Seenity – Home Assignment

Position: Senior Backend Developer

Language: Python

Objective

This assignment is designed to evaluate your ability to build a clean, robust, and scalable Python backend solution. It simulates a simplified version of the data transformation and risk evaluation pipeline used in Seenity's platform.

You will build a service that processes incoming risk-related data, applies domain-specific transformation logic, and exposes results via an API and a CSV processing endpoint. This task is intended to assess your coding skills, architecture choices, problem-solving capabilities, and backend experience.

Background

Seenity collects external data to evaluate risks in various domains such as fraud, safety, and financial behavior. In this task, you will simulate a backend module that receives raw risk indicators, applies a transformation pipeline, and calculates an aggregated risk score with contextual logic.

Task Description

Part 1: Risk Processing Engine

Build a Python package named `risk_processor` that exposes a class responsible for transforming input data into structured risk scores. The class should:

- Accept raw indicators such as:
  - Crime index (0–10 scale)
  - Accident rate (0–10 scale)
  - Socioeconomic level (1–10)
  - Weather conditions (categorical)

- Normalize the indicators into values between 0 and 100
- Apply a multi-stage transformation pipeline that includes:
  - Conditional logic based on combinations of fields (e.g., high crime and low socioeconomics should trigger risk penalties)
  - Cross-feature logic (e.g., increased weather severity may amplify accident risks, depending on the socioeconomic context)
  - Weight configuration: weights for each indicator must be configurable via the class constructor
  - Penalty or bonus logic based on risk interaction patterns (e.g., if both crime and weather

are high, risk score should include an amplification factor)
  - Statistical noise or data smoothing to simulate real-world inconsistencies

- Ensure robust error handling and proper validation of inputs (type, range, required fields)

Part 2: API Interface

Create a backend service using FastAPI or Flask that exposes the following endpoints:

1. POST /process
  - Accepts a JSON payload with a city name and raw indicator data
  - Returns a structured JSON response with calculated risk scores and components

2. POST /process-csv
  - Accepts a CSV file upload with rows containing raw data
  - Processes the rows using multi-threading or asynchronous methods for performance
  - Returns a downloadable CSV with results
  - Must include row-level error tracking and should not fail the whole batch due to invalid rows

Part 3: Stability & Security

- All inputs must be validated: type, presence, and value range
- Errors must be reported clearly without exposing internal traces
- Logs should be clean, structured, and free from sensitive data
- The system should be able to handle unexpected input gracefully (e.g., missing fields, corrupt CSV)

Deliverables

- Complete code in a structured project layout
- README.md with:
  - Setup instructions
  - API documentation with examples
  - Description of transformation logic and assumptions
- requirements.txt or pyproject.toml
- (Optional) Dockerfile for packaging the service

Evaluation Criteria

- Code readability, structure, and maintainability
- OOP principles and extensibility
- Input validation and error resilience
- Multi-threaded or async implementation for CSV processing
- Complexity and thoughtfulness of transformation logic
- Adherence to clean architecture and secure patterns