# KONNECT: A SOCKET PROGRAMMING BASED GROUP CHAT WEB APPLICATION

*A project report submitted in fulfilment of the requirements for the Mid-Semester Small Project in*

**COMPUTER NETWORKS (IT-303)**

*Submitted by*

**Aman Jha and Arjun Choudhry**
**{2K18/IT/018, 2K18/IT/031}**

*Under the Guidance of*

**Mrs. Anamika Chauhan**
**Assistant Professor**
**Department of Information Technology**
**Delhi Technological University**
**New Delhi-110042**

**DELHI TECHNOLOGICAL UNIVERSITY (FORMERLY DELHI COLLEGE OF ENGINEERING)**

## ACKNOWLEDGEMENTS

The successful completion of this project wouldn't have been possible without the help and support of many, and we would like to extend our sincere gratitude to them.

We are highly indebted to Mrs. Anamika Chauhan for being a source of constant motivation, and for helping us improve the project to achieve its present shape. We would also like to thank her for giving us valuable insights on Transmission Control Protocol and Internet Protocols.

It would be unfair, should we miss out on thanking our fellow classmates, for some wonderfully insightful discussions on Computer Networks, for working around us, and for making the subject even more interesting and for sharing many unique ideas, both on and off the subject.

Last but not the least we would like to thank our parents for always being there, and guiding us through this journey.

## ABSTRACT

Messaging apps, also known as chat apps or social apps, are applications and platforms that enable instant messaging. With a growing list of advanced features like Multimedia support, file transferring, emoticons, and even online money transfer, these applications are truly changing the way people interact with each other, one user at a time.

Today's world is built on communication between people, which to say the least, is becoming more and more important in our lives. Communication, rather instant transfer of information, is very important in every sphere of life, whether it is personal or professional.

With multitasking being a major focus for people, chat apps are utilized globally by billions of users for both personal as well as commercial uses. The popularity of chat apps has seen a major surge in the last couple of years days as people's preference has shifted towards chat-based apps from text messages, due to major advantages like real-time interaction, improved multi-media support and lower cost.

Being a relatively inexpensive service as compared to text messaging, most of the chat apps provide their service either at a minimal cost or completely free. WhatsApp is now delivering over a hundred billion messages a day to and from its users, and is used by over 2 billion people (all statistics have been taken from TechCrunch.com). Chinese alternative WeChat has over 1.2 billion users of its own, thus showing the rapid growth and demand for user-friendly chat applications.

# 1 INTRODUCTION

A socket-programming based chat application is a python-based application that enables two users to exchange data in the form of messages across a network using the **socket** module. It is able to send and receive messages with ease and does not require any third-party libraries, modules or applications.

A web chat application is a web-based application built using the **Flask framework** for back-end implementation and **JavaScript** for front-end Implementation. The application uses **SocketIO**, a cross-browser **JavaScript** library that emulates a persistent, bi-directional communication channel between the client and server.

Messaging can be achieved across networks by the process of transmitting bits from a source to destination location in the same network. The scope of this data transfer can be small (Local Area Network), MAN (Metropolitan Area Network) or WAN (Wide Area Network), i.e. from a small room to even the whole world.

In our project, we have made two applications that can conduct message transfer in the form of a chat room application as long as both sender and receiver are on the same network and responsive to the network, irrespective of the size and distance of nodes.

Every message transfer and for that matter any data transfer on a network must be governed by some set of rules which specify the packet behaviour and many more such parameters. These are called protocols. Both Application uses TCP (Transmission Control Protocol) to be able to send and receive data.

Thus, in this work, we describe our implementation of Client-Server Interactions based on the approach of socket programming. We create **Konnect**, a web application created on socket.io with bootstrap front-end using the Flask framework, HTML tags, and a standard Javascript template customized to our liking.

We also show the implementations of client-server connections using sockets in Python, which are capable of handling multiple clients by broadcasting messages between one-another, and show the peer-to-peer communication between a single client-server connection using a simple application built using sockets and the **tkinter** API.

# 2 METHODOLOGY

## 2.1 Client- Server Communication over the TCP/IP Protocol

A **Client-Server Network** is a computer network in which one centralized, powerful computer (the server) is a hub to which many personal computers or workstations (called clients) are connected. The client runs programs and accesses data that is stored on the server.

Clients and servers exchange messages in a request-response messaging pattern. The client sends a request, and the server returns a response. This exchange of messages is an example of inter-process communication. The architecture is referred to as **two-tier architecture** means that the clients acts as one tier and the server process acts as the other tier.

The **TCP/IP Model** is the acronym for the **Transmission Control Protocol/Internet Protocol** model, and consists of four layers:

- Process/Application Layer
- Host-to-Host/Transport Layer
- Internet Layer
- Network Access/Link Layer

The TCP protocol is responsible for end-to-end communication and error-free delivery of data. It is known to provide reliable and error-free communication between end systems. It is a connectionless-oriented protocol, over which call request and call accept packets are used to establish connection. It also has an acknowledgement feature and controls the flow of the data through a flow control mechanism.

IP protocol stands for the Internet protocol and is responsible for delivering packets from source host to the destination host by looking at the IP addresses in the packet header. In our Application we have used **IPv4 protocol**.

## 2.2 Socket Programming

Socket programming is a way of connecting two nodes on a network to communicate with each other. One Socket (or a node) listens to a particular port at an IP address, while another socket reaches out to the other to form a connection.

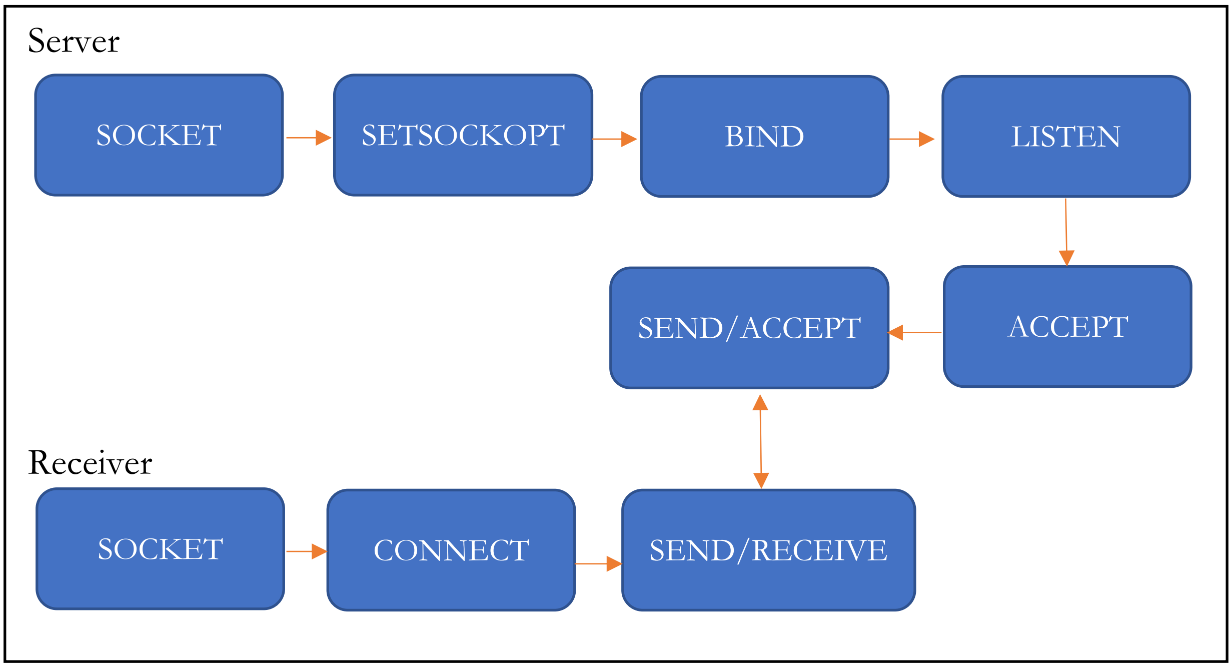
The server forms the listener socket while the client reaches out to the server. We are using TCP sockets in our project because they are telephonic, connection-oriented, and reliable. They also have an acknowledgement feature and control the flow of the data through a flow control mechanism.

Server Characteristics:

- Waits for a request from one of the clients
- Replies to the client with requested data
- Processes the data, and communicates with other clients or servers.

Client Characteristics:

- Initiates requests to servers.
- Waits for replies.
- Receives replies from the server.



## 2.3 Tkinter and Socket Based Chat Application

The Tkinter and Socket Based Chat Application is based on the use of Python sockets and Multithreading to establish a Client-Server Communication. The GUI is implemented using **Tkinter Frameworks** from the Python standard library.

### 2.3.1 Server Architecture for the Chat Application

We are using **TCP Sockets** for setting up our server, and therefore are using importing the **AF_INET** and **SOCK_STREAM** flags.

The tasks for the servers are broken into accepting new connections, handling and receiving messages from clients and broadcasting those messages to other clients over the chat application. The server-side script uses Multi-threading to receive the bytes transmitted over the socket buffer continuously.

### 2.3.2 Client Architecture for the Chat Application

The architecture of the functioning of the client is broken down into receiving bytes from the socket buffer (i.e. server), and sending messages to the server. The client-side script also uses Multi-threading to receive the bytes transmitted over the socket buffer continuously.

## 2.4 Konnect Web-Based Chat Application

We have built a web application using sockets, which have been implemented on both the server and the client level, based on Flask-Sockets (for the Back-end Implementation) and SocketIO - JavaScript (for the Front-End Implementation).

### 2.4.1 SocketIO

SocketIO is a cross-browser JavaScript library that abstracts the client application from the actual transport protocol. It is used to ease the communication between client and server and establish a **persistent, bi-directional communication channel**.

We have incorporated SocketIO in our JavaScript frontend client file - index.html to establish the connection with the server. To put it simply, Socket.IO allows the front-end in our project to interact with the back-end of our project.

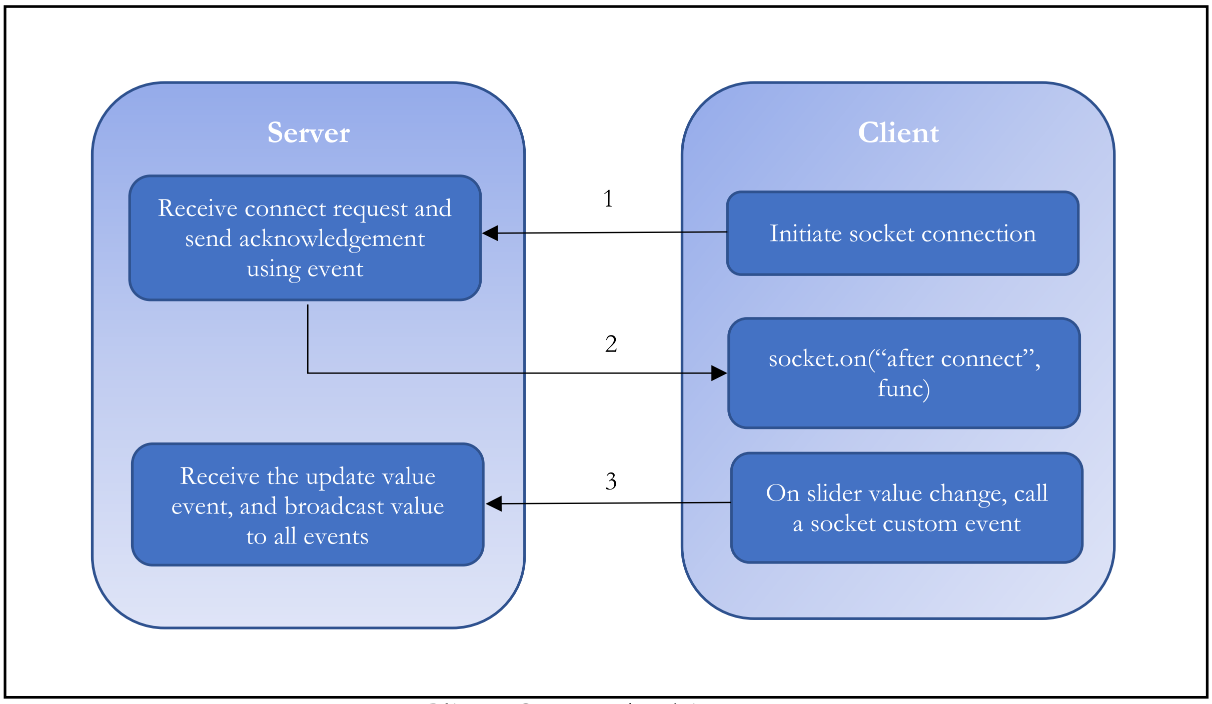
### 2.4.2 Flask-SocketIO

**Flask-SocketIO** gives Flask applications access to low latency bi-directional communications between the clients and the server. The client-side application can use any of the SocketIO official client libraries in Javascript, C++, Java and Swift, or any compatible client to establish a permanent connection to the server.

It is primarily used for initialization of the server, and for sending, receiving and broadcasting messages to and from clients.

### 2.4.3 Server-Side & Client-Side Architecture

The Flask templates are rendered using Jinja2. The frontend uses JavaScript framework.


Server
Client
Receive connect request and send acknowledgement using event
1
Initiate socket connection
2
socket.on("after connect", func)
Receive the update value event, and broadcast value to all events
3
On slider value change, call a socket custom event

Client-Server Architecture

Directory structure :

```
-- Chat Web App
        -- main.py //Declares a SocketIO based Server application
        -- conif.py //Set Flask Configurations
        --. env //Contains flask configurations

        -- application
                -- __init__.py //Builds the core Flask Application
                -- database.py //SQLite3 for table to store and retrieve messages
                -- filters.py //jinja2 template to render the HTML templates
                -- views.py //To define session root channels and handle UI

                -- static
                        -- index.html //Java Script frontend of our Application

                -- templates
                        -- base.html //Defines containers for the application
                        -- history.html //Defines the Message History page view
                        -- index.html //Renders index.html
                        -- login.html //Defines the Login Page view
```

### 2.4.4 Server-Client Communication

We interact using channels over a web socket connection. When the client sends a message on the connect channel, our connect channel will handle the process.

Index.js implements the client socket code on the front-end:

- Creates a socket connection application and connects to the Server
- Creates a new user and saves details in database
- Emit messages over 'event' channel to the server when 'submit' event is executed
- Receives messages over 'message_response' channel

After Creation of the JavaScript frontend, Socket handler is declared on the Flask Backend server side.

main,py ( main application file ) implements the server socket establishment code:

- Creates a socket based flask application instance using .env files
- Looks for connection
- Emits message 'message_response' channel based on the files parsed over the 'event' channel.
- Maintains a database with all the message history details.

### 2.4.5 Application Deployment and Testing

Import flask, socket io and SQLite3 modules for python shell.

1. Edit the environment file with your host IP address
2. Run main.py file over your Operating system Python Shell.
3. Login to the Chat application using the following navigation address in your browser: 0.0.0.0:5000. ("0.0.0.0" is the default Host IP address in the .env file).

# 3 RECEIVE AND SEND ALGORITHMS

Algorithm 1 Receive

```
Input:
Conn, message

Output:
0 :   while TRUE do:
1 :        pack ← Conn.recv(1024)
2 :        pack ← pack.decode("utf8")
3 :        message.list.insert(pack)
4 :   end while
5 :   return
```

**message.list.insert(pack):** Displays the final unpacked message in the message box.

Algorithm 2 Send

```
Input:
Conn, display_message

Output:
0 :   message ← Bytes(display_message, UTF8)
1 :    Conn.send(message)
2 :   return
```

# 4 IMPLEMENTATION ENVIRONMENTS

## 4.1 Tkinter-based Chat Application

| | |
|---|---|
| Programming Languages Used | Python 3.7 |
| Operating System | Windows and Linux |
| Library, Packages or APIs Used | Tkinter<br>Socket<br>Threading |
| Interface Design (GUI) | Created using Tkinter |

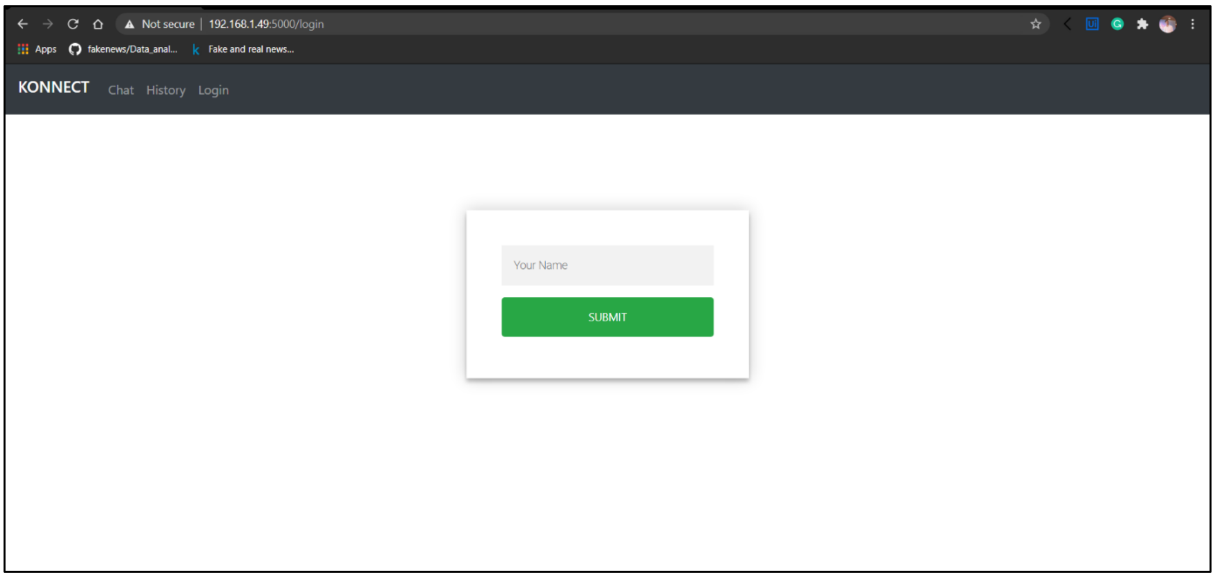
Table 1: Details of the implementation environment of the program, to ensure stability and compatibility across systems and operating systems.

## 4.2 Web-based Chat Application

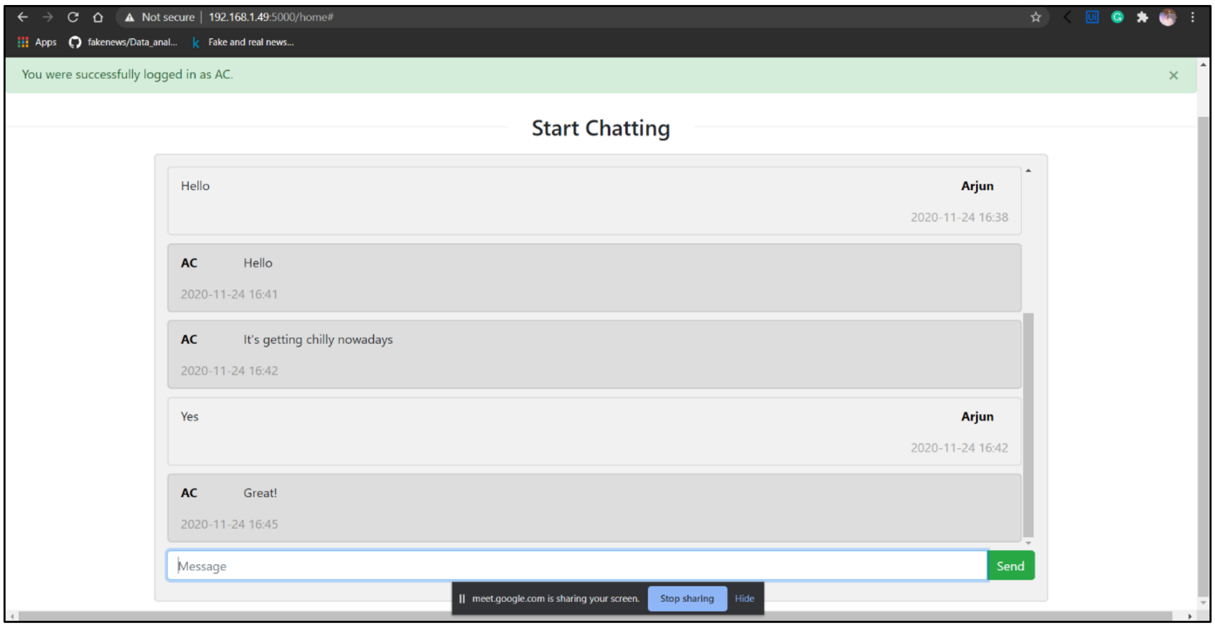
| | |
|---|---|
| Programming Languages Used | Python 3.7, Javascript, HTML |
| Operating System | Windows and Linux |
| Library, Packages or APIs Used | SocketIO<br>Flask-SocketIO |
| Interface Design (GUI) | Created using Tkinter |

Table 2: Details of the implementation environment of the program, to ensure stability and compatibility across systems and operating systems.

# 5 GUI AND RESULTS



**Home Page of the Web App**


Not secure | 192.168.1.49:5000/home#

You were successfully logged in as AC.

Start Chatting

Hello Arjun 2020-11-24 16:38

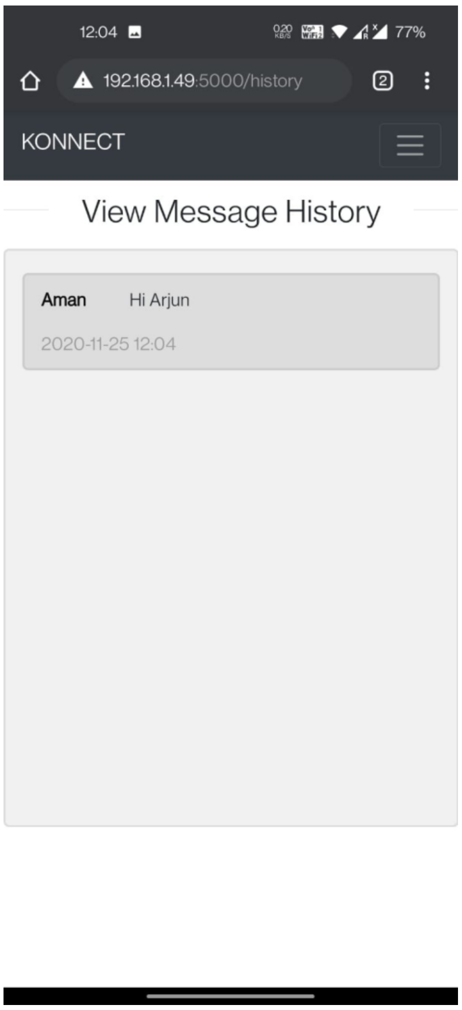
AC Hello 2020-11-24 16:41

AC It's getting chilly nowadays 2020-11-24 16:42

Yes Arjun 2020-11-24 16:42

AC Great! 2020-11-24 16:45

**Chat Room of the Web App**

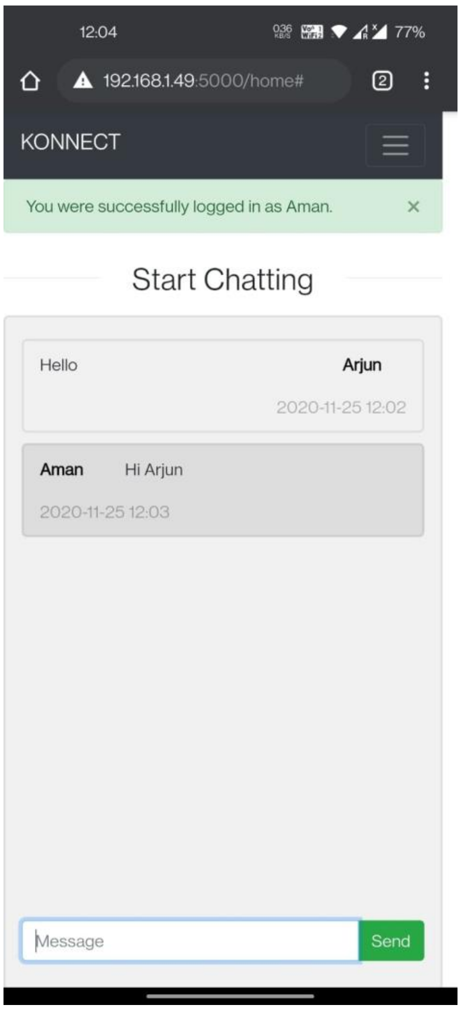

12:04 77%

192.168.1.49:5000/history

KONNECT

View Message History

Aman Hi Arjun 2020-11-25 12:04


12:04 77%

192.168.1.49:5000/home#

KONNECT

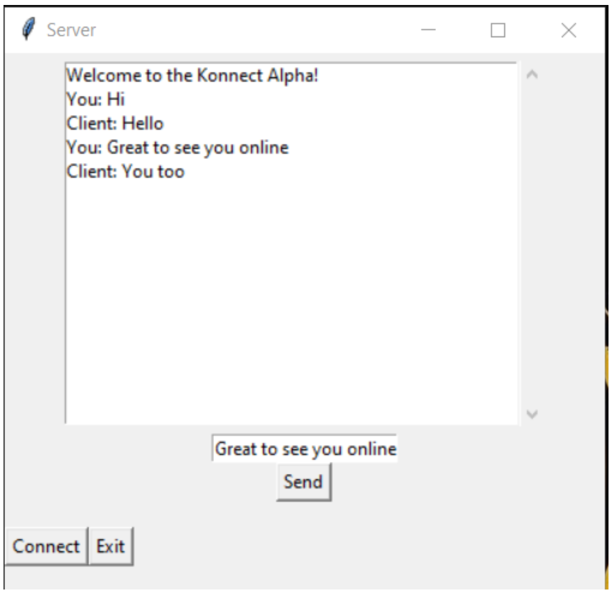
You were successfully logged in as Aman.

Start Chatting

Hello Arjun 2020-11-25 12:02

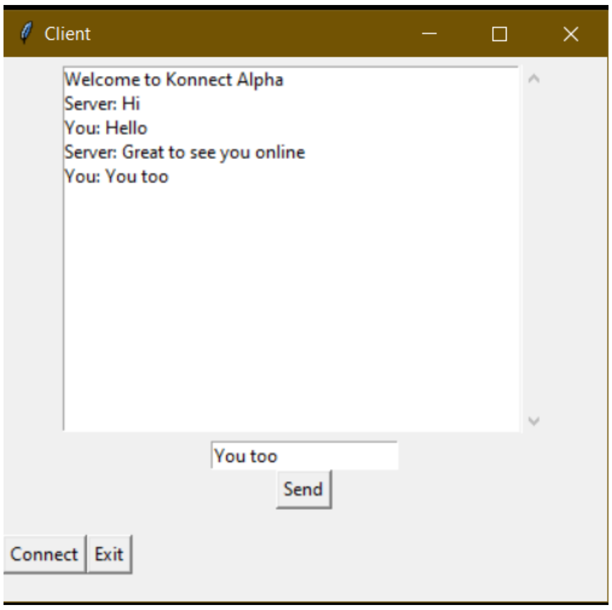
Aman Hi Arjun 2020-11-25 12:03

Web Application Working Simultaneously on Phone and Desktop



```
Welcome to the Konnect Alpha!
You: Hi
Client: Hello
You: Great to see you online
Client: You too
```

**Tkinter based Chat Application Server**



```
Welcome to Konnect Alpha
Server: Hi
You: Hello
Server: Great to see you online
You: You too
```

**Tkinter based Chat Application Server**

Both our Tkinter application and Web application are working as expected, with the web application also communicating between multiple devices on the same network, like between a phone and a computer. Our Tkinter application has a simple, minimalistic UI, while our web application has a clean UI, in line with top chat applications.

## 6 CONCLUSION

We successfully created 2 chatting applications, one using Tkinter, and the other using Flask framework. Both applications are sending messages across users, as well as between client and server properly, and have a simple interface for users to get accustomed to. There is also massive scope to further improve this project. Thus, we successfully met the requirements for the project.

## 7 FUTURE WORK

There is great potential to add features to this project, and some of the features on our roadmap include:

1. Support and implementation for Open Emoji API.
2. Ability to share files of various formats.
3. Login UI and system for enhanced user privacy.
4. Ability to display shared images right in the chat room.
5. A dark theme option for users for comfortable usage in dim environments.

## 8 REFERENCES

1. Flask-SocketIO, https://flask-socketio.readthedocs.io/en/latest/

2. SocketIO, https://socket.io/docs/v3

3. Tkinter Framework, https://docs.python.org/3/library/tkinter.html

4. Socket Framework, https://docs.python.org/3/library/socket.html

# APPENDIX A

## Tkinter Program Code:

### Server

```
import socket
from threading import Thread
from tkinter import messagebox
from os import path
from random import uniform
from tkinter import*

BUFFER_SIZE = 4096


def connect():                              ##connects to client
    SERVER_HOST = "192.168.1.49"
    SERVER_PORT = 33000
    ##SEPARATOR = "<SEPARATOR>"
    s = socket.socket()
    s.bind((SERVER_HOST, SERVER_PORT))
    s.listen(5)
    global client_socket
    client_socket, address = s.accept()
    messagebox.showinfo(f"{address}" + "is connected")
    msg_list.insert(END, "Welcome to the Konnect Alpha!")
    receive_thread = Thread(target=receive)
    receive_thread.start()



def receive(): ##recieve bytes
    while True:
        try:
            msg = client_socket.recv(1024).decode("utf8")##client_socket is client
socket object
            msg_list.insert(END,"Client: "+ msg) #message list is tkinter lise
        except OSError:
            break

def send():
    display_mess = my_msg.get()
    mess = bytes(display_mess, "utf-8")
    client_socket.send(mess)
    msg_list.insert(END, "You: "+display_mess)

    if display_mess == "{quit}":
        client_socket.close()
        root.quit()



def on_closing(event=None):
    my_msg.set("{quit}")
    send()

root=Tk()
root.title("Server")
root.geometry('400x360')
messages_frame = Frame(root,padx=5,pady=5)
my_msg = StringVar() # For the messages to be sent.
my_msg.set("Enter your messages here")

scrollbar = Scrollbar(messages_frame)

msg_list = Listbox(messages_frame, height=15,
width=50,yscrollcommand=scrollbar.set)
```

```
scrollbar.pack(side=RIGHT, fill=Y)
msg_list.pack(side=LEFT, fill=BOTH)
msg_list.pack()
messages_frame.pack()

entry_field = Entry(root, textvariable=my_msg)
entry_field.bind("<Return>", send)
entry_field.pack()

send_button = Button(root, text="Send", command=send) ##uses the send
function
send_button.pack()

mybutton=Button(root,text="Connect",command=connect)
mybutton.pack(side = "left")

button4=Button(root,text="Exit",command=exit) ##the exit command
button4.pack(side = "left")

root.protocol("WM_DELETE_WINDOW", on_closing)
root.mainloop()
```

**Client**

```
import socket

from _compression import BUFFER_SIZE

from tkinter import messagebox

from threading import Thread

from os import path
from time import sleep

from random import uniform

from tkinter import*

from time import sleep


BUFFER_SIZE = 4096


def connect():              ##connects to server

    host = "192.168.1.49"

    port = 33000

    global s

    s = socket.socket()

    s.connect((host, port))

    messagebox.showinfo("connected", "")

    msg_list.insert(END, "Welcome to Konnect Alpha")

    receive_thread = Thread(target=receive)

    receive_thread.start()


def receive(): ##recive bytes

    while True:

        try:

            msg = s.recv(1024).decode("utf8")
            msg_list.insert(END,"Server: "+ msg) ##msg list is tkinter list
        except OSError:
            break


def send():
    display_mess = my_msg.get()
    mess = bytes(display_mess, "utf-8")
    s.send(mess)
    msg_list.insert(END, "You: " + display_mess)


    if display_mess == "{quit}":
        s.close()
        root.quit()


def on_closing(event=None):
    my_msg.set("{quit}")
    send()


root=Tk()
root.title("Client")
root.geometry('400x360')
messages_frame = Frame(root,padx=5,pady=5)
```

```
my_msg = StringVar() # For the messages to be sent.

my_msg.set("Enter your messages here.")


scrollbar = Scrollbar(messages_frame)


msg_list = Listbox(messages_frame, height=15,
width=50,yscrollcommand=scrollbar.set)


scrollbar.pack(side=RIGHT, fill=Y)

msg_list.pack(side=LEFT, fill=BOTH)

msg_list.pack()

messages_frame.pack()


entry_field = Entry(root, textvariable=my_msg)

entry_field.bind("<Return>", send)

entry_field.pack()


send_button = Button(root, text="Send", command=send) ##uses the send
function

send_button.pack()


mybutton=Button(root,text="Connect",command=connect)

mybutton.pack(side='left')
```

```
button4=Button(root,text="Exit",command=exit) ##the exit coomand

button4.pack(side='left')



root.protocol("WM_DELETE_WINDOW", on_closing) ##send {quit}

root.mainloop()
```

## APPENDIX B

**Due to the large size of the source code, and multiple files, we have provided a link for the source code for reference.**

**Konnect Web Application Source Code:**
https://drive.google.com/drive/folders/1SulPEvyDSMphQYfGL_gY-lMW3Vfd3kxx?usp=sharing