



Università degli Studi di Bologna  
Campus di Cesena

---

Corso di Laurea Triennale in Ingegneria e Scienze  
Informatiche

Relazione per il corso di Programmazione di Reti

## Progetto di un sistema di chat client-server

Candidato:  
Nicola Graziotin [nicola.graziotin@studenti.unibo.it](mailto:nicola.graziotin@studenti.unibo.it)  
Matricola 0001081557

Anno Accademico 2023/2024

## Indice

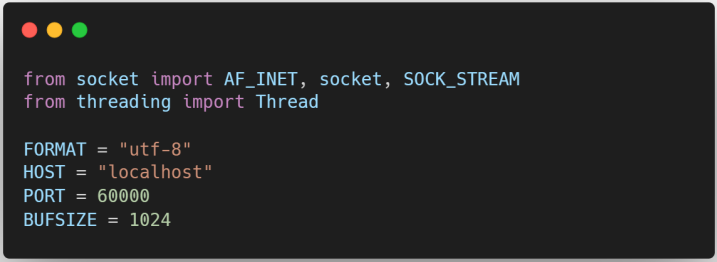
<b>1</b>	<b>Implementazione del server</b>	<b>2</b>
<b>2</b>	<b>Implementazione del client</b>	<b>2</b>
<b>3</b>	<b>Gestione degli errori e delle eccezioni</b>	<b>3</b>
<b>4</b>	<b>Funzionamento del sistema</b>	<b>4</b>
<b>5</b>	<b>Requisiti per eseguire il codice</b>	<b>4</b>
<b>6</b>	<b>Link alla repository</b>	<b>4</b>

# 1 Implementazione del server

Il server è implementato con una connessione TCP, è gestito da un thread che si mette in attesa finché qualche client si connetta.

Il server viene impostato sull'indirizzo di loopback "127.0.0.1", chiamato anche localhost, ed ho scelto la porta "60000".

Appena un client si connette, viene creato un nuovo thread che gestisce la ricezione di pacchetti. Se un pacchetto contiene una determinata stringa "close", il server chiude la connessione con il client che ha inviato la stringa.



```
from socket import AF_INET, socket, SOCK_STREAM
from threading import Thread

FORMAT = "utf-8"
HOST = "localhost"
PORT = 60000
BUFSIZE = 1024
```

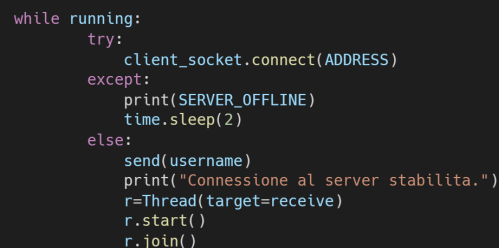
Il server si occupa di inviare in broadcast il messaggio corretto verso tutti i client connessi in ascolto.

# 2 Implementazione del client

Il client appena avviato, crea un thread principale che gestisce la corretta connessione con il server. Appena viene stabilita la connessione, avvia un thread che si mette in attesa di ricevere pacchetti dal server.

Se il server non è online oppure non risponde viene mostrato un messaggio che lo specifica.

Se la connessione viene interrotta, il thread di ricezione viene chiuso e il thread principale si occupa di ristabilire la connessione.



```
while running:
    try:
        client_socket.connect(ADDRESS)
    except:
        print(SERVER_OFFLINE)
        time.sleep(2)
    else:
        send(username)
        print("Connessione al server stabilita.")
        r=Thread(target=receive)
        r.start()
        r.join()
```

Quando l'utente invia un messaggio, il client si occupa di inviare il pacchetto contenente il messaggio.

Se l'utente chiude il programma, viene inviato un messaggio specifico al server "close", in modo da chiudere la connessione.

### 3 Gestione degli errori e delle eccezioni

Nell'implementazione del server viene gestita l'eccezione "ConnectionResetError", causata dall'interruzione involontaria della connessione con il client. Viene stampato l'errore e chiude il thread.

```
while True:
    try:
        msg = client.recv(BUFSIZE)
        if msg == bytes("{close}", FORMAT):
            close_client(client, ("%s:%s si è disconnesso." % client_address), nome)
            break
        broadcast(msg, nome+": ")
    except ConnectionResetError:
        close_client(client, ("%s:%s si è disconnesso a causa di un problema." % client_address), nome)
        break
```

Nell'implementazione del client viene gestita l'eccezione "ConnectionResetError", e vengono gestite "Exception" generiche nella ricezione di pacchetti, vengono gestite "Exception" generiche anche nell'invio di pacchetti e nella connessione con il server.

```
while running:
    try:
        message = client_socket.recv(BUFSIZE).decode(FORMAT)
        main_text.insert(tk.END, message)
    except ConnectionResetError:
        print("Connessione al server interrotta.")
        break
    except Exception:
        pass
```

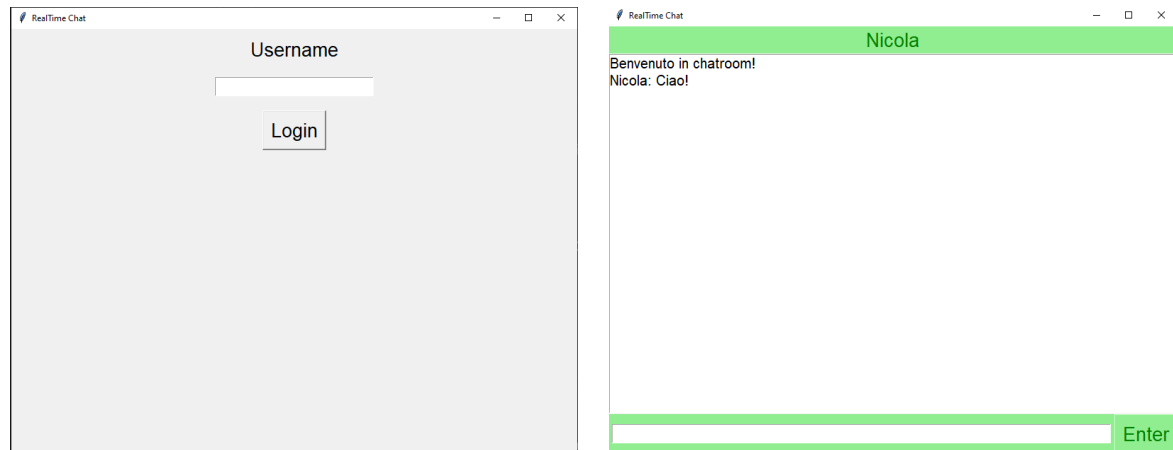
## 4 Funzionamento del sistema

Il sistema è composto da un server e uno o più client che si connettono con il server con un socket TCP. Il server appena viene avviato si mette in attesa di accettare connessioni.

L'utente avvia il programma che apre una finestra con un login. L'utente inserisce il suo username e preme il bottone "Login", verrà aperta la pagina della chat.

Per inviare un messaggio l'utente deve inserire il testo nella barra inferiore e premere il tasto invio della tastiera o in alternativa il bottone "Enter".

I messaggi verranno mostrati nell'area centrale del programma. Per uscire dal programma basta chiudere la finestra con la X in alto a destra.



## 5 Requisiti per eseguire il codice

Per eseguire il codice sono necessari i moduli:

- `tkinter`
- `time`
- `socket` (`AF_INET`, `socket`, `SOCK_STREAM`)
- `threading` (`Thread`)

## 6 Link alla repository

[github.com/NicolaGraziotin/Reti-RealTimeChat](https://github.com/NicolaGraziotin/Reti-RealTimeChat)