Application 1 : Rule Engine with AST:

Data Model and Database Setup:

```
# db_setup.py
import sqlite3

# Connect to the SQLite database
conn = sqlite3.connect('rules.db')
cursor = conn.cursor()

# Create the rules table
cursor.execute('''CREATE TABLE IF NOT EXISTS rules (
    rule_id INTEGER PRIMARY KEY,
    rule_expression TEXT
)''')

# Create the nodes table to represent AST nodes
cursor.execute('''CREATE TABLE IF NOT EXISTS nodes (
    node_id INTEGER PRIMARY KEY,
    rule_id INTEGER,
    type TEXT,
    left_node INTEGER,
    right_node INTEGER,
    value TEXT,
    FOREIGN KEY (rule_id) REFERENCES rules(rule_id)
)''')

conn.commit()
```

```
conn.close()
```

Define AST Node Class and Rule Parser:

```
# ast_nodes.py
import re

class Node:
    def __init__(self, node_type, left=None, right=None, value=None):
        self.type = node_type  # "operator" or "operand"
        self.left = left       # Left child
        self.right = right     # Right child
        self.value = value     # Condition (for operands)

def parse_condition(condition):
    if '>' in condition:
        field, value = condition.split('>')
        return ('>', field.strip(), int(value.strip()))
    elif '<' in condition:
        field, value = condition.split('<')
        return ('<', field.strip(), int(value.strip()))
    elif '=' in condition:
        field, value = condition.split('=')
        return ('=', field.strip(), value.strip())

def create_rule(rule_string):
    tokens = re.split(r'(\(|\)|AND|OR)', rule_string)
    tokens = [token.strip() for token in tokens if token.strip()]

    def parse(tokens):
        if len(tokens) == 1:
```

```
            condition = tokens[0]
            operator, field, value = parse_condition(condition)
            return Node(node_type='operand', value=(operator, field, value))

        left = parse([tokens[0]])
        right = parse([tokens[2]])
        return Node(node_type='operator', left=left, right=right, value=tokens[1])

    return parse(tokens)
```

Combine Rules:

```
# combine_rules.py
def combine_rules(rules):
    if len(rules) == 1:
        return rules[0]

    combined_root = Node('operator', left=rules[0], right=rules[1], value='AND')
    for rule in rules[2:]:
        combined_root = Node('operator', left=combined_root, right=rule, value='AND')
    return combined_root
```

Evaluate Rules Against Data:

```
# evaluate_rule.py
def evaluate_rule(ast, data):
    if ast.type == 'operand':
        operator, field, value = ast.value
```

```
        if operator == '>':
            return data.get(field, 0) > value
        elif operator == '<':
            return data.get(field, 0) < value
        elif operator == '=':
            return data.get(field, '') == value

    elif ast.type == 'operator':
        left_result = evaluate_rule(ast.left, data)
        right_result = evaluate_rule(ast.right, data)
        if ast.value == 'AND':
            return left_result and right_result
        elif ast.value == 'OR':
            return left_result or right_result
```

API Endpoints:

```
# main.py
from fastapi import FastAPI, HTTPException
from ast_nodes import create_rule
from combine_rules import combine_rules
from evaluate_rule import evaluate_rule

app = FastAPI()
rules_db = {}  # Temporary in-memory storage

@app.post("/create_rule")
async def create_rule_endpoint(rule_string: str):
    try:
        rule_ast = create_rule(rule_string)
```

```
        rule_id = len(rules_db) + 1
        rules_db[rule_id] = rule_ast
        return {"rule_id": rule_id}
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))

@app.post("/combine_rules")
async def combine_rules_endpoint(rule_ids: list[int]):
    try:
        rules = [rules_db[rule_id] for rule_id in rule_ids]
        combined_rule = combine_rules(rules)
        return {"combined_rule": combined_rule}
    except KeyError:
        raise HTTPException(status_code=404, detail="One or more rules not found.")

@app.post("/evaluate_rule")
async def evaluate_rule_endpoint(rule_id: int, data: dict):
    try:
        rule = rules_db[rule_id]
        result = evaluate_rule(rule, data)
        return {"result": result}
    except KeyError:
        raise HTTPException(status_code=404, detail="Rule not found.")
```

Frontend:

```
<!-- index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Rule Engine</title>
</head>
<body>
    <h1>Rule Engine</h1>
    <div>
        <input id="rule" placeholder="Enter Rule e.g., age > 30 AND salary > 50000">
        <button onclick="createRule()">Create Rule</button>
        <div id="rule-result"></div>
    </div>
    <div>
        <input id="rule-id" placeholder="Enter Rule ID">
        <input id="data" placeholder="Enter Data e.g., {'age': 35, 'salary': 60000}">
        <button onclick="evaluateRule()">Evaluate Rule</button>
        <div id="evaluation-result"></div>
    </div>

    <script>
        async function createRule() {
            const rule = document.getElementById("rule").value;
            const response = await fetch("/create_rule", {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ rule_string: rule })
            });
            const data = await response.json();
            document.getElementById("rule-result").innerText = `Rule ID: ${data.rule_id}`;
        }

        async function evaluateRule() {
```

```
        const ruleId = document.getElementById("rule-id").value;
        const data = JSON.parse(document.getElementById("data").value);
        const response = await fetch(`/evaluate_rule?rule_id=${ruleId}`, {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify(data)
        });
        const result = await response.json();
        document.getElementById("evaluation-result").innerText = `Result: ${result.result}`;
      }
    </script>
</body>
</html>
```