# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**"Jnana Sangama", Belagavi-590 014**

**A Mini - Project Report**

On

# "GUARD TALK (Secure Platform for Communication)"

Submitted in partial fulfillment of the requirements for the **MINI PROJECT (BCD586)** course of the 5th semester

**Bachelor of Engineering**

*In*

**Computer Science & Engineering (DATA SCIENCE)**

Submitted by


**Ms. Bhoomika Gowda H G** (4AI22CD008)

**Ms. M M Bhavya** (4AI22CD031)

**Ms. Meghana N S** (4AI22CD034)

**Ms. Rishitha N** (4AI22CD043)


**Under the guidance of**

**Mrs. Shalini I S , B.E., M.Tech.**

Assistant Professor


**Department of CS&E (DATA SCIENCE)**

**Adichunchanagiri Institute of Technology**

**CHIKKAMAGALURU - 577102**


**2024-25**

# ADHICHUNCHANAGIRI INSTITUTE OF TECHNOLOGY

**Jyothinagar, Chikkamagaluru-577102**

**DEPARTMENT OF CS&E (DATA SCIENCE)**

# CERTIFICATE

This is to certify that the Mini project work entitled **"GUARD TALK(Secure Platform for Communication)** is a bonafied work carried out by **Ms. Bhoomika Gowda H G (4AI22CD008), Ms. M M BHAVYA (4AI22CD031), Ms. Meghana N S (4AI22CD034), Ms. Rishitha N (4AI22CD043)** in partial fulfillment for the **Mini Project (BCD586)** course of 5th semester Bachelor of Engineering in **Computer Science and Engineering (Data Science)** of the Visvesvaraya Technological University, Belagavi during the academic year **2024 -2025**. It is certified that all corrections and suggestions indicated for Internal Assessment have been incorporated in the report deposited in the department library. The Mini project report has been approved as it satisfies the academic requirements in respect of Project Work prescribed for the said Degree.

**Signature of the Guide**
**Ms. Shalini I S** B.E., M.Tech
**Assistant Professor**

**Signature of Coordinator**
**Mrs. Shilpa K V.** B.E., M.Tech
**Assistant Professor**

**Signature of the HOD**
**Dr. Adarsh M J** B.E., M.Tech., Ph.D
**Associate Professor and Head**

# ABSTRACT

The Real-Time One-to-One Chat Application is designed to facilitate secure and ephemeral communication between two users. This chat application allows users to exchange text-based messages in real-time, ensuring privacy and temporary storage of messages. The key features of this system are real-time message exchange, automatic message erasure upon user departure, and the ability to store important messages in a Google Spreadsheet.

The primary objective of this project is to provide users with a platform for secure communication, where messages are immediately erased once users navigate away from the chat room. This feature ensures that the system does not retain any chat history, thereby safeguarding user privacy. The chat messages are temporarily stored on the client-side during the chat session and are erased automatically when the user exits or refreshes the page. This eliminates the need for long-term storage on the server and offers a simple, temporary messaging environment.

In addition to the ephemeral nature of the messages, the system also allows users to flag certain messages as important. If a message is marked as important by either the sender or receiver, it is stored in a Google Spreadsheet for future reference. This feature ensures that valuable or critical information can be saved while other non-essential messages are discarded after the conversation ends. Google Sheets is used as a lightweight, easily accessible storage solution, enabling users to review important messages without overwhelming the system with large amounts of data.

This system provides an efficient and secure platform for users who need a private, temporary communication channel. It can be beneficial in environments where privacy is essential, such as customer support, confidential conversations, or time-sensitive communications where information needs to be recorded for future reference without long-term storage of all messages. The combination of ephemeral message deletion and selective message storage offers a unique approach to real-time communication with a focus on privacy and data control.

# ACKNOWLEDGEMENTS

We express our humble pranamas to his **holiness Divine Soul Parama Poojya Jagadguru Padmabushana Sri Sri Sri Dr.Balagangadharanatha Maha Swamiji and Parama Poojya Jagadguru Sri Sri Sri Dr. Nirmalanandanatha Maha Swamiji Pontiff, Sri Adichunchanagiri Maha Samsthana Matt and Sri Sri Gunanatha Swamiji**, Chikkamagaluru branch, Sringeri who have showered their blessings on us.

The completion of any project involves the efforts of many people. We have been lucky enough to have received a lot of help and support from all quarters during the making of this project, so with gratitude, we take this opportunity to acknowledge all those whose guidance and encouragement helped us emerge successful.

We express our gratitude **to Dr. C K Subbaraya**, Director, Adichunchanagiri Institute of Technology.

We express our sincere thanks to our beloved principal, **Dr. C T Jayadeva** for having supported us in our academic endeavors.

We are also indebted to **Dr. Adarsh M J**, HOD of CS&E (DATA SCIENCE) Department, for the facilities and support extended towards us.

We thank our project coordinator **Mrs. Shilpa K V ,**Asst. Professor, Department of CS&E (DATA SCIENCE), for her lively correspondence and assistance in carrying on with this project.

We are thankful to the resourceful guidance, timely assistance and graceful gesture of our guide **Mrs. Shalini I S** , Asst. Professor, Department of CS&E (DATA SCIENCE), who has helped us in every aspect of our project work.

We would be very pleased to express our heart full thanks to all the teaching and non-teaching staff of CS&E (DATA SCIENCE) Department and our friends who have rendered their help, motivation and support.

**BHOOMIKA GOWDA H G**

**M M BHAVYA**

**MEGHANA N S**

**RISHITHA N**

# CONTENTS

# List of Figures

# List of Snapshots

## Chapter 1

# Introduction

## 1.1 Background

- **Context**: The rapid advancement of digital communication has fundamentally altered the way people connect and interact. With the increasing reliance on technology, there is a growing demand for efficient and seamless communication tools. Chat applications have emerged as a popular solution, providing real-time text-based communication across various platforms. However, existing chat applications often lack innovative features, robust security measures, and a seamless user experience.
- **Problem**: In today's digital age, designing and developing effective communication system is essential for both personal and professional interactions. Many existing chat applications offer basic functionalities but often lack features that enhance user experience, security, and integration with other tools. Users frequently encounter issues such as lack of privacy. Hence this project aims to design and develop a chat application(GUARD TALK) which provides a platform to conversate through texts and it ensures some important information in the spreadsheet which helps to enhance information efficiently.
- **Opportunity**: Developing a new chat application called GUARD TALK that addresses these shortcomings, offers a superior user experience, enable instant messaging between one-to-one user and foster real-time interaction.

## 1.2 Problem Statement

- **Overview of the Problem**: The core problem revolves around designing and implementing a system that facilitates real-time communication between users, while ensuring a absolute, secure, and scalable experience. It can deliver a vigorous, secure, and scalable solution that provides a smooth and engaging user experience.
- **Specific Issues**:
  - Choosing a suitable datasheet: to store messages, considering factors like scalability, performance, and data integrity.
  - Message Expiration: Implementing a mechanism to automatically delete messages after a certain period or when the user exits the chat.

## 1.3 Objective of the System

- To create a strong, secure, and effective communication platform and provide a responsive user interface. Security is a top priority, with end-to-end encryption for messages and secure user authentication methods.

➢ **Key Goals**

- **User Authentication:** Secure user registration and login mechanism. Robust password hashing and salting for enhanced security.
- **User Experience:** The interface should be interactive and easy to use, providing positive user experience.
- **Security:** User data should be protected through secure authentication and data encryption.
- **Message Disappearance:** Automatic deletion of messages from the server and client-side upon user exit from the chat screen.
- **Reliability:** The application should be consistently available and reliable.

## 1.4 Significance of the System

- **Real-time Communication**: Enables instant messaging, fostering seamless and efficient interactions.
- **Selective Message Preservation:** Allows users to prioritize important messages, ensuring they're not lost in the conversation flow.
- **Organized Conversation History:** By storing crucial messages in a structured format like a Spreadsheet(Excel), users can easily refer them later.
- **Secure Storage:** Leveraging Spreadsheet's(Excel) robust security measures, sensitive information can be protected.
- **Remote Work:** Remote teams can maintain effective communication by selectively saving important messages.

## 1.5 Scope of the Project

- Sign up and login functionality (email/password).
- Consider two-factor authentication for enhanced security.
- User-friendly chat interface with starred messages.
- End-to-end encryption for messages.
- Automatically delete messages from the server and client-side after the specified duration.
- Encrypt messages to ensure privacy and security.

## 1.6 Methodology

- **Approach**: The system will be developed using a **web-based platform**, leveraging modern technologies such as **Python and its libraries**. The system will interact with a **spreadsheet**(like **EXCEL**) to store important/starred messages.

- **Iterative Development:** Break down the development process into smaller iterations or sprints. Each sprint focuses on delivering specific features, such as user registration, message sending and message receiving.
- **One-to-One Communication:** This implies a direct conversation between two users.
- **Disappearing Messages:** Messages should automatically vanish after a certain period of inactivity or when the user exits the chat screen.
- **Data Storage:** Excel for storing user's important or starred messages.

## 1.7 Target Audience

- **Privacy-conscious individuals**: People who value privacy and want to control the lifespan of their messages.
- **Businesses:** Companies that want to enhance internal communication especially for sensitive information or time-sensitive discussions.
- **Government agencies:** Government organizations that require secure and temporary communication channels for classified information or sensitive discussions.
- **Online communities:** Communities that value privacy and want to create a safe space for their members to communicate without fear of their messages being saved or shared.

## 1.8 Overview of the Report

This report is structured into several chapters that detail the development and design of the **Guard Talk(secure platform for communication)**. The following chapters include:

- **Chapter 2: System Design** – Describes the architecture and design of the system.
- **Chapter 3: Implementation** – Discusses the system's development and the technologies used.
- **Chapter 4: Testing and Validation** – Details of the testing process and results.
- **Chapter 5: Results and Discussions** – Presents and results obtained and discusses the limitations
- **Chapter 6: Conclusion and Future enhancement -** Summarizes the project and suggests future improvements.

## Chapter 2

# System Design

This chapter describes the technical design of GUARD TALK(secure platform for communication), this chapter explaining its architecture, components, and working.

## 2.1 System Architecture

- **High-Level Overview**:
  - Establish persistent connections between the server and clients to facilitate real- time message delivery.
  - Allow users to send text messages to each other.
  - Display incoming/outgoing messages in a chat window in real-time.
- **Architecture Diagram**:
  - Present a diagram showing the key components: frontend (UI), backend server, and data storage.
- **Components**:
  - **Frontend**: Flet is the frontend framework.
  - **Backend**: users_db.py file acts as a simple backend for storing user credentials.
  - It manages user authentication through sign-in and sign-up functionalities.
  - **Data Storage**: Excel is used to store the starred messages.

## 2.2 Module Design

- The system is divided into functional modules, each handling a specific task.
  - User Sign up and Sign in
  - Excel (Starred messages stored)
  - Overviewed

## 2.3 User Interface (UI) Design

- **Main Screens**
  - **Login Screen**: Users enter credentials to access the system.
  - **Chatting Screen**: Displays the conversation between two users.

## 2.4 Technology Stack

- **Frontend:** Flet is used as the frontend.
- **Backend:** user_db.py
- **Data Storage**: EXCEL for reliable data storage.

## 2.5 Use Case Diagram

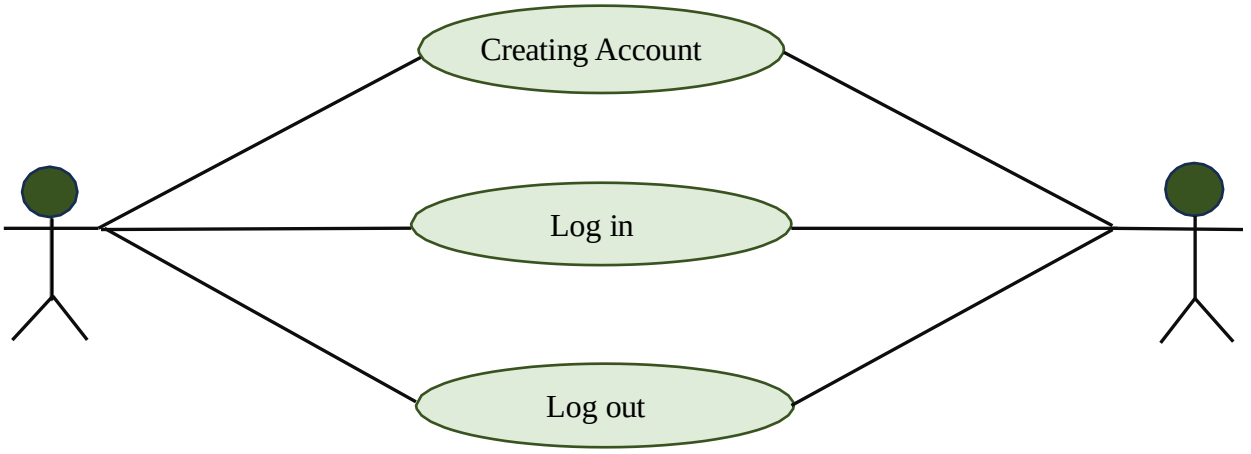


Fig 2.1 : Authentication System

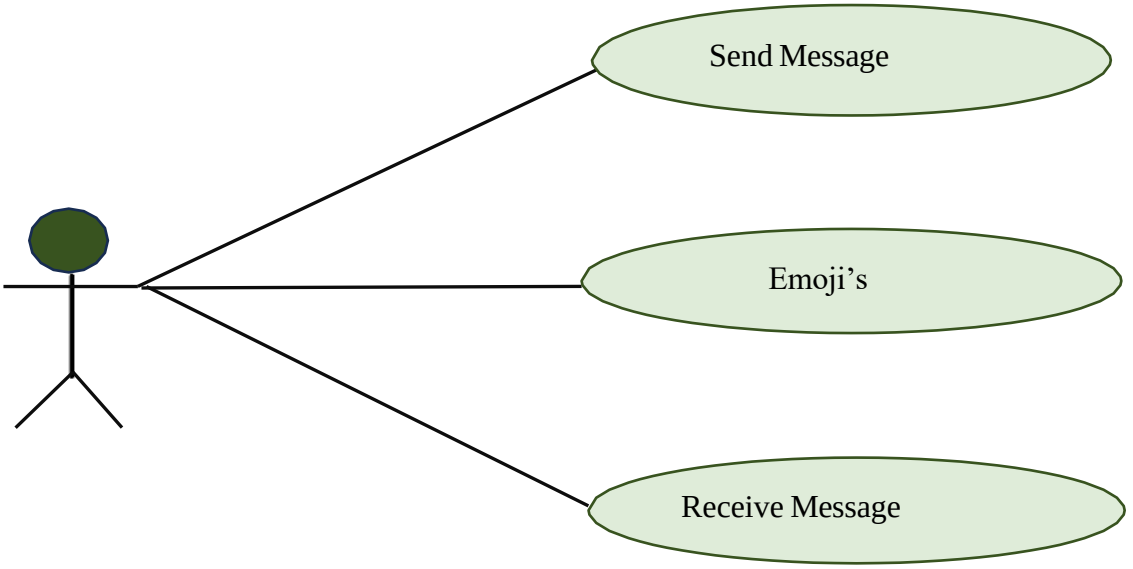


Fig 2.2 : Chat Form

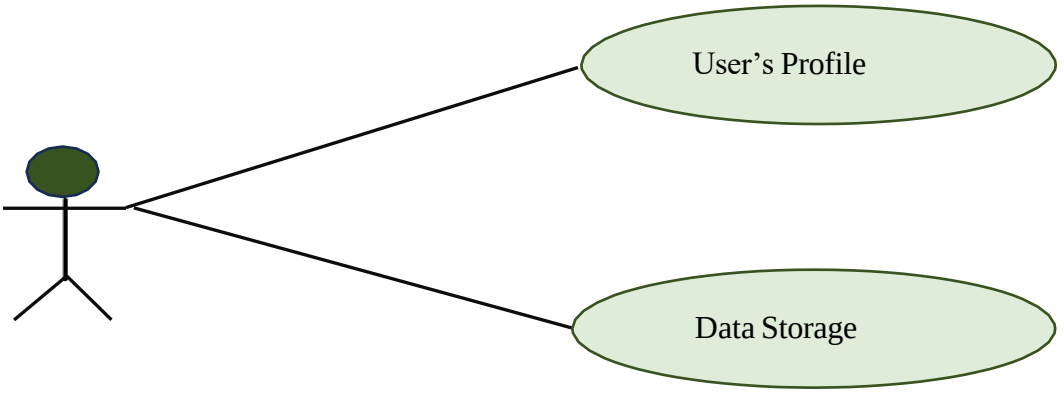


Fig 2.3 : Maintenance

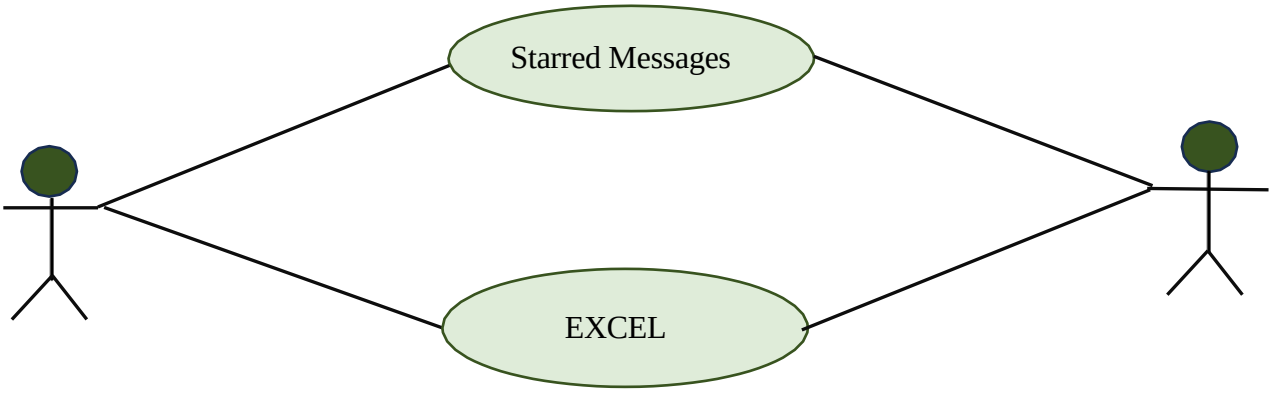


Fig 2.4 : Monitor

## 2.6 GUARD TALK Infrastructure

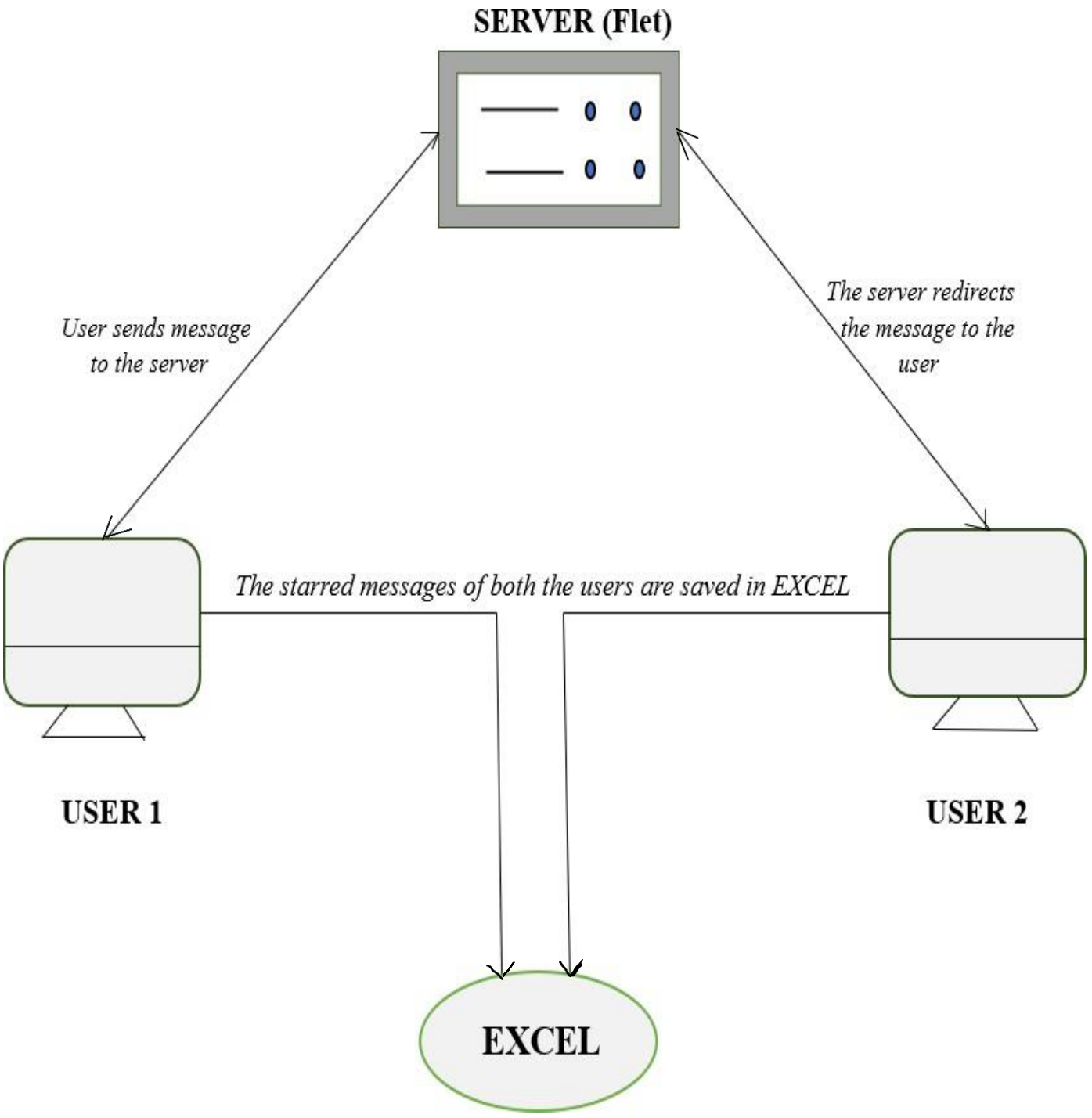


Fig 2.5 Guard Talk Architecture

## Chapter 3

# Implementation

The provided code implements a chat application using the Flet framework for the frontend and a simple in-memory database for user authentication. Here's a breakdown of the code:

## 3.1 Backend Implementation

- This file defines a class Users DB that simulates a user database. It contains a list of dictionaries representing users with their usernames and passwords.
- **read_db:** This function checks if a given username and password combination exists in the users list. It returns True if a match is found, False otherwise.
- **write_db:** This function adds a new user with the provided username and password to the users list. It returns True if successful.[1]

## 3.2 Frontend Implementation

- **main.py:** This is the main entry point for the application. It defines several functions for handling user interactions and manages the application state.
- **sign_in and sign_up:** These functions call the corresponding methods in Users DB to authenticate users. Upon successful login, it sets a user session and redirects to the chat view.
- **on_message:** This function handles incoming messages received through Flet's pub/sub mechanism. It renders the message based on its type (chat message or login message).
- **send_message_click:** This function sends the user's message through the pub/sub channel.
- **toggle_star:** This function toggles the starred state of a message and updates the starred message list (stored in a global dictionary starred messages).
- **save_to_excel:** This function (not used in the current implementation) saves starred messages for a user to an Excel file.
- **signin_form.py and signup_form.py:** These files define reusable user controls for the sign-in and sign-up forms. They handle user input, validation, and call the appropriate functions in main.py to submit the form.[1]

### 3.2.1 Flet UI Components

The code uses various Flet UI components to build the user interface[2]

- **ft.Page:** This represents the main application window.
- **ft.Column, ft.Row:** These are used to arrange UI elements vertically and horizontally.
- **ft.Text, ft.TextField:** These display text and allow user input.
- **ft.IconButton, ft.ElevatedButton:** These represent clickable buttons.
- **ft.Dropdown:** This allows users to select an emoji from a list.
- **ft.ListView:** This displays a list of chat messages.
- **ft.Container:** This is a generic container for grouping other controls.
- **ft.Banner, ft.AlertDialog:** These display popups for messages and confirmations.

### 3.2.2 Running the Application

1. **Install the required dependencies:**

pip install -r requirements.txt (replace with your actual requirements file path)[2]

- anyio
- certify
- flet
- flet-core
- h11
- http core
- httpx
- idna
- oauthlib
- packaging
- repath
- rfc3986
- six
- sniffio
- watchdog
- websocket-client
- websockets

2. **Run the application:**

python main.py

3. **Web Browser:**

The application will launch in a web browser window (system's default browser).

## Chapter 4

# Testing

This chapter covers the testing processes and methodologies applied to the Guard Talk. Testing is a comprehensive process that includes functional, usability, security, performance, compatibility, network, and localization testing.

## 4.1 Testing Objectives

- Verify that users can send and receive text messages successfully.
- Ensure messages are displayed correctly in the chat window.
- Confirm that messages disappear when a user exits the chat screen.
- Verify the correct display of user names, timestamps, and message content.

## 4.2 Testing Environment

- **Hardware**: Laptop/PC with minimum GB RAM and multi-core processor.
- **Software**:
  - Backend and frontend: users_db.py for backend, flet.py and main.py for frontend.
  - Data Storage: Spreadsheet(Excel).
  - Testing Tools: pip install -r requirements.txt and flet run main.py is used to run the application.
- **Operating System**: Windows 11
- **Browser**: Google Chrome and Microsoft Edge for cross-browser testing.

## 4.3 Types of Testing

### 4.3.1 Unit Testing

- **Objective**: To ensure the correct functionality of individual components within a real-time chat application, unit testing is crucial. The primary objective is to isolate and test each component independently to identify and fix bugs early in the development process.
- **Test Cases**:
  - **User Authentication**:
    - Test user registration and login.
    - Verify password hashing and salting.
    - Ensure session management and token-based authentication.

- **User Interface:**
- Test UI components like chat boxes, message input fields, and user lists.proper rendering of messages and user information.
- **Message Model:**
- Test message creation with valid and invalid data.
- Verify message properties (sender, content).
- Ensure proper serialization and deserialization of messages.

### 4.3.2 Integration Testing

- **Objective**: To verify the seamless integration of various components in a real-time chat application, ensuring smooth communication and data synchronization between users.
- **Test Cases**:
  - **Message Sending and Receiving:**
    - Send a message from one user to another.
    - Verify that the message is received and displayed correctly on the recipient's screen.
    - Test with various message lengths and formats (text, emojis).
  - **User Login and Logout:**
    - Test successful login and logout.
    - Verify that unauthorized users cannot access private chats or send messages.
  - **Real-time Message Delivery:**
    - Send a message and verify that it is delivered to the recipient's client in real-time.
    - Test with different network conditions (slow, fast, intermittent).

### 4.3.3 Functional Testing

- **Objective**: To ensure the real-time chat application functions as intended, providing a seamless and efficient communication experience for users. This testing will validate core features, user interface, and system performance under various conditions.
- **Test Cases**:
  - **User Registration and Login:**
    - Verify successful registration with valid credentials.
    - Verify error message for invalid credentials (e.g., weak password, duplicate username).
    - Verify successful login with valid credentials.
  - **Performance and Reliability:**
    - Verify application performance under heavy load, concurrent messages.
    - Verify application stability and reliability under various network conditions.

# FLOW CHART

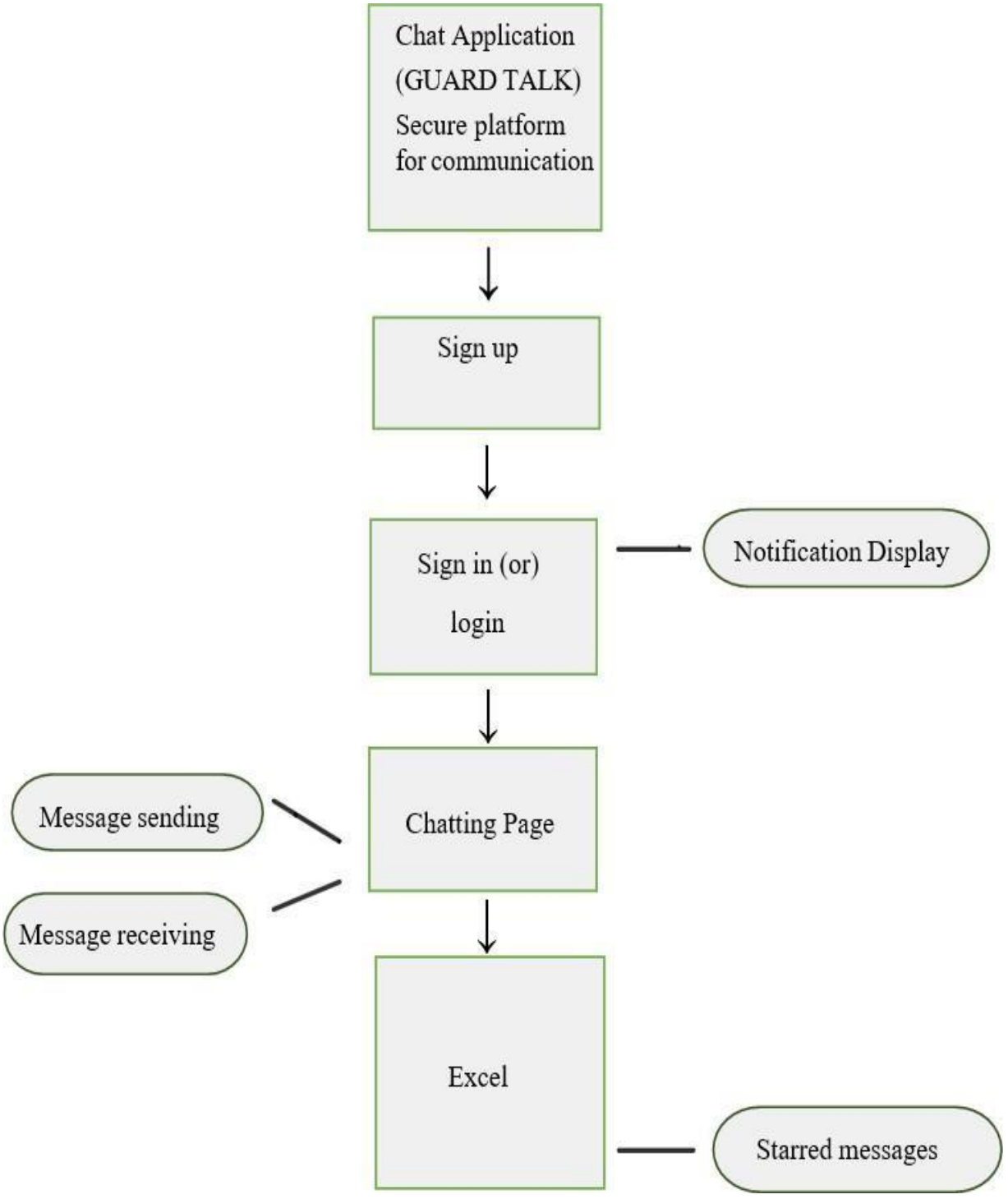


Fig 4.1 Flow chart for Guard talk

## Chapter 5

# Results and Discussion

This chapter summarizes the results of the Guard Talk project, discussing its effectiveness, reliability, and alignment with the intended objectives. The chapter also covers any challenges encountered, key insights, and recommendations for future improvements.

### 5.1 Results

➢ The below Figures depicts the complete process of chat application steps one after the other.

- This depicts the creation of an account by user 1

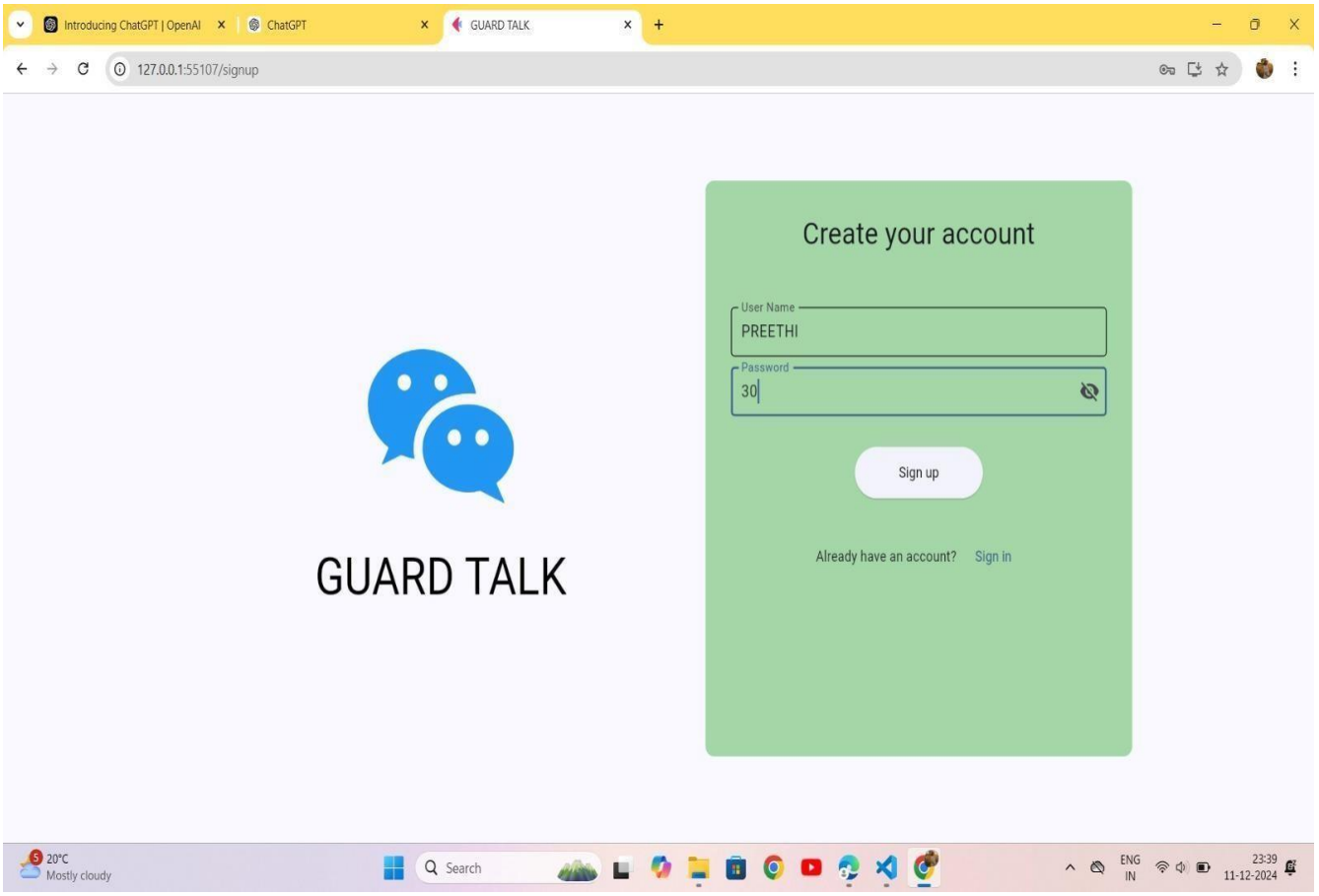


Fig: 5.1 Creation of account by user 1

- This depicts the notification of successful creation of account of user 1

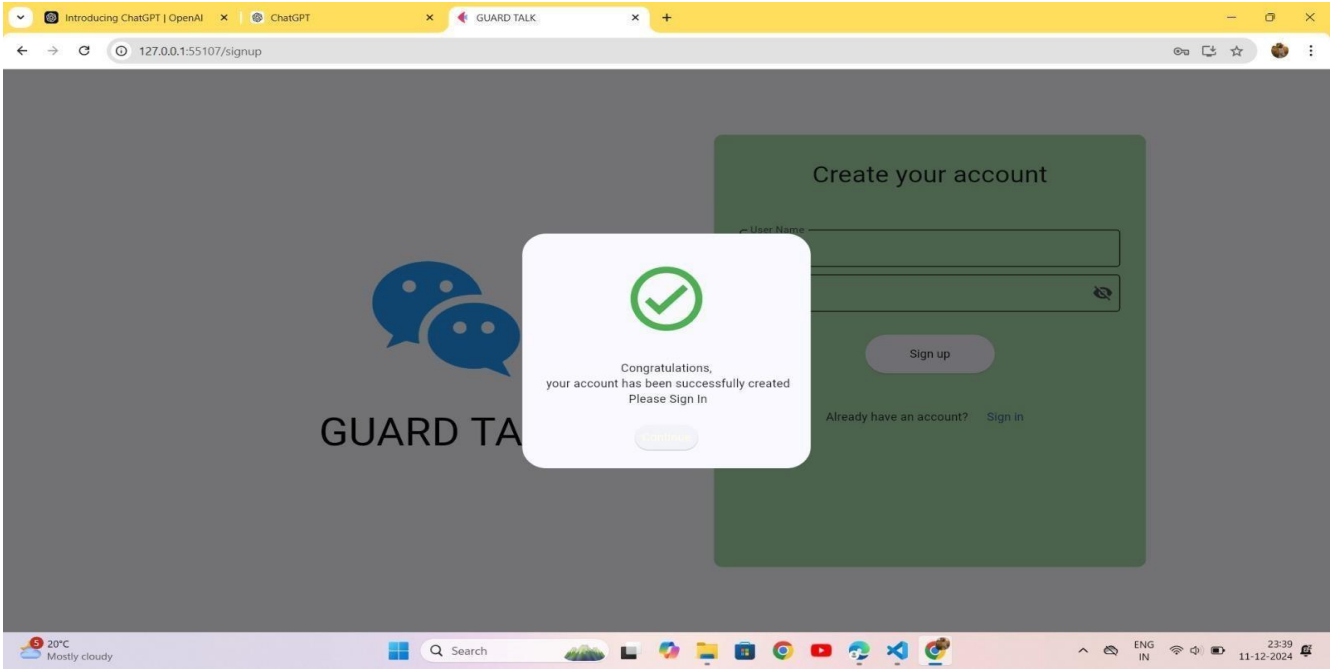


Fig: 5.2 Successful creation of account by user 1

- This depicts the sign in of user 1 to his/her account

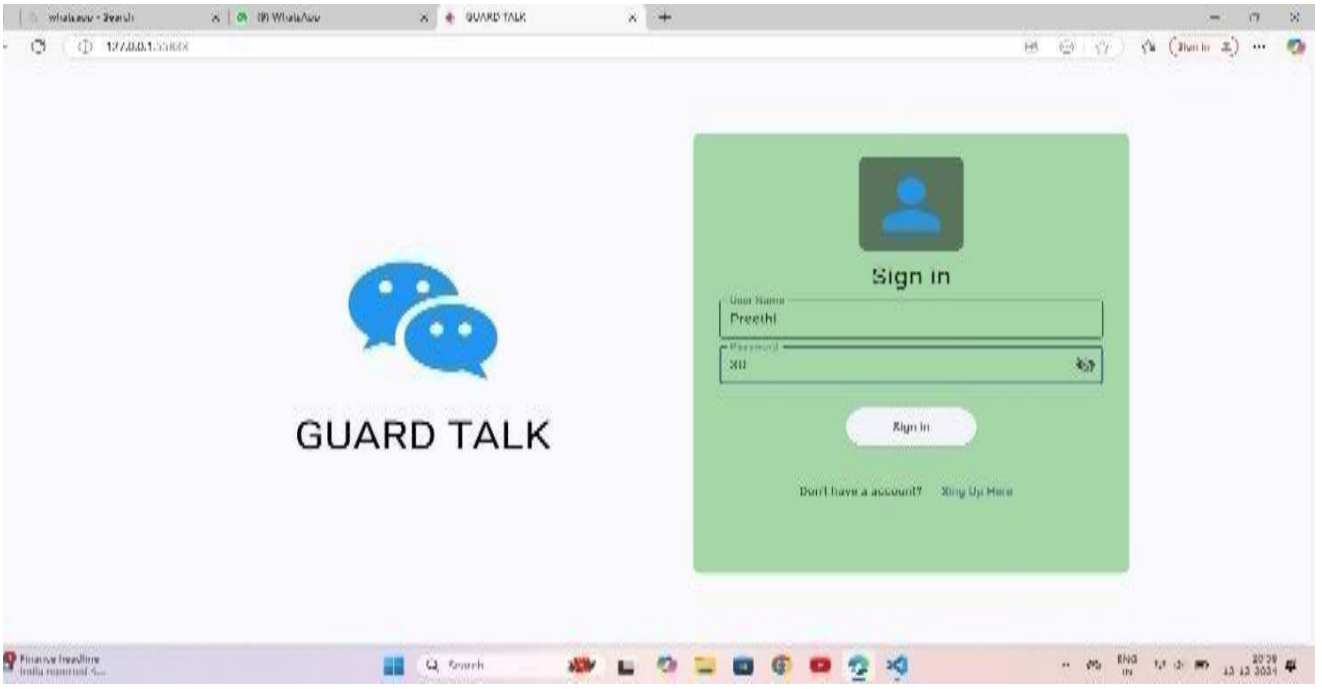


Fig: 5.3 Sign in by user 1

- This depicts the creation of account by user 2

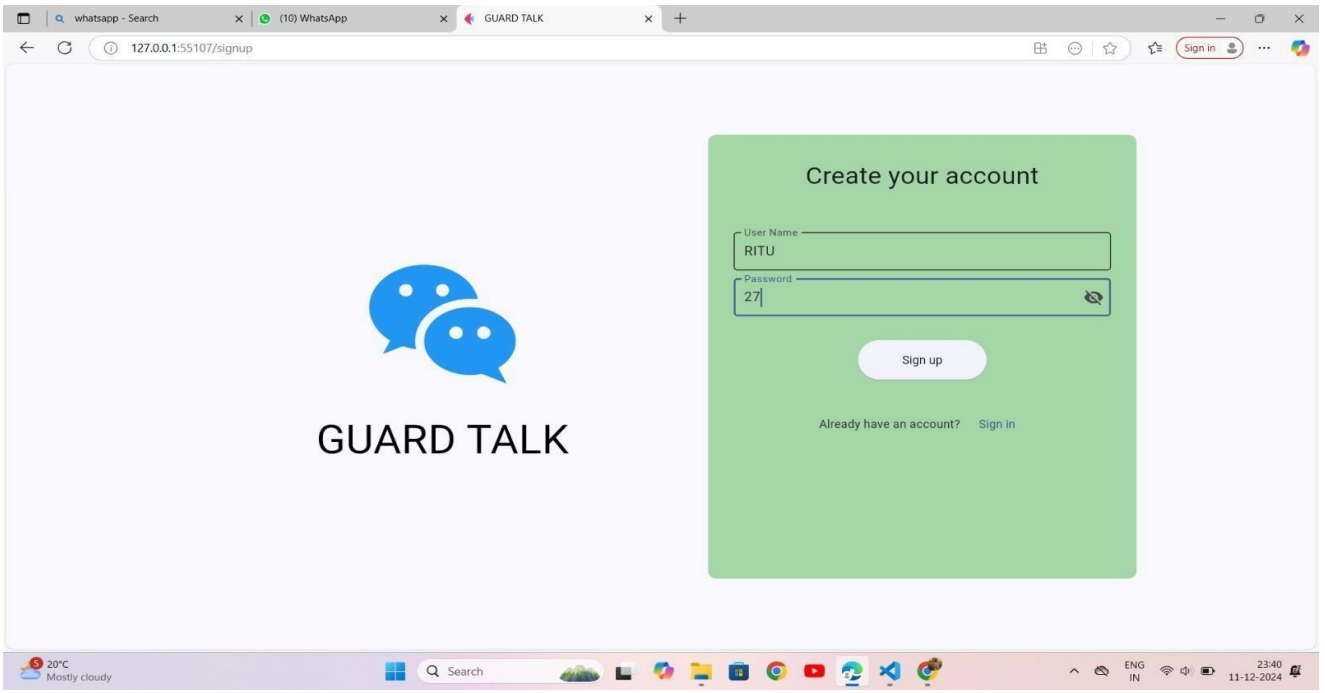


Fig:5.4 Creation of account by user 2

- This depicts the sign in of user 2 to his/her account

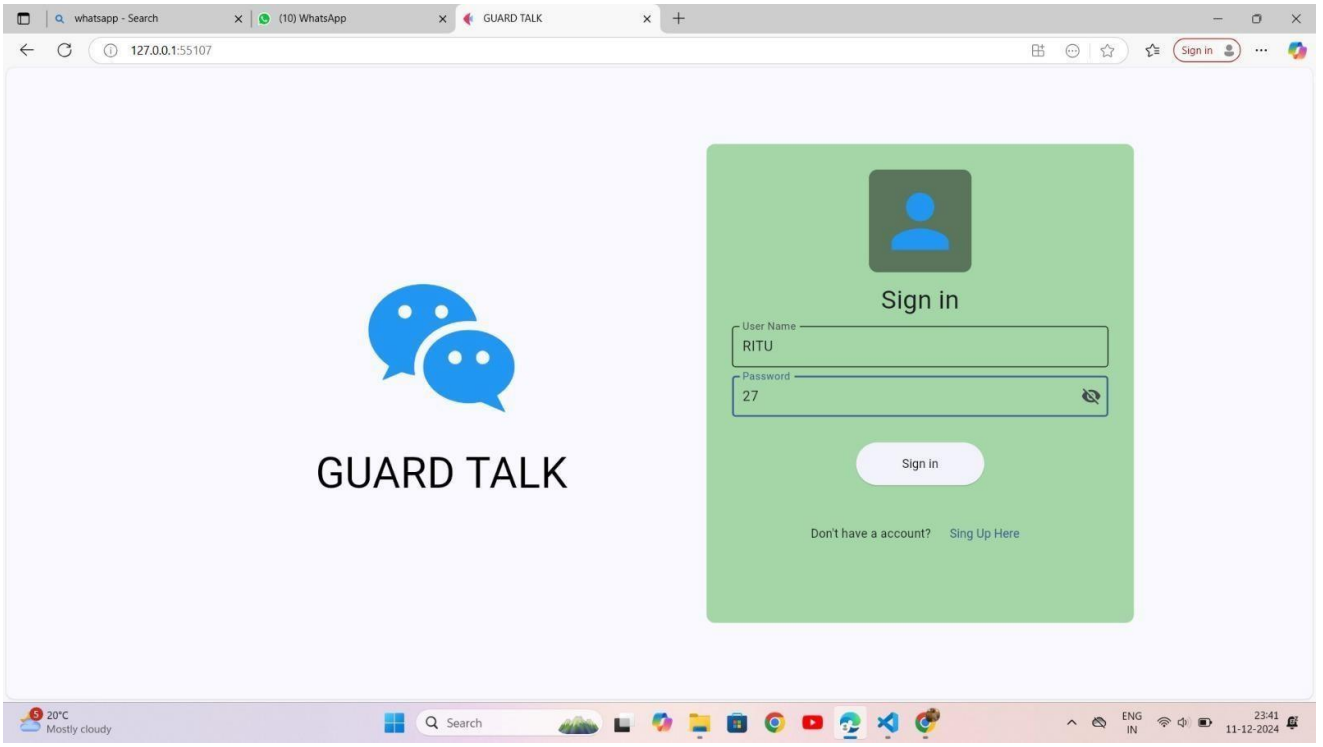


Fig: 5.5 Sign in by user 2

- This depicts the joining of both the users to the chatting page

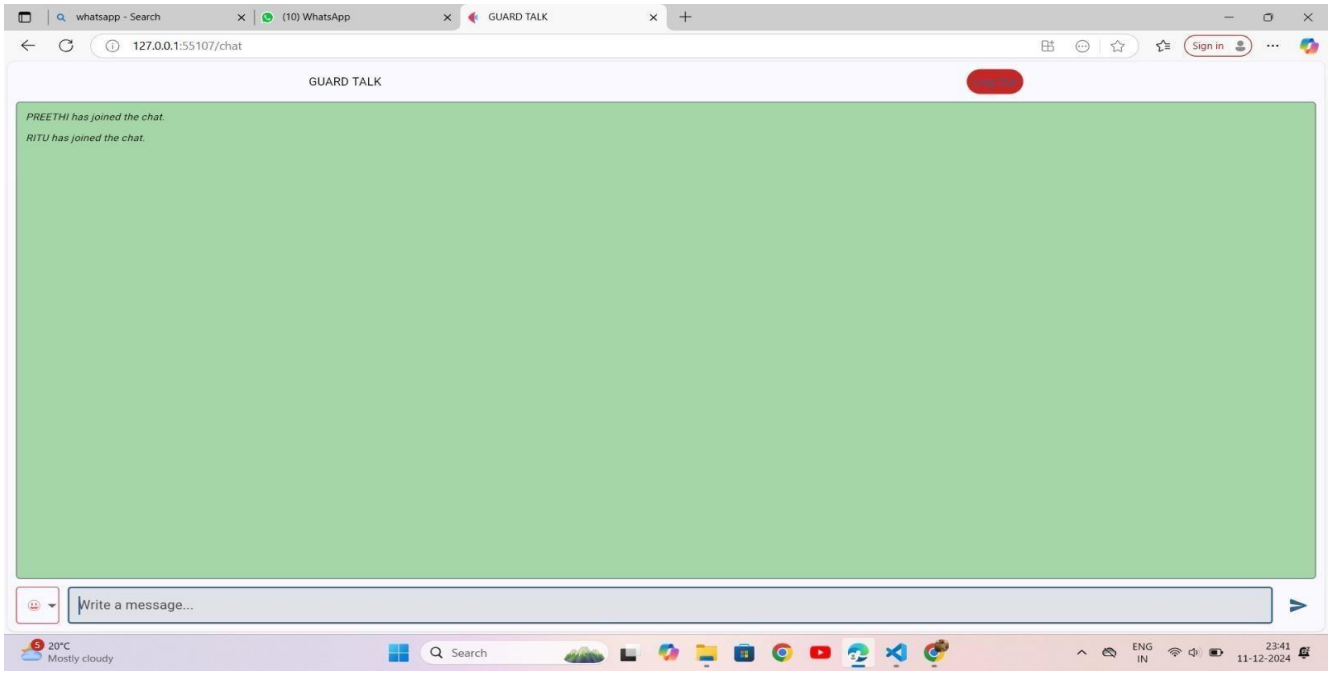


Fig:5.6 Both users joined the chatting page

- This depicts the chatting page display of both the users

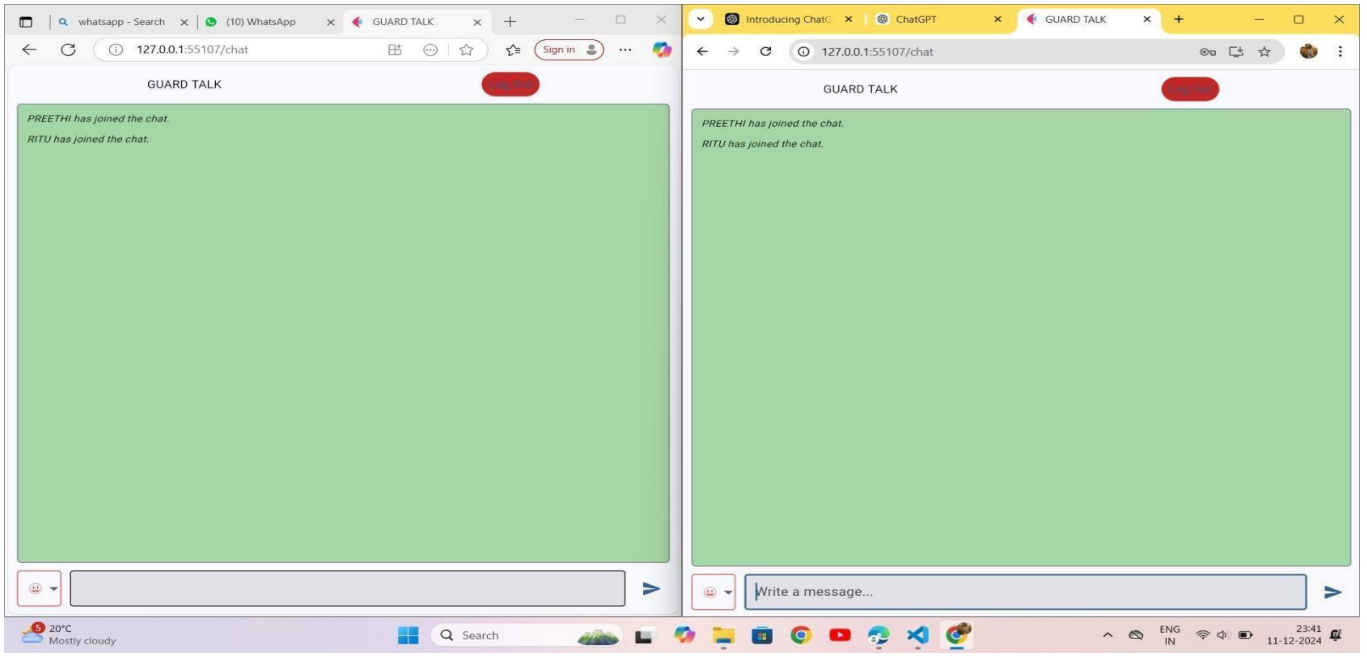


Fig:5.7 View of two chatting page

- This depicts the  chatting of both the users

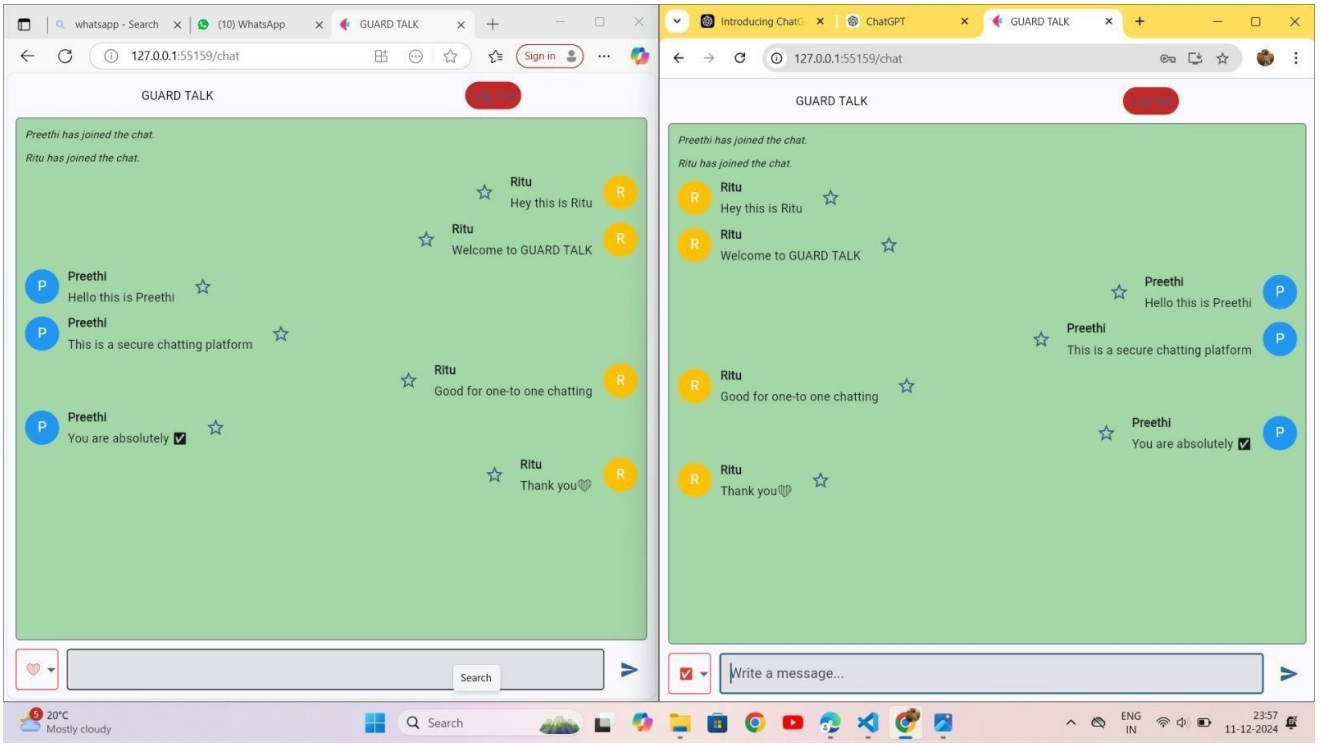


Fig: 5.8 Chatting between both the user

- This depicts the starring of important messages by the users

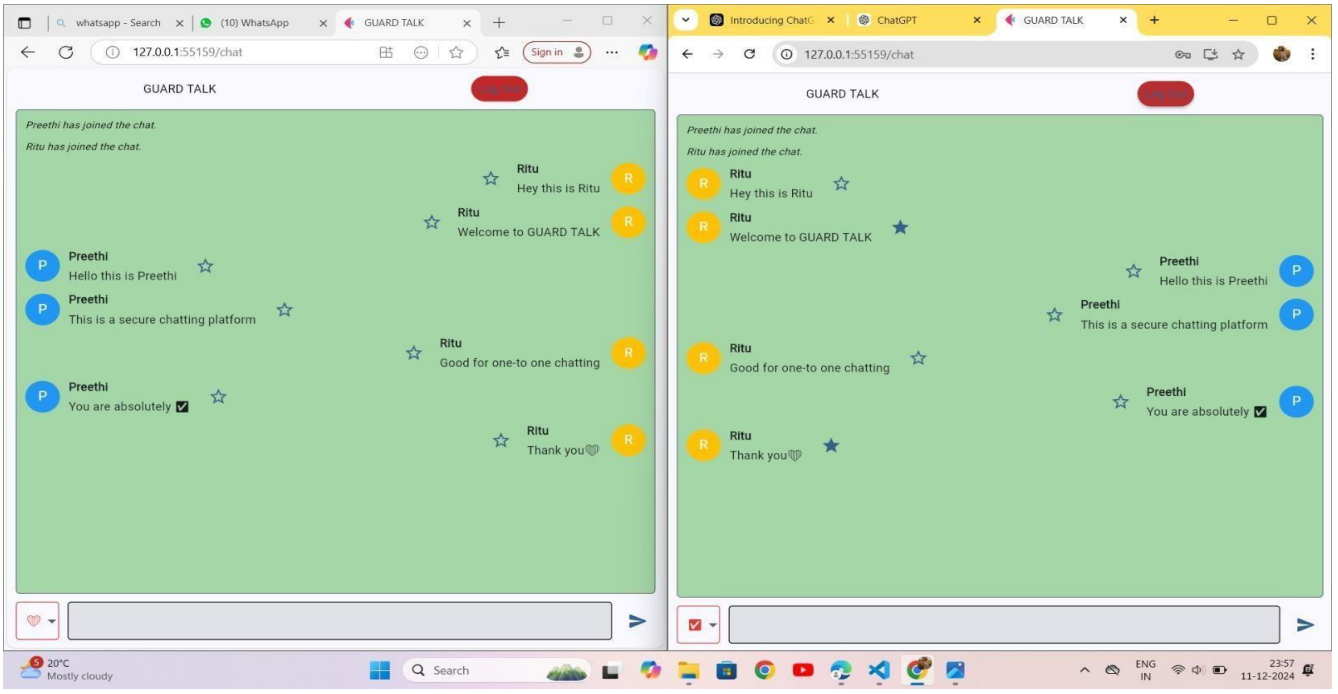


Fig:5.9  Starring the important messages by the users

- This depicts the display of  starred messages when signed in again

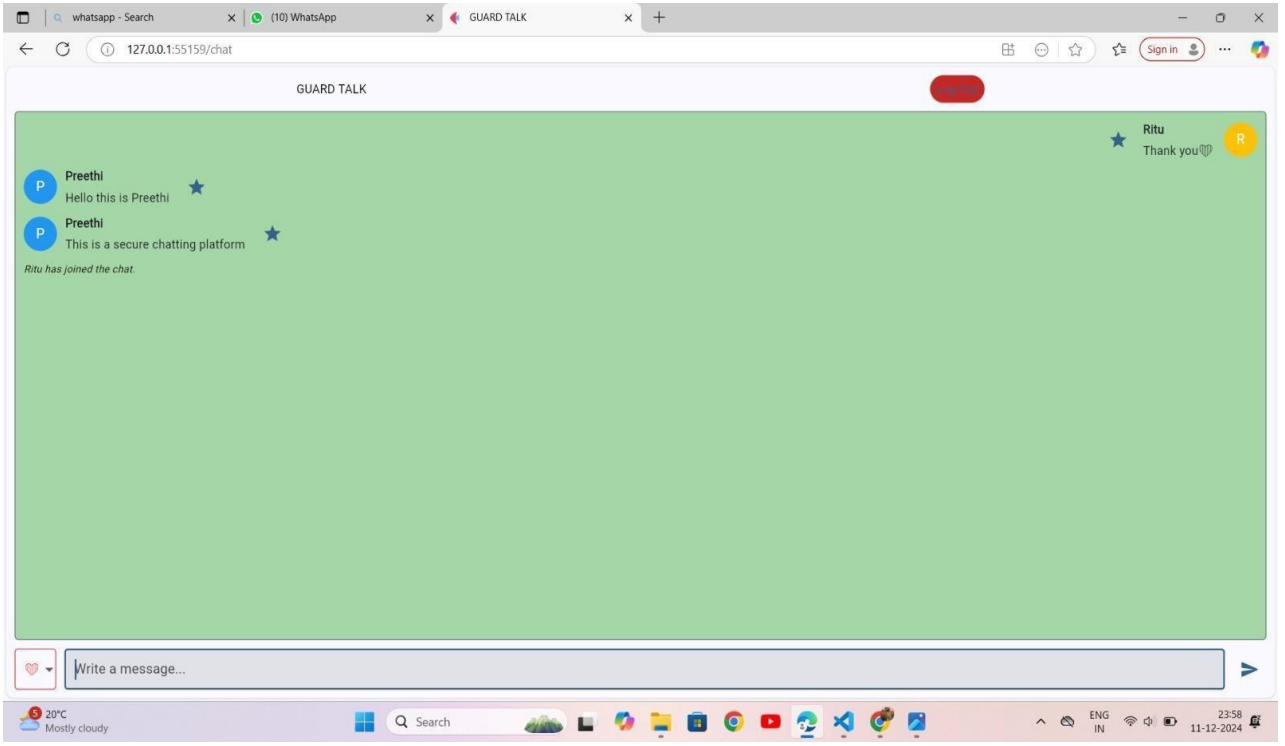


Fig:5.10 Display of starred messages

- This depicts the starred messages of user 1 which get stored in EXCEL

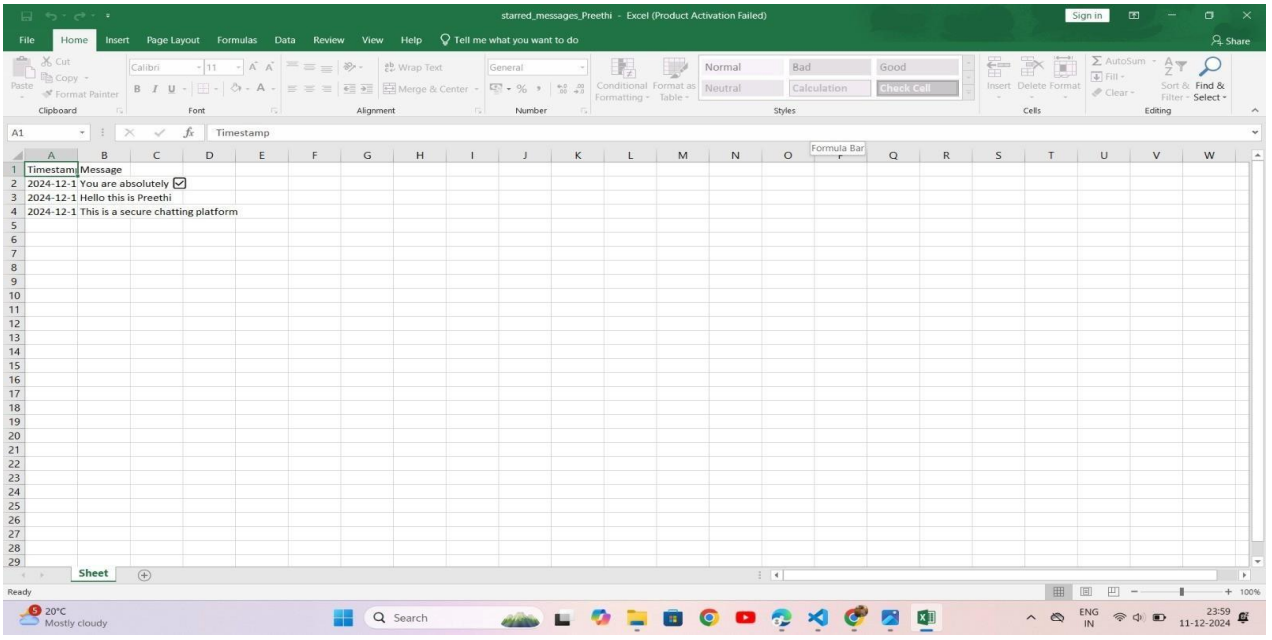


Fig 5.11 Starred messages of user 1 which are stored in EXCEL

- This depicts the starred messages of user 2 which get stored in EXCEL

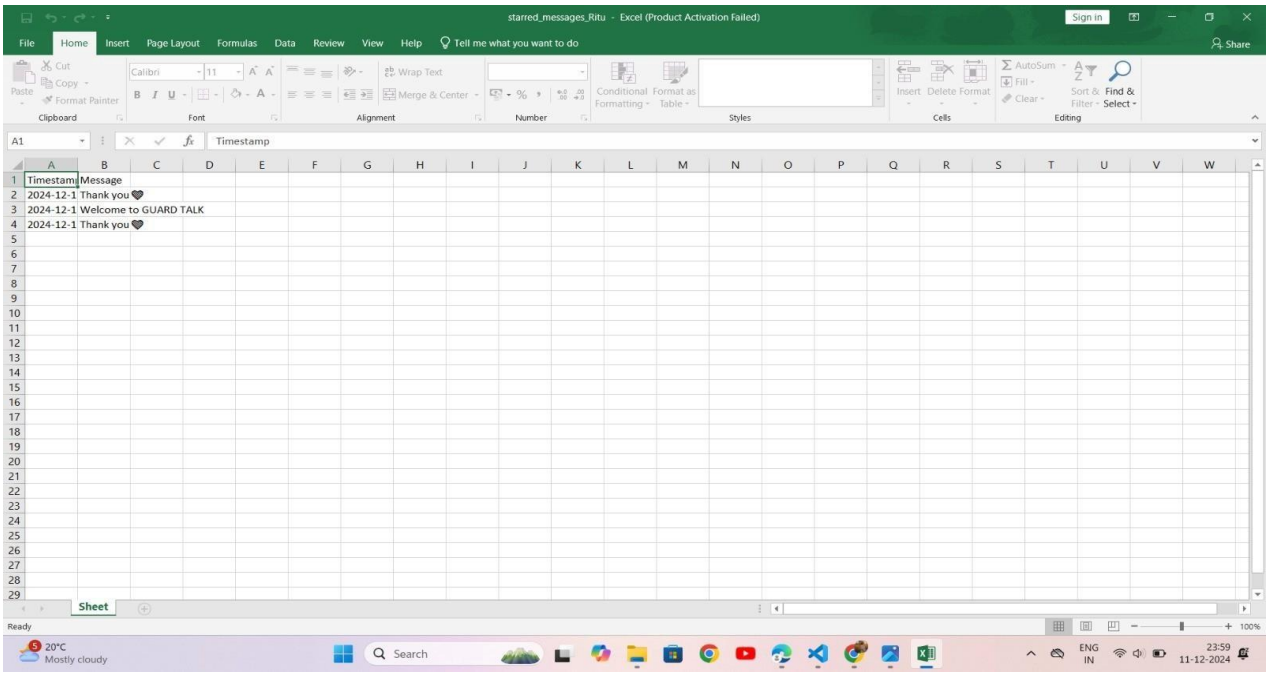


Fig 5.12 Starred messages of user 2 which are stored in EXCEL

## Chapter 6

# Conclusion and Future Enhancements

### 6.1 Conclusion

- This GUARD TALK(secure platform for communication) successfully supports one-to-one conversations, providing a seamless user experience. However, messages are temporary and will be erased when users navigate away from the chatroom, ensuring privacy and a clean interface. Notably, important messages are stored in a Google Spreadsheet (Excel), allowing for easy tracking and retrieval.
- A key feature of this application is the implementation of disappearing messages. Upon exiting the chat screen, messages are automatically deleted from the server, ensuring user privacy and security. This feature enhances the user experience by providing a sense of confidentiality and control over their conversations.
- This approach balances real-time interaction with selective data retention, ensuring both security and utility. The ultimate goal of this project is to offer an efficient, user-friendly chat experience with essential features for message management.

### 6.2 Future Enhancements

To further increase the effectiveness and usability of the Student Attendance System, the following enhancements are recommended:

- **Group Chat**: Implement group chat features, allowing users to create and join groups to facilitate discussions with multiple participants.
- **Voice and Video Calls:** Integrate real-time voice and video calling capabilities for richer communication.
- **Offline Messaging:** Store and deliver messages to users even when they are offline, ensuring uninterrupted communication.
- **Message Reactions and Replies:** Introduce features like message reactions and threaded replies for more engaging conversations.
- **Database Optimization:** Optimize the database schema and query performance to handle increasing amounts of data.

# APPENDIX - A

## CODE

### This is the code for chat_message.py

```
import flet as ft

#Message:
   class Message
   def _init_(self, user: str, text: str, message_type: str,starred=False):
      self.user = user
      self.text = text
      self.message_type = message_type
      self.starred = starred #New attribute to track if a message is starred


class ChatMessage(ft.Row):
   def _init_(self, message: Message, toggle_star):
      super()._init_()
      self.message        =        message
      self.toggle_star    =    toggle_star
      self.controls = [
         ft.CircleAvatar(
            content=ft.Text(self.get_initials(message.user)),
            color=ft.colors.WHITE,
            bgcolor=self.get_avatar_color(message.user),
         ),
         ft.Column(
            [
               ft.Text(message.user, weight="bold"),
               ft.Text(message.text, selectable=True),
            ],
            tight=True,
            spacing=5,
         ),
         ft.IconButton(
            icon=ft.icons.STAR if message.starred else ft.icons.STAR_BORDER,
            tooltip="Star message",
            on_click=self.on_star_click),
      ]
```

```
    def on_star_click(self, e):
        self.message.starred = not self.message.starred
        self.toggle_star(self.message)
        self.controls[2].icon = (
            ft.icons.STAR if self.message.starred else ft.icons.STAR_BORDER
        )
        self.update()

    def get_initials(self, user: str):
        return user[:1].capitalize()

    def get_avatar_color(self, user: str):
        colors_lookup = [
            ft.colors.AMBER,
            ft.colors.BLUE,
            ft.colors.BROWN,
            ft.colors.CYAN,
            ft.colors.GREEN,
            ft.colors.INDIGO,
            ft.colors.LIME,
            ft.colors.ORANGE,
            ft.colors.PINK,
            ft.colors.PURPLE,
            ft.colors.RED,
            ft.colors.TEAL,
            ft.colors.YELLOW,
        ]
        return colors_lookup[hash(user) % len(colors_lookup)]
```

## This is the code for main.py

```
import flet as ft
import openpyxl
from openpyxl.styles import Alignment
from datetime import datetime
from signin_form import *
from signup_form import *
from users_db import *
```

```
from chat_message import *

# Global dictionary to store starred messages
starred_messages = {}  # Key: user, Value: list of starred messages

def save_to_excel(user: str, messages: list):
    file_name = f"starred_messages_{user}.xlsx"

    # Create a new workbook or load existing
    one try:
        wb = openpyxl.load_workbook(file_name)
        ws = wb.active
    except FileNotFoundError:
        wb = openpyxl.Workbook()
        ws = wb.active
        ws.append(["Timestamp", "Message"])  # Add header row

    # Add messages to the Excel file
    for message in messages:
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        ws.append([timestamp, message.text])

    # Save the workbook
    wb.save(file_name)

# Function to toggle the starred state of a message
def toggle_star(message: Message):
    user = message.user
    if user not in starred_messages:
        starred_messages[user] = []

    if message.starred:
        if message not in starred_messages[user]:
            starred_messages[user].append(message)
            save_to_excel(user, [message])  # Save the new starred message
    else:
        if message in starred_messages[user]:
            starred_messages[user].remove(message)
            save_to_excel(user, starred_messages[user])  # Update the Excel file
```

```
def main(page: ft.Page):
    page.title = "GUARD TALK"
    page.vertical_alignment = ft.MainAxisAlignment.CENTER
    page.horizontal_alignment = ft.CrossAxisAlignment.CENTER

    # ** Functions**
     def
    dropdown_changed(e):
        new_message.value = new_message.value + emoji_list.value
        page.update()

    def close_banner(e):
        page.banner.open = False
        page.update()

    def open_dlg():
        page.dialog = dlg
        dlg.open = True
        page.update()

    def close_dlg(e):
        dlg.open = False
        page.route = "/"
        page.update()

    def sign_in(user: str, password: str):
        db = UsersDB()
        if not db.read_db(user, password):
            print("User no exist ...")
            page.banner.open = True
            page.update()
        else:
            print("Redirecting to chat...")
            page.session.set("user", user)
            page.route = "/chat"
            page.pubsub.send_all(
                Message(
                    user=user,
                    text=f"{user} has joined the chat.",
                    message_type="login_message",
                )
            )
```

```
        page.update()

    def sign_up(user: str, password: str):
        db = UsersDB()
        if db.write_db(user, password):
            print("Successfully Registered User...")
            open_dlg()

    def on_message(message: Message):
        if message.message_type == "chat_message":
            m = ChatMessage(message, toggle_star)
        elif message.message_type == "login_message":
            m = ft.Text(message.text, italic=True, color=ft.colors.GREY, size=12)
        chat.controls.append(m)
        page.update()

    page.pubsub.subscribe(on_message)

    def send_message_click(e):
        page.pubsub.send_all(
            Message(
                user=page.session.get("user"),
                text=new_message.value,
                message_type="chat_message",
            )
        )
        new_message.value = ""
        page.update()

    def btn_signin(e):
        page.route = "/"
        page.update()

    def btn_signup(e):
        page.route = "/signup"
        page.update()

    def btn_exit(e):
        user = page.session.get("user") # Get the current user
        if user:
            # Clear all non-starred messages
            chat.controls = [
```

```
            ChatMessage(msg, toggle_star)
            for u, messages in starred_messages.items()
            for msg in messages
        ]

        # Remove starred messages of the user logging out
        starred_messages.pop(user, None)
    page.session.remove("user")
    page.route = "/"
    page.update()

# **        Aplication UI          ****
principal_content = ft.Column(
    [
        ft.Icon(ft.icons.WECHAT,        size=200,        color=ft.colors.BLUE),
        ft.Text(value="GUARD TALK", size=50, color=ft.colors.GREY),
    ],
    height=400,
    width=600,
    alignment=ft.MainAxisAlignment.CENTER,
    horizontal_alignment=ft.CrossAxisAlignment.CENTER,
)
emoji_list        =        ft.Dropdown(
    on_change=dropdown_changed,
    options=[
        ft.dropdown.Option(""),
        ft.dropdown.Option(""),
        ft.dropdown.Option(""),
        ft.dropdown.Option(""),
        ft.dropdown.Option(""),
        ft.dropdown.Option(""),
        ft.dropdown.Option(""),
        ft.dropdown.Option("
        "),
        ft .d r op do wn. Op tion ("),
        ft.dropdown.Option(""),
        ft.dropdown.Option(""),
        ft.dropdown.Option("
        "
        ),
        ft.dropdown.Option(""),
        ft.dropdown.Option("+"),
        ft.dropdown.Option("    "),
```

```
            ft.dropdown.Option(""),
            ft.dropdown.Option("    "),
            ft.dropdown.Option(""),
            ft.dropdown.Option(""),
            ft.dropdown.Option(""),
            ft.dropdown.Option(""),
            ft.dropdown.Option(""),
            ft.dropdown.Option(""),
            ft.dropdown.Option(""),
            ft.dropdown.Option("  "),
            ft.dropdown.Option("    "),
            ft.dropdown.Option(""),
        ],
        width=50,
        value="",
        alignment=ft.alignment.center,
        border_color=ft.colors.AMBER,
        color=ft.colors.AMBER,
    )

    signin_UI = SignInForm(sign_in, btn_signup)
    signup_UI = SignUpForm(sign_up, btn_signin)

    chat = ft.ListView(
        expand=True,
        spacing=10,
        auto_scroll=True,
    )

    new_message = ft.TextField(
        hint_text="Write a message...",
        autofocus=True,
        shift_enter=True,
        min_lines=1,
        max_lines=5,
        filled=True,
        expand=True,
        on_submit=send_message_click,
    )
```

```
page.banner = ft.Banner(
    bgcolor=ft.colors.BLACK45,
    leading=ft.Icon(ft.icons.ERROR, color=ft.colors.RED, size=40),
    content=ft.Text("Log in failed, Incorrect User Name or Password"),
    actions=[
        ft.TextButton("Ok", on_click=close_banner),
    ],
)

dlg = ft.AlertDialog(
    modal=True,
    title=ft.Container(
        content=ft.Icon(
            name=ft.icons.CHECK_CIRCLE_OUTLINED, color=ft.colors.GREEN,
  size=100
        ),
        width=120,
        height=120,
    ),
    content=ft.Text(
        value="Congratulations,\n your account has been successfully created\n Please
  Sign In",
        text_align=ft.TextAlign.CENTER,
    ),
    actions=[
        ft.ElevatedButton(
            text="Continue", color=ft.colors.WHITE, on_click=close_dlg
        )
    ],
    actions_alignment="center",
    on_dismiss=lambda e: print("Dialog dismissed!"),
)

# **      Routes      **
def route_change(route):
    if page.route == "/":
        page.clean()
        page.add(
            ft.Row(
                [principal_content, signin_UI],
                alignment=ft.MainAxisAlignment.CENTER,
```

```
            )
        )

    if page.route == "/signup":
        page.clean()
        page.add(
            ft.Row(
                [principal_content, signup_UI],
                alignment=ft.MainAxisAlignment.CENTER,
            )
        )

    if page.route == "/chat":
        if page.session.contains_key("user"):
            page.clean()
            page.add(
                ft.Row(
                    [
                        ft.Text(value="GUARD TALK", color=ft.colors.GREY),
                        ft.ElevatedButton(
                            text="Log Out",
                            bgcolor=ft.colors.RED_800,
                            on_click=btn_exit,
                        ),
                    ],
alignment=ft.MainAxisAlignment.SPACE_AROUND,
                )
            )
            page.add(
                ft.Container(
                    content=chat,
                    border=ft.border.all(1, ft.colors.OUTLINE),
                    border_radius=5,
                    padding=10,
                    expand=True,
                )
            )
            page.add(
                ft.Row(
                    controls=[
                        emoji_list,
                        new_message,
```

```
                    ft.IconButton(
                        icon=ft.icons.SEND_ROUNDED,
                        tooltip="Send message",
                        on_click=send_message_click,
                    ),
                ],
            )
        )

    else:
        page.route = "/"
        page.update()

page.on_route_change = route_change
page.add(
    ft.Row([principal_content, signin_UI], alignment=ft.MainAxisAlignment.CENTER)
)


ft.app(target=main, view=ft.WEB_BROWSER)
```

## This is the code for Signin_form.py

```
import flet as ft
# SignIn Form
class SignInForm(ft.UserControl):
    def _init_(self, submit_values,btn_signup):
        super()._init_()
        #Return values user and password
        self.submit_values = submit_values
        #Route to Sign Up Form
        self.btn_signup = btn_signup

    def btn_signin(self, e):
        if not self.text_user.value:
            self.text_user.error_text="Name cannot be blank!"
            self.text_user.update()
        if not self.text_password.value:
            self.text_password.error_text="Password cannot be blank!"
            self.text_password.update()
        else:
```

```
        #Return values 'user' and 'password' as arguments
        submit_values(self.text_user.value,self.text_password.value)

    def build(self):
        self.signin_image = ft.Container(
            content=ft.Icon(name=ft.icons.PERSON,       color=ft.colors.BLUE,       size=100),
            width=120,
            height=120,
            bgcolor=ft.colors.BLACK45,
            border_radius=10,
        )
        self.title_form=ft.Text(value="Sign in",text_align=ft.TextAlign.CENTER,size=30,  )

        self.text_user        =        ft.TextField(label="User        Name")
        self.text_password = ft.TextField(
            label="Password", password=True, can_reveal_password=True
        )
        self.text_signin=ft.ElevatedButton(text="Sign
    in",color=ft.colors.WHITE,width=150,height=50,on_click=          self.btn_signin)
    self.text_signup=ft.Row(controls=
    Here",on_click=self.btn_signup)],alignment=ft.MainAxisAlignment.CENTER)
    return ft.Container( width=500, height=560,
            bgcolor=ft.colors.TEAL_800,
            padding=30,
            border_radius=10,
            alignment=ft.alignment.center,
            content=ft.Column(
                [
                    self.signin_image,
                    self.title_form,
                    self.text_user,
                    self.text_password,
                    ft.Container(height=10),
                    self.text_signin,
                    ft.Container(height=20),
                    self.text_signup,
                ],
                horizontal_alignment=ft.CrossAxisAlignment.CENTER)
```

## This is the code for Signup_form.py

```
import flet as ft

# SignUp Form
class SignUpForm(ft.UserControl):
    def _init_(self, submit_values,btn_signin):
        super()._init_()
        #Return values user and password
        self.submit_values = submit_values
        #Route to signup Form
        self.btn_signin = btn_signin


    def btn_signup(self, e):
        if not self.text_user.value:
            self.text_user.error_text="Name cannot be blank!"
            self.text_user.update()
        if not self.text_password.value:
            self.text_password.error_text="Password cannot be blank!"
            self.text_password.update()
        else:
            #Return values 'user' and 'password' as arguments
            self.submit_values(self.text_user.value,self.text_password.value)
    def build(self):
        self.title_form=ft.Text(value="Create your
      account",text_align=ft.TextAlign.CENTER,size=30, )
        self.text_user = ft.TextField(label="User Name")
        self.text_password = ft.TextField(
            label="Password", password=True, can_reveal_password=True
        )
        self.text_signup=ft.ElevatedButton(text="Sign
      up",color=ft.colors.WHITE,width=150,height=50,on_click= self.btn_signup)
      self.text_signin=ft.Row(controls=[ft.Text(value="Already have an
      account?"),ft.TextButton(text="Sign
      in",on_click=self.btn_signin)],alignment=ft.MainAxisAlignment.CENTER)return
      ft.Container( width=500, height=560,
            bgcolor=ft.colors.TEAL_800,
            padding=30,
            border_radius=10,
```

```
            alignment=ft.alignment.center,
            content=ft.Column(
                [
                    self.title_form,
                    ft.Container(height=30),
                    self.text_user,
                    self.text_password,
                    ft.Container(height=10),
                    self.text_signup,
                    ft.Container(height=20),
                    self.text_signin,],
                horizontal_alignment=ft.CrossAxisAlignment.CENTER,
            ),
        )
```

## This is the code for users_db.py

```
class UsersDB():

    users_list=[{"user":"mario","password":"mario1"},{"user":"peach","password":"peac
    h1"}]

    def read_db(self,user_name:str,password:str):
        #print("Iniciando sesion. ")
        for i in self.users_list:
            if  (i["user"]==user_name  and  i["password"]==password):
                return True
        return False

    def write_db(self,user_name:str,password:str):

    #print("Creando usuario.. ")
    self.users_list.append({"user":user_name,"passwo
        rd":password})
    return True
```

## These are requirements_txt

anyio==3.6.2
certifi==2023.5.7
flet==0.7.1
flet-core==0.7.1
h11==0.14.0
httpcore==0.16.3
httpx==0.23.3
idna==3.4
oauthlib==3.2.2
packaging==23.1
repath==0.9.0
rfc3986==1.5.0
six==1.16.0
sniffio==1.3.0
watchdog==2.3.1
websocket-client==1.5.1
websockets==10.4

# References

## Citation Format

[1] Online Resources :

To create a frontend chatting page
https://www.youtube.com/watch?si=NEbIDVMcxTIUpE_S&v=YDZPp0EnzEA&feature=youtu.be

Connecting Excel files to the user_Db using openpyxl

https://www.youtube.com/watch?si=X4_s3l9Qy0JINgA&v=C0ivFxEw4Nk&feature=youtu.be

[2] Software Tools:

To install python libraries in Visual studio code[2]
https://youtu.be/ThU13tikHQw?si=sjlSoxl_mP1YMT7f