Progetto laboratorio Programmazione di Reti
Traccia 1: Sistema di Chat Client-Server

Gambacorta Giuseppe

14 maggio 2024

1 Introduzione

Lo scopo di questo progetto è quello di sviluppare un sistema di chat client-server che consenta a più utenti di connettersi simultaneamente a una chatroom condivisa, inviare e ricevere messaggi in tempo reale. Il server deve essere in grado di gestire le connessioni provenienti da più client e di instradare i messaggi tra di essi, garantendo una comunicazione fluida e affidabile. Il client, d'altro canto, deve fornire un'interfaccia intuitiva che consenta agli utenti di connettersi al server, inviare messaggi e visualizzare i messaggi ricevuti dagli altri partecipanti alla chatroom.

1.1 Requisiti funzionali

- il server deve essere in grado di accettare connessioni multiple e gestire la trasmissione dei messaggi in una chatroom condivisa
- Implementazione di una GUI funzionante per i client che consente agli utenti di connettersi al server,inviare messaggi e visualizzare la chat globale
- Il sistema, sia lato server che client, deve gestire propriamente errori ed eccezioni

2 Dettagli sistem

2.1 Server

Dopo aver scelto una specifica porta o utilizzato quella di default, una volta avviato, il server entra in modalità di ascolto, accettando connessioni in entrata dai client. Ogni volta che un nuovo client si connette, il server dedica un thread e una socket per gestire la comunicazione con quel client. Questo approccio consente al server di gestire simultaneamente le richieste provenienti da più client senza bloccare il flusso di esecuzione principale.
Ogni client, a sua volta, rimane costantemente in ascolto per eventuali messaggi in arrivo dalla chatroom. I messaggi ricevuti vengono immagazzinati in un buffer locale all'interno del client, in attesa di essere elaborati dal server.
Per garantire una comunicazione efficace tra i client, il server implementa un meccanismo di polling periodico. A intervalli regolari, il server interroga tutti i client connessi, raccogliendo i messaggi presenti nei buffer locali e ordinandoli in base alla sequenza temporale. Una volta raccolti e ordinati i messaggi, il server li trasmette a tutti i client connessi, garantendo così che tutti i partecipanti alla chatroom ricevano i messaggi nell'ordine corretto, ponendo rimedio a eventuali ritardi di ricezione da parte del server.

2.2 Client

Ogni client è dotato di un'interfaccia grafica utente (GUI) dedicata. Prima di connettersi al server, gli utenti devono inserire un nickname e specificare la porta su cui il server è in ascolto. Queste informazioni sono essenziali per stabilire una connessione con il server e identificare chi è il mittente dei singoli messaggi. La GUI del client fornisce un'interfaccia intuitiva per l'invio dei messaggi. Gli utenti possono digitare il testo del messaggio nella finestra di input e inviarlo al server facendo clic su un pulsante. Mentre i client inviano messaggi al server, rimangono anche in ascolto per eventuali messaggi in arrivo dalla chatroom. I messaggi ricevuti vengono visualizzati immediatamente nella finestra della chat, consentendo agli utenti di leggere e rispondere tempestivamente alle comunicazioni degli altri partecipanti alla chatroom. All'interno della GUI del client è presente un campo di testo dedicato che indica lo stato della connessione ed eventuali errori. Questo campo di testo fornisce agli utenti un feedback immediato sullo stato della connessione al server, informandoli se la connessione è attiva e stabilita correttamente o se si sono verificati problemi durante il processo di connessione.

La logica di invio e ricezione dei dati è separata dalla GUI e viene gestita da un oggetto distinto, situato all'interno del file "client.py". Questo approccio permette di mantenere la separazione delle responsabilità all'interno dell'applicazione. Utilizzando la dependency injection, l'oggetto responsabile della comunicazione con il server è inserito nella GUI in modo trasparente, consentendo una gestione più modulare e una maggiore facilità di manutenzione del codice.

2.3 Struttura comunicazione

Il client e il server comunicano utilizzando una struttura dati personalizzata, progettata per incorporare informazioni rilevanti per la comunicazione. Questa struttura dati include una sezione iniziale di byte che contiene metadati cruciali per la gestione dei messaggi, come il timestamp di invio e il nome del mittente. La progettazione di questa struttura dati consente una facile espansione in quanto la parte contenente i metadati è gestita separatamente dal resto dei dati del messaggio. Ciò significa che è possibile aggiungere ulteriori informazioni senza dover modificare significativamente la struttura o il processo di gestione dei dati. La codifica e decodifica dei messaggi in arrivo e ricezione è delegata a un oggetto apposito totalmente slegato dalla logica del Server e dei Client. Questo approccio favorisce la flessibilità e la scalabilità del sistema di comunicazione tra client e server.

3 Guida utilizzo applicazione

versione Python utilizzata : 3.12.3

3.1 Server

Per avviare il server, eseguire il file 'server.py' utilizzando l'interprete Python. È possibile specificare una porta personalizzata come argomento del comando, altrimenti il server utilizzerà la porta predefinita 8888.
Una volta avviato il server, esso entrerà in modalità di ascolto, pronto ad accettare connessioni in entrata dai client. Ogni volta che un nuovo client si connette, il server mostrerà un messaggio indicando l'avvenuta connessione e fornirà informazioni sul client appena connesso, come ad esempio il suo indirizzo IP e la porta utilizzata per la connessione. Inoltre, il server monitora attivamente le disconnessioni dei client. Quando un client si disconnette, il server mostra un messaggio indicando la disconnessione e fornisce il numero aggiornato di clienti attivi. La gestione delle connessioni e delle disconnessioni dei client permette al server di tenere traccia degli utenti attivi e di fornire un feedback immediato all'amministratore di sistema sullo stato corrente del server. Per terminare l'esecuzione del server, è sufficiente premere 'Ctrl + C' sulla tastiera per interrompere l'esecuzione del programma.

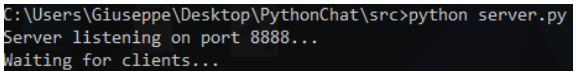

Figura 1: Avvio server

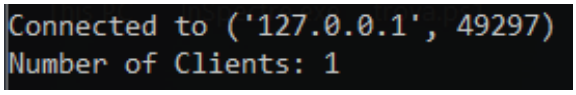

Figura 2: Connessione client

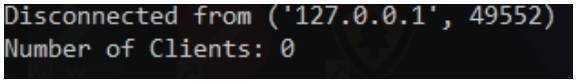

Figura 3: Disconnessione Client

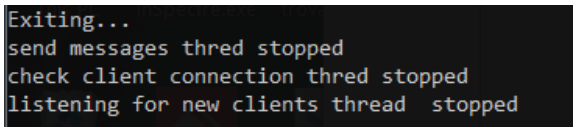

Figura 4: Chiusura Server, con relativo stop dei vari thread

3.2 Client

Per avviare la GUI del client, eseguire il file 'clientGUI.py' utilizzando l'interprete Python. Una volta avviata l'applicazione, verrà visualizzata una finestra che consente agli utenti di inserire il loro nickname e la porta del server a cui desiderano connettersi. Prima di tutto bisogna impostare il nickname e dopo sarà possibile collegarsi al server tramite un clic sul pulsante "Connect". Se la connessione viene stabilita con successo, sarà possibile inviare messaggi. All'interno della finestra della chat, è presente un campo di testo dove gli utenti possono digitare i loro messaggi. Dopo aver inserito il testo del messaggio, è sufficiente premere il tasto "Send" per inviare il messaggio al server e agli altri partecipanti alla chatroom. La finestra della chat visualizza i messaggi inviati e ricevuti, consentendo agli utenti di seguire la conversazione in corso e partecipare attivamente con i propri contributi.

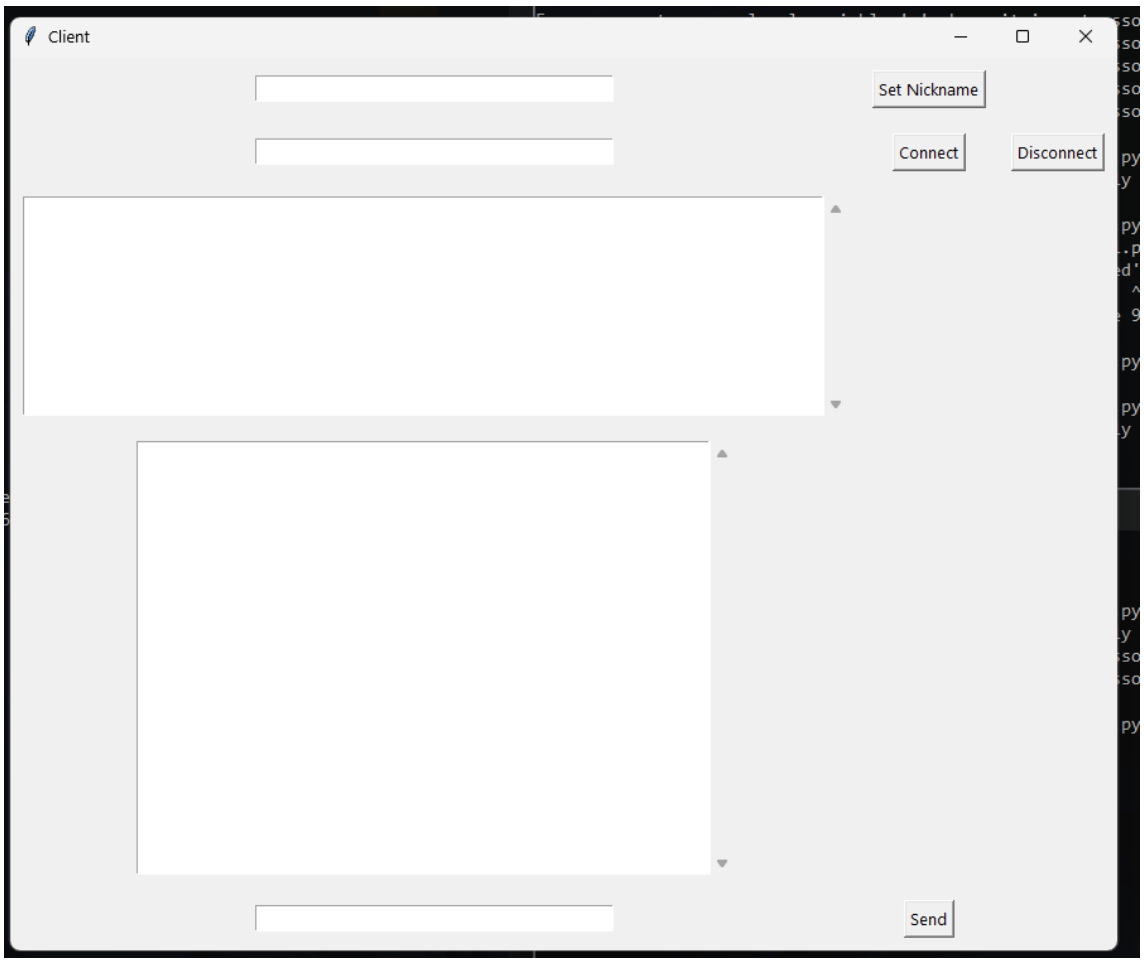

Figura 5: Interfaccia Client

4 Note di sviluppo

- Il software è stato testato a "mano", l'assenza di test è dovuta alla mia limitata esperienza con i test unitari, in particolare per applicazioni multi-thread. Tuttavia, è evidente che la mancanza di test automatizzati può impattare anche un progetto di queste dimensioni.
- Al momento, i messaggi di errore che si verificano alla disconnessione sono stati lasciati intenzionalmente visibili.