Architecture Documentation

This document provides a detailed overview of the architecture, including its components, interactions, and workflows. The architecture is designed to support a rule-based system for integrating external services, sensors, and automation solutions.

Overview

The architecture is divided into several layers and components, ensuring scalability, modularity, and seamless integration of external systems. The primary layers include:

1. **Client Layer**
2. **API Gateway Layer**
3. **Core Services**
4. **Service Layer**
5. **Data Layer**
6. **External Systems**

Layers and Components

1. Client Layer

The client layer is responsible for providing a user interface and enabling communication with the backend services.

- **React UI**: A web-based interface that allows users to define rules, view service statuses, and manage notifications.
- **WebSocket Client**: Handles real-time communication for updates and notifications.

2. API Gateway Layer

This layer serves as the entry point for all client requests, ensuring secure and efficient routing of data.

- **Spring Gateway**: A lightweight and efficient gateway that routes API requests to the appropriate core services.
- **Authentication**: Ensures secure access to the system by validating user credentials and authorizing requests.

3. Core Services

The core services implement the business logic and manage the orchestration of services.

- **Rule Engine**: Processes if-then rules defined by users, validates them, and executes them based on triggers.
- **Service Registry**: Maintains a registry of available services and their capabilities.
- **Service Manager**: Coordinates the invocation of services based on the defined rules.

4. Service Layer

The service layer integrates with internal and external services, providing functionality for triggers, actions, and notifications.

- **Environmental Sensor**: Collects real-time data from custom-programmed sensors.
- **Weather Service**: Fetches weather forecast data from an external API.
- **Building Automation**: Interfaces with the building control system to monitor and control various parameters.
- **Timer Service**: Triggers rules at predefined intervals.
- **Notification Service**: Sends notifications to users based on predefined templates.

5. Data Layer

The data layer is responsible for persistent storage of system data.

- **PostgreSQL**: Stores rule definitions, service configurations, and historical data.

6. External Systems

The architecture integrates with external systems to fetch data or perform actions.

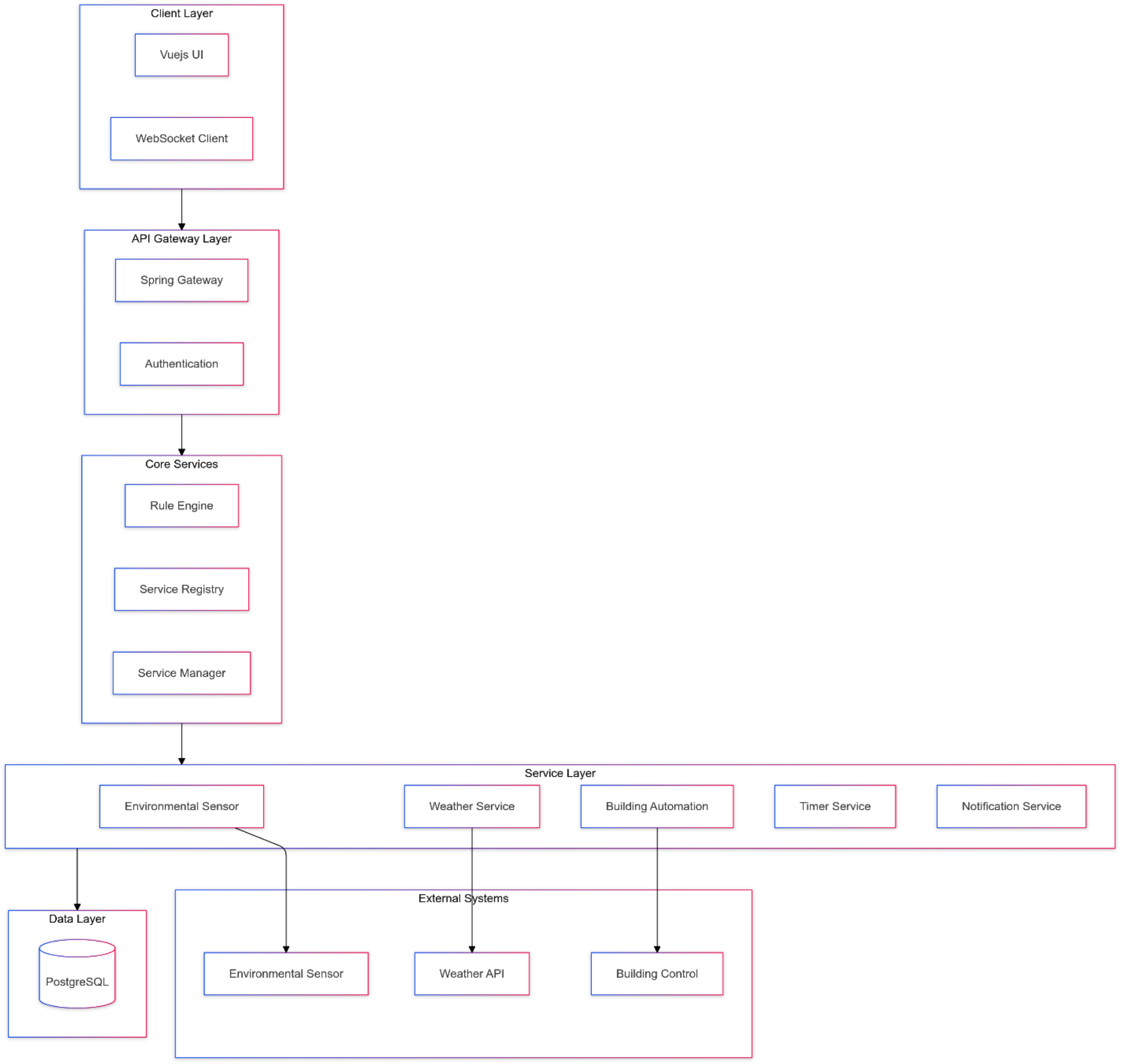
- **Environmental Sensor**: Physical sensors deployed in the environment.
- **Weather API**: A third-party service providing weather data.
- **Building Control**: A system for controlling and monitoring building automation parameters.

Workflow

Rule Creation and Execution

1. **Create Rule**: The user defines a rule through the React UI.
2. **Validate Rule**: The API Gateway forwards the request to the Rule Engine for validation.
3. **Store Rule**: The Rule Engine stores the validated rule in the database via the Service Manager.
4. **Rule Execution**:

- The Rule Engine periodically checks trigger conditions.
- If the trigger conditions are met, the Rule Engine fetches data from external systems or internal services.
- The Rule Engine executes the corresponding action and updates the rule history in the database.



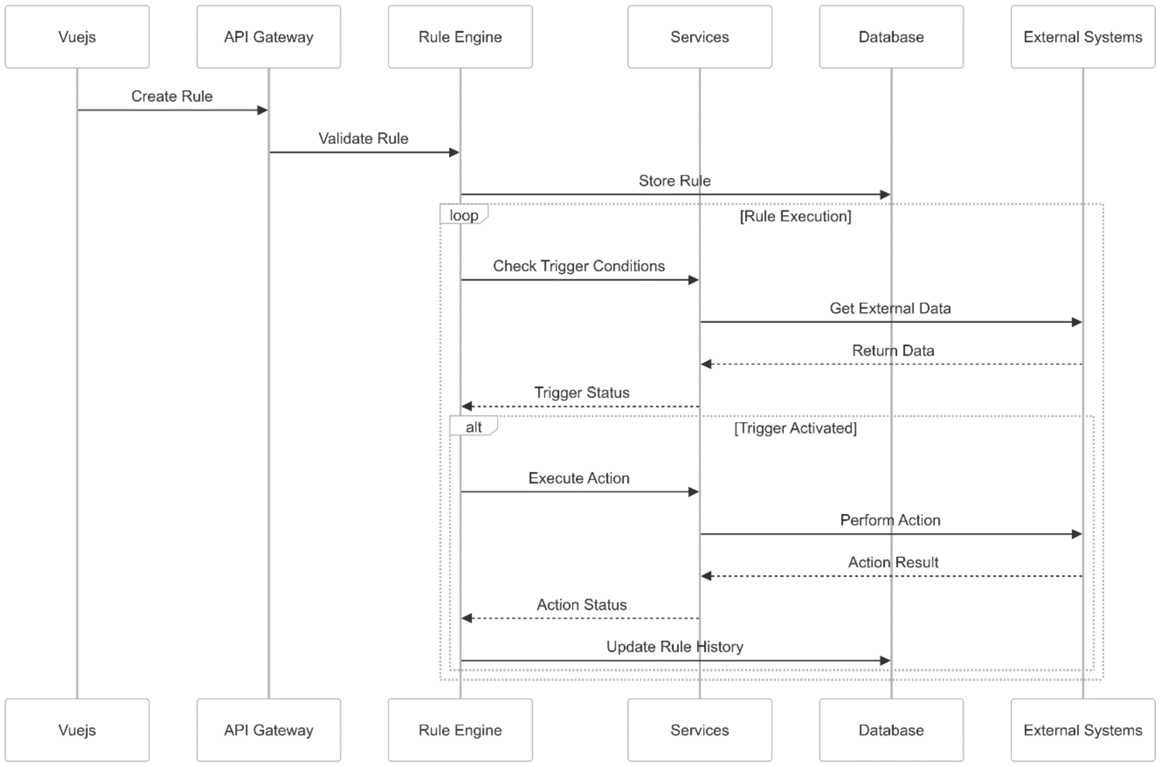
Sequence Diagram

Rule Execution Process

1. **React UI**:

- Sends a request to create a rule to the API Gateway.

2. **API Gateway**:
   - Routes the request to the Rule Engine.
3. **Rule Engine**:
   - Validates the rule and stores it in the database.
   - Periodically evaluates the rule's trigger conditions.
4. **Services**:
   - Fetches data or performs actions as directed by the Rule Engine.
5. **Database**:
   - Stores and retrieves rule definitions and histories.
6. **External Systems**:
   - Provides external data (e.g., weather forecasts) or executes actions (e.g., adjusting building controls).



Summary

This architecture provides a robust and scalable framework for managing rules and integrating multiple services. The layered design ensures separation of concerns, while the use of modern technologies like Spring Gateway, PostgreSQL, and React ensures reliability and maintainability. The system is well-suited for applications in IoT, automation, and rule-based decision-making.