Application 1 : Rule Engine with AST

Objective:

Develop a simple 3-tier rule engine application(Simple UI, API and Backend, Data) to determine user eligibility based on attributes like age, department, income, spend etc.The system can use Abstract Syntax Tree (AST) to represent conditional rules and allow for dynamic creation,combination, and modification of these rules.

Data Structure:

- Define a data structure to represent the AST.
- The data structure should allow rule changes
- E,g One `data structure` could be `Node` with following fields
    - `type`: String indicating the node type ("operator" for AND/OR, "operand" for conditions)
    - `left`: Reference to another `Node` (left child)
    - `right`: Reference to another `Node` (right child for operators)
    - `value`: Optional value for operand nodes (e.g., number for comparisons)

Data Storage

- Define the choice of database for storing the above rules and application metadata
- Define the schema with samples.

Sample Rules:

- `rule1 = "((age > 30 AND department = 'Sales') OR (age < 25 AND department = 'Marketing')) AND (salary > 50000 OR experience > 5)"`
- `rule2 = "((age > 30 AND department = 'Marketing')) AND (salary > 20000 OR experience > 5)"`

API Design:

1. **`create_rule(rule_string)`:** This function takes a string representing a rule (as shown in the examples) and returns a `Node` object representing the corresponding AST.
2. **`combine_rules(rules)`:** This function takes a list of rule strings and combines them into a single AST. It should consider efficiency and minimize redundant checks. You can explore different strategies (e.g., most frequent operator heuristic). The function should return the root node of the combined AST.

3. **`evaluate_rule(JSON data)`:** This function takes a `JSON` representing the combined rule's AST and a dictionary `data` containing attributes (e.g., `data = {"age": 35, "department": "Sales", "salary": 60000, "experience": 3}`). The function should evaluate the rule against the provided data and return True if the user is of that cohort based on the rule, False otherwise.

Test Cases:

1. Create individual rules from the examples using `create_rule` and verify their AST representation.
2. Combine the example rules using `combine_rules` and ensure the resulting AST reflects the combined logic.
3. Implement sample JSON data and test `evaluate_rule` for different scenarios.
4. Explore combining additional rules and test the functionality.

Bonus:

- Implement error handling for invalid rule strings or data formats (e.g., missing operators, invalid comparisons).
- Implement validations for attributes to be part of a catalog.
- Allow for modification of existing rules using additional functionalities within `create_rule` or separate functions.This could involve changing operators, operand values, or adding/removing sub-expressions within the AST.
- Consider extending the system to support user-defined functions within the rule language for advanced conditions (outside the scope of this exercise).

Application 2 : Real-Time Data Processing System for Weather Monitoring with Rollups and Aggregates

Objective:

Develop a real-time data processing system to monitor weather conditions and provide summarized insights using rollups and aggregates. The system will utilize data from the OpenWeatherMap API (https://openweathermap.org/).

Data Source:

The system will continuously retrieve weather data from the OpenWeatherMap API. You will need to sign up for a free API key to access the data. The API provides various weather parameters, and for this assignment, we will focus on:

- `main`: Main weather condition (e.g., Rain, Snow, Clear)
- `temp`: Current temperature in Centigrade
- `feels_like`: Perceived temperature in Centigrade
- `dt`: Time of the data update (Unix timestamp)

Processing and Analysis:

- The system should continuously call the OpenWeatherMap API at a configurable interval (e.g., every 5 minutes) to retrieve real-time weather data for the metros in India. (Delhi, Mumbai, Chennai, Bangalore, Kolkata, Hyderabad)
- For each received weather update:
    - Convert temperature values from Kelvin to Celsius (tip : you can also use user preference).

Rollups and Aggregates:

1. **Daily Weather Summary:**
    - Roll up the weather data for each day.
    - Calculate daily aggregates for:
        - Average temperature
        - Maximum temperature
        - Minimum temperature
        - Dominant weather condition (give reason on this)
    - Store the daily summaries in a database or persistent storage for further analysis.
2. **Alerting Thresholds:**
    - Define user-configurable thresholds for temperature or specific weather conditions (e.g., alert if temperature exceeds 35 degrees Celsius for two consecutive updates).
    - Continuously track the latest weather data and compare it with the thresholds.
    - If a threshold is breached, trigger an alert for the current weather conditions. Alerts could be displayed on the console or sent through an email notification system (implementation details left open-ended).
3. **Implement visualizations:**
    - To display daily weather summaries, historical trends, and triggered alerts.

Test Cases:

1. **System Setup:**
    - Verify system starts successfully and connects to the OpenWeatherMap API using a valid API key.
2. **Data Retrieval:**
    - Simulate API calls at configurable intervals.
    - Ensure the system retrieves weather data for the specified location and parses the response correctly.
3. **Temperature Conversion:**

    - Test conversion of temperature values from Kelvin to Celsius (or Fahrenheit) based on user preference.
4. **Daily Weather Summary:**
    - Simulate a sequence of weather updates for several days.
    - Verify that daily summaries are calculated correctly, including average, maximum, minimum temperatures,and dominant weather condition.
5. **Alerting Thresholds:**
    - Define and configure user thresholds for temperature or weather conditions.
    - Simulate weather data exceeding or breaching the thresholds.
    - Verify that alerts are triggered only when a threshold is violated.

Bonus:

- Extend the system to support additional weather parameters from the OpenWeatherMap API (e.g., humidity, wind speed) and incorporate them into rollups/aggregates.
- Explore functionalities like weather forecasts retrieval and generating summaries based on predicted conditions.

Evaluation:

The assignment will be evaluated based on the following criteria:

- Functionality and correctness of the real-time data processing system.
- Accuracy of data parsing, temperature conversion, and rollup/aggregate calculations.
- Efficiency of data retrieval and processing within acceptable intervals.
- Completeness of test cases covering various weather scenarios and user configurations.
- Clarity and maintainability of the codebase.
- (Bonus) Implementation of additional features.

Artifacts To be Submitted For the Assignments

- Codebase - preferably a github handle
- Readme should be comprehensive - build instructions and design choices are mandatory
- Readme should contain dependencies to download for setting up and running the application. E.g a webserver, Database all can be docker or podman containers