1. Introducción

En esta práctica se va a implementar un sistema domótico para el hogar con dispositivos IoT.

Un sistema de domótica de hogar permite conectar distintos tipos de dispositivos como interruptores, sensores o motores para realizar acciones cotidianas como encender una luz, medir una temperatura o subir una persiana. Estos sensores y actuadores se usan en conjunto con reglas que permiten hacer cosas como: "encender la luz de la puerta al anochecer", "cerrar las persianas al pulsar un botón", "encender la caldera si la temperatura baja de 21 grados", etc.

Existen múltiples protocolos para comunicación IoT abiertos e incluso algunos fabricantes implementan el suyo propietario, pero en este caso nos centraremos en MQTT, de los más usados.

Se proponen dos opciones para que el usuario interactúe con el sistema:

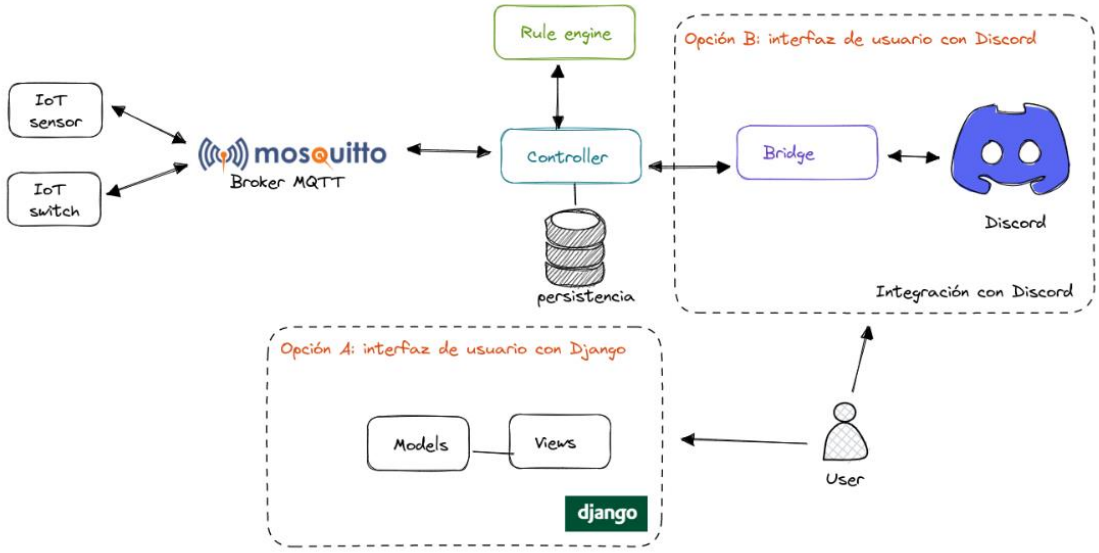
- Opción A, Django

  Proyecto de Django que permita gestionar dispositivos, reglas y eventos generados.

- Opción B, Discord

  Desarrollar un bot de Discord que permita, a través de mensajes publicados en una sala y leyendo acciones sobre el bot, actúar sobre los dispositivos, reglas y eventos del sistema.

En el siguiente esquema se pueden observar los distintos componentes que lo forman:



Consta de varios componentes que se explicarán en detalle más adelante:

- Dispositivos IoT

    - Sensores
    - Interruptores
    - Relojes

- Broker MQTT Mosquitto
- Controller Gestión del sistema general: lee mensajes del broker, llama al Rule engine y realiza acciones sobre los dispositivos a través del broker.
- Persistencia

  Almacena la información de los dispositivos registrados y las reglas.

- Rule engine

  Entidad que, ante un evento, comprueba las reglas y lanza acciones

- Bridge

  (Sólo en el caso de escoger la opción B)

  Aplicación puente que comunica la aplicación web con Discord

- Aplicación de gestión

  (Sólo en el caso de escoger la opción A)

  Aplicación web basada en Django para gestionar los dispositivos, su estado

- Discord

  Es una plataforma que soporta, entre otras cosas, la creación de salas de chat de texto, vídeo y audio

Y los siguientes actores:

- Dispositvo IoT (device)

  Publican sus cambios de estado y pueden recibir acciones a realizar usando MQTT

- Controlador (controller)

  Recibe y envía mensajes a los dispositivos IoT usando MQTT. Permite gestionar los dispositivos

- Traductor (bridge)

  Recibe eventos y los traduce a acciones para el bot. Recibe acciones del bot y las traduce a acciones sobre los dispositivos.

- Bot

  Publica mensajes en Discord y recibe acciones

Dispositivo IoT

Se van a soportar de tres tipos, interruptores, sensores y relojes, para los que habrá que implementar al menos uno de cada tipo.

Para cada funcionalidad que soporta un dispositivo se define un topic y las cadenas de los distintos estados. Un ejemplo para un interruptor:

- topic: `home/climate/boiler_switch`
  - state_on: `ON`
  - state_off: `OFF`
- command_topic: `home/climate/boiler_switch/set`
  - payload_on: `ON`
  - payload_off: `OFF`
- ...

Si se quiere consultar el estado del interruptor se publicará un mensaje en topic y el dispositivo responderá con uno de los estados. Si se quiere cambiar el estado se publicará un mensaje en el topic command_topic con el payload correspondiente y el dispositivo responderá con el estado al que ha cambiado.

Para los sensores, el dispositivo puede publicar cada cierto tiempo (o con cada cambio) el cambio de estado y lo publica en su topic. Es posible también preguntar por su estado como en el ejemplo anterior.

Controlador

Es un suscriptor y publicador en MQTT para recibir los cambios de estado y realizar cambios en los dispositivos. Persiste los cambios en los dispositivos y genera eventos.

Permite recibir acciones a realizar sobre los dispositivos IoT

Rule engine

Permite gestionar las reglas del sistema, recibir eventos y comprobar las reglas para realizar acciones

Opción Django

Proyecto Django que presenta vistas para gestionar los dispositivos, reglas y consultar los eventos generados.
Se puede usar la parte de persistencia que viene con Django, los modelos, como persistencia para todo el proyecto.

No es necesario que implemente autenticación ni que que las vistas tengan una apariencia determinada mediante CSS.

Opción Discord

Bridge

Hace de traductor entre la integración con Discord y el sistema. Estará conectado con Discord para poder recibir acciones y para publicar cambios de estado.

Bot

Recibe los mensajes de acciones y publica mensajes en la sala para la que está configurada.

Broker MQTT

Usaremos Mosquitto, el broker MQTT más ligero y popular

Dudas frecuentes

- ¿Qué persistencia se recomienda usar?

  Si se usa Django, lo más sencillo es usar SQLite, que lo soporta de serie. Aunque se opte por la opción Discord, se podría usar sólo la capa de persistencia de Django con SQLite. Otra opción es usar directamente ficheros para almacenar la información en un formato a definir por el el alumno: json, yaml, etc.

- ¿Tiene que haber algún prefijo en los topics a usar?

  Sí, al usar un broker compartido cada dispositivo tendrá un topic con la forma:

  ```
  redes2/GRUPO/PAREJA/ID
  ```

  Donde:

  - redes2/ Es fijo
  - GRUPO Son los 4 dígitos del grupo al que pertenece el alumno p.ej. 2303
  - PAREJA Son los dos dígitos de la pareja p.ej. 01
  - ID Identificador del dispositivo

Recomendaciones de desarrollo

- Entender bien el alcance del proyecto y escribirlo: requisitos y casos de uso
- Resolver dudas de diseño
- Documentar el diseño

- Pensar en qué se ha de probar para que cumplan con los casos de uso: si el sistema tiene que hacer A, ¿cómo me aseguro de que es así de manera reproducible?
- Reutilizar lo más posible código común encapsulado en clases:
    - Conexión con el broker

      ¿Clase abstracta, funciones comunes?

    - Recepción y envío de mensajes
    - Persistencia
- Empezar probando desde lo más pequeño e ir ampliando

  En los ejemplos anteriores se prueban primero las partes de conexión con el broker, luego recepción y envío de mensajes, clases que los usan, etc

- Ante un comportamiento no esperado, volver atrás en las pruebas hasta descubrir qué cambio ha provocado el error. Si se tienen pruebas es fácil volver atrás y determinar el origen

Implementación

TODO: Completar por el alumno

Conclusiones

TODO: Completar por el alumno

2. Objetivos de aprendizaje

1. Conocer uno de los principales protcolos de comunicación IoT: MQTT
2. Aprender a gestionar un sistema con varios componentes definidos y la interacción entre ellos, sobre todo si algunos de ellos no los hemos desarrollado nosotros

3. Afianzar los conocimientos adquiridos en la asignatura sobre comunicación asíncrona, aplicaciones distribuídas y lenguaje Python
4. Continuar aprendiendo a probar mientras se desarrolla
5. Usar uno de los frameworks web más populares: Django
6. Crear un bot en una plataformas con un API abierto
    1. Familiarizarse con OAuth2

3. Funcionalidad

- (1) Gestión de dispositivos IoT: se tiene que poder añadir un dispositivo con su identificador, tipo, editar y borrar

Para añadir un dispositivo, dependiendo de la opción escogida, se podrá hacer externamente creando el fichero y notificando al sistema que se ha añadido un nuevo dispositivo

- (2) Gestión de reglas básicas: dado-cuando-entonces para los disparadores soportaremos los operadores ==, > y <

Bastará con definir un formato textual que el sistema tratará p.ej. "si sala mayor que 25 enciende caldera"Configuración del sistema: conexión MQTT, topics por defecto, persistencia, contraseñas para API

- (3) Registro de un dispositivo: mediate su id, si está registrado se procesan sus mensajes y se pueden realizar acciones sobre él. En caso contrario se rechazan.
- (4) Un dispositivo puede:

1. comunicar un cambio de estado
2. Recibir una acción si lo soporta

- Conectar con el broker MQTT
- Generar eventos internos
- Comprobar reglas para ver si un evento encaja y se desencadena una acción
- Realizar acciones sobre dispositivos Cambiar el estado de un interruptor

Django

1. Consultar, añadir, editar y borrar dispositivos
2. Consultar, añadir, editar y borrar reglas
3. Consultar eventos internos del sistema

Discord

1. Gestionar credenciales de API Discord
2. Publicar cambios de estado en una sala a través del bot
3. Consultar, añadir, editar y borrar reglas
4. Recibir acciones sobre un dispositivo a través del bot
5. Enviar eventos como mensajes a una sala

Limitaciones

- Un dispositivo en sí no tiene persistencia, al arrancar elige un estado y lo comunica
- Los eventos que se comprueban contra las reglas son efímeros, si el rule engine cae y se recupera, no procesa eventos antiguos

Requisitos

Traducir a requisitos la funcionalidad a implementar

TODO: Completar por el alumno

Casos de uso

TODO: Completar por el alumno

Decisiones

- ¿Controller y Rule engine han de ser aplicaciones separadas?, ¿por qué?, ¿qué ventajas tiene una y otra opción?
- ¿Cómo se comunican Controller y Rule engine en la opción escogida?
- ¿Tiene sentido que alguno de estos componentes compartan funcionalidad?, ¿qué relación hay entre ellos?
- ¿Cuántas instancias hay de cada componente?
- ¿Controller y Bridge han de ser aplicaciones separadas?, ¿por qué?, ¿qué ventajas tiene una y otra opción?
- ¿Cómo se comunican Controller y Bridge en la opción escogida?

TODO: Completar por el alumno

4. Entregables

Documentación

- Completar las partes que faltan en este documento, incluyendo las decisiones de diseño (razonadas)
- Especificación de comunicación con cada tipo de sensor

- Esquema de datos usado en la persistencia
- Clases principales y relación entre ellas, herencia. Puede ser en formato esquema o texto
- Limitaciones de la solución escogida

  Nuestro sistema no es capaz de hacer esto por esta razón

Código

- `dummy-switch.py`

  Ejecutable que instancia un interruptor. Recibe como argumentos:

  - Datos para conectar al broker
    - Host

      `--host`

      Por defecto `redes2.ii.uam.es`

    - Port

      `-p,--port`

      Por defecto `1883`

  - Probalidad de fallo al recibir una acción

    `-P,--probability`

    Por defecto 0.3 (falla 3 de cada 10 veces)

  - Id del dispositivo (cliente del broker)

    `Id`

  Ejemplo de llamada:

  ```
  dummy-switch.py --host redes2.ii.uam.es --port 1883 --probability 0.4
  1
  ```

- `dummy-sensor.py` Aplicación que instancia un sensor y envía cambios de estado cada cierto tiempo entre dos valores máximo y mínimo. Similar al anterior salvo que envía cambios de estado cada cierto tiempo configurable. Recibe como argumentos:
  - Tiempo entre envíos Tiempo en segundos tras los que simula un cambio de estado

`-i,--interval`

Por defecto 1

- Valor mínimo Valor mínimo a enviar

  `-m,--min`

  Por defecto 20

- Valor máximo

  Valor máximo a enviar `-M,--max`

  Por defecto 30

- Incremento

  Incremento entre min y max

  `--increment`

  Por defecto 1

- Ejemplo de llamada:
- `dummy-sensor.py --host redes2.ii.uam.es --port 1883 --min 20 --max 30 --increment 1 --interval 1 2
- `dummy-clock.py`
  - --time

Hora de inicio, de la que se parte, en formato: HH:MM:SS. P. ej. Si `--time 09:00:00` el sistema empieza a enviar 09:00:00, 09:00:01, etc. Si no se indica, se usa la hora actual del sistema

- --increment

Incremento entre envíos en segundos. P. ej. si 60 envía cambios de 60 segundos desde la hora de inicio (09:00:00, 09:01:00, ...). Si no se indica, es 1

- --rate

Frecuencia de envío en segundos. P. ej. si se pone 10, envía 10 mensajes por segundo. Si no se indica, es 1

- `controller.py`

  Aplicación que conecta con el broker, recibe mensajes de los dispositivos y les envía mensajes a través del broker Instancia un rule engine

  Recibe como argumentos:

    - Datos para conectar al broker
        - Host

          `--host`

          Por defecto `redes2.ii.uam.es`

        - Port

          `-p,--port`

          Por defecto `1883`

        - Database

          Al ser SQLite, el nombre del fichero a conectar. Dependiendo de la opción escogida para usar la persistencia puede variar.

- `rule-engine.py`

  En el caso de optar por aplicación separada de `controller.py`. Recibe opciones similares a éste.

- `bridge.py`

  En el caso de optar por opción Discord y aplicación separada de `controller.py`. Recibe opciones similares a éste.

En caso de usar un proyecto Django, estos ejecutables pueden integrase en el proyecto en un directorio auxiliar tipo `tools`

- `tests`

  Un directorio con, al menos, las pruebas pedidas.

- `simulator.sh`

  Script que lance a todos los actores y simule cambios de estado para comprobar que todo funciona. Sería el equivalente a un tester de pruebas de integración.

Configuración

Todo lo necesario para poder ejecutar el sistema p.ej. si se opta por Discord, tokens, urls, etc. necesarios para poder hacer funcionar el sistema

Si se usa SQLite, se puede entregar el fichero con el esquema ya cargado pero no tiene que tener ninguna entrada, ningún dispositivo registrado, regla, etc.

Pruebas

Se han de entregar, al menos, los siguientes tests unitarios:

- `test_device.py`
  - Conecta correctamente con el broker
  - Si no conecta con el broker da error
  - Probar que el sistema lee bien los parámetros por línea de comandos
  - (switch) Cambia de estado ante una acción
  - (sensor) Cambia de estado en intervalos entre min y max
- `test_controller.py`
  - Conecta correctamente con el broker
  - Si no conecta con el broker da error
  - Ante un mensaje de sensor, desencadena el mecanismo para comprobar reglas
  - Ante una respuesta de RuleEngine para realizar una acción, realiza la acción sobre el dispositivo
  - Se lee correctamente la información de los dispositivos de la persistencia

5. Referencias

- Discord
  - Página principal
  - Portal del desarrollador
  - Crear un bot de Discord
  - discord.py, librería para interactuar con Discord en Python
- MQTT
  - Página principal de MQTT
  - Cómo funciona MQTT
  - Mosquitto
  - Instancia de test Mosquitto de Eclipse: http://test.mosquitto.org
  - Paho, librería Python para MQTT https://pypi.org/project/paho-mqtt/
- Django
  - Django
  - Tutorial de Django
- Unit test Python
  - Página principal
- Mock
  - Página principal
- Persistencia
  - SQLite
  - SQLAlquemy

6. Recursos

- Plantillas para aplicación Django

https://moodle.uam.es/pluginfile.php/4436702/mod_book/chapter/10723/plantillas-django-1.0.tar.gz