

Prompts used in creating application:

Can you make it so that the client get shown their own username when they show a message, the same way they see the usernames of others?

is there any way to make the text colored?

What I want it to do is keep some form of server state shared across all users, I want this functionality to make it so that

1. new users joining can see all previous messages since joining

2. messages can be deleted by sender if desired

3. when a user sends a message, it doesnt do this double print thing it does now, i.e. it shows once for me typing the text and entering, and is then echoed again with name could we make it so that when an operation such as delete happens, the entire terminal clears and the whole chat is resent to each client?

I didnt do the stuff you told me in my second to last prompt, could you combine those two into one so I can do it all step by step for the final result

can you tell me what exactly you are changing in chatclient and chatserver, like line by line changes

chatclient

```
import socket
```

```
import threading
```

```
import sys
```

```
from chatcore.protocol import encode_message, decode_message
```

```
from chatcore.tls import create_client_context, wrap_client_socket
```

```
if len(sys.argv) < 2:
```

```
    print("Usage: python ChatClient.py [SERVER_IP]")
```

```
    sys.exit(0)
```

```
SERVER = sys.argv[1]
```

```
PORT = 5555
```

```
username = input("Enter username: ")
```

```
def listen(sock):
```

```
    """Background thread to receive messages."""
```

```
    while True:
```

```
        try:
```

```
            raw = sock.recv(1024)
```

```
            if not raw:
```

```
                print("Server disconnected.")
```

```
                break
```

```
            sender, text = decode_message(raw)
```

```
            if sender:
```

```
                print(f"\n[ {sender} ] {text}")
```

```
            except:
```

```
breakdef start_client():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
        try:
            sock.connect((SERVER, PORT))
        except:
            return
        try:
            print("Could not connect to server.")
            tls_context = create_client_context(verify=False)
            tls_sock = wrap_client_socket(
                sock,
                tls_context,
                server_hostname=SERVER or "localhost",
            )
        except Exception as e:
            print(f"TLS handshake failed: {e}")
            return
        print("Connected over TLS! Type messages and press Enter.")
        thread = threading.Thread(target=listen, args=(tls_sock,), daemon=True)
        thread.start()
        while True:
            try:
                msg = input()
                encoded = encode_message(username, msg)
                tls_sock.sendall(encoded)
                print(f"[{username}] {msg}")
            except KeyboardInterrupt:
                print("\nExiting...")
                break
    if __name__ == "__main__":
        start_client()
        chat_server
import socket
import threading
from chatcore.protocol import encode_message, decode_message
from chatcore.handlers import broadcast_message, on_client_join, on_client_leave
from chatcore.tls import create_server_context, wrap_server_connection
HOST = "10.102.48.123"
PORT = 5555
clients = [] # holds sockets
def handle_client(conn, addr):
```

```
on_client_join(addr)
clients.append(conn)
try:
    while True:
        raw = conn.recv(1024)
        if not raw:
            break
        sender, text = decode_message(raw)
        if text is None:
            continue
        print(f"[{sender}] {text}")
        # Broadcast raw bytes exactly as received
        broadcast_message(sender, raw, clients, exclude=conn)
except ConnectionResetError:
    pass
finally:
    clients.remove(conn)
    on_client_leave(addr)
    conn.close()
def start_server():
    print(f"Server running on {HOST}:{PORT} ...")
    tls_context = create_server_context(
        certfile="server.crt",
        keyfile="server.key",
    )
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
        server.bind((HOST, PORT))
        server.listen()
        while True:
            conn, addr = server.accept()
            try:
                tls_conn = wrap_server_connection(conn, tls_context)
            except Exception as e:
                print(f"TLS handshake failed with {addr}: {e}")
                conn.close()
                continue
            thread = threading.Thread(target=handle_client, args=(tls_conn, addr),
                                      daemon=True)
            thread.start()
def handle_delete_request(self, client_id, msg_id):
    for msg in self.history:
```

```
if msg.msg_id == msg_id:
if msg.sender_id != client_id:
self.send_to(client_id, {
"type": "error",
"message": "You cannot delete someone else's message."
})
return
self.history = [m for m in self.history if m.msg_id != msg_id]
deletion_notice = {
"type": "delete_broadcast",
"msg_id": msg_id
}
return
self.broadcast(deletion_notice)
self.send_to(client_id, {
"type": "error",
"message": "Message ID not found."
})
if __name__ == "__main__":
start_server()
PS C:\Users\andyn\OneDrive\Desktop\3825 project4\NetworkProject> python3
ChatServer.py
Server running on 10.102.48.123:5555 ...
[+] Client connected: ('10.102.48.123', 58288)
[-] Client disconnected: ('10.102.48.123', 58288)
Exception in thread Thread-1 (handle_client):
Traceback (most recent call last):
File "C:\Program
Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2
kfra8p0\Lib\threading.py", line 1045, in _bootstrap_inner
self.run()
File "C:\Program
Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2
kfra8p0\Lib\threading.py", line 982, in run
self._target(*self._args, **self._kwargs)
File "C:\Users\andyn\OneDrive\Desktop\3825
project4\NetworkProject\ChatServer.py",
line 52, in handle_client
success = state.delete_message(msg_id, sender)
^^^^^^^
UnboundLocalError: cannot access local variable 'msg_id' where it is not associated
```

```
with a value
import socket
import threading
from chatcore.protocol import encode_message, decode_message
from chatcore.handlers import broadcast_message, on_client_join, on_client_leave
from chatcore.tls import create_server_context, wrap_server_connection
from chatcore.state import ChatState
from chatcore.protocol import encode_refresh
HOST = "10.102.48.123"
PORT = 5555
clients = [] # holds sockets
state = ChatState()
def send_full_history(conn):
    for msg in state.all_messages():
        display_text = f'# {msg["id"]} {msg["text"]}'
        payload = encode_message(msg["sender"], display_text)
        try:
            conn.sendall(payload)
        except:
            pass
def handle_client(conn, addr):
    on_client_join(addr)
    clients.append(conn)
    send_full_history(conn)
    try:
        while True:
            raw = conn.recv(1024)
            if not raw:
                break
            sender, text = decode_message(raw)
            if text is None:
                continue
            # /history request from client (used for refresh)
            if text == "/history":
                send_full_history(conn)
                continue
            # /delete command
            if text.startswith("/delete "):
                try:
                    msg_id = int(text.split()[1])
                except:
```

```
conn.sendall(encode_message("SYSTEM", "Usage: /delete <id>"))
continue
success = state.delete_message(msg_id, sender)
if not success:
    conn.sendall(encode_message("SYSTEM", f'Cannot delete message
#{msg_id}'))
continue
# Notify all users
broadcast_message("SYSTEM", encode_message("SYSTEM", f'Message
#{msg_id} deleted by {sender}'),clients)
# Tell all clients to refresh
refresh_msg = encode_refresh()
broadcast_message("SYSTEM", refresh_msg, clients)
continue
# NORMAL CHAT MESSAGE
stored = state.add_message(sender, text)
display_text = f'#{{stored['id']}} {{text}}'
payload = encode_message(sender, display_text)
print(f'[{{sender}}] {{display_text}}')
# Broadcast to EVERYONE (including sender)
broadcast_message(sender, payload, clients)
except ConnectionResetError:
    pass
finally:
    clients.remove(conn)
    on_client_leave(addr)
    conn.close()
def start_server():
    print(f"Server running on {HOST}:{PORT} ...")
    tls_context = create_server_context(
        certfile="server.crt",
        keyfile="server.key",
    )
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
        server.bind((HOST, PORT))
        server.listen()
    while True:
        conn, addr = server.accept()
        try:
            tls_conn = wrap_server_connection(conn, tls_context)
```

```
except Exception as e:  
    print(f"TLS handshake failed with {addr}: {e}")  
    conn.close()  
    continue  
thread = threading.Thread(target=handle_client, args=(tls_conn, addr),  
    daemon=True)  
thread.start()  
if __name__ == "__main__":  
    start_server()  
give me the whole handleclient func  
import socket  
import threading  
import sys  
import os  
import platform  
from chatcore.protocol import encode_message, decode_message  
from chatcore.tls import create_client_context, wrap_client_socket  
if len(sys.argv) < 2:  
    print("Usage: python ChatClient.py [SERVER_IP]")  
    sys.exit(0)  
SERVER = sys.argv[1]  
PORT = 5555  
username = input("Enter username: ")  
def clear_terminal():  
    if platform.system() == "Windows":  
        os.system("cls")  
    else:  
        os.system("clear")  
def refresh_chat(sock):  
    clear_terminal()  
    # Ask server for full history  
    sock.sendall(encode_message(username, "/history"))  
def listen(sock):  
    """Background thread to receive messages."""  
    while True:  
        try:  
            raw = sock.recv(1024)  
            if not raw:  
                print("Server disconnected.")  
                break  
            text = raw.decode(errors="ignore")
```

```
# Handle server refresh signal
if text.startswith("[REFRESH]"):
    refresh_chat(sock)
    continue
sender, decoded_text = decode_message(raw)
if sender:
    print(f"[{sender}] {decoded_text}")
except:
    break
def start_client():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
        try:
            sock.connect((SERVER, PORT))
        except:
            print("Could not connect to server.")
    return
tls_context = create_client_context(verify=False)
try:
    tls_sock = wrap_client_socket(
        sock,
        tls_context,
        server_hostname=SERVER or "localhost",
    )
except Exception as e:
    print(f"TLS handshake failed: {e}")
    return
print("Connected over TLS! Type messages and press Enter.")
thread = threading.Thread(target=listen, args=(tls_sock,), daemon=True)
thread.start()
while True:
    try:
        msg = input()
        encoded = encode_message(username, msg)
        tls_sock.sendall(encoded)
    except KeyboardInterrupt:
        print("\nExiting...")
        break
if __name__ == "__main__":
    start_client()
can you give me proper server code for handle client?
import socket
```

```
import threading
from chatcore.protocol import encode_message, decode_message
from chatcore.handlers import broadcast_message, on_client_join, on_client_leave
from chatcore.tls import create_server_context, wrap_server_connection
from chatcore.state import ChatState
from chatcore.protocol import encode_refresh
HOST = "10.0.0.115"
PORT = 5555
clients = [] # holds sockets
state = ChatState()
def send_full_history(conn):
    for msg in state.all_messages():
        display_text = f"# {msg['id']} {msg['text']}"
        payload = encode_message(msg["sender"], display_text)
        try:
            conn.sendall(payload)
        except:
            pass
def handle_client(conn, addr):
    on_client_join(addr)
    clients.append(conn)
    send_full_history(conn)
    try:
        while True:
            raw = conn.recv(1024)
            if not raw:
                break
            sender, text = decode_message(raw)
            if text is None:
                continue
            # /history request from client (used for refresh)
            if text == "/history":
                send_full_history(conn)
                continue
            # /delete command
            if text.startswith("/delete "):
                try:
                    msg_id = int(text.split()[1])
                except:
                    conn.sendall(encode_message("SYSTEM", "Usage: /delete <id>"))
                    continue
```

```
success = state.delete_message(msg_id, sender)
if not success:
    conn.sendall(encode_message("SYSTEM", f'Cannot delete message
#{msg_id}'))
    continue
# Notify all users
broadcast_message("SYSTEM", encode_message("SYSTEM", f'Message
#{msg_id} deleted by {sender}'), clients)
# Tell all clients to refresh
refresh_msg = encode_refresh()
broadcast_message("SYSTEM", refresh_msg, clients)
continue
# NORMAL CHAT MESSAGE
stored = state.add_message(sender, text)
display_text = f'#{stored["id"]} {text}'
payload = encode_message(sender, display_text)
print(f'[ {sender} ] {display_text}')
# Broadcast to EVERYONE (including sender)
broadcast_message(sender, payload, clients)
except ConnectionResetError:
    pass
finally:
    clients.remove(conn)
    on_client_leave(addr)
    conn.close()
def start_server():
    print(f"Server running on {HOST}:{PORT} ...")
    tls_context = create_server_context(
        certfile="server.crt",
        keyfile="server.key",
    )
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
        server.bind((HOST, PORT))
        server.listen()
    while True:
        conn, addr = server.accept()
        try:
            tls_conn = wrap_server_connection(conn, tls_context)
        except Exception as e:
            print(f'TLS handshake failed with {addr}: {e}')
            conn.close()
```

```
continue
thread = threading.Thread(target=handle_client, args=(tls_conn, addr),
daemon=True)
thread.start()
if __name__ == "__main__":
start_server()
fixed it, I dont like where this has gotten, can we make the commands feature a module
for adding numerous commands?
I rolled back a bit, here is my current chatclient and chatserver
chatserver
import socket
import threading
from chatcore.protocol import encode_message, decode_message
from chatcore.handlers import broadcast_message, on_client_join, on_client_leave
from chatcore.tls import create_server_context, wrap_server_connection
HOST = "10.0.0.115"
PORT = 5555
clients = [] # holds sockets
def handle_client(conn, addr):
on_client_join(addr)
clients.append(conn)
try:
while True:
raw = conn.recv(1024)
if not raw:
break
sender, text = decode_message(raw)
if text is None:
continue
print(f"[{sender}] {text}")
# Broadcast raw bytes exactly as received
broadcast_message(sender, raw, clients, exclude=conn)
except ConnectionResetError:
pass
finally:
clients.remove(conn)
on_client_leave(addr)
conn.close()
def start_server():
print(f"Server running on {HOST}:{PORT} ...")
tls_context = create_server_context()
```

```
certfile="server.crt",
keyfile="server.key",
)
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
    server.bind((HOST, PORT))
    server.listen()
    while True:
        conn, addr = server.accept()
        try:
            tls_conn = wrap_server_connection(conn, tls_context)
        except Exception as e:
            print(f"TLS handshake failed with {addr}: {e}")
            conn.close()
            continue
        thread = threading.Thread(target=handle_client, args=(tls_conn, addr),
                                  daemon=True)
        thread.start()
    def handle_delete_request(self, client_id, msg_id):
        for msg in self.history:
            if msg.msg_id == msg_id:
                if msg.sender_id != client_id:
                    self.send_to(client_id, {
                        "type": "error",
                        "message": "You cannot delete someone else's message."
                    })
        return
        self.history = [m for m in self.history if m.msg_id != msg_id]
    deletion_notice = {
        "type": "delete_broadcast",
        "msg_id": msg_id
    }
    return
    self.broadcast(deletion_notice)
    self.send_to(client_id, {
        "type": "error",
        "message": "Message ID not found."
    })
if __name__ == "__main__":
    start_server()
    chatClient
    import socket
```

```
import threading
import sys
from chatcore.protocol import encode_message, decode_message
from chatcore.tls import create_client_context, wrap_client_socket
if len(sys.argv) < 2:
    print("Usage: python ChatClient.py [SERVER_IP]")
    sys.exit(0)
SERVER = sys.argv[1]
PORT = 5555
username = input("Enter username: ")
def listen(sock):
    """Background thread to receive messages."""
while True:
    try:
        raw = sock.recv(1024)
        if not raw:
            print("Server disconnected.")
            break
        sender, text = decode_message(raw)
        if sender:
            print(f"\n[{sender}] {text}")
        except:
            break
    def start_client():
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
            try:
                sock.connect((SERVER, PORT))
            except:
                return
            try:
                print("Could not connect to server.")
                tls_context = create_client_context(verify=False)
                tls_sock = wrap_client_socket(
                    sock,
                    tls_context,
                    server_hostname=SERVER or "localhost",
                )
            except Exception as e:
                print(f"TLS handshake failed: {e}")
            return
        print("Connected over TLS! Type messages and press Enter.")
```

```
thread = threading.Thread(target=listen, args=(tls_sock,), daemon=True)
thread.start()
while True:
    try:
        msg = input()
        encoded = encode_message(username, msg)
        tls_sock.sendall(encoded)
    except KeyboardInterrupt:
        print("\nExiting...")
        break
    if __name__ == "__main__":
        start_client()
how are we going to implement delete? are we going to add message numbers?
I kept state.py from before
# chatcore/state.py
# Shared in-memory state for all clients.
import threading
class ChatState:
    def __init__(self):
        self._messages = [] # list of dicts: {"id", "sender", "text", "deleted"}
        self._next_id = 1
        self._lock = threading.Lock()
    def add_message(self, sender, text):"""Store a new message and return its record."""
        with self._lock:
            mid = self._next_id
            self._next_id += 1
            msg = {
                "id": mid,
                "sender": sender,
                "text": text,
                "deleted": False,
            }
            self._messages.append(msg)
        return msg
    def delete_message(self, msg_id, requester):
        """Mark a message as deleted if requester is the sender."""
        with self._lock:
            for msg in self._messages:
                if msg["id"] == msg_id:
                    if msg["sender"] != requester:
```

```
return False # not your message
msg["deleted"] = True
msg["text"] = "[deleted]"
return True
return False # not found
def all_messages(self):
    """Return a shallow copy of the current message list."""
    with self._lock:
        return list(self._messages)
where is this?
stored = state.add_message(sender, text)
display_text = f"# {stored['id']} {text}"
payload = encode_message(sender, display_text)
broadcast_message(sender, payload, clients)
state not being recognized in server
so where we are now I need two things, lets start with number one
I need the input of a message to be prettier, right now its messy, instead of this looking
like
wssp
[andy] #2 wssp
i need it to look like
[andy] wsspor
[andy] #2 wssp
I want it to say [andy] on the left while they are typing, and then I want to avoid the
double print basically
Connected over TLS! Type messages and press Enter.
[ibll] wassup
[ibll]
[ibll] #2 wassup
Connected over TLS! Type messages and press Enter.
[bill] hi
[bill] [bill] #3 hi
obvious issues, plus the [name] part only appears the first time
import socket
import threading
import sys
from chatcore.protocol import encode_message, decode_message
from chatcore.tls import create_client_context, wrap_client_socket
if len(sys.argv) < 2:
    print("Usage: python ChatClient.py [SERVER_IP]")
```

```
sys.exit(0)
SERVER = sys.argv[1]
PORT = 5555
username = input("Enter username: ")
def listen(sock):
    """Background thread to receive messages."""
    while True:
        try:
            raw = sock.recv(1024)
            if not raw:
                print("Server disconnected.")
                break
            sender, text = decode_message(raw)
            if sender:
                sys.stdout.write("\r") # return to start of line
            sys.stdout.write(f"[ {sender} ] {text}\n")
            sys.stdout.write(f"[ {username} ] ") # redraw prompt
            sys.stdout.flush()
        except:
            break
    def start_client():
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
            try:
                sock.connect((SERVER, PORT))
            except:
                return
            try:
                print("Could not connect to server.")
                tls_context = create_client_context(verify=False)
                tls_sock = wrap_client_socket(
                    sock,
                    tls_context,
                    server_hostname=SERVER or "localhost",
                )
            except Exception as e:
                print(f"TLS handshake failed: {e}")
            return
        print("Connected over TLS! Type messages and press Enter.")
        thread = threading.Thread(target=listen, args=(tls_sock,), daemon=True)
        thread.start()
        while True:
            try:
```

```
msg = input(f"[{username}] ")  
if not msg.strip():  
    continue # ignore empty inputs  
encoded = encode_message(username, msg)  
tls_sock.sendall(encoded)  
except KeyboardInterrupt:  
    print("\nExiting...")  
    break  
if __name__ == "__main__":  
    start_client()  
good?  
so close,  
Connected over TLS! Type messages and press Enter.  
[andy] yo  
[andy] #1 yo  
[bill] #2 whats up bro  
[andy]  
nope not that, I mean the [andy] yo before [andy] #1 yo  
i just want [andy] #1 yo, that second one is the problemis this correct?  
import socket  
import threading  
import sys  
from chatcore.protocol import encode_message, decode_message  
from chatcore.tls import create_client_context, wrap_client_socket  
if len(sys.argv) < 2:  
    print("Usage: python ChatClient.py [SERVER_IP]")  
    sys.exit(0)  
SERVER = sys.argv[1]  
PORT = 5555  
username = input("Enter username: ")  
def listen(sock):  
    """Background thread to receive messages."""  
    while True:  
        try:  
            raw = sock.recv(1024)  
            if not raw:  
                print("Server disconnected.")  
                break  
            sender, text = decode_message(raw)  
            if sender:
```

```
sys.stdout.write("\r") # return to start of line
sys.stdout.write(f"[{sender}] {text}\n")
sys.stdout.write(f"[{username}] ") # redraw prompt
sys.stdout.flush()
except:
break
def start_client():
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
try:
sock.connect((SERVER, PORT))
except:
print("Could not connect to server.")
return
tls_context = create_client_context(verify=False)
try:
tls_sock = wrap_client_socket(
sock,
tls_context,server_hostname=SERVER or "localhost",
)
except Exception as e:
print(f"TLS handshake failed: {e}")
return
print("Connected over TLS! Type messages and press Enter.")
thread = threading.Thread(target=listen, args=(tls_sock,), daemon=True)
thread.start()
while True:
try:
msg = input(f"[{username}] ")
if not msg.strip():
continue # ignore empty inputs
sys.stdout.write("\r")
sys.stdout.flush()
encoded = encode_message(username, msg)
tls_sock.sendall(encoded)
except KeyboardInterrupt:
print("\nExiting...")
break
if __name__ == "__main__":
start_client()
sadly, it does not
```

Connected over TLS! Type messages and press Enter.

[andy] wassup
[andy] #1 wassup
[andy]

nope

Enter username: andy

Connected over TLS! Type messages and press Enter.

[bill] #1 wassup
[andy] fuck
[andy] #2 fuck
[andy]

would it not be easier to just not resend my own messages from server and add the message id to client or something (idk how to fix race condition issue
screw delete, before we add anything how are we dealing with the latency issue on ids?
I might assign id 17 to a message as I send it, while the server has already assigned that to a message currently on its way to me from someone else, how do we solve I feel like were doing entirely too much here, there has to be some way to just completely remove the output from entering something in terminal
please do not use slang terms like cooked or cursed.

Im over this, lets just leave it be, if needed the chat can just be refreshed
is /history just going to be a refresh? if so, lets rename it /refresh and just have it clear the terminal and echo the entire conversation history

where did my message numbers go?

now I just need the refresh to have message ids
clear terminal not working

where is this

```
if msg.strip() == "/refresh":  
    clear_terminal() # <-- first  
    encoded = encode_message(username, "/refresh")  
    tls_sock.sendall(encoded)  
    continue
```

?

/delete id currently does nothing

I don't really like how we're doing commands, is there any way we can implement a basic system to capture any commands, check if they are valid, then execute them? I feel like our current approach is becoming spaghetti code
please be more specific with where these chunks should go

how can I make it so all of the currently active users messages are red (and make it so it can be changed with a command)

can you just give me a copy paste of my whole client file

I would like to add a command /exit for the user to exit the application
just gimme the new client file

perfect, now I need to add ID numbers to client messages to prevent multiple people
with the same name

lets do [name#0000], make it so that no two people in the chat have the same number,
additionally can we cap the people that can be connected at 1000?

sorry, I meant 10000 people

give me full server file

can we change it to a limit of 10000 starting at 0 not 1, I want it to be #0000 - #9999

perfect, but im noticing now my messages arent colored until I refresh?

I would like the chat to refresh every time a message is received by client, how should I
do this

stop, tell me exactly what to change and where, this code behaves differently than the
previous in more ways than I wanted

how can I add color to my messages printed via chat history

can we add a way to search through messages for a certain string Every so often its like the
memory of these AI chats refreshes and you forget things,

here is our chatserver code, lets implement searching using the dynamic command
system implemented earlier, we should only need big changes in commands and state
files

```
import socket
```

```
import threading
```

```
from chatcore.protocol import encode_message, decode_message
```

```
from chatcore.handlers import broadcast_message, on_client_join, on_client_leave
```

```
from chatcore.tls import create_server_context, wrap_server_connection
```

```
from chatcore.state import ChatState
```

```
from chatcore.commands import CommandContext, handle_command
```

```
HOST = "10.0.0.115" # change if needed
```

```
PORT = 5555
```

```
clients = [] # active TLS sockets
```

```
state = ChatState() # shared message history
```

```
# ----- CLIENT ID MANAGEMENT -----
```

```
next_client_id = 0 # will increment per client
```

```
client_ids = {} # conn -> numeric ID
```

```
MAX_CLIENTS = 10000 # hard user cap
```

```
def send_full_history(conn):
```

```
    """
```

Sends the entire chat history to a single client.

Each message is shown with its message ID (#n).

```
    """
```

```
for msg in state.all_messages():
    display_text = f'#{msg['id']} {msg['text']}'
    payload = encode_message(msg["sender"], display_text)
    try:
        conn.sendall(payload)
    except Exception:
        break
def handle_client(conn, addr):
    """
    Handle a single client connection.
    Parses commands and broadcasts messages with unique sender IDs.
    """
    global next_client_id
    on_client_join(addr)
    clients.append(conn)
    # ---- ASSIGN CLIENT ID ----
    client_id = next_client_id
    next_client_id += 1
    client_ids[conn] = client_id
    # Tell the client their unique ID
    conn.sendall(encode_message("SYSTEM", f'/id {client_id}'))
    try:
        while True:
            raw = conn.recv(1024)
            if not raw:
                break
            sender, text = decode_message(raw)
            if text is None:
                continue
            # Build command context
            ctx = CommandContext(
                conn=conn,
                sender=sender,
                state=state,
                clients=clients,
                send_full_history=send_full_history,
                broadcast_message=broadcast_message,
            )
            # 1) Run commands like /refresh, /delete
            if handle_command(ctx, text):
                continue # command handled, skip normal broadcast
```

```

# 2) NORMAL MESSAGE → store and broadcast
stored = state.add_message(sender, text)
display_text = f'#{stored['id']} {text}'
# sender label WITH ID
sender_id = client_ids[conn]
sender_label = f'{sender}#{sender_id:04d}' # zero-padded 5 digits
print(f'{sender_label} {display_text}')
payload = encode_message(sender_label, display_text)
broadcast_message(sender_label, payload, clients)
except ConnectionResetError:
    # client killed connection
    pass
finally:
    # cleanup
    if conn in clients:clients.remove(conn)
    if conn in client_ids:
        del client_ids[conn]
        on_client_leave(addr)
        conn.close()
def start_server():
    print(f"Server running on {HOST}:{PORT} ...")
    # TLS configuration
    tls_context = create_server_context(
        certfile="server.crt",
        keyfile="server.key",
    )
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
        server.bind((HOST, PORT))
        server.listen()
        while True:
            conn, addr = server.accept()
            # PREVENT OVER-CAPACITY
            if len(clients) >= MAX_CLIENTS:
                conn.sendall(encode_message("SYSTEM", "Server full (10,000 users
max)."))
                conn.close()
                continue
            # TLS handshake
            try:
                tls_conn = wrap_server_connection(conn, tls_context)

```

```
except Exception as e:  
    print(f"TLS handshake failed with {addr}: {e}")  
    conn.close()  
    continue  
    thread = threading.Thread(  
        target=handle_client,  
        args=(tls_conn, addr),  
        daemon=True  
    )  
    thread.start()  
if __name__ == "__main__":  
    start_server()  
what is @register?  
tell me how to do it without that unknown command search  
# chatcore/commands.py  
# Central command handling for the chat server.  
from dataclasses import dataclass  
from typing import Callable, List, Any  
from chatcore.protocol import encode_message  
@dataclass  
class CommandContext:  
    conn: Any # the client's socket  
    sender: str # username of the client  
    state: Any # ChatState instance  
    clients: List[Any] # list of all client sockets  
    send_full_history: Callable # function(conn) -> None  
    broadcast_message: Callable # function(sender, payload, clients)  
    # encode_message already imported here  
    def cmd_refresh(ctx: CommandContext, args: str) -> None:  
        """  
        /refresh  
        Clears the client's screen (on client side) and re-sends history.  
        Server side: just send full history to this client.  
        """  
        ctx.send_full_history(ctx.conn)  
    def cmd_delete(ctx: CommandContext, args: str) -> None:  
        """  
        /delete <id>  
        Marks one of the sender's messages as deleted.  
        """
```

```

args = args.strip()
if not args:
    return
ctx.conn.sendall(encode_message("SYSTEM", "Usage: /delete <id>"))
try:
    msg_id = int(args.split()[0])
except ValueError:
    ctx.conn.sendall(encode_message("SYSTEM", "Usage: /delete <id>"))
return
success = ctx.state.delete_message(msg_id, ctx.sender)
if not success:
    ctx.conn.sendall(
        encode_message("SYSTEM", f"Cannot delete message #{msg_id}"))
return
# Notify all clients that deletion occurred
notice = encode_message(
    "SYSTEM", f"Message #{msg_id} deleted by {ctx.sender}")
)
ctx.broadcast_message("SYSTEM", notice, ctx.clients)
# Registry of supported commands
COMMANDS = {
    "refresh": cmd_refresh,
    "delete": cmd_delete,
}
def handle_command(ctx: CommandContext, text: str) -> bool:
    """
    Detect and execute a slash command.
    Returns:
        True -> the text was a command (handled or rejected)
        False -> not a command; treat as a normal chat message
    """
    if not text.startswith("/"):
        return False
    # Strip leading "/" and split into name + args
    body = text[1:].strip()
    if not body:
        ctx.conn.sendall(
            encode_message("SYSTEM", "Empty command. Try /refresh or /delete <id>."))

    return True

```

```
parts = body.split(maxsplit=1)
name = parts[0].lower()
args = parts[1] if len(parts) > 1 else ""
cmd = COMMANDS.get(name)
if cmd is None:
    ctx.conn.sendall(
        encode_message("SYSTEM", f"Unknown command: /{name}")
    )
return True
cmd(ctx, args)
return True

def cmd_search(ctx, query):
    if not query.strip():
        ctx.conn.sendall(encode_message("SYSTEM", "Usage: /search <text>"))
        return
    matches = ctx.state.search(query)
    if not matches:
        ctx.conn.sendall(encode_message("SYSTEM", f"No messages contain '{query}'"))
        return
    ctx.conn.sendall(encode_message("SYSTEM", f"Found {len(matches)} match(es):"))
    for msg in matches:
        display = f"#{{msg['id']}} {msg['text']}"
        payload = encode_message(msg["sender"], display)
        ctx.conn.sendall(payload)
COMMANDS["/search"] = cmd_search

# chatcore/state.py
# Shared in-memory state for all clients.
import threading
class ChatState:
    def __init__(self):
        # list of dicts: {"id", "sender", "text", "deleted"}
        self._messages = []
        self._next_id = 1
        self._lock = threading.Lock()
    def add_message(self, sender: str, text: str) -> dict:
        """Store a new message and return its record."""
        with self._lock:
            msg_id = self._next_id
            self._next_id += 1
            msg = {
                "id": msg_id,
```

```

"sender": sender,
"text": text,
"deleted": False,
}
self._messages.append(msg)
return msg
def search(self, query):
query = query.lower()
with self._lock:return [
msg for msg in self._messages
if query in msg["text"].lower()
]
def delete_message(self, msg_id: int, requester: str) -> bool:
"""
Mark a message as deleted if the requester is the sender.
Returns True if deleted, False if not found or not allowed.
"""
with self._lock:
for msg in self._messages:
if msg["id"] == msg_id:
if msg["sender"] != requester:
return False
msg["deleted"] = True
msg["text"] = "[deleted]"
return True
return False
def all_messages(self) -> list:
"""Return a shallow copy of the current message list."""
with self._lock:
return list(self._messages)

```

works great, now lets make it so when a client joins the chat, all other clients are notified
why cant I upload a file to this chat?
can you look at this copy pasted document and check if my project has met all requirements?

COMP Network/Info Assurance
Course Project
This project
intends you to design a simple, fully functional “Client/Server Reliable Chat Application” in your favorite programming language. The application should have following component.

Fig: - Simple Client Server Chat

Application Diagram

The application

should implement the following functionalities:-

The

application will contain a server and two or more than two clients (for chatting)

-

Each

client should be able to connect with the server. Once connected, the server will assign a unique identifier name to each of these clients.

-

Once

a new client (say Client A) is connected to the server, in response, the server will provide the list of all the available clients connected to it. In the simplest scenario, you can just have two clients connected to the server.

-

Client

(i.e. Client A) will use the identification name provided by the server to send the message to another client (say Client B).

-

You

can decide the implementation you want for an identification number. (It is recommended to use a combination of Name + random number, so that a client can know with whom he/she is talking to).

-

Once

the server receives the message from a client (Client A), it will forward the message to the intended client (Client B) and vice-versa. (For simplicity: only implement for the case when both of the clients are available for the chat).

-

A

client can disconnect from the server by sending “.exit” message anytime. Server upon receiving “.exit” message from the client, will close the connection with it.

The assignment.

Augment the regular messaging app with some other

modern feature. There are many existing implementations of a simple python chat on the internet. You will get one of them working and then add 4 features.

- Implement chatting between multiple (more than 2) clients

- Implement simple encryption (e.g TLS) [1]

- Message receipt confirmation

- Replies to previous messages

- Message searching

- Message deletion

- Temporary messages

- Any other feature you can think of.

You can choose

any programming language (Python/Java preferable) to develop a chat application. However, it is your responsibility to make sure your code is easy to understand and run. Therefore, you should provide helpful comments in the code. During your development/demonstration, you can run the clients and server on the same machine.

Important: Please don't forget to cite the sources you take help from. Particularly if you use GENAI, list the prompts used to generate your solution

Groups: 2-3 person in one group (no more than 3. Can be individual)

Deliverables:

- Initial

Steps:-

Pick

your team

- Choose

a tutorial or use GenAI

-
Working
Chat Application

-
Use
a tutorial to create a working chat application.

-
Submit
works.
the code and a document describing the tutorial, how you implemented it, how it

-
Augmentation
Plan

-
Create
a design document for the chat application. In addition to the text description, use class diagram, flow chart, or state transition diagram to explain your tentative implementation and design changes. (20 pts)

-
Code
of Framework:
Submit code framework for both client and server. It should have the major data structures and functions (APIs) defined based on the design document, and the connection between clients and server. (20 pts)

-
D4
Final Report and Code:

Submit a final report (>= 4 pages) including design and evaluation results in the report, and final code showing (a) communication between the clients, (b) instructions on how to install and run the program, (c) screenshots of the input and output by 11:59 pm, Nov 30th (30 pts)
New References

<https://pythonprogramming.net/server-chatroom-sockets-tutorial-python-3/>

<https://www.geeksforgeeks.org/simple-chat-room-using-python/>

<https://github.com/TiagoValdrich/python-socket-chat>

<https://pandeyshikha075.medium.com/building-a-chat-server-and-client-in-python-with-socket-programming-c76de52cc1d5>

Older Reference & Help

[1] “TLS/SSL

wrapper for socket objects”, <https://docs.python.org/2/library/ssl.html>

[2] “Creating a

simple chat Client/Server application”

<http://pirate.shu.edu/~wachsmut/Teaching/CSAS2214/Virtual/Lectures/chat-client-server.html>

[3] “NETWORK

PROGRAMMING - SERVER & CLIENT A : BASICS”

https://bogotobogo.com/python/python_network_programming_server_client.php

can you give me code to add sending client list to new clients? i forgot to add that username not defined

here is current handle client, give me fixed one to copy paste

def handle_client(conn, addr):

"""

Handle a single client connection.

Parses commands and broadcasts messages with unique sender IDs.

"""

```
global next_client_id
```

```
on_client_join(addr)
```

```
clients.append(conn)
```

```
# ---- ASSIGN CLIENT ID ----
```

```
client_id = next_client_id
```

```
next_client_id += 1
```

```
client_ids[conn] = client_id
```

```
# Tell the client their unique ID
```

```
conn.sendall(encode_message("SYSTEM", f"/id {client_id}"))
```

```
# ---- SEND LIST OF CURRENTLY CONNECTED CLIENTS ----
```

```
if len(client_ids) > 1: connected = []
```

```
for c, cid in client_ids.items():
```

```
if c is not conn: # exclude the joining client
```

```
# Note: we do NOT know each user's name until they send first message
```

```
connected.append(f"{username}#{cid:04d}")
```

```
if connected:
```

```
msg = "Connected clients: " + ", ".join(connected)
```

```
try:
```

```
conn.sendall(encode_message("SYSTEM", msg))
```

```
except:
```

```
pass
```

```
else:
```

```
# They are the first client
```

```
conn.sendall(encode_message("SYSTEM", "You are the first client here."))
try:
    while True:
        raw = conn.recv(1024)
        if not raw:
            break
        sender, text = decode_message(raw)
        if text is None:
            continue
        # Build command context
        ctx = CommandContext(
            conn=conn,
            sender=sender,
            state=state,
            clients=clients,
            send_full_history=send_full_history,
            broadcast_message=broadcast_message,
        )
        # 1) Run commands like /refresh, /delete
        if handle_command(ctx, text):
            continue # command handled, skip normal broadcast
        # 2) NORMAL MESSAGE → store and broadcast
        stored = state.add_message(sender, text)
        display_text = f"#{{stored['id']}} {{text}}"
        # sender label WITH ID
        sender_id = client_ids[conn]sender_label = f"{{sender}}{{sender_id:04d}}" # zero-padded 5 digits
        print(f"[{{sender_label}}] {{display_text}}")
        payload = encode_message(sender_label, display_text)
        broadcast_message(sender_label, payload, clients)
    except ConnectionResetError:
        # client killed connection
        pass
    finally:
        # cleanup
        if conn in clients:
            clients.remove(conn)
        if conn in client_ids:
            del client_ids[conn]
            on_client_leave(addr)
        conn.close()
```

this still gives clients#xxxx i want username#xxxx
give me copy and drop changes
the first client still stays as client not username, why
idk where to drop client code
client usernames not defined in commands.py
Server running on 10.0.0.115:5555 ...
[+] Client connected: ('10.0.0.115', 55745)
[+] Client connected: ('10.0.0.115', 55747)
[-] Client disconnected: ('10.0.0.115', 55747)
Exception in thread Thread-2 (handle_client):
Traceback (most recent call last):
File "C:\Program
Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2
kfra8p0\Lib\threading.py", line 1045, in _bootstrap_inner
self.run()
File "C:\Program
Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2
kfra8p0\Lib\threading.py", line 982, in run
self._target(*self._args, **self._kwargs)
File "C:\Users\andyn\OneDrive\Desktop\3825 project4\NetworkProject\ChatServer.py",
line 91, in handle_client
ctx = CommandContext(
^^^^^^^^^^^^^

TypeError: CommandContext.__init__() missing 1 required positional argument:
'client_usernames'how hard would it be to add a /whisper user message command to send private
messages that are not seen by users other than sender or receiver (even on refresh)
pause that, previous didnt work
PS C:\Users\andyn\OneDrive\Desktop\3825 project4\NetworkProject> python3
ChatServer.py
Server running on 10.0.0.115:5555 ...
[+] Client connected: ('10.0.0.115', 57734)
[+] Client connected: ('10.0.0.115', 57736)
[-] Client disconnected: ('10.0.0.115', 57736)
Exception in thread Thread-2 (handle_client):
Traceback (most recent call last):
File "C:\Program
Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2
kfra8p0\Lib\threading.py", line 1045, in _bootstrap_inner
self.run()
File "C:\Program

```
File "C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2\kfra8p0\Lib\threading.py", line 982, in run
    self._target(*self._args, **self._kwargs)
File "C:\Users\andyn\OneDrive\Desktop\3825 project4\NetworkProject\ChatServer.py", line 78, in handle_client
    raw = conn.recv(1024)
^^^^^^^^^^^^^^^^^

File "C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2\kfra8p0\Lib\ssl.py", line 1295, in recv
    return self.read(buflen)
^^^^^^^^^^^^^^^^^

File "C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2\kfra8p0\Lib\ssl.py", line 1168, in read
    return self._sslobj.read(len)
^^^^^^^^^^^^^^^^^

ssl.SSLError: [SSL: DECRYPTION_FAILED_OR_BAD_RECORD_MAC] decryption failed or bad record mac (_ssl.c:2580)
[-] Client disconnected: ('10.0.0.115', 57734)
Exception in thread Thread-1 (handle_client):
Traceback (most recent call last):
File "C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2\kfra8p0\Lib\threading.py", line 1045, in _bootstrap_inner
    self.run()
File "C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2\kfra8p0\Lib\threading.py", line 982, in run
    self._target(*self._args, **self._kwargs)
File "C:\Users\andyn\OneDrive\Desktop\3825 project4\NetworkProject\ChatServer.py", line 78, in handle_client
    raw = conn.recv(1024)
^^^^^^^^^^^^^^^^^

File "C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2\kfra8p0\Lib\ssl.py", line 1295, in recv
    return self.read(buflen)
^^^^^^^^^^^^^^^^^

File "C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2\kfra8p0\Lib\ssl.py", line 1168, in read
    return self._sslobj.read(len)
^^^^^^^^^^^^^^^^^
```

```
kfra8p0\Lib\ssl.py", line 1168, in read
    return self._sslobj.read(len)
^^^^^^^^^^^^^^^^^^^^^
ssl.SSLError: [SSL: DECRYPTION_FAILED_OR_BAD_RECORD_MAC] decryption
failed or bad record mac (_ssl.c:2580)
where to put client code snippet
PS C:\Users\andyn\OneDrive\Desktop\3825 project4\NetworkProject> python3
ChatServer.py
Server running on 10.0.0.115:5555 ...
[+] Client connected: ('10.0.0.115', 65002)
[-] Client disconnected: ('10.0.0.115', 65002)
Exception in thread Thread-1 (handle_client):
Traceback (most recent call last):
File "C:\Program
Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2
kfra8p0\Lib\threading.py", line 1045, in _bootstrap_inner
self.run()
File "C:\Program
Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2
kfra8p0\Lib\threading.py", line 982, in run
self._target(*self._args, **self._kwargs)File "C:\Users\andyn\OneDrive\Desktop\3825
project4\NetworkProject\ChatServer.py",
line 96, in handle_client
ctx = CommandContext(
^^^^^^^^^^^^^^^^^
TypeError: CommandContext.__init__() missing 1 required positional argument:
'client_usernames'
every client disconnects now except first
server code
import socket
import threading
import ssl
from chatcore.protocol import encode_message, decode_message
from chatcore.handlers import broadcast_message, on_client_join, on_client_leave
from chatcore.tls import create_server_context, wrap_server_connection
from chatcore.state import ChatState
from chatcore.commands import CommandContext, handle_command
HOST = "10.0.0.115" # change if needed
PORT = 5555
clients = [] # active TLS sockets
```

```

state = ChatState() # shared message history
client_usernames = {} # conn -> username
# ----- CLIENT ID MANAGEMENT -----
next_client_id = 0 # will increment per client
client_ids = {} # conn -> numeric ID
client_usernames = {} # conn -> username
MAX_CLIENTS = 10000 # hard user cap
def send_full_history(conn):
    """
    Sends the entire chat history to a single client.
    Each message is shown with its message ID (#n).
    """
    for msg in state.all_messages():
        display_text = f"# {msg['id']} {msg['text']}"
        payload = encode_message(msg["sender"], display_text)
        try:
            conn.sendall(payload)
        except Exception:
            break
    def handle_client(conn, addr):
        """
        Handle a single client connection.
        Parses commands and broadcasts messages with unique sender IDs.
        """
        global next_client_id
        on_client_join(addr)
        clients.append(conn)
        # ----- ASSIGN CLIENT ID -----
        client_id = next_client_id
        next_client_id += 1
        client_ids[conn] = client_id
        # Tell the client their unique ID
        conn.sendall(encode_message("SYSTEM", f"/id {client_id}"))
        # ----- SEND LIST OF CONNECTED CLIENTS (username#id) -----
        # NOTE: usernames are only known after clients send a first message
        other_labels = []
        for c, cid in client_ids.items():
            if c is conn:
                continue
            uname = client_usernames.get(c, "client") # fallback if no messages yet
            other_labels.append(f"{uname}#{cid:04d}")

```

```
if other_labels:
    msg = "Connected clients: " + ", ".join(other_labels)
else:
    msg = "You are the first client here."
try:
    conn.sendall(encode_message("SYSTEM", msg))
except Exception:
    pass
try:
    while True:
        try:
            raw = conn.recv(1024)
        if not raw:
            break
        except (ConnectionResetError, ssl.SSLError):
            break
        sender, text = decode_message(raw)
        if text is None:
            continue
        # Store username if it's the first message from this client if conn not in client_usernames:
        client_usernames[conn] = sender
        # Build command context
        ctx = CommandContext(
            conn=conn,
            sender=sender,
            state=state,
            clients=clients,
            send_full_history=send_full_history,
            broadcast_message=broadcast_message,
            client_usernames=client_usernames, # <-- this MUST match
        )
        # 1) COMMAND HANDLING
        if handle_command(ctx, text):
            continue
        # 2) NORMAL MESSAGE → store & broadcast
        stored = state.add_message(sender, text)
        display_text = f"#{{stored['id']}} {{text}}"
        # sender label WITH ID
        sender_id = client_ids[conn]
        sender_label = f'{sender}#{sender_id:04d}'
```

```
print(f"[{sender_label}] {display_text}")
payload = encode_message(sender_label, display_text)
broadcast_message(sender_label, payload, clients)
except ConnectionResetError:
    pass
finally:
    # ---- CLEANUP ----
    if conn in clients:
        clients.remove(conn)
    if conn in client_ids:
        del client_ids[conn]
    if conn in client_usernames:
        del client_usernames[conn]
    on_client_leave(addr)
    conn.close()
def start_server():
    print(f"Server running on {HOST}:{PORT} ...")
    # TLS configuration
    tls_context = create_server_context(certfile="server.crt",
                                         keyfile="server.key",
                                         )
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
        server.bind((HOST, PORT))
        server.listen()
        while True:
            conn, addr = server.accept()
            # PREVENT OVER-CAPACITY
            if len(clients) >= MAX_CLIENTS:
                conn.sendall(encode_message("SYSTEM", "Server full (10,000 users
max)."))
                conn.close()
                continue
            # TLS handshake
            try:
                tls_conn = wrap_server_connection(conn, tls_context)
            except Exception as e:
                print(f"TLS handshake failed with {addr}: {e}")
                conn.close()
                continue
            thread = threading.Thread(  
    )
```

```
target=handle_client,
args=(tls_conn, addr),
daemon=True
)
thread.start()
if __name__ == "__main__":
start_server()
client code
import socket
import threading
import sys
import os
import platform
from chatcore.protocol import encode_message, decode_message
from chatcore.tls import create_client_context, wrap_client_socket
# ----- COLOR DEFINITIONS -----
COLORS = {
"reset": "\033[0m",
"red": "\033[31m",
"green": "\033[32m",
"yellow": "\033[33m",
"blue": "\033[34m",
"magenta": "\033[35m",
"cyan": "\033[36m",
"white": "\033[37m",
}
MY_COLOR = COLORS["red"] # default self-message color
# -----
if len(sys.argv) < 2:
print("Usage: python ChatClient.py [SERVER_IP]")
sys.exit(0)
SERVER = sys.argv[1]
PORT = 5555
username = input("Enter username: ")
CHAT_HISTORY = []
def clear_terminal():
"""Clear the terminal window."""
if platform.system() == "Windows":
os.system("cls")
else:
sys.stdout.write("\033[2J\033[H") # ANSI wipe
```

```
sys.stdout.flush()
def listen(sock):
    """Background thread to receive chat messages."""
    global MY_COLOR
    while True:
        try:
            raw = sock.recv(1024)
            if not raw:
                print("Server disconnected.")
                break
            sender, text = decode_message(raw)
            if sender:
                # Color my own messages only
                color = MY_COLOR if sender == username else COLORS["reset"]
                # store message
                CHAT_HISTORY.append((sender, text))
                clear_terminal() for s, t in CHAT_HISTORY:
                    if s.startswith(username): # your own messages
                        print(f"\033[{color}[{s}] {t} {COLORS['reset']}")"
                else:
                    print(f"\033[{s}] {t}\033[0m")
                #sys.stdout.write("\r") # move cursor to start of line
                #sys.stdout.write(f"\033[{color}[{sender}] {text} {COLORS['reset']}\\n")
                sys.stdout.write(f"\033[{username}\033[0m")
                sys.stdout.flush()
            except Exception:
                break
        def start_client():
            with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
                try:
                    sock.connect((SERVER, PORT))
                except:
                    print("Could not connect to server.")
                    return
                tls_context = create_client_context(verify=False)
                try:
                    tls_sock = wrap_client_socket(
                        sock, tls_context, server_hostname=SERVER or "localhost"
                    )
                except Exception as e:
```

```
print(f"TLS handshake failed: {e}")
return
print("Connected over TLS! Type messages and press Enter.")
thread = threading.Thread(target=listen, args=(tls_sock,), daemon=True)
thread.start()
tls_sock.sendall(encode_message(username, "/register"))
# ----- SEND LOOP -----
while True:
try:
msg = input(f"[{username}] ").strip()
if not msg:
continue
# ===== LOCAL COMMAND: /exit =====
if msg == "/exit":
print("[SYSTEM] Exiting chat...")
tls_sock.close()
sys.exit(0)
# ===== LOCAL COMMAND: /color <color> =====
if msg.startswith("/color "):
global MY_COLOR
color_name = msg.split(maxsplit=1)[1].lower()
if color_name not in COLORS or color_name == "reset":
print("[SYSTEM] Valid colors: red, green, yellow, blue, magenta, cyan,
white")
continue
MY_COLOR = COLORS[color_name]
print(f"[SYSTEM] Color changed to {color_name}")
continue
# ===== REMOTE COMMAND: /refresh =====
if msg == "/refresh":
clear_terminal()
tls_sock.sendall(encode_message(username, "/refresh"))
continue
# ===== NORMAL MESSAGE =====
tls_sock.sendall(encode_message(username, msg))
except KeyboardInterrupt:
print("\n[SYSTEM] Interrupted. Closing client...")
tls_sock.close()
sys.exit(0)
if __name__ == "__main__":
start_client()
```

```
commands code
# chatcore/commands.py
# Central command handling for the chat server.
from dataclasses import dataclass
from typing import Callable, List, Any
from chatcore.protocol import encode_message
@dataclass
class CommandContext:
    conn: Any # the client's socket
    sender: str # username of the client
    state: Any # ChatState instance
    clients: List[Any] # list of all client sockets
    send_full_history: Callable # function(conn) -> None
    broadcast_message: Callable # function(sender, payload, clients)
    client_usernames: dict# encode_message already imported here
    def cmd_refresh(ctx: CommandContext, args: str) -> None:
        """
        /refresh
        Clears the client's screen (on client side) and re-sends history.
        Server side: just send full history to this client.
        """
        ctx.send_full_history(ctx.conn)
    def cmd_delete(ctx: CommandContext, args: str) -> None:
        """
        /delete <id>
        Marks one of the sender's messages as deleted.
        """
        args = args.strip()
        if not args:
            return
        ctx.conn.sendall(encode_message("SYSTEM", "Usage: /delete <id>"))
        try:
            msg_id = int(args.split()[0])
        except ValueError:
            ctx.conn.sendall(encode_message("SYSTEM", "Usage: /delete <id>"))
            return
        success = ctx.state.delete_message(msg_id, ctx.sender)
        if not success:
            ctx.conn.sendall(
                encode_message("SYSTEM", f"Cannot delete message #{msg_id}"))
```

```

)
return
# Notify all clients that deletion occurred
notice = encode_message(
    "SYSTEM", f"Message #{msg_id} deleted by {ctx.sender}"
)
ctx.broadcast_message("SYSTEM", notice, ctx.clients)
# Registry of supported commands
COMMANDS = {
    "refresh": cmd_refresh,
    "delete": cmd_delete,
}
def handle_command(ctx: CommandContext, text: str) -> bool:
    """Detect and execute a slash command.

    Returns:
        True -> the text was a command (handled or rejected)
        False -> not a command; treat as a normal chat message
    """
    if not text.startswith("/"):
        return False
    # Strip leading "/" and split into name + args
    body = text[1:].strip()
    if not body:
        ctx.conn.sendall(
            encode_message("SYSTEM", "Empty command. Try /refresh or /delete <id>.")
        )
        return True
    parts = body.split(maxsplit=1)
    name = parts[0].lower()
    args = parts[1] if len(parts) > 1 else ""
    cmd = COMMANDS.get(name)
    if cmd is None:
        ctx.conn.sendall(
            encode_message("SYSTEM", f"Unknown command: /{name}")
        )
        return True
    cmd(ctx, args)
    return True
def cmd_register(ctx: CommandContext, args: str):
    ctx.client_usernames[ctx.conn] = ctx.sender

```

```
ctx.conn.sendall(encode_message("SYSTEM", f"Username registered as {ctx.sender}"))
COMMANDS["register"] = cmd_register
def cmd_search(ctx, query):
if not query.strip():
    ctx.conn.sendall(encode_message("SYSTEM", "Usage: /search <text>"))
    return
matches = ctx.state.search(query)
if not matches:
    ctx.conn.sendall(encode_message("SYSTEM", f"No messages contain '{query}'"))
    return
ctx.conn.sendall(encode_message("SYSTEM", f"Found {len(matches)} match(es):"))
for msg in matches:
    display = f"# {msg['id']} {msg['text']}"
    payload = encode_message(msg["sender"], display)
    ctx.conn.sendall(payload)
COMMANDS["search"] = cmd_search
disconnects on message send
i pasted your code and it dont work
just give me full chatserver file
immediate disconnect
give me nwe client
same thing
give me final files
same
what is the format of clients variable in server
which variable contains usernames of connected
give me code for username tracking
never had a cleanup function
code issues?import socket
import threading
from chatcore.protocol import encode_message, decode_message
from chatcore.handlers import broadcast_message, on_client_join, on_client_leave
from chatcore.tls import create_server_context, wrap_server_connection
from chatcore.state import ChatState
from chatcore.commands import CommandContext, handle_command
HOST = "10.0.0.115" # change if needed
PORT = 5555
clients = [] # active TLS sockets
state = ChatState() # shared message history
```

```

# ----- CLIENT ID MANAGEMENT -----
next_client_id = 0 # will increment per client
client_ids = {} # conn -> numeric ID
MAX_CLIENTS = 10000 # hard user cap
client_usernames = {}
def send_full_history(conn):
    """
    Sends the entire chat history to a single client.
    Each message is shown with its message ID (#n).
    """
    for msg in state.all_messages():
        display_text = f"# {msg['id']} {msg['text']}" "payload = encode_message(msg['sender'],
        display_text)
        try:
            conn.sendall(payload)
        except Exception:
            break
    def handle_client(conn, addr):
        """
        Handle a single client connection.
        Parses commands and broadcasts messages with unique sender IDs.
        """
        global next_client_id
        on_client_join(addr)
        clients.append(conn)
        # ----- ASSIGN CLIENT ID -----
        client_id = next_client_id
        next_client_id += 1
        client_ids[conn] = client_id
        # Tell the client their unique ID
        for _, username in client_usernames:
            conn.sendall(encode_message("SYSTEM", f"Connected Users: {username}"))
        conn.sendall(encode_message("SYSTEM", f"/id {client_id}"))
        try:
            while True:
                raw = conn.recv(1024)
                if not raw:
                    break
                sender, text = decode_message(raw)
                if text is None:

```

```

continue
# Store username if first message from this client
if conn not in client_usernames:
    client_usernames[conn] = sender
# Build command context
ctx = CommandContext(
    conn=conn,
    sender=sender,
    state=state,
    clients=clients,
    send_full_history=send_full_history,broadcast_message=broadcast_message,
)
# 1) Run commands like /refresh, /delete
if handle_command(ctx, text):
    continue # command handled, skip normal broadcast
# 2) NORMAL MESSAGE → store and broadcast
uname = client_usernames.get(conn, sender)
stored = state.add_message(uname, text)
display_text = f"#{{stored['id']}} {{text}}"
# sender label WITH ID
sender_id = client_ids[conn]
sender_label = f'{uname}#{sender_id:04d}' # zero-padded 5 digits
print(f'[ {sender_label}] {display_text}')
payload = encode_message(sender_label, display_text)
broadcast_message(sender_label, payload, clients)
except ConnectionResetError:
    # client killed connection
    pass
finally:
    # cleanup
    if conn in clients:
        clients.remove(conn)
    if conn in client_ids:
        del client_ids[conn]
    on_client_leave(addr)
    conn.close()
def start_server():
    print(f"Server running on {HOST}:{PORT} ...")
    # TLS configuration
    tls_context = create_server_context(

```

```
certfile="server.crt",
keyfile="server.key",
)
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
    server.bind((HOST, PORT))
    server.listen()
    while True:
        conn, addr = server.accept()
        # PREVENT OVER-CAPACITY
        if len(clients) >= MAX_CLIENTS: conn.sendall(encode_message("SYSTEM", "Server full (10,000 users max)."))
        conn.close()
        continue
    # TLS handshake
    try:
        tls_conn = wrap_server_connection(conn, tls_context)
    except Exception as e:
        print(f"TLS handshake failed with {addr}: {e}")
        conn.close()
        continue
    thread = threading.Thread(
        target=handle_client,
        args=(tls_conn, addr),
        daemon=True
    )
    thread.start()
if __name__ == "__main__":
    start_server()
is clients a list of conn tuples
I fear our /refresh is causing duplicates in chat)history in client
wouldnt that be two separate messages sent to client and therefore two separate listens?
```