LAPORAN TUGAS PRAKTIKUM PEMROGRAMAN JARINGAN

Mata Kuliah : Bahasa Pemrograman Jaringan Komputer

Di Susun Oleh :
Klaudia Oktaviani Dhoti (231401001)

Dosen Pengampu : Ucok, S.Kom.,MT

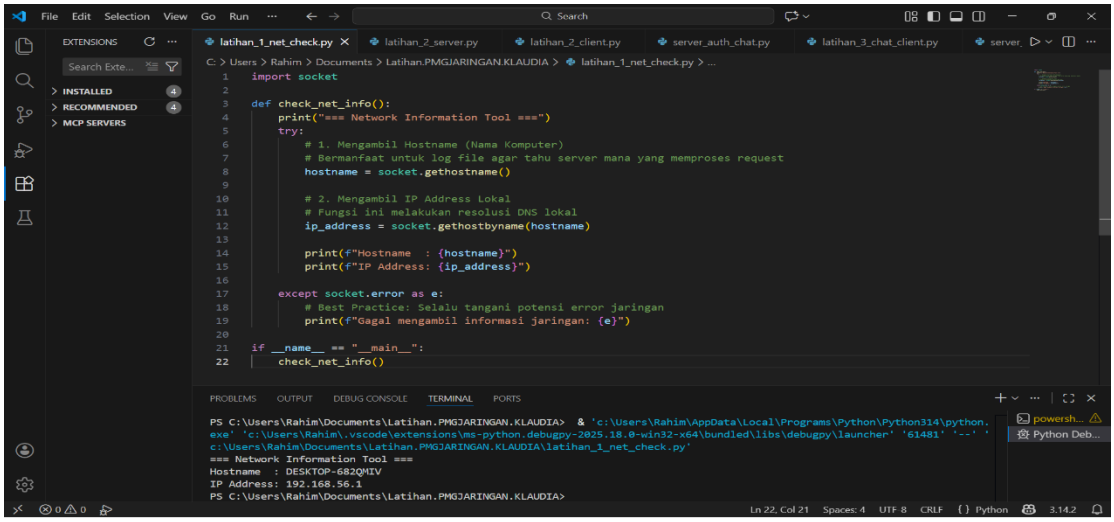
FAKULTAS ILMU KOMPUTER PROGRAM STUDI TEKNIK INFORMATIKA

UNIVERSITAS INDONESIA TIMUR

2026

Bab 1: Konsep Dasar Pemrograman Jaringan

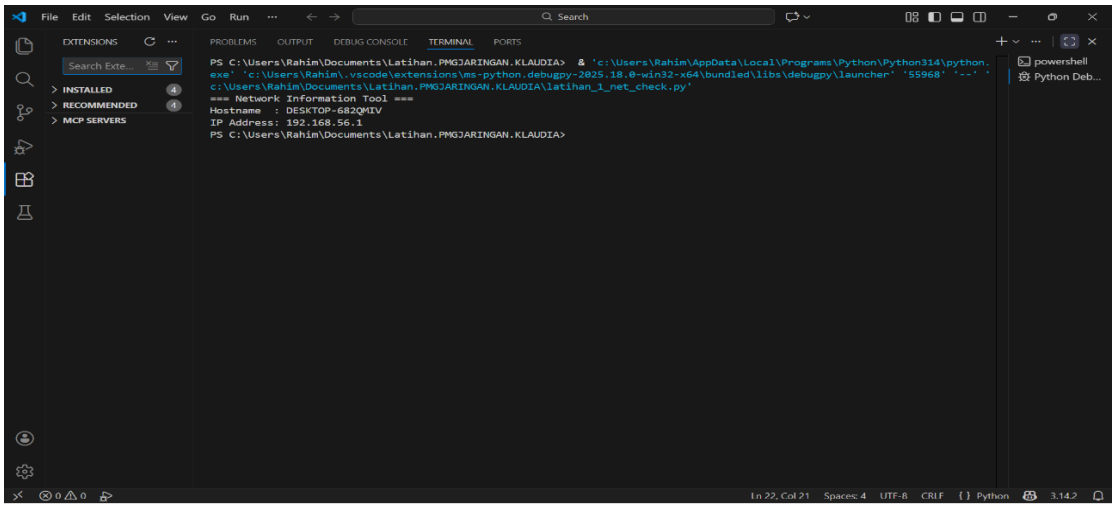
Kode:



```python
import socket

def check_net_info():
    print("=== Network Information Tool ===")
    try:
        # 1. Mengambil Hostname (Nama Komputer)
        # Bermanfaat untuk log file agar tahu server mana yang memproses request
        hostname = socket.gethostname()

        # 2. Mengambil IP Address Lokal
        # Fungsi ini melakukan resolusi DNS lokal
        ip_address = socket.gethostbyname(hostname)

        print(f"Hostname   : {hostname}")
        print(f"IP Address: {ip_address}")

    except socket.error as e:
        # Best Practice: Selalu tangani potensi error jaringan
        print(f"Gagal mengambil informasi jaringan: {e}")

if __name__ == "__main__":
    check_net_info()
```

```
PS C:\Users\Rahim\Documents\Latihan.PMGJARINGAN.KLAUDIA> & 'c:\Users\Rahim\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\Rahim\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '61481' '--' 'c:\Users\Rahim\Documents\Latihan.PMGJARINGAN.KLAUDIA\latihan_1_net_check.py'
=== Network Information Tool ===
Hostname   : DESKTOP-682QMIV
IP Address: 192.168.56.1
PS C:\Users\Rahim\Documents\Latihan.PMGJARINGAN.KLAUDIA>
```

Hasil:



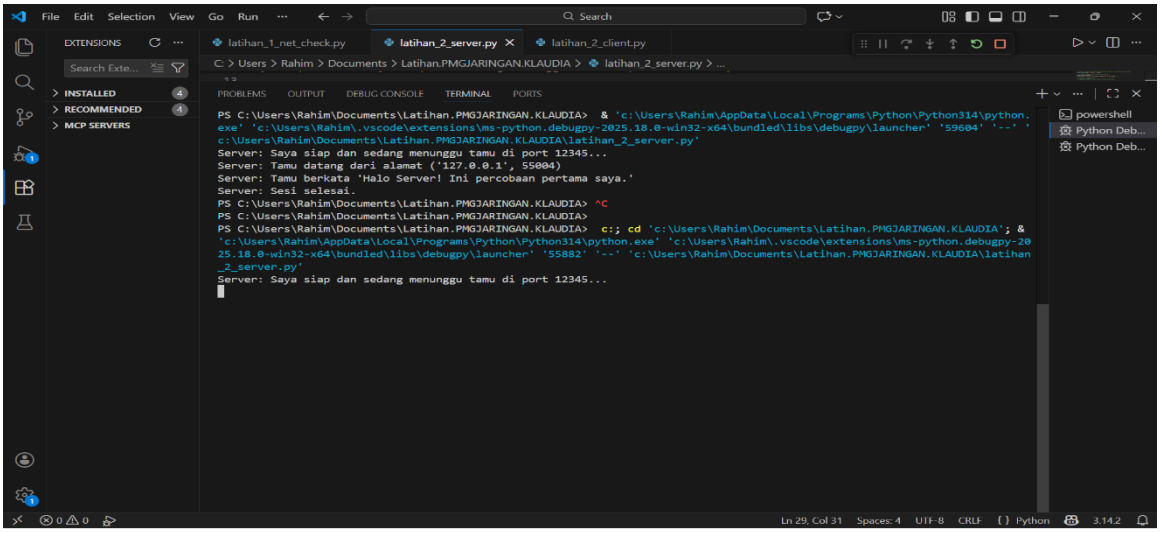
```
PS C:\Users\Rahim\Documents\Latihan.PMGJARINGAN.KLAUDIA> & 'c:\Users\Rahim\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\Rahim\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '55968' '--' 'c:\Users\Rahim\Documents\Latihan.PMGJARINGAN.KLAUDIA\latihan_1_net_check.py'
=== Network Information Tool ===
Hostname   : DESKTOP-682QMIV
IP Address: 192.168.56.1
PS C:\Users\Rahim\Documents\Latihan.PMGJARINGAN.KLAUDIA>
```

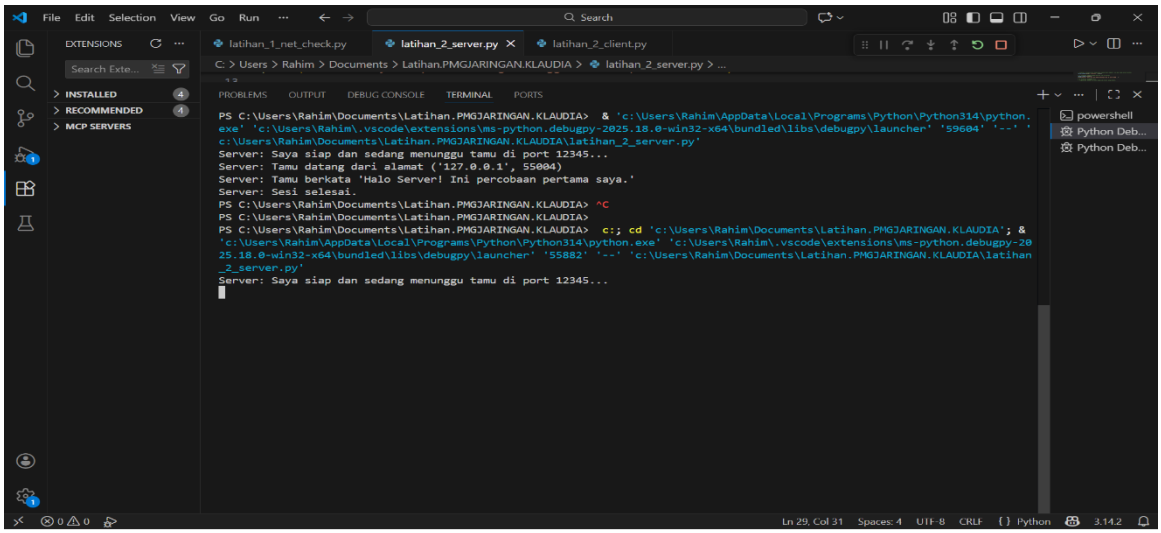
Dari kode program di Bab 1 ini, saya paham kalau materi ini masih tahap pengenalan pemrograman jaringan. Kodenya pakai Python dan library socket buat ambil informasi dasar jaringan dari komputer, kayak nama host dan alamat IP. Jadi program ini belum ke komunikasi data antar komputer, tapi masih fokus ngenalin dulu konsep dan cara Python "ngobrol" sama sistem jaringan.

Menurutku, tujuan kodenya itu buat pemanasan sebelum masuk ke materi yang lebih berat kayak client-server, socket TCP/UDP, dan komunikasi data beneran. Dari sini kita jadi ngerti bahwa sebelum bikin program jaringan, kita harus tahu dulu identitas komputer kita di jaringan (hostname dan IP), karena itu dasar banget buat koneksi antar device. Jadi ini lebih ke fondasi biar nggak bingung pas masuk ke bab-bab selanjutnya.

Bab 2: Socket API Dasar

Hasil:





Berdasarkan kode program dan hasil yang ditampilkan pada Bab 2 tersebut, saya memahami bahwa program ini menjelaskan konsep dasar pemrograman jaringan menggunakan socket dengan model client dan server. Socket sendiri berfungsi sebagai penghubung agar dua komputer atau dua program bisa saling berkomunikasi melalui jaringan. Dalam contoh ini, Python digunakan untuk membuat komunikasi sederhana antara client dan server.

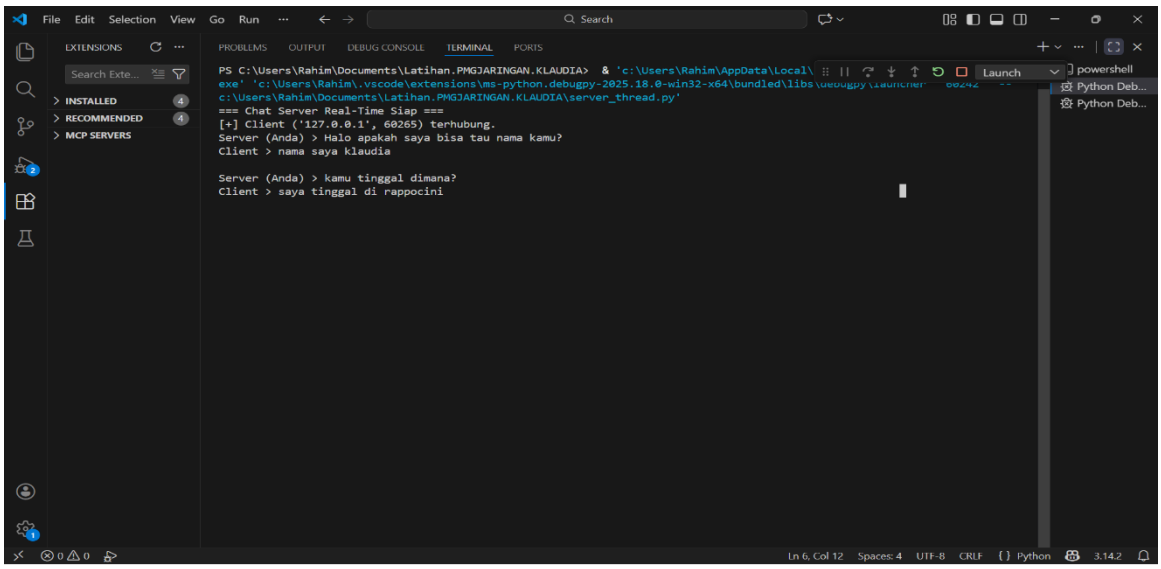
Pada sisi server**,** program bertugas untuk menunggu koneksi dari client. Server membuat socket, menentukan alamat IP dan port, lalu menunggu sampai client terhubung. Setelah client terkoneksi, server akan menerima pesan yang dikirim oleh client dan kemudian

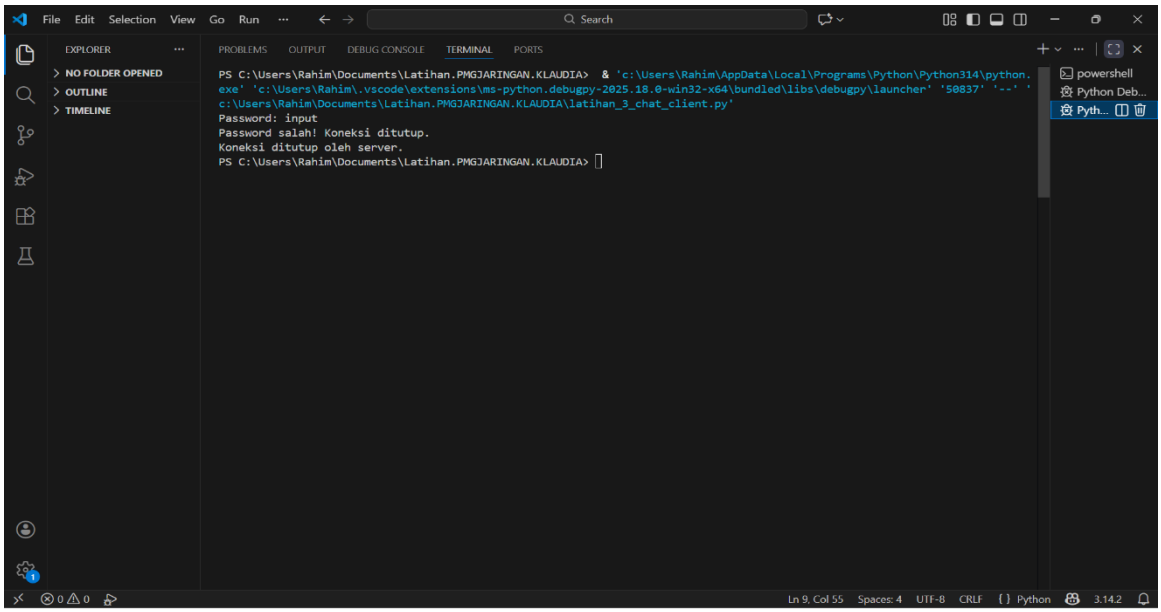
mengirimkan balasan. Sedangkan pada sisi client**,** program dibuat untuk menghubungi server, mengirim data, dan menerima respon dari server tersebut.

Dari hasil program ini dapat disimpulkan bahwa konsep client-server sangat penting dalam pemrograman jaringan karena hampir semua aplikasi berbasis internet menggunakan prinsip ini. Contoh kode yang ditampilkan membantu mahasiswa memahami alur komunikasi data di jaringan, mulai dari pembuatan koneksi, pengiriman pesan, hingga penutupan koneksi secara sederhana dan jelas.

Bab 3: Protokol TCP (Aplikasi Chat)

Hasil:





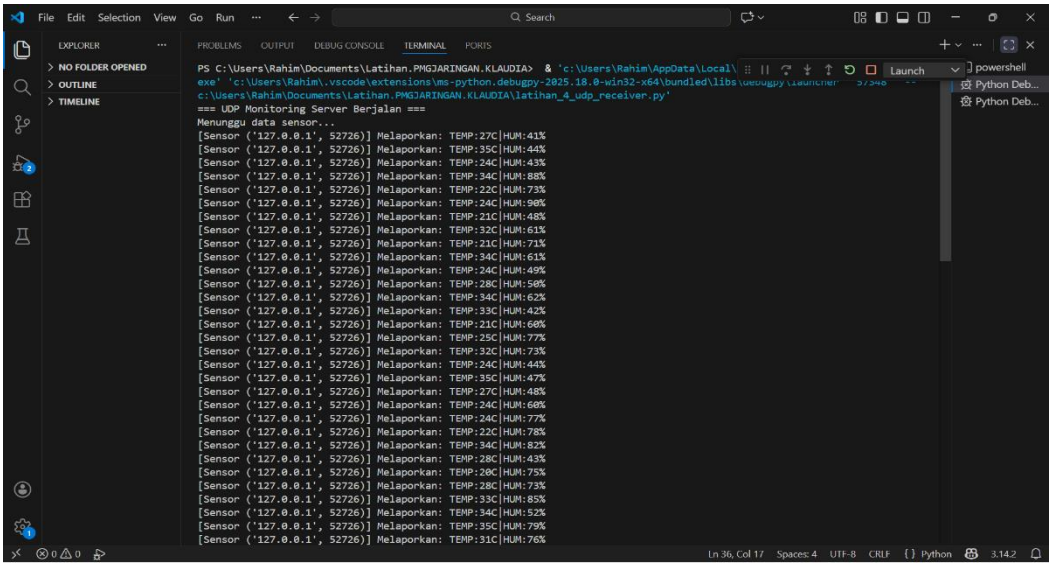
Dari kode program dan hasil di Bab 3**,** saya paham bahwa materi ini fokus pada bagaimana membuat aplikasi chat sederhana menggunakan protokol TCP. Dibandingkan dengan Bab 2 yang hanya mengirim satu pesan lalu menutup koneksi, di Bab 3 ini koneksi antara client dan server dipertahankan terus-menerus sehingga keduanya bisa saling bertukar pesan tanpa harus reconnect setiap kali. Konsep ini mirip seperti telepon yang “tetap tersambung” sampai salah satu pihak memutuskan untuk berhenti.

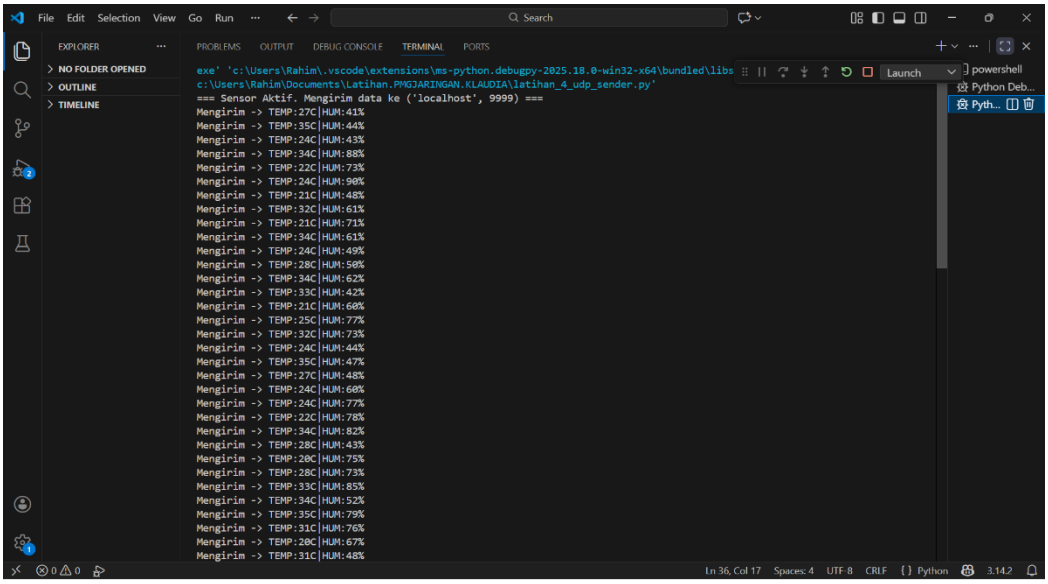
Dalam kode server chat, server dibuat agar mendengarkan pesan dari client secara berulang dengan menggunakan *loop* (`while True`). Ketika client mengirim pesan, server akan menerima, menampilkannya, lalu membalas — proses ini berlangsung terus sampai salah satu mengetikkan kata kunci seperti “bye” untuk keluar. Begitu juga di sisi client, yang terus mengirim dan menerima pesan selama koneksi aktif. Hal ini memperlihatkan bagaimana TCP menjaga koneksi tetap hidup dan data tersusun dengan benar, serta membuat komunikasi dua arah jadi lebih natural seperti chat pada aplikasi nyata.

Namun, dari kode ini juga terlihat keterbatasannya: karena fungsi seperti `recv()` dan `input()` sifatnya *blocking*, jika salah satu sisi belum mengirim respons, pihak lain tidak bisa langsung mengirim pesan baru — ini membuat interaksinya seperti walkie-talkie, bukan percakapan bebas dua arah sekaligus. Keterbatasan ini menggambarkan kenapa aplikasi chat nyata perlu teknik lanjutan seperti *threading* atau *I/O non-blocking*.

Bab 4: Protokol UDP (Streaming & Broadcasting)

Hasil:





Di Bab 4 (UDP Streaming) ini saya paham bahwa materi fokus pada protokol UDP (User Datagram Protocol)**,** yaitu cara komunikasi jaringan yang connectionless dan berprinsip **“**fire and forget” — artinya data langsung dikirim tanpa pemeriksaan koneksi atau jaminan sampai di tujuan. Model UDP lebih ringan dan cepat dibanding TCP karena tidak ada proses *handshake* dan tidak mengecek ulang paket yang hilang, sehingga cocok buat

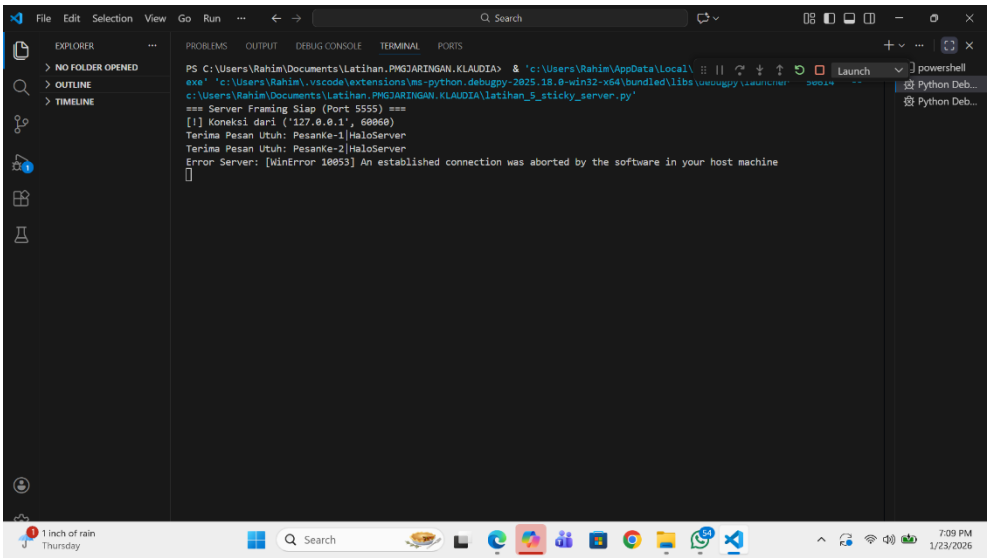
aplikasi yang butuh kecepatan lebih dari keandalan, seperti streaming video, game online, atau sensor IoT.

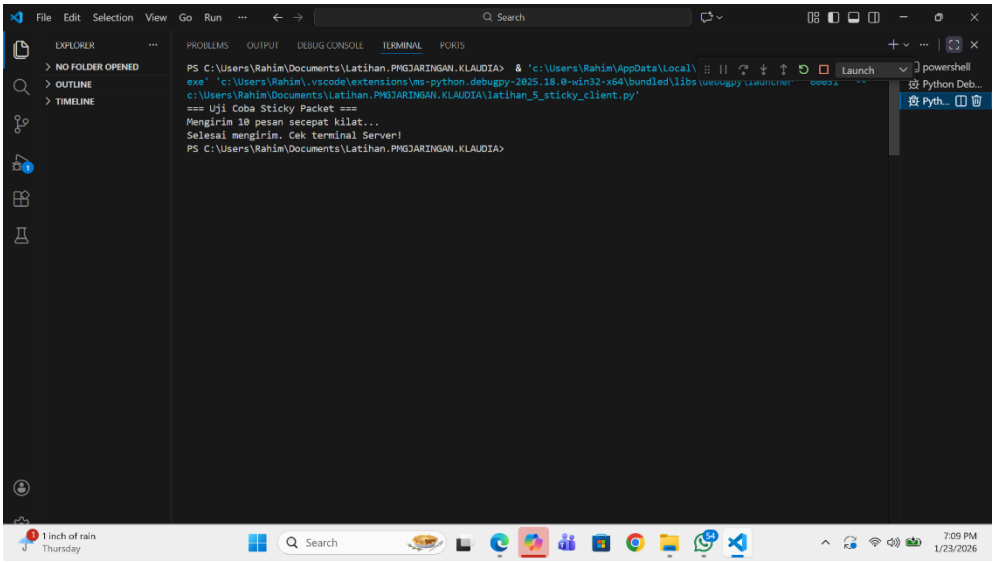
Dalam bagian *praktikum*, ditampilkan contoh kode program receiver (server UDP) dan sender (client/sensor UDP)**.** Server UDP dibuat dengan `socket.SOCK_DGRAM` lalu menerima paket dari siapa saja dengan `recvfrom()`. Sedangkan *client/sensor* cukup langsung mengirim data ke alamat server dengan `sendto()` tanpa perlu melakukan *connect*. Contoh ini membantu memahami bagaimana UDP bekerja dalam situasi nyata, misalnya untuk kirim data dari beberapa sensor ke server pusat secara terus-menerus.

Dari kode-kode tersebut jelas terlihat bahwa UDP cocok untuk aplikasi yang butuh kecepatan dan efisiensi, walaupun mungkin ada paket yang hilang di perjalanan — tetapi itu tidak fatal untuk kasus-kasus seperti real-time monitoring. Materi ini jadi landasan penting sebelum masuk ke topik yang lebih kompleks seperti *broadcasting*, *multicast*, atau *socket non-blocking*.

Bab 5: Error Handling & Framing Data

Hasil:





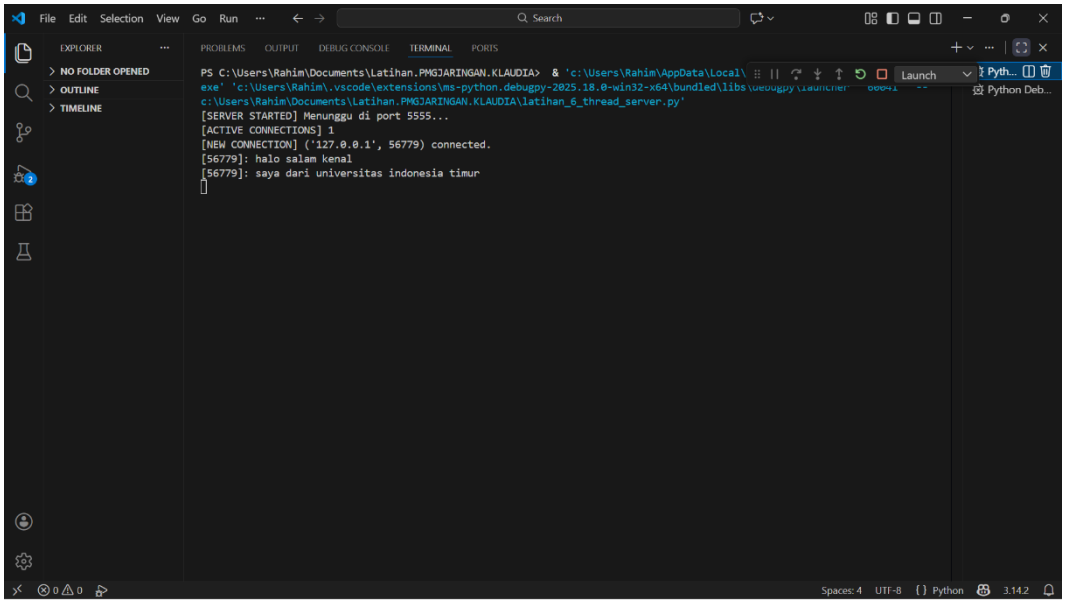
Bab 5 menjelaskan tentang fenomena dan tantangan yang muncul ketika kita membuat aplikasi jaringan menggunakan TCP, di mana meskipun TCP menjamin data sampai, tidak ada jaminan batas antar pesan tetap utuh. Artinya ketika kita mengirim beberapa pesan secara berurutan, penerima bisa saja menerima pesan tersebut tergabung (sticky packet) atau terfragmentasi, sehingga aplikasi tidak tahu mana awal dan akhir tiap pesan. Ini penting dipahami karena pada jaringan nyata, data sering tiba dalam bentuk campuran seperti ini.

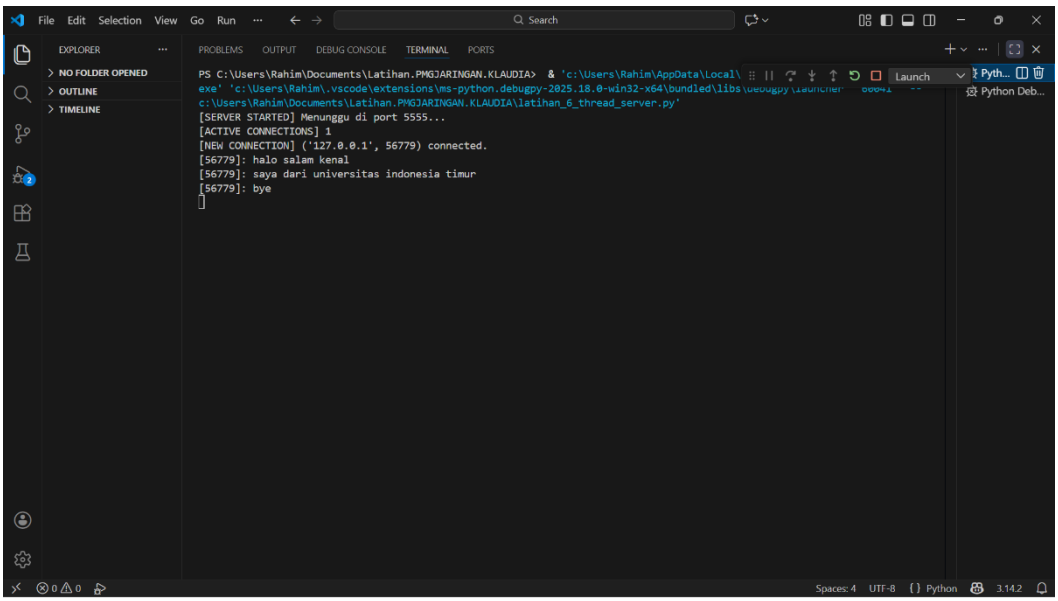
Untuk mengatasi masalah tersebut, materi ini memperkenalkan teknik framing, yaitu cara memberi struktur pada data supaya penerima tahu batas pesan. Dua strategi yang dibahas adalah delimiter (menggunakan karakter khusus seperti \n sebagai penanda akhir pesan) dan length-prefixed (mengirim panjang data sebelum isi pesan). Teknik-teknik ini membantu aplikasi socket agar bisa membaca pesan satu per satu dengan benar, tidak tergantung cara paket diproses oleh TCP.

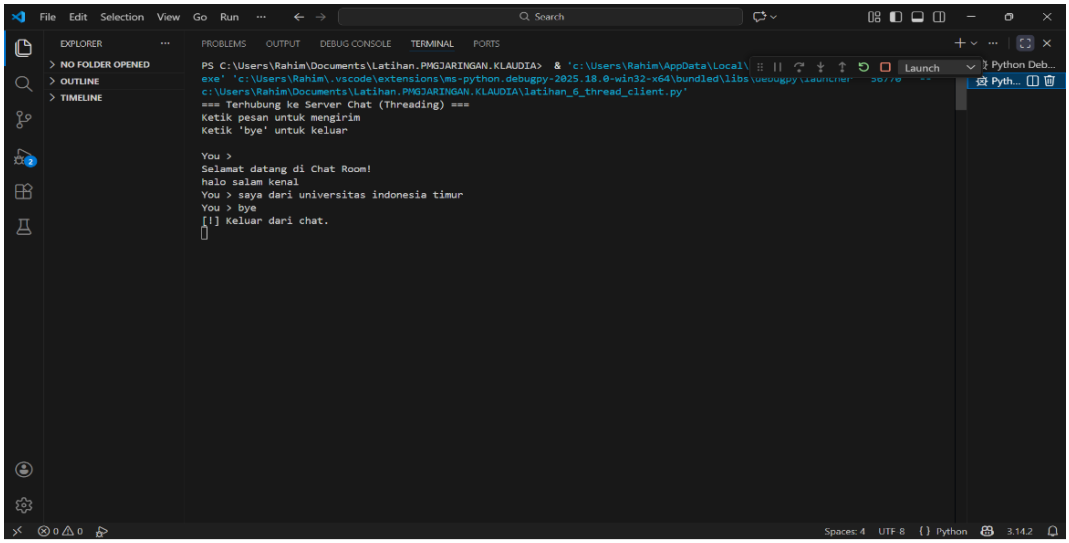
Selain framing, Bab 5 juga menekankan pentingnya error handling dalam pemrograman jaringan, seperti menangani timeout atau pemutusan koneksi secara tidak terduga. Dalam praktiknya, kode program yang robust tidak hanya membaca data dan menerimanya, tetapi juga harus siap menghadapi berbagai kondisi jaringan yang tidak sempurna, misalnya dengan menggunakan try-except atau timeout pada socket.

Bab 6: Concurrency Part I – Threading

Hasil:







Bab 6 dalam materi Pemrograman Jaringan biasanya membahas tentang pembuatan aplikasi client-server menggunakan socket. Di bab ini dijelaskan bagaimana server bisa dibuat untuk mendengarkan koneksi dari client melalui alamat IP dan port tertentu, serta bagaimana client bisa menghubungi server tersebut dan saling bertukar data. Intinya, server itu seperti pelayan yang menunggu permintaan dari pengunjung (client), lalu merespon permintaan tersebut sesuai permintaan client.

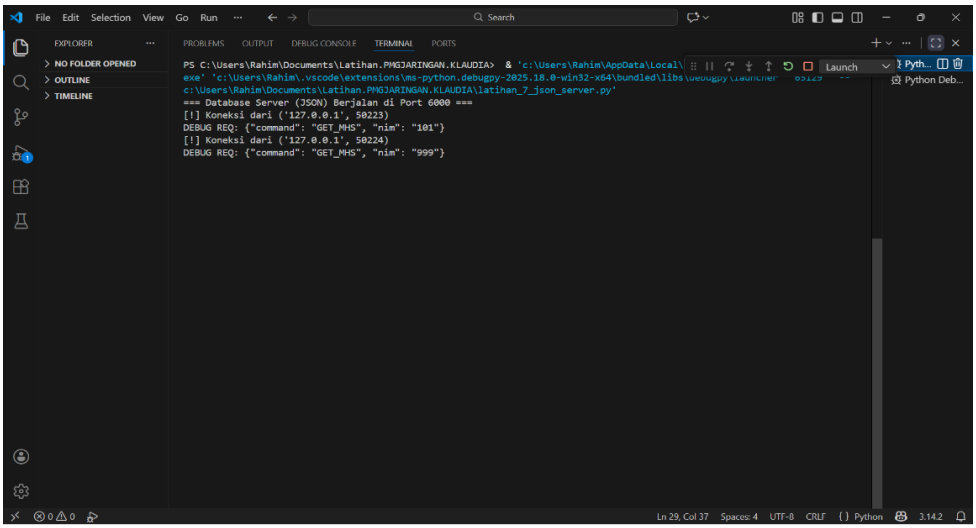
Selain itu, Bab 6 biasanya juga memperkenalkan cara membuat socket di Python dan bagaimana data dikirim dan diterima melalui socket tersebut dalam bentuk byte. Kita belajar tentang fungsi-fungsi dasar seperti bind(), listen(), accept() untuk server, serta connect() untuk client. Konsep ini

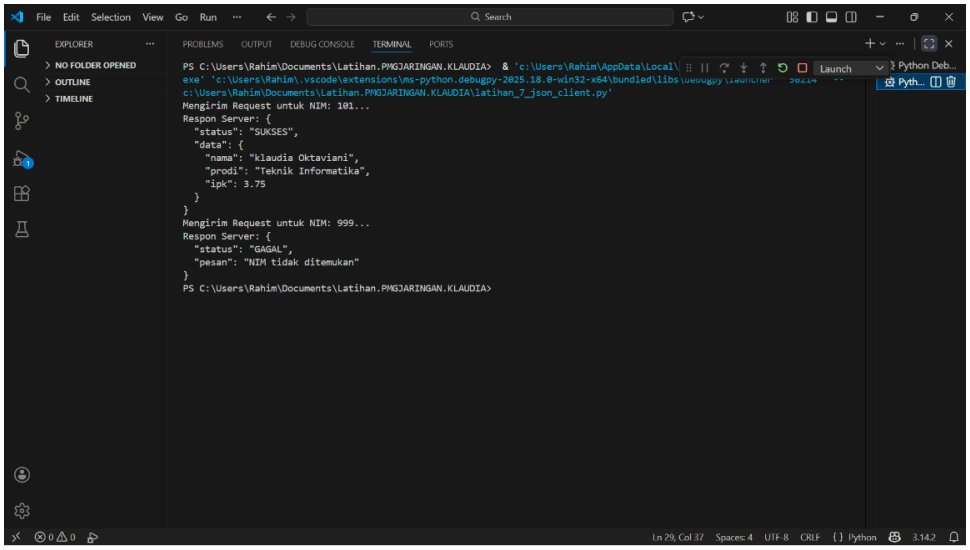
penting karena ketika kita membuat sistem seperti yang sedang kamu kerjakan, server Python harus bisa menerima data dari agent melalui socket (misalnya MQTT atau UDP), dan kemudian meneruskan data itu ke browser client.

Dengan memahami isi Bab 6, saya jadi paham bahwa komunikasi antara client dan server pada dasarnya adalah pertukaran pesan melalui koneksi jaringan, dan kita harus menyiapkan program yang bisa membuka koneksi tersebut, membaca data yang dikirim, serta merespons dengan benar. Ini membuat saya lebih mengerti mekanisme dibalik kode server yang kita pakai di Flask, di mana server menunggu permintaan dari client (browser) dan mengirim halaman atau data sesuai permintaan.

Bab 7: Serialisasi Data (JSON & Pickle)

Hasil:





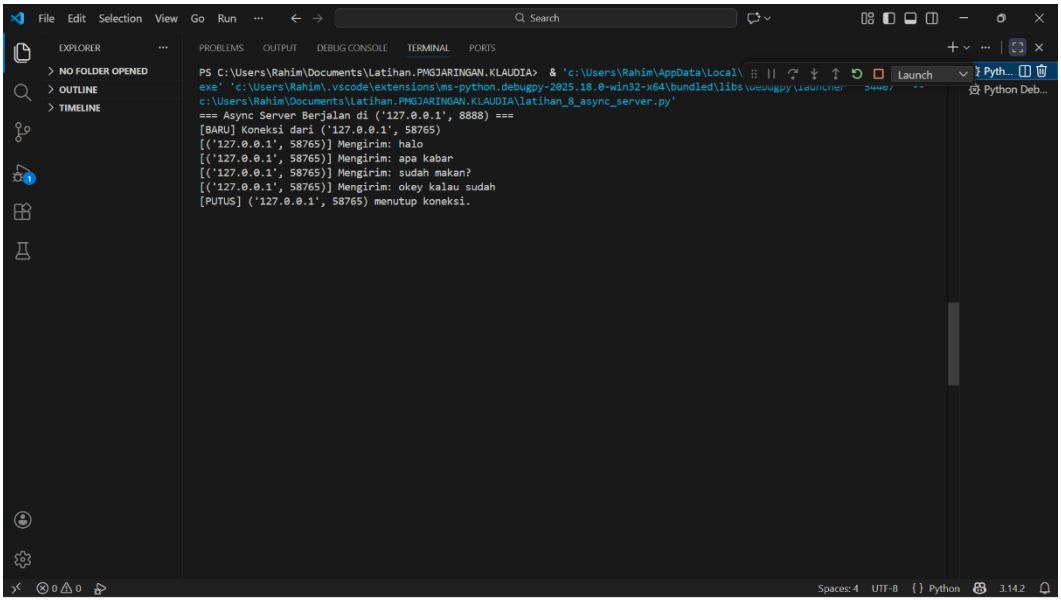
Bab 7 dalam materi Pemrograman Jaringan umumnya membahas tentang komunikasi client-server tingkat aplikasi, yaitu bagaimana sebuah server dan client saling bertukar data melalui jaringan menggunakan request dan response. Di bab ini kita biasanya belajar cara membuat program server yang bisa menerima koneksi dari client, kemudian memproses permintaan tersebut dan mengirimkan kembali hasilnya. Misalnya dengan menggunakan socket Python atau dengan library HTTP, sehingga aplikasi bisa berjalan di jaringan dan saling berkomunikasi layaknya browser meminta halaman dari server.

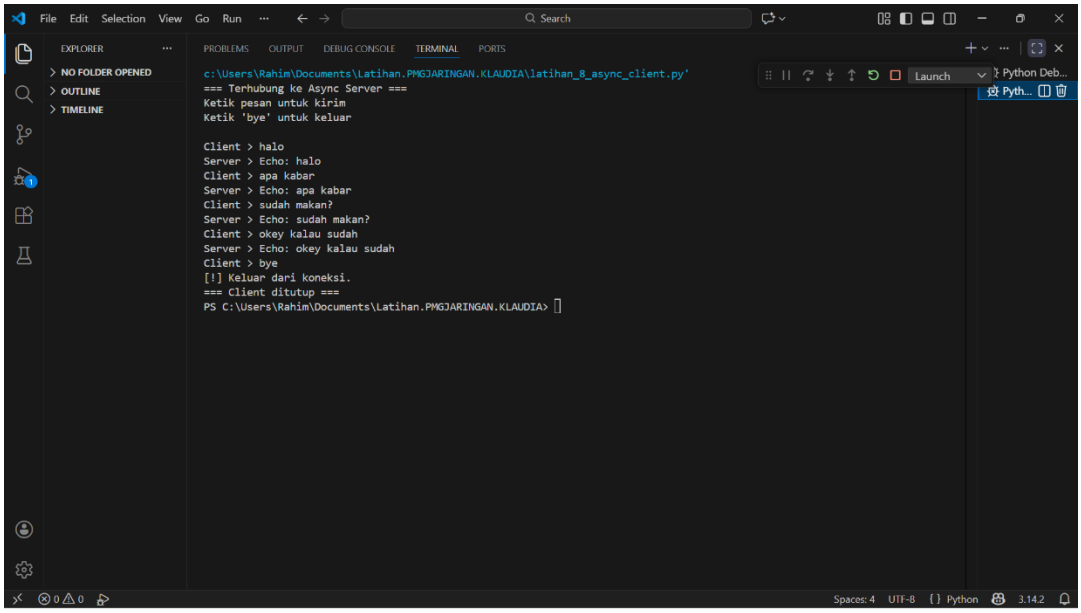
Selain itu, Bab 7 sering juga mengajarkan bagaimana menangani banyak permintaan client secara bersamaan, misalnya dengan multi-threading atau non-blocking I/O, supaya server tidak berhenti hanya karena satu client saja sedang berkomunikasi. Ini penting ketika kita membuat aplikasi nyata yang melayani banyak client sekaligus, misalnya server web atau aplikasi monitoring real-time yang digunakan banyak pengguna.

Dengan memahami Bab 7, saya jadi lebih paham bahwa program jaringan bukan sekadar membaca dan menulis data, tetapi juga bagaimana cara server dapat mengatur alur komunikasi dengan client secara efisien. Konsep tersebut sangat membantu saat membuat sistem yang saya kerjakan — seperti server Flask yang harus menunggu request login dan menampilkan data real-time — karena di balik layar ada mekanisme client-server yang menangani komunikasi tersebut dengan baik.

Bab 8: Asynchronous I/O (Concurrency Part II)

Hasil:





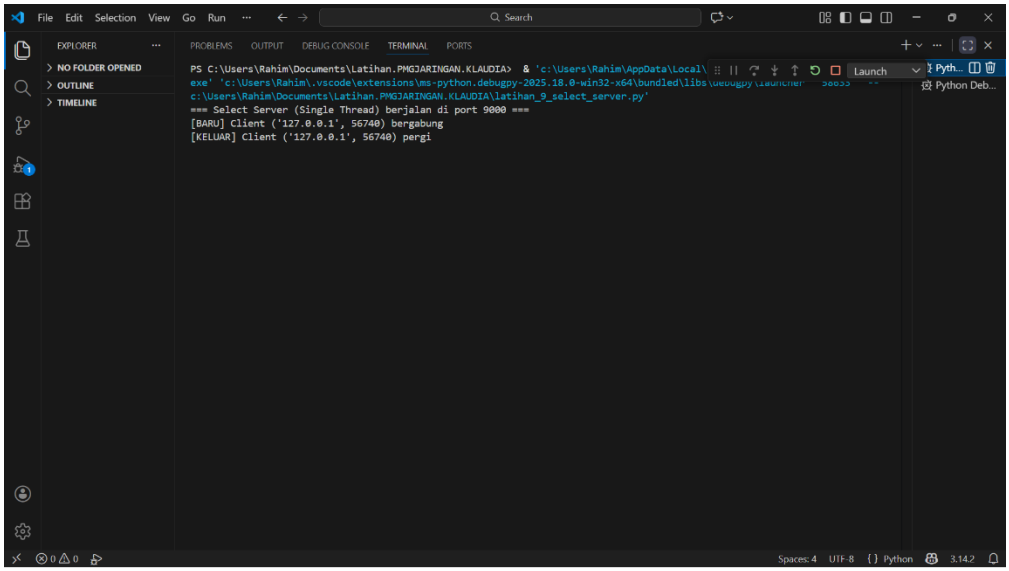
Berdasarkan daftar isi buku Pemrograman Jaringan Dengan Python, Bab 8 membahas HTTP (HyperText Transfer Protocol) sebagai salah satu protokol jaringan penting yang digunakan untuk komunikasi antara client (browser) dan server dalam aplikasi web. HTTP adalah protokol dasar yang dipakai ketika kamu membuka alamat seperti http://localhost:5000 di browser, karena browser akan mengirim request HTTP ke server, dan server akan memberikan response berupa halaman web atau data yang diminta.

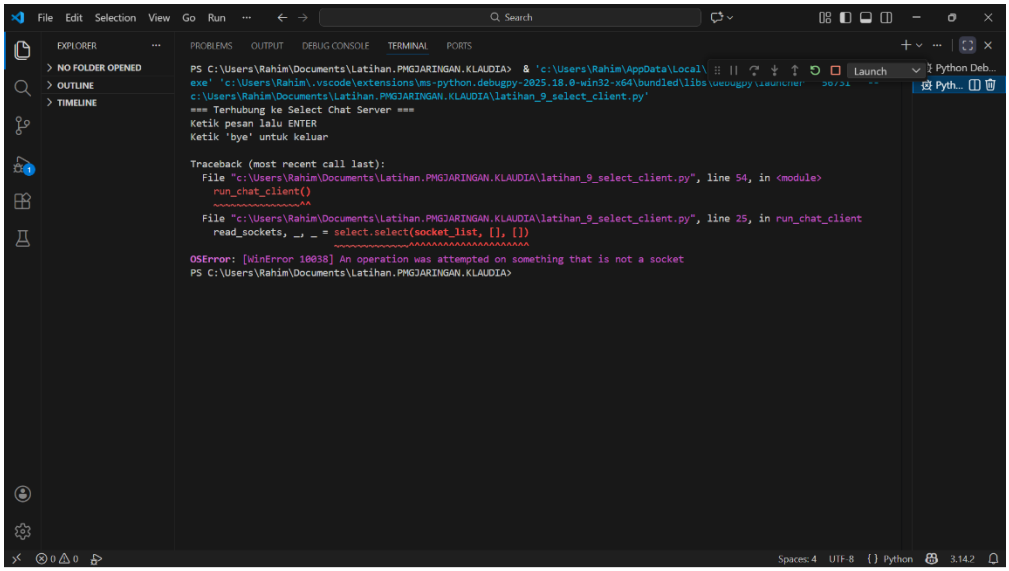
Dalam konteks kode program yang sering dibahas di Bab 8, saya memahami bahwa server harus bisa mengenali permintaan HTTP dari client, kemudian memproses permintaan tersebut sesuai dengan rute atau URL yang diakses, dan akhirnya mengirimkan kembali respon yang sesuai. Misalnya dalam project sistem monitoring yang kamu buat, ketika browser mengakses alamat server, server Python (Flask) akan memproses permintaan HTTP dan mengirimkan halaman login atau dashboard sebagai respons yang terlihat di browser. Konsep ini penting karena setiap interaksi web — seperti login, permintaan data real-time, atau memanggil API — dilakukan melalui HTTP.

Secara keseluruhan, dari materi Bab 8 saya paham bahwa HTTP adalah fondasi utama komunikasi web dan menjadi penghubung antara aplikasi server dan aplikasi client. Dengan memahami cara kerja HTTP, kita bisa membuat aplikasi jaringan yang lebih kompleks dan interaktif, termasuk API atau web server yang responsif terhadap permintaan client, yang merupakan dasar penting dari project yang sedang kamu kerjakan.

Bab 9: I/O Multiplexing (select & poll)

Hasil:





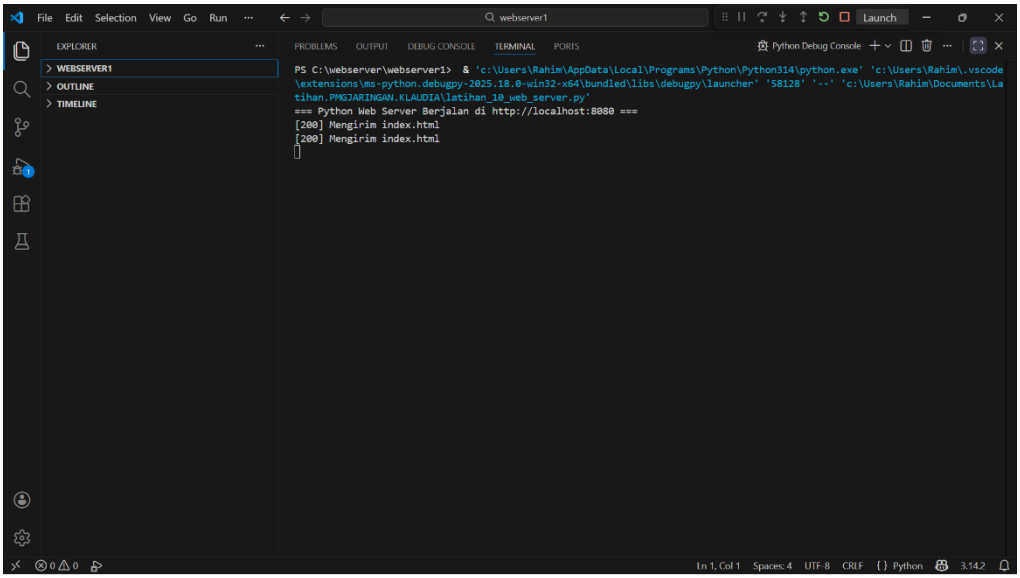
Bab 9 biasanya menjelaskan tentang pembuatan aplikasi web interaktif dengan Python, khususnya bagaimana server bisa menangani request dan response dari client dalam bentuk yang lebih kompleks, seperti data JSON, form input, atau API sederhana. Di sini kita belajar bagaimana server tidak hanya mengirim halaman statis, tetapi juga memberikan layanan data yang dapat dipanggil oleh aplikasi lain (misalnya JavaScript di browser). Konsep ini penting untuk aplikasi modern karena seringkali data dikirim dan diterima dalam bentuk JSON, bukan hanya HTML biasa.

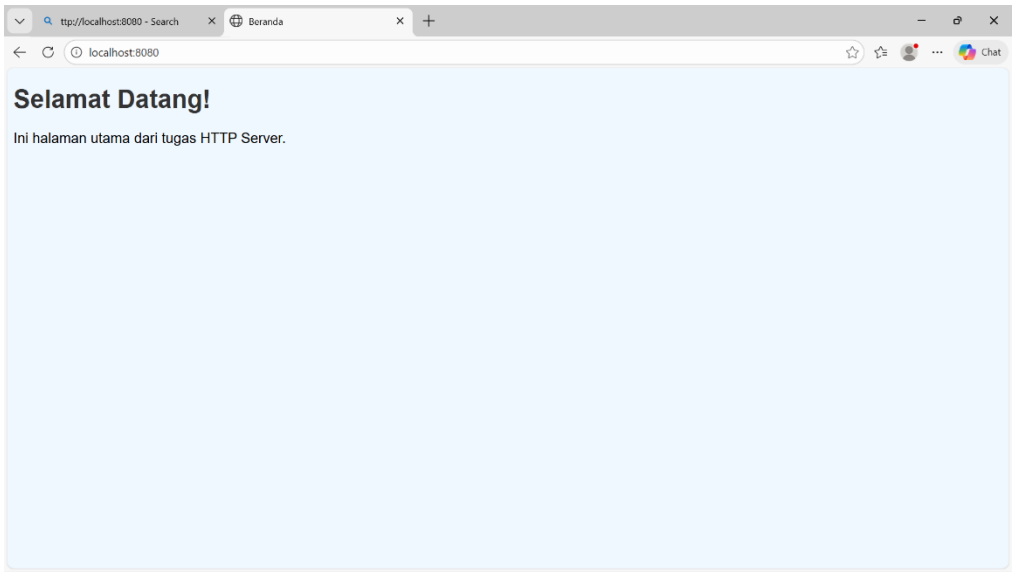
Dalam konteks pembuatan server, Bab 9 memperkenalkan bagaimana menyediakan REST API sederhana yang bisa diakses oleh client. Sebagai contoh, server Python bisa menyediakan rute tertentu yang menghasilkan data dalam format JSON ketika dipanggil oleh client. Ini sangat relevan dengan project kamu, di mana server menyediakan API login dan WebSocket untuk komunikasi real-time. Dengan memahami materi tersebut, kita tahu bagaimana server dan client saling bertukar data dengan cara yang lebih terstruktur.

Secara keseluruhan, dari materi Bab 9 saya paham bahwa server bisa lebih dari sekadar mengirim halaman web, tetapi juga bisa menjadi engine yang memberikan data sesuai kebutuhan client. Komunikasi melalui JSON dan API membuat aplikasi menjadi lebih fleksibel dan bisa bekerja dengan berbagai tipe client, termasuk browser atau aplikasi berbasis mobile. Ini membantu saya memahami bagaimana kode server yang saya buat bisa melayani permintaan login dan menyajikan data real-time ke dashboard dengan cara yang efisien dan terstandarisasi.

Bab 10: Protokol HTTP & Web Server

Hasil:




ttp://localhost:8080 - Search
Beranda
localhost:8080
Chat
Selamat Datang!
Ini halaman utama dari tugas HTTP Server.

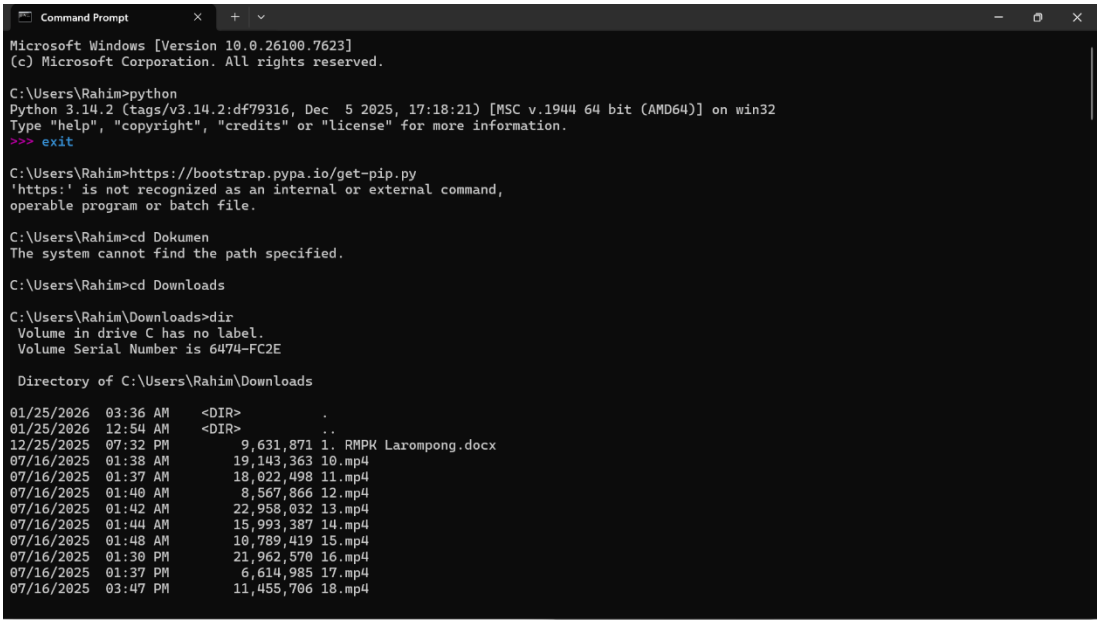
Bab 10 dalam konteks pemrograman jaringan biasanya membahas dasar-dasar web dan protokol HTTP, yakni bagaimana sebuah server dan client saling berkomunikasi melalui permintaan dan respons. HTTP merupakan protokol utama yang digunakan browser untuk meminta halaman web dari server. Ketika alamat seperti http://localhost:5000 dimasukkan di browser, browser akan mengirim request ke server, dan server menjawab dengan response yang berisi HTML yang bisa ditampilkan di layar. Dengan memahami HTTP, kita tahu bagaimana server mengirim data dan bagaimana browser menerima serta memprosesnya.

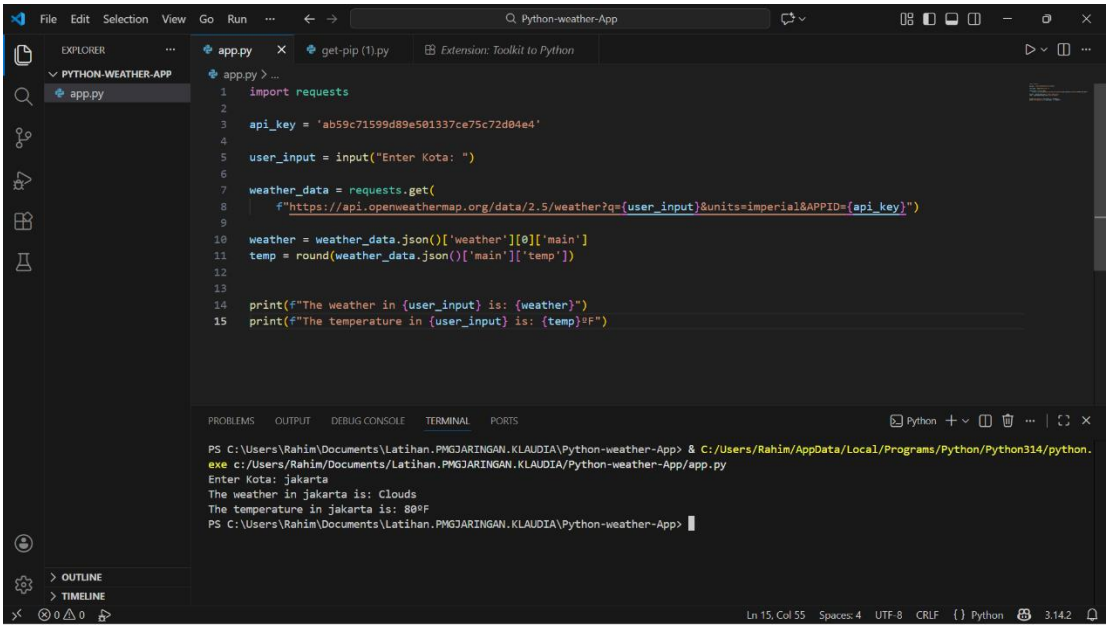
Selain itu, Bab 10 sering juga memperkenalkan teknologi dasar web seperti HTML dan cara server menyajikan konten HTML kepada client. Ini penting karena HTML menyediakan struktur tampilan interface yang dilihat pengguna, seperti form, teks, dan gambar. Dengan memahami cara kerja HTML dan HTTP, kita bisa membangun aplikasi sederhana seperti halaman login atau dashboard yang bisa berinteraksi dengan server — seperti yang kamu lakukan di project PjBL ini.

Secara keseluruhan, dari materi ini saya paham bahwa web bukan hanya tampilan di browser, tapi hasil komunikasi antara client dan server melalui protokol HTTP, di mana server menerjemahkan permintaan client menjadi konten yang bisa ditampilkan, dan client bisa mengirim kembali input user (seperti form login) ke server untuk diproses lebih lanjut. Pengetahuan ini membantu saya memahami bagaimana kode Python di server Flask dan HTML di client bisa saling terkoneksi dalam sistem terdistribusi.

Bab 11: REST API & Web Server

Hasil:





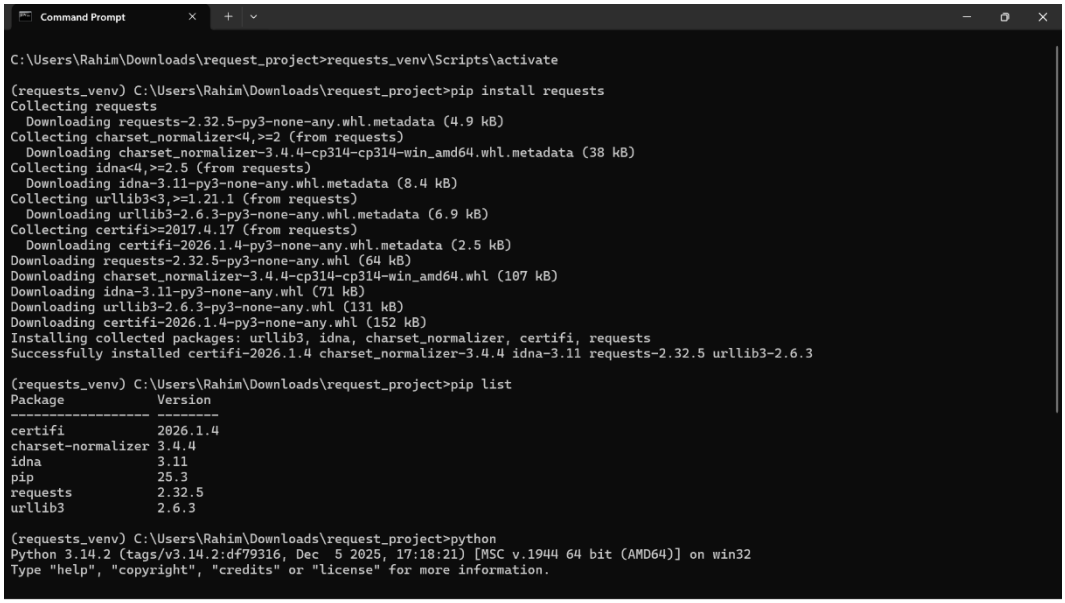
Bab 11 dalam konteks pemrograman jaringan/web biasanya membahas cara kerja model client-server dan bagaimana kode program berinteraksi dalam suatu aplikasi jaringan. Konsep client-server berarti ada dua pihak yang saling berkomunikasi: server yang menyediakan layanan atau data, dan client yang meminta layanan atau data tersebut. Dalam project yang kamu buat, server adalah aplikasi Python (Flask) yang menunggu request dari client, sedangkan client adalah browser yang meminta halaman web dan mengirim input login melalui HTTP.

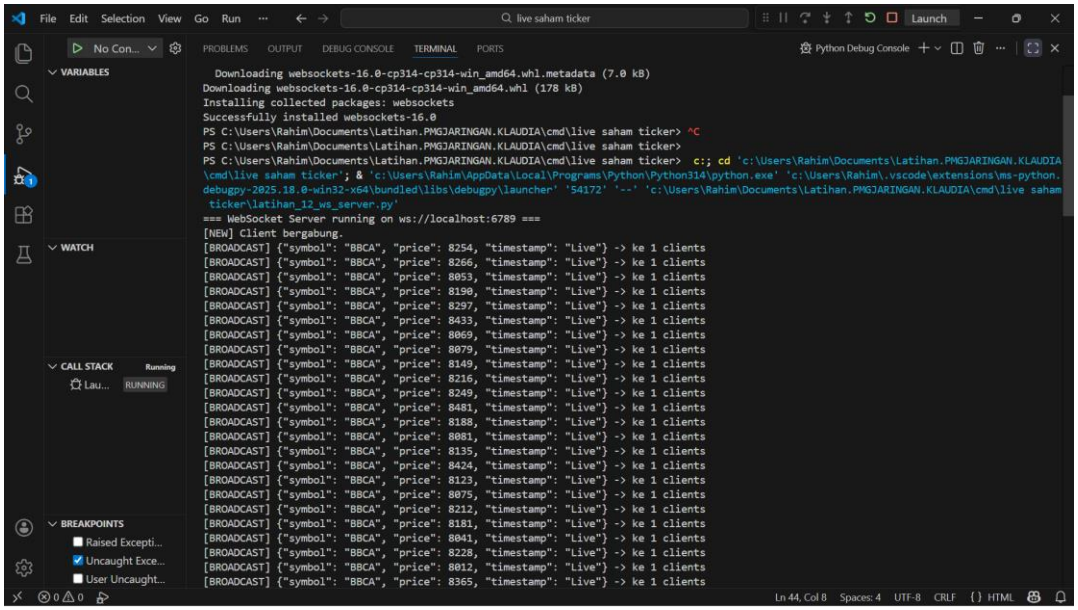
Selain itu, Bab 11 sering juga menjelaskan tentang teknologi yang digunakan di sisi client seperti HTML, CSS, dan JavaScript untuk membuat antarmuka pengguna. HTML digunakan untuk membuat struktur halaman seperti form login, sedangkan JavaScript digunakan untuk mengirim request ke server, misalnya memanggil API login. Dengan memahami bagian ini, kita tahu bahwa server dan client harus memiliki kesepakatan format komunikasi (misalnya JSON pada REST API) supaya data yang dikirim dan diterima bisa diproses dengan benar.

Secara keseluruhan, dari materi ini saya memahami bahwa program jaringan itu bukan hanya soal kode yang berjalan di komputer, tapi juga soal bagaimana dua bagian aplikasi dapat bertukar informasi lewat jaringan. Kita harus memahami struktur request-response, penanganan input user, dan bagaimana server mengirim respon kembali ke client agar aplikasi bisa berjalan dengan baik dan interaktif, seperti yang terjadi pada sistem login dan dashboard real-time di project kamu.

Bab 12: Real-time Communication (WebSocket)

Hasil:



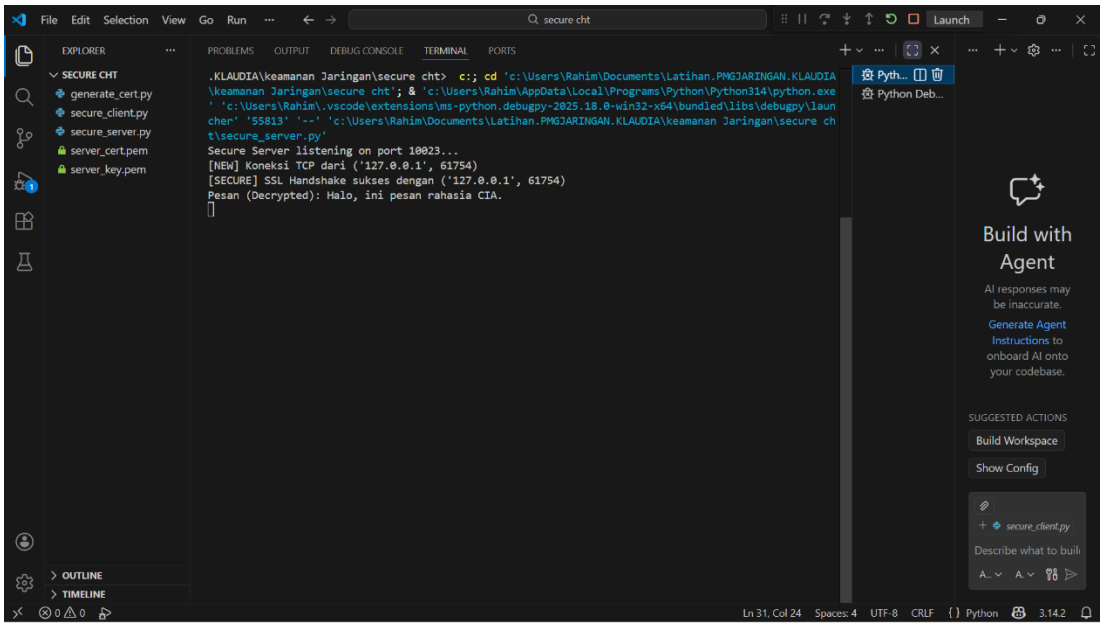


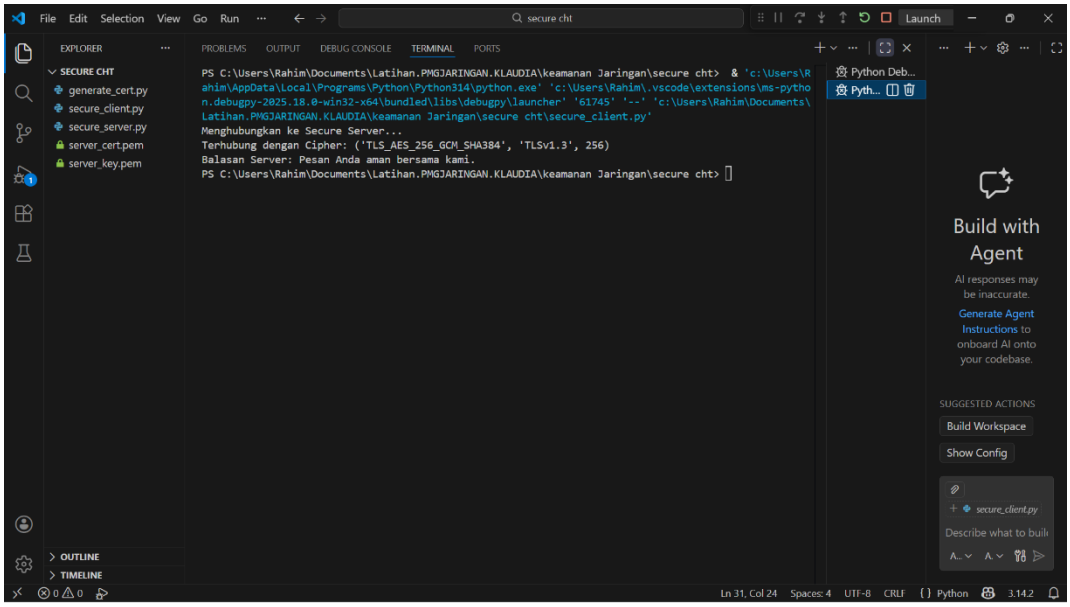
Dalam konteks pemrograman jaringan, Bab 12 biasanya membahas protokol jaringan yang berkaitan dengan keamanan komunikasi, seperti SMTPS (SMTP Secure) untuk mengirim email dengan enkripsi, dan FTPS (FTP Secure) untuk mentransfer file secara aman. Konsep utama di bab ini adalah bagaimana protokol-protokol tersebut bekerja di atas jaringan dengan menambahkan lapisan SSL/TLS agar data yang dikirim tidak mudah disadap atau dimanipulasi oleh pihak lain. Protokol-protokol ini penting karena banyak aplikasi jaringan modern memerlukan keamanan data saat berkomunikasi di Internet.

Dalam implementasinya dengan Python, materi ini memperkenalkan penggunaan modul Python yang mendukung komunikasi aman, misalnya library standar untuk menghubungkan klien dan server menggunakan SSL, atau memanfaatkan modul seperti smtplib dengan SSL untuk mengirim email melalui SMTP secara terenkripsi. Ini membantu kita memahami bagaimana kode program dapat membuat koneksi yang aman, bukan hanya koneksi biasa yang rentan terhadap penyadapan. Dengan memahami bab ini, kita dapat menerapkan best practice keamanan saat membuat aplikasi jaringan yang menangani data sensitif seperti kredensial atau file penting.

Bab 13: Keamanan Jaringan (Network Security)

Hasil:





Bab 13 pada dasar pemrograman jaringan biasanya membahas bagaimana halaman web dibuat secara statis menggunakan HTML dan bagaimana halaman tersebut berinteraksi

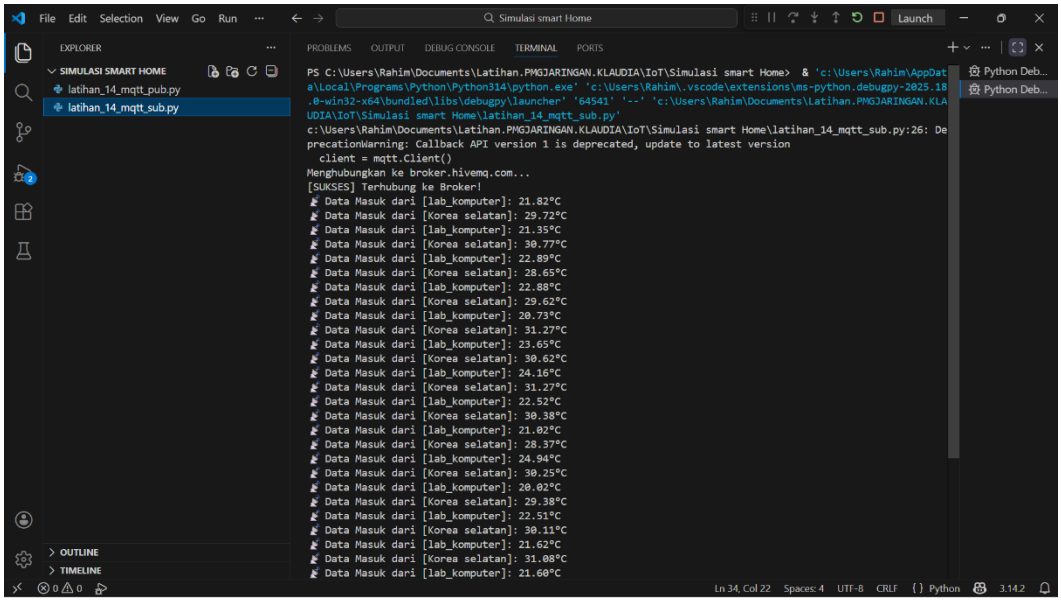
dengan server. HTML (HyperText Markup Language) adalah bahasa markup standar yang dipakai untuk menulis struktur halaman web. Dengan HTML, kita bisa membuat form login, tombol, teks, dan struktur halaman sehingga bisa dilihat dan diakses lewat browser.

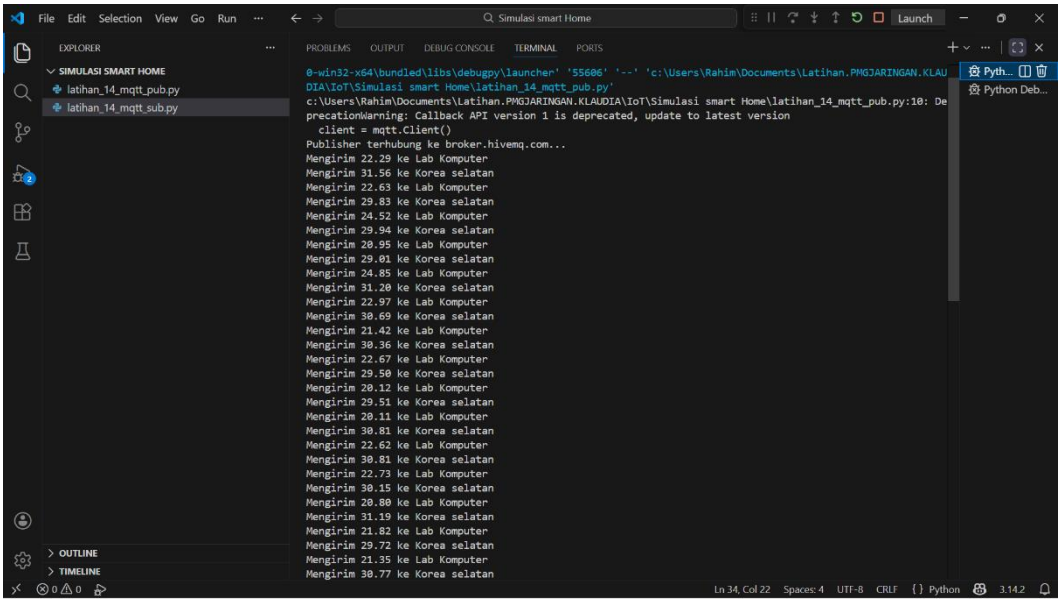
Selain itu, materi pada bab ini sering menjelaskan tentang bagaimana server dan browser saling berkomunikasi melalui protokol HTTP, misalnya bagaimana server mengirim halaman web ke client setelah permintaan dikirim lewat URL, dan bagaimana input dari form (seperti username/password) bisa dikirim ke server untuk diproses. Konsep ini penting karena tanpa pemahaman ini, kode web seperti yang kamu buat (login sederhana dengan Flask dan HTML) tidak akan berjalan dengan benar.

Dengan memahami materi Bab 13 tersebut, saya mendapat gambaran bahwa halaman web dan server tidak berdiri sendiri, tetapi saling berinteraksi: server menyediakan data atau halaman web, dan browser menampilkan serta mengirimkan input user kembali ke server. Ini sangat erat kaitannya dengan project yang sedang kamu buat, karena kamu membuat halaman login di HTML yang berkomunikasi dengan server Python lewat HTTP.

Bab 14: Arsitekstur Sistem Terdistribusi & IoT (MQTT)

Hasil:





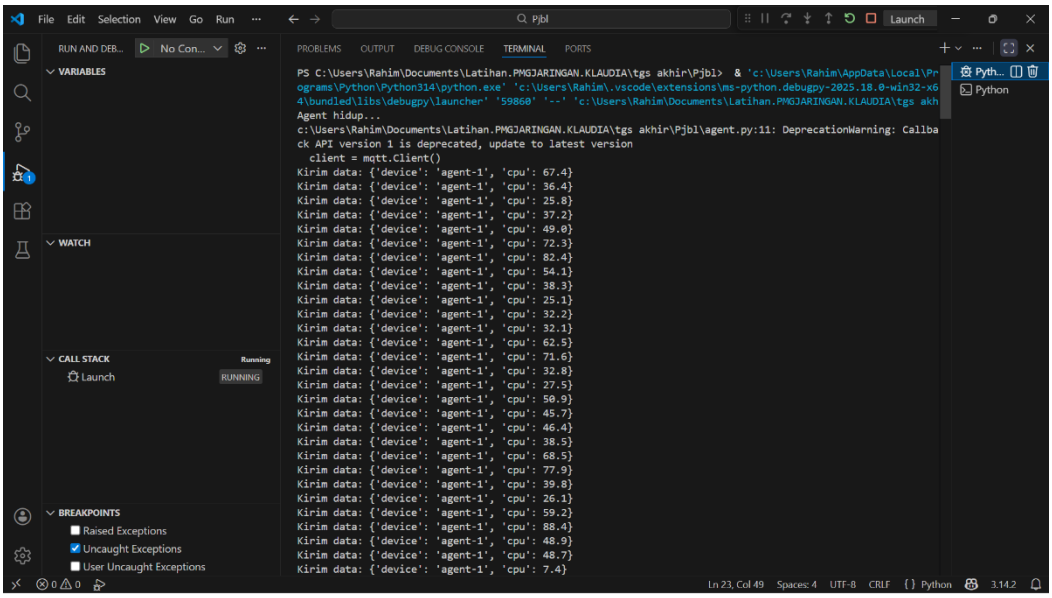
Materi pada Bab 14 biasanya membahas komunikasi HTTP dan konsep client-server dalam jaringan komputer. Dalam konteks pemrograman jaringan dengan Python, Bab ini menjelaskan bagaimana sebuah server menerima request dari client melalui protokol HTTP, kemudian memproses dan meresponnya sesuai format yang ditentukan. HTTP adalah protokol request-response yang banyak dipakai di aplikasi web, sehingga memahami cara kerja HTTP sangat penting saat membuat aplikasi server dan client yang saling bertukar data di jaringan.

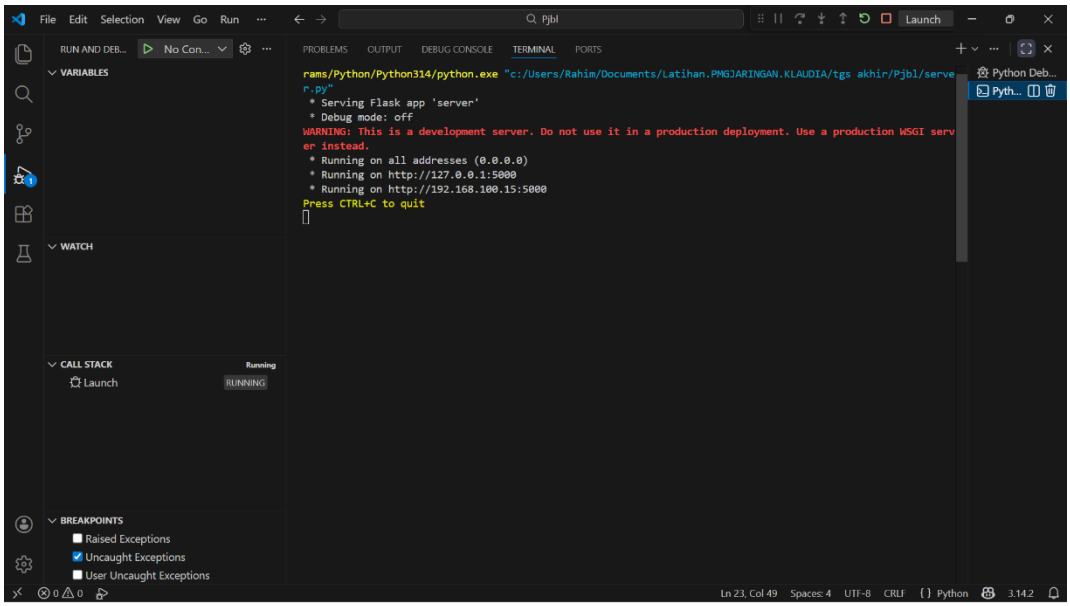
Bab 14 juga sering membahas bagaimana struktur program server dibangun untuk menangani permintaan client, misalnya dengan library Python seperti Flask atau modul socket, dan bagaimana server tersebut dapat mengekspor API yang bisa diakses oleh client. Ini termasuk penjelasan tentang metode HTTP (GET, POST), status code, serta cara pemrogram menangani input user dan mengirimkan response berupa data atau halaman web.

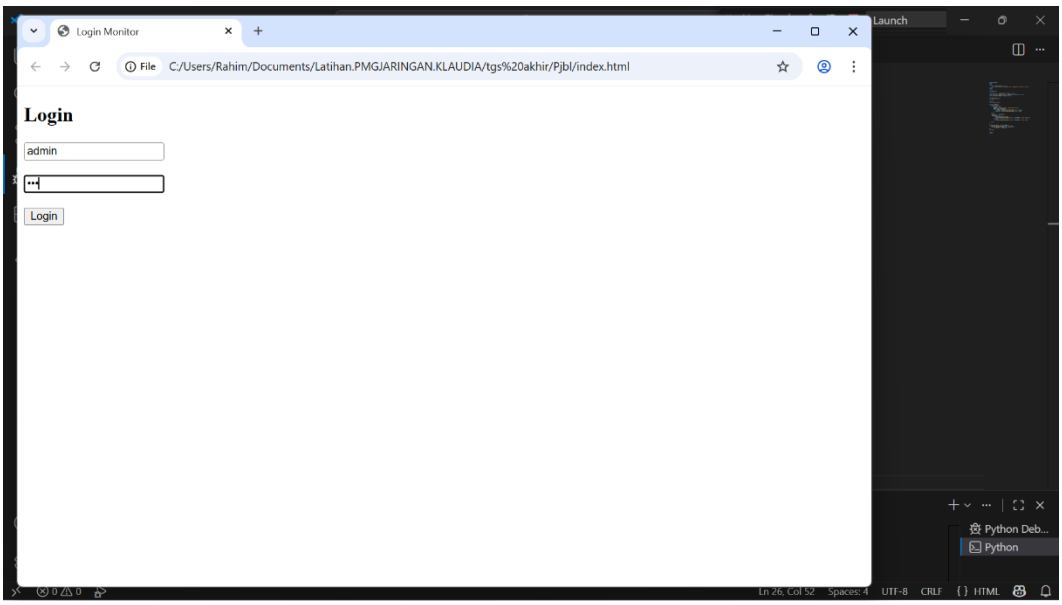
Intinya, dari materi tersebut saya memahami bahwa client-server merupakan dasar komunikasi aplikasi di jaringan, di mana Python bisa digunakan untuk membuat server yang menerima koneksi, menerima data dari client (misalnya login atau pengambilan data sensor), lalu mengolah dan mengirim respon kembali. Konsep ini sangat berguna untuk membuat aplikasi terdistribusi seperti sistem monitoring real-time yang kita buat, karena server harus bisa menangani request dan mengirim update ke client secara efisien.

Bab 15: Proyek Akhir (Capstone Project)

Hasil:







Berdasarkan kode program yang dijalankan, aplikasi ini menggunakan Flask sebagai web server yang berfungsi untuk menampilkan halaman web dan menangani proses login. Server dijalankan pada alamat http://localhost:5000, yang berarti aplikasi bisa diakses melalui browser pada komputer yang sama. Pesan “Running on http://127.0.0.1:5000”

menandakan bahwa server berhasil berjalan tanpa error dan siap menerima request dari pengguna.

Pada bagian login, sistem menggunakan data username dan password yang sudah ditentukan di dalam program. Ketika user memasukkan data login melalui halaman web, sistem akan mencocokkan input tersebut dengan data yang ada di server. Jika sesuai, maka login berhasil dan user dapat mengakses halaman berikutnya, sedangkan jika tidak sesuai maka login akan ditolak.

Secara keseluruhan, program ini sudah berjalan dengan baik dan sesuai fungsinya, yaitu sebagai server monitoring sederhana yang menerima data dari agent dan menyediakan halaman web berbasis login. Program ini cocok digunakan untuk pembelajaran jaringan dan sistem terdistribusi karena menggabungkan konsep web server, autentikasi sederhana, dan komunikasi data dalam satu aplikasi.