TP 4.2

Alumno: Lezana Mauricio Sebastian
LU:57241

Actividad 1: Para el desarrollo general de esta actividad voy a desarrollar dos archivos. Ambos en python donde por un lado habra un servidor y por el otro una o mas instancias de un cliente

1.Diseñar y desarrollar una aplicación de chat en red utilizando sockets para permitir la comunicación en tiempo real entre varios usuarios.

2. Crear una aplicación de chat en la que al menos dos usuarios puedan comunicarse a través de una red local o a través de Internet utilizando sockets.

5. Manejar adecuadamente los errores de conexión y desconexión de clientes, y proporcionar una mejora en la experiencia de chat.

1, 2 y 5.

Vamos a diseñar y desarrolar dicha aplicacion de chat con dos scripts de python

Servidor

En lo personal, siento que es autoexplicativo el codigo pero voy a proceder a explicarlo para demostrar que tengo conocimiento en lo que hace el mismo

La funcion broadcast como su nombre idica es la funcion que transmite a todos los clientes. La funcion recibe el mensaje a enviar y quien lo envio (si no se le especifica su valor por defecto es null), ya que despues debe comprobar quien envio dicho mensaje para no mandarselo. Personalmente manejo a las excepciones desconectando al cliente que mando los mensajes y terminando el for

```python
import socket
import threading

host = '127.0.0.1'
port = 49999

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((host, port))

server.listen()

print("Servidor de Redes WOW!")
print(f"[Host : Puerto de ejecución] : {host}:{port}")

clients = []
usernames = []
# Función para enviar un mensaje a todos los clientes
def broadcast(message, _client=None):
    for client in clients:
        if client != _client:
            try:
                message_with_line=(message.decode('utf-8') + '\n').encode('utf-8')
                client.send(message_with_line)
            except:
                # Manejo de errores si no se puede enviar el mensaje
                client.close()
                remove_client(client)
                break;
```

La funcion remove_client remueve un cliente de la conexion.
La funcion handle_message esta permanentemente recibiendo y enviando los mensajes que le llegan al servidor.
La funcion receive_connections permanentemente espera nuevas conexiones y las pone en un hilo con el manejador de mensajes

Para terminar al final del archivo se llama a la ultima funcion mencionada para ejecutarla.

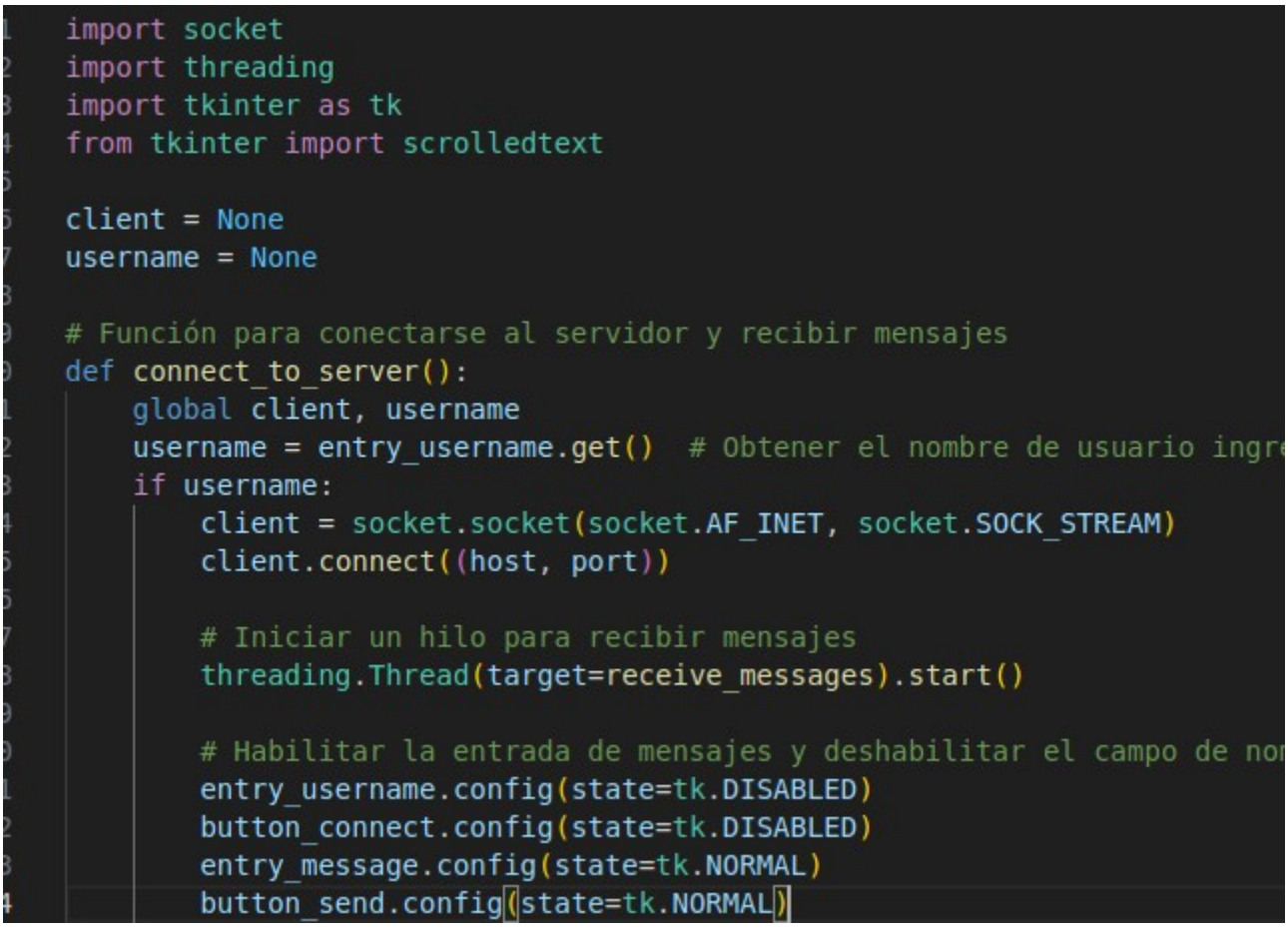
```python

# Función para manejar la desconexión del cliente
def remove_client(client):
    if client in clients:
        index = clients.index(client)
        username = usernames[index]
        broadcast(f"El usuario: {username} se ha desconectado.\n".encode
        clients.remove(client)
        usernames.remove(username)
        client.close()



# Función para manejar los mensajes de cada cliente
def handle_messages(client):
    while True:
        try:
            message = client.recv(1024)
            broadcast(message, client)
        except:
            # Manejo de desconexión del cliente
            remove_client(client)
            break
def receive_connections():
    while True:
        client, address = server.accept()

        client.send("@username".encode("utf-8"))
        username = client.recv(1024).decode('utf-8')

        clients.append(client)  # Corregido: Se estaba añadiendo la lista
        usernames.append(username)

        print(f"{username} se ha unido a través de {str(address)}")

        message = f"El usuario: {username} se ha unido al chat".encode("
        broadcast(message, client)
        client.send("Conexión exitosa al servidor".encode("utf-8"))

        thread = threading.Thread(target=handle_messages, args=(client,)
        thread.start()

receive_connections()
```

Clientes:

La funcion connect_to_server conecta con al servidor

La funcion send_message envia mensajes y maneja los errores de los mismos de haber

La funcion receive_messages permanentemente escucha mensajes en la espera de estos y los muestra de haberlos



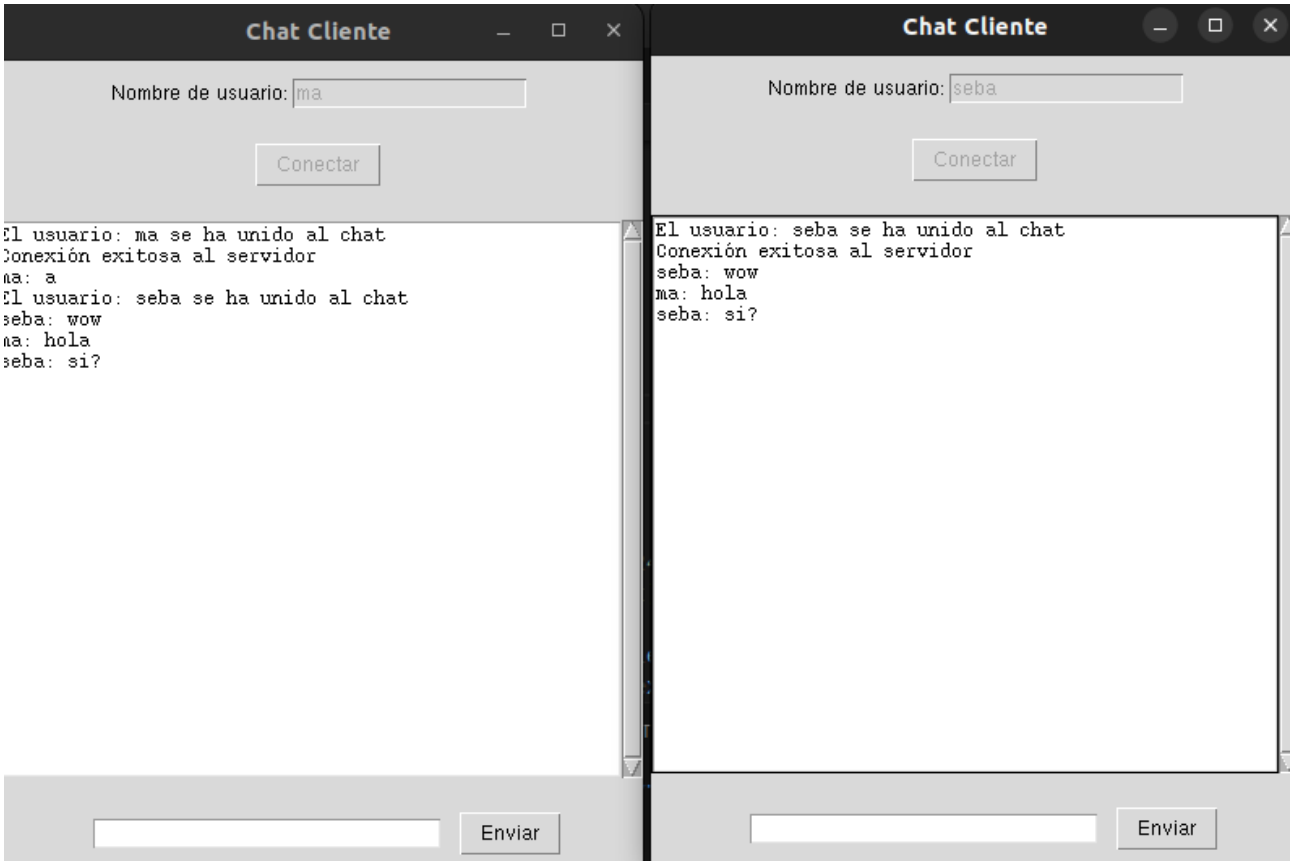
```
import socket
import threading
import tkinter as tk
from tkinter import scrolledtext

client = None
username = None

# Función para conectarse al servidor y recibir mensajes
def connect_to_server():
    global client, username
    username = entry_username.get()  # Obtener el nombre de usuario ingre
    if username:
        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client.connect((host, port))

        # Iniciar un hilo para recibir mensajes
        threading.Thread(target=receive_messages).start()

        # Habilitar la entrada de mensajes y deshabilitar el campo de nor
        entry_username.config(state=tk.DISABLED)
        button_connect.config(state=tk.DISABLED)
        entry_message.config(state=tk.NORMAL)
        button_send.config(state=tk.NORMAL)
```

```python
# Función para recibir mensajes del servidor y mostrarlos en la interfaz
def receive_messages():
    while True:
        try:
            message = client.recv(1024).decode('utf-8')
            if message == "@username":
                # El servidor solicita el nombre de usuario, lo enviamos
                client.send(username.encode('utf-8'))
            else:
                # Mostrar cualquier otro mensaje recibido
                chat_display.config(state=tk.NORMAL)
                chat_display.insert(tk.END, message + '\n')  # Mostrar el mensaje recibido
                chat_display.yview(tk.END)  # Desplazar hacia el último mensaje
                chat_display.config(state=tk.DISABLED)
        except:
            print("Error al recibir mensajes.")
            break

# Función para enviar el mensaje escrito por el usuario
def send_message():
    global username
    message = entry_message.get()
    if message:
        full_message = f"{username}: {message}"  # Formatear el mensaje con el nombre de usuari
        client.send(full_message.encode('utf-8'))  # Enviar el mensaje al servidor
        entry_message.delete(0, tk.END)  # Limpiar el campo de entrada de mensajes
```

```python
# Configuración de la interfaz gráfica
window = tk.Tk()
window.title("Chat Cliente")
window.geometry("400x500")

# Dirección y puerto del servidor
host = '127.0.0.1'
port = 49999

# Nombre de usuario
frame_username = tk.Frame(window)
frame_username.pack(pady=10)

label_username = tk.Label(frame_username, text="Nombre de usuario:")
label_username.pack(side=tk.LEFT)
entry_username = tk.Entry(frame_username)
entry_username.pack(side=tk.LEFT)

# Botón para conectar al servidor
button_connect = tk.Button(window, text="Conectar", command=connect_to_server)
button_connect.pack(pady=10)

# Área de chat donde se muestran los mensajes
chat_display = scrolledtext.ScrolledText(window, wrap=tk.WORD, state=tk.DISABLED)
chat_display.pack(pady=10, fill=tk.BOTH, expand=True)

# Entrada para escribir mensajes
frame_message = tk.Frame(window)
frame_message.pack(pady=10)
```

```python
# Área de chat donde se muestran los mensajes
chat_display = scrolledtext.ScrolledText(window, wrap=tk.WORD, state=tk.DISABLED)
chat_display.pack(pady=10, fill=tk.BOTH, expand=True)

# Entrada para escribir mensajes
frame_message = tk.Frame(window)
frame_message.pack(pady=10)

entry_message = tk.Entry(frame_message, width=30, state=tk.DISABLED)  # Se desactiva hasta la conexión
entry_message.pack(side=tk.LEFT, padx=10)

# Botón para enviar mensajes
button_send = tk.Button(frame_message, text="Enviar", command=send_message, state=tk.DISABLED)
button_send.pack(side=tk.LEFT)

# Iniciar la interfaz gráfica
window.mainloop()
```

3 y 4. Gracias a la libreria tkinter podemos hacer una interfaz grafica para los clientes en python rapidamente con el siguiente resultado



6. Implementa comandos especiales en el cliente, como “/listar” para ver la lista de usuarios conectados al chat y “/quitar” para desconectarse del servidor

Con estos cambios podemos incluir ambas funciones

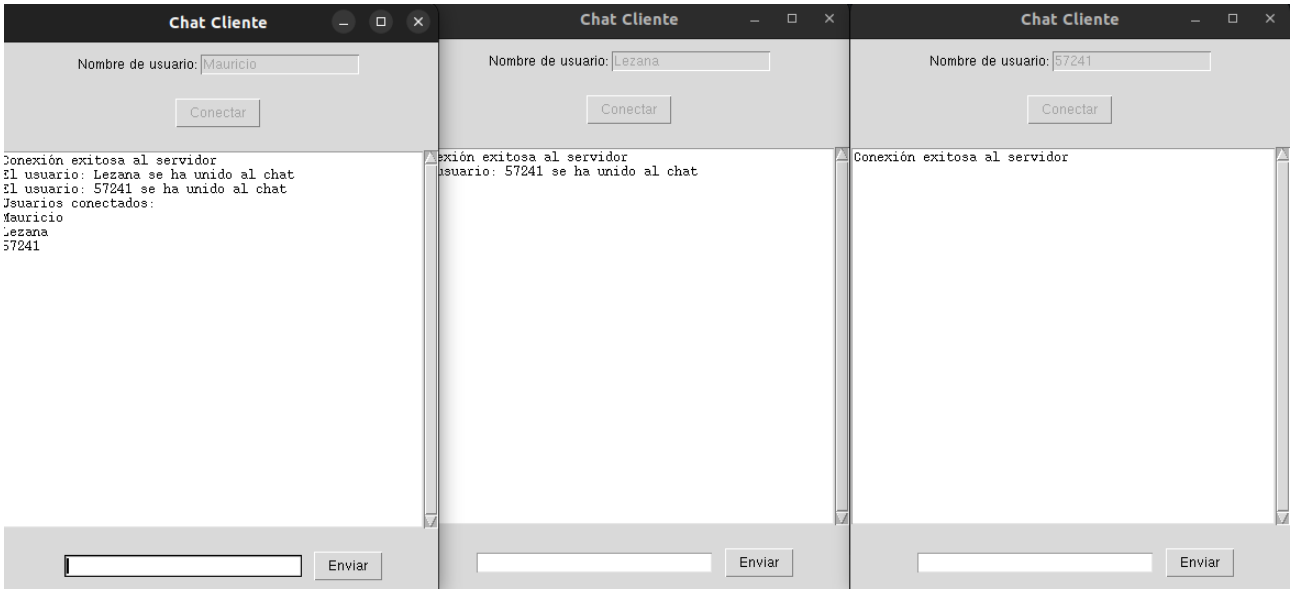
cambios en el servidor:

```python
# Función para manejar los mensajes de cada cliente
def handle_messages(client):
    while True:
        try:
            message = client.recv(1024).decode('utf-8')

            # Manejar comandos especiales
            if message.startswith('/listar'):
                send_user_list(client)
            elif message.startswith('/quitar'):
                remove_client(client)
                break
            else:
                broadcast(message.encode('utf-8'), client)
        except:
            remove_client(client)
            break
```

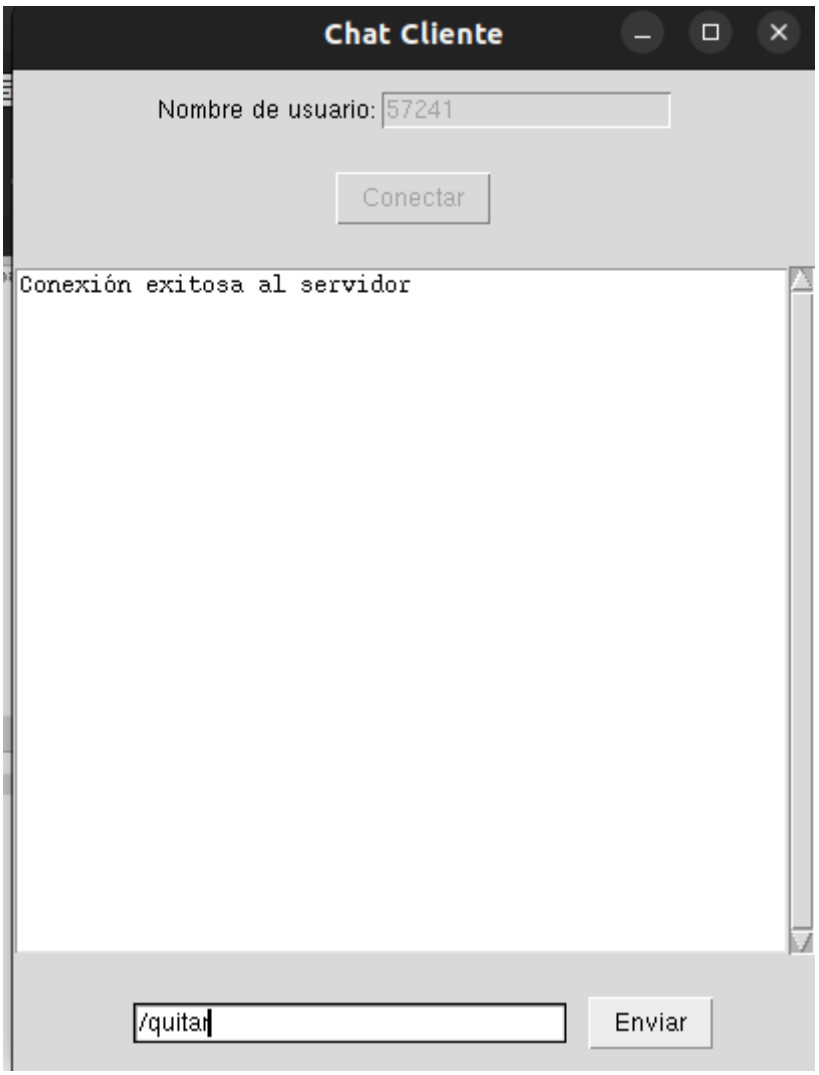
cambios en el cliente

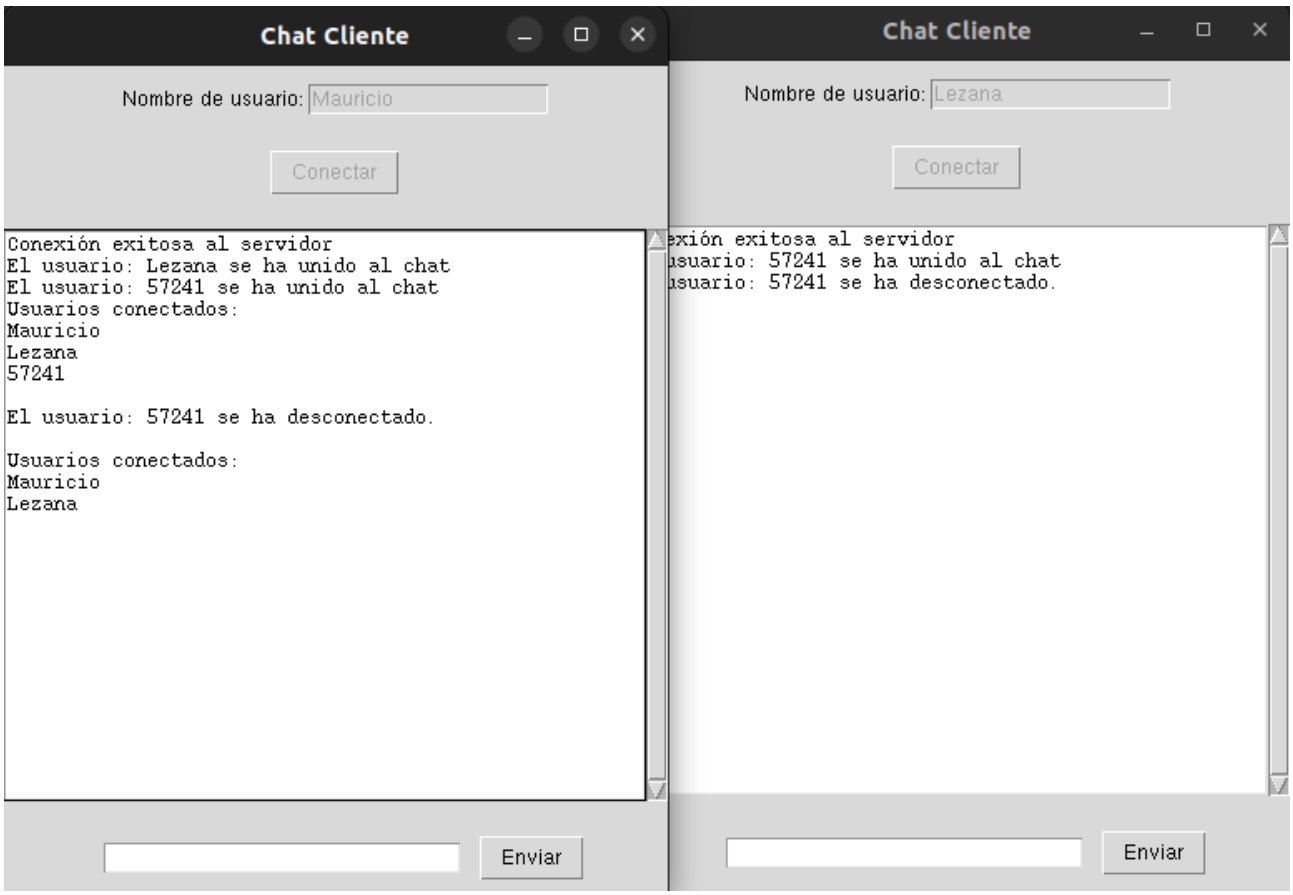
```python
def send_message():
    global username
    message = entry_message.get()
    if message:
        if message =="/listar":
            client.send(message.encode('utf-8'))
        elif message == "/quitar":
            client.send(message.encode('utf-8'))
            window.quit()  # Cierra la ventana cuando se desconecta
        else:
            full_message = f"{username}: {message}"
            chat_display.config(state=tk.NORMAL)
            chat_display.insert(tk.END, full_message + "\n")
            chat_display.config(state=tk.DISABLED)
            chat_display.yview(tk.END)
            try:
                client.send(full_message.encode('utf-8'))
            except:
                messagebox.showerror("Error de conexión", "No se pudo enviar el mensaje. Conexión perdida.")
        entry_message.delete(0, tk.END)
```

```python
def send_message():
    global username
    message = entry_message.get()
    if message:
        if message =="/listar":
            client.send(message.encode('utf-8'))
        elif message == "/quitar":
            client.send(message.encode('utf-8'))
            window.quit()  # Cierra la ventana cuando se desconecta
        else:
            full_message = f"{username}: {message}"
            chat_display.config(state=tk.NORMAL)
            chat_display.insert(tk.END, full_message + "\n")
            chat_display.config(state=tk.DISABLED)
            chat_display.yview(tk.END)
            try:
                client.send(full_message.encode('utf-8'))
            except:
                messagebox.showerror("Error de conexión", "No se pudo enviar el mensaje. Conexión perdida.")
        entry_message.delete(0, tk.END)
```



si el usuario 57241 envia el mensaje /quitar


Chat Cliente
Nombre de usuario:
57241
Conectar
Conexión exitosa al servidor
/quitar
Enviar

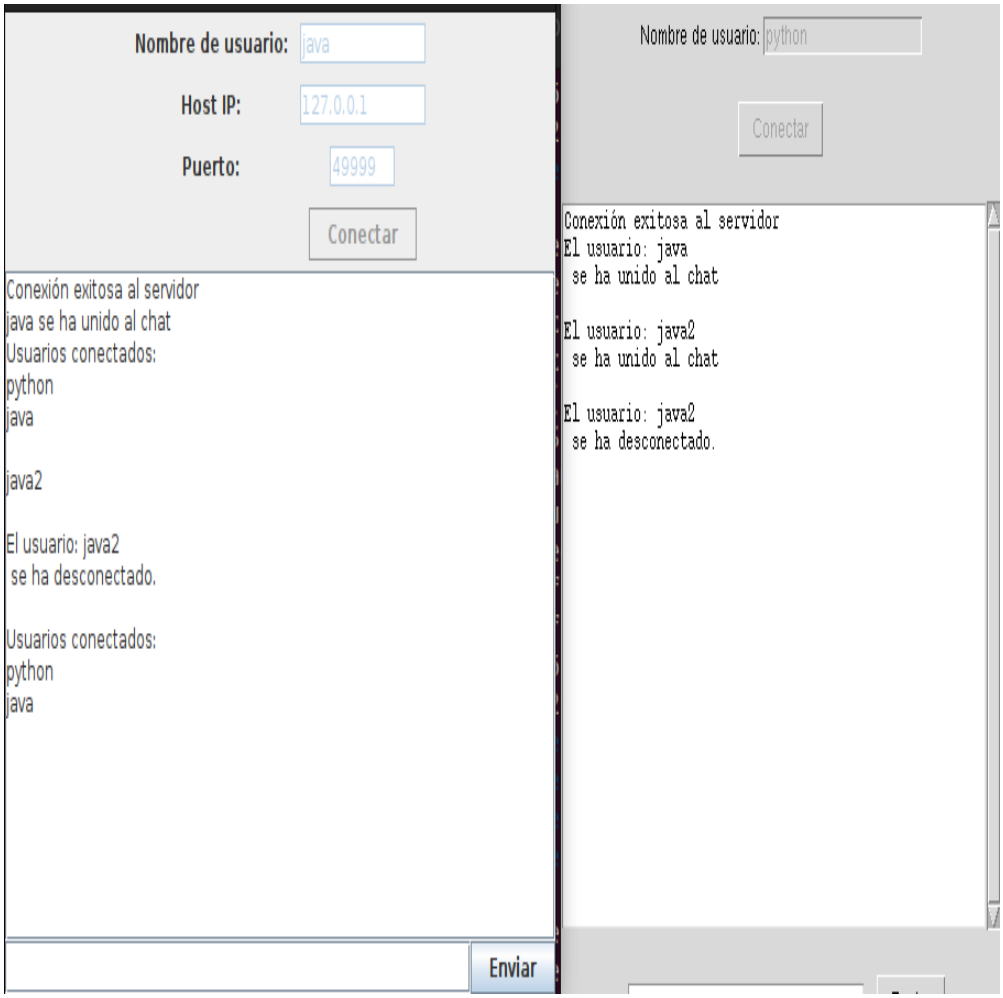


Ejercicio 2: Utilizando como base el desarrollo del ejercicio N°1: diseñar e implementar un programa, basado en la comunicación por sockets, sobre alguno de los siguientes escenarios:

Mi eleccion fue:

1. Nuevo Cliente de Chat Se requiere la programación de un nuevo cliente de chat, en un lenguaje diferente al seleccionado en el ejercicio 1, para este punto el mismo debe poder conectar al servidor desarrollado en el punto anterior. El nuevo cliente de chat debe solicitar al usuario el ingreso de parámetros: Host IP y puerto del servidor, luego mostrar con un mensaje si la conexión ha sido exitosa. El comportamiento y funcionalidades deben ser las mismas que el cliente desarrollado en el ejercicio 1.

Para el otro cliente use JAVA en conjunto con las librerias awt y swing. La comunicacion entre ambos servicios funciona perectamente como podemos ver en la captura de abajo.



El codigo

```java
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ChatClient {
    private String username;
    private String hostIP;
    private int port;
    private Socket socket;
    private BufferedReader reader;
    private PrintWriter writer;

    private JFrame frame;
    private JTextArea chatArea;
    private JTextField messageField;
    private JTextField usernameField;
    private JTextField hostField;
    private JTextField portField;
    private JButton connectButton;

    Run | Debug
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new ChatClient().createAndShowGUI());

    }
```

```java
private void connectToServer() {
    username = usernameField.getText().trim();
    hostIP = hostField.getText().trim();
    try {
        port = Integer.parseInt(portField.getText().trim());
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(frame, "Por favor, ingresa un puerto vá
        return;
    }

    if (username.isEmpty() || hostIP.isEmpty()) {
        JOptionPane.showMessageDialog(frame, "Por favor, ingresa todos los da
        return;
    }

    try {
        socket = new Socket(hostIP, port);
        writer = new PrintWriter(socket.getOutputStream(), true);
        reader = new BufferedReader(new InputStreamReader(socket.getInputStre



        writer.println(username); // Fixed: Send username with a newline

        // Start a thread to listen for messages from the server
        new Thread(this::receiveMessages).start();

        chatArea.append("Conexión exitosa al servidor\n");
```

```
private void connectToServer() {

        // Disable connection fields
        connectButton.setEnabled(false);
        usernameField.setEnabled(false);
        hostField.setEnabled(false);
        portField.setEnabled(false);
        messageField.setEnabled(true); // Enable message field after connecti

    } catch (IOException e) {
        JOptionPane.showMessageDialog(frame, "No se pudo conectar al servidor
    }
}

private void sendMessage() {
    String message = messageField.getText().trim();
    if (message.isEmpty()) return;

    if (message.equals("/quitar")) {
        writer.println(message);
        closeConnection();
        frame.dispose();
    } else if (message.equals("/listar")) {
        writer.println(message); // Send the /listar command to the server
    } else {
        String fullMessage = username + ": " + message;
        chatArea.append(fullMessage + "\n"); // Show the user's own message i
        writer.println(fullMessage); // Send the message to the server
    }
```

```java
private void receiveMessages() {
    String message;
    try {
        while (true) {
            message=reader.readLine();
            String receivedMessage = message; // Declare a Ffinal or effectively final variable


            if (receivedMessage.startsWith("@username")) {

                SwingUtilities.invokeLater(() -> chatArea.append(username));
            } else {
                SwingUtilities.invokeLater(() -> chatArea.append("\n"+ receivedMessage + "\n"));
            }

        }
    } catch (IOException e) {
        SwingUtilities.invokeLater(() -> chatArea.append("Conexión perdida con el servidor.\n"));
        e.printStackTrace();
    }
}
    private void closeConnection() {
        try {
            if (socket != null) socket.close();
            if (reader != null) reader.close();
            if (writer != null) writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

Adicionalmente tuve que modificar la funcion de broatcast del servidor

```python
def broadcast(message, _client=None):
    for client in clients:
        if client != _client:
            try:
                message_with_line=(message.decode('utf-8') + '\n').encode('utf-8')
                client.send(message_with_line)
            except:
                # Manejo de errores si no se puede enviar el mensaje
                client.close()
                remove_client(client)
```