Technical Assessment: The Legacy Ledger Audit

Time Limit: 2-4 Hours

Context

We maintain a legacy internal service (legacy_ledger.py) used by our support staff to search for users and process manual credit adjustments.

Recently, this service has been flagged for critical issues:

1. **Security:** A recent automated scan flagged potential data leakage vulnerabilities.
2. **Performance:** Support staff report that the application "freezes" when multiple transactions are processed at once.

Your Mission

You have been provided with the source code legacy_ledger.py. Your task is to refactor this file to meet production standards.

Requirements

1. **Security Hardening (Priority #1):** * Identify and fix the vulnerability allowing unauthorized data access in the search endpoint.
    - Ensure the application is resistant to common web exploits (specifically SQL Injection).
2. **Performance Optimization:** * The process_transaction endpoint simulates a communication delay with a banking core (represented by time.sleep).
    - Refactor the architecture so that the API remains responsive to new requests, even if one request is waiting on the "banking core."
    - *Note:* You may use asyncio, threading, or background task queues as you see fit.
3. **Data Integrity:** * Ensure that database updates are atomic and safe.

Deliverables

- A refactored legacy_ledger.py (or a folder of files if you choose to split it up).
- A brief NOTES.md explaining:
    - The specific vulnerabilities you found.
    - Why you chose your specific performance solution.

Constraints

- You may switch web frameworks (e.g., from Flask to FastAPI/Sanic/Starlette) if desired.
- You must keep sqlite3 as the database for this assessment (to keep the setup simple).

- The API signatures (inputs/outputs) must remain compatible with the original version.