|  |  |
| --- | --- |
|  | **DEPARTMENT OF COMPUTER ENGINEERING** |

**Mini Project Report**

|  |  |
| --- | --- |
| Semester | S.E. Semester IV – Computer Engineering |
| Subject | Skill Base Lab Course: Python Programming (CSL405) |
| Subject Professor In-charge | Prof. Swapnil S. Sonawane |
| Assisting Teachers | Prof. Swapnil S. Sonawane |

|  |  |
| --- | --- |
| Roll Numbers | Name of Students |
| 21102A0014 | Deep Salunkhe |
| 21102A0003 | Omkar Patil |
| 21102A0005 | Pranav Redij |
| 21102A0037 | Sukant Thombare |

**Name of the Project: Chat Application**

**Project Description:We have built a chat application project that allows multiple clients to connect to a server using sockets and exchange messages in real-time. The project is divided into two parts, the front-end and the back-end.**

**The front-end is developed using the tkinter module in Python, which provides a graphical user interface for the chat application. It allows the user to enter their name and message and send it to the server using a send button. The chat box displays all the messages received from the server in real-time.**

**The back-end is developed using Flask and the threading module in Python. It creates a server that listens for incoming client connections using sockets. Once a client connects, the server creates a new thread to handle the client connection. The server then broadcasts any message received from a client to all the other connected clients.**

**Project Code:**

**Client:**

import tkinter as tk

from tkinter import ttk

import threading

import socket

*# Create a socket object*

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

*# Define host and port for server*

HOST = 'localhost'

PORT = 5000

*# Connect to the server*

client\_socket.connect((HOST, PORT))

*# Define function to send message to server*

def send\_message():

    message = message\_entry.get()

    sender = sender\_entry.get()

    data = {'message': message, 'sender': sender}

    client\_socket.sendall(str(data).encode())

    message\_entry.delete(0, tk.END)

    if not sender\_entry['state'] == 'disabled':

        sender\_entry.config(state='disabled')

*# Create the GUI*

root = tk.Tk()

root.title('Chat App')

root.geometry('500x500')

root.configure(bg='black')

*# Create widgets*

sender\_label = ttk.Label(root, text='Name:', foreground='white', background='black')

sender\_entry = ttk.Entry(root)

message\_label = ttk.Label(root, text='Message:', foreground='white', background='black')

message\_entry = ttk.Entry(root)

send\_button = ttk.Button(root, text='Send', command=send\_message)

chat\_box = tk.Text(root, height=20, width=60, foreground='white', background='black')

scrollbar = ttk.Scrollbar(root, command=chat\_box.yview)

chat\_box.config(yscrollcommand=scrollbar.set, state='disabled')

*# Add widgets to grid*

sender\_label.grid(row=0, column=0, pady=10, padx=10)

sender\_entry.grid(row=0, column=1, pady=10, padx=10)

message\_label.grid(row=1, column=0, pady=10, padx=10)

message\_entry.grid(row=1, column=1, pady=10, padx=10)

send\_button.grid(row=2, column=1, pady=10, padx=10, sticky='e')

chat\_box.grid(row=3, column=0, columnspan=2, pady=10, padx=10)

scrollbar.grid(row=3, column=2, sticky='ns', pady=10)

*# Define function to update chat box with received messages*

def update\_chat\_box(message):

    chat\_box.config(state='normal')

    chat\_box.insert(tk.END, message + '\n')

    chat\_box.config(state='disabled')

*# Define function to receive messages from server*

def receive\_messages():

    while True:

        data = client\_socket.recv(1024)

        message = data.decode()

        update\_chat\_box(message)

*# Create a new thread to receive messages from server*

receive\_thread = threading.Thread(target=receive\_messages)

receive\_thread.start()

*# Run the GUI*

root.mainloop()

**Server:**

import socket

from threading import Thread

from flask import Flask, render\_template

*# Create a Flask app instance*

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY'] = 'secret!'

*# Define host and port for server*

HOST = 'localhost'

PORT = 5000

*# Create a socket object*

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

*# Set socket option to allow re-use of the address and port*

server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

*# Bind the socket to the host and port*

server\_socket.bind((HOST, PORT))

*# Listen for incoming client connections*

server\_socket.listen()

*# Define list of connected clients*

clients = []

*# Define function to broadcast message to all connected clients*

def broadcast\_message(sender, message):

    for client in clients:

        if client != sender:

            try:

*# Send the message to the client*

                client.sendall(f'{sender}: {message}'.encode())

            except socket.error:

*# If there's an error sending the message, remove the client from the list of connected clients*

                clients.remove(client)

                print(f'Client {client} disconnected')

*# Define function to handle client connection*

def handle\_client(client\_socket, addr):

    print(f'Client connected from {addr}')

*# Add the client socket to the list of connected clients*

    clients.append(client\_socket)

    while True:

*# Receive data from the client*

        data = client\_socket.recv(1024)

*# If there's no data, the client has disconnected*

        if not data:

*# Remove the client from the list of connected clients and close the connection*

            clients.remove(client\_socket)

            client\_socket.close()

            break

*# Decode the received data into a string*

        message = data.decode()

*# Broadcast the message to all connected clients*

        broadcast\_message(addr, message)

*# Define a route for the index page*

@app.route('/')

def index():

    return render\_template('index.html')

if \_\_name\_\_ == '\_\_main\_\_':

    print(f'Server running on {HOST}:{PORT}')

*# Listen for incoming client connections*

    while True:

*# Accept an incoming client connection and create a new thread to handle the connection*

        client\_socket, addr = server\_socket.accept()

        client\_thread = Thread(target=handle\_client, args=(client\_socket, addr))

        client\_thread.start()

**Result/ Output:**

Graphical user interface, text

Description automatically generated

**References:**

[**https://python.plainenglish.io/create-a-basic-lan-chat-room-with-python-f334776bf70c**](https://python.plainenglish.io/create-a-basic-lan-chat-room-with-python-f334776bf70c)

[**https://www.geeksforgeeks.org/simple-chat-room-using-python/**](https://www.geeksforgeeks.org/simple-chat-room-using-python/)