Project Overview: Rule Engine Using Abstract Syntax Tree (AST)

Here's a summary of my project, my understanding of how it works, and the technologies/tools involved:

Objective:

The goal of this project is to build a **dynamic rule engine** that allows users to:

1. **Create rules** using custom conditions.
2. **Evaluate user eligibility** based on these rules.
3. Store rules in a **MongoDB** database, allowing users to update or delete existing rules.
4. Use **Abstract Syntax Tree (AST)** to dynamically evaluate the conditions provided in the rules and determine if a user qualifies based on inputs such as age, income, department, and experience.

The application has a **web interface** where users can:

- **Input rules** (with rule name and rule expression).
- See a **list of existing rules**.
- Provide user information (age, income, etc.) and **evaluate eligibility** based on both custom rules and predefined standard rules.

Here's a summary of your project, my understanding of how it works, and the technologies/tools involved:

Project Overview: Rule Engine Using Abstract Syntax Tree (AST)

Objective:

The goal of this project is to build a **dynamic rule engine** that allows users to:

1. **Create rules** using custom conditions.
2. **Evaluate user eligibility** based on these rules.
3. Store rules in a **MongoDB** database, allowing users to update or delete existing rules.
4. Use **Abstract Syntax Tree (AST)** to dynamically evaluate the conditions provided in the rules and determine if a user qualifies based on inputs such as age, income, department, and experience.

The application has a **web interface** where users can:

- **Input rules** (with rule name and rule expression).
- See a **list of existing rules**.
- Provide user information (age, income, etc.) and **evaluate eligibility** based on both custom rules and predefined standard rules.

How It Works:

1. Rule Creation

- Users input a **rule name** and **rule expression** (e.g., age > 25 && income > 30000).
- This rule is sent to the backend (Spring Boot application) and saved in the **MongoDB** database.
- The rule is then displayed in a table, allowing users to see all stored rules.

2. Rule Storage

- Rules are stored in **MongoDB** using a repository (RuleRepository).
- The rules consist of two fields:
  - **Rule Name**: A name to identify the rule.
  - **Rule Expression**: The conditional logic (e.g., age > 18 && experience > 2).
- Rules can be **updated** or **deleted** from the database using buttons in the rule list.

3. Evaluation of User Eligibility

- Users input values such as **age, income, department, and experience**.
- The backend evaluates the user's eligibility by:
  - Fetching all stored rules from MongoDB.
  - Using **AST** (simulated using JavaScript's ScriptEngine) to evaluate the rule expressions dynamically.
  - Returning whether the user is **eligible or not** based on the given input and rules.

4. Predefined Standard Rules

- In addition to custom rules, the system uses **predefined standard rules** (e.g., age >= 18 && income >= 20000).
- If custom rules do not match, these predefined rules are used for evaluation.

Technologies and Tools Used:

Frontend:

1. **HTML**: Used to create the structure of the web interface.
2. **CSS**: Styling for a clean, modern, and dynamic user interface. It uses Google Fonts for typography and media queries for responsiveness.
3. **JavaScript**:
   - Handles user input and interactions with the backend.
   - Sends **AJAX requests** using fetch to create, update, delete, and evaluate rules.
   - Dynamically updates the rule table and displays evaluation results.

Backend:

1. **Spring Boot**:
   - The main backend framework that handles REST API creation.
   - Manages the business logic of rule creation, deletion, updating, and evaluation.

   - Provides endpoints such as /rules (for CRUD operations) and /rules/evaluate (for eligibility checking).
   - Uses **JavaScript's ScriptEngine** for dynamic rule evaluation (simulating AST).
2. **MongoDB**:
   - NoSQL database used to store and manage rules.
   - The repository layer (RuleRepository) interacts with MongoDB using **Spring Data MongoDB**.
3. **Maven**:
   - Build automation and dependency management tool.
   - Manages project dependencies such as Spring Boot, MongoDB, and others.
4. **Java 8**:
   - Core programming language for the backend.
   - Features like Optional and ScriptEngineManager are used for rule evaluation and logic control.

Tools:

1. **Eclipse IDE**: For writing and managing the Java code and handling the Maven project structure.
2. **Postman**: For testing the API endpoints (CRUD operations and evaluation).
3. **MongoDB**: For managing rules in a NoSQL database.
4. **Google Chrome Developer Tools**: For inspecting the frontend, testing JavaScript, and troubleshooting errors.

Project Flow Summary:

1. **User Inputs a Rule**: Rule creation form sends the data to the backend.
2. **Backend Stores Rule**: Rule is stored in MongoDB and displayed on the web interface.
3. **User Evaluates Eligibility**: User provides data like age and income, which are sent to the backend.
4. **AST Evaluation**: The backend evaluates the rule expressions using AST and responds with an eligibility result.
5. **Result Display**: The result (eligible or not) is shown on the web page.

Evaluate the user eligibility via Postman:-

To evaluate the user eligibility via **Postman**, you can send a **POST** request to the backend's /evaluate endpoint, passing user data such as age, income, department, and experience. Here's how you can do it step-by-step:

Postman Request for User Eligibility Evaluation

1. URL

- **POST**: http://localhost:8080/rules/evaluate

2. Method

- **POST** request.

3. Headers

- **Content-Type**: application/json

4. Body (JSON)

You'll need to provide user information in JSON format. Here's an example of what the body might look like:

```
{
   "age": 30,
   "income": 50000,
   "department": "IT",
   "experience": 5
}
```

- **age**: The age of the user.
- **income**: The user's income.
- **department**: The department they belong to (e.g., "IT", "Sales").
- **experience**: The user's work experience in years.

5. Expected Response

The backend will evaluate the eligibility of the user based on the input rules (both custom and standard). You should get a response like this:

```
{
   "eligible": true
}
```

Or, if the user does not meet the conditions:

```
{
   "eligible": false
}
```

Step-by-Step Process in Postman:

1. **Open Postman**.
2. **Create a New Request**:
    - Set the method to **POST**.
    - In the **URL** field, type: http://localhost:8080/rules/evaluate.
3. **Set Headers**:

- Click on the **Headers** tab.
- Add the header: Content-Type: application/json.

4. **Enter Body Data**:
   - Click on the **Body** tab.
   - Choose **raw** and then select **JSON**.
   - Paste the following JSON data (or similar based on your test case):

```
{
  "age": 30,
  "income": 50000,
  "department": "IT",
  "experience": 5
}
```

5. **Send Request**:
   - Click the **Send** button.
6. **View the Response**:
   - Check the **response** in the lower part of the Postman window.
   - You should see something like this:

```
{
  "eligible": true
}
```

Backend Setup:

Make sure your **Spring Boot backend** is running and listening on port 8080, and pMongoDB is up and running, as the evaluation process depends on stored rules.

Project Structure

Here’s how the project structure will look:

```
RuleEngineAST/
│
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/
│   │   │       └── example/
│   │   │           └── ruleengine/
│   │   │               ├── controller/
│   │   │               │   └── RuleController.java
│   │   │               ├── model/
│   │   │               │   └── Rule.java
│   │   │               ├── repository/
│   │   │               │   └── RuleRepository.java
│   │   │               ├── service/
│   │   │               │   └── RuleService.java
│   │   │               └── RuleEngineASTApplication.java
│   │   ├── resources/
│   │   │   ├── application.properties
│   │   │   └── static/
│   │   │       ├── index.html
│   │   │       ├── styles.css
│   │   │       └── scripts.js
│   └── test/
│       └── java/
│           └── com/
│               └── example/
│                   └── ruleengine/
├── pom.xml
└── README.md
```