REDES DE COMUNICACIÓN II
PRÁCTICA 3

AUTORES
CARLOS GARCÍA SANTA

GRUPO2323: PAREJA 04

INGENIERÍA INFORMÁTICA
ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD AUTÓNOMA DE MADRID

28/06/2024

Índice

Introducción

Este documento detalla la implementación y funcionalidad de un sistema de gestión de dispositivos IoT que permite la creación, edición, eliminación y visualización de dispositivos, relojes, sensores, interruptores, reglas y eventos asociados. Utilizando Django como framework de desarrollo y SQLite como sistema de gestión de base de datos, se ha desarrollado una interfaz de usuario sencilla y una lógica que interactúa con los dispositivos mediante la librería MQTT.

Implementación

1.1 Componentes Clave

1. **Dispositivos IoT**:

Se ha implementado una clase abstracta en django de la que heredan estos 3 dispositivos, que corresponden a los dispositivos requeridos:

- **Interruptores**: Los interruptores tienen dos estados posibles ‘ON’ y ‘OFF’ este estado se puede manipular a través de los mensajes de MQTT. Los interruptores pueden fallar en este cambio de estado o intentar cambiar a un estado en el que ya están, si este ocurre se manda un mensaje de error, de lo contrario se manda un mensaje de que el cambio se ha realizado correctamente. En todos estos casos se proporciona el ID del interruptor, o el estado/acción que se ha intentado realizar.
- **Sensores**: Los sensores tienen la capacidad de medir un valor float con una unidad de medida asociada, estos sensores mandan un mensaje cada vez que realizan una medición cada cierto tiempo.
- **Relojes**: Los relojes son dispositivos sin estado que emiten su hora a intervalos determinados.

Por otro lado, en django también se han implementado como modelos otros de los componentes clave de la aplicación:

- **Reglas:** Las reglas vienen definidas por: un nombre; una condición, la cual se representa con el nombre del sensor, los símbolos “==”, “<” y “>” y un valor; el sensor asociado a la regla; una acción definida por un estado “ON” u “OFF” y el interruptor asociado; el interruptor asociado.
- **Eventos:** Tienen una marca de tiempo de cuando se han producido, un sensor asociado que ha generado el evento, y el valor que se ha generado en el evento.

2. **Broker MQTT (Mosquitto)**:
   - Actúa como intermediario en la comunicación entre los dispositivos IoT y el controlador.
   - Facilita la suscripción a topics específicos y la publicación de mensajes.
3. **Controller**:
   - Se suscribe a topics de MQTT para recibir actualizaciones de estado de los dispositivos.
   - Cada vez que se recibe un cambio de alguno de los sensores, estos se tratan como eventos que son enviados al Rule Engine. Si el Rule

Engine determina que una regla se cumple el controlador recibe una acción a realizar asociada a un interruptor, y envía el mensaje al interruptor para que este cambie de estado, en el caso de que el interruptor falle se vuelve a enviar un mensaje para que lo reintente.
    - Los mensajes que le llegan son impresos por pantalla para poder observar el flujo de la aplicación
4. **Persistencia**:
    - Utiliza SQLite, gestionada por Django, para almacenar información sobre dispositivos, reglas y eventos generados.
5. **Rule Engine**:
    - Comprueba las reglas almacenadas cuando se reciben eventos y ejecuta las acciones correspondientes asociadas a controladores si las condiciones de las reglas se cumplen.
    - Si no se cumple una regla no se envían mensajes al controlador.

1.2 Interacción del Usuario

La interfaz de usuario proporcionada por Django es accesible a través de un navegador web ejecutando python3 manage.py runserver, permitiendo una gestión intuitiva y visual de todos los componentes del sistema domótico:

- **Página principal**: Ofrece una vista general y permite la navegación a las secciones de dispositivos, reglas y eventos.
- **Gestión de dispositivos**: Interfaces específicas para añadir nuevos dispositivos, editar su configuración o eliminarlos.
- **Gestión de reglas**: Permite la creación, modificación o eliminación de reglas que automatizan las interacciones entre dispositivos basadas en lógicas específicas.
- **Consulta de eventos**: Muestra un registro de todos los eventos generados por el cumplimiento de las reglas o por acciones directas en los dispositivos.

2. Requisitos

Requisitos Funcionales

1. **Gestión de Dispositivos:**
    - El sistema debe permitir a los usuarios añadir, editar y eliminar dispositivos IoT.
    - Cada dispositivo debe ser identificable por un identificador único y asociado a un tipo específico (interruptor, sensor, reloj).
2. **Gestión de Reglas:**
    - Los usuarios deben poder crear, modificar y eliminar reglas que definan la interacción entre dispositivos.

    - Las reglas deben poder especificar condiciones bajo las cuales se activan ciertas acciones en los dispositivos.
3. **Visualización y Gestión de Eventos:**
    - El sistema debe registrar eventos generados por acciones automáticas o manuales sobre los dispositivos.
    - Los usuarios deben poder consultar un historial de eventos.
4. **Interfaz de Usuario:**
    - Debe proporcionar una interfaz clara y accesible para la gestión de dispositivos, reglas y eventos.
    - Debe ser accesible desde un navegador web estándar sin necesidad de software adicional.
5. **Integración MQTT:**
    - El sistema debe poder comunicarse con dispositivos usando el protocolo MQTT.
    - Debe suscribirse a topics específicos para recibir estados de dispositivos y enviar comandos a dispositivos.

Requisitos No Funcionales

1. **Usabilidad:**
    - La interfaz de usuario debe ser intuitiva y fácil de usar para personas sin conocimientos técnicos avanzados.
2. **Rendimiento:**
    - El sistema debe procesar eventos y comandos en tiempo real o con mínima latencia.
3. **Escalabilidad:**
    - El sistema debe ser capaz de manejar un aumento en el número de dispositivos y usuarios sin degradación significativa del rendimiento.

3. Casos de uso

Caso de Uso 1: Control Automático de interruptor de motor (evento de sensor activa interruptor)

Descripción

Este caso de uso es el que se ha simulado e implementado en el simulador, consiste en un sensor de temperatura de un motor y un interruptor de ese mismo motor, cuando el sensor registra una temperatura determinada el interruptor del motor se enciende simulando que el motor se ha intentado arrancar al llegar a una cierta temperatura.

Actores

- **Sensor de temperatura**: Dispositivo IoT que mide la temperatura de un motor.
- **Interruptor Inteligente**: Dispositivo IoT que controla el arranque del motor.
- **Rule Engine:** El motor de reglas.
- **Controlador**: El sistema.

Flujo Principal

1. **Medicion del sensor**: El sensor del motor periódicamente mide la temperatura y envía los datos al controlador a través de su topic MQTT.
2. **Evaluación de Reglas y envió de comandos**: El controlador recibe los datos genera un evento y evalúa la regla preestablecida:
    - Si la temperatura es mayor que el umbral establecido, el Rule Engine genera una acción a realizar asociada al interruptor del motor, que en este caso sería “ON” y el id del interruptor.
    - Si la temperatura no supera el umbral establecido el Rule Engine no envía ningún mensaje al controlador.
3. **Actuación del Interruptor**: El controlador recibe el mensaje del Rule Engine y se lo transmite al interruptor determinado, este recibe el comando y actúa en consecuencia, cambiando su estado a “ON”.

Postcondiciones

- El interruptor del motor esta en estado “ON”
- Los eventos de cambio de estado se registran en el sistema para futura referencia.

Decisiones de diseño

1. ¿Controller y Rule engine han de ser aplicaciones separadas?, ¿por qué?, ¿qué ventajas tiene una y otra opción?

Separar el *Controller* y el *Rule Engine* en aplicaciones distintas ofrece claridad funcional y modularidad. Esto permite que cada componente se concentre en responsabilidades específicas: el *Controller* maneja la interacción directa con los dispositivos y el *Rule Engine* procesa la lógica de las reglas para tomar decisiones basadas en los datos recibidos. Sin embargo, esto no tiene porque ser necesariamente así, se podría hacer una bloque monolítico donde el controlador gestione las reglas, este enfoque podría resultar más fácil y sencillo de implementar, pero aportaría menor claridad y escalabilidad, por lo tanto se ha descartado.

2. ¿Cómo se comunican Controller y Rule Engine en la opción escogida?

Controller y Rule Engine se comunican a través de MQTT, utilizando los topics /rule_engine y /rule_engine_send específicos para recibir y enviar mensajes, respectivamente, desde el controlador.

3. ¿Cuántas instancias hay de cada componente?

Se ha implementado una única instancia de cada uno, pero realmente se podría implementar una cola de trabajadores con múltiples Rule Engine para agilizar el procesado de eventos.

Conclusiones

El desarrollo de este sistema IoT utilizando MQTT y Django ha sido complejo, utilizar MQTT para la comunicación entre los dispositivos IoT ha funcionado muy bien debido a su simplicidad y eficiencia, pero ha costado entenderlo con la profundidad necesaria para implementarlo correctamente junto con Django, sin embargo, la decisión de separar las funcionalidades del sistema en diferentes componentes, como el Controller y el Rule Engine, ha facilitado mucho las cosas, permitiéndome gestionar mejor cada parte del sistema de forma independiente.