Documentation: Rule Engine and Weather Monitoring System

Overview

This document provides a comprehensive overview of two applications developed for the assignment:

1. **Rule Engine with AST**: A backend-focused application for dynamically creating, combining, and evaluating user-defined rules.
2. **Real-Time Data Processing System for Weather Monitoring with Rollups and Aggregates:** A real-time weather tracking system with customizable preferences and alert functionality.

The focus is on explaining the design choices, key features, and testing processes for each application, while the technical setup and deployment instructions are available in the GitHub repository.

GitHub Repository: GitHub Link

Application 1: Rule Engine with AST

Executive Summary

The **Rule Engine with AST** enables users to create and manage complex conditional rules. It uses Abstract Syntax Trees (AST) to represent rules and provides APIs for creating, combining, and evaluating them. The application supports dynamic rule management and evaluates user attributes against defined rules.

Key Features

- **Rule Creation**: Parses user-defined rules into AST structures.
- **Rule Combination**: Combines multiple rules using logical operators (`AND`, `OR`).
- **Rule Evaluation**: Validates user data against combined rules and returns the result.
- **UI and API Integration**: A simple UI complements the RESTful APIs for seamless interaction.

Design Choices

1. **AST Structure**: Abstract Syntax Trees simplify the representation of complex, nested rules.

2. **SQLite Database**: Chosen for its simplicity and portability to store rules and metadata.
3. **Flask Framework**: Lightweight and efficient, ideal for backend and API development.

Testing Process

- **Create Rule**: Verify that rule strings are parsed correctly into AST structures.
- **Combine Rules**: Test combining multiple rules with logical operators.
- **Evaluate Rules**: Validate user attributes against rules with sample data.
- **UI Testing**: Test the web interface for smooth interaction with rule creation and evaluation.

Known Issues

- **Rule Modification**: Currently, rules cannot be edited after creation.

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

Executive Summary

The Real-Time Data Processing System for Weather Monitoring with Rollups and Aggregates integrates with the OpenWeatherMap API to fetch and display live weather data. It supports user-defined preferences for temperature units, thresholds for weather alerts, and persistent storage of weather summaries.

Key Features

- **Real-time Weather Updates**: Fetches live data for multiple cities.
- **Forecasts**: Displays a 5-day weather forecast with daily summaries.
- **Custom Preferences**: Users can toggle temperature units (Celsius/Fahrenheit).
- **Weather Alerts**: Sends email notifications for severe weather conditions.

Design Choices

1. **RESTful API Integration**: Fetches live data with minimal latency using OpenWeatherMap's API.
2. **UI Optimization**: A user-friendly web interface displays weather summaries and alerts.
3. **SQLite Database**: Stores user preferences and weather data for persistence.
4. **Docker Compatibility**: Designed to support containerization for easy deployment.

Testing Process

- **Real-time Updates**: Test API calls with valid and invalid city names.
- **Preference Management**: Verify switching between Celsius and Fahrenheit.
- **Weather Alerts**: Simulate alerts by setting thresholds for temperature and humidity.
- **UI Testing**: Ensure correct display of weather data and user inputs.

Known Issues

- **Unit Conversion**: Inconsistencies may appear in temperature unit toggling logic.

Security and Performance Considerations

Security

- **Environment Variables**: Sensitive data like API keys and email credentials are stored as environment variables.
- **Input Validation**: All user inputs are sanitized to prevent injection attacks.
- **SSL/TLS**: Recommended for secure communication in production.

Performance

- **Database Optimization**: Indexing improves query performance for large datasets.
- **Multithreaded Tasks**: Asynchronous background tasks fetch weather data to ensure responsiveness.
- **Caching**: (Planned) Implement caching for frequently accessed data.

Testing and Validation Overview

1. **Manual Testing**:
    - Test APIs via Postman to validate endpoints.
    - Use the web interface for real-time interaction with both applications.
2. **Automated Testing**:
    - Pytest can be used for backend logic validation (future enhancement).
3. **Edge Case Handling**:
    - Validate missing attributes in rule evaluation.
    - Handle invalid city names in the weather application gracefully.

Submission Instructions

1. **GitHub Repository**:
    - The complete codebase for both applications is available on GitHub: GitHub Link.
2. **Setup and Deployment**:
    - Refer to the GitHub README for setup instructions, including environment setup and testing processes.
3. **Testing Applications**:
    - Follow the steps outlined in this document for manual and API-based testing.
4. **Known Issues**:
    - See the sections on known limitations for both applications.
5. **Enhancements**:
    - Include recommendations for performance and security improvements in future iterations.