01) Overview

The pipeline is built using a **microservices-style architecture** focused on modularity, scalability, and efficiency. It performs the following steps:

1. **Data Ingestion** from REST and GraphQL APIs (asynchronous).
2. **Data Transformation** into a normalized internal structure.
3. **Data Storage** into a NoSQL MongoDB database.
4. **RESTful API** exposure via FastAPI for downstream clients.

02) Performance

- Uses aiohttp and asyncio for **non-blocking HTTP calls**.
- Batch inserts using insert_many() into MongoDB.
- Caching and connection pooling handled by FastAPI + Uvicorn.

03) Reliability

- Uses try/except to catch network/database errors.
- Conditional writes to MongoDB only when data exists.
- Server auto-reloads with --reload during dev.
- Error logging creates a log file for each critical failure, ensuring traceability of when and why the error occurred in the log folder.

04)

- Input is validated and sanitized.
- MongoDB access can be secured with authentication (outside PoC scope).

05)Parallel Processing and Multi-Threading

- **Async I/O for I/O-Bound Operations**-asyncio.gather allows both fetch operations to run concurrently.
- **MongoDB** writes are also non-blocking using motor

06) Error Handling

- Every API call is wrapped with try/except
- Handles empty data, connection timeouts, etc.

07) Logging System

- Uses Python’s logging module to track critical events.
- On each critical error, a new .log file is created with a unique timestamped filename.
- Automatically cleans up old logs, keeping only the latest 100 to avoid clutter or excessive disk use.