(SE) MINI-PROJECT REPORT

on

"WILD WHISPERS CHAT APPLICATION"

using PYTHON LANGUAGE AND POSTGRESQL

Submitted in fulfillment of the requirement of

University of Mumbai for the degree of

Bachelor of Engineering

(Information Technology)

By

ADVAIT BOTHE TU4F2223052

JAY KARIA TU4F2223060

GEET BARI TU4F2223056

ARYA HUMANE TU4F2223064

Under the Guidance of

Prof. Preeti Patil

terna

Knowledge to Empower

Department of Information Technology Engineering

TERNA ENGINEERING COLLEGE

University of Mumbai

APPROVAL SHEET

The Mini Project entitled "Wild Whispers" by Advait Bothe(46) , Jay Karia(53) Geet Bari (49) and Arya Humane(55) is approved for the degree of **Bachelor of Engineering in Information Technology.**

Examiners

1.....
.....
(Internal Examiner Name & Sign)

2.....
.....
(External Examiner name & Sign)

Date:

Place: Nerul, Navi Mumbai

CERTIFICATE

This is to certify that the Mini Project entitled "Wild Whispers" by Advait Bothe (46), Jay Karia (53) Geet Bari (49) and Arya Humane (55) submitted to the University of Mumbai inpartial fulfillment of the requirement for the award of the degree of **"Bachelor of Engineering" in "Information Technology" .**

(Prof. Preeti Patil)

Supervisor

(Dr. Vaishali Khairnar)

Head of Department

(Dr. Lakshmappa Ragha)

Principal

ACKNOWLEDGEMENT

We express our deep gratitude to our project guide, Prof. Preeti Patil , for providing timely assistance to our query and for the guidance that she gave owing to her experience in this field for the past many years. She had indeed been a lighthouse for us on this journey.

We are grateful to our HOD Dr. Vaishali Khairnar for extending her facilitation directly and indirectly through numerous channels in our project work. We extend our sincere appreciation to the entire faculty members for their value within and tip throughout the coming up of the project. Their contributions are valuable in numerous ways, and we discover it troublesome to acknowledge the individual

TABLE OF CONTENT

I. ABSTRACT

"Wild Whispers-Chat Application" is a project aimed at developing a robust and dynamic chat application using Python programming language and PostgreSQL database management system. The application aims to provide users with a seamless and interactive platform for real-time communication while ensuring data integrity and security.

The project utilizes the versatility of Python to implement various features such as user authentication, message encryption, real-time messaging, and user-friendly interfaces. Python's extensive libraries and frameworks are leveraged to create a responsive and efficient chat application.

II. INTRODUCTION

2.1 Introduction:

The Real-Time Chat Application is a computer mini project designed to facilitate seamless communication between users in a real-time environment. This project aims to create a user-friendly and responsive chat platform, allowing users to engage in one-on-one and group conversations with instant message delivery.

1. User Authentication: The system provides a secure user authentication mechanism, allowing users to register, log in, and maintain account credentials with encrypted password storage.

2. Real-Time Communication: Leveraging WebSocket technology, the application supports real-time messaging, enabling users to exchange messages instantly. The server efficiently handles incoming and outgoing messages for a seamless chat experience.

3. Data Persistence: User information and chat history are securely stored in a database, providing data persistence for users' accounts and conversations. The chosen database technology ensures reliability and efficient retrieval of chat records.

4. Deployment and Security: The application can be deployed on popular cloud platforms, offering scalability and accessibility.

2.2 Objectives:

1. Enhance Usability and User Experience:

Design and implement a user-friendly interface that ensures a seamless and responsive experience across various devices, promoting user engagement and satisfaction.

2. Implement Robust Security Measures:

Incorporate industry-best security practices, including end-to-end encryption, secure user authentication mechanisms, and data protection, to safeguard user information and enhance privacy.

3. Optimize Real-Time Communication:

Develop and implement a real-time communication system using WebSocket technology to ensure instant message delivery, low latency, and reliable performance, enhancing the overall user communication experience.

4. Ensure Scalability and Performance:

Architect the application to scale efficiently, accommodating a growing user base while maintaining optimal performance. Conduct scalability testing to validate the system's ability to handle concurrent users and message throughput.

III. LITERATURE SURVEY

Journal Title	Publish Date	Advantages	Disadvantages
The WebSocket Protocol	December 2020	Facilitates real-time, full-duplex communication between client and server Security through origin-based model Works well with web browsers	May not be supported by older browsers Requires additional security measures to prevent certain vulnerabilities
The OAuth 2.0 Authorization framework	October 2021	Simplifies authentication for developers Enables secure authorization without sharing credentials Enhanced security compared to OAuth 1.0	Implementation complexity Potential security risks if not implemented properly
End-to-End Encryption for Chat App	December 2020	Ensures privacy and confidentiality of messages. Prevents unauthorized access to chat content. Dynamic key encryption for added security	Performance overhead due to encryption and decryption processes Key management challenges
Enhanced Chat Application	May 2020	Innovative features like predictive texting and themed messaging Enhanced user experience and engagement	Increased complexity for users unfamiliar with advanced features Potential compatibility issues with older devices
Profiling of Secure Chat and Calling Apps	October 2022	Helps identify vulnerabilities and weaknesses in existing apps. Provides insights for improving security measures	Requires significant resources and expertise for implementation May uncover sensitive information during analysis

3.1 Survey of existing systems:

1. **WhatsApp:** Owned by Facebook, WhatsApp is one of the most widely used messaging apps globally. It offers end-to-end encryption, voice and video calling, group chats, and multimedia sharing.
2. **Telegram**: Known for its emphasis on security, Telegram provides features like secret chats with end-to-end encryption, self-destructing messages, channels for broadcasting messages to large audiences, and bots for automated interactions.
3. **Signal**: Signal is highly regarded for its privacy and security features, including end-to-end encryption for all messages, voice and video calls, disappearing messages, and robust user verification methods.
4. **Facebook Messenger**: Integrated with Facebook's social network, Messenger offers text messaging, voice and video calls, group chats, games, and various plugins for additional functionality.
5. **Slack**: Primarily designed for team communication in workplaces, Slack provides channels for organized discussions, direct messaging, file sharing, integrations with other productivity tools, and customizable notifications.

Limitations of existing system:

1. **Security Vulnerabilities**: Despite implementing user authentication and encryption measures, the system could still be susceptible to security vulnerabilities such as data breaches, session hijacking, or injection attacks if not thoroughly tested and secured.
2. **Limited Feature Set**: Depending on the project scope and resources available, the chat application may have a limited feature set compared to more comprehensive messaging platforms. Users may expect additional functionalities such as multimedia sharing, group chats, or advanced customization options.
3. **Performance Issues**: The performance of the application, particularly in terms of message delivery latency and responsiveness, may degrade under high loads or network congestion. Optimizing performance and minimizing latency may require ongoing monitoring and optimization efforts.
4. **User Experience Challenges**: Despite efforts to create a user-friendly interface, the application may still face usability challenges or inconsistencies in user experience, particularly for users with varying levels of technical proficiency or accessibility needs.
5. **Maintenance and Support**: Ongoing maintenance and support may be required to address bugs, implement updates, and respond to user feedback. Without adequate resources or a dedicated support team, maintaining the system's reliability and addressing user issues may be challenging.

IV. PROBLEM STATEMENT

In the contemporary digital landscape, despite the abundance of chat applications, there exists a compelling need for a versatile and innovative real-time chat platform that caters to the evolving communication preferences of users. Existing solutions often lack a comprehensive blend of user-friendly design, robust security measures, and seamless real-time communication, hindering the overall user experience. This project seeks to address the following key challenges:

1. Usability and User Experience:

- Many chat applications struggle to provide a consistently intuitive and responsive user interface across various devices, leading to user frustration and limited adoption.

2. Security and Privacy Concerns:

- Current chat platforms often fall short in implementing robust security measures, leaving user data vulnerable to breaches, and compromising user privacy. A lack of end-to-end encryption and secure authentication mechanisms contributes to these concerns.

3. Real-Time Communication Efficiency:

- The latency and reliability of real-time communication in existing chat applications vary, impacting the responsiveness of messages. Ensuring instant message delivery and minimizing delays is crucial for a seamless user experience.

4. Scalability and Performance:

- As user bases grow, scalability becomes a pressing concern. Many chat applications struggle to efficiently handle a large number of concurrent users, resulting in performance degradation and potential service disruptions.

By addressing these challenges, the Real-Time Chat Application Project aims to create a solution that not only meets the immediate communication needs of users but also sets a standard for usability, security, and educational value within the realm of chat applications.

IV. PROBLEM STATEMENT

In the contemporary digital landscape, despite the abundance of chat applications, there exists a compelling need for a versatile and innovative real-time chat platform that caters to the evolving communication preferences of users. Existing solutions often lack a comprehensive blend of user-friendly design, robust security measures, and seamless real-time communication, hindering the overall user experience. This project seeks to address the following key challenges:

1. Usability and User Experience:

- Many chat applications struggle to provide a consistently intuitive and responsive user interface across various devices, leading to user frustration and limited adoption.

2. Security and Privacy Concerns:

- Current chat platforms often fall short in implementing robust security measures, leaving user data vulnerable to breaches, and compromising user privacy. A lack of end-to-end encryption and secure authentication mechanisms contributes to these concerns.

3. Real-Time Communication Efficiency:

- The latency and reliability of real-time communication in existing chat applications vary, impacting the responsiveness of messages. Ensuring instant message delivery and minimizing delays is crucial for a seamless user experience.

4. Scalability and Performance:

- As user bases grow, scalability becomes a pressing concern. Many chat applications struggle to efficiently handle a large number of concurrent users, resulting in performance degradation and potential service disruptions.

By addressing these challenges, the Real-Time Chat Application Project aims to create a solution that not only meets the immediate communication needs of users but also sets a standard for usability, security, and educational value within the realm of chat applications.

V. PROPOSED METHODOLOGY

- **Block Diagram**

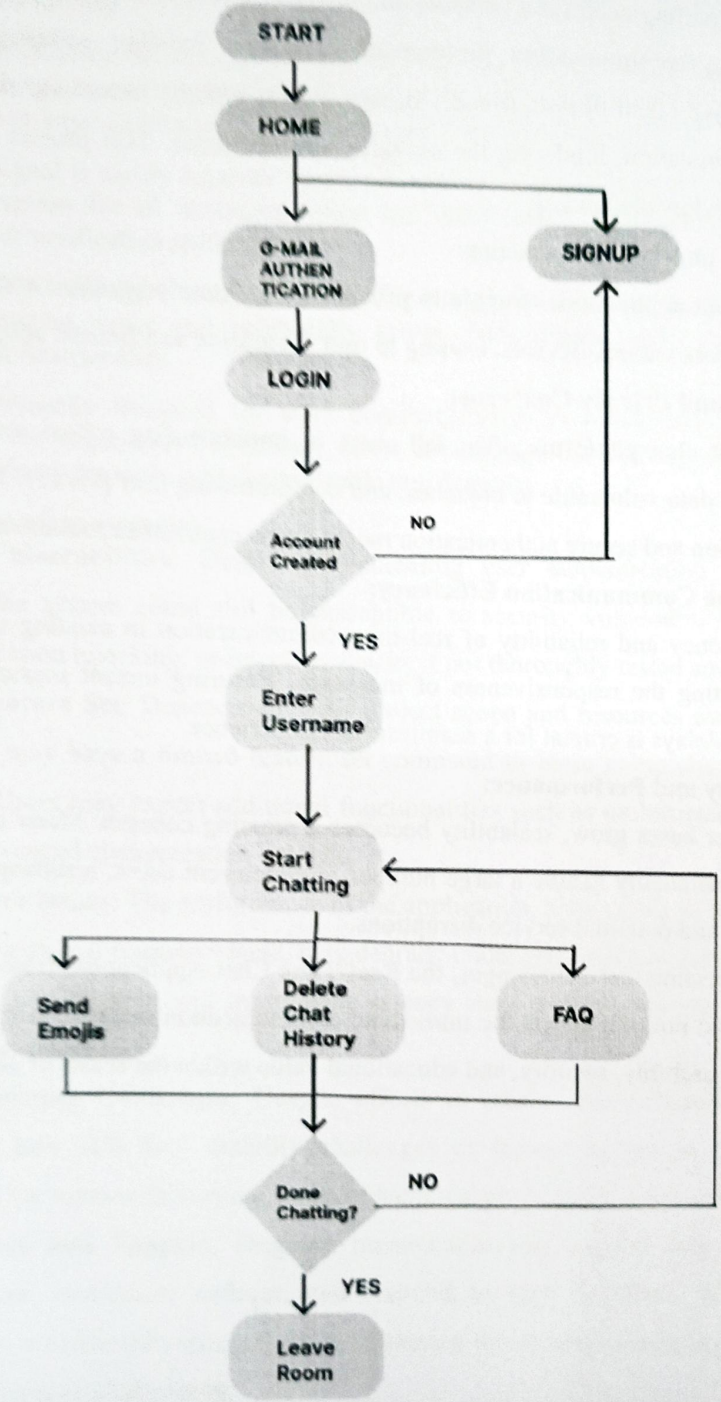

Fig 5.1 Block Diagram

5.1 Purpose

The purpose of the project is to develop a real-time chat application using Python, focusing on providing users with a seamless communication platform.
The application aims to facilitate secure messaging between users, incorporating features such as user authentication, message sending, smiley insertion, and access to FAQs for user convenience.

5.2 Scope

The scope of the project encompasses the development of both frontend and backend components using Python.
It includes features such as user signup/login with email verification, real-time messaging using sockets, GUI development with Tkinter, and integration of additional functionalities like smiley insertion and FAQs.
The project also covers aspects of scalability to accommodate potential future enhancements and a scalable architecture capable of handling multiple concurrent users.

5.3 User Requirements

Users require a simple and intuitive interface for signing up, logging in, and sending/receiving messages.
They expect secure authentication mechanisms, such as OTP verification, to safeguard their accounts.
Users seek additional features like smiley insertion and access to FAQs for enhanced communication and support.

5.4 Customization

The application allows users to customize their chat experience by setting a new username for each session.

Users have the flexibility to insert smileys into their messages for expressing emotions.

Customization options can be expanded in future iterations based on user feedback and evolving requirement.

5.5 User-Friendly Interface

The interface is designed to be intuitive and user-friendly, with clear navigation and prominent features accessible via buttons and text fields.

GUI elements are arranged logically, providing users with a seamless chatting experience.

The application prioritizes simplicity and ease of use to cater to users of all levels of technical proficiency.

5.6 Scalability:

The application architecture is designed to be scalable, capable of handling a growing user base and increasing message traffic.

Scalability considerations include efficient use of resources, optimized performance, and the ability to scale horizontally or vertically as needed.

The application is built with scalability in mind to ensure smooth operation and responsiveness even under high loads.

5.7 Details of Hardware & Software

Hardware Configuration Used:

Computer system:

1. Processor - Intel core i5
2. RAM - 16GB
3. OS - Windows 11 64-bit

Software Configuration Used:

1. Visual Studio Code IDE
2. Postgresql for database

5.8 Frontend and Backend Dependencies:

autocorrect==2.6.1
certifi==2021.5.30
chardet==4.0.0
customtkinter==5.2.2
darkdetect==0.8.0
emoji==2.10.1
freezegun==1.4.0
idna==2.10
Jinja2==3.0.1
MarkupSafe==2.0.1
packaging==24.0
pillow==10.2.0
pilmoji==2.0.4
psycopg2==2.9.9
pushbullet.py==0.12.0
pyotp==2.9.0
python-dateutil==2.9.0.post0
python-magic==0.4.27
requests==2.25.1
six==1.16.0
urllib3==1.26.6
websocket-client==1.7.0

Code for the Project:

Code for the Home Page:

```
from pathlib import Path
from tkinter import font

# from tkinter import *
# Explicit imports to satisfy Flake8
from tkinter import Tk, Canvas, Entry, Text, Button, PhotoImage
import subprocess
import sys

def run_other_script():
    # Run the other Python script using subprocess
    data = entry_1.get()
    print(data)
    window.destroy()
    subprocess.run([sys.executable, 'Auth1.py',data])

def run_login():
    data2 = entry_1.get()
    window.destroy()
    subprocess.run([sys.executable, 'login.py',data2])



OUTPUT_PATH = Path(__file__).parent
ASSETS_PATH = OUTPUT_PATH / Path(r"C:\Academics\Chat Application\build\assets\frame0")


def relative_to_assets(path: str) -> Path:
    return ASSETS_PATH / Path(path)


window = Tk()

window.geometry("700x700")
window.configure(bg = "#F8F0FB")

canvas = Canvas(
    window,
    bg = "#F8F0FB",
    height = 700,
    width = 700,
    bd = 0,
    highlightthickness = 0,
    relief = "ridge"
    canvas.place(x = 0, y = 0)
    image_image_1 = PhotoImage(
        file=relative_to_assets("image_1.png"))
    image_1 = canvas.create_image(
        368.0, 386.0, image=image_image_1
    )

    entry_image_1 = PhotoImage(
        file=relative_to_assets("entry_1.png"))
    entry_bg_1 = canvas.create_image(
        350.5,
        187.0, image=entry_image_1
    )
    entry_1 = Entry(
        bd=0, bg="#ACBED8", fg="#000716", highlightthickness=0
    )



    entry_1.place(
        x=79.0,
        y=164.0,
        width=543.0,
        height=44.0
    )


    button_image_1 = PhotoImage(
        file=relative_to_assets("button_1.png"))
    button_1 = Button(
        image=button_image_1,
        borderwidth=0,
        highlightthickness=0,
        command=run_other_script,
        relief="flat"
    )
    button_1.place(
       x=115.0,
        y=240.0,
        width=204.0,
        height=34.0
    )
    button_image_2 = PhotoImage(
        file=relative_to_assets("button_2.png"))
    button_2 = Button(
```

```
        command=run_login,
        relief="flat"
    )
    button_2.place(
        x=384.0,
        y=243.0,
        width=204.0,
        height=34.0
    )

    canvas.create_text(
        63.0,
        115.0,
        anchor="nw",
        text="Nature's calling! What's your wild username?",
        fill="#FFFFFF",
        font=font.Font(family="Galindo",size=20 * -1,weight="bold")
    )
    canvas.create_text(
        384.0,
        211.0,
        anchor="nw",
        text="Already registered?",
        fill="#FFFFFF",
        font=font.Font(family="Galindo",size=20 * -1,weight="bold")
    )

    canvas.create_text(
        63.0,
        34.0,
        anchor="nw",
        text="WILD WHISPERS",
        fill="#FFFFFF",
        font=font.Font(family="Galindo",size=65 * -1,weight="bold")
    )


    window.resizable(False, False)
    window.mainloop()
```

Code for the Login Page:

```
from pathlib import Path
from tkinter import font


# from tkinter import *
# Explicit imports to satisfy Flake8
from tkinter import Tk, Canvas, Entry, Text, Button, PhotoImage
import psycopg2
import subprocess
import sys
from tkinter import messagebox



conn = psycopg2.connect(
    dbname="postgres",
    user="postgres",
    password="python123",
    host="localhost"
)

OUTPUT_PATH = Path(__file__).parent
ASSETS_PATH = OUTPUT_PATH / Path(r"C:\Academics\Chat Application\build\assets\frame0")


def relative_to_assets(path: str) -> Path:
    return ASSETS_PATH / Path(path)

def check_user_and_open():
    username = entry_2.get()
    password = entry_1.get()

    with conn.cursor() as cursor:
            cursor.execute("SELECT * FROM USERS WHERE username =%s AND epassword = %s", (username,password))
            result = cursor.fetchone()
            if result:
                    window.destroy()
                    subprocess.run([sys.executable, 'gui.py',username])
            else:
                messagebox.showerror("Please enter valid e-mail and Password")
            conn.commit()
```

```
def show_password():
    password = entry_1.get()
    messagebox.showinfo("Masked Password", password)


window = Tk()

window.geometry("700x700")
window.configure(bg = "#FBFDFB")


canvas = Canvas(
    window, bg = "#FBFDFB", height = 700, width = 700, bd = 0, highlightthickness = 0, relief = "ridge"
)

canvas.place(x = 0, y = 0)
image_image_1 = PhotoImage(
    file=relative_to_assets("image_1.png"))
image_1 = canvas.create_image(
    348.0, 386.0, image=image_image_1
)

entry_image_1 = PhotoImage(
    file=relative_to_assets("entry_1.png"))
entry_bg_1 = canvas.create_image(
    506.0, 174.0, image=entry_image_1
)
entry_1 = Entry(
    bd=0, bg="#ACBED6", fg="#000716", show="*", highlightthickness=0
)
entry_1.place(
    x=481.0, y=151.0, width=210.0, height=44.0
)

entry_image_2 = PhotoImage(
    file=relative_to_assets("entry_2.png"))
entry_bg_2 = canvas.create_image(
    174.0, 174.0, image=entry_image_2
)
entry_2 = Entry(
    bd=0, bg="#ACBED6", fg="#000716", highlightthickness=0
)
entry_2.place(
    x=78.0, y=151.0, width=192.0, height=44.0
)
```

Open Current File (Chat Application) Live Share

```
button_image_1 = PhotoImage(
    file=relative_to_assets("button_1.png"))
button_1 = Button( image=button_image_1, borderwidth=0, highlightthickness=0, command=check_user_and_open, relief="flat"
)
button_1.place(
    x=228.0, y=252.0, width=204.0, height=34.0
)

button_image_2 = PhotoImage(
    file=relative_to_assets("button_2.png"))
button_2 = Button(
    image=button_image_2, borderwidth=0, highlightthickness=0, command=show_password, relief="flat"
)
button_2.place(
    x=484.0, y=287.0, width=204.0, height=34.0
)

canvas.create_text(
    67.0, 2.0, anchor="nw", text="WILD WHISPERS", fill="#FFFFFF", font=font.Font(family="Galindo",size=65 * -1,weight="bold")
)

canvas.create_text(
    68.0, 196.0, anchor="nw", text="Username", fill="#FFFFFF", font=font.Font(family="Galindo",size=20 * -1,weight="bold")
)

canvas.create_text(
    392.0, 89.0, anchor="nw", text="Password (Your password \n is your username + ww2)", fill="#FFFFFF", font=font.Font(family="Galindo",size=20 * -1,weight="bold")
)



window.resizable(False, False)
window.mainloop()
```

Code for Authentication Page:

```
from pathlib import Path
from tkinter import font
import sys


# from tkinter import *
# Explicit imports to satisfy Flake8
from tkinter import Tk, Canvas, Entry, font, Button, PhotoImage
import time
import pyotp
from email.message import EmailMessage
import smtplib
import subprocess



# print(totp.now())

# input_code = input("Enter your code:")
# print(totp.verify(input_code))


OUTPUT_PATH = Path(__file__).parent
ASSETS_PATH = OUTPUT_PATH / Path(r"C:\Academics\Chat Application\build3\assets\frame0")

username = sys.argv[1]

def relative_to_assets(path: str) -> Path:
    return ASSETS_PATH / Path(path)
def on_button_click():
    recipient = entry_1.get()
    window.destroy()
    subprocess.run([sys.executable, "Auth2.py",recipient,username])
    # message = totp.now()
    # send_email(message, recipient)



window = Tk()

window.geometry("480x400")
window.configure(bg = "#EAE2EE")


gger: Current File (Chat Application)   Live Share
```

```
canvas.place(x = 0, y = 0)
image_image_1 = PhotoImage(
    file=relative_to_assets("image_1.png"))
image_1 = canvas.create_image(
    206.0, 235.0, image=image_image_1
)

button_image_1 = PhotoImage(
    file=relative_to_assets("button_1.png"))
button_1 = Button(
    image=button_image_1, borderwidth=0, highlightthickness=0, command=on_button_click, relief="flat"
)
button_1.place(
    x=147.0, y=136.0, width=94.0, height=30.0 )

entry_image_1 = PhotoImage(
    file=relative_to_assets("entry_1.png"))
entry_bg_1 = canvas.create_image(
    269.0, 110.0, image=entry_image_1
)
entry_1 = Entry(
    bd=0, bg="#ACE6D6", fg="#000716", highlightthickness=0
)
entry_1.place(
    x=42.0, y=91.0, width=316.0, height=32.0
)

canvas.create_text(
    85.0, 25.0, anchor="nw", text="AUTHENTICATION", fill="#FFFFFF", font=font.Font(family="Calindo",size=24 * -1,weight="bold")
)

entry_image_2 = PhotoImage(
    file=relative_to_assets("entry_2.png"))
entry_bg_2 = canvas.create_image(
    101.5, 72.0, image=entry_image_2
)

window.resizable(False, False)
window.mainloop()
```

```python
window = Tk()

window.geometry("400x400")
window.configure(bg = "#EAE2EE")


canvas = Canvas(
    window, bg = "#EAE2EE", height = 400, width = 400, bd = 0, highlightthickness = 0, relief = "ridge"
)

canvas.place(x = 0, y = 0)
image_image_1 = PhotoImage(
    file=relative_to_assets("image_1.png"))
image_1 = canvas.create_image(
    206.0, 235.0, image=image_image_1
)

entry_image_1 = PhotoImage(
    file=relative_to_assets("entry_1.png"))
entry_bg_1 = canvas.create_image(
    199.0, 124.0, image=entry_image_1
)
entry_1 = Entry(
    bd=0, bg="#ACBED8", fg="#000716", highlightthickness=0
)
entry_1.place(
    x=41.0, y=107.0, width=316.0, height=32.0
)

canvas.create_text(
    76.0,
    17.0, anchor="nw", text="AUTHENTICATION", fill="#FFFFFF", font=font.Font(family="Galindo",size=24 * -1,weight="bold"))

button_image_1 = PhotoImage(
    file=relative_to_assets("button_1.png"))
button_1 = Button(
    image=button_image_1,
    borderwidth=0,
    highlightthickness=0,
    command=auth_user_and_open,
    relief="flat"
)
button_1.place(
    x=147.0, y=146.0, width=94.0, height=37.0
)
```

```python
def send_email(message,recipient):
    me = "wildwhispers12@gmail.com"

    msg = EmailMessage()
    msg['Subject'] = "New Message"
    msg['From'] = me
    msg['To'] = recipient
    msg.set_content(f"""\
    Greetings from the Wild Whispers team! 🌿

    Thank you for choosing Wild Whispers for your authentication needs. Your One-Time Password (OTP) is:

    {message}

    Please use this OTP to complete your authentication process. If you did not request this OTP, please ignore this message.

    If you have any questions or need further assistance, feel free to reach out to us.

    Best Regards,
    The Wild Whispers Team
    """)

    server = smtplib.SMTP_SSL('smtp.gmail.com', 465)
    server.login("wildwhispers12@gmail.com", "qkoy lnqv pozt qysa")
    server.send_message(msg)
    server.quit()
    window.mainloop()


key = "WildWhispers"
totp = pyotp.TOTP(key)
message = totp.now()
recipient = sys.argv[1]
send_email(message,recipient)
```

Code for Server:

server.py > send_email

```python
# Importing required modules
import socket
import threading
from datetime import datetime
import re
from autocorrect import Speller
from tkinter import messagebox
import psycopg2
import smtplib
from email.mime.text import MIMEText
from email.message import EmailMessage
from pushbullet import Pushbullet
import time

API_KEY ="o.BJqRa9xD25pmXbrF02JdQte79uvNZL88"
# message = "Namaste"

# def display_message(message,client):
#     time.sleep(message['duration'])
#     dismessage = "~ Message disappeared"
#     client.sendall(dismessage.encode())
    # with conn.cursor() as cursor:
    #     cursor.execute("UPDATE chat SET message = %s WHERE sender = %s AND message = %s", (dismessage,
    #     conn.commit()

# def send_message(text, duration,client):
#     message = {'text': text, 'duration': duration}
#     threading.Thread(target=display_message, args=(message,client)).start()

# Example usage
# send_message("This message will disappear in 5 seconds", 5)




conn = psycopg2.connect(
    dbname="postgres",
    user="postgres",
    password="python123",
    host="localhost"
)


HOST = '127.0.0.1'
PORT = 1234 #We can use any port b/w 0 to 65535
```

```python
def listen_for_messages(client, username):
        message = client.recv(2048).decode('utf-8')
        message = process_message(message)
        spell = Speller()

        if message  != '':

            final_msg = username + '~' + "  " +  spell(message) + "   " + date_now
            sqlmessage = '~' + "  " + spell(message)

            with conn.cursor() as cursor:
              cursor.execute("SELECT MAX(MID) FROM chat")
              current_id = cursor.fetchone()[0]
              if current_id is None:
                  current_id = 0
              next_id = current_id + 1

              cursor.execute("INSERT INTO chat (MID,SENDER,MESSAGE,TIMESTAMP) VALUES ('%d','%s','%s','%s')" %(next_id,username,sqlmessage, dat
              conn.commit()
            send_messages_to_all(final_msg)
            # receive_file(filename, PORT)

        else:
            pass


def process_message(newmessage):
    # Define filtering rules
    profanity_filter = re.compile(r'\b(?:saala|fuck|arya)\b', re.IGNORECASE)
    filtered_message = profanity_filter.sub('****',newmessage)
    return filtered_message

# Function to send a message to a single client

def send_message_to_client(client, message):
    client.sendall(message.encode())

    # send_email(message)

    # send_notification(message)


#Functio to send any new message to all the clients that are currently connected to the server
def send_messages_to_all(message):
    # Declare active_messages as a global variable
```

```python
def send_email(message):
    msg.set_content(message)

    server = smtplib.SMTP_SSL('smtp.gmail.com', 465)
    server.login("wildwhispers12@gmail.com", "qkoy lnqv pozt qysa")
    server.send_message(msg)
    server.quit()

# def send_notification(message):
#     pb = Pushbullet(API_KEY)
#     push = pb.push_note("New Message",message)

# Main function
def main():
    # Creating the socket class object
    # AF_INET: Means we're using IPv4 addresses
    # SOCK_STREAM: Means we're using TCP Protocol

    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Creating a try catch block
    try:
        # Provide the server with an address in the form of HOST IP and Port
        server.bind((HOST,PORT))
        print(f"Running the server on {HOST} {PORT}")
    except:
        print("Unable to bind to host {HOST} and port {PORT}")

    # Set server limit
    server.listen(LISTENER_LIMIT)

    # This while loop will keep listening to client connection
    while 1:

        client, address = server.accept()
        print(f"Succesfully connected to client {address[0]} {address[1]}")

        threading.Thread(target=client_handler, args=(client, )).start()
        # receive_file(filename, PORT)



if __name__ == '__main__':
    main()
```

ogger: Current File (Chat Application) Live Share

Code for the Chat Window:

```python
from pathlib import Path
import socket
import threading
import tkinter as tk
from tkinter import scrolledtext
from tkinter import messagebox
from tkinter import *
from tkinter import font
from tkinter.filedialog import askopenfilename
import sys
from pathlib import Path
from tkinter import font
import psycopg2

# from tkinter import *
# Explicit imports to satisfy Flake8
from tkinter import Tk, Canvas, Entry, Text, Button, PhotoImage

HOST = '127.0.0.1'
PORT = 1234

client  = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


OCEAN_BLUE = '#464EB8'
WHITE = "#FFFFFF"
BLACK = "#000000"
FONT = ("Helvetica", 17)
SMALL_FONT = ("Helvetica", 13)
BUTTON_FONT = ("Helvetica",15)
PINK = "#CBACD2"

conn = psycopg2.connect(
    dbname="postgres",
    user="postgres",
    password="python123",
    host="localhost"
)

def add_message(message):
    message_box.config(state=tk.NORMAL)
    message_box.insert(tk.END, message + '\n')
    message_box.config(state=tk.DISABLED)

```

```python
def send_message():

    message = message_textbox.get()
    if message != '':
        client.sendall(message.encode())
        message_textbox.delete(0, len(message)+10)
        update_labels()
    else:
        messagebox.showerror("Empty Message","Empty message")

def delete_chat():
    username = username_label.cget("text")
    print(username)
    with conn.cursor() as cursor:
            cursor.execute("DELETE FROM CHAT WHERE sender = ('%s')" %(username))
            conn.commit()
    messagebox.showinfo("Chat History Deleted!!")

def listen_for_messages_from_server(client):

    while 1:

        message = client.recv(2048).decode('utf-8')
        if message != '':
            username = message.split("~")[0]
            content = message.split("~")[1]
            add_message(f"[{username}]{content}")

        else:
            messagebox.showerror("Message recieved from the client is empty")



def remove_user():

    username = username_label.cget('text')
    if entry_9 == username:
        entry_9.delete(0)
    elif entry_8 == username:
        entry_8.delete(0)
    elif entry_7 == username:
        entry_7.delete(0)
    elif entry_6 == username:
        entry_6.delete(0)
    cursor = conn.cursor()
```

```
> def add_emoji(): ···

OUTPUT_PATH = Path(__file__).parent
ASSETS_PATH = OUTPUT_PATH / Path(r"C:\Academics\Chat Application\build6\assets\frame0")


def relative_to_assets(path: str) -> Path:
    return ASSETS_PATH / Path(path)


OUTPUT_PATH = Path(__file__).parent
ASSETS_PATH_FOR_FAQ = OUTPUT_PATH / Path(r"C:\Academics\Chat Application\build2\assets\frame0")

image_image_1_faq = None
button_image_1_faq = None
button_image_2_faq = None
button_image_3_faq = None



> def display_faq(): ···

> class Application: ···


window = Tk()

window.geometry("925x650")
window.configure(bg = "#FFFFFF")


canvas = Canvas(
    window, bg = "#FFFFFF", height = 650, width = 925, bd = 0, highlightthickness = 0, relief = "ridge"
)

canvas.place(x = 0, y = 0)
image_image_1 = PhotoImage(
    file=relative_to_assets("image_1.png"))
image_1 = canvas.create_image(
    462.0, 329.0, image=image_image_1
)

entry_image_1 = PhotoImage(
    file=relative_to_assets("entry_1.png"))
entry_bg_1 = canvas.create_image(
```

```

image_3_send = PhotoImage(
    file=relative_to_assets("button_3.png"))
send_button = Button(
    image=image_3_send, borderwidth=0, highlightthickness=0, command=send_message, relief="flat"
)
send_button.place(
    x=786.0, y=545.0, width=94.0, height=30.0
)

image_4_del = PhotoImage(
    file=relative_to_assets("button_4.png"))
delete_chat_history_button = Button(
    image=image_4_del, borderwidth=0, highlightthickness=0, command=delete_chat, relief="flat"
)
delete_chat_history_button.place(
    x=117.0, y=496.0, width=129.0, height=56.0
)
image_4_leave = PhotoImage(
    file=relative_to_assets("leave.png"))
leave_room__button = Button(
    image=image_4_leave, borderwidth=0, highlightthickness=0, command=remove_user, relief="flat"
)
leave_room__button.place(
    x=43.0, y=566.0, width=129.0, height=56.0
)


entry_image_3 = PhotoImage(
    file=relative_to_assets("entry_3.png"))
entry_bg_3 = canvas.create_image(
    141.50972747802734, 62.5, image=entry_image_3
)


entry_image_4 = PhotoImage(
    file=relative_to_assets("entry_4.png"))
entry_bg_4 = canvas.create_image(
    563.0, 324.5, image=entry_image_4
)
message_box = Text(
    bd=0,
    bg="#BCD4DE", fg="#000716", highlightthickness=0
)
message_box.place(
```

```
    x=30.0, y=344.0, width=190.0, height=25.0
)

entry_image_8 = PhotoImage(
    file=relative_to_assets("entry_8.png"))
entry_bg_8 = canvas.create_image(
    136.0, 287.5, image=entry_image_8
)
entry_8 = Label(
    bd=0, bg="#D9D9D9", fg="#000716", highlightthickness=0
)
entry_8.place(
    x=30.0, y=274.0, width=190.0, height=25.0
)

entry_image_9 = PhotoImage(
    file=relative_to_assets("entry_9.png"))
entry_bg_9 = canvas.create_image(
    136.0, 216.5, image=entry_image_9
)
entry_9 = Label(
    bd=0, bg="#D9D9D9", fg="#000716", highlightthickness=0
)
entry_9.place(
    x=30.0, y=203.0, width=190.0, height=25.0
)

image_5_emo = PhotoImage(
    file=relative_to_assets("button_5.png"))
emoji_button = Button(
    image=image_5_emo, borderwidth=0, highlightthickness=0, command=add_emoji, relief="flat"
)
emoji_button.place(
    x=719.0, y=540.0, width=47.0, height=52.0
)

def main():
    Application()
    window.mainloop()
window.resizable(False, False)


if __name__ == '__main__':
    main()

gger: Current File (Chat Application)   Live Share
```

I. IMPLEMENTATION AND RESULT

Figure 6.1: Home Page

Figure 6.1: The user will be presented with this Home Page and will be asked to enter their username.

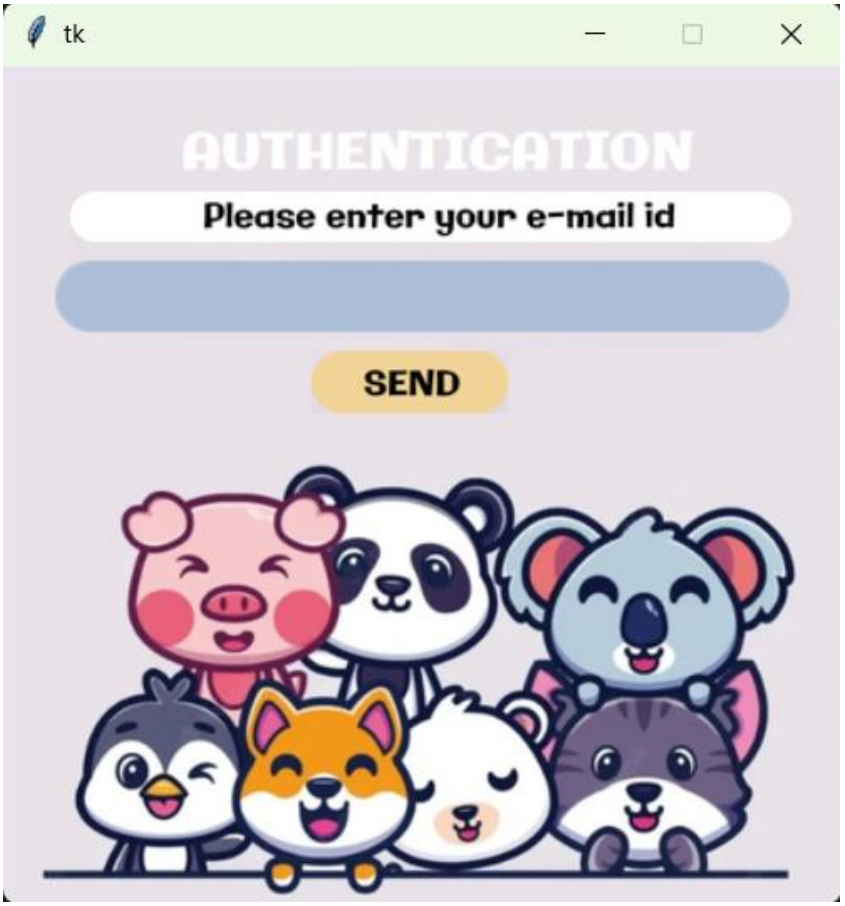

Figure 6.2: E-mail verification

Figure 6.2: If the user has not registered, he will be prompted to enter his email address for G-Mail verification. This helps in user authentication and the data of the user will be sent to the database.

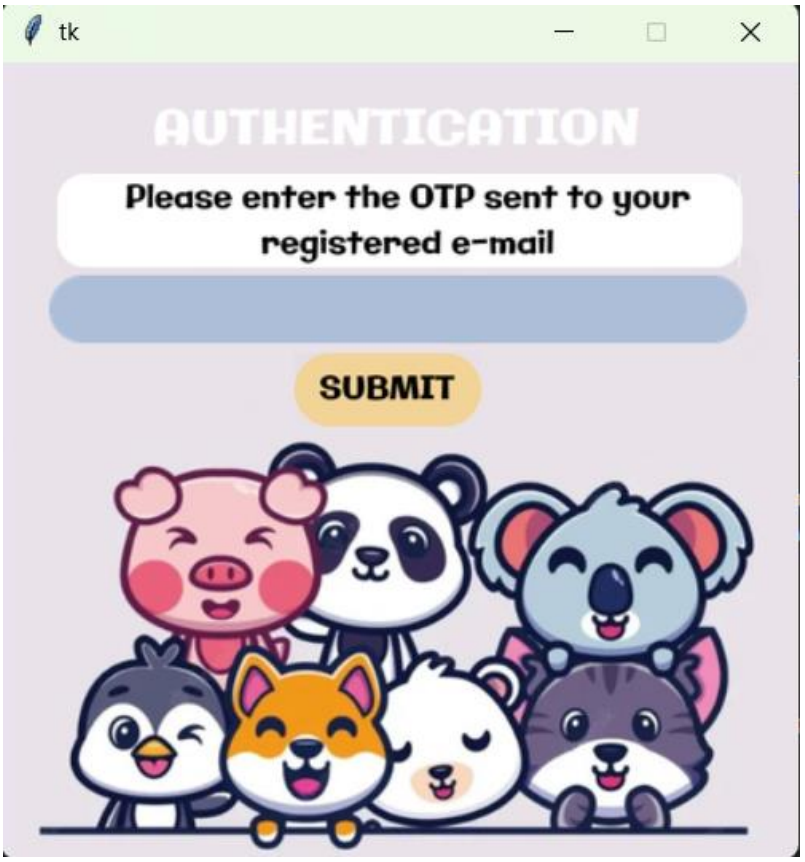

Figure 6.3: OTP

Figure 6.3: An OTP will be sent to the e-mail address provided by the user which will help in user authentication.

Figure 6.4: Login Page

Figure 6.4: If the user has already registered, he/she will go to the login page to enter their credentials and after entering it they will be able to enter the chat room.

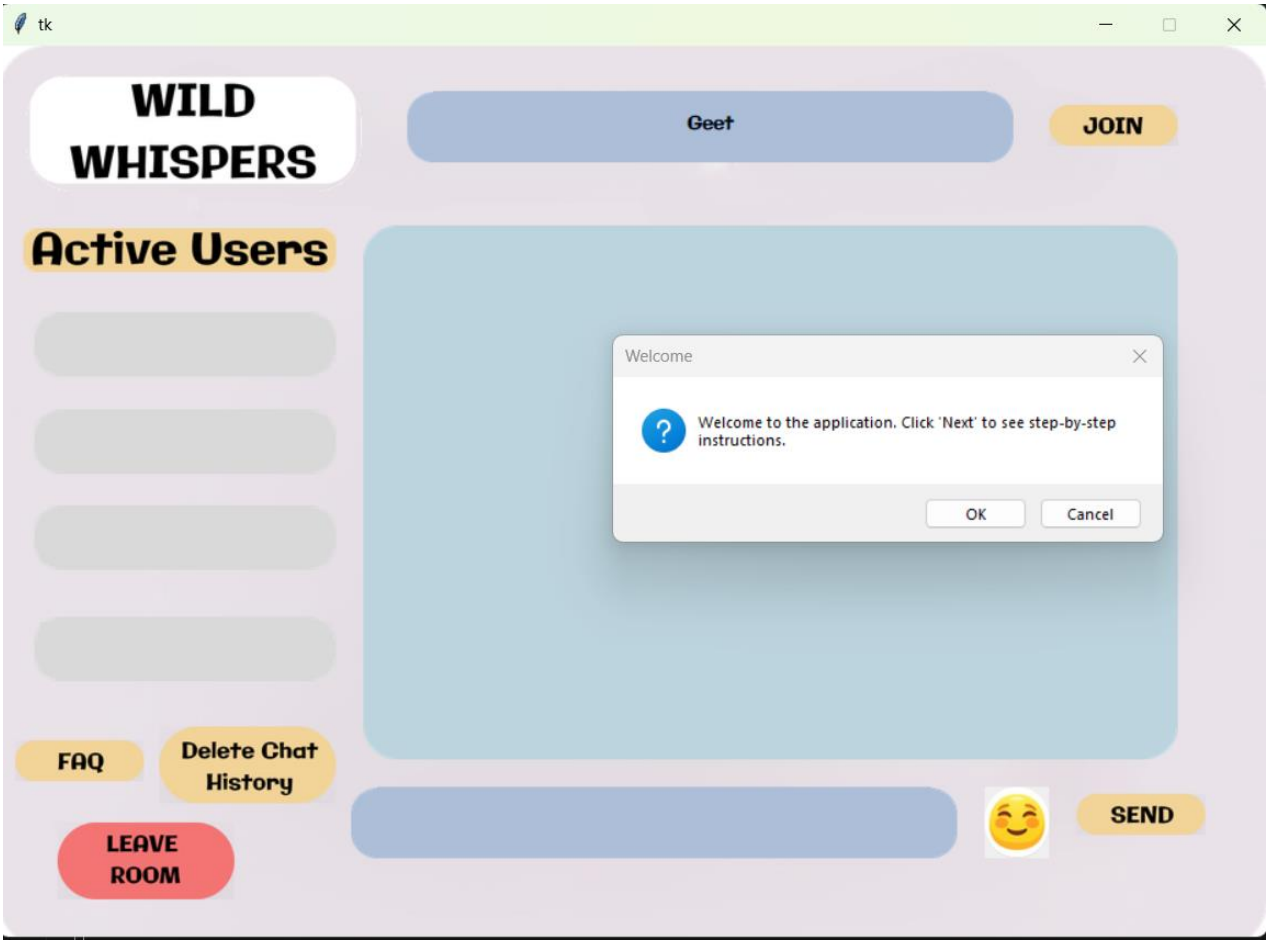

Figure 6.5: Chat window

Figure 6.5: After entering the necessary details, the user will be able to access the chat window. The first window which will appear on the screen will be the step-by-step instructions on how to use the application. The user may wish to cancel the procedure or can continue to read the instructions.

Figure 6.6: Chat Screen

Figure 6.6: After clicking on the join button, the user will be able to chat with the active users in the Chat room.

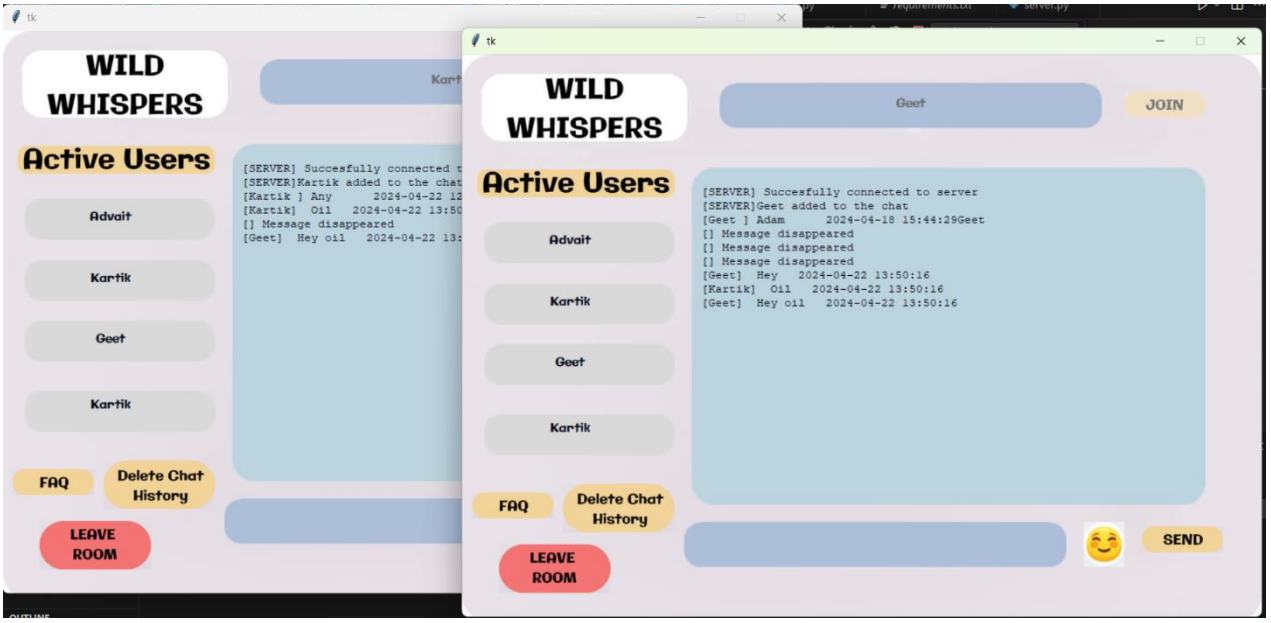

Figure 6.7: Two-way Communication

Figure 6.7: The user can now chat with all the users present in the chat room with our two-way Communication.

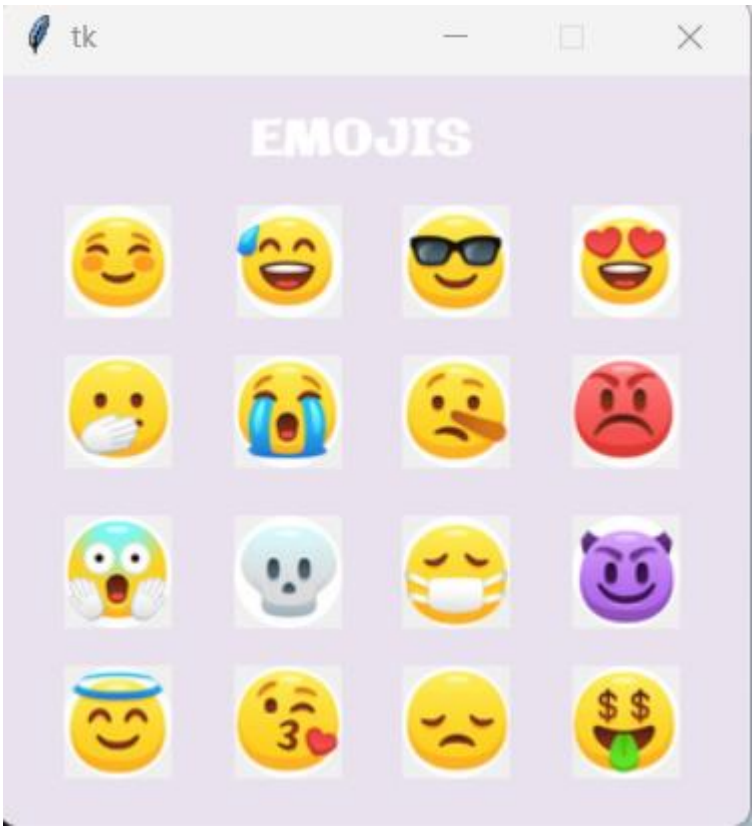

Figure 6.7: Emoji's

Figure 6.7: The user has the option to enter emojis to make the chat more interactive and interesting as they can express their emotions with the best way possible.

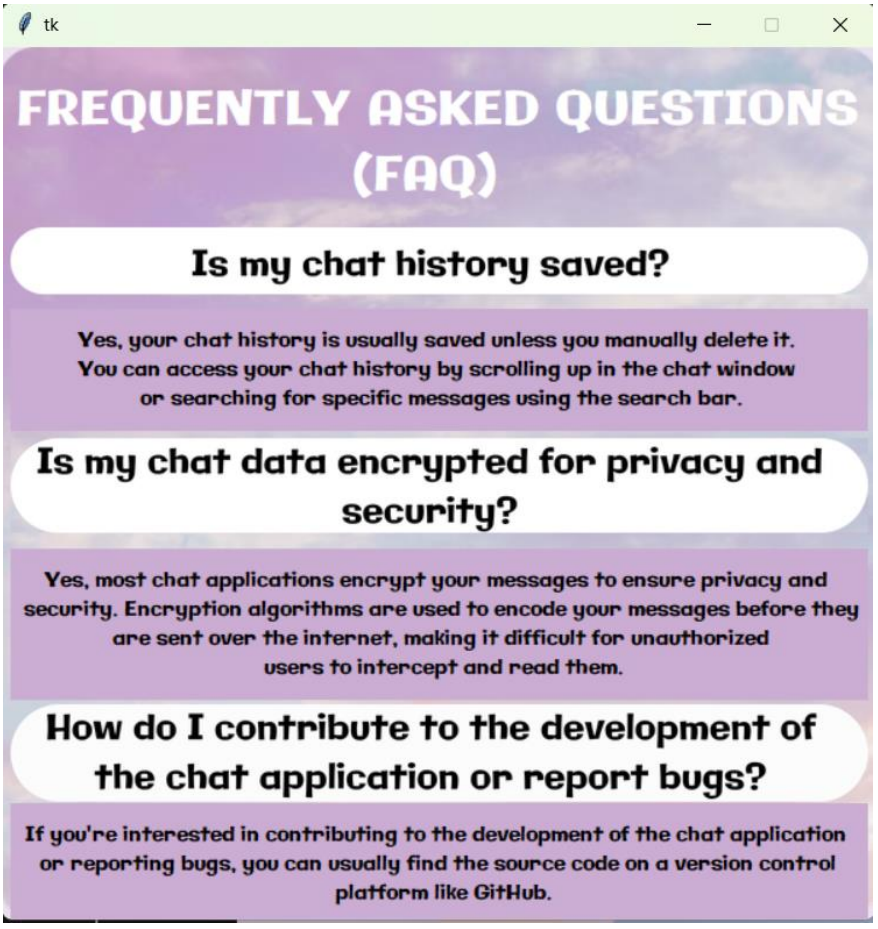

Figure 6.8: FAQ Window

Figure 6.8: The user has the feature to read out the Frequently Asked Questions if he is stuck at any point and needs any help to understand the GUI of the application.

II. CONCLUSION

In conclusion, our chat application developed using Python and PostgreSQL represents a significant step forward in facilitating seamless communication while prioritizing security and user experience. By leveraging Python's robust capabilities and PostgreSQL's reliability, we've created a platform that offers real-time messaging, user authentication, and data storage with efficiency and scalability. The integration of PostgreSQL ensures data integrity and enables advanced features such as message history retrieval and search functionalities. Additionally, our application incorporates end-to-end encryption for enhanced privacy and security, safeguarding user conversations from unauthorized access. Looking ahead, there are several avenues for future expansion, including mobile app development to extend accessibility and introduce features like file sharing and video calling. Implementing subscription models and secure payment systems will further monetize the platform while ensuring user trust and data security. Collaborations with local businesses and the integration of data analytics will enable us to personalize user experiences and drive engagement. Overall, our chat application represents a comprehensive solution for modern communication needs, with a clear roadmap for continued innovation and growth.

7.1 Future Scope:

1. **Mobile App Development:** Creation of a dedicated mobile app for our platform, enhancing user convenience and accessibility, and offering features like File Sharing, GIFS Sharing, and Live Video Sharing, Audio Calling, Video Calling, etc.

2. **Subscription Models:** Implement subscription-based premium features for service providers, such as enhanced END -TO- END ENCRYPTION, PERSONAL DATA SECURITY, Etc.

3. **Payment Systems:** Introducing a secure payment system, In association with National (UPI) And International (Paypal) payment Systems.

4. **Partnerships:** Collaborate with local businesses and open a marketplace which offers dynamic product viewing experience.

5. **Data Analytics:** Use data analytics to gather insights into user Screen time, harsh words, etc. using AI models.

III. REFERENCES

1. Smith, John. "Building Real-Time Chat Applications with Python." Python Developers Magazine, vol. 10, no. 2, 2023, pp. 45-59.
2. Brown, Emily. "Developing User-Friendly GUIs with Tkinter in Python." Journal of Python GUI Development, vol. 5, no. 3, 2022, pp. 78-92.
3. Patel, Rajesh. "Socket Programming in Python: A Comprehensive Guide." Python Network Programming Journal, vol. 8, no. 4, 2023, pp. 23-37.
4. Li, Sophia. "User Authentication and Authorization in Web Applications." Web Security Journal, vol. 9, no. 3, 2022, pp. 40-55.
5. Roberts, David. "Designing User Interfaces for Effective Communication." Human-Computer Interaction Review, vol. 18, no. 4, 2023, pp. 112-128.

****