Documento de diseño

Switch

El programa dummy_switch.py describe la clase Switch que simula un dispositivo con dos estados, encendido y apagado, ya sea una lámpara o una caldera. Su único objetivo es encenderse o apagarse cuando le llega un mensaje de encendido o apagado respectivamente. También tiene la función de toggle que es cambiar de encendido a apagado y viceversa. Cada acción tiene un porcentaje de fallo que se puede introducir por terminal.

Sensor

El programa dummy_sensor.py describe la clase Sensor que simula cualquier tipo de sensor que devuelva valores numéricos, como uno de temperatura o de luz. Se le puede introducir un valor máximo, mínimo y un incremento, que básicamente es un contador cíclico (vuelve al mínimo cuando sobrepasa el máximo) desde el mínimo al máximo en saltos del valor especificado en el incremento.

Clock

El programa dummy_clock.py describe la clase Clock que simula un reloj. Se le puede introducir una hora de inicio por defecto. Básicamente, manda mensajes cada cierto intervalo (también especificable) con la hora con cierto incremento.

Django

Modelos

Actuator

Los actuadores son clases abstractas que engloban a todos los dispositivos, ya que estos tienen algunos atributos en común como el nombre o el tema.

Switch

Representa un dummy_switch pero no tiene su comportamiento, es decir no es él quien cambia ya que solo sería un valor en una tabla y no representaría un elemento real.

Sensor

Representa un dummy_sensor pero no tiene su comportamiento, es decir no es él quien cambia ya que solo sería un valor en una tabla y no representaría un elemento real.

Clock

Representa un dummy_clock pero no tiene su comportamiento, es decir no es él quien cambia ya que solo sería un valor en una tabla y no representaría un elemento real.

Rule

Define una regla, que contiene principalmente, un activador, un switch, un threshold, y un tipo. El activador puede ser un sensor, un switch o un clock. El switch de la rule es lo que se activará o apagará dependiendo del valor del activador. La threshold es el valor límite donde cambia un elemento y el tipo es una flag que puede ser o 1 o 0, esta flag simplemente representa si el switch se activa cuando el activador está por encima o por debajo de la threshold.

Controller

Gestiona todos los dispositivos del sistema de domótica como los switches, clocks y sensores. Recibe todos los mensajes y actúa en consecuencia aplicando las reglas. Para ello cada vez que llega un mensaje de un sensor, un reloj o un switch comprueba todas las reglas. Filtra por activador (que sea el mismo que ha mandado un mensaje al controlador) y comprueba el tipo de la regla y si procede, manda una orden de encendido o apagado al switch de la regla.

Test

Para probar el servicio se tienen varias opciones, el simulate.sh que dura uso 17 segundos, este, lanza un controlador, un sensor y un switch. Además, lanza el programa rule.py que crea una regla entre el sensor y el switch.
Otra forma de probar es lanzar el programa tests/test_devices.py y los dummies que quieras, este recibirá todos los mensajes y los mostrará por pantalla.
Además, también se puede probar usando el tests/test_controler.py que comprueba el controlador.
Por último, siempre se pueden lanzar los dummies y el controlador a mano por consola de comandos además se tendrá que lanzar el servidor de Django con el comando make runserver que lanzará el servidor en el localhost:8001. Si accedes a esa dirección, saldrá una tabla con estado de las reglas que se actualizan en tiempo real y se pueden crear, editar o borrar.

Cosas para tener en cuenta

- Un switch no puede activarse a sí mismo, si se quiere usar un switch como activador, tiene que haber una regla previa que lo active.
- Si se hace make se borra la base de datos, y las migraciones y se crean desde 0. Además, crea un super usuario con usuario y contraseña ***alumnodb*** para acceder a la consola de administración.
- La persistencia se hace cada vez que el controlador recibe un mensaje creando o modificando los datos del modelo Django que representa el dispositivo.
- Los mensajes contienen toda la información de la clase que manda el mensaje en formato de diccionario pasado a bytes con json.dumps() además de un mensaje, por ejemplo para que un switch avise de su estado manda:

```
message_dict = {
            'aid': self.aid,
            'name': self.name,
```

```
            'theme': self.theme,
            'state': self.state,
            'broker_host': self.broker_host,
            'port': self.port,
            'fail': self.fail,
            'message': message,
        }
```

- Ningún dispositivo persiste su propio estado por sí solo ya que usamos mqtt para que el sistema sea liviano y que por ejemplo una lampara tuviera que editar la base de datos no tendría sentido.

Limitaciones

Nuestro sistema de domotica tiene algunas restricciones, por ejemplo:

- El estado de los dispositivos es una cadena de texto ya que al heredar todos de Activator y el clock tener una hora como estado, nos pareció que era lo más adecuado
- Un dummy_switch no puede lanzarse antes del controlador o no quedará registrado en la base de datos ya que la persistencia la hace el controlador. El switch cuando se arranca manda su estado y ahí el controlador persiste el switch, si el controlador no está arrancado, no recibe el mensaje.
- El sistema no concibe reglas contradictorias como enciéndete cuando el sensor llege a 25 y apágate cuando el sensor llegue a 25, ya que nos parecía fuera del alcance de esta práctica