LAPORAN TUGAS SOCKET PROGRAMMING
II2120 JARINGAN KOMPUTER

Dosen Pengampu:

Ir. I Gusti Bagus Baskara Nugraha, S.T., M.T., Ph.D.

Oleh:

MochiLabtekV (K-02)

Fathimah Nurhumaida Ramadhani	18223052
Nurul Na'im Natifah	18223106

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

I. PENDAHULUAN

Socket Programming

Socket programming adalah teknik pemrograman yang memungkinkan dua node berkomunikasi satu sama lain. Melalui socket programming, kita dapat membangun aplikasi jaringan seperti server-client yang saling bertukar data. Dalam socket programming, ada dua tipe socket yang umum dikenal:

1. UDP (User Datagram Protocol) adalah protokol pada layer transport yang berjalan di atas Internet Protocol (IP). UDP digunakan untuk komunikasi yang membutuhkan kecepatan dan toleransi terhadap kehilangan data, tanpa jaminan urutan atau keandalan data. Hal ini disebabkan oleh ketiadaan mekanisme koneksi pada UDP, sehingga data dikirimkan tanpa persetujuan penerima dan tanpa kepastian bahwa data akan sampai.
2. TCP (Transmission Control Protocol) adalah protokol pada layer transport yang juga berjalan di atas IP. Berbeda dengan UDP, TCP menjamin keandalan dan urutan data melalui mekanisme koneksi dan pengakuan data. TCP memastikan semua data sampai dengan benar dan lengkap, serta mengirim ulang data yang hilang jika diperlukan.

Pada tugas ini, kami diminta membuat aplikasi chat room sederhana menggunakan socket programming. Aplikasi ini diprogram dalam bahasa Python dan memanfaatkan protokol transport UDP. Meskipun begitu, kami mencoba mengimplementasikan konsep TCP meskipun tetap menggunakan UDP.

Aplikasi ini dirancang agar server mampu menerima pesan dari client dan meneruskannya ke client lain. Untuk melakukannya, pengguna perlu memasukkan IP Address dan port yang akan digunakan terlebih dahulu. Seperti pada aplikasi chat room umumnya, setiap client memiliki username unik. Aplikasi ini juga dilengkapi tampilan GUI.

Cara Kerja Program

Program pertama kali dijalankan melalui terminal. Langkah pertama adalah membuka server terlebih dahulu sebelum menjalankan client. Ketika program server dijalankan,

pengguna akan diminta memasukkan IP Address dan port. Sebelumnya, perangkat telah dipastikan terhubung ke jaringan internet yang sama. Oleh karena itu, IP Address yang digunakan adalah IP Address perangkat server, dan port dapat dipilih sesuai keinginan pengguna. Sebelum server dibuka, pengguna juga diminta memasukkan password untuk chatroom.

Setelah server berjalan, program client dapat dijalankan. Client akan memasukkan IP Address dan port server, kemudian memilih port client. Setelahnya, client diarahkan untuk login dengan memasukkan username dan password. Kemudian, client diminta memasukkan password chatroom; client baru dapat bergabung ke server setelah memasukkan password yang benar. Server akan menyambut client dengan pesan "{username} has joined the chat."

Aplikasi ini memerlukan minimal 2 client untuk berfungsi, sehingga diperlukan akun client lain. Setelah semua client siap, pengguna dapat saling bertukar pesan melalui aplikasi ini.

Alur Perjalanan Pesan:
1. Client mengirim pesan
2. Server menerima pesan dan mengirimkan ACK sebagai konfirmasi
3. Client menerima pesan ACK (tidak ditampilkan)
4. Server meneruskan pesan ke client lain
5. Client lain menerima pesan

Penggunaan ACK ini berdasarkan konsep TCP, di mana server mengirimkan konfirmasi penerimaan pesan. Pada dasarnya, UDP tidak memiliki jaminan pengiriman pesan karena tidak ada konfirmasi.

II. SPESIFIKASI PROGRAM

Tabel 1: Spesifikasi Wajib

No	Spesifikasi	Nilai	Selesai
1	Server mampu menerima pesan yang dikirim client dan mencetaknya ke layar.	15	☑
2	Server mampu meneruskan pesan satu client ke client lain.	15	☑
3	Client mampu mengirimkan pesan ke server dengan IP dan port yang ditentukan pengguna.	15	☑
4	Client mampu menerima pesan dari client lain (yang diteruskan oleh server), dan mencetaknya ke layar.	15	☑
5	Client harus memasukkan *password* untuk dapat bergabung ke chatroom.	10	☑
6	Client memiliki username yang unik.	10	☑

Tabel 2: Spesifikasi Tambahan

No	Spesifikasi	Nilai	Selesai
1	Aplikasi mengimplementasikan TCP over UDP.	15	☑
2	Seluruh pesan dienkripsi menggunakan algoritma kriptografi klasik simetris, misal cipher Vigenere atau Caesar.	5	☐
3	Seluruh pesan dienkripsi menggunakan algoritma kriptografi modern simetris, misal cipher RC4.	10	☐
4	Seluruh pesan dienkripsi menggunakan algoritma kriptografi modern asimetris, misal cipher RSA, atau kombinasi algoritma kriptografi modern asimetris dan modern simetris.	15	☐
5	Seluruh pesan dienkripsi menggunakan algoritma Double-Rachet atau MLS.	20	☐

6	Aplikasi memiliki GUI.	5	☑
7	Aplikasi mampu digunakan untuk mengirimkan dan menerima pesan bertipe file biner.	5	☑
8	Aplikasi mampu menunjukkan apabila integritas pesan telah rusak, baik dengan memanfaatkan *checksum* ataupun *digital signature*.	10	☐
9	Aplikasi mampu menyimpan pesan-pesan lampau meskipun telah ditutup; mekanisme dan tempat penyimpanan bebas, baik di *client* maupun di *server*.	10	☐
10	Aplikasi mampu mengotentikasi pengguna.	5	☑
11	Aplikasi diprogram menggunakan paradigma *object oriented programming* atau pemrograman berorientasi objek	5	☐

III. PENGUJIAN PROGRAM

Lingkungan pengujian program menggunakan sistem operasi Windows. Pada pengujian ini, device yang berperan sebagai Server dan device yang berperan sebagai Client diuji pada jaringan IP Address yang sama dan dihubungkan ke hotspot yang sama. Bahasa pemrograman yang digunakan adalah Python, dan IDE yang digunakan adalah Terminal/Command Prompt, lalu dilanjutkan dengan GUI.

Tata Cara Pengujian Program

1. Kedua device dipastikan sudah terhubung pada hotspot yang sama.
2. Source code dan data-data yang diperlukan dapat diakses via repository GitHub: https://github.com/MochiLabtekV/14_Tugas-Socket-Programming
3. “Pull” dilakukan untuk menduplikasi repository ke komputer lokal.
4. Pada device server, file “server.py” dijalankan di Terminal.
5. IP Address device yang digunakan server, port server, dan set password chatroom dimasukkan sesuai perintah di Terminal.
6. GUI server berhasil terbuka.
7. Pada device client, file “client.py” dijalankan di Terminal.

8. IP Address server, port server, dan port client dimasukkan di Terminal.
9. Client diberi opsi register dan login, dapat memilih register jika ingin membuat akun baru dan login jika ingin menggunakan akun yang sudah ada.
10. Autentikasi client dengan memasukkan username dan password.
11. Autentikasi chatroom dengan memasukkan password chatroom.
12. GUI client berhasil terbuka.
13. GUI server akan menampilkan notifikasi bergabungnya client dan mencatat identitas client yang bergabung di “List of Connected Clients.”
14. Client mencoba untuk mengirim pesan. Pesan akan ditampilkan di GUI client yang lain dan GUI server.
15. Server akan menampilkan ACK di GUI server sebagai implementasi TCP over UDP dan antisipasi packet loss.

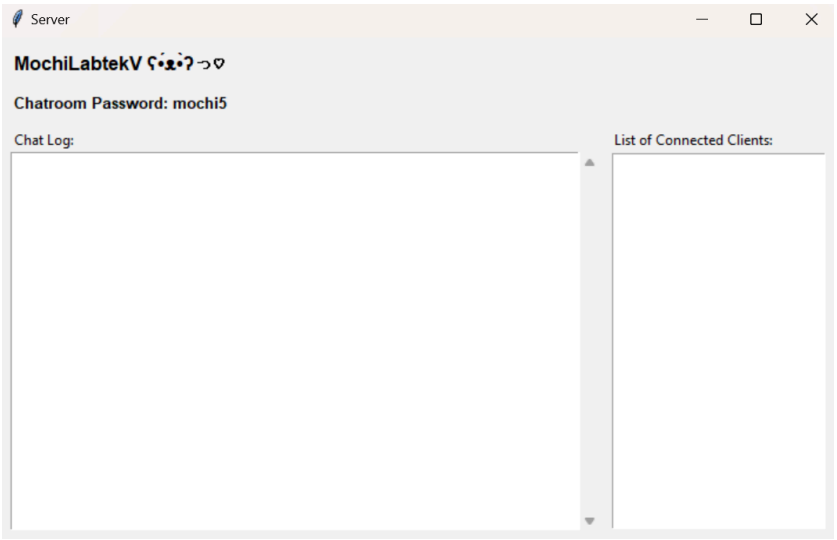
Tangkapan Layar Pengujian Program

1. Jalankan program server pada perangkat yang dijadikan sebagai server.

```
PS C:\Users\USER\Documents\Jarkom\14_Tugas-Socket-Programming\src> py server.py
Enter Server IP: 192.168.12.160
Enter Server port: 55555
Set Chatroom password: mochi5
```

Gambar 1: Tampilan Terminal server

2. Tampilan GUI awal server (spesifikasi tambahan nomor 6).



Gambar 2: Tampilan GUI awal server

3. Terminal Client 1 (spesifikasi wajib nomor 3).

```
PS C:\Users\Zek\Python\14_Tugas-Socket-Programming\src> py client.py
Insert Server IP Address: 192.168.12.160
Insert Server Port Number: 55555
Insert Client Port Number: 12345
```

Gambar 3: Tampilan terminal Client (1)

4. Berhasil menggunakan fitur Register

```
Welcome! Please choose an option:
1. Register
2. Login
Enter your choice: 1
Insert username: nafakeren
Insert password: 182
Registration successful! Welcome, nafakeren!
```

Gambar 4: Tampilan Terminal saat client sukses registrasi

5. Gagal menggunakan fitur Register akibat username yang sudah ada (spesifikasi wajib nomor 6).

```
Welcome! Please choose an option:
1. Register
2. Login
Enter your choice: 1
Insert username: aku123
Insert password: 123
Username already taken. Please try again.
```

Gambar 5: Tampilan Terminal saat client gagal registrasi

6. Client salah saat memasukkan password chatroom.

```
Enter chatroom password: mochi2
Wrong password!
```

Gambar 6: Tampilan Terminal saat password chatroom salah

7. Client benar saat memasukkan *password* untuk dapat bergabung ke chatroom (spesifikasi wajib nomor 5).

```
Enter chatroom password: mochi5
Password is correct. Welcome to the chatroom!
```

Gambar 7: Tampilan Terminal saat password chatroom benar

8. Terminal Client 2 dengan port berbeda (spesifikasi wajib nomor 3)

```
PS C:\Users\Zek\Python\14_Tugas-Socket-Programming\src> py client.py
Insert Server IP Address: 192.168.12.160
Insert Server Port Number: 55555
Insert Client Port Number: 54321
```

Gambar 8: Tampilan Terminal client (2)

9. Gagal menggunakan fitur Login karena salah memasukkan password akun (spesifikasi tambahan nomor 10).

```
Insert username: fathykeren
Insert password: wow
Invalid username or password. Please try again.
```

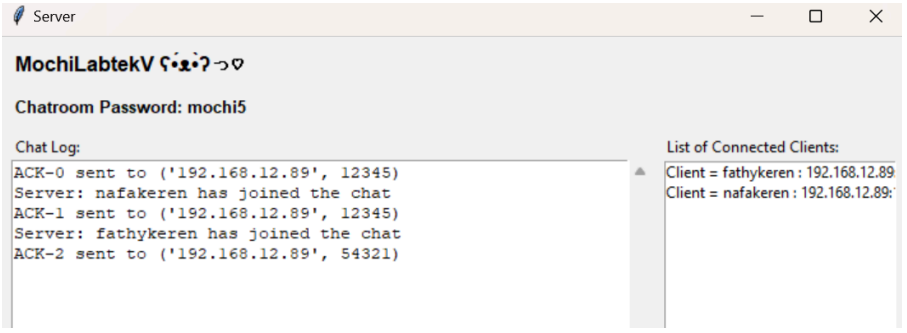
Gambar 9: Tampilan Terminal saat password client salah

10. Berhasil masuk ke chatroom setelah input username, user password, dan chatroom password benar (spesifikasi tambahan nomor 10).

```
Welcome! Please choose an option:
1. Register
2. Login
Enter your choice: 2
Insert username: fathykeren
Insert password: keren
Login successful! Welcome back, fathykeren!
Enter chatroom password: mochi5
Password is correct. Welcome to the chatroom!
```

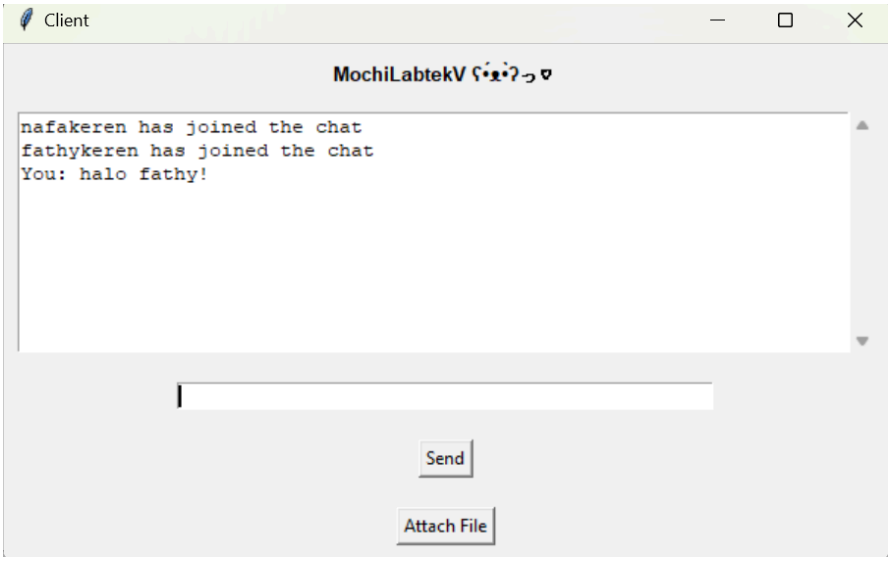
Gambar 10: Tampilan Terminal saat password client benar

11. GUI: tampilan server saat client bergabung ke chatroom.



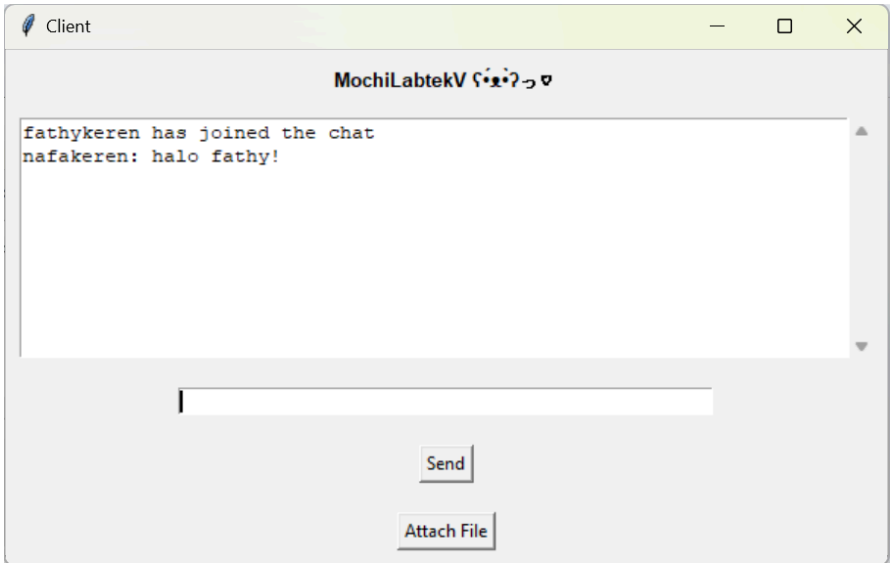
Gambar 11: Tampilan GUI server saat client sudah terhubung

12. Tampilan GUI Client 1 saat mengirimkan pesan ke server (spesifikasi wajib nomor 3).



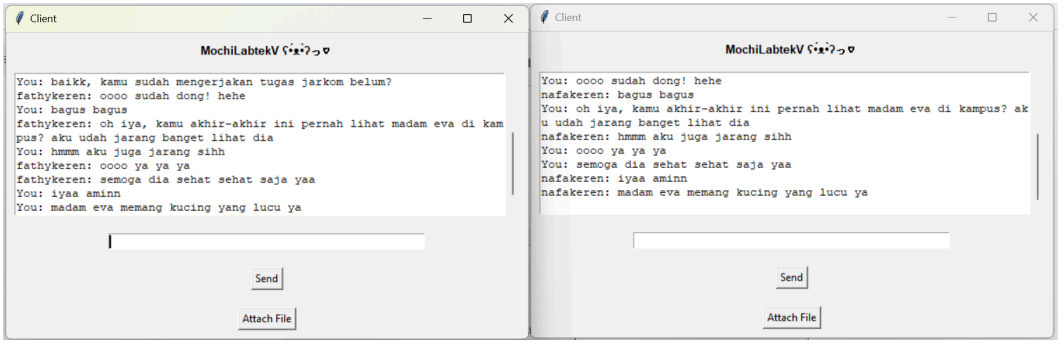
Gambar 12: Tampilan GUI client pengirim

13. Tampilan GUI Client 2 saat menerima pesan dari Client 1 (spesifikasi wajib nomor 4)



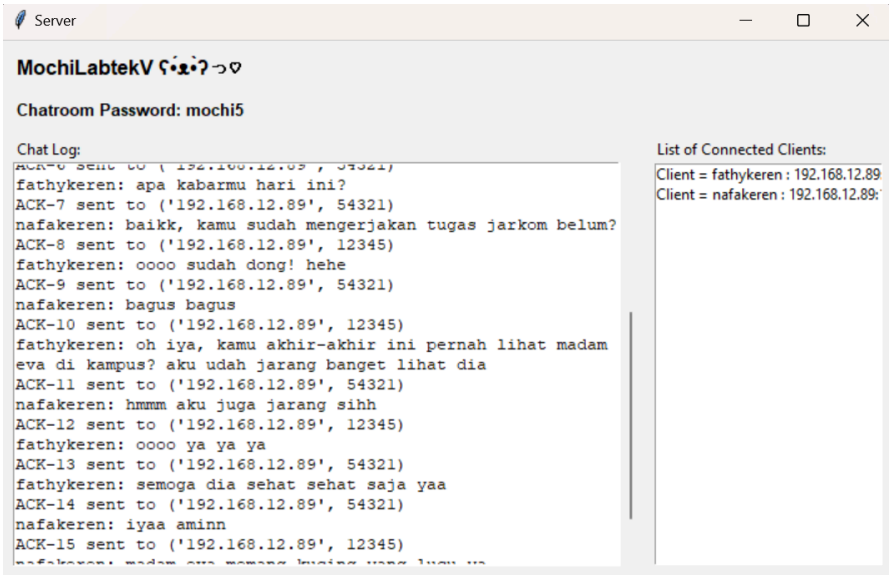
Gambar 13: Tampilan GUI client penerima

14. Perbandingan tampilan GUI Client 1 dan Client 2, tidak ada pesan yang hilang (spesifikasi wajib nomor 2, spesifikasi tambahan nomor 6).



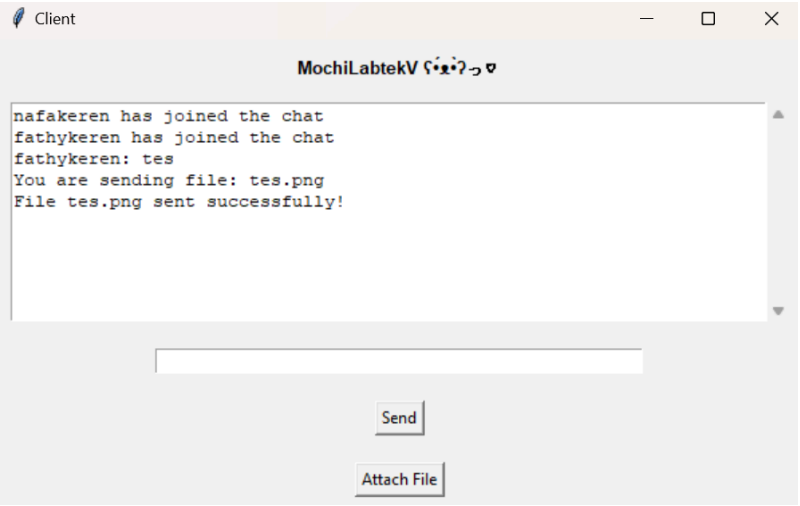
Gambar 14: Tampilan GUI client side-by-side

15. Tampilan server saat Client 1 dan Client 2 bertukar pesan (spesifikasi wajib nomor 1 & 2, spesifikasi tambahan nomor 1)



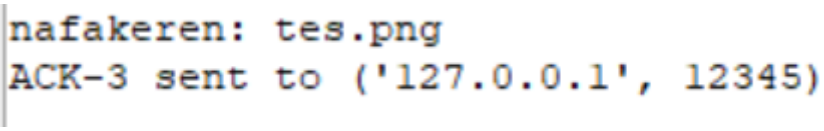
Gambar 15: Tampilan GUI server selama client terhubung

16. Tampilan client setelah mengirim binary file



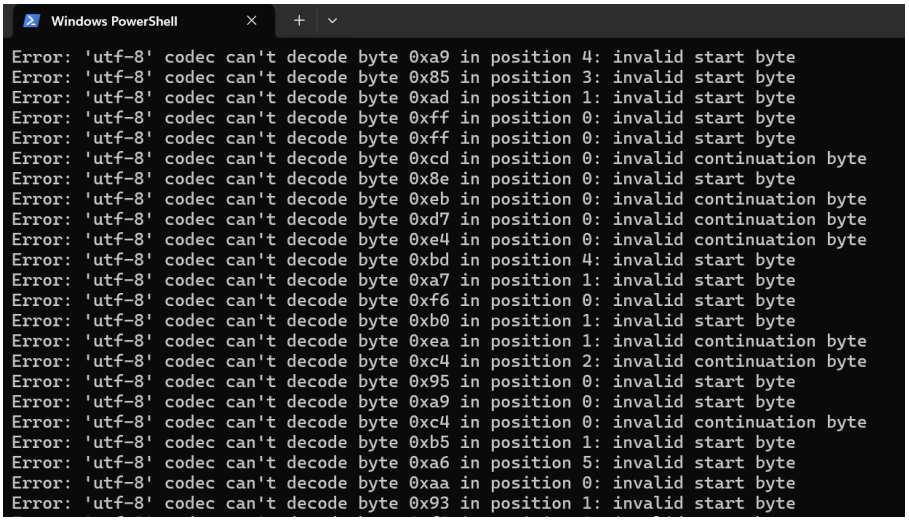
Gambar 16: Tampilan GUI client setelah mengirim file biner

17. Tampilan server saat client mengirim binary file



Gambar 17: Tampilan GUI server setelah client mengirim file biner

18. Error dikarenakan file encoding belum tepat untuk pengiriman binary file



Gambar 18: Error pada Terminal server setelah pengiriman file biner

IV. KODE PROGRAM

Program-program utama

Source code server.py

```
import socket
import threading
from tkinter import *
from tkinter import scrolledtext, messagebox
from utilities import is_valid_ip
import os

# Set up UDP server for receiving messages
ipAddress = input("Enter Server IP: ")
while not is_valid_ip(ipAddress):
    print("Invalid IP address. Please enter a valid IPv4 address.")
    ipAddress = input("Enter Server IP: ")

portServer = int(input("Enter Server port: "))
chatroom_password = input("Set Chatroom password: ")

udp_server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
udp_server.bind((ipAddress, portServer))

clients = set()
client_usernames = {}

def update_client_list():
    list_clients.delete(0, END)
    for client in clients:
         list_clients.insert(END, f"Client = {client_usernames[client]}
: {client[0]}:{client[1]}")  # Displaying username

def start_server():
    while True:
        try:
            message, client_address = udp_server.recvfrom(1024)
            message = message.decode('utf-8')

            # Check if it's an authentication message
            if message.startswith("AUTH |"):
                _, username, password = message.split("|", 2)
```

```
                    if password.strip() == chatroom_password:  # Check
password
                    clients.add(client_address)
                                    client_usernames[client_address]  =
username.strip()
                      udp_server.sendto("AUTH_SUCCESS".encode('utf-8'),
client_address)
                        notify_clients(client_address, "has joined the
chat")
                    update_client_list()  # Update client list after a
new client joins
                else:
                       udp_server.sendto("AUTH_FAILED".encode('utf-8'),
client_address)
            else:
                        # If the message is an ACK, skip logging and
incrementing ACK
                if message.startswith("ACK"):
                    continue  # Skip processing this message
                else:
                    # Handle chat messages and log them
                                forward_message(message.encode('utf-8'),
client_address)

            # Send ACK after processing the message
                send_ack(client_address)  # Send ACK only once after
processing the message
        except Exception as e:
            print(f"Error: {e}")  # Debug output for any exceptions

# Handle file transfer
def handle_file_transfer(message, sender_address):
    _, filename = message.decode('utf-8').split('|', 1)
    filename = filename.strip()

    try:
        with open(filename, 'wb') as f:
            while True:
                data, addr = udp_server.recvfrom(1024)
                  if not data:  # Stop sending if an empty message is
received
```

```
                    break
                f.write(data)

            window.after(100, lambda: chat_log.insert(END, f"File
{filename} received from {client_usernames[sender_address]}.\n"))
                        print(f"File {filename} received from
{client_usernames[sender_address]}.")  # Debugging line

        # Confirmation dialog to open a file
        window.after(100, lambda: confirm_open_file(filename))

        # Send the file to all other clients
        with open(filename, 'rb') as f:
            file_data = f.read(1024)
            while file_data:
                for client in clients:
                    if client != sender_address:
                        udp_server.sendto(file_data, client)
                file_data = f.read(1024)
            send_ack(sender_address)  # Only send ACK once after file
transfer
    except Exception as e:
        print(f"Error receiving file: {e}")  # Debugging line
            window.after(100, lambda: chat_log.insert(END, f"Error
receiving file: {e}\n"))

# Confirm to open the received file
def confirm_open_file(filename):
    if messagebox.askyesno("Open File", f"File {filename} received. Do
you want to open it?"):
        try:
            os.startfile(filename)  # Windows only
        except Exception as e:
            messagebox.showerror("Error", f"Could not open file: {e}")

# Notify all clients when a new client connects
def notify_clients(client_address, message):
    connect_message = f"{client_usernames[client_address]} {message}"
    for client in clients:
        udp_server.sendto(connect_message.encode('utf-8'), client)
```

```
            window.after(100,   lambda:   chat_log.insert(END,   f"Server:
{connect_message}\n"))

# Forward message to other clients
def forward_message(message, sender_address):
    # Decode the message for logging
    decoded_message = message.decode('utf-8')

    # Check if the message is an ACK
    if decoded_message.startswith("ACK"):
        return  # Do not log or process ACK messages

    # Extract the actual message part
     actual_message = decoded_message.split(":", 1)[-1].strip()  # Get
only the message part
    username = client_usernames[sender_address]  # Get the username of
the sender

    # Log the actual message content with username
        window.after(100,  lambda:  chat_log.insert(END,  f"{username}:
{actual_message}\n"))

    # Forward the message to other clients
    for client in clients:
        if client != sender_address:
                                        udp_server.sendto(f"{username}:
{actual_message}".encode('utf-8'), client)  # Send with username

# Send ACK using UDP
sequence_number = 0

def send_ack(client_address):
    global sequence_number
    try:
        # Prepare the ACK message with sequence number
        ack_message = f"ACK-{sequence_number}"

        # Send ACK message to the client
        udp_server.sendto(ack_message.encode('utf-8'), client_address)

        # Print ACK log with sequence number
```

```
            window.after(100, lambda: chat_log.insert(END, f"{ack_message}
sent to {client_address}\n"))

            # Increment the sequence number for the next ACK
            sequence_number += 1
        except Exception as e:
            print(f"Error sending ACK: {e}")

# Function to initialize the GUI
def initialize_gui():
    global window, chat_log, list_clients

    window = Tk()
    window.title("Server")

    main_frame = Frame(window)
    main_frame.grid(row=0, column=0, padx=10, pady=10, sticky="ew")

     chatroom_label = Label(main_frame, text="MochiLabtekV ʕ·´ᴥ`?っ♡ ",
font=("Arial", 12, "bold"))
     chatroom_label.grid(row=0, column=0, columnspan=2, pady=(0, 10),
sticky="w")

       password_label = Label(main_frame, text=f"Chatroom Password:
{chatroom_password}", font=("Arial", 10, "bold"))
     password_label.grid(row=1, column=0, columnspan=2, pady=(0, 10),
sticky="w")

    Label(main_frame, text="Chat Log:").grid(row=2, column=0, padx=(0,
10), sticky="w")
         chat_log = scrolledtext.ScrolledText(main_frame, width=60,
height=20)
    chat_log.grid(row=3, column=0, padx=(0, 10))

     Label(main_frame, text="List of Connected Clients:").grid(row=2,
column=1, sticky="w")
    list_clients = Listbox(main_frame, width=30, height=20)
    list_clients.grid(row=3, column=1)

    thread = threading.Thread(target=start_server)
    thread.daemon = True
```

```
    thread.start()
    window.mainloop()

initialize_gui()
```

Source code client.py

```
import socket
import threading
from tkinter import *
from tkinter import scrolledtext, filedialog, messagebox
from utilities import validate_input, is_valid_ip
from reglog import register_client, login_client
import os

client = None
server_address = None
username = None

def command_prompt():
    global username  # Ensure username is global
    print("Welcome! Please choose an option:")
    print("1. Register")
    print("2. Login")
    choice = input("Enter your choice: ")

    if choice == "1":
        username = register_client()
        if username:
            authenticate()
    elif choice == "2":
        username = login_client()
        if username:
            authenticate()
    else:
        print("Invalid choice. Please enter 1 or 2.")

def authenticate():
    global username
    client_input_password = input("Enter chatroom password: ")
```

```
                    client.sendto(f"AUTH | {username} |
{client_input_password}".encode(), server_address)

    response, _ = client.recvfrom(1024)
    response = response.decode()

    if response == "AUTH_SUCCESS":
        print("Password is correct. Welcome to the chatroom!")
                initialize_gui()    # Initialize GUI on successful
authentication
    else:
        print("Wrong password!")
        exit()

def receive_message():
    global client
    while True:
        try:
            message, _ = client.recvfrom(1024)  # UDP recv
            message = message.decode('utf-8')
             if not message.startswith("ACK-"):  # Only process non-ACK
messages
                chat_log.insert(END, f"{message}\n")
                if message.startswith("FILE |"):
                    handle_received_file(message)
                send_ack()  # Send ACK only for non-ACK messages
        except Exception as e:
            print(f"Error receiving message: {e}")
            break

def send_message():
    global client, username
    message = entry_message.get()
    if message and client:
        formatted_message = f"{username}: {message}"
        chat_log.insert(END, f"You: {message}\n")
                    client.sendto(formatted_message.encode('utf-8'),
server_address)  # UDP send
        entry_message.delete(0, END)
        send_ack()  # Send ACK via UDP
```

```
def send_file(filepath):
    global client, username
    if filepath:
        filename = os.path.basename(filepath)
        chat_log.insert(END, f"You are sending file: {filename}\n")

        # Send file information to the server
         client.sendto(f"FILE | {username}:{filename}".encode('utf-8'),
server_address)

        # Send binary file data to the server
        with open(filepath, 'rb') as file:
            data = file.read(1024)
            while data:
                 client.sendto(data, server_address)  # Send file data
to the server
                data = file.read(1024)

        # Send a signal that the file has been fully sent
        client.sendto(b'', server_address)  # Send an empty message as
a signal that file sending is complete
        chat_log.insert(END, f"File {filename} sent successfully!\n")
        send_ack()  # Send ACK after file sent

def attach_file():
    filepath = filedialog.askopenfilename()  # Dialog to choose a file
    if filepath:
        send_file(filepath)  # Send the selected file to the server

def handle_received_file(message):
    # Parse the message to get the sender's username and file name
    _, sender_username, filename = message.split(':', 2)  # Split the
message to get sender information and file name

    # Save the received file
    with open(filename, 'wb') as f:
        while True:
            data, _ = client.recvfrom(1024)  # Receiving file data
from the server
            if not data:  # Check for empty data to break the loop
                break
```

```
            f.write(data)

    # Update the chat log that the file has been received
            chat_log.insert(END,   f"File   {filename}   received   from
{sender_username}.\n")

    # Display a message box to open a file
    confirm_open_file(filename)

def confirm_open_file(filename):
    if messagebox.askyesno("Open File", f"File {filename} received. Do
you want to open it?"):
        if os.path.isfile(filename):
            try:
                os.startfile(filename)  # Opening file (Windows)
            except Exception as e:
                  messagebox.showerror("Error", f"Could not open file:
{e}")
        else:
              messagebox.showerror("Error", f"File {filename} does not
exist.")

def send_ack():
    ack_message = "ACK"
    client.sendto(ack_message.encode('utf-8'), server_address)  # Send
ACK to the server

def initialize_gui():
    global window, chat_log, entry_message

    window = Tk()
    window.title("Client")

        chat_label  =  Label(window,  text="MochiLabtekV  ʕ·´ᴥ·`ʔっ♡  ",
font=("Arial", 10, "bold"))
    chat_label.grid(column=0, row=0, padx=10, pady=(10, 5))

    chat_log = scrolledtext.ScrolledText(window, width=70, height=10)
    chat_log.grid(column=0, row=1, padx=10, pady=10)

    entry_message = Entry(window, width=60)
```

```
    entry_message.grid(column=0, row=2, padx=10, pady=10)

    send_button = Button(window, text="Send", command=send_message)
    send_button.grid(column=0, row=3, padx=10, pady=10)

          attach_file_button   =   Button(window,   text="Attach   File",
command=attach_file)
    attach_file_button.grid(column=0, row=4, padx=10, pady=10)

    entry_message.bind("<Return>", lambda event: send_message())

    threading.Thread(target=receive_message, daemon=True).start()
    window.mainloop()

def setup_client():
    global client, server_address, IpAddress, portServer
    IpAddress = input("Insert Server IP Address: ")
    while not is_valid_ip(IpAddress):
              print("Invalid  IP  address.  Please  enter  a  valid  IPv4
address.")
        IpAddress = input("Insert Server IP Address: ")

    portServer = int(input("Insert Server Port Number: "))
    clientPort = int(input("Insert Client Port Number: "))

    # Create UDP socket
    client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    client.bind(('', clientPort))  # Bind to the input clientPort
    server_address = (IpAddress, portServer)

    command_prompt()

setup_client()
```

Program tambahan (refactoring)

Source code utilities.py

```python
def is_valid_ip(ip):
    parts = ip.split('.')
    if len(parts) != 4:
        return False
    for part in parts:
        if not part.isdigit() or not 0 <= int(part) <= 255:
            return False
    return True

def isallnumber(string):
    angka = ['0','1','2','3','4','5','6','7','8','9']
    for i in string:
        if i not in angka:
            return False
    return True

def validate_input(user_input):
    # Check if input is not all digits
    if isallnumber(user_input):
        return False

    # Check for allowed characters (alphanumeric, _, -)
    allowed_characters = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_-"

    for char in user_input:
        if char not in allowed_characters:
            return False

    return True
```

Source code reglog.py

```python
from utilities import validate_input
import csv

csv_file = 'client_data.csv'

# Helper functions for CSV handling
def read_csv():
    with open(csv_file, mode='r') as file:
        reader = csv.DictReader(file)
        return [row for row in reader]

def write_csv(id, username, password):
    with open(csv_file, mode='a', newline='') as file:
        writer = csv.writer(file)
        writer.writerow([id, username, password])

# Register client with unique username
def register_client():
    user_data = read_csv()
    username = input("Insert username: ")
    password = input("Insert password: ")

    if any(user['username'] == username for user in user_data):
        print("Username already taken. Please try again.")
        return None
    else:
        user_id = len(user_data) + 1
        write_csv(user_id, username, password)
        print(f"Registration successful! Welcome, {username}!")
        return username

# Login client with username and password check
def login_client():
    user_data = read_csv()
    username = input("Insert username: ")
    password = input("Insert password: ")

    for user in user_data:
        if user['username'] == username and user['password'] == password:
            print(f"Login successful! Welcome back, {username}!")
            return username
    print("Invalid username or password. Please try again.")
    return None

```

V. PEMBAGIAN TUGAS

Tabel 3: Kontribusi Masing-Masing Anggota

Nama dan NIM Anggota	Rincian Kontribusi
Fathimah Nurhumaida Ramadhani (18223052)	• Membuat kode program spesifikasi wajib nomor 1, 2, 3, 4, 5, 6 • Membuat kode program spesifikasi tambahan nomor 6 & 10 • Menyusun laporan
Nurul Na'im Natifah (18223106)	• Membuat kode program spesifikasi wajib nomor 1, 2, 3, 4 • Membuat kode program spesifikasi tambahan nomor 1 & 7 • Menyusun laporan

VI. TAUTAN

Tautan repository GitHub:
https://github.com/MochiLabtekV/14_Tugas-Socket-Programming.git

Tautan video demonstrasi (YouTube):
https://www.youtube.com/watch?v=OPPJgZwZw6s

DAFTAR PUSTAKA

Cisco Networking Academy. (n.d.). Packet Tracer. Diakses dari https://www.netacad.com/portal/resources/packet-tracer

Kurose, J. F., & Ross, K. W. (2021). Computer Networking: A Top-Down Approach (8th ed.). Pearson Education.