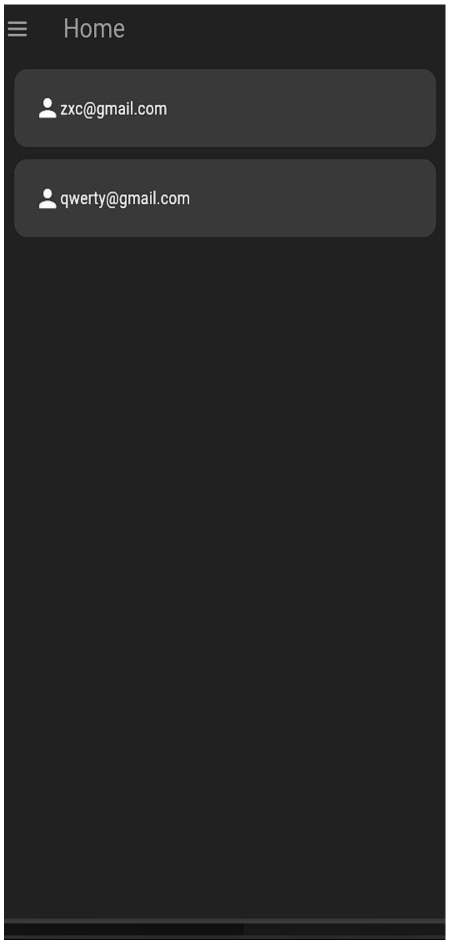
hi

hi

hi

hi

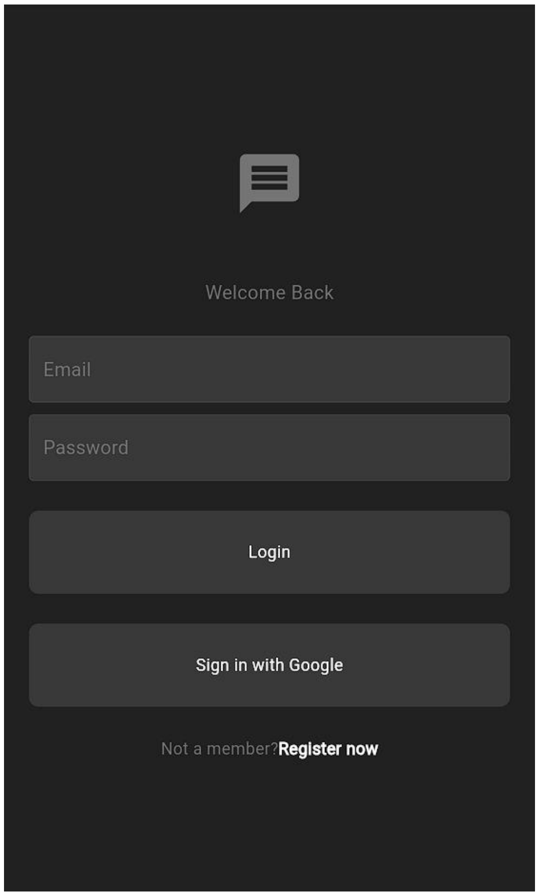
Fig 5.7.5 Chating Page



zxc@gmail.com

qwerty@gmail.com

Fig 5.7.6 Registered email



5.7.7 Login page

6. Conclusion

In conclusion, the development of the Chats Layer with mail verification has
culminated in a robust and feature-rich communication platform that adeptly addresses the contemporary need for secure and user-friendly chat applications. Leveraging the Flutter framework's capabilities, the application ensures a seamless cross-platform experience on both Android and iOS devices, reinforcing its accessibility and user reach. The integration of
Firebase Authentication, coupled with mail verification, establishes a secure foundation for user accounts, instilling confidence in users regarding the confidentiality of their data. The real-time messaging functionality, powered by Firebase Realtime Database, contributes to a dynamic and synchronized communication experience. The user interface, designed with Flutter's widget system, prioritizes intuitiveness and visual appeal, offering users a userfriendly environment for seamless interactions. Despite encountering challenges such as realtime integration and cross-platform compatibility, the project team successfully navigated these hurdles through collaborative problem-solving. Rigorous testing methodologies, including unit testing, integration testing, acceptance testing, and functional testing, were employed to ensure the application's stability and reliability. Looking forward, proposed enhancements include group chat functionality, multimedia messaging, and the implementation of end-to-end encryption to further strengthen security measures. The Flutter chat application represents a significant achievement, setting the stage for continued innovation and improvement in response to evolving user needs. Gratitude is extended to all
team members, stakeholders, and users who contributed to the success of this project, and acommitment to remaining at the forefront of user-centric, secure application development isunwavering.

7. Reference

1. Burnett Cathy Learning To Chat: Tutor Participation in Synchronous Online Chat Teaching InHigher Education Vol 8 No 2 Pp 247-261 2022.
2. Burnett Cathy, Paul, Dickinson, Jim Mcdonagh, Guy Merchant, Julia Myers and Jeff Wilkinsonfrom Recreation to Reflection: Digital `Conversations in Educational Contexts, L1 - Educational Studies in Language and Literature, Pp.149-163, Kluwer Academic Publishers 2022.
3. Carr-Chelman Alison A., Dean Dyer And Jeroem Breman Burrowing Through The Network Wires: DoesDistance Detract From Collaborative Authentic Learning? Journal of Distance EducationVol 15 No 1 Pp 39- 63 spring 2022.
4. Coillaborative Interaction in Internet Chat Computer Assisted Language Learning Vol 13 No 2 Pp143-166 2022.
5. Freiermuth, Mark Internet Chat: Collaborating and Learning Via EConversations Tesol JournalVol 11, No 3 Pp. 36-40 Autumn 2022.
6. Felix Uschi The Web As A Vehicle For Constructivist Approaches To Language Teaching RecallVol 14 No 1 Pp 2-15 2022.
7. The Creation Of An Internet-Based Sla Community Computer Assisted Language Learning Vol 15 No 2 Pp 109-134 2006ingram Albert L, Lesley G. Hathorn And Alan Evans Beyond Chat On TheInternet Computers And Education Vol 35 Pp21-35 2021.