Project Documentation: Rule Engine with AST

# Project Overview

The Rule Engine application is a 3-tier system designed to evaluate user eligibility based on various attributes such as age, department, income, and experience. Utilizing an Abstract Syntax Tree (AST) to represent conditional rules, the application allows for dynamic creation, combination, and modification of these rules.

# Tech Stack

Backend: Java Spring Boot

Database: PostgreSQL

Frontend: React with Vite

Testing: Postman

# Installation and Setup

## Prerequisites

Java Development Kit (JDK) 11 or higher

PostgreSQL database server

React +Vite (for frontend)

Maven (for backend dependencies)

## Backend Setup

1. Clone the Repository:

git clone <repository-url>

cd rule-engine-backend

2. Configure PostgreSQL:

Create a new database (e.g., rule\_engine\_db).

Update the application.properties file with your database credentials:

spring.datasource.url=jdbc:postgresql://localhost:5432/rule\_engine\_db  
spring.datasource.username=<your-username>  
spring.datasource.password=<your-password>

3. Add Dependencies:

Here are the main Spring Boot dependencies used in the project and Ensure the following dependencies are included in pom.xml file:

1. \*\*Spring Boot Starter Web\*\*: Provides all the necessary dependencies to build a web application, including embedded Tomcat and MVC architecture.

2. \*\*Spring Boot Starter Data JPA\*\*: Simplifies database access with Spring Data JPA, providing support for creating repositories and interacting with databases in an object-oriented way.

3. \*\*PostgreSQL Driver\*\*: Enables the application to communicate with the PostgreSQL database server.

4. \*\*Lombok\*\*: Reduces boilerplate code by automating common tasks like generating getters and setters, constructors, and equals/hashCode methods.

4. Run the Backend Application:

mvn spring-boot:run

## Frontend Setup

1. Navigate to Frontend Directory:

cd rule-engine-frontend

2. Install Dependencies:

npm install

3. Run the Frontend Application:

npm run dev

# API Design

## Functions Overview

1. create\_rule(rule\_string): This function parses a string representing a rule and returns a Node object that represents the corresponding AST.

2. combine\_rules(rules): This function takes a list of rule strings and combines them into a single AST, optimizing for efficiency and minimizing redundant checks.

3. evaluate\_rule(JSON data): This function evaluates the combined AST against provided attributes in a dictionary and returns True or False based on user eligibility.

# Database Schema

Rules Table:

CREATE TABLE rules (  
 id SERIAL PRIMARY KEY,  
 rule\_string TEXT NOT NULL,  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);

Sample Data:

INSERT INTO rules (rule\_string) VALUES   
 ('((age > 30 AND department = ''Sales'') OR (age < 25 AND department = ''Marketing'')) AND (salary > 50000 OR experience > 5)'),  
 ('((age > 30 AND department = ''Marketing'')) AND (salary > 20000 OR experience > 5)');

# Testing

To test the API endpoints, use Postman. You can make GET and POST requests to the following endpoints:

- Create Rule: POST /api/rules/create

- Combine Rules: POST /api/rules/combine

- Evaluate Rule: POST /api/rules/evaluate

## Sample Postman Requests

1. Create Rule:

URL: http://localhost:8080/api/rules/create

Body:  
{  
 "rule\_string": "((age > 30 AND department = 'Sales') OR (age < 25 AND department = 'Marketing')) AND (salary > 50000 OR experience > 5)"  
}

2. Evaluate Rule:

URL: http://localhost:8080/api/rules/evaluate

Body:  
{  
 "data": {  
 "age": 35,  
 "department": "Sales",  
 "salary": 60000,  
 "experience": 3  
 }  
}

# Conclusion

This documentation provides an overview of the Rule Engine application, detailing its setup, functionality, and testing procedures. The use of Spring Boot, PostgreSQL, and React with Vite facilitates a robust and scalable architecture for evaluating user eligibility based on complex rules.