1. Buat program async untuk mengambil data dari 3 URL secara parallel

    1. **Import Pustaka**:
        - asyncio digunakan untuk menjalankan fungsi asinkron.
        - aiohttp adalah pustaka yang memungkinkan kita untuk melakukan permintaan HTTP secara asinkron.
    2. **Fungsi fetch(url)**:
        - Fungsi ini adalah fungsi asinkron yang bertugas untuk mengambil konten dari URL yang diberikan.
        - aiohttp.ClientSession() digunakan untuk membuat sesi HTTP. Menggunakan sesi ini memungkinkan kita untuk melakukan beberapa permintaan dengan lebih efisien.
        - session.get(url) melakukan permintaan GET ke URL yang diberikan.
        - await response.text() mengembalikan konten respons sebagai teks.
    3. **Fungsi main()**:
        - Di dalam fungsi ini, kita mendefinisikan daftar URL yang ingin kita ambil datanya.
        - tasks = [fetch(url) for url in urls] membuat daftar tugas asinkron untuk setiap URL.
        - await asyncio.gather(*tasks) menjalankan semua tugas secara bersamaan dan menunggu hingga semuanya selesai.
        - Setelah semua hasil diperoleh, kita mencetak setiap hasil ke konsol.
    4. **Menjalankan Program**:
        - if __name__ == '__main__': memastikan bahwa kode di bawahnya hanya dijalankan jika file ini dieksekusi sebagai program utama.
        - asyncio.run(main()) memulai eksekusi fungsi main().

Dengan menggunakan pendekatan ini, kita dapat mengambil data dari beberapa sumber secara efisien dan cepat. Ini sangat berguna dalam aplikasi yang memerlukan pengambilan data dari banyak API atau situs web secara bersamaan.

1.Program Asynkron mengambil data 3 URL Secara paralel.py - D:\PARAMADINA UNIVERSITY CIKARANG\SEMESTER 6\Pemrograman Asinkron\UAS\1.Program Asynkron mengambil data 3 URL Secara paralel.py (3.12.6)

File Edit Format Run Options Window Help

```
import asyncio
import aiohttp

async def fetch(url):
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as response:
            return await response.text()

async def main():
    urls = [
        'https://example.com/1',
        'https://example.com/2',
        'https://example.com/3'
    ]
    tasks = [fetch(url) for url in urls]
    results = await asyncio.gather(*tasks)
    for result in results:
        print(result)

if __name__ == '__main__':
    asyncio.run(main())
```

IDLE Shell 3.12.6

File Edit Shell Debug Options Window Help

```
Python 3.12.6 (tags/v3.12.6:a4a2d2b, Sep  6 2024, 20:11:23) [MSC v.1940 64 bit (AMD64)] on win3
2
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:\PARAMADINA UNIVERSITY CIKARANG\SEMESTER 6\Pemrograman Asinkron\U
AS\1.Program Asynkron mengambil data 3 URL Secara paralel.py
Squeezed text (50 lines).
Squeezed text (50 lines).
Squeezed text (50 lines).
>>>
```

Ln: 8 Col: 0

1.Program Asynkron mengambil data 3 URL Secara paralel.py - D:\PARAMADINA UNIVERSITY CIKARANG\SEMESTER 6\Pemrograman Asinkron\UAS\1.Program Asynkron mengambil data 3 URL Secara paralel.py (3.12.6)

File Edit Format Run Options Window Help

```
import asyncio
import aiohttp

async def fetch(url):
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as response:
            return await response.text()

async def main():
    urls = [
        'https://example.com/1',
        'https://example.com/2',
        'https://example.com/3'
    ]
    tasks = [fetch(url) for url in urls]
    results = await asyncio.gather(*tasks)
    for result in results:
        print(result)

if __name__ == '__main__':
    asyncio.run(main())
```

IDLE Shell 3.12.6

File Edit Shell Debug Options Window Help

```
Python 3.12.6 (tags/v3.12.6:a4a2d2b, Sep  6 2024, 20:11:23) [MSC v.1940 64 bit (AMD64)] on win3
2
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:\PARAMADINA UNIVERSITY CIKARANG\SEMESTER 6\Pemrograman Asinkron\U
AS\1.Program Asynkron mengambil data 3 URL Secara paralel.py
<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvet
ica Neue", Helvetica, Arial, sans-serif;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 2em;
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
    @media (max-width: 700px) {
        div {
            margin: 0 auto;
            width: auto;
        }
```

Ln: 6 Col: 6

2. Buatlah program FastAPI yang menangani permintaan POST async.

Saya telah membuat program FastAPI lengkap yang menangani berbagai jenis permintaan POST async! Program ini mencakup:

Fitur Utama:

- 5 endpoint POST berbeda dengan use case yang berbeda
- Async/await patterns yang proper
- Pydantic models untuk validasi data
- Error handling dengan HTTPException
- Simulasi operasi database dan API eksternal

Endpoint yang tersedia:

1. /users - Membuat user baru dengan validasi async
2. /tasks - Membuat task dengan notifikasi async
3. /process-data - Memproses data dengan concurrent operations
4. /external-api - Memanggil API eksternal menggunakan httpx
5. /batch-process - Memproses data dalam batch secara async

2.program FastAPI yang menangani permintaan POST async.py - D:\PARAMADINA UNIVERSITY CIKARANG\SEMESTER 6\Pemrograman Asinkron\UAS\2.program FastAPI yang menangani permintaan POST async.py (3.12.6)

File Edit Format Run Options Window Help

```python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import asyncio
import httpx
from typing import Optional, List
import json

app = FastAPI(title="Async POST API", version="1.0.0")

# Pydantic models untuk request bodies
class UserCreate(BaseModel):
    name: str
    email: str
    age: Optional[int] = None

class TaskCreate(BaseModel):
    title: str
    description: str
    priority: str = "medium"

class DataProcessing(BaseModel):
    items: List[str]
    operation: str

# Simulasi database (dalam aplikasi nyata, gunakan database)
users_db = []
tasks_db = []

@app.get("/")
async def root():
    return {"message": "FastAPI Async POST Handler - Server Running!"}

# 1. POST sederhana untuk membuat user
@app.post("/users")
async def create_user(user: UserCreate):
    """Membuat user baru secara async"""
    # Simulasi operasi async (misalnya validasi email)
    await asyncio.sleep(0.1)  # Simulasi delay

    # Cek apakah email sudah ada
    for existing_user in users_db:
        if existing_user["email"] == user.email:
            raise HTTPException(status_code=400, detail="Email already exists")

    new_user = {
        "id": len(users_db) + 1,
        "name": user.name,
        "email": user.email,
        "age": user.age
    }
    users_db.append(new_user)

    return {"message": "User created successfully", "user": new_user}

# 2. POST dengan operasi async eksternal (simulasi API call)
@app.post("/tasks")
async def create_task(task: TaskCreate):
```

IDLE Shell 3.12.6

File Edit Shell Debug Options Window Help

```
Python 3.12.6 (tags/v3.12.6:a4a2d2b, Sep  6 2024, 20:11:23) [MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:\PARAMADINA UNIVERSITY CIKARANG\SEMESTER 6\Pemrograman Asinkron\UAS\2.program FastAPI yang menangani permintaan POST async.py
INFO:     Started server process [36m10716[0m]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

Ln: 5 Col: 0