Project Report

Rule Engine Application with Abstract Syntax Trees (AST)

Author: Harish

Date: [Project Completion Date]

Abstract

This project report documents the development of a rule engine application, which leverages Abstract Syntax Trees (AST) to create, combine, and evaluate rules. The project is implemented using Flask, SQLAlchemy, and SQLite to provide a scalable solution for rule-based decision making.

Introduction

This rule engine project allows for the efficient management and evaluation of complex rules through the use of AST. The key objectives of this project are to enable users to define, combine, and evaluate rules within a structured 3-tier architecture. The application is built using Flask (for backend), SQLAlchemy (ORM), SQLite (database), and Tkinter (UI).

System Architecture

The application follows a 3-tier architecture consisting of backend, frontend, and testing layers. The backend handles rule processing, storage, and retrieval. The frontend provides a Tkinter-based UI, while an automated testing script ensures the functionality of various components.

Features and Functionalities

1. Create Rule: Allows users to define new rules, each assigned a unique ID.

2. Combine Rules: Enables the combination of multiple rules by ID, creating a composite rule.

3. Evaluate Rule: Accepts rule ID and JSON data, evaluating complex rule structures based on input parameters.

API Endpoints

The application provides the following API endpoints:

1. Create Rule
   - Endpoint: /create_rule
   - Method: POST
   - Description: Accepts a rule in JSON format and returns an assigned rule ID.

2. Combine Rules
   - Endpoint: /combine_rules
   - Method: POST
   - Description: Combines multiple rule IDs into a new composite rule.

3. Evaluate Rule
   - Endpoint: /evaluate_rule
   - Method: POST
   - Description: Evaluates a composite rule using provided data in JSON format.

Implementation Details

Backend: The main.py script uses Flask to handle HTTP requests and SQLAlchemy as an ORM.

Frontend: The Tkinter UI (rlg.py) provides user interaction for creating and evaluating rules.

Testing: An automated test script (test.py) uses the requests library to validate app functionality.

Results and Discussion

The rule engine successfully performs rule creation, combination, and evaluation. Key challenges, such as handling complex rule structures, were resolved through the use of ASTs.

Future Enhancements

Possible future improvements include expanding the UI functionality, adding more complex rule types, and integrating machine learning techniques for adaptive rule management.

Conclusion

This project effectively demonstrates the power of ASTs in creating a scalable rule engine application. The implementation offers a user-friendly experience for managing complex rule-based decisions.

References

Resources: Flask, SQLAlchemy, SQLite, Python documentation, and relevant AST literature.