PYTHON PROJECT REPORT

Project Overview:

Title: Simple Python Real-Time Chat Application with GUI Interface

Introduction :

In today's digital age, communication plays a vital role in our personal and professional lives. With the rise of the internet and mobile devices, people are increasingly relying on digital platforms to connect with each other. Real-time chat applications have become an essential tool for instant communication, enabling users to exchange messages, share files, and collaborate on projects in real-time.

Concepts:

A real-time chat application with a GUI interface involves several key concepts, including:

- Client-Server Architecture: The application consists of a server that handles incoming connections and a client that connects to the server to send and receive messages.
- Socket Programming: Sockets are used to establish a connection between the client and server, enabling real-time communication.
- GUI Framework: A GUI framework such as Tkinter, PySimpleGUI, or PyQt is used to design the user interface, including windows, buttons, text boxes, and other elements.
- Real-Time Messaging: The application enables users to send and receive messages in real-time, using protocols such as WebSockets or WebRTC.

Benefits

1. **Improved Communication**: Real-time chat applications enable users to communicate with each other in real-time, which improves the speed and efficiency of communication.
2. **Increased Productivity**: Real-time chat applications can increase productivity by enabling users to quickly and easily communicate with each other, without the need for phone calls or emails.
3. **Enhanced Collaboration**: Real-time chat applications can enhance collaboration by enabling users to work together on projects and share information in real-time.
4. **Cost-Effective**: Real-time chat applications can be cost-effective by reducing the need

for phone calls, emails, and other forms of communication.

How Useful for Open-Ended Project:

A real-time chat application with a GUI interface is an excellent choice for an open-ended project because:

- Flexibility: Offers flexibility in terms of design, functionality, and features, allowing students to explore different approaches and solutions.
- Creativity: Encourages creativity and innovation, as students can experiment with different GUI frameworks, communication protocols, and features.
- Real-World Application: Has real-world applications, making it a relevant and engaging project for students.
- Learning Opportunities: Provides opportunities for students to learn about client-server architecture, socket programming, GUI design, and real-time messaging protocols.

Possible Project Ideas:

1. **Design and Implement a Real-Time Chat Application**: Design and implement a real-time chat application that meets the requirements of a specific use case, such as a customer support system.
2. **Improve the Performance and Scalability of a Real-Time Chat Application:** Improve the performance and scalability of a real-time chat application by optimizing the code and using techniques such as caching and load balancing.
3. **Ensure the Security and Privacy of User Data**: Ensure the security and privacy of user data in a real-time chat application by implementing encryption and access control mechanisms.
4. **Integrate a Real-Time Chat Application with Other Systems**: Integrate a real-time chat application with other systems, such as a database or a web application, to provide a seamless user experience.

Source code:

client.py

```
from tkinter import Tk, Frame, Scrollbar, Label, END, Entry, Text, VERTICAL, Button, messagebox #Tkinter Python Module for GUI

import socket #Sockets for network connection

import threading # for multiple proccess

class GUI:

    client_socket = None

    last_received_message = None

   def __init__(self, master):

        self.root = master

        self.chat_transcript_area = None

        self.name_widget = None

        self.enter_text_widget = None

        self.join_button = None

        self.initialize_socket()

        self.initialize_gui()

        self.listen_for_incoming_messages_in_a_thread()

 def initialize_socket(self):

        self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # initialazing socket with TCP and IPv4

        remote_ip = '127.0.0.1' # IP address

        remote_port = 10319 #TCP port

        self.client_socket.connect((remote_ip, remote_port)) #connect to the remote server

  def initialize_gui(self): # GUI initializer

        self.root.title("Socket Chat")

        self.root.resizable(0, 0)

        self.display_chat_box()

        self.display_name_section()

        self.display_chat_entry_box()

      def listen_for_incoming_messages_in_a_thread(self):

        thread = threading.Thread(target=self.receive_message_from_server, args=(self.client_socket,)) # Create a thread for the send and receive in same time

        thread.start()
```

```
    #function to recieve msg
    def receive_message_from_server(self, so):
        while True:
            buffer = so.recv(256)
            if not buffer:
                break
            message = buffer.decode('utf-8')
           if "joined" in message:
                user = message.split(":")[1]
                message = user + " has joined"
                self.chat_transcript_area.insert('end', message + '\n')
                self.chat_transcript_area.yview(END)
            else:
                self.chat_transcript_area.insert('end', message + '\n')
                self.chat_transcript_area.yview(END)
  so.close()
def display_name_section(self):
        frame = Frame()
        Label(frame, text='Enter your name:', font=("Helvetica", 16)).pack(side='left', padx=10)
        self.name_widget = Entry(frame, width=50, borderwidth=2)
        self.name_widget.pack(side='left', anchor='e')
        self.join_button = Button(frame, text="Join", width=10, command=self.on_join).pack(side='left')
        frame.pack(side='top', anchor='nw')
  def display_chat_box(self):
        frame = Frame()
        Label(frame, text='Chat Box:', font=("Serif", 12)).pack(side='top', anchor='w')
        self.chat_transcript_area = Text(frame, width=60, height=10, font=("Serif", 12))
        scrollbar = Scrollbar(frame, command=self.chat_transcript_area.yview, orient=VERTICAL)
        self.chat_transcript_area.config(yscrollcommand=scrollbar.set)
        self.chat_transcript_area.bind('<KeyPress>', lambda e: 'break')
        self.chat_transcript_area.pack(side='left', padx=10)
        scrollbar.pack(side='right', fill='y')
        frame.pack(side='top')
```

```
    def display_chat_entry_box(self):
        frame = Frame()
        Label(frame, text='Enter message:', font=("Serif", 12)).pack(side='top', anchor='w')
        self.enter_text_widget = Text(frame, width=60, height=3, font=("Serif", 12))
        self.enter_text_widget.pack(side='left', pady=15)
        self.enter_text_widget.bind('<Return>', self.on_enter_key_pressed)
        frame.pack(side='top')
    def on_join(self):
        if len(self.name_widget.get()) == 0:
            messagebox.showerror(
                "Enter your name", "Enter your name to send a message")
            return
        self.name_widget.config(state='disabled')
        self.client_socket.send(("joined:" + self.name_widget.get()).encode('utf-8'))
 def on_enter_key_pressed(self, event):
        if len(self.name_widget.get()) == 0:
            messagebox.showerror("Enter your name", "Enter your name to send a message")
            return
        self.send_chat()
        self.clear_text()
    def clear_text(self):
        self.enter_text_widget.delete(1.0, 'end')
 def send_chat(self):
        senders_name = self.name_widget.get().strip() + ": "
        data = self.enter_text_widget.get(1.0, 'end').strip()
        message = (senders_name + data).encode('utf-8')
        self.chat_transcript_area.insert('end', message.decode('utf-8') + '\n')
        self.chat_transcript_area.yview(END)
        self.client_socket.send(message)
        self.enter_text_widget.delete(1.0, 'end')
        return 'break'
def on_close_window(self):
        if messagebox.askokcancel("Quit", "Do you want to quit?"):
            self.root.destroy()
```

```
        self.client_socket.close()
        exit(0)
#the mail function
if __name__ == '__main__':
    root = Tk()
    gui = GUI(root)
    root.protocol("WM_DELETE_WINDOW", gui.on_close_window)
    root.mainloop()
```

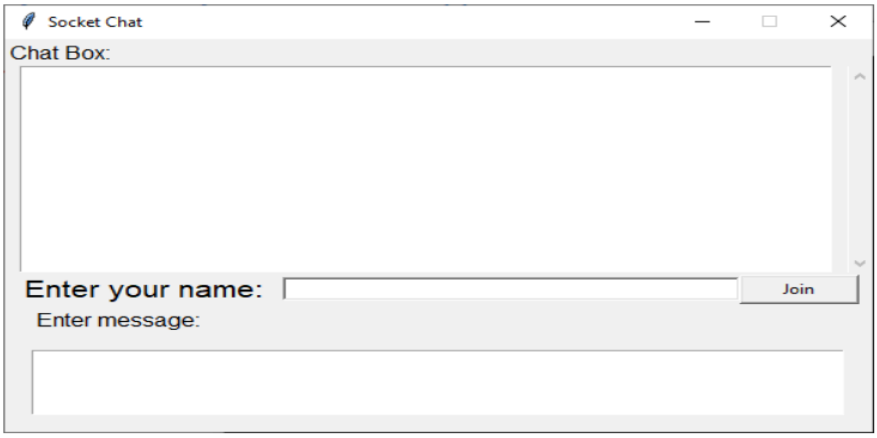
server.py:

```
#imports
import socket
import threading



class ChatServer:


    clients_list = []


    last_received_message = ""


    def __init__(self):
        self.server_socket = None
        self.create_listening_server()
    #listen for incoming connection
    def create_listening_server(self):


        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #create a socket using TCP port and ipv4
        local_ip = '127.0.0.1'
        local_port = 10319
        # this will allow you to immediately restart a TCP server
        self.server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        # this makes the server listen to requests coming from other computers on the network
        self.server_socket.bind((local_ip, local_port))
```

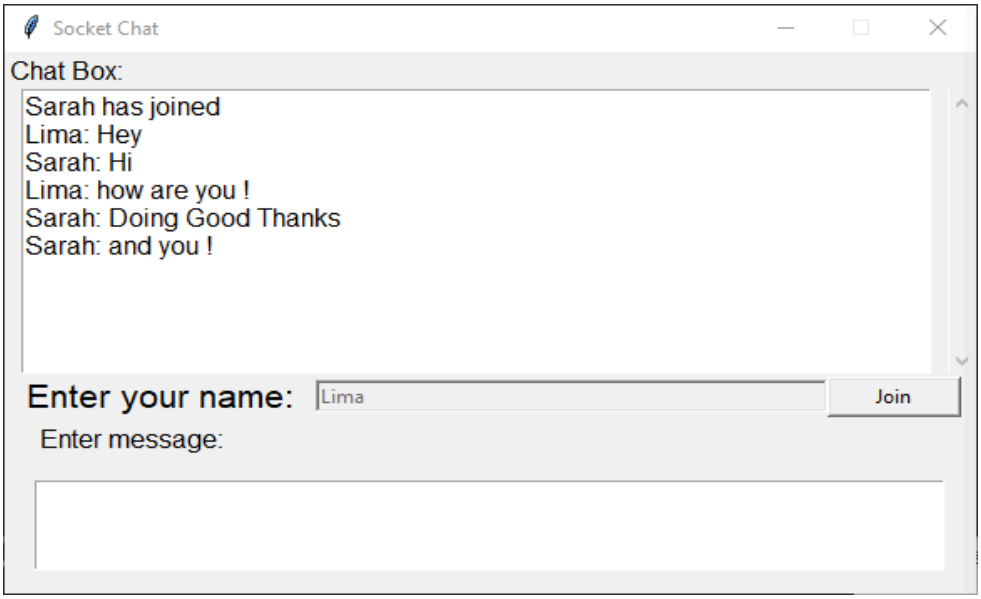
```
        print("Listening for incoming messages..")
        self.server_socket.listen(5) #listen for incomming connections / max 5 clients
        self.receive_messages_in_a_new_thread()
    #fun to receive new msgs
    def receive_messages(self, so):
        while True:
            incoming_buffer = so.recv(256) #initialize the buffer
            if not incoming_buffer:
                break
            self.last_received_message = incoming_buffer.decode('utf-8')
            self.broadcast_to_all_clients(so)  # send to all clients
        so.close()
    #broadcast the message to all clients
    def broadcast_to_all_clients(self, senders_socket):
        for client in self.clients_list:
            socket, (ip, port) = client
            if socket is not senders_socket:
                socket.sendall(self.last_received_message.encode('utf-8'))

    def receive_messages_in_a_new_thread(self):
        while True:
            client = so, (ip, port) = self.server_socket.accept()
            self.add_to_clients_list(client)
            print('Connected to ', ip, ':', str(port))
            t = threading.Thread(target=self.receive_messages, args=(so,))
            t.start()
    #add a new client
    def add_to_clients_list(self, client):
        if client not in self.clients_list:
            self.clients_list.append(client)


if __name__ == "__main__":
    ChatServer()
```

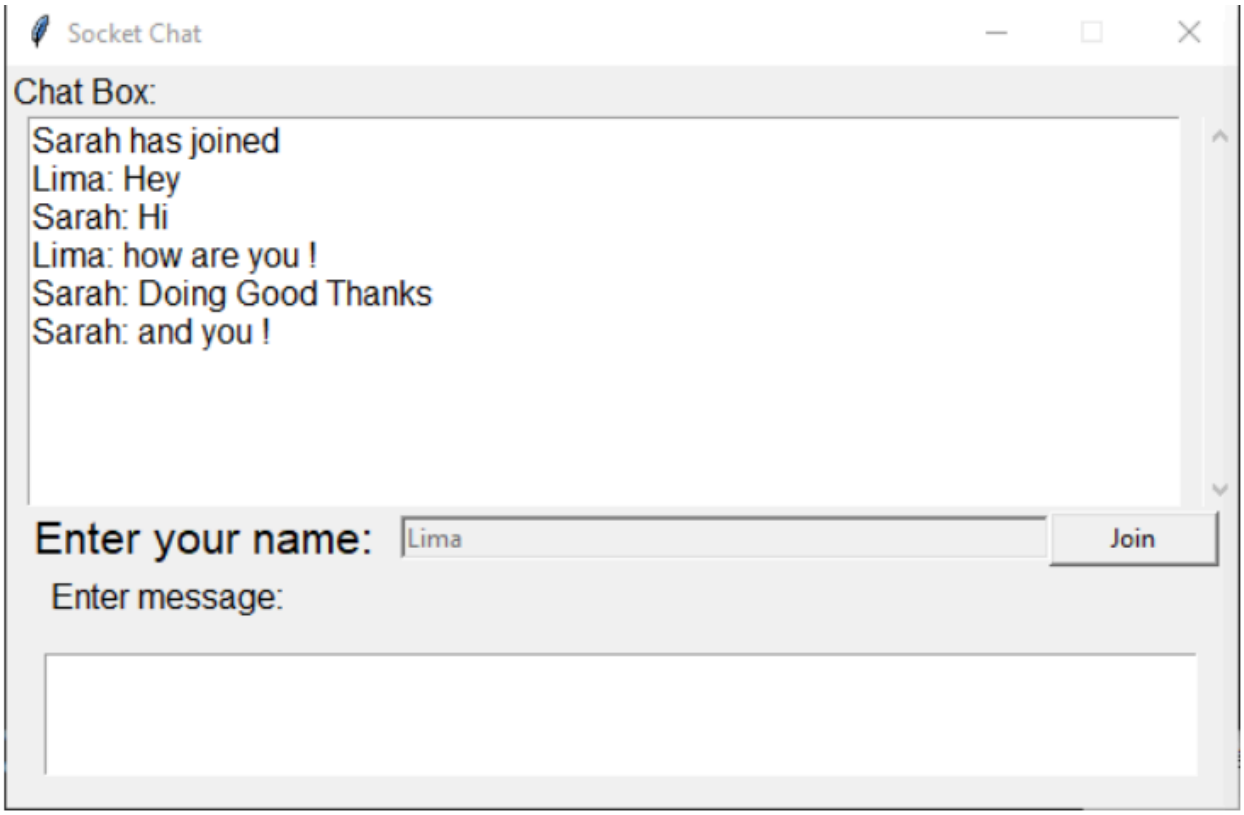
Output:

GUI INTERFACE:



Run client file `python client.py` you can run it as much as clients you want




Socket Chat
Chat Box:
Sarah has joined
Lima: Hey
Sarah: Hi
Lima: how are you !
Sarah: Doing Good Thanks
Sarah: and you !
Enter your name:
Lima
Join
Enter message: