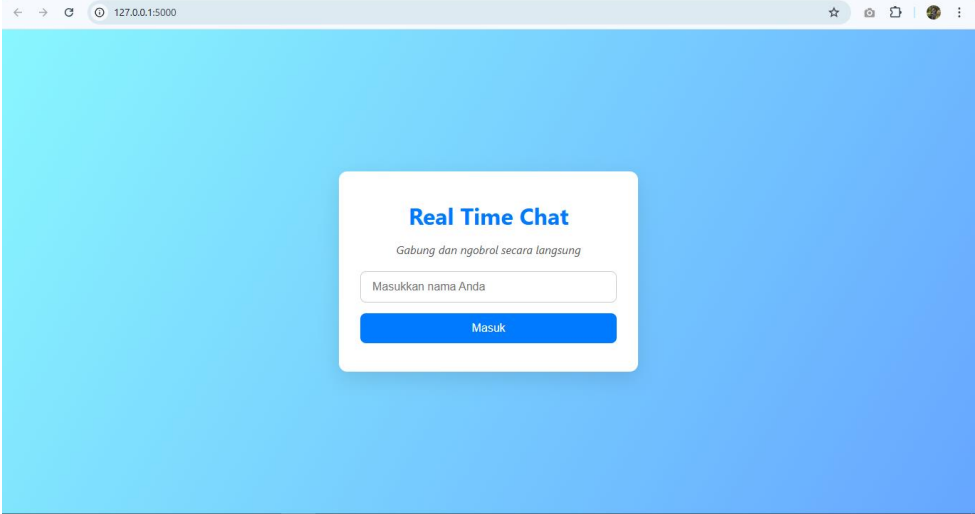
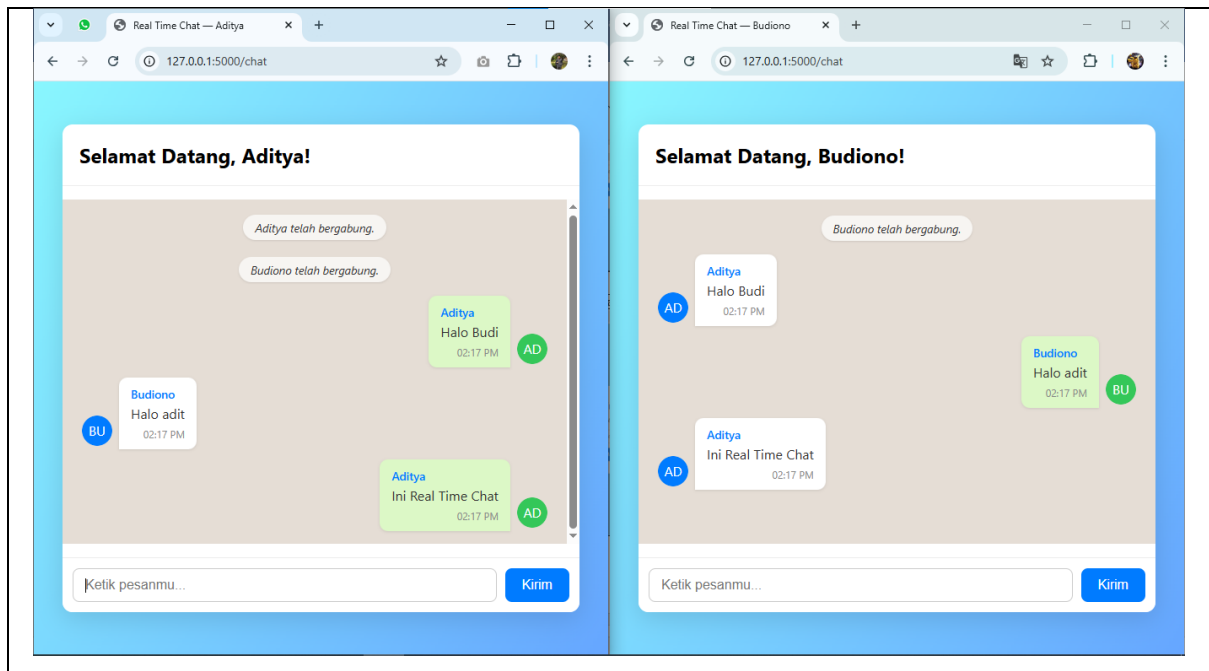
	<h2>Network Programing</h2>
Nama	Aditya Nur Khaerullah
NPM	714242005
Matakuliah	Network Programing

Halaman Login


Halaman Chat



PENJELASAN APLIKASI

Aplikasi ini adalah sebuah sistem chat *real-time* berbasis web yang dibangun menggunakan Python dengan framework **Flask** dan ekstensi **Flask-SocketIO**. Pengguna dapat mengaksesnya melalui browser, memasukkan nama pengguna untuk identifikasi, dan langsung bergabung ke dalam sebuah ruang chat publik. Dengan memanfaatkan teknologi **WebSocket**, aplikasi ini memungkinkan pengiriman dan penerimaan pesan secara instan ke semua pengguna yang terhubung tanpa perlu me-refresh halaman, lengkap dengan notifikasi saat ada pengguna yang bergabung atau keluar.

Fitur Utama Aplikasi Chat

Aplikasi ini memiliki beberapa fitur inti yang bekerja sama untuk menciptakan pengalaman chat yang interaktif dan multi-pengguna.

1. Identifikasi Pengguna (Login Sederhana)

- A. **Deskripsi:** Sebelum dapat bergabung ke dalam ruang chat, setiap pengguna harus memasukkan nama (username) terlebih dahulu. Nama ini akan digunakan sebagai identitas mereka selama sesi chat berlangsung.
- B. **Implementasi Teknis:**
 - Fitur ini ditangani oleh route / pada app.py yang menyajikan halaman login.html.

- Saat pengguna mengirimkan form, nama yang dimasukkan akan disimpan ke dalam **session Flask** (`session['username'] = ...`). Session adalah mekanisme untuk menyimpan informasi spesifik pengguna di sisi server selama pengguna tersebut aktif di browser.

2. Ruang Chat Publik (Public Chat Room)

A. **Deskripsi:** Semua pengguna yang login akan masuk ke dalam satu ruang chat yang sama. Artinya, setiap pesan yang dikirim akan dapat dilihat oleh semua orang yang sedang online.

B. Implementasi Teknis:

- Flask-SocketIO memiliki konsep "**Rooms**". Meskipun tidak terlihat langsung oleh pengguna, secara teknis semua pengguna dimasukkan ke dalam sebuah ruangan bernama 'ruang_chat_utama' saat mereka terhubung (`join_room(room)`).
- Saat pesan dikirim, server menyiarkannya secara spesifik ke semua anggota di ruangan tersebut (`emit(..., room=room)`).

3. Pengiriman Pesan Real-Time (Real-Time Messaging)

A. **Deskripsi:** Ini adalah fitur utama aplikasi. Pengguna dapat mengetik pesan dan mengirimkannya. Pesan tersebut akan muncul secara instan di layar semua pengguna lain tanpa perlu me-refresh halaman.

B. Implementasi Teknis:

- **Frontend:** JavaScript menangkap event pengiriman form, mengambil teks dari kolom input, dan mengirimkannya ke server melalui event `socket.emit('message', ...)`.
- **Backend:** Dekorator `@socketio.on('message')` di `app.py` menerima event ini. Server kemudian langsung menyiarkan pesan tersebut kembali ke semua klien. Kecepatan "real-time" ini dimungkinkan oleh koneksi **WebSocket** yang selalu terbuka.

4. Notifikasi Status Pengguna (Join/Leave Notifications)

A. **Deskripsi:** Aplikasi secara otomatis memberikan notifikasi ketika seorang pengguna baru bergabung ke dalam chat atau ketika seorang pengguna keluar (menutup tab/browser).

B. Implementasi Teknis:

- Saat klien berhasil terhubung dan mengirim event join, server akan menyiarkan event status dengan pesan "{username} telah bergabung."
- Saat koneksi klien terputus (disconnect), server secara otomatis mendeteksi hal ini dan menyiarkan event status lain dengan pesan "{username} telah keluar."

5. Antarmuka Berbasis Web (Web-Based Interface)

A. **Deskripsi:** Seluruh aplikasi dapat diakses dengan mudah melalui browser web standar (seperti Chrome, Firefox, Edge) tanpa perlu menginstal program atau aplikasi khusus di komputer pengguna.

B. Implementasi Teknis:

- **Flask** berperan sebagai web server yang menyajikan file HTML (login.html dan chat.html).
- Tampilan dan nuansa aplikasi dibuat menggunakan **HTML** untuk struktur, **CSS** untuk styling, dan **JavaScript** untuk menangani semua interaktivitas di sisi klien, terutama komunikasi dengan server Socket.IO.

PERANCANGAN DAN IMPLEMENTASI

1. Arsitektur Sistem

Sistem terdiri dari tiga komponen utama:

- **Klien (Browser):** Menampilkan antarmuka login dan chat (login.html, chat.html). Logika JavaScript pada klien menangani koneksi WebSocket, pengiriman pesan, dan penampilan pesan yang diterima.
- **Web Server (Flask):** Menangani *routing* HTTP untuk menyajikan halaman HTML. Flask juga mengelola sesi untuk menyimpan username pengguna.
- **Server WebSocket (Flask-SocketIO):** Menangani semua komunikasi *real-time*. Server mendengarkan *event* dari klien (misalnya, join, message) dan menyiarkan pesan ke semua klien yang terhubung.

2. Struktur Proyek

Proyek ini disusun dengan struktur standar Flask untuk memisahkan logika aplikasi dan template:

```
/proyek-chat/
|-- app.py # Logika utama server Flask & SocketIO
|-- templates/
|       |-- login.html # Halaman untuk memasukkan username
|       |-- chat.html # Halaman utama aplikasi chat
|-- static|-- style.css # File styling css halaman client
```

3. Implementasi Backend

Kode backend bertanggung jawab untuk:

- **Routing:** / untuk halaman login dan /chat untuk halaman utama.
- **Manajemen Sesi:** session['username'] digunakan untuk menyimpan nama pengguna.
- **Event Handling SocketIO:**
 - @socketio.on('join'): Dijalankan saat klien berhasil terhubung ke halaman chat, lalu menyiarkan status "bergabung".
 - @socketio.on('message'): Menerima pesan dari satu klien, lalu menyiarkannya ke semua klien lain dengan menyertakan nama pengirim.
 - @socketio.on('disconnect'): Dijalankan saat klien menutup koneksi, lalu menyiarkan status "keluar".

4. Implementasi Frontend

- **login.html:** Sebuah form HTML sederhana yang mengirimkan data username ke server melalui metode POST.
- **chat.html:**
 - **HTML:** Menyediakan struktur untuk kotak pesan, daftar pesan, dan form input.
 - **JavaScript:** Bagian terpenting dari klien. Skrip ini melakukan:
 1. Inisialisasi koneksi Socket.IO ke server.
 2. Mengirim *event* join setelah koneksi berhasil.
 3. Menangkap event submit pada form, lalu mengirimkan isi input ke server melalui *event* message.
 4. Mendengarkan *event* response dan status dari server, lalu menambahkan pesan yang diterima ke dalam daftar pesan di layar.
 - **style.css:** file untuk menyimpan kode css.

Hubungannya dengan Network Programming

1. Model Klien-Server

Ini adalah fondasi paling dasar dari network programming. Aplikasi kita secara jelas menerapkan model ini:

- Server: File `app.py` yang berjalan di sebuah mesin bertindak sebagai server. Ia "mendengarkan" pada alamat IP dan port tertentu (misalnya, `127.0.0.1:5000`), menunggu koneksi dan permintaan data dari klien.
- Klien: Browser web (Chrome, Edge, dll.) yang dibuka oleh pengguna bertindak sebagai klien. Ia yang memulai koneksi ke server untuk meminta halaman web dan mengirim serta menerima pesan chat.

2. Penggunaan Protokol Jaringan

Network programming selalu bergantung pada protokol (aturan komunikasi). Aplikasi kita menggunakan beberapa lapisan protokol:

- HTTP (HyperText Transfer Protocol): Digunakan untuk komunikasi awal. Saat Anda membuka `http://127.0.0.1:5000`, browser (klien) mengirim permintaan HTTP ke server Flask untuk mendapatkan file HTML, CSS, dan JavaScript.
- WebSocket: Ini adalah protokol kunci untuk fungsionalitas chat. Setelah halaman dimuat, koneksi WebSocket dibuat. Berbeda dengan HTTP, koneksi ini tetap terbuka dan memungkinkan komunikasi dua arah (bidirectional) secara instan. Server bisa mengirim pesan ke klien kapan saja tanpa harus menunggu permintaan baru dari klien. Ini adalah pola komunikasi jaringan yang lebih canggih.
- TCP/IP: Di lapisan yang lebih rendah, semua komunikasi di atas (HTTP dan WebSocket) berjalan di atas TCP/IP untuk memastikan data terkirim secara andal dan berurutan antara klien dan server.

3. Abstraksi Soket (Socket Abstraction)

Inti dari network programming adalah soket (kombinasi IP address dan port yang menjadi titik akhir komunikasi).

- Pada contoh pertama (menggunakan modul socket), kita memanipulasi soket secara manual.
- Dalam aplikasi Flask-SocketIO ini, kita menggunakan abstraksi tingkat tinggi. Kita tidak lagi menulis kode `socket.bind()` atau `socket.listen()`. Library Flask-SocketIO yang melakukannya untuk kita. Kita cukup bekerja dengan konsep yang lebih mudah seperti "events" (`@socketio.on('message')`) dan "rooms". Ini adalah contoh bagaimana network

programming modern sering kali menggunakan framework untuk menyederhanakan tugas-tugas kompleks.

4. Manajemen Koneksi dan Sesi

Network programming tidak hanya tentang mengirim data, tetapi juga mengelola siapa yang terhubung.

- Server kita melacak semua klien yang aktif.
- Dengan menggunakan session Flask, server dapat mengasosiasikan sebuah koneksi dengan identitas tertentu (sebuah username). Ini memungkinkan server untuk mengetahui "siapa" yang mengirim pesan, sebuah tugas manajemen koneksi yang fundamental.

Secara singkat, aplikasi chat ini adalah studi kasus sempurna yang mendemonstrasikan bagaimana prinsip-prinsip network programming—seperti arsitektur klien-server, protokol komunikasi, dan manajemen koneksi—diterapkan untuk menciptakan aplikasi web yang fungsional dan interaktif.