Open Threat Feed Ingestion and Intelligence API

Assignment Brief:
Cybersecurity systems rely heavily on threat intelligence to detect and respond to malicious activity. In this assignment, you will build a mini threat intelligence processing pipeline that takes real-world, publicly available threat feeds, parses and processes them, enriches them with context, and exposes the processed data through an API interface.

This assignment is designed to test your ability to:

- Ingest and normalize disparate data sources,
- Design a clean and modular processing pipeline,
- Use Python effectively for parsing and enrichment,
- Demonstrate basic architectural thinking and documentation.
- Expose threat data via a RESTful API using Node.js.

You are given three real-world threat feeds (IP and URL based). Your job is to build a system that:

- Periodically or on-demand fetches these feeds.
- Normalizes and enriches the threat indicators (IOCs).
- Stores them in a retrievable format.
- Exposes them through a simple Node.js API.

The final solution should give a basic, end-to-end view of how raw threat intel feeds are turned into actionable data in real-world SOC or SIEM systems.
You are expected to keep your code well-organized and explain your architectural decisions. Think like a backend engineer building a microservice that could be deployed in a larger ecosystem.

Input Feeds:

- http://www.blocklist.de/lists/apache.txt – list of IPs
- http://www.spamhaus.org/drop/drop.txt – list of IPs/subnets
- https://osint.digitalside.it/Threat-Intel/lists/latesturls.txt – list of malicious URLs

Requirements:

1. Threat Feed Ingestion Module (Python):
    - Create a Python script that downloads and parses all 3 feeds.
    - Normalize all entries into a common schema.

2. Enrichment & Filtering Module (Python):
    - For IP addresses, enrich using a public API like ipinfo.io or a mocked local DB (to avoid key issues).
    - For URLs, filter only .exe, .zip, and suspicious extensions (optional ML bonus below).
    - De-duplicate entries.
    - (Bonus ML) – Write a small classifier in Python to label URLs as suspicious or benign using keyword heuristics or logistic regression.

3. Storage Layer:
- Save the processed and enriched IOCs in any of the following (choose one):
    - JSON file
    - MongoDB
    - Elastic Search

4. API Interface (Node.js):
- Build a simple REST API in Node.js to:
    - GET /iocs – return all IOCs
    - GET /iocs?type=ip or /iocs?source=spamhaus – filter by type or source
    - POST /refresh – triggers ingestion + enrichment pipeline again

Use ExpressJS. Avoid databases if not needed, and can read from JSON.

5. Architecture Documentation:
- Submit a README.md or ARCHITECTURE.md explaining:
- Modules
- Flow diagram or brief explanation
- How you'd scale this system for production (batching, queueing, etc.)
- Any assumptions

Expected Time:
- 6–8 hours of focused work
- You may take up to 4 days

Evaluation Criteria:
- Code quality (Python + Node.js)
- Architectural understanding
- Handling of edge cases (timeouts, invalid data)
- Use of clean folder structure and modularity
- (Bonus) Use of ML or smart filtering
- Clear documentation and reasoning