LAPORAN PROJEK AKHIR SEMESTER
Praktikum Socket Programming

“Fitur Chat Client-Server”

Disusun Oleh:
Elnathan Natanael [2256301014]
Sabam Lumban Gaol [2256301029]
Simon Sibarani [2256301031]
Syaira Aulia Putri [2256301034]

Instruktur:
Silvana Rasio Henim, S.ST, M.T.
Thesa Nabila Balqis, S.Tr.T.

Politeknik Caltex Riau
Jurusan Teknologi Informasi
Program Studi Teknologi Rekayasa Komputer

2024/2025

Daftar Isi

BAB 1
Pendahuluan

1.1. Latar Belakang

Aplikasi chat telah menjadi salah satu alat komunikasi utama dalam kehidupan sehari-hari, dengan berbagai keunggulan dan kekurangan dari setiap platform. Misalnya, WhatsApp menyediakan fitur lengkap seperti pengiriman file, panggilan suara, dan video call, namun memiliki kelemahan dalam efisiensi penggunaan bandwidth. Sebaliknya, aplikasi chat seperti Facebook Messenger menawarkan kesederhanaan namun kurang dalam fitur pengiriman file. Berdasarkan kelebihan dan kekurangan tersebut, diperlukan sebuah solusi inovatif berupa aplikasi chat baru yang menggabungkan keunggulan kedua platform tersebut.

Pada project ini, kami berencana untuk mengembangkan aplikasi chat berbasis client-server yang memanfaatkan Socket Programming dengan GUI yang ringan, efisien dalam penggunaan bandwidth, dan memiliki antarmuka sederhana. Aplikasi ini juga akan menghadirkan fitur tambahan seperti menampilkan foto dalam sesi chat untuk meningkatkan personalisasi.

Pengembangan aplikasi akan berpedoman pada beberapa kriteria evaluasi utama. Di tahap awal, aplikasi diharapkan berjalan tanpa error, mendukung pengiriman pesan dari client ke server, serta memungkinkan server merespons pesan tersebut. Fitur lanjutan akan mencakup kemampuan server menangani permintaan dari beberapa client secara bersamaan menggunakan teknik seperti multithreading. Selain itu, aplikasi akan mendukung pengiriman pesan multimedia, broadcast ke seluruh client, dan mencatat waktu pengiriman serta penerimaan pesan. Sebagai puncaknya, aplikasi akan menyimpan pesan pada server untuk keperluan pengarsipan, meningkatkan fleksibilitas komunikasi antar client.

Dengan pendekatan ini, aplikasi yang dihasilkan tidak hanya memenuhi kebutuhan dasar komunikasi, tetapi juga menawarkan fitur yang lebih fleksibel, ringan, dan inovatif dibandingkan aplikasi yang ada saat ini.

1.2. Rumusan Masalah

1) Bagaimana merancang dan mengembangkan aplikasi chat berbasis client-server dengan memanfaatkan Socket Programming yang efisien dalam penggunaan bandwidth dan memiliki antarmuka GUI yang sederhana?
2) Bagaimana mengimplementasikan fitur utama seperti pengiriman pesan teks, pesan multimedia, broadcast pesan, serta pencatatan waktu pengiriman dan penerimaan pesan?

3) Bagaimana memastikan aplikasi dapat menangani komunikasi dari beberapa client secara bersamaan menggunakan teknik seperti multithreading?
4) Bagaimana membuat server mampu menyimpan pesan sebagai arsip, sehingga dapat digunakan kembali untuk kebutuhan tertentu?
5) Bagaimana menambahkan elemen personalisasi seperti menampilkan foto dalam sesi chat untuk meningkatkan pengalaman pengguna?

1.3. Tujuan

Tujuan dari pengembangan aplikasi ini adalah untuk menciptakan sebuah aplikasi chat berbasis client-server yang efisien dalam penggunaan bandwidth dan memiliki antarmuka sederhana. Aplikasi ini dirancang untuk mendukung fitur-fitur utama seperti pengiriman pesan teks, multimedia, broadcast pesan, dan pencatatan waktu pengiriman serta penerimaan pesan. Selain itu, aplikasi ini akan mampu menangani komunikasi dari banyak client secara bersamaan menggunakan teknik multithreading, menyediakan fitur penyimpanan pesan di server sebagai arsip, serta menampilkan foto untuk meningkatkan pengalaman pengguna.

Bab 2
Landasan Teori

2.1. Socket Programming

Socket programming adalah metode pemrograman yang memanfaatkan socket sebagai media komunikasi. Socket berfungsi sebagai saluran atau terowongan yang memungkinkan pertukaran data dua arah secara bolak-balik. Dengan menggunakan socket programming, komunikasi dapat dilakukan antar bahasa pemrograman yang berbeda, antar tingkatan pengguna, antar komputer, atau kombinasi dari semuanya. Socket bertindak sebagai sebuah cara untuk berkomunikasi dengan program atau node lain melalui file descriptor, di mana proses komunikasi dilakukan seperti membaca atau menulis file. Perbedaan utamanya, pada file descriptor, tujuan komunikasi adalah sebuah file, sementara pada socket, tujuannya adalah komputer atau node lain. Setelah koneksi berhasil dibuat dengan fungsi socket(), antarmuka penggunaannya hampir sama dengan file. Socket ini pada dasarnya adalah abstraksi perangkat lunak yang berperan sebagai "terminal" dalam hubungan antara dua mesin atau proses yang saling terhubung.

Socket memiliki elemen-elemen utama berikut:

1. Protokol
2. Local IP
3. Local Port
4. Remote IP
5. Remote Port

2.2. TCP Server

TCP, atau Transmission Control Protocol, adalah protokol komunikasi yang memungkinkan komputer bertukar informasi melalui jaringan. Sementara itu, IP, atau Internet Protocol, berfungsi untuk mengidentifikasi alamat IP dari aplikasi atau perangkat guna mengirimkan data dan membentuk lapisan jaringan dalam model OSI. TCP bertanggung jawab untuk mengatur bagaimana data ditransfer melalui jaringan, termasuk memastikan pengiriman data yang andal.

Saat pengguna mengirimkan permintaan HTTP ke server, pertama-tama koneksi TCP dibuat, karena HTTP berjalan di atas TCP sebagai lapisan transport. Ketika pengguna mengetikkan URL ke dalam browser, browser akan membuat socket TCP menggunakan alamat IP dan nomor port, lalu mulai mengirimkan data melalui socket tersebut. Data ini dikirim dalam bentuk paket byte melalui jaringan. Server kemudian merespons permintaan tersebut. Keunggulan koneksi TCP adalah adanya pengakuan dari server untuk setiap paket yang diterima, sehingga klien dapat mengirim ulang data jika ada

paket yang hilang. Setiap paket dilengkapi nomor urut yang membantu server menyusun kembali data dengan benar setelah diterima.

2.3. Chatting

Chatting atau chit chat merujuk pada komunikasi antara dua orang atau lebih yang dapat dilakukan secara langsung maupun melalui internet. Saat ini, komunikasi tersebut dapat dilakukan menggunakan pesan teks seperti Short Message Service (SMS) dan Multimedia Messaging Service (MMS) atau melalui platform seperti Slack, Microsoft Teams, serta media sosial seperti Facebook dan Twitter.

Dilihat dari jumlah pesertanya, chatting terbagi menjadi dua jenis: group chat dan private chat. Group chat melibatkan lebih dari dua orang yang berkomunikasi dalam sebuah chat room atau channel, di mana setiap pesan yang dikirim dapat dibaca oleh semua peserta dalam ruangan tersebut. Sementara itu, private chat hanya melibatkan dua orang, sehingga pesan yang dikirim hanya dapat diakses oleh kedua pihak yang berkomunikasi.

2.4. Python

Python adalah bahasa pemrograman yang dapat menjalankan serangkaian instruksi secara langsung (interpretatif) menggunakan pendekatan pemrograman berorientasi objek (Object-Oriented Programming) dan semantik dinamis, yang memudahkan pembacaan sintaksisnya. Python memiliki sintaks dan struktur yang mudah dipelajari, serta sistem manajemen data dan memori otomatis. Selain itu, modul-modul Python terus diperbarui. Python juga menyediakan berbagai fasilitas pendukung dan dapat digunakan di berbagai sistem operasi seperti Linux, Microsoft Windows, Mac OS, Android, Symbian OS, Amiga, Palm, dan lainnya.

2.5. Visual Studio Code

Visual Studio Code adalah editor kode gratis yang dapat dijalankan di perangkat desktop yang menggunakan sistem operasi Windows, Linux, dan MacOS. Editor ini dikembangkan oleh Microsoft, salah satu perusahaan teknologi terkemuka di dunia. Visual Studio Code merupakan perangkat lunak editor yang kuat namun tetap ringan saat digunakan. Editor ini dapat digunakan untuk membuat dan mengedit kode sumber dalam berbagai bahasa pemrograman, seperti JavaScript, TypeScript, dan Node.js.

Beberapa fitur utama dari Visual Studio Code antara lain:

1. Pengeditan Dasar
2. IntelliSense
3. Debugging
4. Marketplace Ekstensi
5. Integrasi dengan Github

Bab 3

Progres Final

3.1 Server.py

```
import socket
import threading
import tkinter as tk
from tkinter import ttk
from datetime import datetime
import base64
from ttkthemes import ThemedTk
import os

class ChatServer:
    def __init__(self, host='0.0.0.0', port=55000):
        self.host = host
        self.port = port
        self.server_socket = None
        self.clients = {}
        self.nicknames = {}
        self.profile_pictures = {}
        self.running = False
        self.file_buffers = {}

        # Main Window with Modern Theme
        self.root = ThemedTk(theme="arc")
        self.root.title("Modern Chat Server")
        self.root.geometry("800x600")
        self.root.configure(bg='#006c84')

        # Style Configuration
        self.style = ttk.Style()
        self.style.configure('Modern.TFrame', background='#006c84')
        self.style.configure('Chat.TFrame', background='#6eb5c0')
        self.style.configure('Modern.TButton',
                             background='#006c84',
                             foreground='black',
                             padding=10)
```

```
        self.style.configure('Header.TLabel',
                             background='#006c84',
                             foreground='black',
                             font=('Helvetica', 12, 'bold'))

        # Main Container
        self.main_container = ttk.Frame(self.root,
style='Modern.TFrame')
        self.main_container.pack(fill=tk.BOTH, expand=True, padx=20,
pady=20)

        # Header Frame
        self.header_frame = ttk.Frame(self.main_container,
style='Modern.TFrame')
        self.header_frame.pack(fill=tk.X, pady=(0, 10))

        # Server Status
        self.status_label = ttk.Label(self.header_frame,
                                      text="Server Stopped",
                                      style='Header.TLabel')
        self.status_label.pack(side=tk.RIGHT, padx=10)

        # Chat Area Container
        self.chat_container = ttk.Frame(self.main_container,
style='Chat.TFrame')
        self.chat_container.pack(fill=tk.BOTH, expand=True)

        # Chat Log with Custom Styling
        self.chat_frame = ttk.Frame(self.chat_container,
style='Chat.TFrame')
        self.chat_frame.pack(fill=tk.BOTH, expand=True, padx=10,
pady=10)

        # Tambahkan komponen Listbox di __init__
        self.client_list_frame = ttk.Frame(self.main_container,
style='Modern.TFrame')
        self.client_list_frame.pack(side=tk.LEFT, fill=tk.Y, padx=(0,
10), pady=10)
```

```
        self.client_list_label = ttk.Label(self.client_list_frame,
text="Connected Clients", style='Header.TLabel')
        self.client_list_label.pack(anchor=tk.W, padx=5)

        self.client_listbox = tk.Listbox(self.client_list_frame,
bg='#6eb5c0', fg='#333333', font=('Helvetica', 10))
        self.client_listbox.pack(fill=tk.BOTH, expand=True, padx=5,
pady=5)

        # Custom Chat Log
        self.chat_log = tk.Text(self.chat_frame,
                                wrap=tk.WORD,
                                font=('Helvetica', 10),
                                bg='#6eb5c0',
                                fg='#333333',
                                relief=tk.FLAT)
        self.chat_log.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
        self.chat_log.tag_configure('server',
                                    justify='left',
                                    background='#006c84',
                                    foreground='white',
                                    spacing1=5,
                                    spacing3=5)
        self.chat_log.tag_configure('client',
                                    justify='left',
                                    background='white',
                                    foreground='#333333',
                                    spacing1=5,
                                    spacing3=5)

        # Modern Scrollbar
        self.scrollbar = ttk.Scrollbar(self.chat_frame,
                                       orient='vertical',
                                       command=self.chat_log.yview)
        self.scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        self.chat_log['yscrollcommand'] = self.scrollbar.set

        # Input Area
        self.input_container = ttk.Frame(self.main_container,
```

```
style='Modern.TFrame')
        self.input_container.pack(fill=tk.X, pady=10)

        # Message Entry
        self.msg_frame = ttk.Frame(self.input_container,
style='Modern.TFrame')
        self.msg_frame.pack(fill=tk.X, pady=5)

        self.message_entry = ttk.Entry(self.msg_frame,
                                       font=('Helvetica', 10))
        self.message_entry.pack(side=tk.LEFT, fill=tk.X, expand=True,
padx=(0, 5))

        self.send_button = ttk.Button(self.msg_frame,
                                      text="Send to All",
                                      command=self.send_message_to_all,
                                      style='Modern.TButton')
        self.send_button.pack(side=tk.RIGHT)

        # Server Control Buttons
        self.button_frame = ttk.Frame(self.main_container,
style='Modern.TFrame')
        self.button_frame.pack(fill=tk.X, pady=10)

        self.start_button = ttk.Button(self.button_frame,
                                       text="Start Server",
                                       command=self.start_server,
                                       style='Modern.TButton')
        self.start_button.pack(side=tk.LEFT, padx=5)

        self.stop_button = ttk.Button(self.button_frame,
                                      text="Stop Server",
                                      command=self.stop_server,
                                      style='Modern.TButton',
                                      state=tk.DISABLED)
        self.stop_button.pack(side=tk.LEFT, padx=5)

        # Bind enter key to send message
        self.message_entry.bind('<Return>', lambda e:
```

```
self.send_message_to_all())

    def broadcast(self, message, _client=None):
        for client in self.clients.values():
            if client != _client:
                try:
                    client.send(message)
                except:
                    self.remove_client(client)

    def handle_file_transfer(self, sender_client, message):
        try:
            # Parse file info from message with maxsplit to prevent
splitting file data
            parts = message.split(':', 2)
            if len(parts) != 3:
                raise ValueError("Invalid file message format")

            _, file_name, file_data = parts

            # Clean up base64 data and handle padding
            file_data = file_data.strip()
            missing_padding = len(file_data) % 4
            if missing_padding:
                file_data += '=' * (4 - missing_padding)

            try:
                # Decode file data
                decoded_data = base64.b64decode(file_data)
                file_size = len(decoded_data)

                # Check file size
                if file_size > 100 * 1024 * 1024:  # 100MB limit
                    sender_client.send("ERROR:File too large (max
100MB)".encode('utf-8'))
                    return

                # Save the file to disk
                save_path = os.path.join("received_files", file_name)
```

```
                os.makedirs(os.path.dirname(save_path),
exist_ok=True)

                with open(save_path, 'wb') as file:
                    file.write(decoded_data)

                # Create message with original file data to maintain
padding
                file_message = f"FILE:{file_name}:{file_data}"

                # Broadcast file to all clients
                self.broadcast(file_message.encode('utf-8'),
sender_client)
                self.update_chat_log(f"File '{file_name}' transferred
successfully", 'server')

            except base64.binascii.Error as e:
                self.update_chat_log(f"Error decoding file data:
{e}", 'server')

        except Exception as e:
            self.update_chat_log(f"Error in file transfer: {e}",
'server')

    def handle_client(self, client, nickname):
        while self.running:
            try:
                message = client.recv(65536)
                if not message:
                    self.remove_client(client)
                    break

                decoded_message = message.decode('utf-8')
                self.save_message_to_file(decoded_message)

                if decoded_message.startswith("MSG:"):
                    self.broadcast(message, client)
                    self.update_chat_log(f"{decoded_message[4:]}",
'client')
```

```
                elif decoded_message.startswith("PRIVATE:"):
                    parts = decoded_message.split(':', 3)
                    recipient = parts[1]
                    if recipient in self.clients:
                        self.clients[recipient].send(message)
                elif decoded_message.startswith("FILE:"):
                    self.handle_file_transfer(client,
decoded_message)
            except Exception as e:
                self.update_chat_log(f"An error occurred: {e}",
'server')
                self.remove_client(client)
                break

    def save_message_to_file(self, message):
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        with open("chat_log.txt", "a", encoding='utf-8') as file:
            file.write(f"[{timestamp}] {message}\n")

    def remove_client(self, client):
        for nickname, c in self.clients.items():
            if c == client:
                break
        del self.clients[nickname]
        del self.nicknames[nickname]
        client.close()
        self.broadcast(f"MSG:{nickname} left the
chat!\n".encode('utf-8'))
        self.update_chat_log(f"{nickname} left the chat!", 'server')
        self.update_client_list()  # Perbarui daftar klien

    def accept_connections(self):
        while self.running:
            try:
                client, address = self.server_socket.accept()
                client.send('NICKNAME'.encode('utf-8'))
                nickname = client.recv(1024).decode('utf-8')

                self.clients[nickname] = client
```

```
                self.nicknames[nickname] = client

                self.update_chat_log(f"{nickname} connected!",
'server')
                self.update_client_list()  # Perbarui daftar klien
                client.send('Connected to the
server!'.encode('utf-8'))

                thread = threading.Thread(target=self.handle_client,
args=(client, nickname))
                thread.start()
            except Exception as e:
                if self.running:
                    self.update_chat_log(f"An error occurred: {e}",
'server')
                break

    # Tambahkan metode untuk memperbarui Listbox
    def update_client_list(self):
        self.client_listbox.delete(0, tk.END)
        for nickname in self.clients.keys():
            self.client_listbox.insert(tk.END, nickname)

    def send_message_to_all(self):
        message = self.message_entry.get().strip()
        if message:
            formatted_message = f"MSG:Server: {message}"
            self.update_chat_log(f"Server: {message}", 'server')
            self.message_entry.delete(0, tk.END)
            self.broadcast(formatted_message.encode('utf-8'))

    def update_chat_log(self, message, tag='server'):
        timestamp = datetime.now().strftime("%H:%M")
        self.chat_log.config(state=tk.NORMAL)
        self.chat_log.insert(tk.END, f"[{timestamp}] {message}\n",
tag)
        self.chat_log.see(tk.END)
        self.chat_log.config(state=tk.DISABLED)
```

```
    def start_server(self):
        if not self.running:
            self.running = True
            self.server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
            self.server_socket.bind((self.host, self.port))
            self.server_socket.listen(5)

            self.start_button.config(state=tk.DISABLED)
            self.stop_button.config(state=tk.NORMAL)
            self.status_label.config(text="Server Running")

            server_thread =
threading.Thread(target=self.accept_connections)
            server_thread.start()

            self.update_chat_log("Server Started...", 'server')

    def stop_server(self):
        if self.running:
            self.running = False
            for client in self.clients.values():
                client.close()
            self.clients.clear()
            self.nicknames.clear()

            if self.server_socket:
                self.server_socket.close()
                self.server_socket = None

            self.start_button.config(state=tk.NORMAL)
            self.stop_button.config(state=tk.DISABLED)
            self.status_label.config(text="Server Stopped")

            self.update_chat_log("Server Stopped...", 'server')

    def center_window(self):
        screen_width = self.root.winfo_screenwidth()
        screen_height = self.root.winfo_screenheight()
```

```
        x = (screen_width - 800) // 2
        y = (screen_height - 600) // 2
        self.root.geometry(f"800x600+{x}+{y}")

    def create_tooltip(self, widget, text):
        def enter(event):
            self.tooltip = tk.Toplevel()
            self.tooltip.wm_overrideredirect(True)

self.tooltip.wm_geometry(f"+{event.x_root+10}+{event.y_root+10}")

            label = ttk.Label(self.tooltip, text=text,
background="#ffffe0", relief='solid', borderwidth=1)
            label.pack()

        def leave(event):
            if hasattr(self, 'tooltip'):
                self.tooltip.destroy()

        widget.bind('<Enter>', enter)
        widget.bind('<Leave>', leave)

    def add_tooltips(self):
        self.create_tooltip(self.send_button, "Send message to all
clients (Enter)")
        self.create_tooltip(self.start_button, "Start the chat
server")
        self.create_tooltip(self.stop_button, "Stop the chat server")

    def run(self):
        self.add_tooltips()
        self.center_window()
        try:
            self.root.iconbitmap('server_icon.ico')
        except:
            pass
        self.root.protocol("WM_DELETE_WINDOW", self.stop_server)
        self.root.mainloop()
```

```
if __name__ == "__main__":
    server = ChatServer()
    server.run()
```

Fungsi Utama Program

1. **Server Chat:**
   - Mengelola komunikasi klien secara real-time.
   - Mendukung pesan broadcast (MSG), privat (PRIVATE), dan transfer file (FILE).
2. **Transfer File:**
   - Mengirim dan menerima file hingga 100 MB (dengan Base64 encoding).
   - File disimpan di direktori received_files.
3. **Modern GUI:**
   - Menggunakan ttkthemes untuk antarmuka modern.
   - Fitur: log chat, daftar klien, dan kontrol server (Start/Stop).
4. **Logging Pesan:**
   - Menyimpan semua pesan ke file chat_log.txt.

3.2 Client 1

```
import socket
import threading
import tkinter as tk
from tkinter import filedialog, simpledialog, ttk
import base64
import os
from datetime import datetime
from ttkthemes import ThemedTk
from PIL import Image, ImageTk
import io

class ModernChatClient:
    def __init__(self):
        # Inisialisasi variabel dan pengaturan GUI
        self.client = None
        self.host = '127.0.0.1'
        self.port = 55000
        self.nickname = ""
        self.running = False
        self.images = []  # Menyimpan referensi gambar
```

```
        # Pengaturan GUI
        self.root = ThemedTk(theme="arc")
        self.root.title("Modern Chat")
        self.root.geometry("800x600")
        self.root.configure(bg='#006c84')

        # Style Configuration
        self.style = ttk.Style()
        self.style.configure('Modern.TFrame', background='#006c84')
        self.style.configure('Chat.TFrame', background='#6eb5c0')
        self.style.configure('Modern.TButton',
                             background='#006c84',
                             foreground='black',
                             padding=10)
        self.style.configure('Header.TLabel',
                             background='#006c84',
                             foreground='black',
                             font=('Helvetica', 12, 'bold'))


        # Main Container
        self.main_container = ttk.Frame(self.root,
style='Modern.TFrame')
        self.main_container.pack(fill=tk.BOTH, expand=True, padx=20,
pady=20)

        # Top Frame for Connect Button and Status
        self.top_frame = ttk.Frame(self.main_container,
style='Modern.TFrame')
        self.top_frame.pack(fill=tk.X, pady=(0, 10))

        # Connect Button at Top
        self.connect_button = ttk.Button(self.top_frame,
                                         text="Connect to Server",

command=self.connect_to_server,
                                         style='Modern.TButton')
        self.connect_button.pack(side=tk.LEFT, padx=5)
```

```
        # Connection Status
        self.status_label = ttk.Label(self.top_frame,
                                      text="Disconnected",
                                      style='Header.TLabel')
        self.status_label.pack(side=tk.RIGHT, padx=10)

        # Chat Area Container
        self.chat_container = ttk.Frame(self.main_container,
style='Chat.TFrame')
        self.chat_container.pack(fill=tk.BOTH, expand=True)

        # Chat Log with Custom Styling
        self.chat_frame = ttk.Frame(self.chat_container,
style='Chat.TFrame')
        self.chat_frame.pack(fill=tk.BOTH, expand=True, padx=10,
pady=10)

        # Custom Chat Log
        self.chat_log = tk.Text(self.chat_frame,
                                wrap=tk.WORD,
                                font=('Helvetica', 10),
                                bg='#6eb5c0',
                                fg='#333333',
                                relief=tk.FLAT)
        self.chat_log.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
        self.chat_log.tag_configure('sent',
                                    justify='right',
                                    background='#006c84',
                                    foreground='white',
                                    spacing1=5,
                                    spacing3=5,
                                    lmargin2=50)
        self.chat_log.tag_configure('received',
                                    justify='left',
                                    background='white',
                                    foreground='#333333',
                                    spacing1=5,
                                    spacing3=5,
```

```
                                    rmargin=50)

        # Modern Scrollbar
        self.scrollbar = ttk.Scrollbar(self.chat_frame,
                                       orient='vertical',
                                       command=self.chat_log.yview)
        self.scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        self.chat_log['yscrollcommand'] = self.scrollbar.set

        # Input Area
        self.input_container = ttk.Frame(self.main_container,
style='Modern.TFrame')
        self.input_container.pack(fill=tk.X, pady=10)

        # Message Entry
        self.msg_frame = ttk.Frame(self.input_container,
style='Modern.TFrame')
        self.msg_frame.pack(fill=tk.X, pady=5)

        self.msg_entry = ttk.Entry(self.msg_frame,
                                   font=('Helvetica', 10))
        self.msg_entry.pack(side=tk.LEFT, fill=tk.X, expand=True,
padx=(0, 5))

        self.send_button = ttk.Button(self.msg_frame,
                                      text="Send",
                                      command=self.send_message,
                                      style='Modern.TButton',
                                      state=tk.DISABLED)
        self.send_button.pack(side=tk.RIGHT)

        # Private Message Entry
        self.private_frame = ttk.Frame(self.input_container,
style='Modern.TFrame')
        self.private_frame.pack(fill=tk.X, pady=5)

        self.private_msg_entry = ttk.Entry(self.private_frame,
                                           font=('Helvetica', 10))
        self.private_msg_entry.pack(side=tk.LEFT, fill=tk.X,
```

```
expand=True, padx=(0, 5))

        self.private_button = ttk.Button(self.private_frame,
                                         text="Send Private",

command=self.send_private_message,
                                         style='Modern.TButton',
                                         state=tk.DISABLED)
        self.private_button.pack(side=tk.RIGHT)

        # File Transfer Area
        self.file_frame = ttk.Frame(self.input_container,
style='Modern.TFrame')
        self.file_frame.pack(fill=tk.X, pady=5)

        self.file_progress = ttk.Progressbar(self.file_frame,
                                             orient='horizontal',
                                             length=300,
                                             mode='determinate',

style='Modern.Horizontal.TProgressbar')
        self.file_progress.pack(side=tk.LEFT, fill=tk.X,
expand=True, padx=(0, 5))

        self.send_file_button = ttk.Button(self.file_frame,
                                           text="Send File",
                                           command=self.send_file,
                                           style='Modern.TButton',
                                           state=tk.DISABLED)
        self.send_file_button.pack(side=tk.RIGHT)

        # Bind enter key to send message
        self.msg_entry.bind('<Return>', lambda e:
self.send_message())
        self.private_msg_entry.bind('<Return>', lambda e:
self.send_private_message())


    def connect_to_server(self):
```

```
        if self.running:
            self.chat_log.insert(tk.END, "Already connected to the
server.\n")
            return

        try:
            self.client = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
            self.client.connect((self.host, self.port))
            self.running = True

            if not self.nickname:
                nickname_window = simpledialog.askstring("Nickname",
"Enter your nickname:")
                if nickname_window:
                    self.nickname = nickname_window
                else:
                    self.disconnect_from_server()
                    return

            self.client.send(self.nickname.encode('utf-8'))

            response = self.client.recv(1024).decode('utf-8')
            if response == 'NICKNAME':
                receive_thread =
threading.Thread(target=self.receive)
                receive_thread.start()

                self.send_button.config(state=tk.NORMAL)
                self.send_file_button.config(state=tk.NORMAL)
                self.private_button.config(state=tk.NORMAL)
                self.status_label.config(text="Connected")
                self.chat_log.insert(tk.END, "Connected to the
server.\n")
            else:
                self.chat_log.insert(tk.END, f"Connection failed:
{response}\n")
                self.disconnect_from_server()
```

```
        except Exception as e:
            self.chat_log.insert(tk.END, f"Could not connect to
server: {e}\n")
            self.disconnect_from_server()

    def disconnect_from_server(self):
        self.running = False
        if self.client:
            try:
                self.client.close()
            except:
                pass
            self.client = None

        self.send_button.config(state=tk.DISABLED)
        self.send_file_button.config(state=tk.DISABLED)
        self.private_button.config(state=tk.DISABLED)
        self.status_label.config(text="Disconnected")
        self.chat_log.insert(tk.END, "Disconnected from the
server.\n")

    def send_message(self):
        if not self.running:
            self.chat_log.insert(tk.END, "Cannot send message. Not
connected to server.\n")
            return

        message = self.msg_entry.get().strip()
        if message:
            try:
                timestamp = datetime.now().strftime("%H:%M")
                formatted_message = f"MSG:{self.nickname}:
{message}"
                self.client.send(formatted_message.encode('utf-8'))
                self.chat_log.insert(tk.END, f"[{timestamp}] You:
{message}\n", 'sent')
                self.msg_entry.delete(0, tk.END)
            except Exception as e:
                self.chat_log.insert(tk.END, f"Error sending
```

```
message: {e}\n")
                self.disconnect_from_server()

    def send_private_message(self):
        if not self.running:
            self.chat_log.insert(tk.END, "Cannot send private
message. Not connected to server.\n")
            return

        recipient = simpledialog.askstring("Recipient", "Enter
recipient's nickname:")
        if not recipient:
            return

        message = self.private_msg_entry.get().strip()
        if not message:
            self.chat_log.insert(tk.END, "Message cannot be
empty.\n")
            return

        try:
            timestamp = datetime.now().strftime("%H:%M")
            formatted_message =
f"PRIVATE:{recipient}:{self.nickname}: {message}"
            self.client.send(formatted_message.encode('utf-8'))
            self.chat_log.insert(tk.END, f"[{timestamp}] Private to
{recipient}: {message}\n", 'sent')
            self.private_msg_entry.delete(0, tk.END)
        except Exception as e:
            self.chat_log.insert(tk.END, f"Error sending private
message: {e}\n")
            self.disconnect_from_server()

    def send_file(self):
        if not self.running:
            self.chat_log.insert(tk.END, "Cannot send file. Not
connected to server.\n")
            return
```

```
        file_path = filedialog.askopenfilename(
            title="Select File",
            filetypes=[
                ("Image files", "*.jpg *.jpeg *.png *.gif *.bmp"),
                ("All files", "*.*")
            ]
        )

        if file_path:
            try:
                file_size = os.path.getsize(file_path)
                if file_size > 100 * 1024 * 1024:  # 100MB limit
                    self.chat_log.insert(tk.END, "File too large
(max 100MB)\n")
                    return

                self.file_progress['value'] = 0
                self.file_progress['maximum'] = file_size

                # Read and encode file with proper error handling
                with open(file_path, 'rb') as file:
                    file_data = file.read()
                    encoded_file =
base64.b64encode(file_data).decode('utf-8')

                    # Ensure proper padding
                    while len(encoded_file) % 4 != 0:
                        encoded_file += '='

                    # Send file with encoded data
                    file_message =
f"FILE:{os.path.basename(file_path)}:{encoded_file}"
                    self.client.send(file_message.encode('utf-8'))

                    self.file_progress['value'] = file_size
                    self.chat_log.insert(tk.END, f"Sent file:
{os.path.basename(file_path)}\n", 'sent')

                    # Preview file if it's an image
```

```
                    file_ext =
os.path.splitext(file_path)[1].lower()
                    if file_ext in ['.jpg', '.jpeg', '.png', '.gif',
'.bmp']:
                        self.display_image(file_data,
is_sender=True)

            except Exception as e:
                self.chat_log.insert(tk.END, f"Error sending file:
{e}\n")
            finally:
                self.file_progress['value'] = 0

    def display_image(self, file_data, is_sender=False):
        try:
            if not file_data:
                raise ValueError("Empty image data received")

            # Buat bytes object baru untuk memastikan data tidak
corrupt
            if isinstance(file_data, str):
                # Jika data masih dalam bentuk string base64
                file_data = base64.b64decode(file_data)

            # Gunakan BytesIO dengan proper cleanup
            with io.BytesIO(file_data) as image_buffer:
                image = Image.open(image_buffer)
                # Convert mode jika diperlukan
                if image.mode != 'RGB':
                    image = image.convert('RGB')

                # Resize image
                max_size = (200, 200)
                image.thumbnail(max_size)

                # Convert ke PhotoImage
                photo = ImageTk.PhotoImage(image)
                self.images.append(photo)  # Simpan referensi
```

```
                # Tampilkan di chat
                self.chat_log.insert(tk.END, '\n')
                if is_sender:
                    self.chat_log.insert(tk.END, '', 'sent')
                else:
                    self.chat_log.insert(tk.END, '', 'received')
                self.chat_log.image_create(tk.END, image=photo)
                self.chat_log.insert(tk.END, '\n')
                self.chat_log.see(tk.END)

        except Exception as e:
            self.chat_log.insert(tk.END, f"Error displaying image:
{str(e)}\n")


    def send_file(self):
        if not self.running:
            self.chat_log.insert(tk.END, "Cannot send file. Not
connected to server.\n")
            return

        # Pilih file yang ingin dikirim
        file_path = filedialog.askopenfilename(
            title="Select File",
            filetypes=[("All files", "*.*"), ("Image files", "*.jpg
*.jpeg *.png *.gif *.bmp")]
        )

        if not file_path:
            self.chat_log.insert(tk.END, "No file selected.\n")
            return

        try:
            # Dapatkan ukuran file
            file_size = os.path.getsize(file_path)
            max_size = 100 * 1024 * 1024  # 100 MB

            if file_size > max_size:
                self.chat_log.insert(tk.END, "File too large. Max
```

```
size is 100MB.\n")
                return

            # Baca file dan encode ke base64
            with open(file_path, 'rb') as file:
                file_data = file.read()
                encoded_file =
base64.b64encode(file_data).decode('utf-8')

            # Kirim file ke server
            file_name = os.path.basename(file_path)
            file_message = f"FILE:{file_name}:{encoded_file}"
            self.client.send(file_message.encode('utf-8'))

            # Update progress bar
            self.file_progress['value'] = 100
            self.chat_log.insert(tk.END, f"Sent file:
{file_name}\n", 'sent')

            # Jika file adalah gambar, tampilkan thumbnail
            file_ext = os.path.splitext(file_path)[1].lower()
            if file_ext in ['.jpg', '.jpeg', '.png', '.gif',
'.bmp']:
                self.display_image(file_data, is_sender=True)

        except Exception as e:
            self.chat_log.insert(tk.END, f"Error sending file:
{e}\n")
        finally:
            self.file_progress['value'] = 0



    def receive(self):
        while self.running:
            try:
                message = self.client.recv(65536).decode('utf-8')
                timestamp = datetime.now().strftime("%H:%M")
```

```
                if not message:
                    raise ConnectionError("Connection lost")

                if message.startswith("MSG:"):
                    try:
                        _, sender, text = message.split(":", 2)
                        self.chat_log.insert(tk.END, f"[{timestamp}]
{sender}: {text}\n", 'received')
                    except ValueError:
                        self.chat_log.insert(tk.END, f"[{timestamp}]
Invalid message format\n")

                elif message.startswith("PRIVATE:"):
                    try:
                        _, recipient, sender, private_msg =
message.split(":", 3)
                        if recipient == self.nickname:
                            self.chat_log.insert(tk.END,
f"[{timestamp}] Private from {sender}: {private_msg}\n", 'received')
                    except ValueError:
                        self.chat_log.insert(tk.END, f"[{timestamp}]
Invalid private message format\n")

                elif message.startswith("FILE:"):
                    try:
                        parts = message.split(":", 2)
                        if len(parts) != 3:
                            raise ValueError("Invalid file message
format")

                        _, file_name, encoded_file = parts

                        # Clean up base64 data
                        encoded_file = encoded_file.strip()

                        # Fix padding
                        missing_padding = len(encoded_file) % 4
                        if missing_padding:
                            encoded_file += '=' * (4 -
```

```
missing_padding)

                        try:
                            # Decode file data
                            file_data =
base64.b64decode(encoded_file)

                            # Save file
                            os.makedirs("received_files",
exist_ok=True)
                            received_file_path =
os.path.join("received_files", f"received_{file_name}")

                            with open(received_file_path, 'wb') as
file:
                                file.write(file_data)

                            self.chat_log.insert(tk.END,
f"[{timestamp}] Received file: {file_name}\n", 'received')

                            # Check if file is an image and display
it
                            file_ext =
os.path.splitext(file_name)[1].lower()
                            if file_ext in ['.jpg', '.jpeg', '.png',
'.gif', '.bmp']:
                                self.display_image(file_data)

                        except base64.binascii.Error as be:
                            self.chat_log.insert(tk.END,
f"[{timestamp}] Error decoding file: {str(be)}\n")

                    except Exception as e:
                        self.chat_log.insert(tk.END, f"[{timestamp}]
Error receiving file: {str(e)}\n")

                    except Exception as e:
                        self.chat_log.insert(tk.END, f"[{timestamp}]
Error receiving file: {str(e)}\n")
```

```
            except Exception as e:
                if self.running:
                    self.chat_log.insert(tk.END, f"Disconnected from
server: {str(e)}\n")
                    self.disconnect_from_server()
                break


    def run(self):
    # Add tooltips
        self.add_tooltips()

        # Center window on screen
        self.center_window()

        # Set window icon (if available)
        try:
            self.root.iconbitmap('chat_icon.ico')
        except Exception as e:
            print(f"Error loading icon: {e}")

        self.root.protocol("WM_DELETE_WINDOW", self.on_closing)
        self.root.mainloop()

    def add_tooltips(self):
        # Create tooltips for buttons
        self.create_tooltip(self.send_button, "Send message
(Enter)")
        self.create_tooltip(self.private_button, "Send private
message")
        self.create_tooltip(self.send_file_button, "Send a file to
chat")
        self.create_tooltip(self.connect_button, "Connect to chat
server")

    def create_tooltip(self, widget, text):
        def enter(event):
            self.tooltip = tk.Toplevel()
```

```
            self.tooltip.wm_overrideredirect(True)

self.tooltip.wm_geometry(f"+{event.x_root+10}+{event.y_root+10}")

            label = ttk.Label(self.tooltip, text=text,
background="#ffffe0", relief='solid', borderwidth=1)
            label.pack()

        def leave(event):
            if hasattr(self, 'tooltip'):
                self.tooltip.destroy()

        widget.bind('<Enter>', enter)
        widget.bind('<Leave>', leave)

    def center_window(self):
        # Get screen width and height
        screen_width = self.root.winfo_screenwidth()
        screen_height = self.root.winfo_screenheight()

        # Calculate position
        x = (screen_width - 800) // 2
        y = (screen_height - 600) // 2

        # Set the position of the window to the center of the screen
        self.root.geometry(f"800x600+{x}+{y}")

    def on_closing(self):
        self.running = False
        if self.client:
            try:
                self.client.close()
            except:
                pass
        self.root.destroy()

if __name__ == "__main__":
    client = ModernChatClient()
    client.run()
```

3.3 Fungsi Utama Program

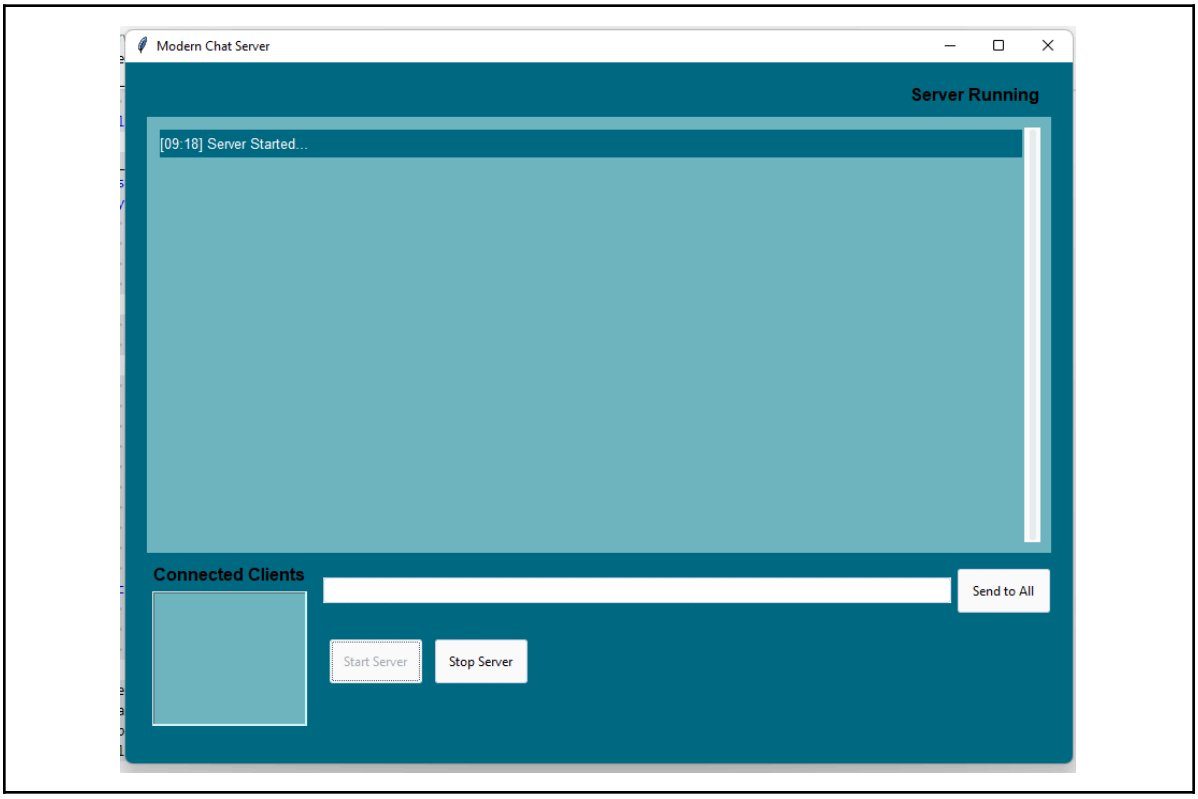
1. **Antarmuka Modern:**
   - Dibangun menggunakan Tkinter dan tema ttkthemes.
   - Menyediakan chat log, area input pesan, dan tombol interaktif.
2. **Fitur Chat:**
   - Pesan Umum: Mengirim dan menerima pesan ke semua pengguna.
   - Pesan Pribadi: Mengirim pesan ke pengguna tertentu.
   - Transfer File: Mengirim file, mendukung tampilan thumbnail untuk gambar.
3. **Koneksi ke Server:**
   - Koneksi socket ke server, mengelola pengiriman/penerimaan pesan dengan format standar (MSG, PRIVATE, FILE).

3.4 Fitur tambahan:

1. Stop server
2. GUI
3. Akses melalui hp → Aplikasi Pydroid 3

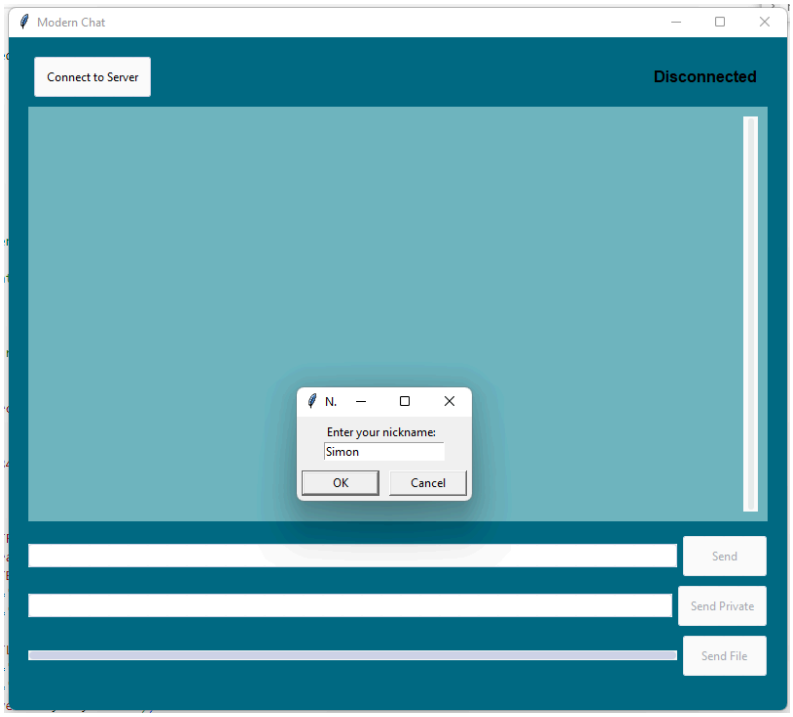
3.5 UI

1. Server.py



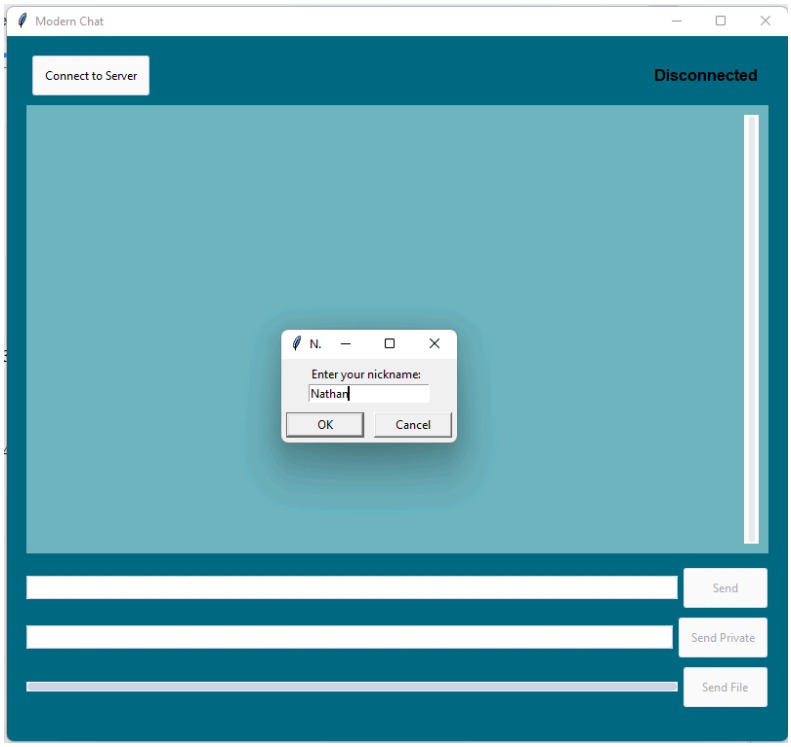
GUI ini adalah antarmuka server chat yang menampilkan log aktivitas, daftar klien terhubung, kolom input pesan, dan tombol untuk mengelola server (Start, Stop, dan Send to All). Server sedang berjalan.

2. **Client 1.py (Perangkat yang sama)**



Run client 1 setelah itu klik Connect to server dan buat nama kita masing masing.

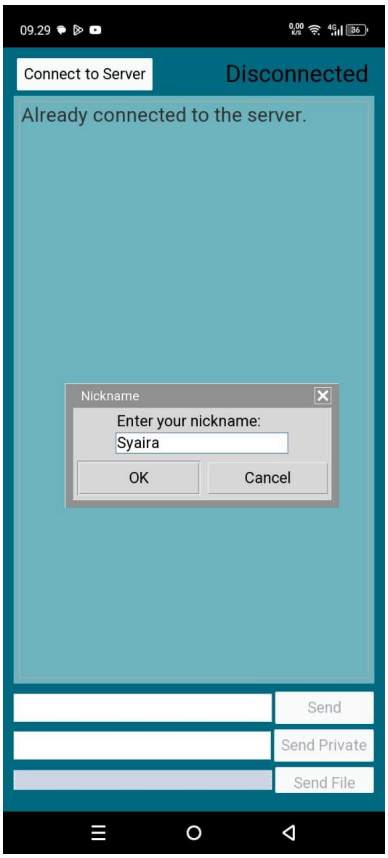
3. **Client 2.py (Berbeda perangkat menggunakan PC)**



antarmuka klien chat. Terdapat dialog untuk memasukkan nama pengguna

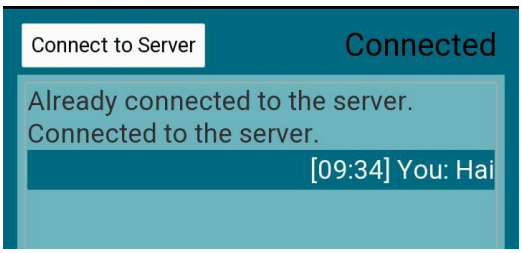
(nickname), tombol "Connect to Server", kolom pesan, dan opsi untuk mengirim pesan umum, pribadi, atau file. Status saat ini "Disconnected".

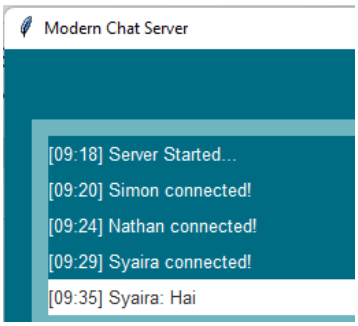
4. **Client 3.py (FITUR TAMBAHAN DARI HANDPHONE)**



menunjukkan versi mobile dari klien chat. Terdapat dialog untuk memasukkan nama pengguna, tombol "Connect to Server", dan fitur pengiriman pesan umum, pribadi, serta file. Status saat ini "Disconnected".

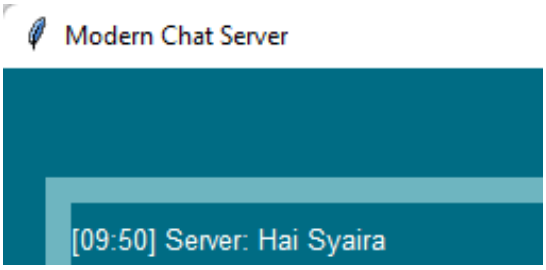
5. **Client 3 mengirimkan pesan ke Server**





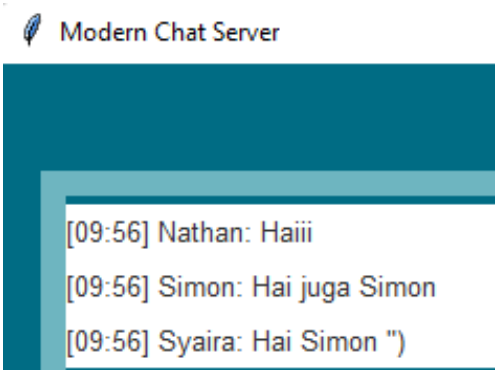
Gambar menunjukkan koneksi ke server berhasil, pesan "Hai" dikirim, dan aktivitas tercatat di log server.

6. **Server dapat merespon chat**



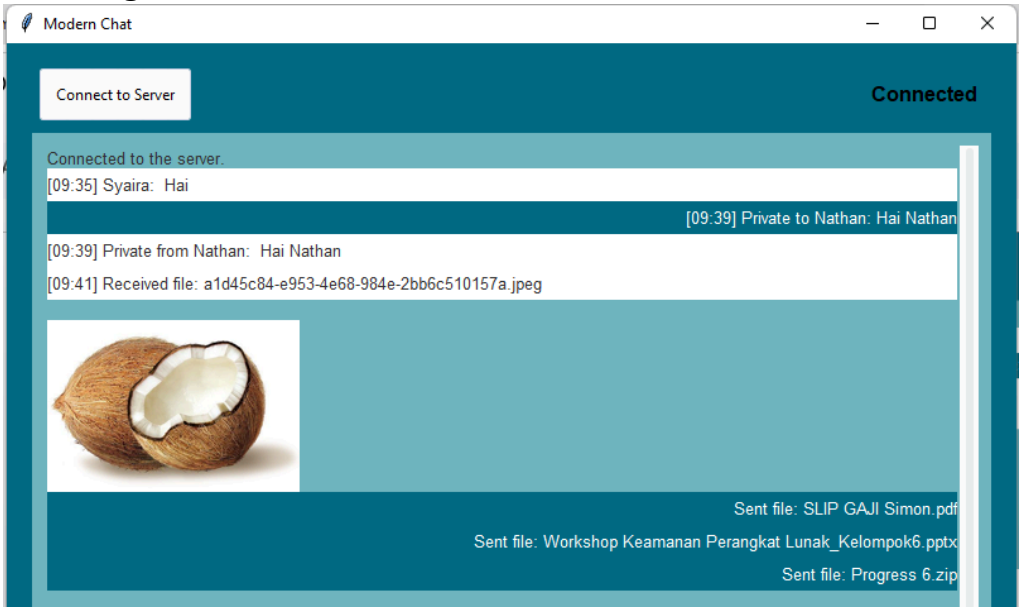
Setelah client mengirim pesan ke server maka server juga dapat merespon chat client

7. **Server dapat menangani permintaan chat dari beberapa Client**



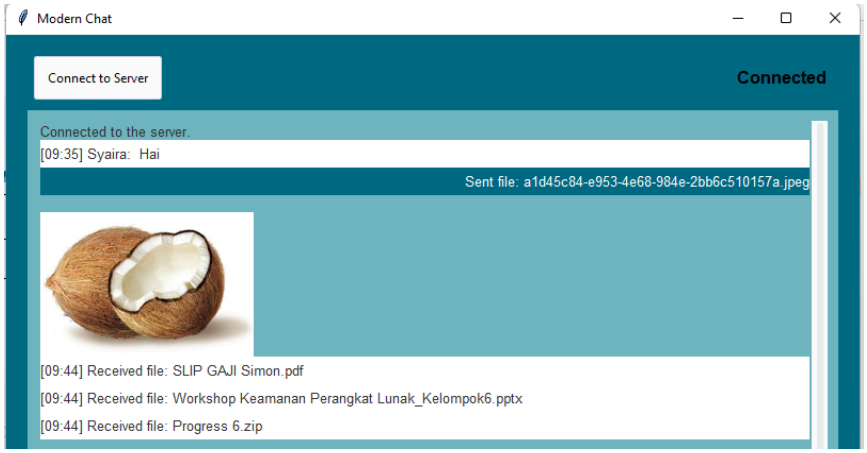
8. Dapat mengirimkan pesan chat berupa file multimedia

Client 1 mengirim



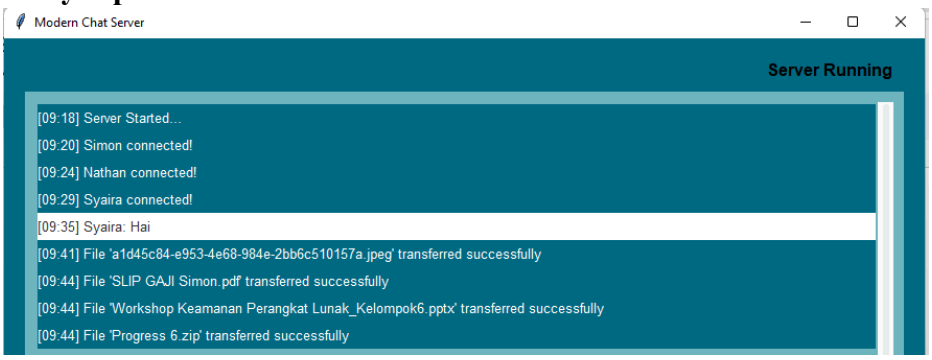
Client 1 mengirim pesan teks dan file multimedia (gambar dan dokumen).

Client lain menerima



Client lain berhasil menerima pesan teks dan semua file yang dikirim.

Server menyimpan



Server mencatat koneksi, pesan, dan transfer file berhasil.

9. **Client 1 mengirimkan pesan ke client lain melalui perantara server (server akan** meneruskan pesan chat ke client yang dituju) berupa private ke Client 2 lainnya

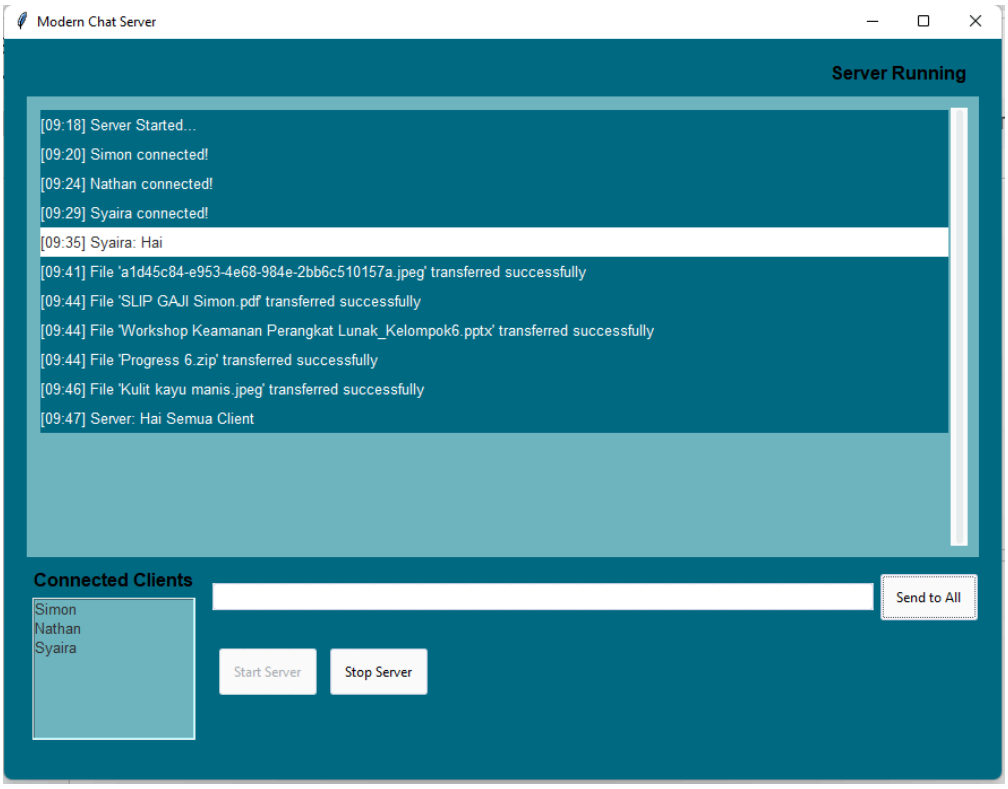
Client 1 mengirim pesan ke Client 2

Client 1 mengirim pesan privat ke Client 2 melalui server.

Client 2 menerima pesan private

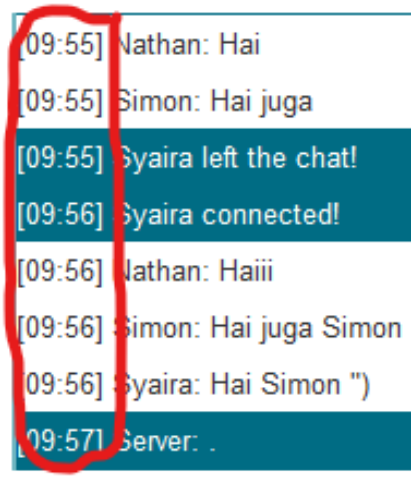
Client 2 menerima pesan privat dari Client 1.

10. Server mengirimkan pesan broadcast ke seluruh Client yang terhubung



antarmuka server chat yang dapat mengirim pesan broadcast ke semua klien yang terhubung. Server juga mencatat log aktivitas seperti koneksi, transfer file, dan pesan yang dikirim. Fitur ini penting untuk memantau aktivitas server secara real-time.

11. Menampilkan waktu untuk semua pesan yang dikirimkan dan diterima



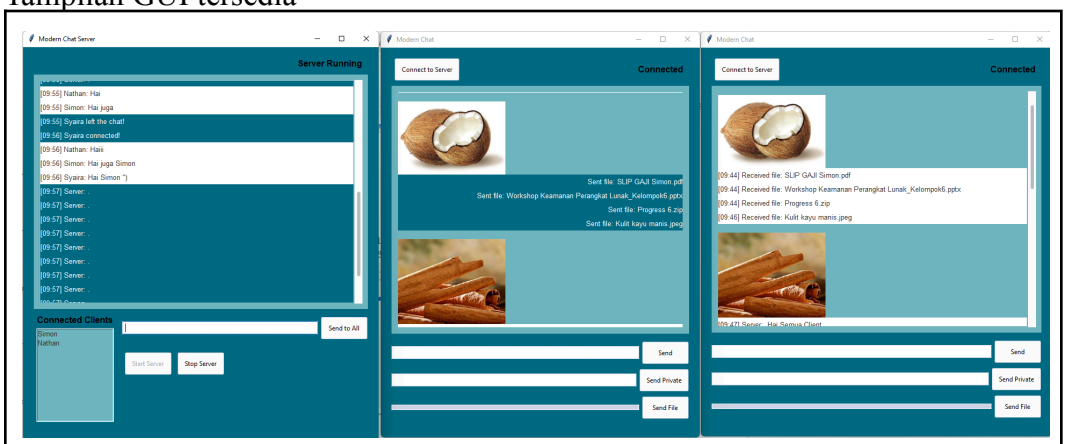
fitur timestamp pada semua pesan yang dikirim dan diterima. Tanda waktu ini memudahkan pengguna untuk mengetahui kapan setiap pesan dikirim atau diterima, berguna untuk melacak kronologi komunikasi.

12. Client 1 mengirimkan pesan ke client lain melalui perantara server isi pesan
chat tersimpan di server (dapat berupa txt atau lainnya)

kemampuan server untuk mencatat log aktivitas komunikasi antar-klien secara rinci, termasuk pesan dan file yang dikirim, yang disimpan untuk keperluan audit atau pemantauan.

13. Bonus:

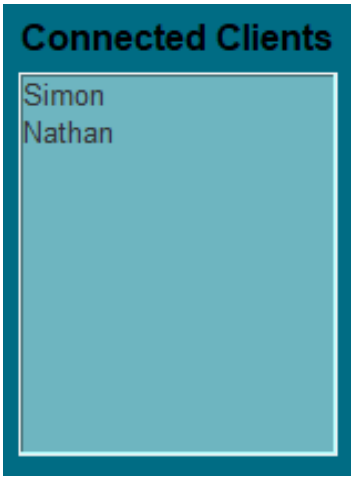
a. Tampilan GUI tersedia



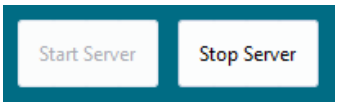
antarmuka GUI yang ramah pengguna dengan fitur pengiriman pesan, file, dan tampilan multimedia, sehingga mendukung pengalaman komunikasi yang interaktif dan visual.

b. Fitur tambahan selain fitur di atas

1. Listbox (Client yang terhubung di server)



2. Start dan Stop server



Connected Clients: Menampilkan daftar klien yang terhubung ke server secara real-time, mempermudah pengelolaan koneksi.
Start dan Stop Server: Memberikan kontrol langsung kepada administrator untuk memulai atau menghentikan server sesuai kebutuhan.
Akses Mobile: Sistem mendukung akses dari perangkat mobile, meningkatkan fleksibilitas dan kemudahan penggunaan.

3. Fitur yang dapat di akses oleh Mobile Phone

tampilan antarmuka berwarna turquoise yang saat ini dalam keadaan terputus dari server. Pengguna diminta untuk memasukkan nickname "Simon" melalui dialog box sebelum bisa terhubung dan menggunakan fitur pengiriman pesan biasa, pesan private, maupun file yang tersedia melalui tombol-tombol di bagian bawah aplikasi.

3.6 Pembagian Kerja

1. **Persentase Pembagian Kerja:**
   a. Simon: 30%
   b. Sabam: 30%
   c. Syaira: 30%
   d. Elnathan: 10%

2. **Poin Nilai dan Kriteria**
   a. 40-49: Aplikasi berjalan dengan baik, tidak ada error ✅ (Sabam dan Simon)

   b. 50-59: Kriteria poin 40 terpenuhi, Client dapat mengirimkan chat ke server, Server dapat merespon chat yang dikirimkan oleh client. ✅ (Syaira, Simon dan Sabam)

   c. 60-69: Kriteria poin 50 terpenuhi, Server dapat menangani permintaan chat dari beberapa client (silahkan menggunakan forking/multiplexing/multithreading) ✅ (Sabam dan Simon)

   d. 70-79: Kriteria poin 60 terpenuhi, Dapat mengirimkan pesan chat berupa file multimedia✅ (Simon dan Syaira)

   e. 80-89: Kriteria poin 70 terpenuhi, Client dapat mengirimkan chat ke client lain melalui perantara server (server akan meneruskan pesan chat ke client yang dituju), Server juga dapat mengirimkan pesan broadcast ke seluruh client yang terhubung, Menampilkan waktu untuk setiap pesan yang dikirim dan diterima. ✅(Syaira, Simon dan Sabam)

   f. 90-100: Kriteria poin 80 terpenuhi, Client dapat mengirimkan chat ke client lain melalui perantara server. Isi pesan chat disimpan di server (isi pesan chat dapat disimpan di file text atau sejenis). ✅ (Syaira dan Sabam)

3. **Tambahan:**
   a. Meperbaiki server, menambahkan button Off server ✅ (Syaira)
   b. Bisa chat dari berbeda perangkat✅ (Simon dan Sabam)
   c. Proposal (Elnathan)

4. **Bonus:**
   a. Tampilan berupa GUI tersedia✅ (Simon)
   b. List Box ✅(Simon)
   c. Menggunakan perangkat Handphone ✅(Simon dan Sabam)

Bab 4
Kesimpulan

Laporan ini merangkum pengembangan aplikasi chat berbasis client-server yang dirancang untuk memenuhi kebutuhan komunikasi modern dengan pendekatan efisien dan inovatif. Proyek ini menggunakan socket programming untuk membangun komunikasi dua arah antara klien dan server, dengan antarmuka grafis (GUI) modern yang dirancang menggunakan Tkinter dan tema dari ttkthemes. Aplikasi ini memiliki fitur utama seperti pengiriman pesan teks dan multimedia, broadcast pesan ke seluruh klien, serta pencatatan waktu pengiriman dan penerimaan pesan, yang semuanya didukung oleh kemampuan server untuk menangani permintaan dari beberapa klien secara bersamaan menggunakan multithreading.

Tambahan fitur meliputi akses melalui perangkat mobile menggunakan aplikasi Pydroid 3, kemampuan server untuk menyimpan pesan sebagai arsip dalam bentuk file teks, dan elemen personalisasi seperti menampilkan daftar klien yang terhubung melalui listbox. Sebagai fitur bonus, aplikasi ini juga mendukung kontrol server dengan tombol Start dan Stop, serta antarmuka yang dapat diakses dari berbagai perangkat, termasuk smartphone.

Secara keseluruhan, proyek ini berhasil menghasilkan aplikasi yang tidak hanya memenuhi kebutuhan komunikasi dasar, tetapi juga menawarkan fleksibilitas, efisiensi bandwidth, dan fitur personalisasi yang meningkatkan pengalaman pengguna. Dengan berbagai fitur yang disediakan, aplikasi ini menunjukkan potensi untuk digunakan sebagai alternatif ringan dan efektif dibandingkan platform chat yang ada saat ini.