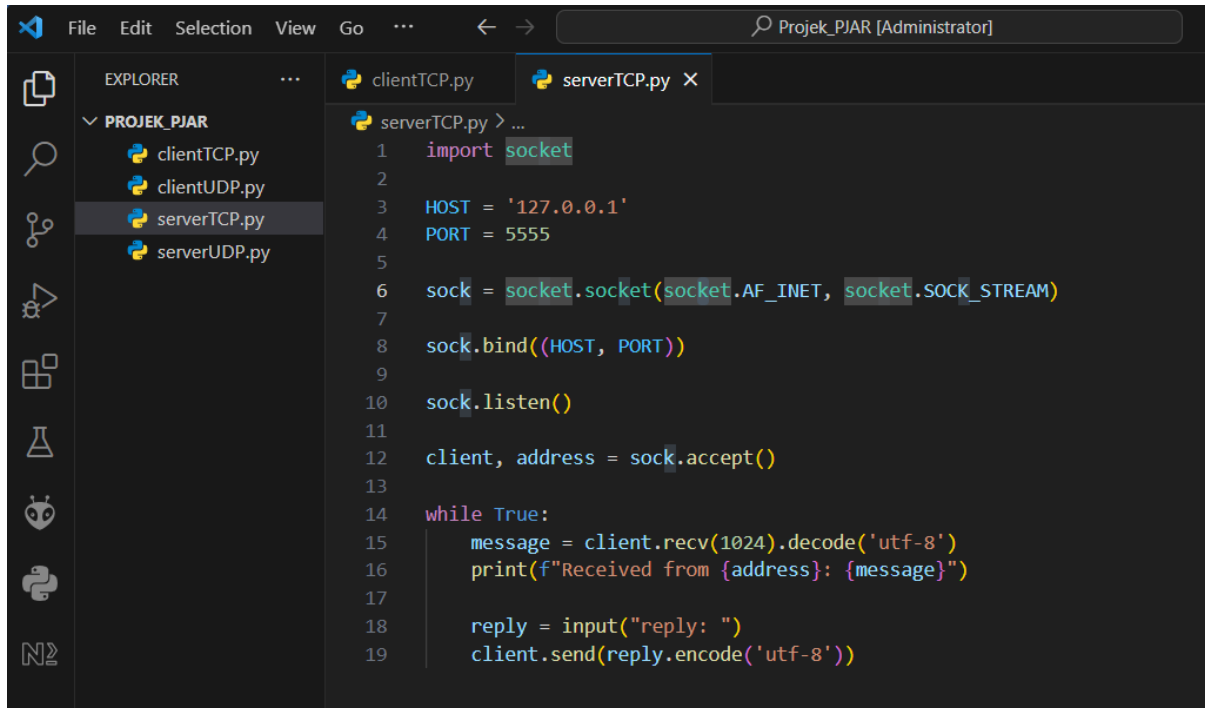


# **TUGAS PROJEK**

Mata Praktikum : Pemrograman Jaringan  
KELAS : 4IA13  
PRAKTIKUM KE : 7  
TANGGAL : 22 Mei 2025  
MATERI : UDP & TCP  
NPM : 51421051  
NAMA : Adji Muhammad Zidane

## serverTCP.py



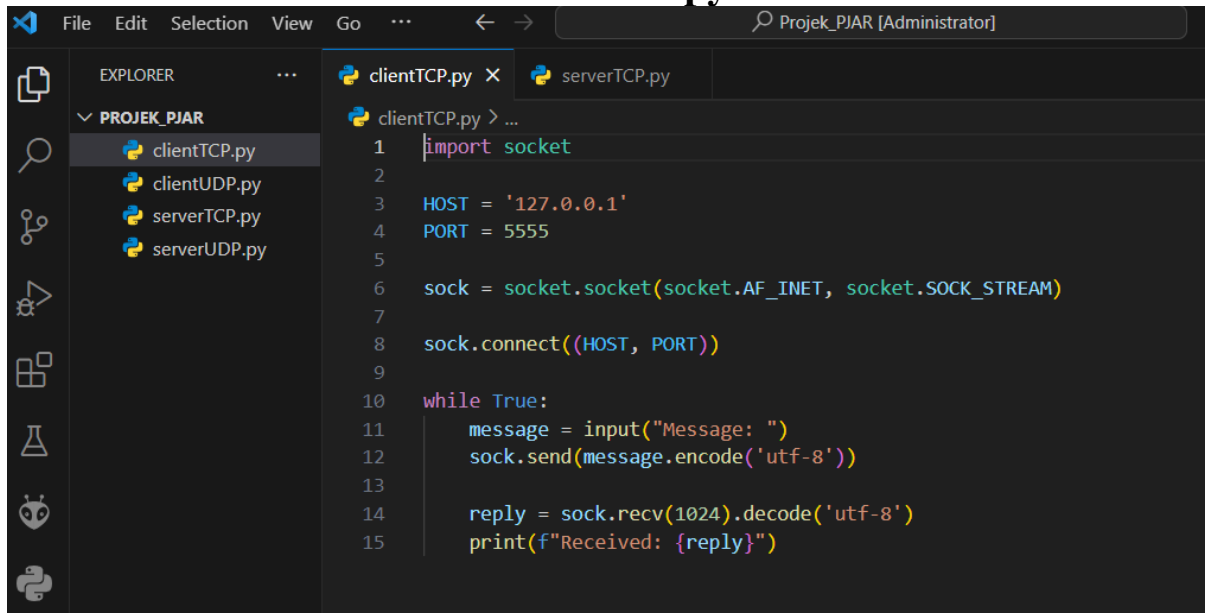
The screenshot shows a code editor with a dark theme. On the left, the 'EXPLORER' sidebar shows a project named 'PROJEK\_PJAR' containing four files: 'clientTCP.py', 'clientUDP.py', 'serverTCP.py' (which is selected and highlighted), and 'serverUDP.py'. The main editor area displays the code for 'serverTCP.py'. The code imports the 'socket' module, sets the host to '127.0.0.1' and the port to 5555, creates a socket object using 'socket.AF\_INET' and 'socket.SOCK\_STREAM', binds it to the host and port, and starts listening. It then enters a 'while True' loop where it accepts incoming connections, receives a message (1024 bytes), decodes it from UTF-8, prints it, prompts the user for a reply, and sends the reply back to the client encoded in UTF-8.

```
1 import socket
2
3 HOST = '127.0.0.1'
4 PORT = 5555
5
6 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7
8 sock.bind((HOST, PORT))
9
10 sock.listen()
11
12 client, address = sock.accept()
13
14 while True:
15     message = client.recv(1024).decode('utf-8')
16     print(f"Received from {address}: {message}")
17
18     reply = input("reply: ")
19     client.send(reply.encode('utf-8'))
```

Gambar di atas adalah implementasi dari server TCP sederhana yang menggunakan modul socket untuk melakukan komunikasi jaringan berbasis protokol TCP/IP. Server dikonfigurasi untuk berjalan di alamat IP lokal (127.0.0.1) dan port 5555. Pertama-tama, server membuat objek socket dengan menggunakan keluarga alamat IPv4 (AF\_INET) dan jenis socket SOCK\_STREAM yang berarti menggunakan protokol TCP. Setelah socket dibuat, server melakukan binding ke alamat dan port yang ditentukan, kemudian mulai mendengarkan koneksi yang masuk. Ketika ada client yang mencoba terhubung, server akan menerima koneksi tersebut dan membuat socket baru khusus untuk komunikasi dengan client tersebut.

Setelah koneksi berhasil, server masuk ke dalam loop tak hingga (while True) untuk terus menunggu dan memproses pesan dari client. Pesan yang diterima dari client dibaca dengan metode recv, kemudian didecode dari bentuk byte ke string menggunakan encoding UTF-8 agar dapat ditampilkan di terminal. Server kemudian menunggu input dari user (admin/server) untuk mengetikkan balasan melalui fungsi input(). Balasan ini kemudian dikirim kembali ke client setelah diubah ke bentuk byte menggunakan metode encode. Dengan demikian, server ini dapat digunakan untuk melakukan komunikasi dua arah secara terus-menerus dengan satu client yang terhubung.

## clientTCP.py

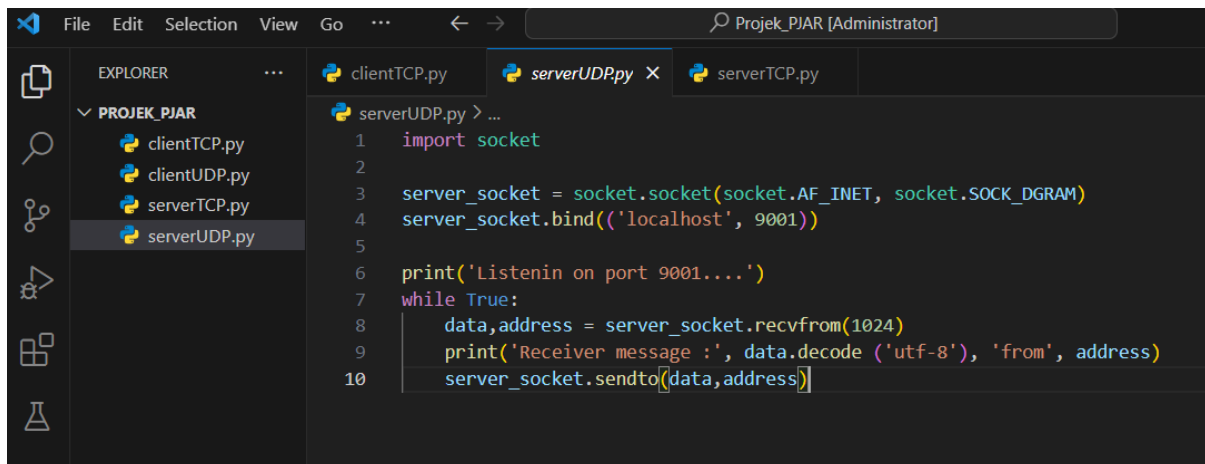


```
1 import socket
2
3 HOST = '127.0.0.1'
4 PORT = 5555
5
6 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7
8 sock.connect((HOST, PORT))
9
10 while True:
11     message = input("Message: ")
12     sock.send(message.encode('utf-8'))
13
14     reply = sock.recv(1024).decode('utf-8')
15     print(f"Received: {reply}")
```

Gambar di atas adalah implementasi dari **client TCP sederhana** yang menggunakan modul socket untuk terhubung ke server melalui jaringan menggunakan protokol TCP/IP. Client dikonfigurasi untuk terhubung ke alamat IP lokal (127.0.0.1) dan port 5555, yang harus sama dengan konfigurasi server agar koneksi dapat berhasil. Pertama-tama, client membuat objek socket dengan tipe AF\_INET untuk IPv4 dan SOCK\_STREAM yang menunjukkan penggunaan protokol TCP. Selanjutnya, client mencoba terhubung ke server menggunakan metode connect dengan memasukkan alamat IP dan port tujuan.

Setelah koneksi berhasil, client masuk ke dalam loop tak hingga (while True) untuk mengirim dan menerima pesan secara terus-menerus. Pada setiap iterasi, program akan meminta input dari user melalui fungsi input() yang kemudian dikirim ke server menggunakan metode send setelah diubah menjadi format byte menggunakan encoding UTF-8. Setelah mengirim pesan, client akan menunggu balasan dari server menggunakan metode recv, yang menerima data dalam ukuran maksimal 1024 byte. Balasan dari server kemudian didekode dari byte ke string dan ditampilkan ke layar. Dengan demikian, client ini memungkinkan komunikasi dua arah secara real-time dengan server yang telah dijalankan sebelumnya.

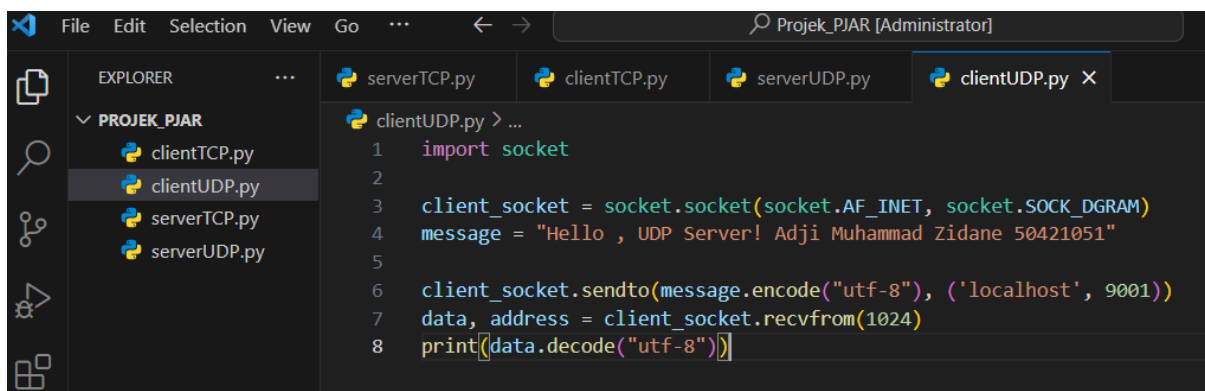
## serverUDP.py



```
1 import socket
2
3 server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
4 server_socket.bind(('localhost', 9001))
5
6 print('Listenin on port 9001....')
7 while True:
8     data,address = server_socket.recvfrom(1024)
9     print('Receiver message :', data.decode('utf-8'), 'from', address)
10    server_socket.sendto(data,address)
```

Server membuat socket dengan tipe SOCK\_DGRAM untuk komunikasi berbasis UDP, lalu mengikatnya ke alamat localhost dan port 9001. Server kemudian masuk ke dalam loop tak hingga untuk terus menerima pesan dari client. Pesan diterima dengan metode recvfrom, yang juga mengembalikan alamat pengirim. Setelah pesan diterima dan ditampilkan, server langsung mengirim kembali pesan yang sama ke client menggunakan sendto, menjadikannya server echo. Tidak seperti TCP, UDP bersifat connectionless sehingga tidak memerlukan proses koneksi awal antar client dan server.

## clientUDP.py



```
1 import socket
2
3 client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
4 message = "Hello , UDP Server! Adji Muhammad Zidane 50421051"
5
6 client_socket.sendto(message.encode("utf-8"), ('localhost', 9001))
7 data, address = client_socket.recvfrom(1024)
8 print(data.decode("utf-8"))
```

Client membuat socket dengan tipe SOCK\_DGRAM untuk komunikasi UDP, kemudian mengirimkan pesan teks ke server yang berada di alamat localhost dan port 9001 menggunakan metode sendto. Setelah itu, client menunggu balasan dari server menggunakan recvfrom, menerima data beserta alamat pengirimnya. Pesan yang diterima kemudian didekode dan ditampilkan. Karena menggunakan UDP, komunikasi ini berlangsung tanpa proses koneksi terlebih dahulu (connectionless), dan cocok untuk pertukaran data ringan dan cepat.