

# PROTOCOLO DE COMUNICACIÓN PARA CHAT EN TIEMPO REAL

## 1. Introducción

Este documento describe el protocolo de comunicación y la arquitectura de un sistema de chat en tiempo real basado en Python. El sistema consta de tres componentes principales:

- Servidor: Maneja la comunicación entre los clientes.
- Cliente: Aplicación en Tkinter que permite a los usuarios enviar y recibir mensajes.
- Bot de Telegram: Conectado al servidor TCP, retransmite mensajes entre Telegram y el chat.

El sistema permite la comunicación en tiempo real entre varios clientes y la integración con Telegram.

---

## 2. Arquitectura del Sistema

### 2.1 Servidor

El servidor actúa como un intermediario entre los clientes, recibiendo mensajes y reenviándolos a todos los clientes conectados.

- Utiliza sockets TCP para la comunicación.
- Maneja múltiples conexiones con threading.
- No inicia la comunicación, solo responde a las peticiones de los clientes.
- Puerto de escucha: 2000

### 2.2 Cliente (Aplicación en Tkinter)

Cada cliente se conecta al servidor y permite al usuario:

- Enviar y recibir mensajes en una interfaz gráfica.
- Enviar mensajes con "Enter" y escribir en varias líneas con "Shift + Enter".
- Manejar reconexiones automáticas si se pierde la conexión.
- Interfaz adaptable y configurable en Tkinter.

### 2.3 Bot de Telegram

El bot de Telegram se conecta al servidor TCP como un cliente más, permitiendo que:

- Los mensajes del chat TCP aparezcan en un grupo de Telegram.
  - Los mensajes enviados en Telegram se reenvíen al chat TCP.
  - Se manejen respuestas directas a mensajes en Telegram.
  - Usa asincio para manejar eventos en tiempo real sin bloquear la ejecución.
-

### 3. Protocolo de Comunicación

El protocolo de comunicación sigue el modelo de cliente-servidor, donde los clientes deben conocer la IP y el puerto (2000) del servidor para conectarse.

#### 3.1 Formato de Mensajes

Los mensajes siguen una estructura estandarizada con el formato:

nombre\_usuario: mensaje

Donde:

- nombre\_usuario indica quién envía el mensaje.
- mensaje es el contenido enviado.

#### 3.2 Tipos de Mensajes

- nombre\_usuario: mensaje → Envía un mensaje al chat.
- nombre\_usuario se ha unido al chat. → Notificación de conexión.
- nombre\_usuario ha abandonado el chat. → Notificación de desconexión.

El servidor valida las peticiones y responde según corresponda.

---

### 4. Explicación del Código

#### 4.1 Servidor (server.py)

- Escucha en 0.0.0.0:2000 y acepta conexiones de clientes.
- Usa threading para manejar múltiples clientes simultáneamente.
- Guarda los clientes en una lista y retransmite mensajes a todos.
- Maneja reconexiones y caídas de clientes.

#### 4.2 Cliente (cliente.py)

- Se conecta al servidor y permite enviar/recibir mensajes.
- Interfaz en Tkinter con scrolledtext para el chat y Text para la entrada de mensajes.
- Reconexión automática si se pierde la conexión.
- Permite escribir en varias líneas y enviar con "Enter".
- Maneja cierre de aplicación y envía notificación de desconexión.

#### 4.3 Bot de Telegram (bot.py)

- Se conecta al servidor como un cliente más.
- Recibe mensajes del chat TCP y los reenvía a Telegram.
- Recibe mensajes de Telegram y los envía al chat TCP.

- Usa asyncio para manejar eventos sin bloquear la ejecución.
- Permite responder mensajes en Telegram con referencia al chat.

---

## 5. Limitaciones y Propuestas de Mejora

### 5.1 Limitaciones

- Actualmente no hay autenticación de usuarios.
- No hay registro de mensajes en una base de datos.
- El sistema solo permite un chat global, no chats privados.
- La conexión del bot a Telegram puede sufrir retrasos por asyncio.

### 5.2 Posibles Mejoras

- Implementar usuarios con autenticación y contraseñas.
- Permitir múltiples salas de chat en el servidor.
- Guardar mensajes en una base de datos SQLite o MySQL.
- Agregar encriptación en los mensajes para mayor seguridad.

---

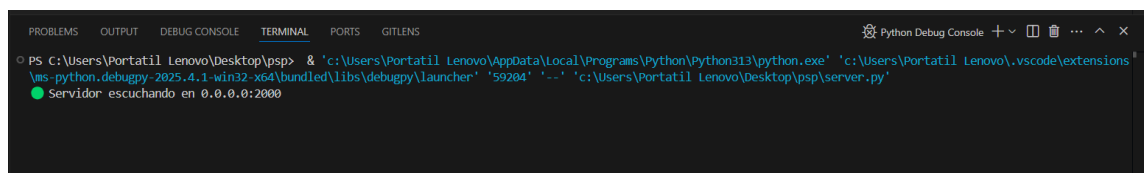
## 6. Enlace para Pruebas

Para unirse al grupo de Telegram y probar el funcionamiento del programa, usa el siguiente enlace: [Unirse al grupo de Telegram](#)

## 7. Capturas de Pantalla

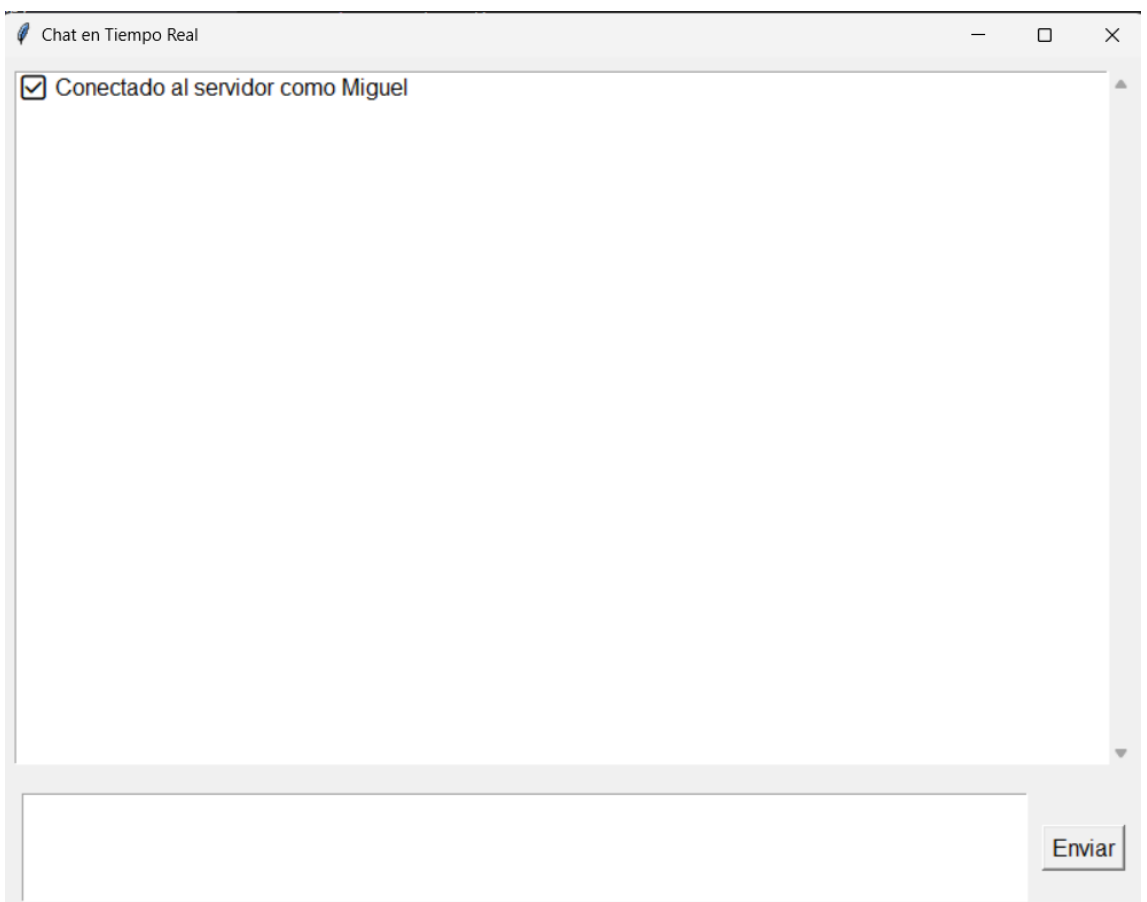
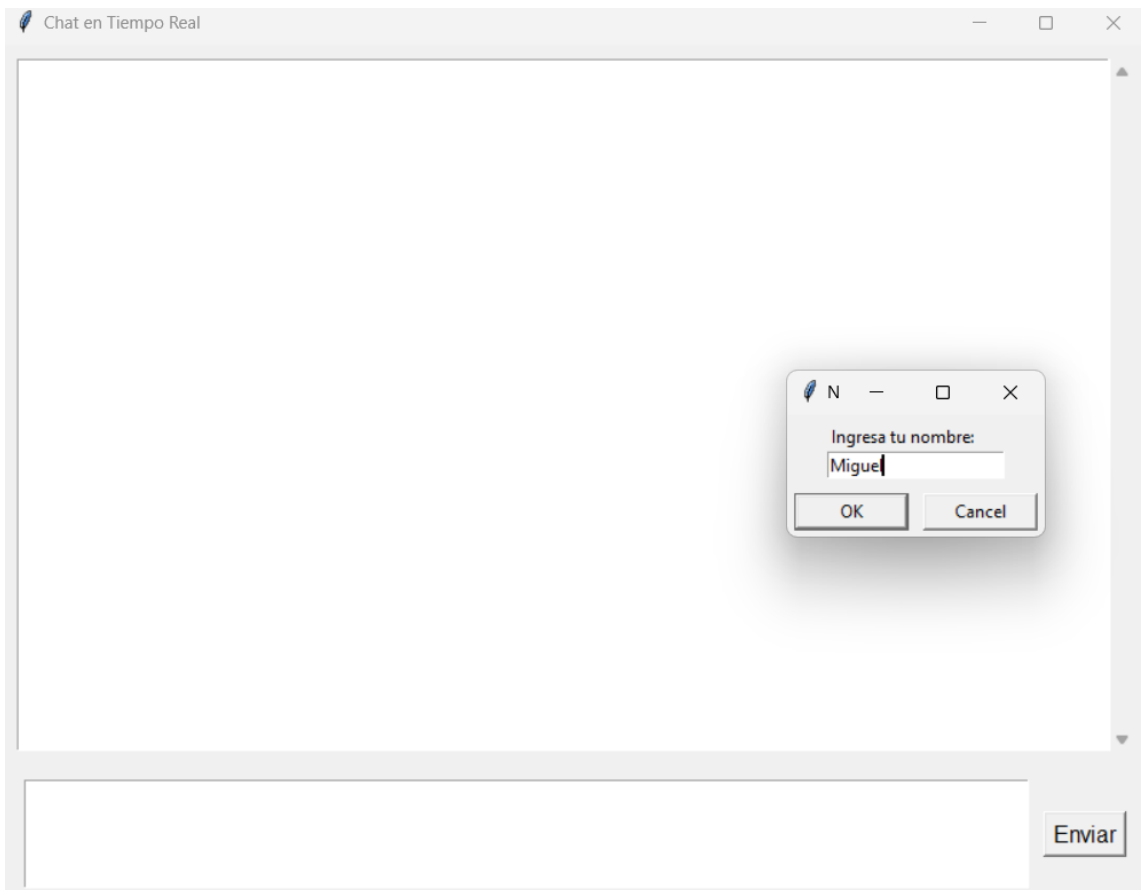
A continuación, se pueden insertar capturas de pantalla para ilustrar el funcionamiento del sistema:

- Inicio del servidor: Indicación de que el servidor está en ejecución y listo para recibir conexiones.

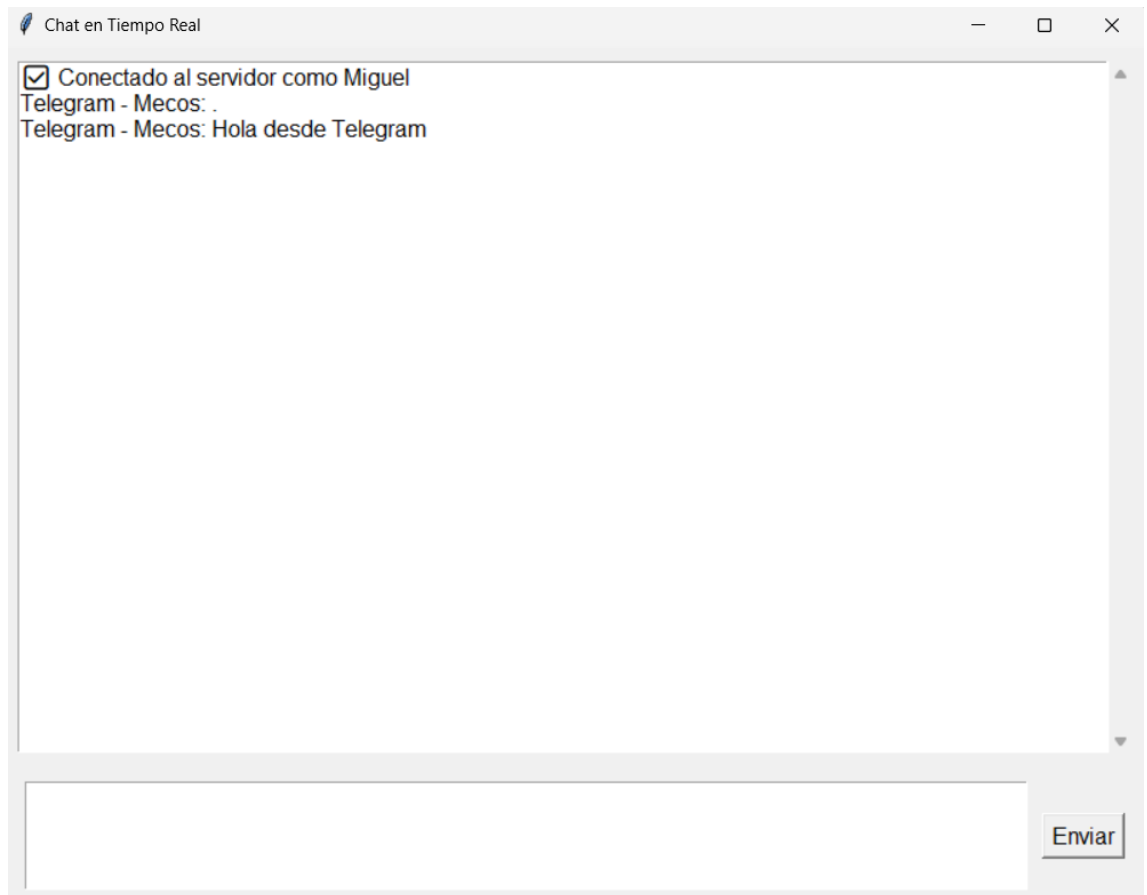


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GIT LENS
Python Debug Console + - [ ] ... ^ x
PS C:\Users\Portatil Lenovo\Desktop\psp> & 'c:\Users\Portatil Lenovo\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\Portatil Lenovo\.vscode\extensions\ms-python.debugpy-2025.4.1-win32-x64\bundle\libs\debugpy\launcher' '59204' '-' 'c:\Users\Portatil Lenovo\Desktop\psp\server.py'
● Servidor escuchando en 0.0.0.0:2000
```

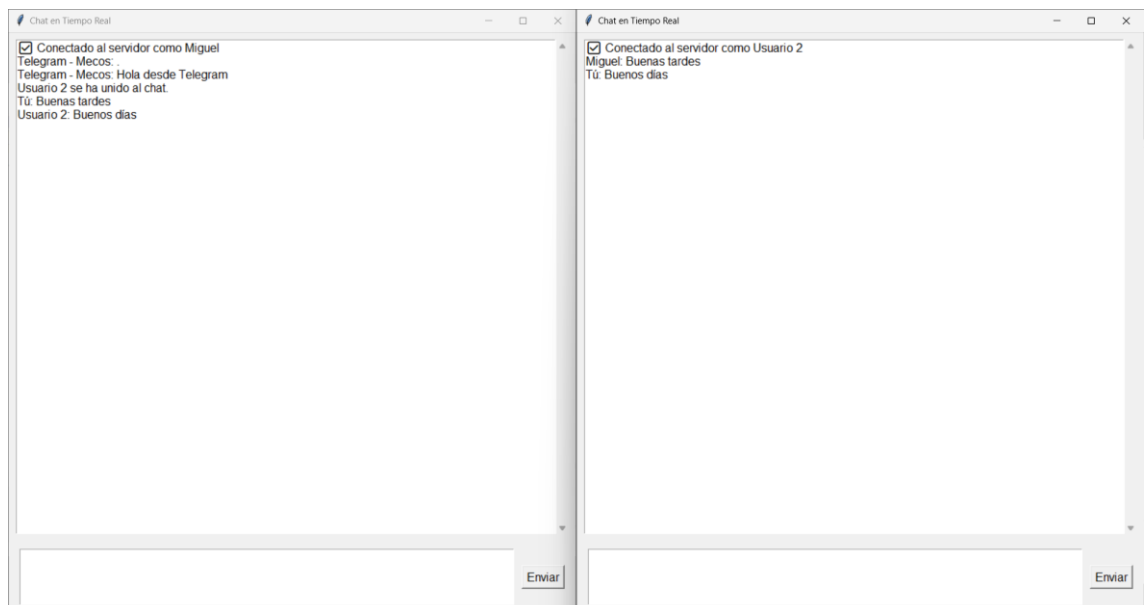
- Conexión de un cliente: Captura de la interfaz del cliente con el chat activo.



- Interacción con Telegram: Mensajes enviados desde Telegram y su reflejo en el chat del cliente.



- Envío y recepción de mensajes: Ejemplo de cómo los usuarios se comunican en tiempo real.



## 6. Conclusión

Este sistema de chat en tiempo real permite la comunicación entre varios clientes y la integración con Telegram. Su arquitectura basada en sockets TCP lo hace rápido y eficiente.

Con algunas mejoras, podría ser una solución completa para sistemas de chat con seguridad y escalabilidad mejorada.