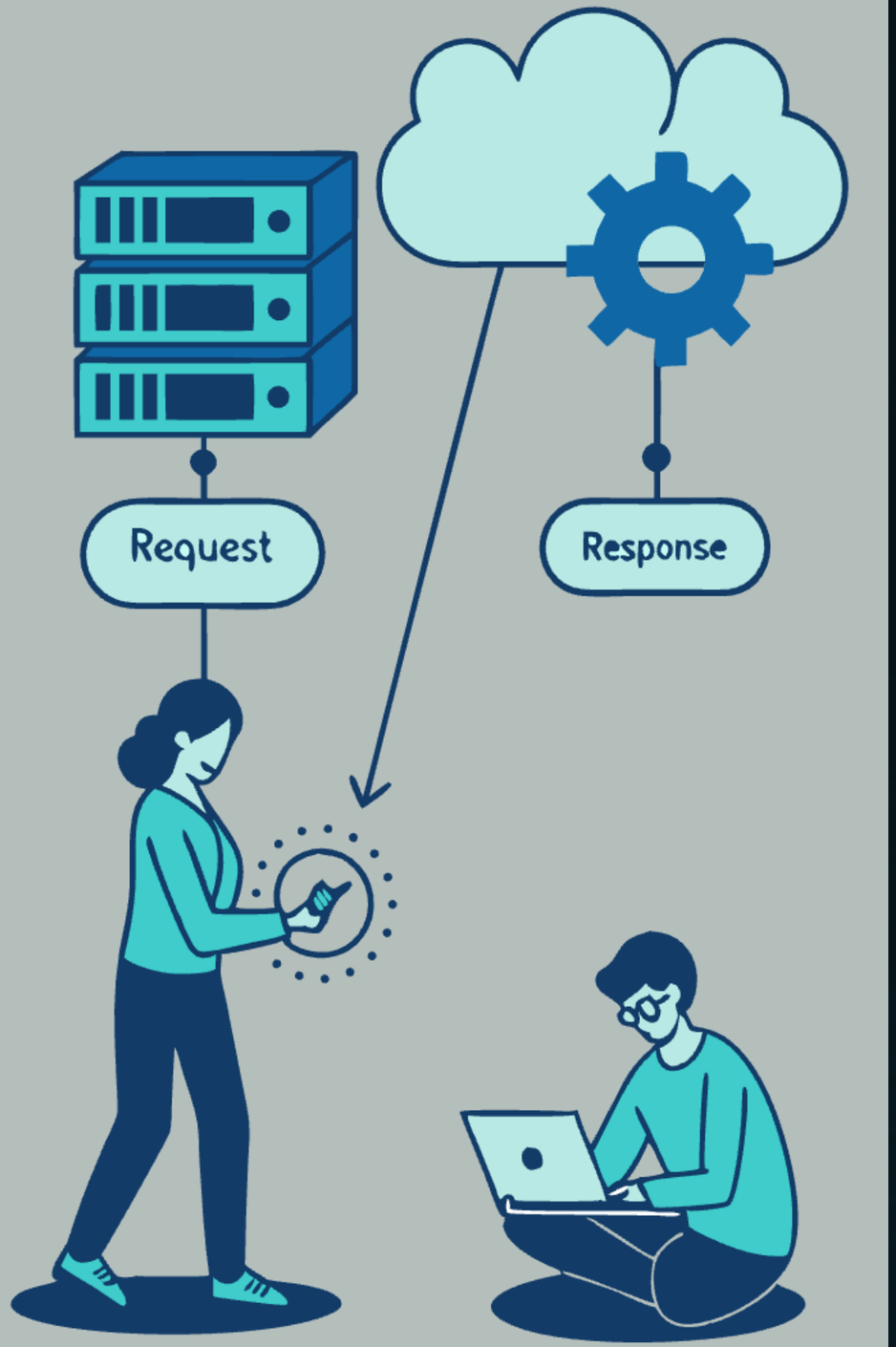
Program Web Server Sederhana Berbasis TCP Socket Programming

Laporan ini membahas pembuatan program web server sederhana berbasis pemrograman socket TCP. Pemrograman socket TCP digunakan untuk membangun aplikasi web yang mengoperasikan protokol TCP/IP, standar komunikasi Internet. Melalui praktik ini, kita dapat memahami konsep dan protokol web dasar, serta meningkatkan keterampilan pemrograman.

Ihab Hasanain Akmal
Faisal Ihsan Santoso
Arie Farchan Fyrzatullah.



Batasan Masalah

Satu Bahasa Pemrograman

Laporan hanya memfokuskan pada pembuatan server web sederhana menggunakan satu bahasa pemrograman saja.

Protokol HTTP 1.1

Hanya membahas implementasi protokol HTTP versi 1.1 untuk permintaan dan respon web.

Lingkungan Lokal

Hanya membahas implementasi server web pada lingkungan pengembangan lokal, bukan pada lingkungan produksi.

Fokus Kesederhanaan

Fokus pada implementasi server web yang sederhana dan mudah dipahami, tanpa fitur kompleks seperti cache atau load balancing.

Tiga File Utama

server_single_thread.py

Aplikasi Server Single ThreadContoh sederhana dari server web yang menggunakan satu thread pemrosesan untuk menangani permintaan.

server_multi_thread.py

Aplikasi Server Banyak ThreadContoh server web yang menggunakan multi-threading untuk menangani permintaan secara paralel.

client_single_thread.py

Aplikasi ClientContoh sederhana dari klien yang berkomunikasi dengan server web yang menggunakan satu thread.

Source Code

server__single__thread.py

server_single.py

```python
import socket
import os
import mimetypes
import time

def handle_client(conn, addr):
    try:
        req = conn.recv(1024).decode()
        print(f"[REQUEST from {addr}]:\n{req}")

        if not req:
            return
        time.sleep(5)

        path = req.split()[1].lstrip('/') or 'home.html'

        if os.path.isfile(path):
            with open(path, 'rb') as f:
                content = f.read()
            ctype = mimetypes.guess_type(path)[0] or 'application/o
            headers = (
                f"HTTP/1.1 200 OK\r\n"
                f"Content-Type: {ctype}\r\n"
                f"Content-Length: {len(content)}\r\n"
                f"Connection: close\r\n\r\n"
            ).encode()
            conn.sendall(headers + content)
        else:
            resp = (
                "HTTP/1.1 404 Not Found\r\n"
                "Content-Type: text/html\r\n"
                "Connection: close\r\n\r\n"
                "<html><body><h1>404 Not Found</h1></body></html>"
            ).encode()
            conn.sendall(resp)
    except Exception as e:
        print(f"[ERROR]: {e}")
    finally:
        conn.close()

def start_server(port=2025):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
        server.bind(('', port))
        server.listen(10)
        print(f"[STARTING] Web Server running on port {port}...")
        try:
            while True:
                conn, addr = server.accept()
                handle_client(conn, addr)
        except KeyboardInterrupt:
            print("\n[SHUTTING DOWN] Server stopped.")

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

Source Code

server_multi_thread.py

server_multi.py

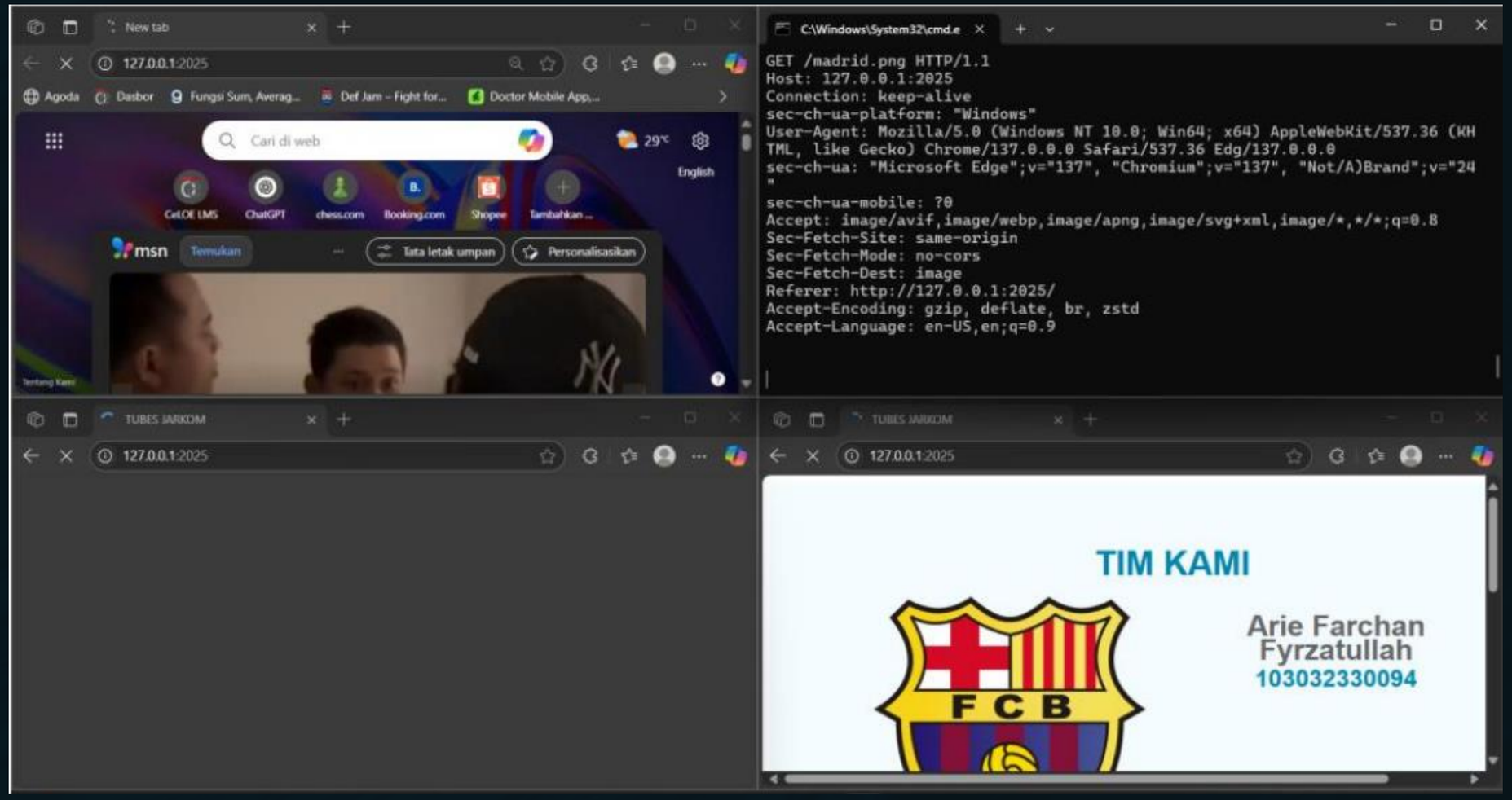
```python
import socket
import threading
import os
import mimetypes
import time

def handle_client(conn, addr):
    try:
        req = conn.recv(1024).decode()
        print(f"[REQUEST from {addr}]:\n{req}")
        if not req:
            return
        time.sleep(5)
        path = req.split()[1].lstrip('/') or 'home.html'

        if os.path.isfile(path):
            with open(path, 'rb') as f:
                content = f.read()
            ctype = mimetypes.guess_type(path)[0] or 'application/octet-strea
            headers = (
                f"HTTP/1.1 200 OK\r\n"
                f"Content-Type: {ctype}\r\n"
                f"Content-Length: {len(content)}\r\n"
                f"Connection: close\r\n\r\n"
            ).encode()
            conn.sendall(headers + content)
        else:
            resp = (
                "HTTP/1.1 404 Not Found\r\n"
                "Content-Type: text/html\r\n"
                "Connection: close\r\n\r\n"
                "<html><body><h1>404 Not Found</h1></body></html>"
                "Content-Type: text/html\r\n"
                "Connection: close\r\n\r\n"
                "<html><body><h1>404 Not Found</h1></body></html>"
            ).encode()
            conn.sendall(resp)
    except Exception as e:
        print(f"[ERROR]: {e}")
    finally:
        conn.close()

def start_server(port=2026):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
        server.bind(('', port))
        server.listen(10)
        print(f"[STARTING] Web Server running on port {port}...")

        try:
            while True:
                conn, addr = server.accept()
                threading.Thread(target=handle_client, args=(conn, addr), daemon=True).start()
                print(f"[ACTIVE CONNECTIONS]: {threading.active_count() - 1}")
        except KeyboardInterrupt:
            print("\n[SHUTTING DOWN] Server stopped.")

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

Source Code client_single_thread.py

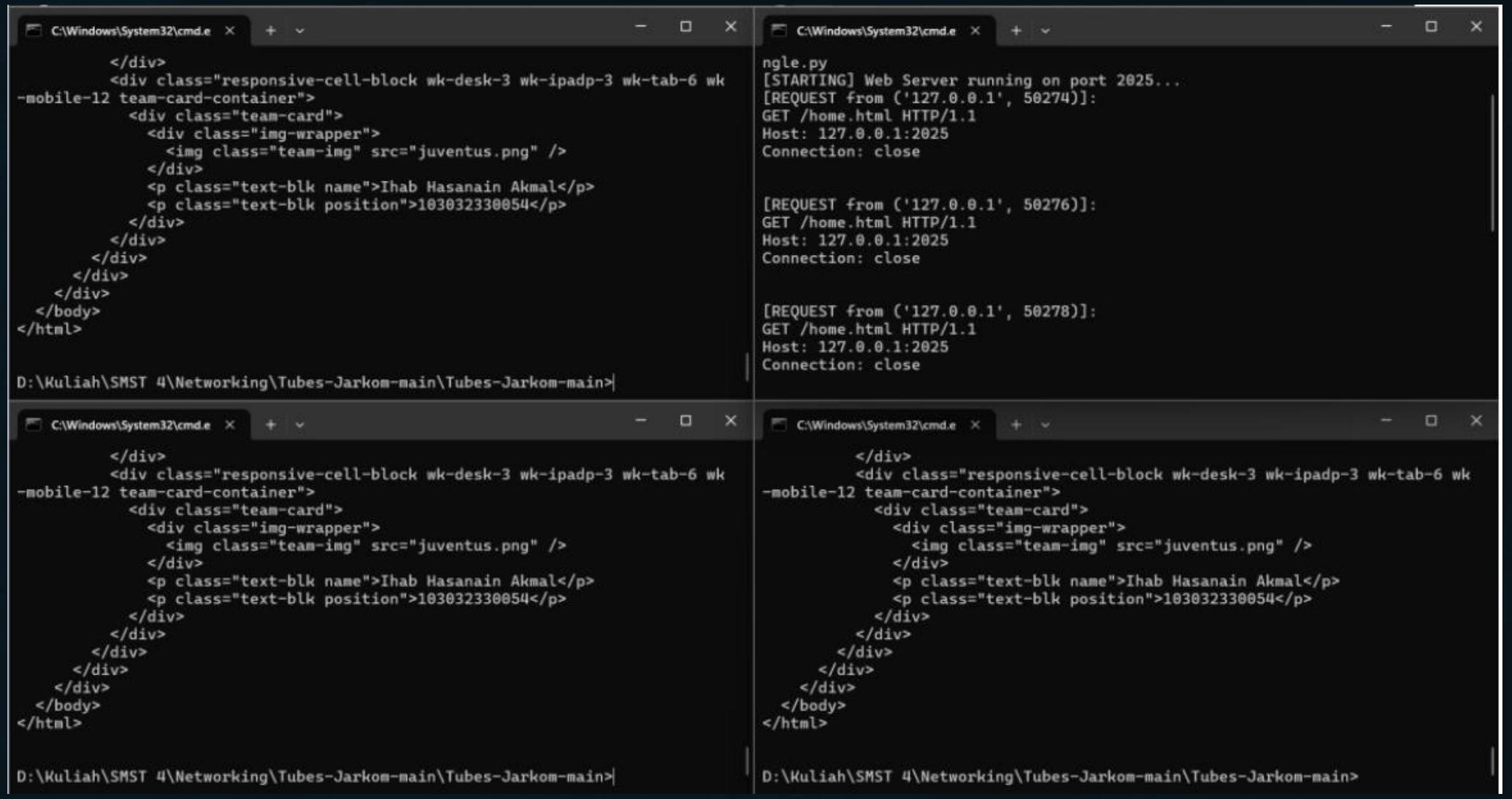
client_single.py

```python
import socket
import sys

def http_client(server_host, server_port, filename):
    try:
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client_socket.connect((server_host, int(server_port)))
        request_line = f"GET /{filename} HTTP/1.1\r\n"
        headers = (
            f"Host: {server_host}:{server_port}\r\n"
            "Connection: close\r\n\r\n"
        )
        http_request = request_line + headers
        client_socket.sendall(http_request.encode())
        response = b''
        while True:
            data = client_socket.recv(4096)
            if not data:
                break
            response += data
        print("[RESPONSE FROM SERVER]:")
        print(response.decode('utf-8', errors='replace'))

        client_socket.close()

    except Exception as e:
        print(f"[ERROR]: {e}")

if __name__ == "__main__":
    if len(sys.argv) != 4:
        print("Usage: python client.py <server_host> <server_port> <filename>")
    else:
        _, host, port, file = sys.argv
        http_client(host, port, file)

```

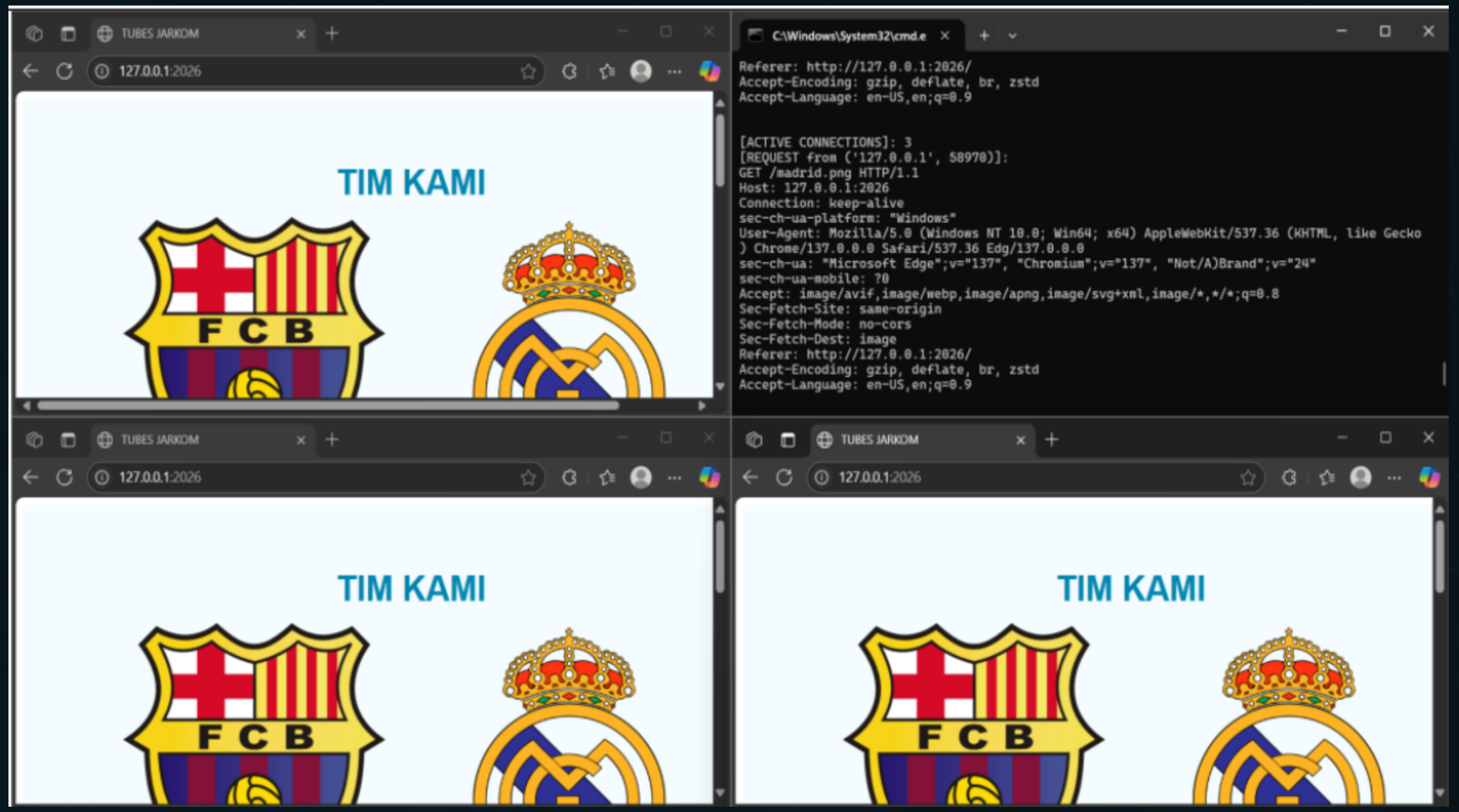
Hasil Program Single Threading



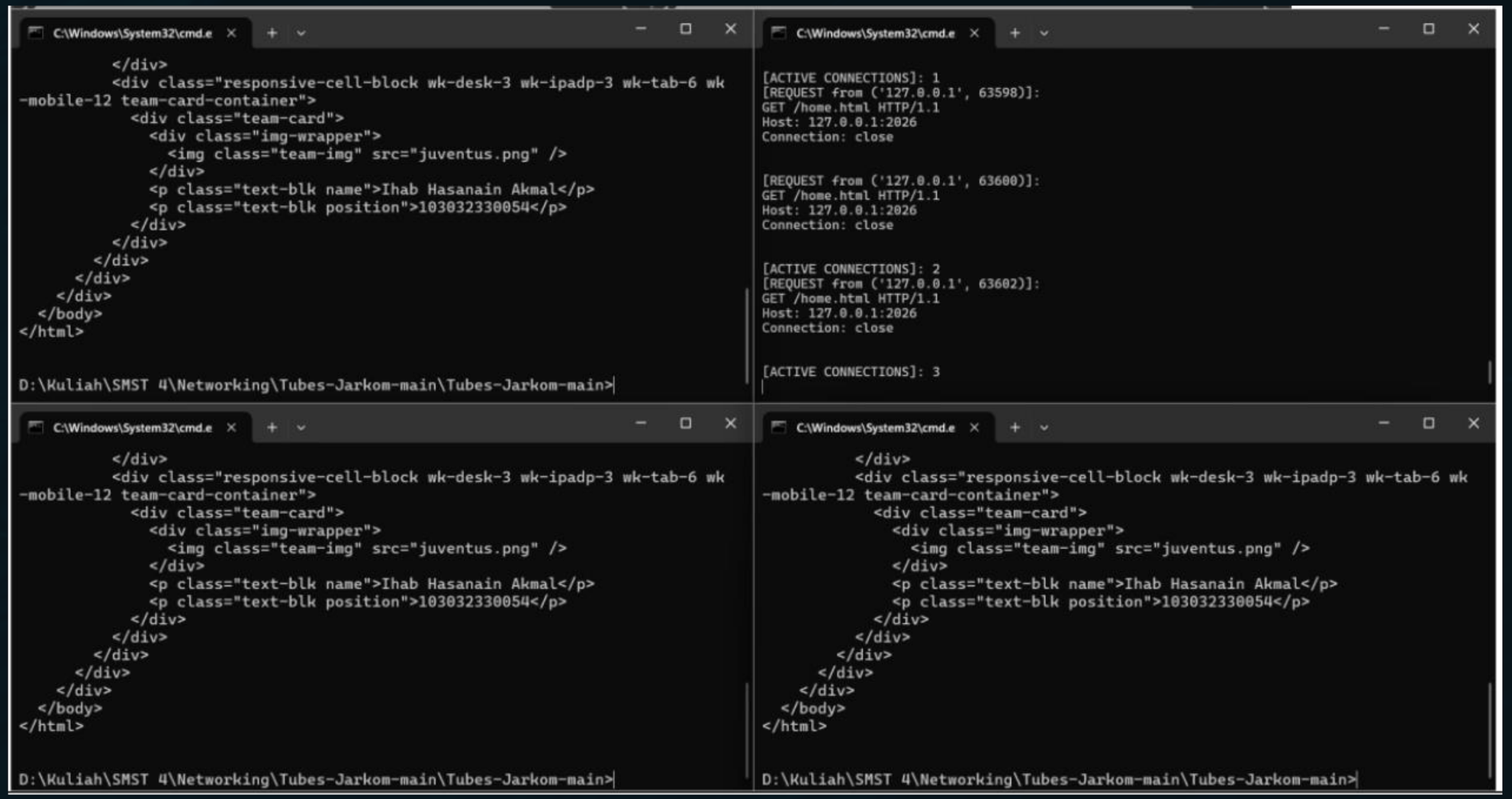
Hasil Program Single Threading

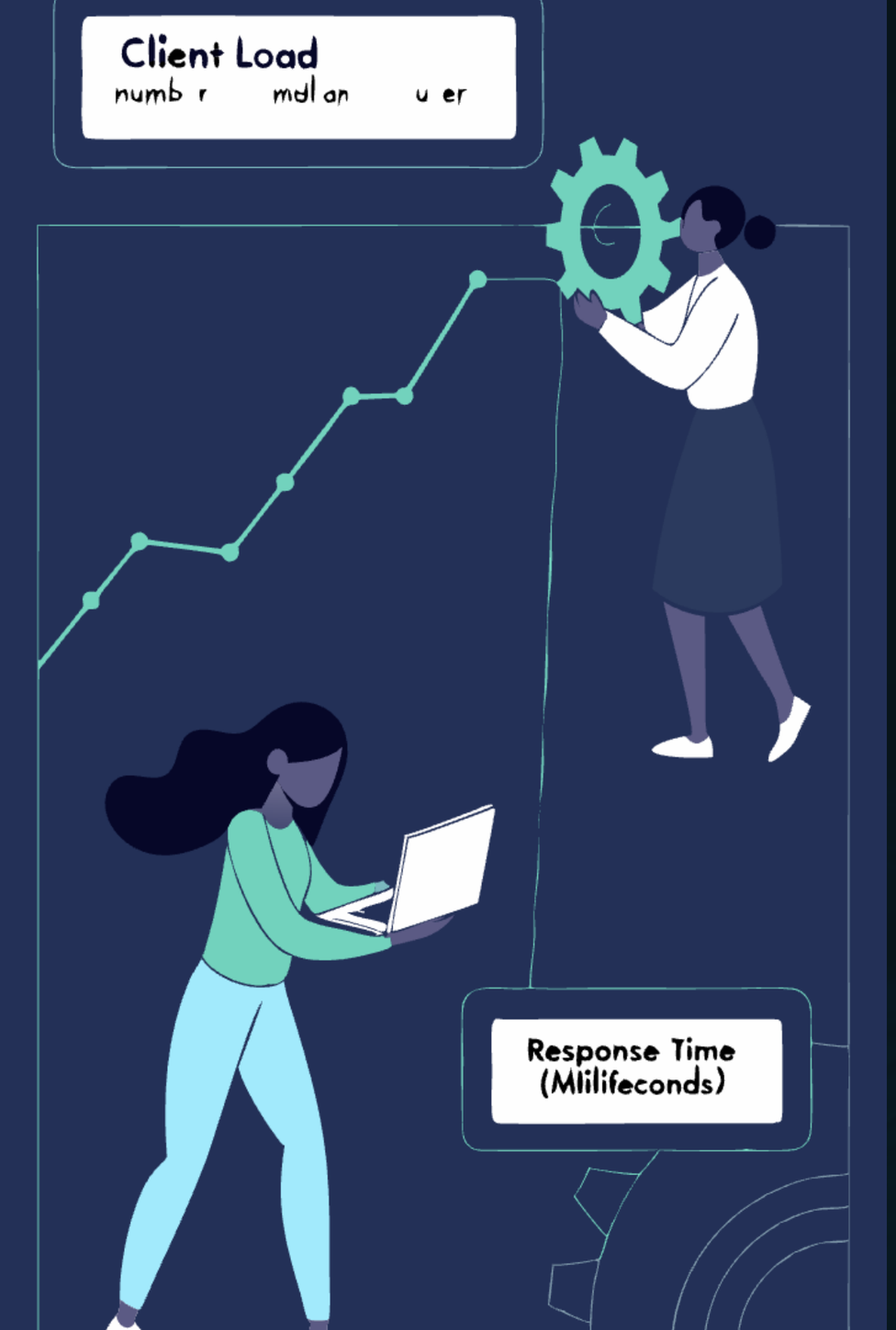


Hasil Program Multi Thread



Hasil Program Multi Thread





Kesimpulan

1x

Efisiensi

Multi-threading jauh lebih efisien dibandingkan single-threading

n

Paralelisme

Memproses beberapa request secara bersamaan

↓

Waktu Tunggu

Mengurangi waktu tunggu antar client

Dari hasil pengujian ini dapat disimpulkan bahwa penggunaan metode multi-threading pada server jauh lebih efektif dan efisien dibandingkan single-threading, terutama saat menangani banyak koneksi client secara bersamaan. Multi-threading memungkinkan server memproses beberapa request secara paralel, sehingga meningkatkan kecepatan respons dan mengurangi waktu tunggu antar client.

Sebaliknya, single-threading memiliki keterbatasan karena hanya dapat memproses satu request dalam satu waktu, yang dapat menyebabkan antrean saat beban koneksi meningkat.