Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación Android para la gestión de un invernadero inteligente con control por voz utilizando OpenHAB y Kafka

Autor: Jesús Vinuesa Serrano
Tutora: María Teresa Ariza Gómez

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación Android para la gestión de un invernadero inteligente con control por voz utilizando OpenHAB y Kafka

Autor:
Jesús Vinuesa Serrano

Tutora:
María Teresa Ariza Gómez

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Trabajo Fin de Grado: Aplicación Android para la gestión de un invernadero inteligente con control por voz utilizando OpenHAB y Kafka

Autor: Jesús Vinuesa Serrano

Tutora: María Teresa Ariza Gómez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Sevilla, 2023

A mi gran familia

Agradecimientos

En este pequeño apartado se me permite declarar de forma resumida unos agradecimientos hacia todas aquellas personas cercanas que me han influido positivamente durante la carrera. Por ello, me gustaría remarcar que estas palabras son solo una pequeñísima parte del gran sentimiento de gratitud que tengo hacia cada una de las personas que voy a nombrar a continuación.

Dichos agradecimientos tienen que empezar dirigiéndose a tres personas que conforman la base de mi vida y son los principales responsables de que hoy me encuentre escribiendo estas palabras: mis padres y mi hermano. Ellos han sabido escucharme en cada uno de mis momentos, tanto buenos como malos, me han aconsejado en multitud de ocasiones y me han dado los empujones necesarios cuando las cosas no se veían agradables, todo ello de manera incondicional. Gracias.

El paso por la universidad me ha dado la oportunidad de conocer a nuevas personas las cuales han hecho que éste fuera mucho más ameno, sobre todo en estos últimos años. Agradecer especialmente a Ishika y Marcelino por todas las vivencias compartidas durante este recorrido. Agradecer también a Diego y Nacho, mis compañeros de toda la vida.

Agradecer de todo corazón a mi tutora, María Teresa Ariza Gómez, por haberme brindado la oportunidad de realizar este trabajo. Por las extensas reuniones, por los consejos y por todos los conocimientos impartidos en sus clases, gracias.

Me gustaría acabar recordando a aquellos integrantes de mi familia que ya no están conmigo. Agradecerles, desde aquí, cada segundo que he vivido junto a ellos.

Jesús Vinuesa Serrano

Sevilla, 2023

Resumen

Con el paso del tiempo, el mundo se ha vuelto más digitalizado gracias a Internet. En el transcurso de los días, miles son los elementos que usamos de forma cotidiana y que se encuentran conectados a dicha tecnología intercambiando información. Es el llamado Internet de las cosas (IoT).

Este trabajo pretende poner de manifiesto la gran importancia del IoT en nuestra época. Con él se podrá gestionar y controlar exhaustivamente las condiciones ambientales existentes en un invernadero a través de una aplicación Android. El uso de la voz para su control en el caso de querer manipular el dispositivo móvil lo menos posible, así como la vigilancia del agua empleada para el riego de los cultivos son algunas de las funcionalidades de esta app. Permitirá, entre otras cosas, aprovechar todos los recursos disponibles, con el menor costo y contaminación ambiental posibles.

Entre las tecnologías utilizadas destacan OpenHAB, núcleo del proyecto encargado de la automatización del sistema y Kafka, responsable del envío de mensajes entre OpenHAB y Voice2JSON. Este último será el programa utilizado para la transcripción de comandos de audio a texto, lo que permitirá el empleo de la voz humana.

También se hará uso de MQTT, protocolo de comunicación empleado para la interconexión entre OpenHAB y la placa ESP32, configurada por medio de Arduino.

Abstract

Over time, the world has become more connected thanks to the Internet. Over the course of every day, thousands of elements that we use on a daily basis are connected to this technology. This is the so-called Internet of Things (IoT).

This work aims to highlight the great importance of IoT in our time. With it, it will be possible to comprehensively manage and control the environmental conditions in a greenhouse through an Android application. The use of voice control in the case of wanting to manipulate the mobile device as little as possible, as well as the monitoring of the water used for irrigation of crops are some of the features of this app. It will allow, among other things, to take advantage of all available resources, with the lowest possible cost and environmental pollution.

The technologies used include OpenHAB, the core of the project responsible for system automation, and Kafka, responsible for sending messages between OpenHAB and Voice2JSON. The latter will be the program used for the transcription of audio commands to text, which will allow the use of the human voice.

We will also make use of MQTT, the communication protocol used for the interconnection between OpenHAB and the ESP32 board, configured by Arduino.

Índice

Índice de Tablas

Índice de Figuras

Notación

MQTT	Message Queuing Telemetry Transport
OpenHAB	Open Home Automation Bus
JSON	JavaScript Object Notation
IoT	Internet of Things
SQL	Structured Query Languaje
REST	Representational State Transfer
OS	Operating System
PC	Personal computer
USB	Universal Serial Bus
UART	Universal Asynchronous Receiver/Transmitter
BLE	Bluetooth Low Energy
RAM	Random Access Memory
SoC	System on a Chip
PCB	Printed Circuit Board
MIT	Massachusetts Institute of Technology
IPA	International Phonetic Alphabet
JSFG	JSpeech Grammar Format
MFCC	Mel Frequency Cepstral Coefficients
G2P	Grapheme to Phoneme
FST	Finite State Transducer
ASF	Apache Software Foundation
CSV	Comma Separated Values
API	Application Programming Interfaces
JVM	Java Virtual Machine
JDBC	Java Database Connectivity
OAuth	Open Authorization
UTF-8	8-bit Unicode Transformation Format
M2M	Machine to Machine
QoS	Quality of Service
I2C	Inter Integrated Circuit
SPI	Serial Peripheral Interface
LGPL	GNU Lesser General Public License
GPL	GNU Public License
PCM	Pulse Code Modulation
WAV	Waveform Audio Format
HTTP	HyperText Transfer Protocol
PWM	Pulse Width Modulation
ROM	Read Only Memory
URL	Uniform Resource Locator

GPIO	General Purpose Input/Output
RGB	Red Green Blue
ADC	Analog to Digital Converter
DAC	Digital to Analogue Converter
SRAM	Static Random Access Memory
ISR	In Sync Replica
VCP	Virtual Com Port
MSI	Microsoft Windows Installer

1 INTRODUCCIÓN

"Si quieres ser feliz toda una vida pon un huerto, da tranquilidad al cultivar, satisfacción al recolectar y salud al consumir sus cosechas…"

Proverbio chino

1.1 Motivación del trabajo

El internet de las cosas (IoT) describe la red de objetos físicos, las "cosas", que llevan incorporados sensores, software y otras tecnologías con el fin de conectarse e intercambiar datos con otros dispositivos y sistemas a través de la red pública o privada. Estos dispositivos van desde objetos domésticos comunes, como bombillas, hasta herramientas industriales sofisticadas, como los robots industriales, y se pueden clasificar dentro de dos grupos distintos: actuadores (es decir, envían las instrucciones a un objeto) o sensores (recopilan datos y los envían a otro lugar). Con más de siete mil millones de dispositivos IoT conectados en la actualidad, se prevé que este número aumente a veintidós mil millones para el año 2025.

La idea de IoT existe desde hace mucho tiempo. Sin embargo, una colección de avances tecnológicos recientes ha permitido que sea utilizado actualmente de forma muy amplia:

- Tecnología de sensores de bajo costo y potencia. Sensores asequibles y fiables hacen posible el IoT para más fabricantes.
- Conectividad. La gran cantidad de protocolos de red para Internet facilita la conexión de sensores a la nube y a otros sistemas para lograr una transferencia de datos eficiente.
- Plataformas de informática en la nube. El aumento de la disponibilidad de plataformas en la nube permite a las empresas y a los consumidores acceder a la infraestructura que necesitan para escalar sin tener que administrarlo todo.
- Aprendizaje automático y analítica. Esto, junto con el acceso a cantidades grandes y variadas de datos almacenados en la nube, hacen que las empresas y consumidores puedan recopilar información de forma más rápida y fácil.
- Inteligencia artificial (IA) conversacional. Los avances en las redes neuronales han llevado el procesamiento del lenguaje natural (PLN) a los dispositivos IoT (como los asistentes personales Alexa, Cortana o Siri), siendo atractivos, asequibles y viables para el uso doméstico.

Si hablamos del uso generalizado que se les puede dar a los dispositivos IoT, podremos clasificarlos en seis subgrupos:

- IoT para consumo: es decir, para el uso cotidiano: domótica, asistentes de voz, etc.
- IoT comercial: habitual en el sector salud o en la industria del transporte. Por ejemplo, un

marcapasos cardíaco inteligente, un tracker de paquetería, etc.

- IoT militar (IoMT): destinado para el ámbito militar: robots de vigilancia, wearables biométricos para monitorizar soldados en campo, etc.
- IoT industrial (IIoT): usos típicos en el sector manufacturero y energético. Por ejemplo, sistemas digitales de control, smart agricultura, análisis de datos industriales, etc. En ocasiones, el IIoT recibe el nombre de cuarta ola de la revolución industrial, o bien Industria 4.0.
- IoT de infraestructuras: usados para crear Smart cities, Smart buildings, etc. Con todo tipo de sensores de medición y sistemas de control, incidencias, mantenimiento predictivo, etc.
- IoT en medicina (IoMT): usado para monitorizar y conocer el estado de los pacientes

En este trabajo se abordará el subgrupo IIoT, concretamente, la agricultura inteligente.

La tecnología ha jugado un papel fundamental en el desarrollo de este sector en las últimas décadas. Según [24], el primer avance tecnológico introducido, hace 19 años, fue la automatización de los sistemas de fertilización. Posteriormente, le siguió otra revolución: la implementación de sistemas de control de plagas. Y por último la digitalización a gran escala.

La digitalización y la conectividad de los invernaderos tiene dos grandes consecuencias: cultivar más y mejor y minimizar el impacto de la agricultura en el medio ambiente. Un menor consumo de agua y energía, la aplicación precisa de fertilizantes (evitando la contaminación del suelo y los acuíferos) y la disminución del uso de productos químicos permiten una producción más ecológica y sostenible desde el punto de vista medioambiental.

Para cubrir la demanda mundial en 2050, el sector agrícola necesitará producir el 50% más de alimentos que a principios de la década pasada. A pesar de que en el pasado ya se hizo frente al crecimiento importante de la demanda de comida (últimas décadas del siglo XX), las proyecciones de la Organización de las Naciones Unidas para la Alimentación y la Agricultura (FAO) señalan que el cambio climático, la falta de inversión y el retraso en la digitalización del campo pueden poner en riesgo el sistema de alimentación global. Por lo que, para dicha organización, el aumento de la financiación y de la tecnología son clave para el crecimiento sostenible de la actividad agrícola.

1.2 Objetivos y enfoque

Este trabajo tendrá los siguientes objetivos:

- Análisis y funcionamiento de OpenHAB.
- Funcionamiento del bróker Kafka junto con OpenHAB.
- Funcionamiento con comandos por voz.
- Instalación de sensores y actuadores con Arduino y conectividad con el bróker MQTT.
- Creación del servidor REST.
- Creación del cliente móvil.
- Conexión de la base de datos con el servidor REST.

1.2.1 Objetivos específicos

- Lectura de los datos procedentes de los sensores y envío de órdenes a los actuadores: se leerán los datos recibidos de los sensores Arduino y se enviarán órdenes que controlarán los actuadores. Se usarán los siguientes elementos:
    - Sensores
        - Sensor de temperatura y humedad DHT11: para la medición de la temperatura y la humedad del ambiente.
        - Fotorresistencia: para la medición del nivel de luz.
        - Sensor de nivel resistivo: para medir el nivel de agua.
        - Sensor de humedad del suelo: para la medición de la humedad terrestre.
    - Actuadores
        - Motor de agua: motor utilizado para la extracción de agua de un depósito. Este motor estará conectado a un relé de 5V para que pueda funcionar correctamente.
        - Dos LEDs: se utilizarán dos LEDs que simularán un aire acondicionado y un humidificador
        - Bombilla Xiaomi Mi Smart LED Bulb Essential (White and Color): es un actuador del sistema, pero no está conectado a Arduino, sino directamente a OpenHAB.
- Instalación de la placa ESP32: instalación y configuración de la placa ESP32 que estará conectada a los sensores y actuadores anteriormente mencionados (excepto la bombilla Xiaomi) para su gestión.
- Conexión de la placa ESP32 con OpenHAB: se conectará la placa ESP32 a un bróker MQTT público para poder enviar los datos recibidos de los sensores o controlar los actuadores según las ordenes enviadas por el usuario. OpenHAB se conectará al bróker público a través del binding MQTT.
- Instalación y configuración de OpenHAB: se instalará el programa para la gestión automatizada del invernadero, y se configurarán, entre otras cosas, reglas personalizadas.
- Control de la bombilla Xiaomi a través de OpenHAB: se utilizará el binding Xiaomi Wifi devices (Mi IO) para ello.
- Instalación de Kafka y conexión con OpenHAB: se instalará y configurará Kafka para que pueda enviar los mensajes recibidos en su bróker a OpenHAB.
- Instalación y configuración del programa de conversión audio-texto Voice2JSON: se instalará y se configurará el programa Voice2JSON para convertir los comandos por voz recibidos por el usuario a texto y enviarlos al bróker de Kafka.
- Creación y configuración de un servicio REST: implementado usando SPRING, el servicio REST se encargará de:
    - La autentificación de los usuarios: para ello comprobará el usuario y la contraseña introducidas por el usuario y se compararán con las que estén almacenadas en la base de datos PostgreSQL.

    - Consulta de las peticiones del cliente: será el servicio REST quien consulte todas las peticiones recibidas por el cliente.
- Conexión de OpenHAB con el servicio REST: para ello se usará el binding de HTTP. Esto permitirá a OpenHAB saber cuáles son los límites ambientales establecidos por el usuario.
- Creación y configuración de la aplicación móvil para el usuario: configurado a través de Android Studio, la aplicación de usuario se conectará al servicio REST para poder gestionar y controlar completamente el funcionamiento del invernadero. El usuario tendrá la opción de enviar comandos por voz (para ello, el dispositivo utilizado debe disponer de un micrófono incorporado) o mediante comandos predefinidos.

1.2.2 Funcionalidad de cara al cliente o al usuario final

- Acceso a la aplicación si el usuario está registrado.
- Consulta de la temperatura ambiental, humedad ambiental, humedad terrestre, nivel de luminosidad y nivel de agua actual del invernadero.
- Consulta de la cantidad de agua consumida durante el riego.
- Consulta sobre la evolución que ha tenido la temperatura ambiental, humedad ambiental, humedad terrestre e incidencia de luz en el invernadero en forma gráfica.
- Gestión del estado del motor de agua.
- Gestión del estado de la bombilla.
- Gestión del estado del aire acondicionado.
- Gestión del estado del humidificador.
- Consulta y gestión del valor límite de la temperatura ambiental, la humedad ambiental, el nivel de luminosidad del invernadero y la humedad terrestre para el accionamiento automático de algunos actuadores.
- Gestión del invernadero por voz.

1.2.3 Esquema de la arquitectura

En la siguiente figura se muestra el diagrama completo del Trabajo Fin de Grado. Como se puede ver, la gestión del invernadero por parte del cliente comienza con la aplicación Android llamada “ConectaMiHuerto”. El servidor REST será el encargado de atender todas las peticiones de la app, y dependiendo de éstas, realizará consultas a la base de datos o enviará un comando para que su procesamiento. Este comando puede recibirse siendo simplemente una cadena de texto, y en tal caso será enviado directamente al bróker de Kafka, o un fichero de audio que tendrá que ser transcrito a texto mediante Voice2JSON. Una vez realizada la conversión, este programa, a través del script “enviar_comando.sh” enviará también el comando en texto al bróker de Kafka. El fichero Python es el que recibirá estos comandos y se conectará a OpenHAB a través de su API REST, el cual, estableciendo una conexión mediante el protocolo MQTT se conectará con la placa ESP32 para conseguir una comunicación con los sensores y actuadores.

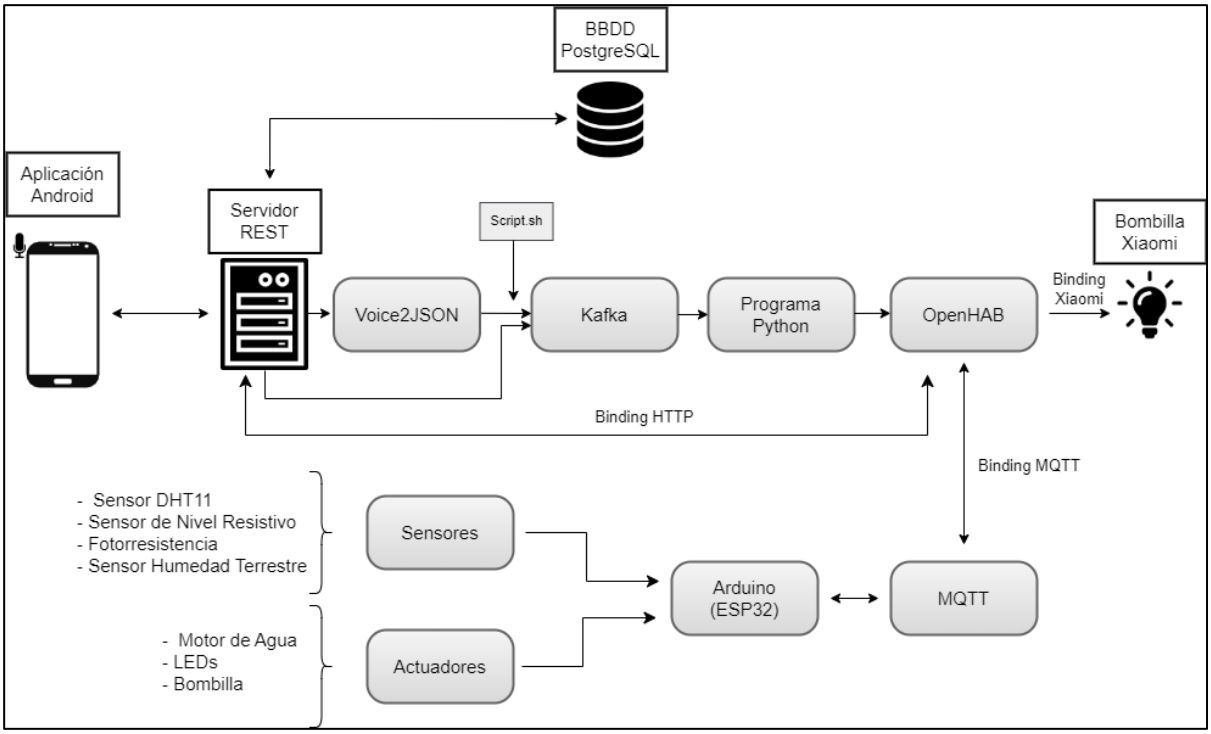

Figura 1-1: Esquema de la arquitectura del Trabajo Fin De Grado

1.3 Estructura de la memoria

La memoria comenzará exponiendo todas las tecnologías que se han necesitado para que el sistema completo funcione correctamente, tanto las necesidades software como hardware. Posteriormente se abordará profundamente el funcionamiento y las amplias capacidades que poseen Voice2JSON y Kafka, así como de OpenHAB y cómo se ha solucionado en este trabajo la conexión entre estos programas. Le seguirá también una explicación detallada sobre el funcionamiento de MQTT y cómo se ha utilizado para interconectar el módulo ESP32 con el resto del sistema a través de OpenHAB. En otro capítulo se expondrá la solución para el acceso del usuario, utilizando una aplicación Android que se conectará a un servidor REST, el cual gestionará las peticiones que se hagan a la base de datos PostgreSQL. Finalizará con un capítulo sobre posibles líneas futuras y conclusión del trabajo.

1.4 Antecedentes

El presente trabajo parte de la idea de conectar OpenHAB con Kafka para posteriormente ampliarse utilizando otros componentes, otorgándole mayor funcionalidad al mismo.

Otros Trabajos Fin de Grado del departamento de Ingeniería Telemática de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla han utilizado OpenHAB como base para su desarrollo. Tal es el caso de "*Aplicación domótica en Android con OpenHAB para el control de los dispositivos del hogar*", realizado en el año 2020, encaminado, como indica el propio título, a la gestión domótica de una vivienda.

2 TECNOLOGÍAS REQUERIDAS

"Cada día sabemos más y entendemos menos"

Albert Einstein

En este capítulo se aclararán cuáles han sido las tecnologías y herramientas necesarias para el desarrollo del trabajo, tanto software como hardware, definiendo el uso de cada una de ellas.

2.1 Recursos software

Se entienden como recursos software los programas que han sido utilizados para la ejecución y gestión del trabajo.

2.1.1 OpenHAB

Programa principal sobre el que se apoya el trabajo. Se trata de una tecnología cuyo software es de código abierto, diseñada para integrar diferentes sistemas de automatización y dispositivos dentro de una única solución.

Figura 2-1: OpenHAB

2.1.2 Kafka

Programa que forma parte de la idea base del trabajo junto con OpenHAB. Usado para enviar los comandos definidos por el usuario a OpenHAB.

Figura 2-2: Kafka

2.1.3 Voice2JSON

Programa de código abierto empleado para la conversión de audio a texto, en formato JSON. El texto resultante de la conversión se enviará al bróker de Kafka.

Figura 2-3: Voice2JSON

2.1.4 MQTT

Se trata de un protocolo de mensajería para Internet de las cosas (IoT). Está diseñado como transporte de mensajes de tipo publicador/suscriptor ideal para conectar dispositivos remotos. Se utiliza en una amplia variedad de industrias, como las telecomunicaciones, la automotriz, la manufacturera, etc.

Figura 2-4: MQTT

2.1.5 Arduino

El software Arduino de código abierto facilita la escritura de algoritmos y la carga en la placa. Este software se puede utilizar con cualquier módulo compatible con Arduino. Se utilizará para programar el módulo ESP32, sensores y actuadores.

Figura 2-5: Arduino

2.1.6 Ping

Aplicación móvil utilizada para probar la conectividad entre el dispositivo móvil y el ordenador a través de la red local.

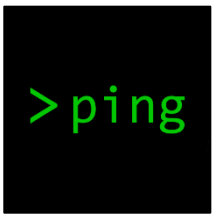

Figura 2-6: Ping

2.2 Herramientas software

Se entienden como herramientas software a los programas que han sido utilizados para facilitar el desarrollo y la gestión del trabajo.

2.2.1 Android Studio

Es el entorno de desarrollo integrado oficial para la plataforma Android. Usado para programar la aplicación móvil del usuario con el que accederá al control y gestión del invernadero.

Figura 2-7: Android Studio

2.2.2 PostgreSQL

Es un sistema de gestión de base de datos relacional de código abierto. Se usará para desarrollar la base de datos donde se almacenará información de interés del invernadero, basado en el lenguaje SQL.

Figura 2-8: PostgreSQL

2.2.3 Spring

Es un framework para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java. Usado para la creación del servicio REST.

Figura 2-9: Spring Framework

2.2.4 GNU/Linux

Es una familia de sistemas operativos tipo Unix compuesto por software libre y de código abierto. Es el sistema operativo utilizado en la máquina virtual que permite el funcionamiento de gran parte del proyecto.

Figura 2-10: Linux

2.2.5 Windows 11 Home

Windows es el nombre de una familia de distribuciones de software para PC, servidores, sistemas empotrados y antiguamente teléfonos inteligentes desarrollados y vendidos por Microsoft y disponible para múltiples plataformas. Es la distribución software sobre la que se instalará el servidor local de OpenHAB.

Figura 2-11: Windows 11

2.2.6 Python

Lenguaje de alto nivel de programación interpretado, multiparadigma, que soporta parcialmente la orientación a objetos, y en menor medida, una programación funcional.

Figura 2-12: Python

2.2.7 Driver Silicon Labs

Los controladores del puerto COM virtual del puente USB a UART CP210x son necesarios para el funcionamiento del dispositivo (módulo ESP32) como un puerto COM virtual para facilitar la comunicación con el host.

Figura 2-13: Silicon Labs

2.2.8 Extractor de tokens de Xiaomi

Es un script que se utiliza como herramienta de extracción para recuperar tokens de todos los dispositivos conectados a la cuenta de Xiaomi almacenados en la nube y claves de cifrado para dispositivos BLE. En el trabajo se empleará para obtener el token de la bombilla Xiaomi.

2.3 Recursos Hardware

Se entienden como recursos hardware a todos los dispositivos físicos empleados para el desarrollo y despliegue del proyecto.

2.3.1 Ordenador portátil

Ordenador portátil utilizado para alojar todos los programas y servidores empleados en el trabajo.

Figura 2-14: Asus Vivobook 16x OLED

2.3.2 Teléfono Móvil

Dispositivo móvil utilizado para acceder al control y gestión del invernadero desde la aplicación Android creada. (Teléfono Huawei P30 ELE-L29)

Figura 2-15: Huawei P30

2.3.3 Punto de Acceso WIFI

Punto de acceso WIFI que permite el establecimiento de una red local y de esta forma una comunicación entre el dispositivo móvil y el ordenador.

Figura 2-16: Punto de acceso WiFi

2.3.4 Módulo ESP32

Modulo compatible con Arduino que hará de puente entre los sensores y actuadores y el resto del invernadero. Es el modelo predecesor de la placa ESP8266.

Algunas de sus características técnicas son:

Tabla 2-1: Características del módulo ESP32

Voltaje de alimentación	5V
Voltaje de entrada/salida	3.3V
Corriente de Funcionamiento	Min. 500mA
SoC	ESP32-WROOM 32
Frecuencia de Reloj	80Mhz / 240Mhz
RAM	512kB
Memoria Flash externa	4MB
Pines I/O	34
Interfaces	SPI, I2C, I2S, CAN, UART
Protocolos Wi-Fi	802.11 b/g/n (802.11n hasta 150 Mbps)
Frecuencia Wi-Fi	V4.2 – BLE y Bluetooth clásico
Antena Inalámbrica	PCB
Dimensiones	56x28x13mm

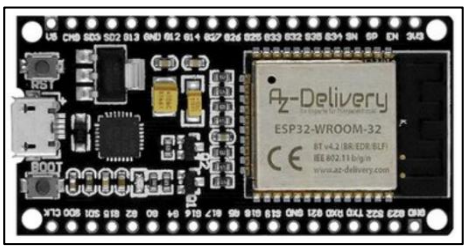

Figura 2-17: Modulo ESP32

2.3.5 Protoboard

Tablero con orificios que se encuentran conectados eléctricamente entre sí de manera interna, habitualmente siguiendo patrones de líneas, en los cuales se pueden insertar componentes electrónicos, cables, prototipado de circuitos y similares. Permitirá conectar los componentes electrónicos del trabajo.

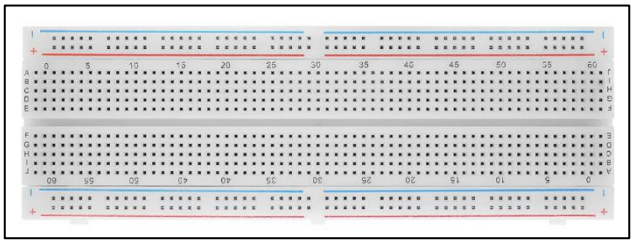

Figura 2-18: Protoboard

2.3.6 Módulo Relé 5V

Es un dispositivo electromagnético que actúa como un interruptor. Se suele emplear para controlar un circuito de salida de mayor potencia que el de entrada. En el trabajo, se utilizará para que el motor de agua pueda gestionarse, ya que necesita un suministro de potencia mayor que el del resto de los componentes electrónicos.

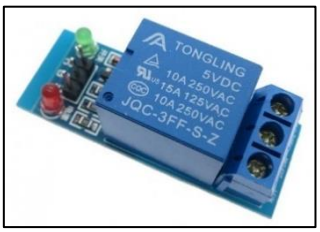

Figura 2-19: Módulo Relé

2.3.7 Motor de agua

Mini motor de agua de 4.5V, modelo JT-DC3V-4.5, utilizado para bombear el agua.

Algunas de las características técnicas:

Tabla 2-2:Características del motor de agua

Voltaje de alimentación	4.5V
Corriente de Funcionamiento	0.18A
Potencia de Funcionamiento	0.81W
Capacidad máxima de bombeo	0.55M
Caudal Máximo	100L/h
Grado de Impermeabilización	IP68

Figura 2-20: Motor de Agua

2.3.8 Portapilas 4xAA

Portapilas para 4 pilas AA que proporcionará la tensión suficiente para el funcionamiento del motor de agua.

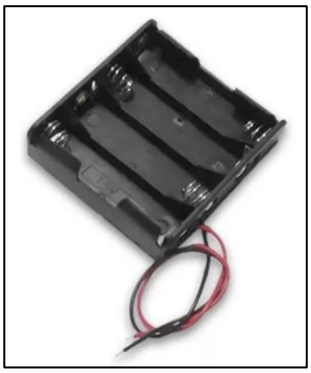

Figura 2-21: Portapilas 4xAA

2.3.9 Cables

2.3.10 Cables de alimentación

2.3.10.1 Cable de alimentación del portátil

Cable de alimentación utilizado para la carga del portátil

2.3.10.2 Cable de alimentación del módulo ESP32

Se utiliza un cable USB-micro USB para alimentar la placa ESP32.

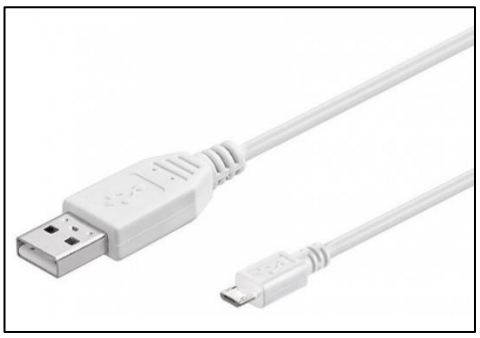

Figura 2-22: Cable USB-Micro USB

2.3.10.3 Cables Arduino

Se utilizan un conjunto de cables puentes para poder conectar los componentes electrónicos entre si.

- Cable puente macho-macho
- Cable puente macho-hembra
- Cable puente hembra-hembra

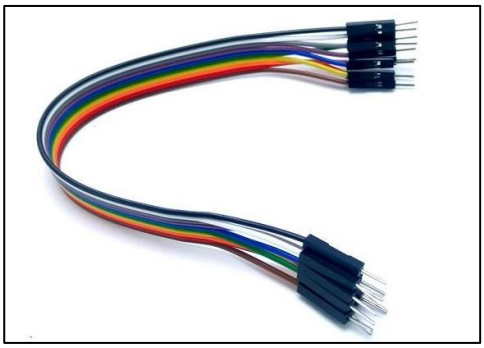

Figura 2-23: Cables Arduino

2.3.11 LEDs

Es una fuente de luz constituida por un material semiconductor dotado de dos terminales. Dispone de una unión p-n que irradia luz cuando se activa.

Figura 2-24: LEDs

2.3.12 Resistencias

Son elementos pasivos que disipan energía en forma de calor según la ley de Joule. Se oponen al flujo de la corriente eléctrica a través de un conductor.

En este trabajo se han utilizado resistencias de valor:

- 1KΩ
- 10KΩ

Figura 2-25: Resistencia

2.3.13 Transistor NPN

Es un dispositivo electrónico que está compuesto por tres regiones semiconductoras interconectadas: N-P-N. Este elemento por lo tanto consta de tres pines de conexión y tiene dos funciones básicas, ser un interruptor electrónico o un amplificador. En el trabajo, será usado como amplificador, ya que el relé necesita una tensión de 5V para poder abrir o cerrar el circuito del motor de agua.

Figura 2-26: Transistor NPN

2.3.14 Módulo DHT11

Este módulo contiene el sensor DHT11 utilizado para medir la temperatura y la humedad relativa. Para medir el aire circundante, dispone de un termistor, para la medida de la temperatura, y un sensor capacitivo de humedad.

Algunas de las características técnicas:

Tabla 2-3: Características del módulo DHT11

Rango de temperatura medible	De 0ºC a 50ºC
Precisión de la temperatura	± 2ºC
Rango de humedad relativa medible	De 20% a 90%
Precisión de la humedad	± 5%
Voltaje de alimentación	3,3 V – 5V
Tasa de muestreo	1Hz
Corriente de funcionamiento	Max 2,5mA

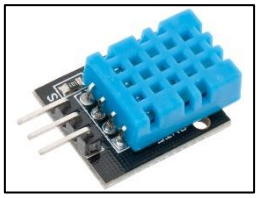

Figura 2-27: Modulo DHT11

2.3.15 Fotorresistencia

También llamado célula fotoeléctrica, es un componente electrónico cuya resistencia se modifica con el aumento de intensidad de luz incidente. Cuanta más luz incida sobre ella, menor resistencia, y viceversa.

Posee una resistencia de 50KΩ en oscuridad y de 500Ω en luz brillante.

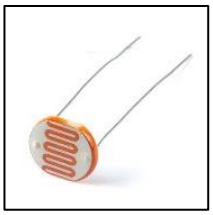

Figura 2-28: Fotorresistencia

2.3.16 Sensor de nivel resistivo

Utilizado para detectar el nivel de agua.

Algunas características técnicas:

Tabla 2-32-4: Características del sensor resistivo

Voltaje de alimentación	5V
Corriente de Funcionamiento	< 20mA
Temperatura de Trabajo	De 10ºC a 30ºC
Señal de voltaje de salida	De 0V a 4.2V

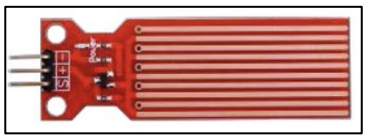

Figura 2-29: Sensor Nivel Resistivo

2.3.17 Sensor de humedad del suelo

Utiliza un dispositivo con electrodos marcados con HW-80 y un convertidor HW-103.

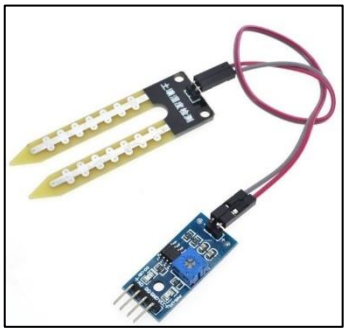

Figura 2-30: Sensor Humedad Terrestre

2.3.18 Bombilla Xiaomi

Bombilla utilizada para simular el funcionamiento del invernadero.

Algunas características técnicas:

Tabla 2-42-5: Características de la bombilla Xiaomi

Flujo Luminoso	800 lúmenes
Potencia Nominal	10 W
Vida útil	25.000 horas
Conectividad	Wi-Fi IEEE 802.11 b/g/n 2.4GHz
Compatible con	Alexa, Google Assistant, IFTTT
App	Mi Home

Figura 2-31: Bombilla Xiaomi

2.3.19 Multímetro

Instrumento eléctrico portátil capaz de medir directamente magnitudes eléctricas activas, como corrientes y tensiones, o pasivas, como resistencias, capacidades, etc. Las medidas pueden realizarse en corriente continua o alterna y en varios márgenes de medidas cada una.

Figura 2-32: Multímetro

2.3.20 Soldador Eléctrico

Es una herramienta eléctrica usada para soldar. Funciona convirtiendo la energía eléctrica en calor, que a su vez provoca la fusión del material utilizado en la soldadura, como el estaño. Utilizado para soldar los cables del motor de agua y del portapilas con los cables Arduino.

Figura 2-33: Soldador Eléctrico

2.3.21 Pilas

Dispositivo que convierte energía química en energía eléctrica. En el trabajo se utilizan 4 pilas del tipo AA de 1.5V para proporcionar el voltaje suficiente para el funcionamiento del motor.

Figura 2-34: Pila

3 RECONOCIMIENTO DE VOZ Y COMUNICACIÓN KAFKA CON OPENHAB

"Los niños deben aprender cómo pensar, no qué pensar"

Margaret Mead

En este capítulo se detallará el funcionamiento de los programas que conforman el núcleo del proyecto (Voice2JSON, Kafka y OpenHAB), los cuales permiten la comunicación entre el usuario final, los actuadores y sensores. Se explicarán cuales son los principales comandos de Voice2JSON utilizados en el Trabajo Fin de Grado necesarios para realizar las transcripciones de audio a texto. También se indicará como se ha configurado el publicador y subscriptor en Kafka, así como su conexión con Voice2JSON. Por último, se especificarán los pasos que han sido necesarios para poner en funcionamiento el servicio de OpenHAB y su unión con Kafka.

3.1 Voice2JSON

Voice2JSON es una colección de herramientas de línea de comandos para el reconocimiento de voz offline en Linux. Es gratuito, de código abierto (MIT) y admite los siguientes idiomas:

Tabla 3-1: Idiomas permitidos en Voice2JSON

Identificación	Idioma
es	Español
cs	Checo
ca	Catalán
de	Alemán
el	Griego
en	Inglés

fr	Francés
hi	Hindi
it	Italiano
ko	Coreano
kz	Kazajo
nl	Holandés
pl	Polaco
pt	Portugués
ru	Ruso
sv	Sueco
vi	Vietnamita
zh	Mandarín

Voice2json admite audio mono de 16 bits a una frecuencia de 16 KHz como entrada, en archivos WAV, para poder hacer la conversión.

3.1.1 Herramientas de línea de comandos

El formato general de un comando voice2json es:

$ voice2json [--debug] [--profile <PROFILE>] <COMMAND> [<COMMAND_ARG>…] ,

donde <PROFILE> puede ser:

- Un nombre de idioma compatible, como puede ser “es” o alguno de los identificadores de la tabla anterior.
- Un nombre de un perfil conocido, como puede ser “es_kaldi_rhasspy” para el idioma español (de España).
- Un directorio que contenga los ficheros profile.yml, sentences.ini u otros específicos del idioma.

  En caso de que no se indique <*PROFILE*>, se utilizará el perfil predeterminado de inglés de EEUU.

Dentro de <*COMMAND*> se puede especificar uno de los siguientes comandos:

Tabla 3-2: Comandos Voice2JSON

Comando	Definición
download-profile	Descargar los archivos necesarios para un perfil
train-profile	Generar los objetos de voz/intención
transcribe-wav	Transcribir archivo WAV a texto
transcribe-stream	Transcribir secuencias de audio en directo a texto
recognize-intent	Reconocer la intención de un JSON o texto
wait-wake	Escuchar la transmisión de audio en directo para la palabra de activación
record-command	Grabar el comando de voz desde la transmisión de audio en directo
pronounce-word	Buscar o adivinar cómo se pronuncia una palabra
speak-sentence	Pronunciar una frase utilizando la conversión de texto a voz
generate-examples	Generar intentos aleatorios
record-examples	Generar y grabar ejemplos de voz
test-examples	Ejemplos de prueba de voz grabada
show-documentation	Ejecutar un servidor HTTP localmente con documentación
print-profile	Mostrar la configuración del perfil
print-downloads	Mostrar información de descarga del archivo de perfil
print-files	Mostrar archivos de perfil de usuario para copias de seguridad
print-version	Mostrar la versión actual de Voice2Json

De estos comandos solo se explicarán con detalle, en los siguientes subapartados, los que han sido utilizados en el trabajo:

3.1.1.1 Download-profile

El comando completo detallado sería:

$voice2json - -profile es_kaldi_rhasspy download-profile

Permite la descarga de todos los ficheros necesarios para el perfil del idioma español.

3.1.1.1.1 El perfil

Un perfil de voice2json es un conjunto de archivos que contienen todo lo necesario para poder realizar las transcripciones del idioma español especificado en el anterior comando y reconocer los comandos de voz. Incluye:

- profile.yml → Un archivo YAML que contiene, entre otros datos, el nombre del idioma del perfil, un código de configuración regional o la voz de eSpeak usada para la pronunciación de palabras.
- sentences.ini → Un archivo plantilla en el que se describirán todos los comandos de voz que se quieren reconocer. En la siguiente figura se muestra parte de la configuración de este fichero:

```
[EstadoLuzBombilla]
states = (enciende | apaga)
(<states>){state} [la] bombilla

[CambiarOnOffBombaDeAgua]
states = (enciende | apaga)
(<states>) [el motor] [la bomba] de agua

[CambiarOnOffAire]
states = (enciende | apaga)
(<states>) el aire

[CambiarOnOffHumidificador]
states = (enciende | apaga)
(<states>) el humidificador
```

Figura 3-1: Archivo sentences.ini

- Modelos voz/intención → Compuestos por:
  - acoustic_model → Un directorio con los artefactos del modelo de voz.
  - intent.pickle.gz → Durante el proceso de entrenamiento, se crea un grafo dirigido que posteriormente se transforma en un transductor de estados finitos. Este transductor es el modelo encargado de convertir secuencias de audio en secuencias de texto mediante el análisis y procesado de los diversos estados y transiciones presentes en el conjunto de datos.

- Diccionarios de pronunciación → Cómo voice2json espera que se pronuncien las palabras. Se detallarán más adelante.
- Modelos lingüísticos → Deducción estadística sobre qué palabras siguen a otras en los comandos de voz introducidos. Se detallarán más adelante.
- Modelos de grafema a fonema → Utilizados para conocer cómo se deben pronunciar las palabras desconocidas.
  - g2p.fst → Un transductor de estado finito creado usando phonetisaurus, una biblioteca utilizada para entrenar, evaluar y usar modelos G2P para el reconocimiento de voz.

- Mapas de fonemas → Utilizados para relacionar fonemas de voz a texto y de texto a voz.
    - espeak-phonemes.txt → Mapa de fonemas de eSpeak.
    - marytts_phonemes.txt → Mapa de fonemas de MaryTTS.
    - Ipa_phonemes.txt → Mapa del Alfabeto Fonético Internacional (IPA).

3.1.1.2 Train-profile

El comando completo detallado sería:

$voice2json - -profile es_kaldi_rhasspy train-profile

Permite generar todos los objetos necesarios de un perfil para el reconocimiento de voz/intención. Esto es, a partir de los comandos de voz que se quieren reconocer, descritos en el archivo sentences.ini, se generarán los modelos lingüísticos, los grafos para cada comando, así como los diccionarios necesarios para poder realizar las conversiones de audio a texto. A partir del apartado 3.1.2 se detallará el uso y funcionamiento de todos estos objetos.

Voice2Json está diseñado para reconocer no solo los comandos de voz especificados en el fichero sentences.ini, sino que también es capaz de transcribir palabras, frases, etc que no estén incluidas en esos comandos. Esto se debe a que Voice2json permite la mezcla de los llamados modelos lingüísticos, expuestos en el apartado 3.1.3.4.

En cada perfil, Voice2json incluye un diccionario base y un modelo lingüístico. El primero contiene las pronunciaciones de todas las palabras posibles y el segundo es un gran modelo entrenado con un contenido muy amplio de texto para el idioma del perfil del español.

3.1.1.3 Transcribe-wav

El comando completo detallado sería:

$voice2json - -profile es_kaldi_rhasspy < nombre_audio.wav

Generará una sola línea de json por cada transcripción donde:

- "text" → es la transcripción más probable de los datos de audio. Se representa como un string.
- "transcribe_seconds" → es el número de segundos que tardó en transcribir el audio. Se representa como un número.
- "wav_name" → es el nombre del archivo de audio WAV. Se representa como un string.
- "wav_seconds" → es la duración del archivo de audio WAV, en segundos. Se representa como un número.

Permite transcribir archivos WAV o datos de audio sin procesar. Como se comentará en el apartado 3.2.5, el resultado de esta instrucción se escribirá en un fichero de texto.

Con este comando, si se le añade el argumento '- -open', voice2json ignorará los comandos de voz introducidos y usará en su lugar el modelo de voz grande ya entrenado que está presente en cada perfil.

3.1.2 Diccionarios de pronunciación

Los diccionarios están compuestos por multitud de líneas en las que, se escribe una palabra y tras ella una lista de fonemas, separados por espacios en blanco, con las que se pronuncia esa palabra. Pueden existir varias pronunciaciones para una misma palabra. Estos fonemas deben coincidir con los que el modelo acústico ha sido entrenado para reconocer.

Un perfil de voice2json puede contener 3 diccionarios:

3.1.2.1 Diccionario base

Este diccionario, llamado "base_dictionary.txt", es un diccionario grande que viene ya construido en cada perfil con la mayoría de las palabras del idioma concreto.

En la siguiente imagen se puede ver un pequeño contenido de este diccionario:

```
a a
aarónica a a ɾ o n i k a
aarónicas a a ɾ o n i k a s
aarónico a a ɾ o n i k o
aarónicos a a ɾ o n i k o s
aba a b a
ababa a b a b a
ababais a b a b ai s
abábamos a b a b a m o s
ababan a b a b a n
ababas a b a b a s
ababilla a b a b i ʎ a
ababillaba a b a b i ʎ a b a
ababillabais a b a b i ʎ a b ai s
ababillábamos a b a b i ʎ a b a m o s
ababillaban a b a b i ʎ a b a n
ababillabas a b a b i ʎ a b a s
ababillada a b a b i ʎ a d a
ababillad a b a b i ʎ a d
ababilladas a b a b i ʎ a d a s
ababillado a b a b i ʎ a d o
ababillados a b a b i ʎ a d o s
ababilláis a b a b i ʎ ai s
ababillamos a b a b i ʎ a m o s
ababillan a b a b i ʎ a n
ababillando a b a b i ʎ a n d o
ababillándome a b a b i ʎ a n d o m e
ababillándonos a b a b i ʎ a n d o n o s
```

Figura 3-2: Diccionario base

3.1.2.2 Palabras personalizadas

Este diccionario, llamado custom_words.txt, es un diccionario pequeño, definido por el usuario, con palabras o pronunciaciones personalizadas.

3.1.2.3 Diccionario

Este diccionario, llamado "dictionary.txt", contiene exactamente el vocabulario necesario para el perfil especificado anteriormente y es generado automáticamente al ejecutar el comando 'train-profile'.

En la siguiente imagen se muestra el contenido de este diccionario, que contiene las palabras que

forman parte de los comandos escritos en el archivo sentences.ini, junto con sus fonemas, :

```
blanca b l a n k a
tres t ɾ e s
seis s ei s
verde b e ɾ d e
motor m o t o ɾ
<unk> SPN
nivel n i b e l
dos d o s
azul a θ u l
diez d ʝ e θ
bomba b o m b a
de d e
cinco θ i n k o
intensidad i n t e n s i d a d
ocho o t͡ʃ o
enciende e n θ ʝ e n d e
el e l
a a
apaga a p a g a
bombilla b o m b i ʎ a
la l a
nueve n we b e
cuatro k wa t ɾ o
color k o l o ɾ
siete s ʝ e t e
uno u n o
agua a g wa
roja r o x a
clap k l a p
```

Figura 3-3: Diccionario

3.1.3 Funcionamiento

Voice2JSON necesita una descripción de los comandos de voz que se desean reconocer en un archivo de configuración llamado sentences.ini por cada perfil. Para dicha configuración, se sigue una gramática JSFG simplificada [14]. Posteriormente, el contenido de este fichero se utiliza para crear un modelo de lenguaje ARPA, el cual permite describir la probabilidad de ocurrencia de secuencias de palabras, y un reconocedor de intenciones.

La funcionalidad central de voice2JSON se puede dividir en dos componentes: voz e intención.

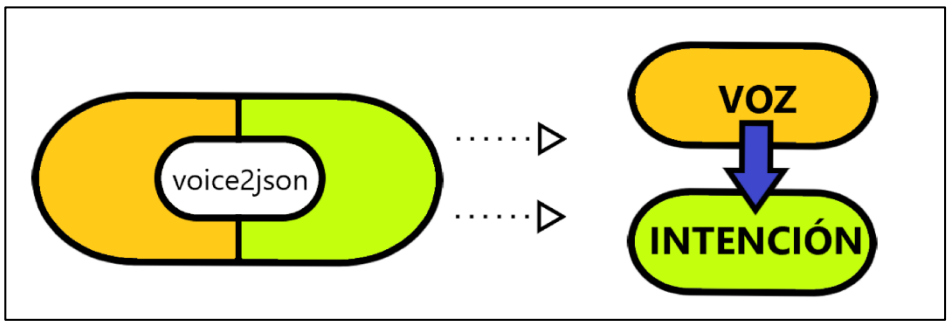

Figura 3-4: Componentes de Voice2JSON

Cuando los comandos de voz son reconocidos por el componente de voz, la transcripción se entrega al reconocedor de intenciones para que la procese. El resultado final de este proceso es un evento JSON estructurado con:

- El texto reconocido a partir del comando de voz, precedido por el campo “text”.
- El nombre de la intención al que pertenece, precedido por el campo “name”.
- Metadatos opcionales sobre el proceso de reconocimiento de voz, como puede ser el tiempo de conversión, fichas, etc.

La transcripción de los comandos de voz se realiza mediante uno de los tres sistemas de código abierto siguientes:

- Pocketsphinx, uno de los motores de reconocimiento de voz de la Universidad Carnegie Mellon, EEUU.
- Kaldi, motor desarrollado por Johns Hopkins
- DeepSpeech, motor de Speech-To-Text que utiliza un modelo con técnicas de aprendizaje automático, desarrollado por Mozilla.

Tanto Pocketsphinx como Kaldi requieren de:

- Un modelo acústico: asigna funciones de audio a fonemas.
- Un diccionario de pronunciación: para mapear fonemas a palabras.
- Un modelo lingüístico: permite describir la frecuencia con la que algunas palabras siguen a otras.

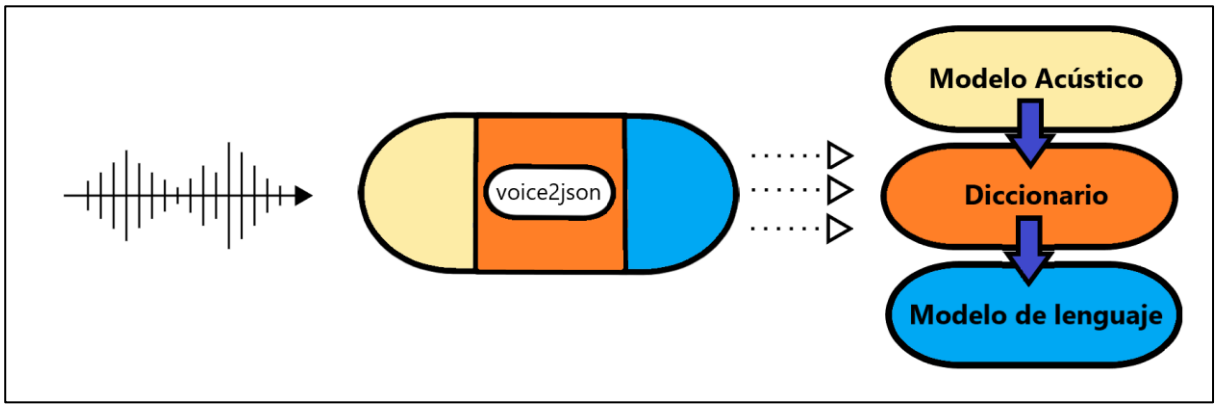

Figura 3-5: Esquema general Voice2JSON

3.1.3.1 Modelo Acústico

Un modelo acústico asigna características acústicas del habla (como el tono, la intensidad, el timbre) a fonemas probables de un idioma determinado. Los fonemas son unidades indivisibles de pronunciación de palabras, específicos del idioma, incluso de lugares concretos. Por lo general, los coeficientes de cepstrum de frecuencia Mel (MFCC) se utilizan como características acústicas, los cuales destacan matemáticamente aspectos útiles del habla humana.

El modelo acústico es un mapeo estadístico entre características de audio (MFCC) y uno o más

fonemas. Este mapeo se aprende a partir de una gran colección de ejemplos de voz junto con sus transcripciones correspondientes.

Se necesita un diccionario de pronunciación preconstruido para mapear las transcripciones a los fonemas antes de que se pueda entrenar un modelo. Sin embargo, recopilar, transcribir y validar estos grandes conjuntos de datos de voz supone un factor limitante en el reconocimiento de voz de código abierto.

3.1.3.2 Diccionario de pronunciación

Se necesita un diccionario que mapee secuencias de fonemas en palabras, tanto para entrenar un modelo acústico como para realizar el reconocimiento de voz.

Durante el entrenamiento, voice2json copia las pronunciaciones de cada palabra de las plantillas de comandos de voz de un gran diccionario de pronunciación preconstruido. La pronunciación de las palabras que no se encuentran en este diccionario se solventa utilizando un modelo de grafema a fonema previamente entrenado.

3.1.3.3 Grafema a fonema

Se puede usar un modelo de grafema a fonema (G2P) para buscar la pronunciación fonética de las palabras. Se trata de un modelo estadístico que asigna secuencias de caracteres (grafemas) a secuencias de fonemas, y se entrena a partir de un gran diccionario de pronunciación prediseñado. Para ello, voice2json utiliza la herramienta Phonetisaurus.

3.1.3.4 Modelo lingüístico

El modelo lingüístico describe con qué frecuencia algunas palabras siguen a otras. Se crea a partir de una gran colección de textos, como pueden ser libros, sitios de noticias, etc. No todas las combinaciones estarán en el material de entrenamiento, por lo que sus probabilidades deben predecirse mediante una heurística.

Los modelos lingüísticos deben estar en formato ARPA de texto plano. Un perfil voice2json suele tener 2 modelos lingüísticos:

3.1.3.4.1 Modelo del idioma base

Este modelo, llamado 'base_languaje_model.txt', es un modelo de idioma que viene ya construido y resume un idioma dado. Se utiliza cuando se indica el argumento '- -open' en el comando transcribe-wav, empleado durante la mezcla de modelos lingüísticos.

En la siguiente imagen se muestra una pequeña parte de este modelo:

```
\data\
ngram 1=234602
ngram 2=3061353
ngram 3=2331482

\1-grams:
-6.208953       &lt      -0.1186114
-6.949357       ''against       -0.1186114
-6.387215       +1      -0.1186114
-6.949357       +10     -0.1186114
-6.949357       +380    -0.1186114
-6.949357       +6      -0.1186114
-6.949357       +8      -0.1186114
-6.949357       --500   -0.1186114
-6.949357       --a     -0.1186114
-6.949357       --adiós -0.1186114
-6.949357       --ahora--dijo   -0.1186114
-6.949357       --alcedo        -0.1186114
-6.949357       --alto  -0.1186114
-6.949357       --alégrate      -0.1186114
-6.949357       --amigo -0.1186113
-6.949357       --antonio       -0.1186114
-6.949357       --aquí  -0.1186114
-6.949357       --arre  -0.1186114
-6.949357       --arrea -0.1186114
-6.949357       --así   -0.1186114
-6.949357       --avant -0.1186114
-6.949357       --añadió        -0.1186114
-6.949357       --bien  -0.1186114
-6.949357       --bien--le      -0.1186114
-6.949357       --bolsón        -0.1186114
```

Figura 3-6: Modelo del idioma base

3.1.3.4.2 Modelo de idioma

Este modelo, llamado languaje_model.txt, resume los comandos de voz válidos para un perfil y se genera automáticamente cuando se utiliza el comando 'train-profile'.

En la siguiente imagen se muestra una parte de este modelo:

```
0 1 <eps> <eps> 0.0
1 2 <eps> <eps> 0.0
2 3 enciende enciende 0.0
2 4 apaga apaga 0.0
3 5 <eps> <eps> 0.0
4 5 <eps> <eps> 0.0
5 6 <eps> <eps> 0.0
6 7 la la 0.0
6 8 <eps> <eps> 0.0
7 9 <eps> <eps> 0.0
8 9 <eps> <eps> 0.0
9 10 bombilla bombilla 0.0
10 11 <eps> <eps> 0.0
0 12 <eps> <eps> 0.0
12 13 intensidad intensidad 0.0
13 14 de de 0.0
13 15 <eps> <eps> 0.0
14 16 la la 0.0
15 17 <eps> <eps> 0.0
16 17 <eps> <eps> 0.0
17 18 bombilla bombilla 0.0
18 19 a a 0.0
18 20 <eps> <eps> 0.0
19 21 <eps> <eps> 0.0
20 21 <eps> <eps> 0.0
21 22 <eps> <eps> 0.0
22 23 nivel nivel 0.0
22 24 nivel nivel 0.0
22 25 nivel nivel 0.0
22 26 nivel nivel 0.0
22 27 nivel nivel 0.0
22 28 nivel nivel 0.0
```

Figura 3-7: Modelo del idioma

3.1.3.4.3 Entrenamiento del modelo lingüístico

Durante el entrenamiento, voice2json genera un modelo de lenguaje personalizado basado en las plantillas de comandos de voz en formato ARPA. Gracias al uso de la biblioteca de código abierto OpenGrm, desarrollado por colaboradores de OHSU y Google Research, Voice2json puede tomar el gráfico de frases intermedio producido durante las fases iniciales del entrenamiento y generar directamente el modelo lingüístico, lo que le permite entrenarse en segundos para millones de posibles comandos de voz.

En la siguiente imagen se muestra de forma gráfica el proceso para la obtención del modelo lingüístico. Se parte de una muestra del contenido del archivo senteces.ini, donde se indica el comando de voz para el encendido de la bombilla. Como se comentará posteriormente en el apartado 3.1.3.5, se genera el grafo de intenciones con el que se podrá obtener el modelo.

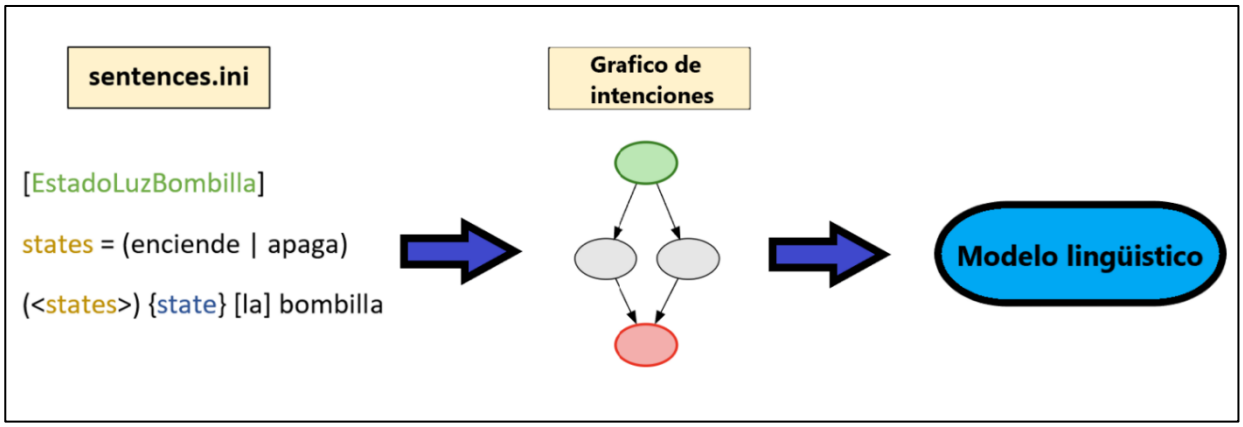

Figura 3-8: Modelo lingüístico

3.1.3.5 Texto a intención

Los sistemas de reconocimiento de voz de voice2json producen transcripciones de texto que se entregan a un sistema de reconocimiento de intención. Cuando ambos sistemas se entrenan juntos a partir del mismo archivo de plantilla, todos los comandos válidos (con pequeñas posibles variaciones) se traducirán a eventos JSON.

Voice2json transforma el conjunto de posibles comandos de voz en un grafo que actúa como un transductor de estados finitos (FST). Cuando se le da una frase válida como entrada, este transductor emitirá la oración transformada junto con palabras "meta" que proporcionan la intención de la frase y las entidades nombradas.

Si, por ejemplo, se tiene la siguiente intención para el estado de una bombilla:

```
[EstadoLuzBombilla]
states = (enciende | apaga)
(<states>) {state} [la] bombilla
```

Figura 3-9: Sintaxis del estado de una bombilla

Y una vez que se entrene con esta plantilla, voice2json generará el siguiente grafo:

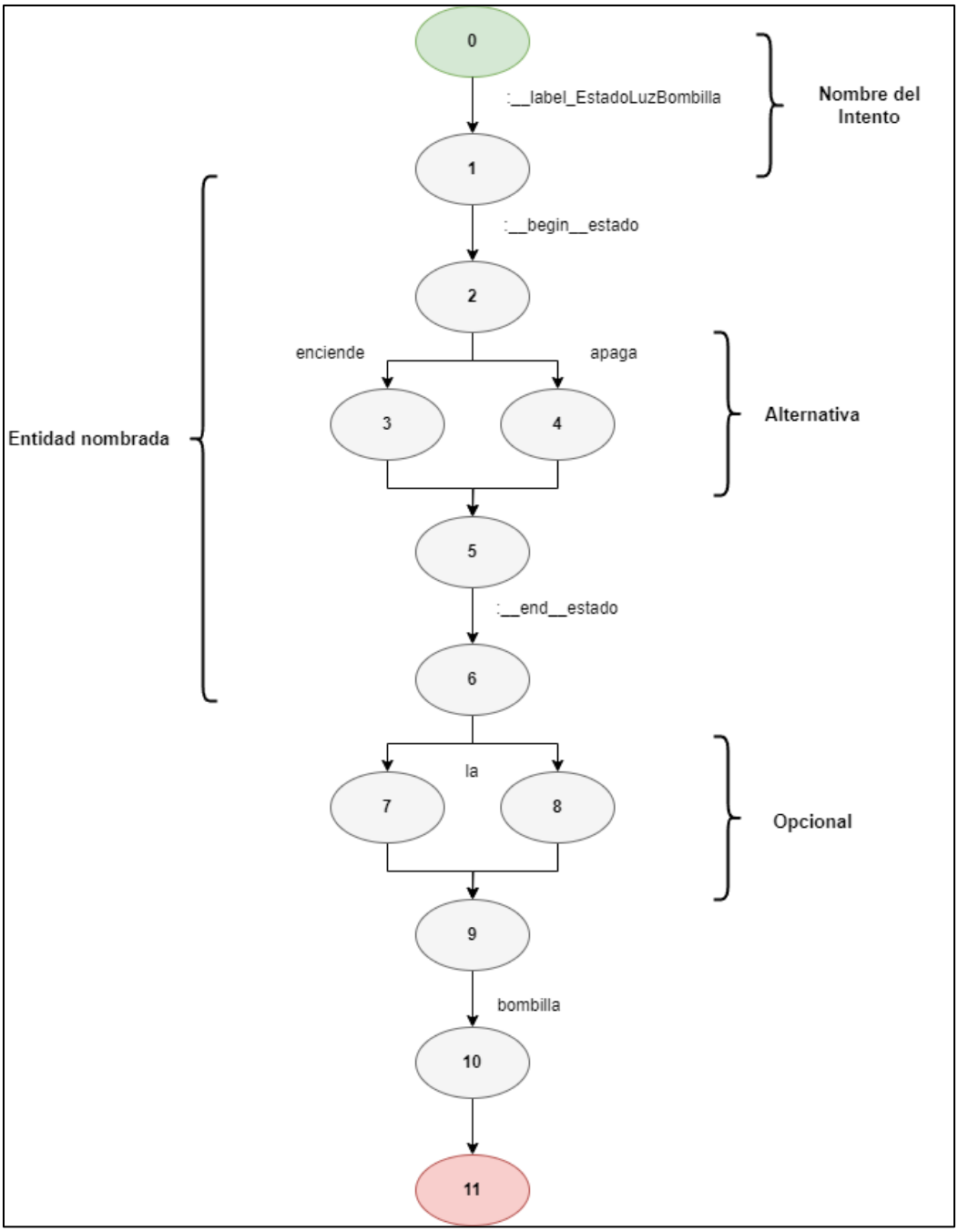

Figura 3-10: Grafo del estado de una bombilla

El FST anterior aceptará las siguientes posibles oraciones:

- Enciende la bombilla
- Enciende bombilla
- Apaga la bombilla
- Apaga bombilla

El resultado cuando el FST acepta cada oración sería:

Tabla 3-3: Estructura oraciones aceptadas

Entrada	Salida
Enciende la bombilla	__label__EstadoLuzBombilla __begin__estado enciende __end__estado la bombilla
Enciende bombilla	__label__EstadoLuzBombilla __begin__estado enciende __end__estado bombilla
Apaga la bombilla	__label__EstadoLuzBombilla __begin__estado apaga __end__estado la bombilla
Apaga bombilla	__label__EstadoLuzBombilla __begin__estado apaga __end__estado bombilla

Se genera una única metapalabra __label__ para cada oración, etiquetándola con el nombre de la intención de la propiedad.

Las metapalabras __begin__ y __end__ son utilizadas por voice2json para construir el evento JSON de cada frase, marcando el principio y el final de un bloque de texto etiquetado en el archivo de plantilla original.

El conjunto de metapalabras utilizado por voice2json es:

Tabla 3-4: Conjunto de metapalabras

Etiqueta	Significado
__label__INTENT	La oración pertenece a la intención nombrada como INTENT.
__begin__TAG	Comienzo de la etiqueta nombrada como TAG.
__end__TAG	Final de la etiqueta nombrada como TAG.
__convert__CONV	Comienzo del convertidor nombrado como CONV.
__converted__CONV	Final del convertidor nombrado como CONV.
__source__SLOT	Nombre de la lista de slots de donde proviene el texto.
__unpack__PAYLOAD	Se decodifica PAYLOAD como un string codificado en base64 para interpretarlo como una etiqueta de arista.

3.1.3.6 El reconocedor de intenciones

El reconocedor de intenciones basado en FST de voice2json se llama fsticuffs. Éste toma el grafo de intenciones generado durante el entrenamiento y lo utiliza para convertir transcripciones del sistema de voz en eventos JSON.

El reconocimiento de intenciones se realiza simplemente pasando la transcripción por el grafo de intenciones, analizando las palabras de salida, así como las metapalabras y generando como resultado el evento JSON.

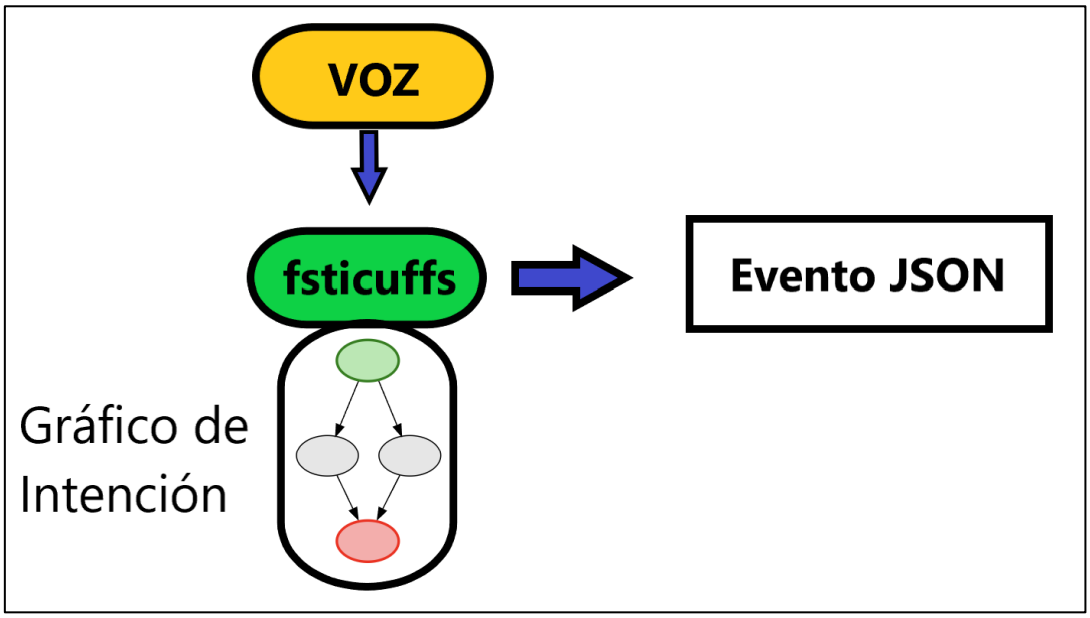

Figura 3-11: Reconocedor de intenciones

Puede ocurrir que fsticuffs recibiera un comando de voz no válido. Por ejemplo, en lugar de recibir la oración "Enciende la bombilla", se recibiera "¿Podrías encender la bombilla?". Dado que la palabra "podrías" no está codificada en la intención, el FST no la reconocerá.

Para solucionarlo, voice2json omite estas palabras "vacías" de forma silenciosa durante el reconocimiento, haciendo que el proceso de transcripción sea más lento, pero permitiendo la entrada de más oraciones.

Los pasos necesarios para la instalación de Voice2JSON en la máquina virtual se detallan en el Anexo A.

3.2 Kafka

Apache Kafka es un proyecto de intermediación de mensajes de código abierto desarrollado por LinkedIn y donado a la Apache Software Foundation (ASF). Está escrito en Java junto con Scala y tiene como objetivo proporcionar una plataforma unificada de alto rendimiento y de baja latencia para la manipulación en tiempo real de distintas fuentes de datos. Puede verse como una cola de mensajes, que funciona bajo el patrón publicador-suscriptor y permite una gran escalabilidad. Kafka se integra con numerosas tecnologías, de esta forma, permite construir flujos de datos en tiempo real entre distintos sistemas y aplicaciones de manera desacoplada.

En el proyecto Kafka se ha utilizado como medio de comunicación entre Voice2JSON-OpenHAB y el servidor REST-OpenHAB, de manera que permitirá enviar los comandos por voz una vez transcritos por el único topic creado 'thing1'. En caso de no tratarse de comandos por voz, estos serán enviados de igual forma a través de este servicio.

La configuración del productor y del consumidor se detallarán en los apartados 3.2.5: Conexión Voice2JSON-Kafka, 5.2: Servidor REST y 3.3.5: Conexión Kafka-OpenHAB.

Tradicionalmente, la arquitectura organizativa empleada para el funcionamiento de las corporaciones era la siguiente:

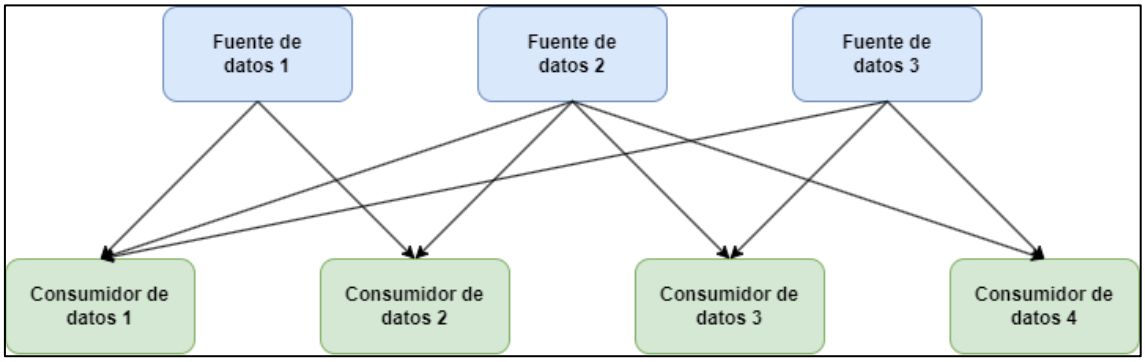

Figura 3-12: Estructura previa a Kafka

Sin embargo, esta estructura implica un mantenimiento y una evolución complejos y costosos en la que hay que definir los formatos de datos intercambiados (como CSV o JSON) y los protocolos para cada conexión (como REST).

Kafka soluciona todos estos inconvenientes, estableciéndose como un sistema intermedio entre los productores y los consumidores:

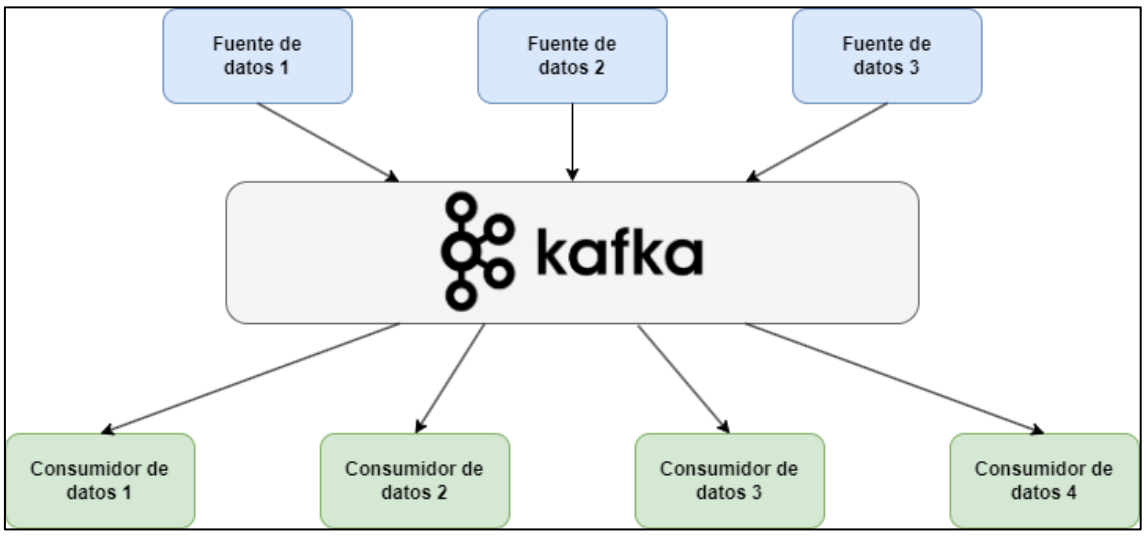

Figura 3-13: Estructura con Kafka

Para ello, combina tres capacidades que son clave para su funcionamiento:

- Publicar (escribir) y suscribirse (leer) a flujos de eventos, incluida la importación y exportación continua de sus datos desde otros sistemas.
- Almacenar flujos de eventos de forma duradera y fiable durante todo el tiempo que se desee.
- Procesar flujos de eventos en el momento en que se producen o a posteriori.

La transmisión de eventos se centra en capturar datos en tiempo real de fuentes, como bases de datos, sensores, dispositivos móviles, etc, en forma de flujos de eventos. Este flujo permite una interpretación continua de la información, de modo que ésta se encuentre en el lugar y momento adecuados.

3.2.1 Patrón Publish/Subscribe

El patrón publicador/subscriptor, también llamado productor/consumidor o publish/subscribe, se emplea para la comunicación entre aplicaciones a través de mensajes. Es decir, se trata de un sistema de eventos distribuidos donde un subscriptor tiene interés por ciertos eventos. El publicador es el que genera los eventos, los cuales serán enviados posteriormente a los subscriptores interesados.

El subscriptor puede incorporar un filtro para consumir solo los mensajes que le interese y suscribirse a uno o varios tipos de mensajes.

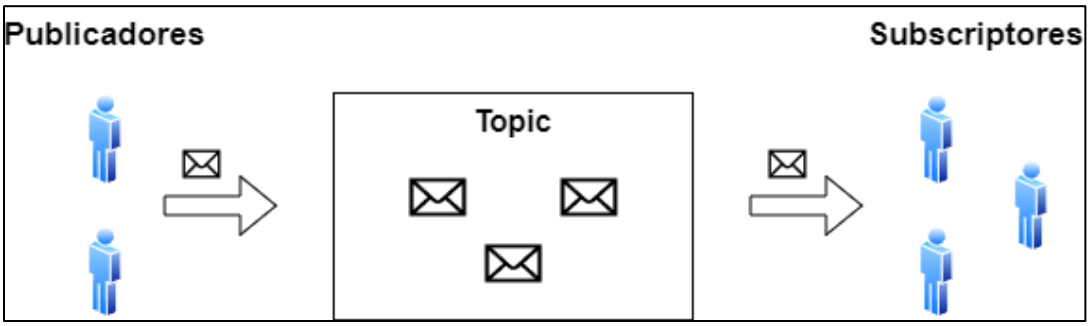

Figura 3-14: Patrón Publish/Subscribe

Este modelo de mensajería es un paradigma asíncrono y desacoplado en el espacio, ya que los productores y los subscriptores no se conocen entre sí. En él, puede existir un proceso intermediario, que es lo ocurre en Apache Kafka, permitiendo el aumento del desacoplamiento, pero también sufriendo el inconveniente de ser un cuello de botella y un punto de fallo.

Por eso, una posible solución a este problema se basa en distribuir este proceso en una red de intermediarios (los llamados brokers), que proporciona la escalabilidad y fiabilidad necesaria. De este modo, si un bróker fallase o se quedara sin conectividad, otro bróker podría tomar su lugar y gestionar las peticiones.

3.2.2 Componentes básicos de Kafka

Los componentes básicos de Kafka son:

3.2.2.1 Evento

Un evento describe la ocurrencia de algún hecho en el mundo real y es comunicado al sistema Kafka a través de un mensaje.

3.2.2.2 Topic

Los topics en Kafka son las categorías en las que se clasifican los mensajes. Así, se puede entender cada topic como si fuera un flujo de datos.

Existen dos tipos de topics:

- **Topics regulares:**

Los temas regulares pueden configurarse con un tiempo de retención o un límite de espacio. Si hay registros que son más antiguos que el tiempo de retención especificado o si se supera el límite de espacio para una partición, Kafka puede eliminar los datos más antiguos para liberar espacio de almacenamiento. Por defecto, los temas se configuran con un tiempo de retención de 7 días, pero también es posible almacenar datos de forma indefinida.

- Topics compactados:

En este caso, los registros no caducan en función de los límites de tiempo o espacio. En su lugar, Kafka trata los mensajes posteriores como actualizaciones de mensajes más antiguos que tengan la misma clave y garantiza que nunca se borrará el último mensaje de una clave.

Como se explicó al inicio del apartado 3.2, el único topic configurado en Kafka se llama 'thing1', por el que se enviarán todos los comandos de texto.

3.2.2.3 Partición

Cada topic está dividido en particiones, que permiten a Kafka distribuir los datos en los nodos conectados (brokers). Es la unidad de paralelismo para dotar a Kafka de una escalabilidad horizontal. A cada mensaje que se quiera escribir en un topic, se le asigna una clave de partición y en caso de que no se le fije ninguna, ésta se calcula de forma aleatoria.

Cada partición tiene un offset asociado que se corresponde con un identificador incremental asignado a cada mensaje. De esta forma, un mensaje se puede identificar por el nombre del topic que lo contiene, la partición a la que pertenece y su offset.

3.2.2.4 Replicación

Una réplica en Kafka consiste en realizar una copia de una partición disponible en otro bróker. Este mecanismo permite a Kafka ser tolerante a fallos y asegura que no haya pérdidas de datos.

Cuando existen varias réplicas disponibles, una de ellas es elegida como líder, y el resto como seguidoras, de modo que cuando están sincronizadas se marcan como ISR (In Sync Replica). Para que un topic se encuentre en buen estado, el valor del ISR debe ser igual al factor de replicación.

3.2.2.5 Productor

Los productores son los clientes conectados a Kafka encargados de publicar mensajes en un bróker. Son los responsables de serializar, particionar, comprimir y repartir la carga entre los brokers en función de las particiones.

La asignación de mensajes a topics puede realizarse con un método llamado round-robin o con alguna función semántica que determine la partición.

Los mensajes están compuestos por una clave, el valor y una marca de tiempo asignada.

3.2.2.6 Consumidor

Los consumidores de Kafka son los clientes conectados suscritos a los topics que consumen los mensajes. Cada consumidor tiene asociado un grupo de consumidores.

3.2.2.7 Bróker

Kafka se ejecuta en un clúster de uno o más servidores llamados brokers y las particiones de todos los temas se distribuyen entre los distintos nodos del clúster. Además, las particiones se replican en varios brokers. Todo esto permite a Kafka entregar flujos masivos de mensajes de forma tolerante a fallos.

3.2.3 Interfaces de Kafka

Hay cinco interfaces principales en Kafka:

- Producer API: Permite a las aplicaciones enviar flujos de datos a los topics del clúster de Kafka.
- Consumer API: Permite a las aplicaciones leer flujos de datos de los topics del clúster de Kafka.
- Connector API: Interfaz de importación/exportación para la conexión con sistemas de terceros.
- Streams API: Biblioteca Java para el procesamiento de flujos de datos, convirtiendo los flujos de entrada en salida y produciendo el resultado.
- Admin API: Se utiliza para gestionar los temas de Kafka, los brokers y otros objetos.

Las interfaces de consumidor y productor se basan en el protocolo de mensajes Kafka y están desacopladas de la funcionalidad principal del programa.

El protocolo de mensajes real de Kafka es binario y, por tanto, permite que los clientes consumidores y productores se desarrollen en cualquier lenguaje de programación. Por ello, Kafka no está vinculado al ecosistema JVM.

3.2.4 Formato de los mensajes

Los mensajes de Kafka son creados por el productor. Estos se identifican por una clave y un valor en formato binario y pueden ser nulos. [25]

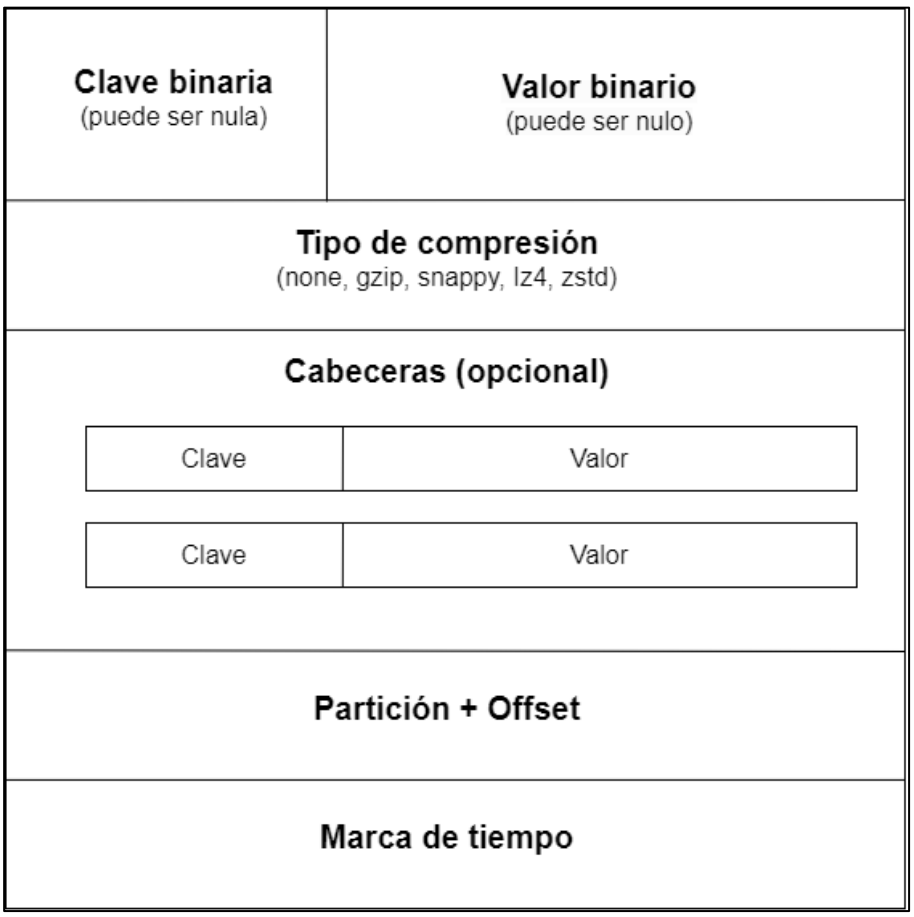

Figura 3-15: Formato de los paquetes Kafka

En la imagen anterior también se puede apreciar como en el mensaje viaja el tipo de compresión utilizado por el productor para enviar todos los datos. El valor por defecto es none (sin compresión), aunque también admite los siguientes tipos: gzip, snappy, lz4 y zstd.

Por otro lado, existen unos encabezados opcionales de pares clave-valor utilizados para la agregación de metadatos en el mensaje.

Por último, al mensaje se le agregará la partición, el offset y una marca de tiempo, establecida por el sistema o por el usuario.

Como es natural, cuando se crea el mensaje, el usuario utilizará objetos de nivel superior para los campos clave-valor. Sin embargo, estos campos se envían como un flujo de bytes y para realizar esta transformación, Kafka proporciona serializadores (serializer) predeterminados, aunque también permite la posibilidad de implementar serializadores personalizados.

3.2.4.1 Serializer

Debido a que los brokers de Kafka solo trabajan con bytes, este instrumento es utilizado para convertir objetos de alto nivel (ya sean cadenas de texto, enteros, etc) en flujos de bytes. Según el contenido que el productor vaya a enviar en los campos clave-valor del mensaje Kafka, se deberán modificar los atributos '*key.serializer*' y '*value.serializer*' en su configuración. Los valores posibles de

estos atributos se corresponden con los tipos de datos java que se puede serializar, como:

- IntegerSerializer
- StringSerializer
- FloatSerializer
- BytesSerializer

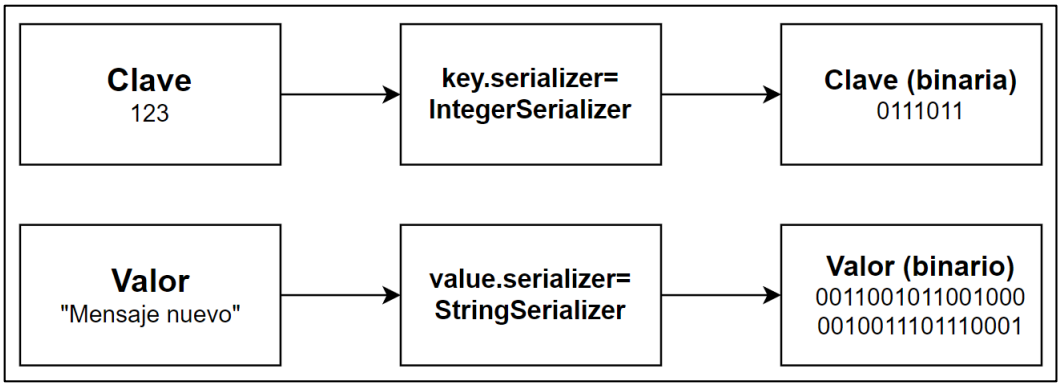

Figura 3-16: Serializer

3.2.4.2 Deserializer

De igual forma, existe un deserializador en la parte del consumidor para transformar el flujo de bytes proveniente de los brokers en objetos legibles. Tal y como se hacía en el productor, hay que modificar los atributos 'key.deserializer' y 'value.deserializer' para obtener los datos de forma correcta. Algunos de los posibles valores para estos atributos son:

- IntegerDeserializer
- StringDeserializer
- FloatDeserializer
- BytesDeserializer

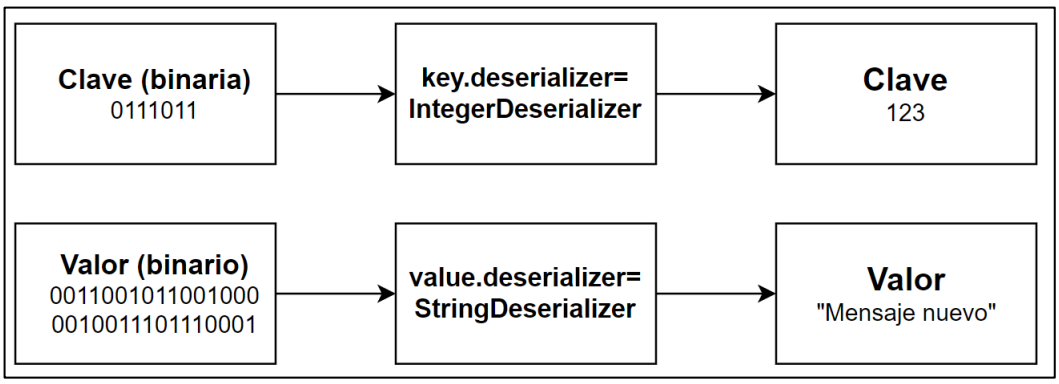

Figura 3-17: Deserializer

Los pasos necesarios para la ejecución de Kafka se explican en el Anexo B.

3.2.5 Conexión Voice2JSON-Kafka

Tal y como se explicará en el apartado 5.2, el servidor REST será el responsable de ejecutar los comandos necesarios para que Voice2JSON realice la transcripción del fichero de audio que se reciba.

El resultado del proceso, compuesto principalmente por los campos “transcribe-seconds”, donde se indica el tiempo necesario para la transcripción, o “text”, donde se almacena el comando convertido, se escribirá en un fichero de texto simple.

Sin embargo, para que el contenido de este fichero forme parte de un mensaje enviado a través de Kafka, se ha creado un shell script que puede considerarse como el productor responsable de enviar los paquetes que contienen los resultados de estas transcripciones.

Un shell script es un archivo de texto que está creado con una serie de instrucciones que son ejecutadas línea a línea por un shell o intérprete de comandos (CLI) de Linux, especificado habitualmente en la primera línea del script. A continuación, se muestra su código:

```
#!/bin/bash

FICHERO=/home/salas/Escritorio/Pruebas/fichero.txt


cd /home/salas/Descargas/kafka_2.13-3.0.0
while true;
do
    if [ -f $FICHERO ]
    then
    echo "Leyendo fichero"
    sleep 1.5
    cat /home/salas/Escritorio/Pruebas/fichero.txt | ./bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic thing1
    rm -f $FICHERO
    rm /home/salas/SDSW/Servidor/apiRest_Servidor/Audios/grabacion.3gp /home/salas/SDSW/Servidor/apiRest_Servidor/Audios/Grabacion.wav

    else
    echo "El fichero no existe"
    fi
    sleep 1
done
```

Figura 3-18: Código enviar_script.sh

Como se puede apreciar en el script, llamado “enviar_script.sh”, el shell utilizado para la ejecución es el BASH, especificado mediante el shebang “#!”. En la variable ‘FICHERO’ se guarda la ubicación del archivo ‘fichero.txt’, donde se escribirá el resultado del proceso de transcripción ordenado por el servidor REST. Sin embargo, en este fichero, el servidor REST también escribirá las ordenes establecidas por el usuario sin voz. Estos procesos se explicarán en el apartado 5.2 cuando se detallen el funcionamiento de los métodos del servicio REST configurados.

Posteriormente, cuando existe este fichero, mediante el comando ‘cat’ se obtiene su contenido y utilizando una tubería, el resultado de la anterior instrucción es enviada como el mensaje del productor Kafka que se ejecuta con el comando:

./bin/Kafka-console-producer.sh –bootstrap-server localhost:9092 –topic thing1

Como argumentos, se especifica la dirección del bróker donde se está ejecutando y el topic ‘thing1’ al que se dirige.

Por último, una vez ejecutada esta instrucción, se elimina el fichero de texto y los dos audios (tanto

en formato 3gp como wav) con el objetivo de no generar errores al realizar posteriores transcripciones. La obtención de los archivos de audio se explicará en el apartado 5.2 y 5.3.

3.3 OpenHAB

Open Home Automation Bus (OpenHAB) es una plataforma de automatización del hogar desarrollada en Java, independiente de la tecnología y de código abierto que se ejecuta como centro del control domótico.

Algunas de las características fundamentales de OpenHAB son:

- Su capacidad para integrar multitud de otros dispositivos y sistemas.
- Proporcionar una interfaz de usuario uniforme y un enfoque común para las reglas de automatización en todo el sistema, independientemente de la cantidad de fabricantes y subsistemas involucrados.

Es un software altamente modular que puede ser ampliado mediante add-ons o complementos. Estos brindan a OpenHAB una amplia gama de capacidades, desde interfaces de usuario hasta la posibilidad de interactuar con un gran número de dispositivos físicos (sensores, cámaras de video, altavoces, etc). Estos complementos pueden proceder de la distribución de OpenHAB o de otras fuentes externas.

OpenHAB se puede configurar de dos maneras diferentes: a través de archivos de configuración o por la interfaz de usuario. A continuación, se detallan algunas ventajas e inconvenientes de usar una u otra forma:

Ventajas de usar archivos de configuración:

- En las versiones anteriores de OpenHAB, el uso de los archivos de configuración era lo habitual, por lo que en muchos ejemplos de la documentación y de los foros se utiliza este enfoque.
- Con configuraciones basadas en archivos es más fácil realizar cambios y actualizaciones masivas, duplicar objetos similares y separar dominios en archivos distintos
- Es más fácil compartir configuraciones entre múltiples instancias de OpenHAB.

Inconvenientes de usar archivos de configuración:

- Son más propensos a errores. El más mínimo error tipográfico produce un error de sintaxis que podría hacer que todo el contenido del archivo deje de ser válido y no sea reconocido.
- La sintaxis, en ocasiones, puede resultar difícil de entender.

Ventajas de usar la interfaz de usuario:

- Facilidad de entendimiento.
- Los things se pueden descubrir automáticamente.

Inconvenientes de usar la interfaz de usuario:

- Complejidad al eliminar cosas obsoletas.
- No todo se puede configurar aún a través de ella.

3.3.1 Componentes básicos de OpenHAB

A continuación, se detallan los componentes básicos con los que trabaja OpenHAB:

3.3.1.1 Things

Los things (cosas) son las entidades que se pueden agregar físicamente a un sistema. No tienen que ser dispositivos físicos, también pueden representar un servicio web o cualquier otra fuente manejable de información y funcionalidad.

3.3.1.1.1 Estados de los Things

Cada thing tiene un estado que permite identificar posibles problemas con el dispositivo o servicio. Los posibles estados se recogen a continuación:

Tabla 3-5: Estados de los things

Estado	Descripción
UNINITIALIZED	Es el estado inicial de un thing cuando se agrega o se inicia en el sistema. Este estado también se asigna si el proceso de inicialización falló o el enlace no está disponible. Los comandos enviados a los canales no se procesarán.

INITIALIZING	Este estado se asigna mientras el enlace inicializa el thing. Depende del enlace cuánto tiempo lleva el proceso de inicialización. Los comandos enviados a los canales no se procesarán.
UNKNOWN	El controlador está inicializado, pero todavía se desconoce si el thing está ONLINE o OFFLINE. Es el propio controlador quién cambia el thing a ONLINE o OFFLINE.
ONLINE	El dispositivo/servicio representado por el thing funciona correctamente y puede procesar comandos.
OFFLINE	El dispositivo/servicio representado por el thing no funciona correctamente y es posible que no procese los comandos.
REMOVING	El dispositivo/servicio representado por el thing debe eliminarse, pero el enlace aún no ha confirmado la eliminación.
REMOVED	El dispositivo/servicio representado por el thing se eliminó del sistema.

Las posibles transiciones entre cada uno de estos estados se representan en el siguiente diagrama:

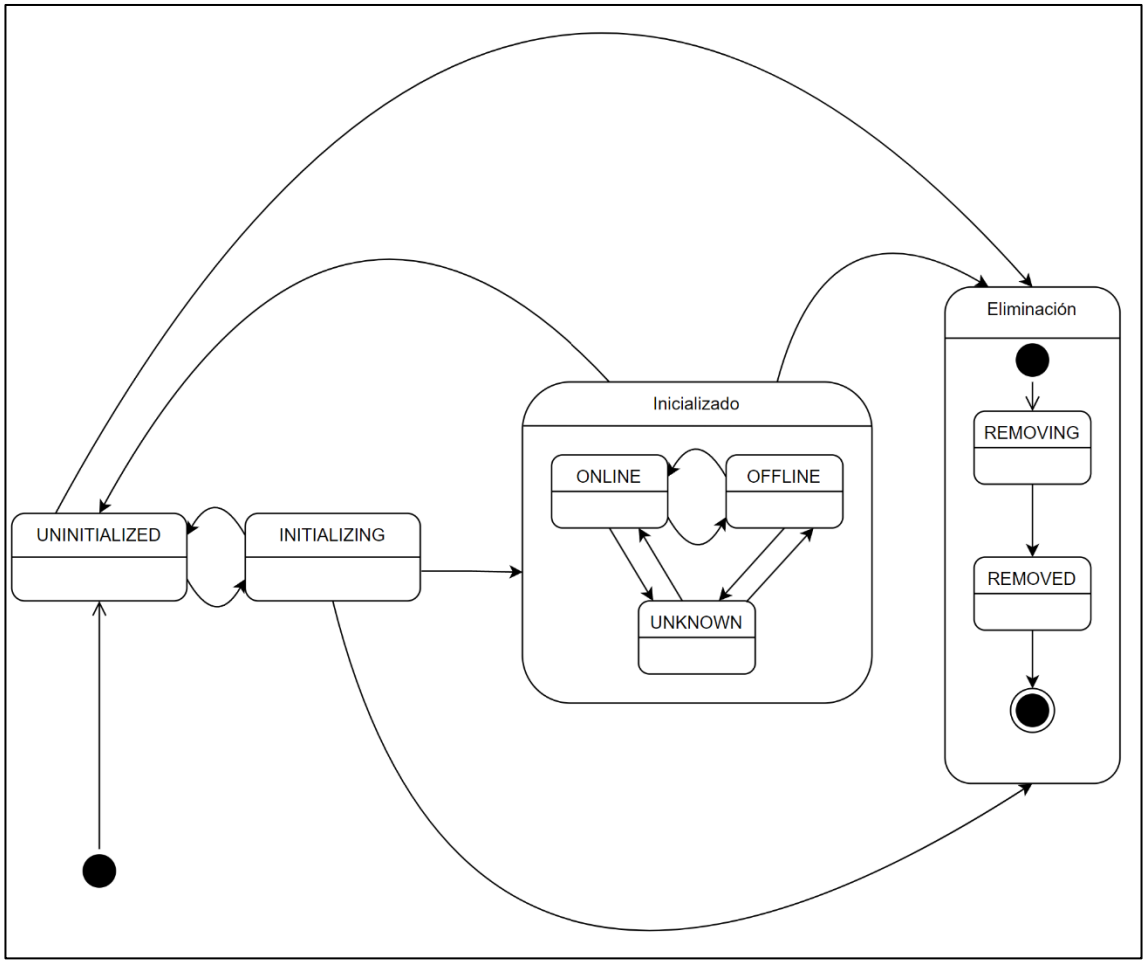

Figura 3-19: Diagrama de estados OpenHAB

Cada estado se especifica aún más con un detalle de estado. Se describen a continuación en la siguiente tabla:

Tabla 3-6: Detalles de los estados

Estado	Detalle de Estado	Descripción
UNINITIALIZED	NONE	Sin detalle.
	HANDLER_MISSING_ERROR	El enlace no está disponible o no se ha iniciado.
	HANDLER_REGISTERING_ERROR	Fallo en la fase de registro del servicio.
	HANDLER_CONFIGURATION_PENDING	No se puede inicializar ya que faltan parámetros de configuración.
	HANDLER_INITIALIZING_ERROR	Fallo en la fase de inicialización.
	BRIDGE_UNINITIALIZED	En puente asociado con el thing no está inicializado.
	DISABLED	El thing fue deshabilitado.
INITIALIZING	NONE	Sin detalle.
UNKNOWN	NONE	Sin detalle.
ONLINE	NONE	Sin detalle.
	CONFIGURATION_PENDING	El thing está esperando para transferir información a un dispositivo.
OFFLINE	NONE	Sin detalle.
	COMMUNICATION_ERROR	Error al comunicarse con el dispositivo (puede ser temporal).
	CONFIGURATION_ERROR	Un problema con la configuración del thing impide la

		comunicación con el dispositivo o servicio representado.
	BRIDGE_OFFLINE	El thing está fuera de línea porque el puente correspondiente lo está también.
	FIRMWARE_UPDATING	El thing está en proceso de actualización de firmware.
	DUTY_CYCLE	El thing está bloqueado para su uso posterior.
	GONE	El thing se eliminó del puente al que pertenecía y no está disponible para su uso.
REMOVING	NONE	Sin detalle.
REMOVED	NONE	Sin detalle.

La lista de todos los things instalados se puede ver en el apartado things del menú principal de OpenHAB:

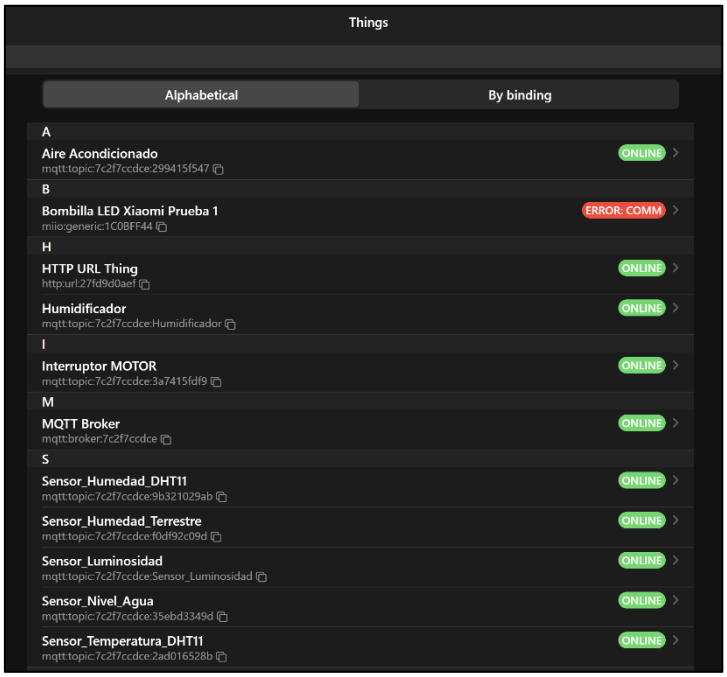

Figura 3-20: Lista de things instalados

En ella podemos encontrar:

- Aire Acondicionado: thing que representa al LED que sustituye un aire acondicionado.
- Bombilla LED Xiaomi: thing que representa la bombilla física.
- HTTP URL Thing: thing utilizado para obtener los límites ambientales establecidos por el usuario.
- Humidificador: thing que representa al LED que sustituye un humidificador.
- Interruptor MOTOR: thing empleado para representar el motor de agua.
- MQTT Broker: thing utilizado para establecer la conexión con el bróker MQTT.
- Sensor_Humedad_DHT11: thing representativo de la humedad del sensor DHT11.
- Sensor_Humedad_Terrestre: thing representativo del sensor de humedad terrestre.
- Sensor_Luminosidad: thing representativo de la fotorresistencia.
- Sensor_Nivel_Agua: thing representativo del sensor resistivo.
- Sensor_Temperatura_DHT11: thing representativo de la temperatura del sensor DHT11.

3.3.1.2 Channel

Los things exponen sus funcionalidades a través de los channels (canales). Que el sistema aproveche una capacidad concreta reflejada por un canal depende de si ha sido configurada para ello.

3.3.1.3 Item

Los ítems representan las capacidades que pueden ser utilizadas por las aplicaciones, ya sea en las interfaces de usuario o en la lógica de automatización. Los ítems tienen un estado y pueden recibir órdenes.

Los siguientes Items están disponibles actualmente en OpenHAB:

Tabla 3-7: Tipos de Items

Nombre del Item	Descripción	Tipos de comandos
Call	Identificar llamadas telefónicas.	Actualizar
Color	Información de color (RGB).	OnOff, AumentarDisminuir, Porcentaje, Actualizar
Contact	Estado de almacenamiento de Items.	AbiertoCerrado, Actualizar
DateTime	Almacena fecha y hora.	Fecha y hora

Dimmer	Item que lleva un valor porcentual para dimmers.	OnOff, AumentarDisminuir, Porcentaje, Actualizar
Group	Item para anidar otros Items o recopilarlos en grupos.	-
Image	Contiene los datos binarios de una imagen.	Actualizar
Location	Almacena coordenadas GPS.	Actualizar
Number	Almacena valores en formato numérico, toma un sufijo de dimensión opcional.	Decimales, Actualizar
Number:<dimension>	Como Number, pero con información de dimensión. adicional para dar soporte de unidad	Cantidad, Actualizar
Player	Permite controlar reproductores	ReproducirPausar, SiguienteAnterior, RebobinarAvance, Actualizar
Rollershutter	Normalmente usado para persianas	UpDown, StopMove, Porcentaje, Actualizar
String	Almacena textos	Cadena, Actualizar
Switch	Normalmente usado para luces	ActivarDesactivar, Actualizar

3.3.1.4 Link

El link (enlace) es el vínculo entre los things y los ítems, aunque concretamente, es una asociación entre un canal y un item. Si un canal está vinculado a un item, está "habilitado", es decir, que la capacidad que representa el item es accesible a través de ese canal. Los canales pueden estar vinculados a varios ítems y los ítems pueden estarlo a varios canales.

3.3.1.5 Binding

Los bindings se pueden considerar como adaptadores de software que permiten que los objetos estén disponibles para el sistema y proporcionar una forma de vincular los ítems a dispositivos físicos. También abstraen los requisitos específicos de comunicación de los dispositivos para que puedan ser tratados de forma más genérica por el framework.

3.3.1.5.1 Bindings instalados

Los bindings que han sido instalados son:

- **HTTP Binding**: utilizado para establecer una conexión HTTP con el servidor REST.

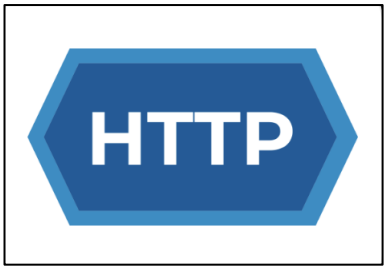

Figura 3-21: Binding HTTP

Para crear el thing 'HTTP URL Thing' mostrado en la figura 3-19 utilizando este binding se tienen que seguir los siguientes pasos:

1. Se especifica el tipo de thing que se quiere crear, en este caso perteneciente al binding HTTP.
2. Para su configuración, los principales parámetros que se tienen que indicar son:
   - La URL "http://192.168.1.62:9000/servidor/valoresLimite" a la que se hará la petición (se detallará en el apartado 5.2 su uso).
   - El método básico para la autenticación.
   - El método HTTP GET.
   - Dado que el servidor devuelve para una consulta dirigida a esta dirección un JSON, el valor especificado en el atributo "Content-Type" será 'application/json'.
3. Posteriormente, es necesario crear los ítems y enlazarlos con el thing mediante los channels. Se explicará a modo de ejemplo como se han configurado el item y el channel para la obtención del límite de la temperatura.

   Dado que el resultado de la URL anterior es un JSON con todos los valores límite, en el channel hay que establecer que dato del JSON se quiere representar a través del item. Para ello, se modifica el atributo 'State Tranformation' estableciéndolo a 'JSONPATH:$.Temperatura'. De esta forma, se obtiene el valor de la temperatura indicado en el JSON por el nombre de Temperatura.

   Por otro lado, se específica que el canal solo se utilizará para leer el dato recibido al realizar la consulta mediante el atributo 'Read/Write Mode'.

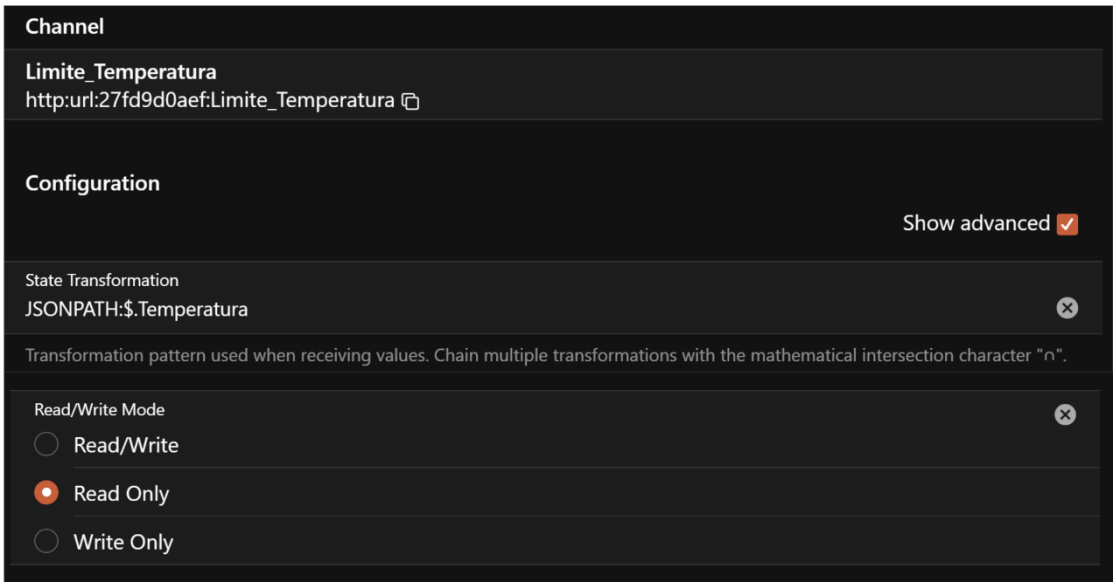

Figura 3-22: Configuración channel de HTTP URL Thing

4. Por último, al añadir el nuevo Item al canal llamado HTTP_URL_Thing_Limite Temperatura, se especifica que el tipo de dato representado es de tipo “String”.

- **MQTT Binding**: utilizado para enviar y recibir datos a la placa ESP32.

Figura 3-23: Binding MQTT

El bróker MQTT empleado para la interconexión es el servidor público proporcionado por EMQX Cloud [30]. En la siguiente figura se muestra toda la configuración necesaria para ejecutar dicho servidor.

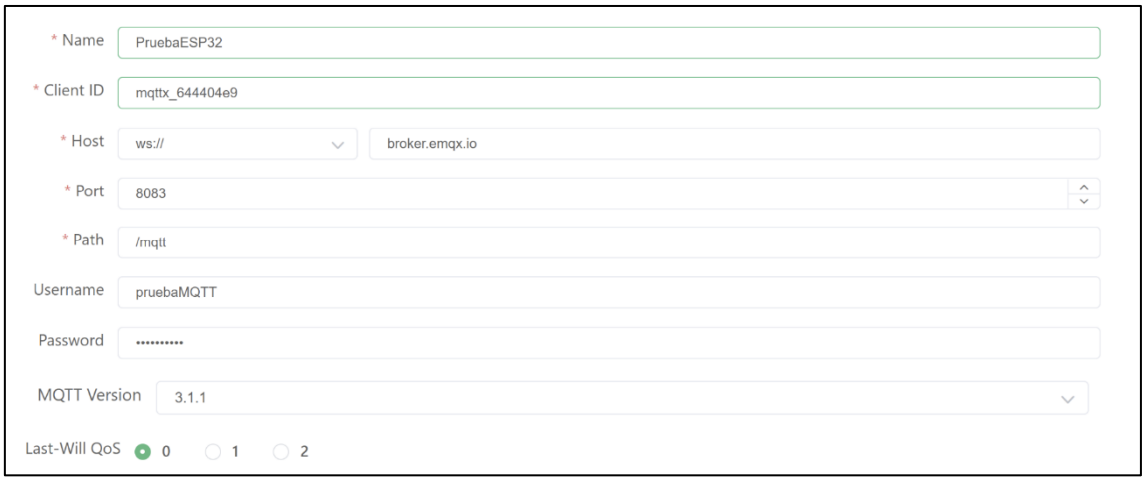

Figura 3-24: Configuración Bróker MQTT

Se especifica un ID de cliente generado automáticamente, la dirección 'broker.emqx.io' donde se aloja el servidor acompañado por el puerto 8083 (este puerto es utilizado para WebShocket, el puerto TCP es el 1883), además del usuario y contraseña 'pruebaMQTT' utilizado por los clientes para poder acceder. Por último, se indica que la versión del protocolo MQTT es la 3.1.1 y el nivel de QoS = 0 (en el apartado 4.1 se explicará el significado de este último parámetro).

Para obtener los datos recibidos del servidor MQTT en OpenHAB, es necesario crear un thing que representa al propio bróker, además del empleado para recibir la información de un topic determinado. Su nombre será 'MQTT Broker' y los pasos necesarios para su configuración son:

1. Igual que el paso número 1 del binding HTTP para crear el thing representativo del bróker MQTT.
2. Se indican en los atributos correspondientes la dirección 'broker.emqx.io' y el puerto 1883 donde se ejecuta el servidor, la versión 3.1.1 de MQTT, el ID del cliente proporcionado en la configuración del servidor, el nivel QoS = 0 y las credenciales de usuario y contraseña anteriormente mencionadas.

Todos los topics creados se explican en el apartado 4.1.4.

Por último, los pasos para la configuración del thing que permite obtener la información de un topic determinado en OpenHAB, son: (se utilizará como ejemplo el thing llamado Sensor_Temperatura_DHT11)

1. Igual que el paso número 1 del binding HTTP.
2. Establecer la dirección del bróker al que se va a conectar enlazándolo con el thing 'MQTT Broker' creado anteriormente en el atributo 'Parent Bridge':

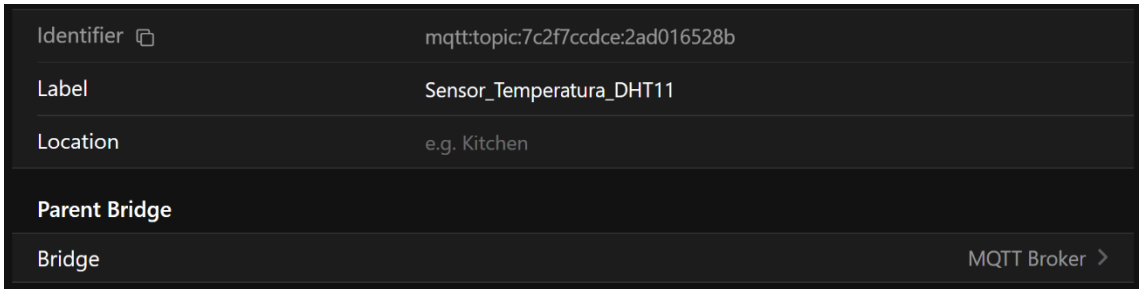

Figura 3-25: Enlace con el thing MQTT Broker

3. Al configurar el channel, se indica el topic 'Sensor/DHT11/Temperatura' para recibir la información de la temperatura enviada desde el bróker.
4. Al enlazarlo con el Item 'SensorTemperatura_DHT11' se especifica el tipo de valor recibido, en este caso de tipo String.

- **Xiaomi Wifi devices (Mi IO) Binding**: utilizado para establecer la conexión con la bombilla Xiaomi.

Figura 3-26: Binding Xiaomi

Los pasos necesarios para configurar el thing 'Bombilla LED Xiaomi Prueba 1' son:

1. Mismo paso que en los anteriores.
2. Especificar la dirección IP, el Token y el Device ID de la bombilla. Para conseguir estos datos, es necesario ejecutar en una terminal el programa de extractor de tokens mencionado en el apartado 2.2.8.

```
Username (email or user ID):
jesusvinuesa.02@gmail.com
Password:

Server (one of: cn, de, us, ru, tw, sg, in, i2) Leave empty to check all available:
de

Logging in...
Logged in.

Devices found for server "de" @ home "444001057769":
   ---------
   NAME:     Mi Smart LED Bulb Essential (White and Color)
   ID:       470548292
   MAC:      EC:4D:3E:3D:A5:3A
   IP:       192.168.1.58
   TOKEN:    6c5ef3e60d75834d2256144686b3428f
   MODEL:    yeelink.light.color5
   ---------
```

Figura 3-27: Ejecución del programa extractor de tokens de dispositivos Xiaomi

Para obtener esta información, es necesario disponer de una cuenta en Xiaomi Cloud.

3. Crear un channel por cada Item que representa la funcionalidad de la bombilla, como puede ser su encendido/apagado, cambio de color, cambio de la intensidad luminosa, etc.

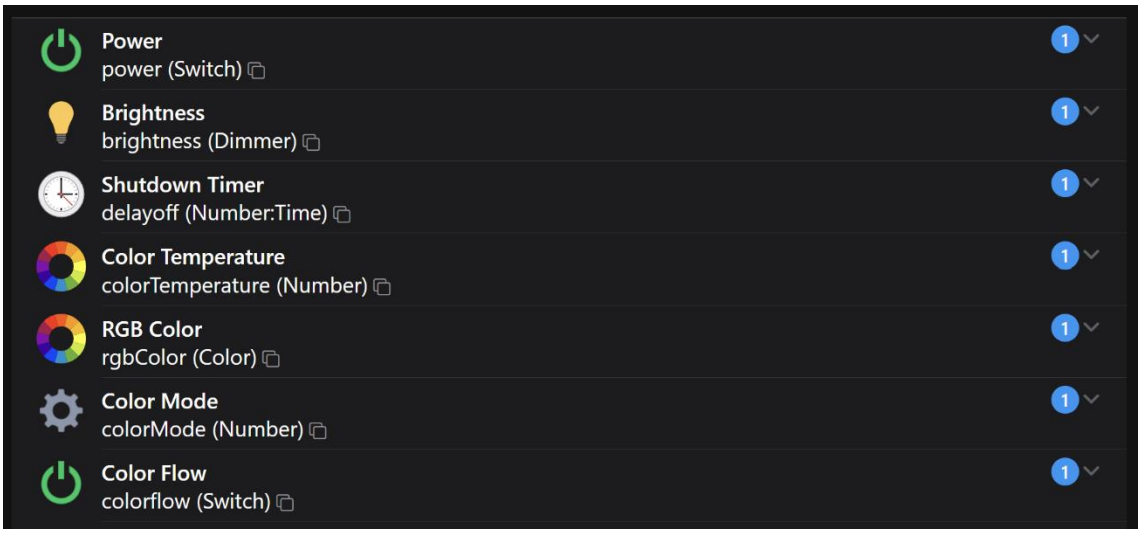

Figura 3-28: Channels para el thing de la bombilla de Xiaomi

3.3.2 Reglas

OpenHAB permite automatizar el control de los dispositivos físicos a través de las reglas. Estas se pueden componer con los siguientes bloques:

Tabla 3-8: Estructura de una regla en OpenHAB

Bloque	Objetivo
Trigger o evento	Cuando ocurre el evento definido, se ejecuta la regla. (Opcional)
Condition o condición	Condición que debe cumplirse para que se ejecute la regla. (Opcional)
Action o acción	Sobre qué se debe actuar cuando se ejecute la regla

De forma que la definición de una regla en OpenHAB siempre será del tipo: "*Cuando ocurra (evento), si (condición), entonces haz (acción)*".

3.3.2.1 Triggers

Como se ha mencionado anteriormente, los triggers definen aquellos eventos que, cuando ocurren, hacen que se ejecute la regla.

Las categorías de reglas que se pueden utilizar para activar una regla son:

Tabla 3-9: Triggers

Evento	Descripción
Items	Comandos, actualizaciones y cambios en el estado de un item individual.
Groups	Los grupos son ítems especiales que tienen a otros ítems como miembros.
Time	Las reglas pueden activarse en momentos específicos.
Channel	Algunos things tienen canales que pueden desencadenar reglas directamente.
Thing	Cuando cambian o actualizan su estado.
System	Cuando ocurren eventos importantes internas de OpenHAB.

Para los triggers basados en el tiempo, se pueden usar expresiones predefinidas para los temporizadores o usar expresiones cron [26], que son cadenas que están compuestas por otras seis (siete opcionalmente) subexpresiones para una mayor precisión. Estas subexpresiones se separan por espacios en blanco y representan, en el siguiente orden:

Tabla 3-10: Subexpresiones

Momento temporal	Valores permitidos
Segundos	Del 0 al 59.
Minutos	Del 0 al 59.
Horas	Del 0 al 23.
Días del mes	Del 1 al 31 (dependiendo del mes).
Mes	Del 0 al 11 (siendo 0 el mes de enero) o utilizando las cadenas: JAN, FEB, MAR, APR, MAY, JUN, AUG, SEP, OCT, NOV, DEC.
Días de la semana	Del 1 al 7 (siendo 1 el domingo) o mediante las cadenas: SUN, MON, TUE, WED, THU, FRI, SAT.
Año (Opcional)	Del 1970 al 2099

Las subexpresiones individuales pueden contener rangos y/o listas. Por ejemplo, en el campo día de la semana, se podría poner “MON-FRI”, “MON, WED, FRI” o “MON-WED, SAT”.

Existen también una serie de comodines que permiten simplificar la sintaxis. Estos son:

- El carácter ‘*’ indica todos los posibles valores para ese campo. Por ejemplo, en el campo día de la semana, un ‘*’ indicaría “todos los días de la semana”.
- El carácter ‘/’ se puede utilizar para especificar incrementos en los valores. Por ejemplo, en el campo de los minutos, ‘0/15’ o ‘/15’ indicaría “cada 15 minutos de la hora, comenzando en el minuto 0”. ‘3/20’ indicaría “cada 20 minutos de la hora, comenzando en el minuto 3”.
- El carácter ‘?’ se utiliza para los campos del día del mes y día de la semana. Este comodín indica “ningún valor específico”. Resulta útil cuando se necesita especificar algo en uno de los dos campos, pero no en el otro.
- El carácter ‘L’ se utiliza también para los campos anteriormente mencionados. Indica “el último”, pero su uso tiene un significado distinto en cada campo. En el día del mes significa “el último día del mes”. En el correspondiente del día de la semana, indica “7” o “SAT” (sábado). Pero si se usa en este campo después de otro valor, indicaría “el último día (día) del mes”. Por ejemplo, “6L” o “FRIL” significaría “el último viernes del mes”. También se puede utilizar este comodín para especificar los días contando desde el último. Por ejemplo, “L-3” quiere decir “el penúltimo día”.
- El carácter ‘W’ se utiliza para especificar los días de la semana (de lunes a viernes) más cercano al día dado. Por ejemplo, poner “15W” en el campo del día del mes significaría “el día de la semana más cercano al día 15 del mes”.
- El carácter ‘#’ se utiliza para especificar “el enésimo” día de la semana del mes. Por ejemplo, el valor “6#3” o “FRI#3” en el campo del día de la semana indicaría “el tercer viernes del mes”.

Para poder utilizar estas expresiones para la planificación temporal, OpenHAB se apoya en el framework Java de Quartz, que destaca por su potencia y sencillez.

3.3.2.2 Conditions

Las condiciones pueden limitar cuándo se puede ejecutar una regla agregando una o varias de ellas. Existen distintos tipos de condiciones disponibles, como:

- Cuando un item tiene un determinado estado.
- Cuando es una hora determinada del día.
- Cuando es un día determinado de la semana o es festivo, fin de semana, etc.
- Cuando se evalúa la condición de un script.

La condición de script permite elegir uno de los muchos lenguajes de script/reglas que dispone OpenHAB para crear cualquier condición, con la restricción de que la última línea de código debe ser el resultado de un valor booleano (true o false).

3.3.2.3 Actions

Las acciones definen los elementos sobre los que se actúan cuando se cumple una regla. Estas acciones pueden ser:

- Habilitar o deshabilitar otras reglas
- Ejecutar otras reglas
- Enviar comandos
- Actualizar el estado de un ítem
- Reproducir algún sonido
- Ejecutar un script

3.3.2.4 Reglas programadas

En el proyecto se han creado diferentes reglas para la automatización del sistema, todas ellas basadas en scripts. Para su programación, se ha utilizado el complemento denominado JavaScript Scripting:

Figura 3-29: Binding JavaScript

Este add-on proporciona soporte para JavaScript (ECMAScript 2022+) y además incluye openhab-js, una biblioteca para apoyar la automatización en OpenHAB. Proporciona un cómodo acceso a la funcionalidad común de OpenHAB dentro de las reglas, incluyendo items, things, acciones, etc.

Existen cuatro reglas utilizadas para el envío de los datos procedentes de los sensores al servidor REST cuando los ítems encargados de almacenarlos cambian de estado. Observando como ejemplo la regla para el envío del nivel de luminosidad, su sintaxis es la siguiente: [Cuando cambie el estado del sensor de la luminosidad, entonces ejecuta el script llamado "EnviarDatoLuminosidad"]. Su configuración se muestra a continuación:

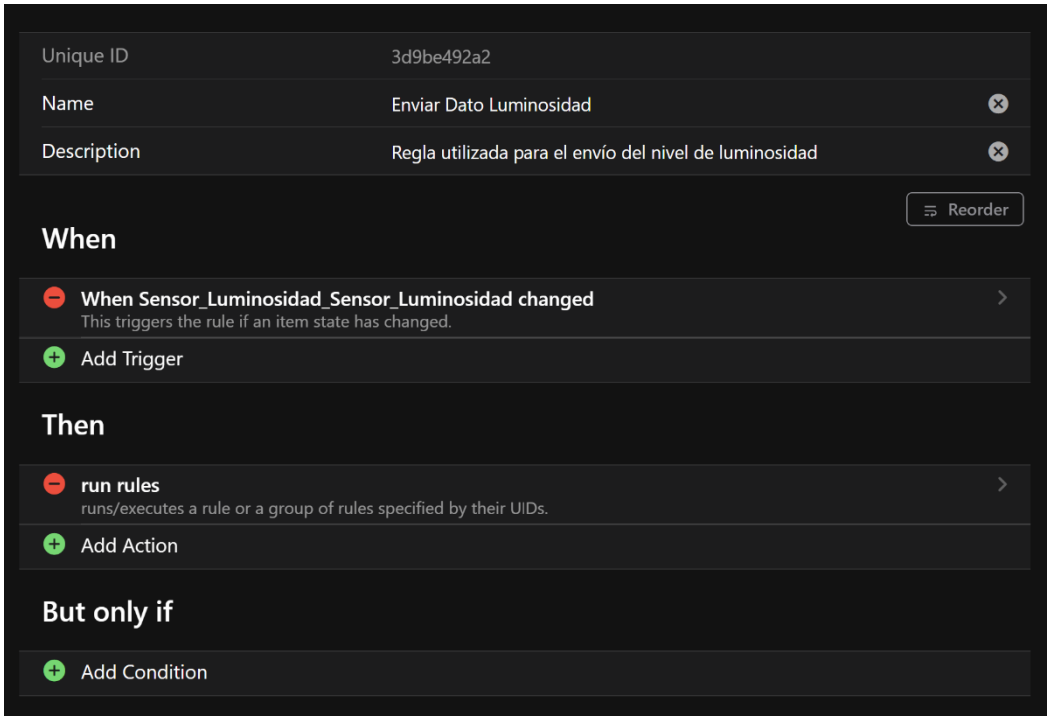

Figura 3-30: Regla OpenHAB para el envío de la luminosidad

Como se puede apreciar, no se establece ninguna condición dentro de la regla. A continuación, se muestra el código del script que se ejecuta cuando se cumple la regla:

```
var luminosidad = items.getItem("Sensor_Luminosidad_Sensor_Luminosidad").state;

actions.HTTP.sendHttpGetRequest("http://192.168.1.62:9000/servidor/datoInt?tipo=SensorLuminosidad&dato=" + luminosidad)
```

Figura 3-31: Código de la primera regla OpenHAB

Este almacena en una variable el estado del item que recoge los datos del sensor y mediante el método '*sendHttpGetRequest',* los envían a la dirección URL especificada.

Otra regla es utilizada para el establecimiento de límites ambientales. Por ejemplo, cuando el usuario establezca un límite de temperatura para el invernadero de unos 21 ºC, y el sensor DHT11 recoja una temperatura ambiental de 25ºC, OpenHAB encenderá automáticamente mediante esta regla el aire acondicionado para que las condiciones se ajusten a las establecidas desde la aplicación móvil. Su configuración es:

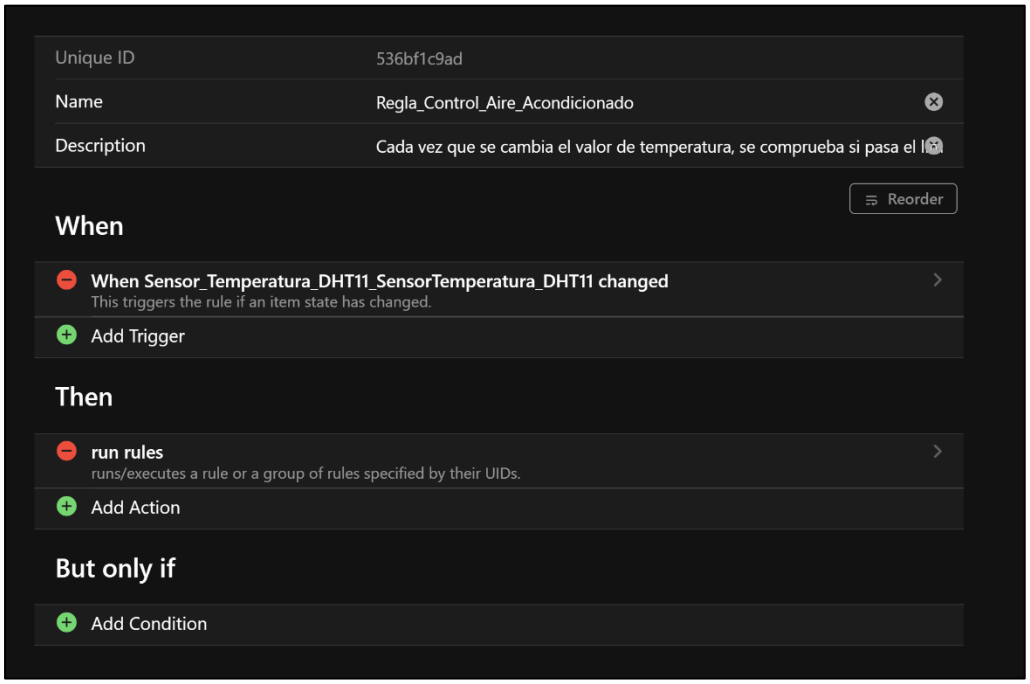

Figura 3-32: Regla OpenHAB para comprobar la temperatura del invernadero

Igual que en la regla anterior no se establecen límites (ya que éstos se comprueban directamente en el script). La configuración del script se muestra a continuación:

```
var aireEncendido = false;
var humidificadorEncendido = false;


var limiteTemperatura = items.getItem("HTTP_URL_Thing_Limite_Temperatura").state;
console.log("Valor límite de la temperatura: ", limiteTemperatura);

var limiteHumedad = items.getItem("HTTP_URL_Thing_Limite_Humedad").state;
console.log("Valor límite de la humedad: ", limiteHumedad);

var temperaturaActual = items.getItem("Sensor_Temperatura_DHT11_SensorTemperatura_DHT11").state;
console.log("Temperatura actual: ", temperaturaActual);

var humedadActual = items.getItem("Sensor_Humedad_DHT11_SensorHumedad_DHT11").state;
console.log("Humedad actual: ", humedadActual);

if (temperaturaActual > limiteTemperatura) {
  console.log("Se supera el límite de temperatura --> Se enciende el aire acondicionado");
  items.getItem("Aire_Acondicionado").sendCommand("ON");
} else {
  console.log("No se supera el limite de temperatura");
  items.getItem("Aire_Acondicionado").sendCommand("OFF");
}

if (humedadActual > limiteHumedad) {
  console.log("Se supera el límite de humedad --> Se enciende el humidificador");
  items.getItem("Humidificador_Canal_Humidificador").sendCommand("ON");
} else {
  console.log("No supera el limite de humedad");
  items.getItem("Humidificador_Canal_Humidificador").sendCommand("OFF");
}
```

Figura 3-33: Código de la regla OpenHAB encargada de comprobar la temperatura y humedad

En este script se recogen en variables los límites de temperatura y humedad establecidos por el usuario y las condiciones ambientales actuales. Posteriormente, se enviará una señal de ON/OFF a los ítems que representan el aire acondicionado y el humidificador, dependiendo de si se cumplen las condiciones previstas para el invernadero.

3.3.3 Persistencia

Aunque este módulo de OpenHAB no se va a implementar finalmente en el proyecto, sí que se ha trabajado y estuvo funcionando conjuntamente con la base de datos, y por ello se procederá a explicar brevemente su funcionalidad y configuración. La decisión de no utilizarlo fue porque se consideró que para evitar problemas de inserción y aumentar la eficiencia de las consultas a la base de datos, ésta se gestionará únicamente desde el propio servidor REST y no desde este y el servidor OpenHAB.

OpenHAB permite almacenar datos a lo largo del tiempo mediante este módulo. Para ello se puede utilizar el servicio de persistencia que viene por defecto instalado llamado rrd4j ó utilizar otros complementos dedicados a este propósito. Entre todos los posibles servicios adicionales, se escogió el complemento llamado JDBC Persistence PostgreSQL:

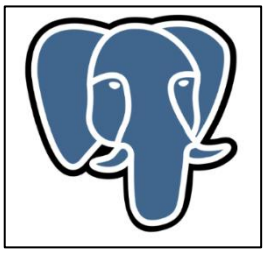

Figura 3-34: Binding PostgreSQL

Este servicio escribe y lee estados de ítems desde y hacia una serie de sistemas de base de datos relacionales que admiten Java Database Connectivity (JDBC). Para configurarlo, es necesario establecer este servicio de persistencia como predeterminado para OpenHAB y editando el fichero *services/runtime.cfg,* dentro de la carpeta raíz, donde se ha instalado OpenHAB, y añadir la siguiente línea:

```
org.openhab.persistence:default=jdbc
```

Figura 3-35: Selección persistencia tipo jdbc

Posteriormente, en otro fichero llamado *services/jdbc.cfg*, se indicarán todas las propiedades necesarias para la configuración del complemento para la persistencia, como la dirección URL de la base de datos en la que se almacenarán los datos, su usuario y contraseña si hubiese, etc. Un ejemplo de esta configuración, para una base de datos llamada openhabdb, sería:

```
url=jdbc:postgresql://192.168.1.62:5432/openhabdb
user=openhab
password=openhab
```

Figura 3-36: Configuración acceso base de datos

Por último, para indicar qué ítems se quieren almacenar y con qué frecuencia, se configura el archivo llamado *persistence/jdbc.persist*:

```
Strategies{
    everyMinute: "0 * * * * ?"
    every5Minutes : "0 */5 * * * ?"
    everyHour : "0 0 * * * ?"
    everyDay : "0 0 0 * * ?"
    default = everyChange
}
Items {
        Temperatura_Sensor_DHT11 : strategy = everyChange, everyDay, restoreOnStartup

        Sensor_Humedad_DHT11 : strategy = everyChange, everyDay, restoreOnStartup

        Sensor_Humedad_Terrestre_Sensor_Humedad_Terrestre7 : strategy = everyChange, everyDay, restoreOnStartup

        Sensor_Luminosidad : strategy = everyChange, everyDay, restoreOnStartup

        Sensor_Nivel_Agua : strategy = everyChange, everyDay, restoreOnStartup

}
```

Figura 3-37: Configuración items persistidos

Como se puede ver, existen dos campos:

- Strategies: utilizado para especificar la frecuencia con la que se envían los datos a la base de datos. En este campo se crean variables de tiempo que serán utilizadas posteriormente utilizando las mismas reglas que en el que se comentó anteriormente en el apartado Triggers utilizando las expresiones Cron.

- Items: en él se especifican los ítems de los que se quieren guardar sus estados, así como con qué frecuencia se envía cada uno utilizando las variables de tiempo definidas anteriormente.

3.3.4 API REST de OpenHAB

A través de la API REST de OpenHAB se puede acceder fácilmente a la mayoría de los aspectos del sistema desde otros programas. Esto incluye acceso a todos los datos relacionados con Ítems, Things, Bindings, así como las capacidades para invocar acciones que pueden cambiar el estado de Ítems o influir en el comportamiento de otros elementos de OpenHAB. Las interacciones con la API REST se basan en el protocolo HTTP.

Para acceder a toda la información relativa a las direcciones URL para las consultas, sus parámetros necesarios, etc, hay que dirigirse a Herramientas de Desarrollo > API Explorer.

Aunque esta interfaz nos muestra una amplia cantidad de información sobre todo lo que se puede configurar o consultar a través de la API REST, este trabajo solo se ha enfocado en la parte de los Items. A través de ella podemos cambiar su estado, consultar cuantos hay instalados, eliminarlos, etc.

A continuación, se muestra una imagen en la que se exponen cada una de las URLs válidas, los métodos HTTP necesarios para realizar las peticiones y una pequeña descripción con sus funcionalidades:

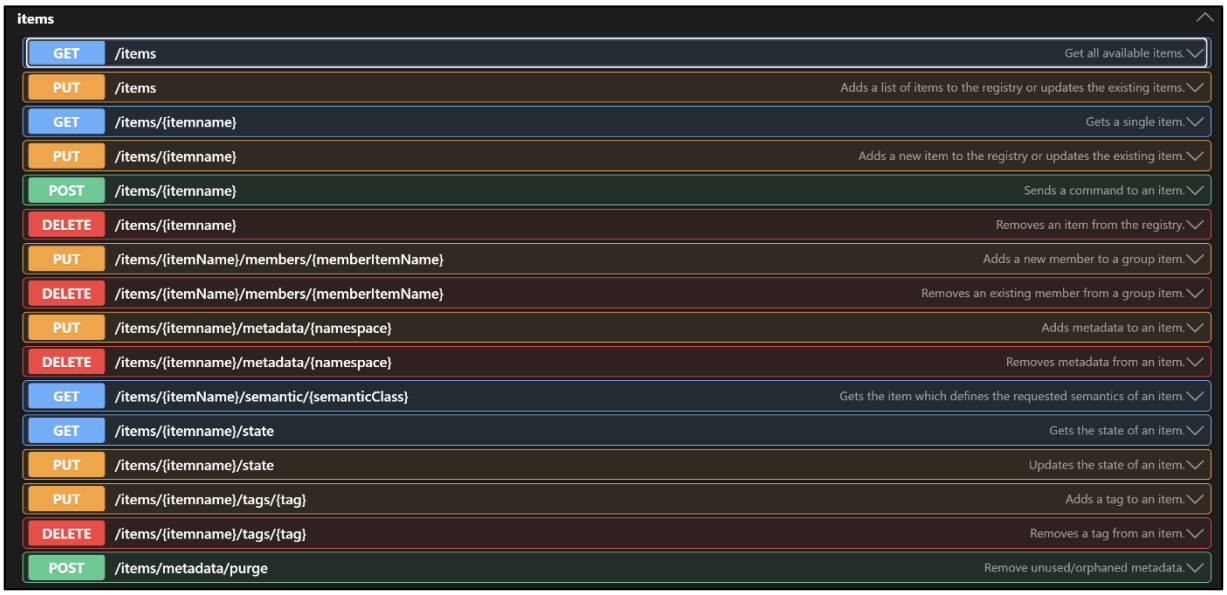

Figura 3-38: Api REST OpenHAB

Concretamente, la que se utilizará será la quinta URL, con la que podremos enviar comandos a los Ílems utilizando el método POST.

Esta dirección URL se compone de:

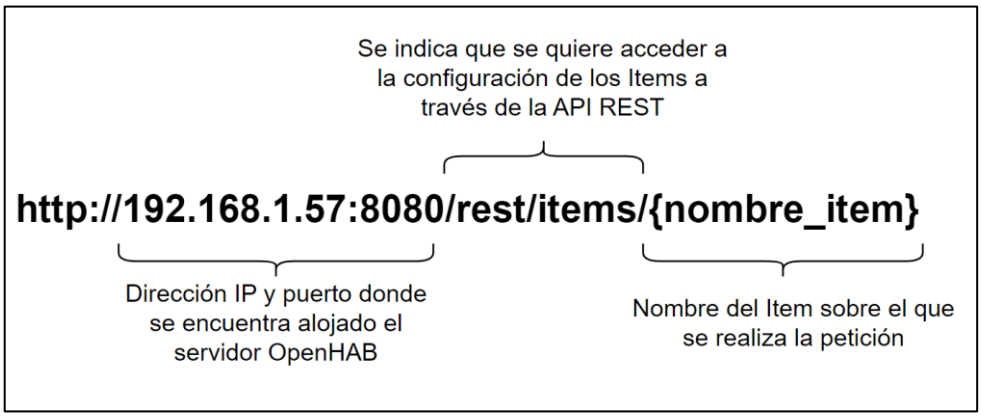

Figura 3-39: Estructura URL

A partir de la versión 3, OpenHAB admite la protección por contraseña para contenidos confidenciales. Para acceder a este tipo de información, la API REST proporciona el mecanismo de Autenticación Básica y el de Autorización OAuth. Ambos mecanismos pueden ser utilizados por la mayoría de los lenguajes de programación y frameworks.

- Autenticación de acceso básica: es un método para que un cliente HTTP proporcione un nombre de usuario y una contraseña al realizar una solicitud. En la autenticación HTTP básica, una solicitud contiene un campo de encabezado en forma de Authorization: Basic <credentials>, donde las credenciales son la codificación Base64 de usuario y contraseñas unidas por dos puntos (:). Esta opción de autenticación hay que habilitarla en la configuración

de OpenHAB desde el menú principal > Configuración > API Security:

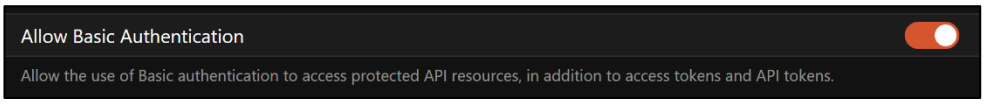

Figura 3-40: Configuración autenticación de acceso básica

- Autorización OAuth (Autorización abierta): es un estándar abierto para la delegación de acceso. Este protocolo proporciona una forma para que los propietarios de recursos proporcionen a un cliente acceso seguro a los recursos del servidor sin necesidad de proporcionar credenciales. Está diseñado específicamente para trabajar con HTTP, que permite a un servidor proporcionar tokens de acceso a clientes, con los que pueden acceder a los recursos protegidos alojados en él. Para crear tokens desde OpenHAB, hay que dirigirse al perfil, pulsar en "Crear nuevo token API" e introducir los datos que se solicitan:

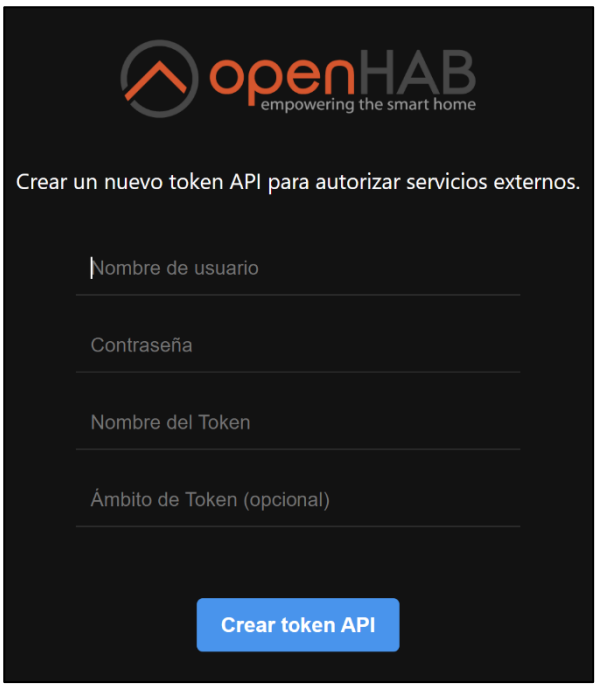

Figura 3-41: Configuración acceso OAuth

Al crearse el token, se mostrará una vez para que el nuevo cliente pueda usarlo en sus peticiones y acceder a todos los recursos del servidor:

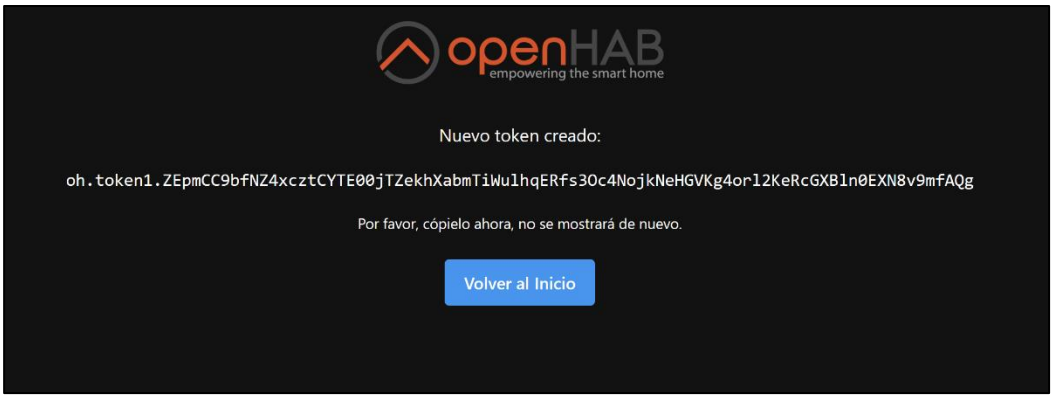

Figura 3-42: Token generado

Opcionalmente y en el caso del ordenador donde se ha instalado el servidor de OpenHAB, ha sido necesario introducir una nueva regla en el cortafuegos de Windows para que este aceptase todas las peticiones provenientes de la subred en la que se ha establecido el sistema. Para ello, dirigiéndose a Panel de control > Sistema y Seguridad > Firewall de Windows Defender > Configuración Avanzada > Regla de entrada > Nueva regla se ha configurado como se muestra a continuación:

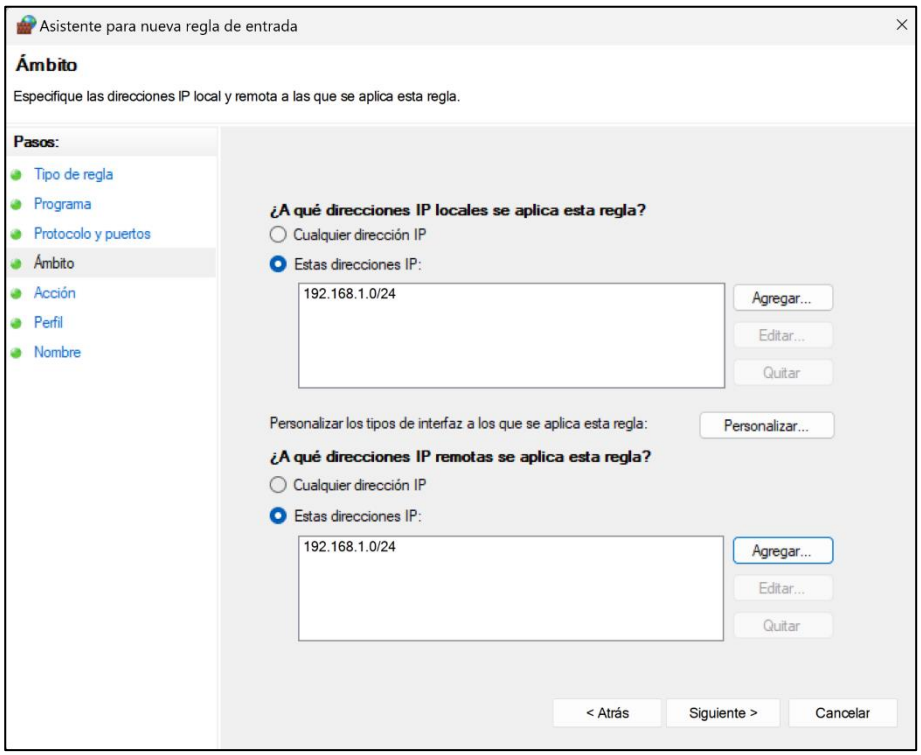

Figura 3-43: Configuración regla del cortafuegos

Aplicando una nueva regla personalizada y añadiendo la dirección IP de la subred de donde pueden proceder las peticiones, se modifica el cortafuegos.

3.3.5 Conexión Kafka-OpenHAB

Tal y como se comentó en el apartado 3.2, Kafka necesita un consumidor suscrito a los topics del bróker y leer los mensajes que el productor haya enviado. En este contexto, OpenHAB actúa como ese consumidor que ejecuta las acciones necesarias según los comandos de voz transcritos por Voice2JSON. Sin embargo, hasta ahora OpenHAB no cuenta con ningún Binding ni complemento que le permita integrarse directamente con el sistema Kafka. Por ello, la solución que se ha adoptado se basa en un código de lenguaje Python, que es el componente intermediario que sirve de enlace entre el consumidor de Kafka y OpenHAB.

Este fichero incorpora la librería KafkaConsumer de la biblioteca Kafka, mediante la cual, nos podemos conectar directamente al bróker. Para su configuración, se especifican los siguientes atributos:

```
consumer = KafkaConsumer(
    'thing1',
    bootstrap_servers=['localhost:9092'],
    auto_offset_reset='earliest',
    enable_auto_commit=True,
    group_id='my-group',
    value_deserializer=lambda x: loads(x.decode('utf-8'))
)
```

Figura 3-44: Configuración Kafka en Python

- El topic del que se quiere leer los mensajes, en este caso “thing1”.
- bootstrap_servers: cadena o lista que indica el host:port donde se ubica el clúster.
- auto_offset_reset: configurado a ‘earliest’ se especifica que se quiere leer todo, desde el principio del topic.
- enable_auto_commit: configurado a “True” el offset del consumidor se confirmará periódicamente en segundo plano.
- group_id: indica el nombre de consumidores al que unirse para la asignación dinámica de particiones (si está habilitado).
- value_deserializer: deserializa los datos codificados en ‘UTF-8’ en un formato JSON.

En la siguiente parte del código, se extrae el mensaje del campo ‘text’ del JSON obtenido del consumidor Kafka, que contiene el comando a ejecutar.

```
    mensaje = mensaje.value
    print('{} consumido'. format(mensaje))

    comando_ejecutar = mensaje['text']
```

Figura 3-45: Obtención del mensaje de Kafka

El comando se almacena en la variable '*comando_ejecutar*'. La conexión con OpenHAB se realiza a través de su API REST, utilizando una librería de Python llamada 'openhab' que soporta los dos mecanismos de autenticación comentados anteriormente. Por cuestiones de rendimiento, el elegido ha sido el modo básico, de modo que solo se tendrá que indicar el usuario y la contraseña del perfil de OpenHAB, además de la dirección URL con la que se podrá cambiar el estado de los Items (dicha dirección, como se mencionó en apartados anteriores, se obtiene de la interfaz API Explorer de OpenHAB):

```
url_base = 'http://192.168.1.57:8080' #Direccion local donde está alojado el servidor openhab y su puerto
url_rest = f'{url_base}/rest/items'
```

Figura 3-46: Configuración para el acceso al servicio REST de OpenHAB

```
oh = openhab.OpenHAB(base_url=url_rest, username="admin", password="admin")
```

Figura 3-47: Conexión con el servicio REST de OpenHAB con autenticación básica

Posteriormente, dependiendo de la orden almacenada en la variable '*comando_ejecutar*', se realizará un tipo de petición u otro utilizando el método req_post. Por ejemplo, para el caso en el que se quiera apagar la bombilla, el cambio de su estado se realiza a través del código que se muestra:

```
if (comando_ejecutar == 'apaga la bombilla' or comando_ejecutar == 'apaga bombilla' or comando_ejecutar == 'Apaga la luz'):
    oh.req_post("/Bombilla_inteligente_Mi_LED_Essential_blanca_y_color_470548292__1C0BFF44_with_token_Power", data="OFF")
```

Figura 3-48: Posibles opciones para apagar la bombilla

En el método 'req_post' se indica el nombre del Item al que va dirigido el comando. Además, en la variable 'data' se indica el nuevo estado que tendrá la bombilla.

Como se puede apreciar en la imagen, existen distintas alternativas para apagar la bombilla. Las dos primeras se corresponden con las posibilidades que se pueden obtener al convertir el comando de voz a texto a través de Voice2JSON. La tercera posibilidad se trata de la orden que puede proceder directamente del servidor REST, es decir, no se trata de un comando de voz, y por tanto no es el resultado de una conversión de Voice2JSON. Este procedimiento se aplica por igual a todos los actuadores del invernadero, cambiando únicamente la parte de la URL que identifica a cada uno de ellos en OpenHAB.

4 COMUNICACIÓN MQTT ENTRE OPENHAB Y LA PLACA ESP32

"En principio, la investigación necesita más cabezas que medios"

Severo Ochoa

En este capítulo se explicará cómo se realiza la gestión y recogida de datos de actuadores y sensores utilizando Arduino. La conexión entre OpenHAB y la placa ESP32 mediante el protocolo MQTT permitirá el envío de estos datos a través de los distintos topics configurados, clasificando el flujo de información que se envíe.

4.1 MQTT

MQTT (acrónimo de Message Queuing Telemetry Transport) es un protocolo de red ligero, de tipo publicador/suscriptor, M2M para el servicio de colas de mensajes. Está diseñado para conexiones con ubicaciones remotas que tienen dispositivos con restricciones de recursos o ancho de banda limitado. Se ejecuta sobre un protocolo de transporte que proporcione conexiones ordenadas, sin pérdidas y bidireccionales, generalmente TCP/IP. La versión de uso común es la 3.1.1, aunque existe la versión 5.0.

4.1.1 Componentes básicos de MQTT

Los elementos básicos MQTT son:

4.1.1.1 Conexión de red

Construcción proporcionada por el protocolo de transporte subyacente que está siendo utilizado por MQTT. Permite la conexión entre el cliente y el servidor a través de un medio apto para enviar un flujo ordenado de datos, sin pérdidas y en ambas direcciones.

4.1.1.2 Mensaje de aplicación

Son los datos transportados por el protocolo MQTT a través de la red para la aplicación. Cuando los mensajes de aplicación son transportados por MQTT, estos tienen asociados un QoS (calidad de

servicio) y un nombre de topic.

4.1.1.3 Cliente

Un programa o dispositivo que utiliza MQTT. Siempre establece la conexión de red con el servidor.

Es capaz de:

- Publicar mensajes de aplicación que puedan interesar a otros clientes.
- Suscribirse a topics para solicitar mensajes de aplicación.
- Darse de baja para eliminar una solicitud de mensajes de aplicación.
- Desconectarse del servidor.

4.1.1.4 Servidor o Bróker

Programa o dispositivo que actúa como intermediario entre aquellos clientes que publican mensajes de aplicación y aquellos que han realizado suscripciones. Es capaz de:

- Aceptar conexiones de red de clientes.
- Aceptar mensajes de aplicación publicados por clientes.
- Procesar las solicitudes de suscripción y cancelación de suscripción de los clientes.
- Reenviar los mensajes de aplicación que coinciden con las suscripciones de los clientes.

4.1.1.5 Suscripción

Una suscripción comprende un filtro de topics y una QoS máxima. Está asociada a una única sesión y una única sesión puede contener más de una suscripción.

4.1.1.6 Nombre de topic

Es una etiqueta adjunta a un mensaje de aplicación que se utiliza para relacionarlo con las suscripciones conocidas por el servidor. El servidor envía una copia del mensaje a cada cliente que tenga una suscripción coincidente.

4.1.1.7 Filtro de topics

Se trata de una expresión contenida en una suscripción, para indicar un interés en uno o más topics.

4.1.1.8 Sesión

Se establece cuando hay una interacción entre un cliente y un servidor. Estas pueden permanecer en función del tiempo de existencia de una conexión de red o varias si están consecutivamente establecidas entre el cliente y el servidor.

4.1.1.9 Paquete de control MQTT

Paquete de información que se envía a través de la conexión de red. La especificación MQTT define catorce tipos diferentes de estos paquetes.

4.1.2 Estructura de un Paquete de Control MQTT

El protocolo MQTT funciona mediante el intercambio de una serie de paquetes de control. Estos paquetes constan de hasta tres partes, siempre en el siguiente orden:

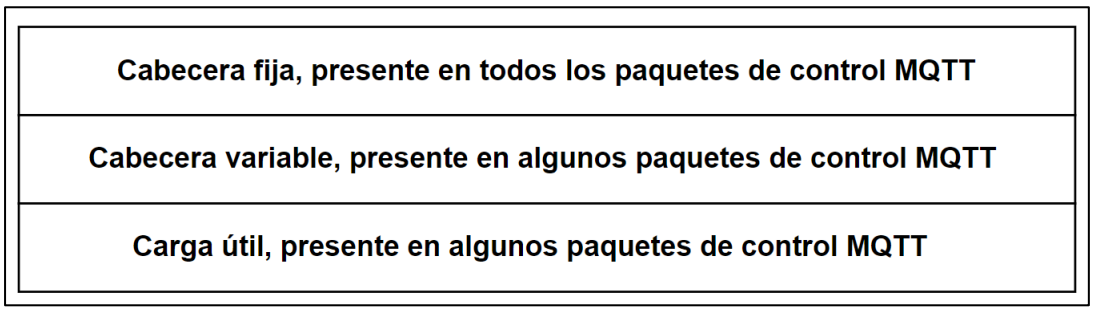

Figura 4-1: Estructura paquete de control MQTT

4.1.2.1 Cabecera fija

Cada paquete de control MQTT contiene una cabecera fija. Su formato es:

Bit	7	6	5	4	3	2	1	0
byte 1	Tipo paquete de control MQTT				Flags específicas de cada tipo de paquete de control MQTT			
byte 2...	Longitud restante							

Figura 4-2: Cabecera fija

4.1.2.1.1 Tipos de Paquetes de Control MQTT

La identificación del tipo del paquete se encuentra en el primer byte, ocupando los bits 7-4. Se representa como un valor sin signo de 4 bits. Estos tipos son:

Tabla 4-1: Tipos de paquetes de control MQTT

Nombre	Valor	Sentido del flujo	Descripción
Reserved	0	Prohibido	Reservado
CONNECT	1	C → S	Petición del cliente para conectarse con el servidor.
CONNACK	2	S → C	Acuse de recibo de conexión.
PUBLISH	3	C → S, S → C	Publicar mensaje.
PUBACK	4	C → S, S → C	Acuse de recibo de publicación.
PUBREC	5	C → S, S → C	Publish recibido (parte 1).
PUBREL	6	C → S, S → C	Publish recibido (parte 2).
PUBCOMP	7	C → S, S → C	Publish recibido (parte 3).
SUBSCRIBE	8	C → S	Petición de suscripción del cliente.
SUBACK	9	S → C	Acuse de recibo de suscripción.
UNSUBSCRIBE	10	C → S	Petición de desinscripción.
UNSUBACK	11	S → C	Acuse de recibo de desinscripción.
PINGREQ	12	C → S	Petición PING.
PINGRESP	13	S → C	Respuesta PING.
DISCONNECT	14	C → S	Cliente desconectándose.
Reserved	15	Prohibido	Reservado

Siendo C → S de Cliente a Servidor y C → S de Servidor a Cliente. En el apartado 4.1.2.4 se describen con mayor profundidad estos tipos de mensajes.

4.1.2.1.2 Flags

Los bits restantes [3-0] del primer byte en la cabecera fija contienen banderas específicas para cada tipo de paquete de control MQTT.

Tabla 4-2: Flags de los paquetes de control MQTT

Paquete de control	Flags de la cabecera fija	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	Reserved	0	0	0	0
CONNACK	Reserved	0	0	0	0
PUBLISH	Usado en MQTT 3.1.1	DUP	QoS	QoS	RETAIN
PUBACK	Reserved	0	0	0	0
PUBREC	Reserved	0	0	0	0
PUBREL	Reserved	0	0	1	0
PUBCOMP	Reserved	0	0	0	0
SUBSCRIBE	Reserved	0	0	1	0
SUBACK	Reserved	0	0	0	0
UNSUBSCRIBE	Reserved	0	0	1	0
UNSUBACK	Reserved	0	0	0	0
PINGREQ	Reserved	0	0	0	0
PINGRESP	Reserved	0	0	0	0
DISCONNECT	Reserved	0	0	0	0

- DUP → Entrega duplicada de un paquete de control PUBLISH.
- RETAIN → Bandera de retención del paquete de control PUBLISH.
- QoS → Calidad de servicio del paquete de control PUBLISH. Existen tres niveles de calidad de servicio que permiten elegir entre minimizar la transmisión de datos y maximizar la fiabilidad:

- QoS 0: El mensaje se entrega según las capacidades de la red subyacente. El receptor no envía ninguna respuesta y el emisor no realiza ningún reintento, por lo que el mensaje llega al receptor una vez o no llega. Conocida como “Una entrega como máximo”.

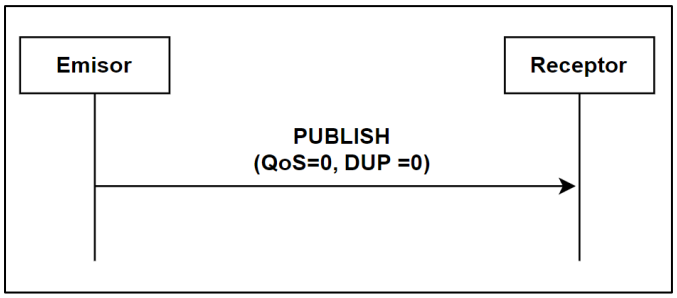

Figura 4-3: Esquema QoS=0

- QoS 1: Esta calidad de servicio garantiza que el mensaje llegue al receptor al menos una vez. Un paquete con dicho QoS tiene un Identificador de paquete en su cabecera variable y es reconocido por un paquete PUBACK. Conocida como “Entregado al menos una vez”.

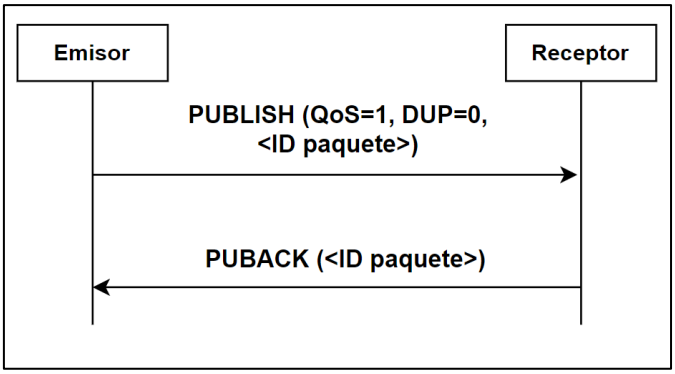

Figura 4-4: Estructura QoS=1

- QoS 2: Se trata de la calidad de servicio más alta, para cuando no son aceptables ni la pérdida ni la duplicación de mensajes. Conocida como “Entregado exactamente una vez”

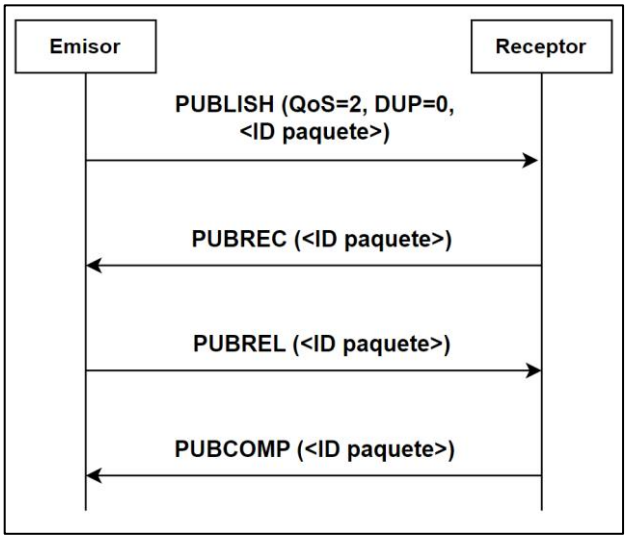

Figura 4-5: Esquema QoS=2

4.1.2.1.3 Longitud restante

La longitud restante es el número de bytes que quedan en el paquete, incluidos los datos de la cabecera variable más los de la carga útil. No se tiene en cuenta los bytes utilizados para codificarla.

4.1.2.2 Cabecera variable

Como se comentó anteriormente, algunos tipos de paquetes de control MQTT contienen esta cabecera con información adicional, ubicada entre la cabecera fija y la carga útil. Su contenido varía en función del tipo de paquete, aunque el campo ID del paquete puede ser común entre varios de ellos. Este ID se asigna de forma independiente entre cliente y servidor.

Los paquetes que tienen este campo son: PUBLISH (si QoS es > 0), PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE y UNSUBACK.

En los casos en los que el paquete sea de tipo SUBSCRIBE, UNSUBSCRIBE y PUBLISH (con QoS > 0) los paquetes de control deben tener un ID distinto de 0. Cada vez que el cliente envíe uno de estos tipos de mensajes, deberá asignarle un ID nuevo, y en caso de reenvío, se deberá utilizar el mismo identificador en todos los reenvíos posteriores.

Los paquetes PUBACK, PUBREC y PUBREL tienen el mismo ID de paquete que el del paquete PUBLISH enviado. De igual forma, los paquetes SUBACK y UNSUBACK deben tener el mismo que los paquetes SUSBCRIBE y UNSUBSCRIBE enviados respectivamente.

4.1.2.3 Carga útil

Algunos paquetes de control MQTT contienen una carga útil como parte final de su estructura en el que se transporta el contenido real del mensaje. En el caso del paquete PUBLISH se corresponde con el mensaje de aplicación.

Tabla 4-3: Carga útil de los paquetes de control MQTT

Paquete de control	Carga útil
CONNECT	Requerido
CONNACK	Ninguno
PUBLISH	Opcional
PUBACK	Ninguno
PUBREC	Ninguno
PUBREL	Ninguno
PUBCOMP	Ninguno
SUBSCRIBE	Requerido
SUBACK	Requerido
UNSUBSCRIBE	Requerido
UNSUBACK	Ninguno
PINGREQ	Ninguno
PINGRESP	Ninguno
DISCONNECT	Ninguno

4.1.3 Paquetes de control

A pesar de los catorce tipos diferentes de paquetes que existen, solo cuatro de ellos son necesarios para la conexión/comunicación/desconexión entre cliente y servidor (CONNECT, PUBLISH, SUBSCRIBE y UNSUBSCRIBE). El resto son utilizados principalmente para mecanismos internos y gestión del flujo de mensajes.

4.1.3.1.1 CONNECT

Es el primer mensaje enviado desde el cliente al servidor una vez establecida la conexión de red. El cliente puede enviar un solo paquete de este tipo y si el servidor recibe más de uno, considerará que ha ocurrido una violación del protocolo y procederá a la desconexión del cliente.

La carga útil contiene uno o varios campos codificados que especifican un ID del cliente, un topic, un mensaje, que define el mensaje de aplicación que se publicará en el topic, un nombre y una contraseña. Todos estos campos, a excepción del ID del cliente son opcionales y se determinan en función de los

flags presentes en la cabecera variable.

4.1.3.1.2 CONNACK

Es el primer paquete enviado por el servidor en respuesta a un paquete CONNECT recibido por un cliente.

Si el cliente no recibe este paquete de respuesta en un tiempo determinado, que dependerá del tipo de aplicación y la infraestructura de comunicación, cerrará la conexión de red.

4.1.3.1.3 PUBLISH

El paquete de control PUBLISH se envía de un cliente a un servidor o de un servidor a un cliente para transportar un mensaje de aplicación.

4.1.3.1.4 PUBACK

El paquete PUBACK se envía como respuesta a un paquete PUBLISH con QoS de nivel 1.

4.1.3.1.5 PUBREC

El paquete PUBREC se envía como respuesta a un paquete PUBLISH con QoS nivel 2. Es el segundo paquete intercambiado cuando QoS = 2.

4.1.3.1.6 PUBREL

El paquete PUBREL se envía como respuesta a un paquete PUBREC. Es el tercer paquete intercambiado cuando QoS = 2.

4.1.3.1.7 PUBCOMP

El paquete PUBCOMP se envía como respuesta a un paquete PUBREL. Es el cuarto y último paquete intercambiado cuando QoS = 2.

4.1.3.1.8 SUBSCRIBE

Este paquete se envía desde el cliente al servidor para crear una o más suscripciones que registran el interés que se tiene por uno o varios topics. El servidor envía paquetes PUBLISH al cliente para reenviar mensajes de aplicación publicados en topics coincidentes. El paquete SUBSCRIBE especifica también la calidad de servicio máxima con la que el servidor puede enviar mensajes de aplicación.

4.1.3.1.9 SUBACK

El servidor envía un paquete SUBACK al cliente para confirmar la recepción y el procesamiento de un paquete SUBSCRIBE.

4.1.3.1.10 UNSUBSCRIBE

El paquete UNSUBSCRIBE es enviado por el cliente al servidor para desinscribirse de algún topic.

4.1.3.1.11 UNSUBACK

El paquete UNSUBACK es enviado por el servidor al cliente para confirmar la recepción del paquete UNSUBSCRIBE.

4.1.3.1.12 PINGREQ

Enviado desde un cliente a un servidor, este se puede utilizar para indicar al servidor que el cliente está activo en ausencia de otros paquetes de control, solicitar que el servidor responda para confirmar que está activo y para asegurar que la conexión de red esté activada.

4.1.3.1.13 PINGRESP

El servidor envía un paquete PINGRESP al cliente en respuesta a un paquete PINGREQ. Indica que el servidor está activo.

4.1.3.1.14 DISCONNECT

El paquete DISCONNECT es el último paquete de control enviado desde el cliente al servidor para indicar que se desconecta completamente.

4.1.4 Arquitectura MQTT del proyecto

En el siguiente diagrama se representa los componentes MQTT utilizados en el proyecto:

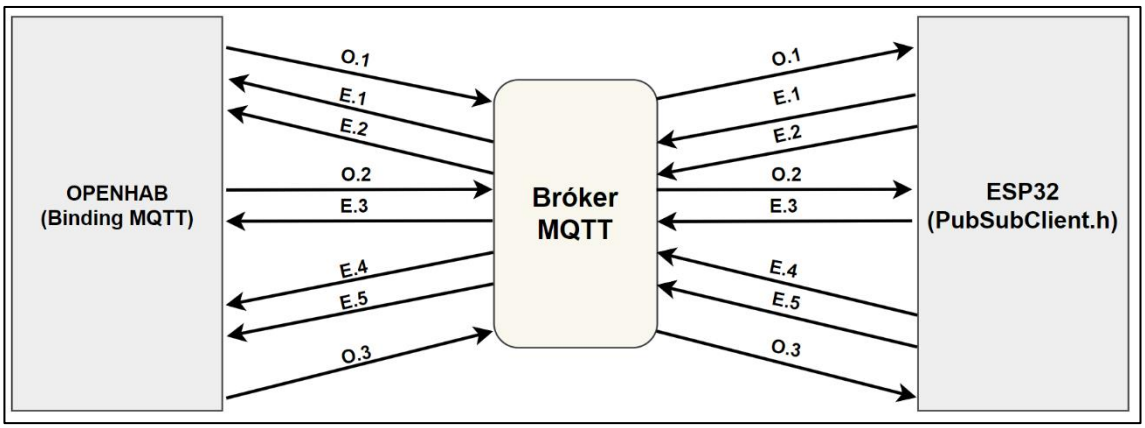

Figura 4-6: Arquitectura MQTT del proyecto

La arquitectura MQTT empleada en el trabajo consta de un servidor y dos clientes que envían o reciben datos al/del bróker en función del tipo de información a través de los topics. Estos clientes son: OpenHAB, mediante la instalación del binding MQTT, explicado en el apartado 3.3.1.5.1 y la placa

ESP32 a través de una librería específica llamada 'PubSubClient.h' que se detallará posteriormente en el capítulo 4.2.3. El bróker utilizado para establecer la conexión entre ambos es el proporcionado por EMQX Cloud [30].

La calidad de servicio (QoS) empleada en la comunicación es de nivel 0 de manera que, tal y como se comentó en el apartado 4.1.2.1.2, el receptor no envía ninguna respuesta al paquete PUBLISH generado por el emisor. Así se consigue una menor latencia, algo muy beneficioso al querer obtener los datos del invernadero en el menor tiempo posible, y un menor consumo de energía.

En la tabla 4-4 se muestran todos los topics que se han configurado, diseñado de forma que por cada uno de ellos se envíen los datos relativos a cada sensor y actuador utilizado en el proyecto. Además, se indica también que rol tienen los clientes en cada uno de ellos, especificando como publicador a aquel que envía los datos y como subscriptor a aquel que los recibe.

Tabla 4-4: Topics MQTT

Cliente OpenHAB			**Cliente ESP32**		
ID	**Topic**	**Tipo**	**ID**	**Topic**	**Tipo**
O.1	Entrada0/1	Publicador	O.1	Entrada0/1	Subscriptor
E.1	Sensor/Luminosidad	Subscriptor	E.1	Sensor/Luminosidad	Publicador
E.2	Sensor/HumedadTerrestre	Subscriptor	E.2	Sensor/HumedadTerrestre	Publicador
O.2	AireAcondicionado	Publicador	O.2	AireAcondicionado	Subscriptor
E.3	Sensor/NivelAgua	Subscriptor	E.3	Sensor/NivelAgua	Publicador
E.4	Sensor/DHT11/Temperatura	Subscriptor	E.4	Sensor/DHT11/Temperatura	Publicador
E.5	Sensor/DHT11/Humedad	Subscriptor	E.5	Sensor/DHT11/Humedad	Publicador
O.3	Humidifcador	Publicador	O.3	Humidificador	Subscriptor

Toda la información que es enviada por el cliente OpenHAB está relacionada con el encendido/apagado de los actuadores, utilizando los enteros '0' (para el apagado) o '1' (para el encendido). Por ejemplo, en el caso de querer encender el motor de agua, OpenHAB enviaría un '1' a través del topic 'Entrada0/1'. Posteriormente, el cliente ESP32, al estar suscrito a este topic, recibirá el mensaje procedente del bróker público y, al comprobar su contenido, enviará la señal de encendido al motor.

Por otro lado, la información enviada por el cliente ESP32 está formada únicamente por los datos recogidos por los sensores instalados. En este caso, por ejemplo, si se obtuviera un nuevo valor de la temperatura, este sería enviado a través del topic 'Sensor/DHT11/Temperatura' por la placa ESP32 al bróker público para ser reenviado posteriormente a OpenHAB.

4.2 Arduino

Arduino es una compañía de desarrollo software y hardware libres, así como una comunidad internacional que diseña y fabrica placas de desarrollo hardware para construir dispositivos digitales y otros que pueden interactuar con los objetos del mundo real. Los productos que vende la compañía son distribuidos como hardware y software libre, bajo la Licencia Pública General de GNU (GPL) y la Licencia Pública General Reducida de GNU (LGPL), permitiendo la manufactura de las placas Arduino y distribución del software por cualquier individuo.

4.2.1 Placa ESP32

El ESP32 Dev Kit C V2 es una placa de desarrollo en torno al chip ESP32 WROOM 32. Incorpora un firmware preinstalado que permite trabajar con lenguaje interpretado y enviar comandos a través del puerto serie (chip CP2102/CH340). Dispone de un regulador de tensión que permite alimentarlo directamente desde el puerto USB.

Entre sus principales características destacan:

- Soporta una gran variedad de protocolos como SPI, I2C, I2S, UART, entre otros.
- Es un sistema en chip (SoC) que integra un microcontrolador Tensilica de 32 bits.
- Cuenta con WiFi de 2.4 GHz (802.11 b/g/n).
- Permite comunicaciones inalámbricas BLE y Bluetooth clásicas.
- Cuenta con un conversor analógico-digital (ADC), un conversor digital-analógico (DAC) y un PWM.

4.2.1.1 Pinout

El ESP32 tiene 38 pines, y cada una de sus funciones se muestran en la siguiente imagen:

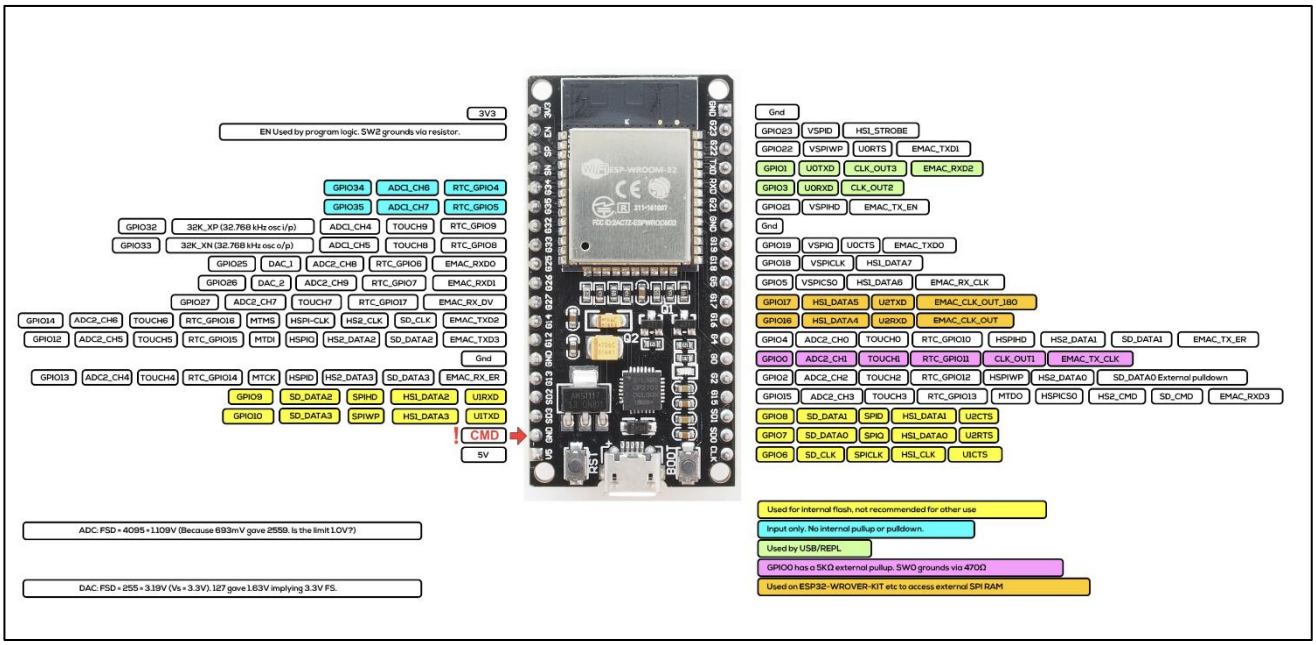

Figura 4-7: Pinout ESP32

Al igual que una placa Arduino normal, el ESP32 Dev Kit C V2 tiene pines digitales de entrada/salida (pines GPIO). Estas entradas/salidas funcionan a 3.3V, por lo que aplicarle una tensión superior a esta (5V) podría destruir el chip.

Dispone a su vez de un conector microUSB que consiste en el chip CP2102 de Silicon Laboratories responsable de la comunicación en serie USB a UART. El chip dispone de la función VCP (Virtual COM Port), que aparece como puerto COM en las aplicaciones de PC (Administrador de dispositivos en Windows).

Para utilizar la placa, es necesario instalar el controlador mencionado en el capítulo 2: Tecnologías Requeridas.

4.2.2 Conexión física de los dispositivos con la placa ESP32

La conexión física establecida entre todos los sensores/actuadores y la placa ESP32 se visualiza en la siguiente imagen:

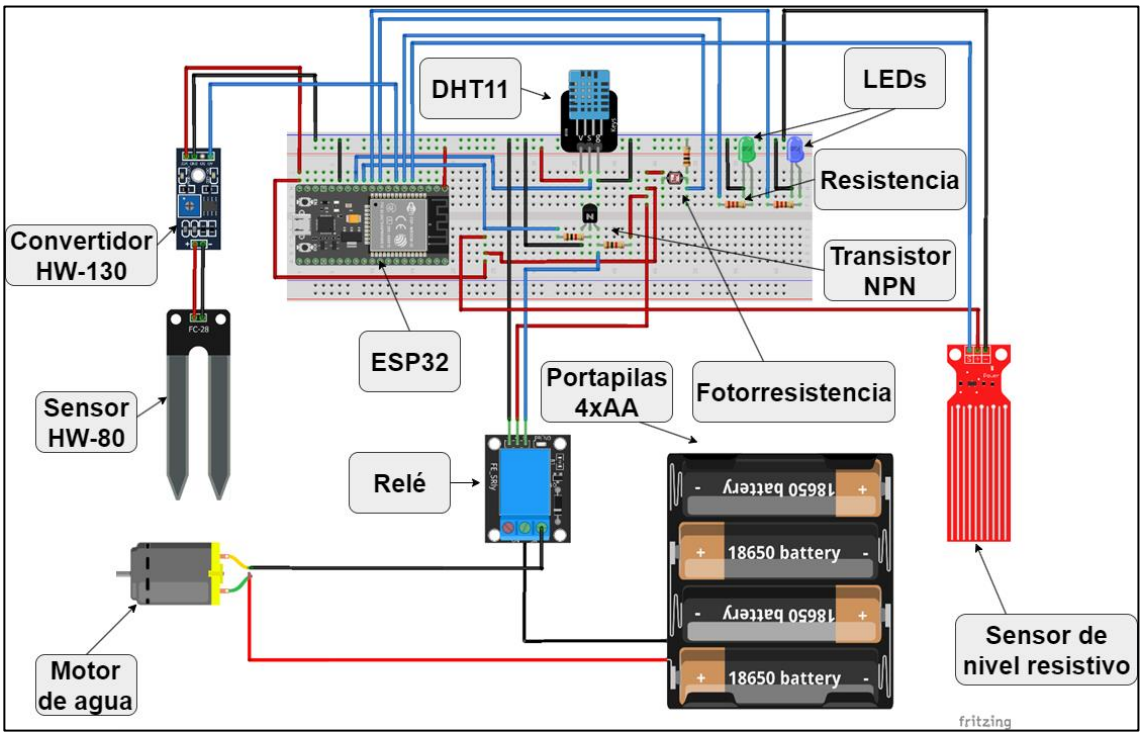

Figura 4-8: Conexión física entre componentes

Los pines utilizados de la placa ESP32 se detallan en la siguiente tabla:

Tabla 4-5: Pines utilizados en el sistema

Pin ESP32	Descripción
PIN V5	Proporciona una tensión de 5V.
PIN GND	Proporciona la toma de tierra del circuito.
GPIO 14	Conectado al sensor DHT11 para recibir los datos de temperatura y humedad ambiental.
GPIO 27	Proporcionar la tensión de 3.3V necesaria en la activación del relé.
GPIO 26	Proporciona la tensión 3.3V necesaria para activar el LED azul.
GPIO 25	Proporciona la tensión 3.3V necesaria para activar el LED verde.

GPIO 32	Conectado al módulo HW-103 para recibir los datos de humedad terrestre medidos por el sensor HW-80.
GPIO 35	Conectado a la fotorresistencia para recibir los datos de luminosidad.
GPIO 34	Conectado al sensor de nivel agua resistivo para recibir datos sobre la cantidad de agua disponible.
PIN 3V3	Proporciona una tensión de 3.3V.

4.2.3 Código Arduino

El código Arduino que he desarrollado está escrito en la placa ESP32, que es quien gestiona tanto lo comunicación con OpenHAB a través de MQTT así como la recogida de datos de los sensores.

El código se puede estructurar en tres bloques bien diferenciados:

- **Bloque de inicialización**: en este bloque se declaran y se inicializan todas las variables necesarias, así como la importación de cada una de las librerías correspondientes requeridas para el funcionamiento del código. Entre ellas, se encuentran:
  - Librería <WiFi.h>: Facilita la comunicación inalámbrica utilizando WiFi, lo que permite conectarse a Internet, interactuar con otros dispositivos y acceder a diferentes servicios y recursos en la red.
  - Librería <PubSubClient.h>: Permite la comunicación a través del protocolo MQTT, facilitando a los dispositivos Arduino actuar como clientes MQTT y establecer una conexión con un bróker para enviar o recibir mensajes.
  - Librería <DHT.h>: Permite crear una instancia con la que poder acceder a la funcionalidad del sensor de temperatura y humedad DHT11.

  En este bloque también se define una función 'callback()' utilizada para que, cuando se reciba un paquete a través del protocolo MQTT, éste sea atendido, extrayendo su mensaje.

- **Bloque de establecimiento**: en este bloque, correspondiente a la función 'setup()' de Arduino, se configura la entrada/salida de los pines, su valor inicial, se inicia el sensor DHT11 y se establece la conexión Wi-Fi con el punto de acceso especificado.

```
dht.begin();
WiFi.begin(ssid, contrasena);
```

Figura 4-9: Inicio del sensor DHT11 y conexión WiFi

Además, en este bloque se establece la conexión del cliente MQTT configurado con el bróker, se asocia con la función callback() explicada anteriormente y se subscribe a los topics

de los que se quieren recibir mensajes.

```
client.setServer(servidorMQTT, puertoMQTT);
client.setCallback(callback);
```

Figura 4-10: Conexión servidor MQTT y asociación con la función callback()

```
client.subscribe("Entrada/01");  //Suscripción al topic Entrada/01
client.subscribe("AireAcondicionado"); //Suscripción al topic AireAcondicionado
client.subscribe("Humidificador"); //Suscripción al topic Humidificador
```

Figura 4-11: Subscripción a los topics

- **Bloque de operación**: este bloque, que se corresponde con la función loop() de Arduino, es el encargado de recoger periódicamente toda la información que envían los sensores a la placa ESP32 y este de enviarlo a los topics del que es el publicador. La recogida de datos está programada de forma que se obtengan de cada sensor en un periodo de tiempo distinto.

  Para conseguirlo, inicialmente se pensó en que la obtención de estos datos se podría hacer en diferentes hilos, de forma que cada uno se encargara de un sensor distinto y posteriormente enviar la información al bróker MQTT. Sin embargo, tal y como se comentó al principio, la placa ESP32 posee un microcontrolador, el cual posee un único núcleo. Esto implica que solo es capaz de ejecutar una instrucción cada cierto tiempo, por lo que la multitarea, es decir, la ejecución en varios hilos a la vez, no es posible. Tampoco sería posible utilizar la función delay(), ya que ésta retrasaría la ejecución de todo el código.

  La solución adoptada fue el uso de la función de temporización millis() que cuenta el tiempo y ejecuta una operación cuando el temporizador se dispare. De esta forma se consigue realizar varias tareas a la vez. A modo de ejemplo, se explicará el funcionamiento de la recogida de los datos para el sensor de luminosidad, siendo igual para el resto.

  Cuando se inicia la ejecución, se crea la variable '*tiempoInicioLuz*' que almacena el tiempo que lleva en funcionamiento el programa mediante la llamada función millis(). En el momento en el que comienza la ejecución del bloque de operación, se inicia la variable '*tiempoTrancurridoLuz*' utilizando la misma función. Posteriormente, se comprueba si el tiempo transcurrido ejecutándose el código es superior o no al tiempo desde su inicio más un tiempo concreto (2 segundos), es decir, se verifica si el programa se ha estado ejecutando durante un periodo de tiempo. Cuando este periodo es superado, se procede a:

  - Inicializar de nuevo la variable '*tiempoInicioLuz*' para que almacene el nuevo tiempo de ejecución del programa.

- Leer el pin correspondiente, convertir el valor obtenido al tipo de dato char y enviarlo al bróker a través del topic adecuado, utilizando la función 'publish(const char *topic, const char *info)'.

```
if (tiempoTranscurridoLuz > (tiempoInicioLuz + 2000)) {

  Serial.println("Se ha llegado a los 2 segundos");
  tiempoInicioLuz = millis();
  char Buf[300];
  int valorLuz = analogRead(adc_idLuz);
  Serial.print("Nivel de luminosidad: ");
  Serial.println(valorLuz);

  luzStr.concat(valorLuz);
  luzStr.toCharArray(Buf, 300);

  if (client.publish("Sensor/Luminosidad", Buf) == true) {

    Serial.println("Nivel de luminosidad enviada al broker");
  } else {

    Serial.println("No se ha podido enviar el nivel de luminosidad");
  }

}
```

Figura 4-12: Configuración fotorresistencia

El periodo de recogida y envío de los datos obtenidos de cada sensor se muestra en la siguiente tabla:

Tabla 4-6: Periodos de muestreo

Sensor	Periodo (min)
Luminosidad	30
Humedad Terrestre	10
Nivel Agua	10
Humedad y Temp. Ambiental (DHT11)	30

Los pasos realizados para la instalación de Arduino, así como de las librerías necesarias para poder configurar la placa ESP32 a través del IDE se explican en el Anexo C.

5 DESCRIPCIÓN DEL ALMACENAMIENTO DE DATOS Y APLICACIÓN ANDROID

"La conectividad es un derecho humano"

Mark Zuckerberg

En este capítulo se explicará el desarrollo y la configuración de la base de datos Postgresql utilizada para el almacenamiento de la información, abordando posteriormente la programación del servidor REST, elemento de gran importancia para el funcionamiento del proyecto. Por último, se hará una descripción profunda sobre la aplicación móvil empleada por el usuario final para la gestión de todo el invernadero.

5.1 Base de datos PostgreSQL

PostgreSQL es un sistema de base de datos relacional de objetos de código abierto con más de treinta y cinco años de desarrollo. Destaca por su capacidad de adaptación y cumplimiento con el estándar SQL, así como la gestión de grandes volúmenes de datos. Entre sus funciones principales destacan:

- Almacenamiento eficiente de datos: organiza y almacena datos garantizando su integridad y consistencia.
- Consultas avanzadas: permite realizar consultas complejas y analizar datos para obtener información valiosa con lenguaje SQL.
- Backend confiable para aplicaciones: empleado principalmente en el desarrollo de aplicaciones web y móviles, pues ofrece un almacenamiento seguro y escalable para los datos de la aplicación.

La base de datos se ha empleado en el proyecto para el almacenamiento de información de interés. Para ello, se creó una base de datos, llamada 'openhabdb', localmente en la máquina virtual ejecutándose en el puerto 5432 con usuario y contraseña 'openhab'.

Las tablas creadas y su sintaxis se muestran a continuación:

- item0001, item0002, item0010, item0011, item0012: estas tablas almacenan los datos procedentes de los sensores de temperatura, humedad ambiental, luminosidad, nivel de agua y humedad terrestre respectivamente. La sintaxis es similar para todas, con una columna "time" para almacenar la fecha en la que se recibe el dato y la columna "value" que almacena el valor:

```
                    Table "public.item0001"
 Column |            Type             | Collation | Nullable | Default
--------+-----------------------------+-----------+----------+---------
 time   | timestamp without time zone |           | not null |
 value  | character(50)               |           |          |
Indexes:
    "item0001_pkey" PRIMARY KEY, btree ("time")
```

Figura 4-1: Tabla item0001

Como se puede observar, la clave primaria está establecida en la columna "time".

- Limites: esta tabla almacena todos los limites ambientales establecidos en el invernadero por el usuario y que son imprescindibles para la configuración de las reglas en OpenHAB. En ella, los datos relacionados con la temperatura, humedad ambiental y terrestre se almacenan en doble precisión, mientras que los que provienen del sensor de luminosidad se almacenan como enteros.

```
                      Table "public.limites"
        Column          |       Type       | Collation | Nullable | Default
------------------------+------------------+-----------+----------+---------
 limite_temperatura     | double precision |           | not null |
 limite_humedad         | double precision |           |          |
 limite_luz             | integer          |           |          |
 limite_humedadterrestre | double precision |           |          |
Indexes:
    "limites_pkey" PRIMARY KEY, btree (limite_temperatura)
```

Figura 5-2: Tabla limites

La clave primaria de esta tabla está establecida en la columna "limite_temperatura", aunque puede ser cualquier otra.

- Usuarios: en esta tabla se almacenan las credenciales de los usuarios que tienen acceso a la aplicación del invernadero. Cada columna, de tipo text, almacenará el nombre del usuario y su contraseña respectivamente.

```
             Table "public.usuarios"
   Column   | Type | Collation | Nullable | Default
------------+------+-----------+----------+---------
 usuario    | text |           | not null |
 contraseña | text |           |          |
Indexes:
    "usuarios_pkey" PRIMARY KEY, btree (usuario)
```

Figura 5-3: Tabla usuarios

La clave primaria de la tabla se corresponde con la columna usuario.

- Motoragua: esta tabla es utilizada para almacenar el tiempo de inicio de funcionamiento del motor y la cantidad de agua consumida en las columnas “time” y “value” respectivamente. Se explicará su uso con más profundidad en el siguiente apartado 5.2: Servidor REST.

La configuración de la base de datos se explica en el Anexo D.

5.2 Servidor REST

Para poder atender a todas las peticiones que el cliente realice a través de la aplicación móvil y para gestionar la base de datos se ha configurado un servidor REST. Este servidor está implementado mediante el framework de Spring y permite ser la puerta de entrada a la gestión y monitorización de todo el invernadero.

El servidor REST tiene como posibles clientes tanto el usuario final, a través de la aplicación móvil, como al servidor OpenHAB y se ejecuta localmente en la máquina virtual con dirección IP 192.168.1.62 y en el puerto 9000. Además, es el responsable de establecer la conexión con la base de datos PostgreSQL, de modo que cualquier operación de inserción, eliminación o modificación de alguna de sus tablas es realizada por él.

Junto con el directorio donde se encuentra el archivo que contiene la función ‘main()’ para la ejecución del servidor, existen otros en los que se codifica la funcionalidad del mismo. Estos son:

- controller: en esta carpeta se encuentra el archivo ‘*FileController*’, donde se definen las URLs de las operaciones que podrá realizar el servidor. Además, se encuentran otras clases utilizadas por este archivo las cuales, como se explicará posteriormente, permiten la definición de variables dentro de una petición.
- service: se encuentran la interfaz ‘*apiRest_ServidorAPI’*, donde se definen los métodos de las operaciones que puede atender el servidor y la clase ‘*apiRest_ServidorImpl’*, donde se implementan estos métodos. Además, aquí se ubica la clase ‘*ConexionBBDD’,* que es la encargada de establecer la conexión con la base de datos mediante la carga del driver correspondiente, especificando la URL donde se aloja y añadiendo el usuario y la contraseña para el acceso a sus tablas.

A continuación, se explicarán todos los tipos de métodos empleados para dar respuesta a las distintas peticiones provenientes de los clientes:

- **Método “subir”:** Este método es llamado por la operación “/audio” de tipo POST y es el encargado de transformar el audio que el cliente envía en formato 3gp a través de la app utilizando la herramienta ffmpeg de línea de comandos y obtener un wav. Posteriormente llama a voice2json para que realice la transcripción. Este método no devuelve nada.

El fichero de audio se envía desde la aplicación móvil, con un determinado nombre, a una ubicación específica del servidor. La herramienta ffmpeg es un convertidor de medios universal, pudiendo leer una amplia variedad de entradas, tales como de video, audio, archivos adjuntos, etc, para filtrarlas y transcodificarlas en diferentes formatos a la salida.

De forma resumida, el funcionamiento de esta herramienta se muestra en la siguiente figura:

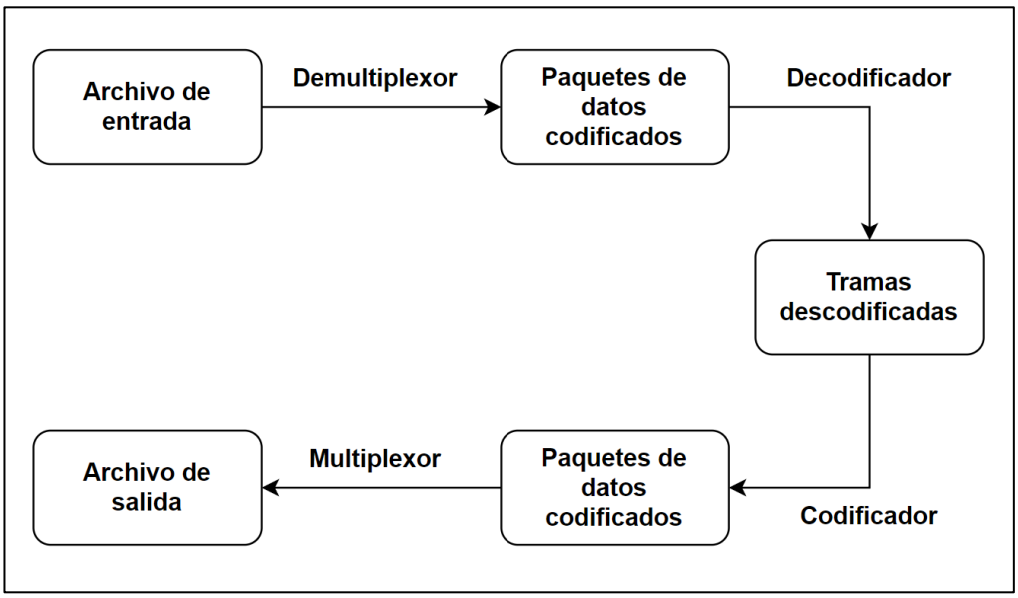

Figura 5-4: Esquema ffmpeg

Ffmpeg llama a una biblioteca incorporada llamada libavformat (que contiene los multiplexores y demultiplexores) para leer los archivos de entrada y obtener de ellos paquetes que contienen los datos codificados. Estos son pasados por el decodificador produciendo frames sin comprimir (videos en raw, audio PCM, etc). Posteriormente, los frames son pasados por el codificador para convertirlos en paquetes codificados. Por último, se pasan al multiplexor para escribir estos paquetes en el archivo de salida.

Para transformar el audio que se recibe de la aplicación móvil, en formato 3gpp, a un formato wav, que es el reconocido por Voice2JSON, se utilizan dos argumentos:

- **-i:** Indica la ubicación del archivo de audio que se quiere convertir.
- **-acodec**: utilizado para especificar el códec de audio, permitiendo codificar y decodificar los datos auditivos en un formato determinado. Entre todos los códecs disponibles de esta herramienta, el utilizado para la conversión es el PCM signed 16-bit Little-endian (*pcm_s16le*). PCM es un método de codificación sin compresión que permite obtener el formato de audio wav sin perdidas.

Esta herramienta es ejecutada desde código Java mediante el uso de la librería 'Runtime', que permite a la aplicación interactuar con el entorno en el que se encuentra.

```
File file3gp = new File("/home/salas/SDSW/Servidor/apiRest_Servidor/Audios/grabacion.3gp");
String fpstr = file3gp.getAbsolutePath();
String outname = "/home/salas/SDSW/Servidor/apiRest_Servidor/Audios/Grabacion.wav";
String exeQuery = "ffmpeg -i " + fpstr + " -acodec pcm_s16le " + outname;
System.out.println(Runtime.getRuntime().exec(exeQuery));
```

Figura 5-5: Configuración de ffmpeg

Una vez convertido el fichero de audio a formato wav, se procede a realizar la transcripción utilizando

Voice2JSON. Para ello, se utiliza la clase 'PocessBuilder', que permite ejecutar nuevos procesos del sistema operativo desde el código Java. El resultado de esta operación se imprime en un archivo de texto que posteriormente será el mensaje enviado a través de Kafka, tal y como se explicó en el apartado 3.2.5.

```
File ficheroTexto = new File("/home/salas/Escritorio/Pruebas/fichero.txt");

ProcessBuilder pb = new ProcessBuilder("voice2json", "-p", "es_kaldi-rhasspy", "transcribe-wav", ficheroWav);
pb.redirectOutput(ProcessBuilder.Redirect.to(ficheroTexto));
try {
    Process proceso = pb.start();

} catch (Exception e) {
    e.printStackTrace();
}
```

Figura 5-6: Configuración para realizar la transcripción

- **Método "consulta"**: Este método es llamado por la operación "/consutla" de tipo GET y es utilizado para obtener los datos almacenados en la base de datos desde una fecha determinada para su representación en la aplicación móvil en forma de gráfico. Devuelve una lista de tipo String.

La consulta a la base de datos para obtener una serie de datos almacenados a partir de una fecha se realiza mediante la consulta: "SELECT * FROM 'nombre_tabla' WHERE time > 'fecha_consulta'". Tanto 'nombre_tabla' como 'fecha_consulta' son dos variables que se envían desde la aplicación móvil junto con la URL, ya que la operación es de tipo GET.

El parámetro que viaja en la URL identificando la fecha de la consulta puede tomar únicamente tres valores:

- A partir de 1 día
- A partir de 3 días
- A partir de 7 días

De esta forma se le permite al usuario la posibilidad de comprobar la evolución del invernadero en distintos tramos temporales.

Como se explicó en el apartado de la base de datos PostgreSQL, las tablas almacenan también las fechas en las que se reciben los datos de los sensores. Estas, tienen el formato "YY/MM/DD HH/MM/SS". Para obtener una variable en este formato y utilizarla en la consulta anterior, se utiliza la clase Calendar de Java. Esta clase permite calcular de forma automática la fecha de hace uno, tres o siete días respecto a la fecha actual, en la que se ejecuta el código.

En el código que se muestra a continuación, se calcula la fecha según lo solicitado por el usuario, restando a la fecha actual un día (para el resto de las opciones solo se tendría que cambiar este número):

```
calendario.add(Calendar.DATE, -1);
String diaUnDia = Integer.toString(calendario.get(Calendar.DATE));
String mesUnDia = Integer.toString(calendario.get(Calendar.MONTH) + 1);
String anoUnDia = Integer.toString(calendario.get(Calendar.YEAR));

fechaConsulta = anoUnDia + "-" + mesUnDia + "-" + diaUnDia;
```

Figura 5-7: Configuración para obtener las fechas

Posteriormente, a partir de la fecha resultante, se obtiene el día, el mes y el año almacenándolas en variables para posteriormente, obtener otra variable llamada "*f'echaConsulta*' juntando las anteriores y que será utilizada en la consulta.

Finalmente se realiza la petición a la base de datos, se obtienen los datos y se almacenan en una lista que será enviada al cliente.

Para que el servidor no envíe todos los datos almacenados en la base de datos, se realiza una serie de operaciones para que el resultado sea una lista con nueve datos y nueve fechas. Lo que se busca es que cada punto que se vaya a mostrar en la gráfica de la aplicación Android se corresponda con la media de un conjunto de datos almacenados en una lista. De tal forma, los datos recibidos se organizan en nueve arrays.

Dependiendo desde cuando se soliciten los datos, las operaciones realizadas serán distintas:

- **A partir de 1 día**

En una lista se almacenan todos los datos almacenados desde este tiempo en la base de datos, y en otra, se almacenan las fechas de cuando fueron guardados ejecutando la sentencia SQL mostrada anteriormente. Una vez terminada la consulta, se cuenta el número total de datos recibidos y se divide entre nueve (ya que nueve son los puntos que se mostrarán en las gráficas de la aplicación). De esta forma, el cociente de esta división, indica el número de elementos que deben tener los nueve arrays y, con ello, realizar la media de cada uno de ellos para que sean representadas en las gráficas.

- **A partir de 3 días**

En este caso, se obtienen los datos almacenados correspondientes para uno de los tres días y cada conjunto se divide entre tres, de forma que, se obtendrán tres arrays, y tres medias correspondientes a cada día.

- **A partir de 7 días**

Con esta opción, se obtienen dos conjuntos de datos para el sexto y séptimo día (obteniéndose dos medias) y para el resto de días, se obtiene un único conjunto, de forma que se consiguen las cinco medias restantes.

- Método "iniciarSesion": Método llamado por la operación "/sesion" de tipo GET, utilizado para comprobar si el cliente que intenta acceder a la aplicación con el usuario y contraseña introducidos son válidos. Devuelve un booleano, indicando si está permitido o no su acceso.

Para ello, se realiza la consulta siguiente a la tabla usuarios donde se almacenan los usuarios y contraseñas de cada uno de los clientes autorizados para acceder a la aplicación: "SELECT * FROM usuarios WHERE usuario = 'nombre_usuario'". La variable 'nombre_usuario', junto con la contraseña introducida por el usuario, se envía como argumento en la URL. Posteriormente, se comprueba si estos parámetros coinciden con los obtenidos de la base de datos, devolviendo true o false cuando

corresponda

```
if (user.equalsIgnoreCase(usuario) && password.equals(contraseña)) {

    acceso = true;

} else {
    acceso = false;
    System.out.println("Acceso denegado");
}
```

Figura 5-8: Comprobación de credenciales

- Método "ActualizarLimites": Este método es llamado por la operación "/valoresLimite" de tipo PUT, empleado para actualizar los valores límites establecidos por el usuario. Estos valores son utilizados por OpenHAB en sus reglas, explicadas en el capítulo anterior. Devuelve un booleano, indicando si se han actualizado todos los valores correctamente o no.

Para actualizar estos datos, se realiza la siguiente consulta de actualización a la base de datos:

"UPDATE limites SET limite_temperatura = 'valor_temperatura', limite_humedad = 'valor_humedad', limite_luz = 'valor_luminosidad', limite_humedadterrestre = 'valor_humedadTerrestre'". Todos estos valores personalizados por el usuario son enviados en el cuerpo del paquete que se envía al servidor. Para obtenerlos, en la definición de la operación, se utiliza la clase personalizada 'Limites', la cual contiene una serie de getters/setters que permiten adquirir los datos del cuerpo del mensaje.

```
@PutMapping("/valoresLimite")
public Map<String, String> Actualizalimites (@RequestBody Limites limites) {
    Float limiteTemperatura = limites.getLimiteTemperatura();
    Float limiteHumedad = limites.getLimiteHumedad();
    int limiteLuminosidad = limites.getLimiteLuminosidad();
    Float limiteHumedadTerrestre = limites.getLimiteHumedadTerrestre();
```

Figura 5-9: Definición de /valoresLimite

La definición de la clase Limites:

```java
public class Limites {

    private Float limiteTemperatura;
    private Float limiteHumedad;
    private int limiteLuminosidad;
    private Float limiteHumedadTerrestre;

    public void setLimiteTemperatura(Float limiteTemperatura) {
        this.limiteTemperatura = limiteTemperatura;
    }

    public void setLimiteHumedad(Float limiteHumedad) {
        this.limiteHumedad = limiteHumedad;
    }

    public void setLimiteLuminosidad (int limiteLuminosidad) {
        this.limiteLuminosidad = limiteLuminosidad;
    }

    public void setLimiteHumedadTerrestre (Float limiteHumedadTerrestre) {
        this.limiteHumedadTerrestre = limiteHumedadTerrestre;
    }

    public Float getLimiteTemperatura() {
        return limiteTemperatura;
    }

    public Float getLimiteHumedad() {
        return limiteHumedad;
    }

    public int getLimiteLuminosidad() {
        return limiteLuminosidad;
    }

    public Float getLimiteHumedadTerrestre() {
        return limiteHumedadTerrestre;
    }


}
```

Figura 5-10: Definición clase limites

Una vez que la base de datos ha sido actualizada correctamente, se envía como resultado true.

- Método "ConsultarLimites": Este método es llamado por la operación "/valoresLimites" de tipo GET y es empleado por OpenHAB para obtener los datos limites que el usuario haya definido a través de la app. Como resultado devuelve una lista Map<String, String>.

En este método únicamente se realiza la petición "SELECT * FROM limites" a la base de datos, los datos obtenidos se almacenan en una lista y se envían como resultado:

```
String sql = "SELECT * FROM limites";
PreparedStatement st = conn.prepareStatement(sql);
ResultSet rs = st.executeQuery();

while (rs.next()) {
    temperatura = rs.getFloat("limite_temperatura");
    listaLimites.add(temperatura);
    humedad = rs.getFloat("limite_humedad");
    listaLimites.add(humedad);
    luz = rs.getFloat("limite_luz");
    listaLimites.add(luz);
    humedadTerrestre = rs.getFloat("limite_humedadterrestre");
    listaLimites.add(humedadTerrestre);
}
```

Figura 5-11: Configuración ConsultarLimites

- Método "ConsultarDatos": utilizado por la operación "/datos" y de tipo GET, es empleado por la aplicación móvil para obtener todos los datos almacenados en la base de datos relativos a la configuración ambiental del invernadero. Devuelve una lista como resultado.

En este método se realizan varias consultas a varias tablas de la base de datos:

```
String sql1 = "SELECT value from item0001 ORDER BY time DESC LIMIT 1";
String sql2 = "SELECT value from item0002 ORDER BY time DESC LIMIT 1";
String sql3 = "SELECT * FROM limites";
String sql4 = "SELECT value from item0010 ORDER BY time DESC LIMIT 1";
String sql5 = "SELECT value from item0011 ORDER BY time DESC LIMIT 1";
String sql6 = "SELECT value from item0012 ORDER BY time ASC LIMIT 1";
```

Figura 5-12: Peticiones enviadas a la base de datos

Como se puede apreciar, en la mayoría de las consultas existe una cláusula llamada "DESC LIMIT 1" que permite obtener el último dato que hay almacenado en cada tabla.

El resultado de cada una de las consultas se almacena en una lista, y ésta es enviada como resultado del método. Sin embargo, existen algunos datos que son modificados antes de ser almacenados en ella. Se corresponden con el valor del nivel de agua y de la humedad terrestre y su motivo es que la información enviada directamente desde los sensores carece de significado por sí misma y para representarla en la aplicación móvil es necesario convertirla en tanto por ciento. Por ello, se realizan las siguientes operaciones matemáticas:

- Para la obtención del nivel de agua, la cuenta realizada es:

$$nivelAgua = \frac{(datoAgua * 100)}{1800}$$

dónde ‘datoAgua’ se corresponde con el dato obtenido de la base de datos. De esta forma, se consigue representar el nivel del agua en porcentaje.

- Para la obtención de la humedad terrestre, la cuenta realizada es:

$$humedadTerrestre = 100 - \left(\frac{(datoHumTierra * 50)}{2047}\right)$$

dónde ‘datoHumTierra’ se corresponde con el dato obtenido de la base de datos. De esta forma, se consigue representar la humedad terrestre en porcentaje.

- Método “EnviarComando”: utilizado por la operación “/comando” y de tipo POST, permite enviar una orden sin voz directamente al bróker de Kafka, como se explicó en el apartado 3.2.5. Al no tratarse de un comando de voz, es innecesario que éste sea tratado por Voice2JSON. Devuelve una lista Map en la que se indica si se ha gestionado correctamente o no.

Para que el comando sea manejado de forma similar a uno con voz, este método crea, al igual que el método “subir”, un fichero de texto en una ubicación específica en el que se escribirá, en formato JSON, el comando completo dentro del campo “text”, dado que, cuando sea enviado al consumidor Kafka, el código Python buscará únicamente dicho campo.

```
String ruta = "/home/salas/Escritorio/Pruebas/fichero.txt";
String texto = "{\"text\": \"" + comando + "\"}";
File fichero = new File(ruta);

if ( !fichero.exists() ) {
    fichero.createNewFile();
    System.out.println("Fichero creado");
}

FileWriter fw = new FileWriter(fichero);
BufferedWriter bw = new BufferedWriter(fw);
bw.write(texto);
bw.close();
```

Figura 5-13: Configuración EnviarComando

- Método “IntroducirDatoIntBBDD”: este método es ejecutado por la operación “/datoInt”, es de tipo GET y es utilizado por OpenHAB para cuando éste recibe un nuevo valor procedente de algún sensor y tenga que ser almacenado en la base de datos. No devuelve ningún dato.

Junto con la URL viajan dos parámetros: el valor del sensor y la tabla en la que se tiene que almacenar. La sentencia SQL que permite la inserción del nuevo dato es: “INSERT INTO ‘nombre_tabla’ (time, value) VALUES (NOW(), ‘’nuevo_valor)”. La función NOW() permite obtener la fecha y la hora local.

```
String sql = "INSERT INTO " + tabla + " (time, value) VALUES ( NOW(),"+ dato +" )";
```

Figura 5-14: Configuración IntroducirDatoIntBBDD

- Método "IntroducirDatoStringBBDD": método ejecutado por la operación "/datoString" y es de tipo GET. Su funcionamiento es similar al anterior. La única diferencia es que el valor del sensor que se envía en la URL es de tipo String con el objetivo de conservar sus decimales.

- Método "contarAguaConsumida": ejecutado por la operación "/datoAgua" y de tipo GET, permite al servidor calcular el tiempo de funcionamiento del motor de agua. Este dato será necesario para la obtención de la cantidad del agua consumida mediante la fórmula:

$$cantidadAgua = 25 * t \ ,$$

siendo 25 una constante que representa la velocidad del motor (en ml/s).

La conexión a la base de datos es posible gracias al fichero ConexionBBDD.java. En él, se carga el driver correspondiente a la base de datos PostgreSQL, se especifica la URL donde se aloja la base de datos y se añade el usuario y la contraseña para el acceso a sus tablas.

5.3 Aplicación Android

Con la aplicación Android llamada Conecta Mi Huerto, el usuario final podrá gestionar y controlar por completo el invernadero, observando el progreso y el estado en el que se encuentra, estableciendo límites ambientales, etc. La aplicación móvil se ha desarrollado usando el IDE de Android Studio, utilizando el lenguaje Java.

A continuación, se muestra un diagrama de casos de uso que representa las funcionalidades que ofrece la aplicación:

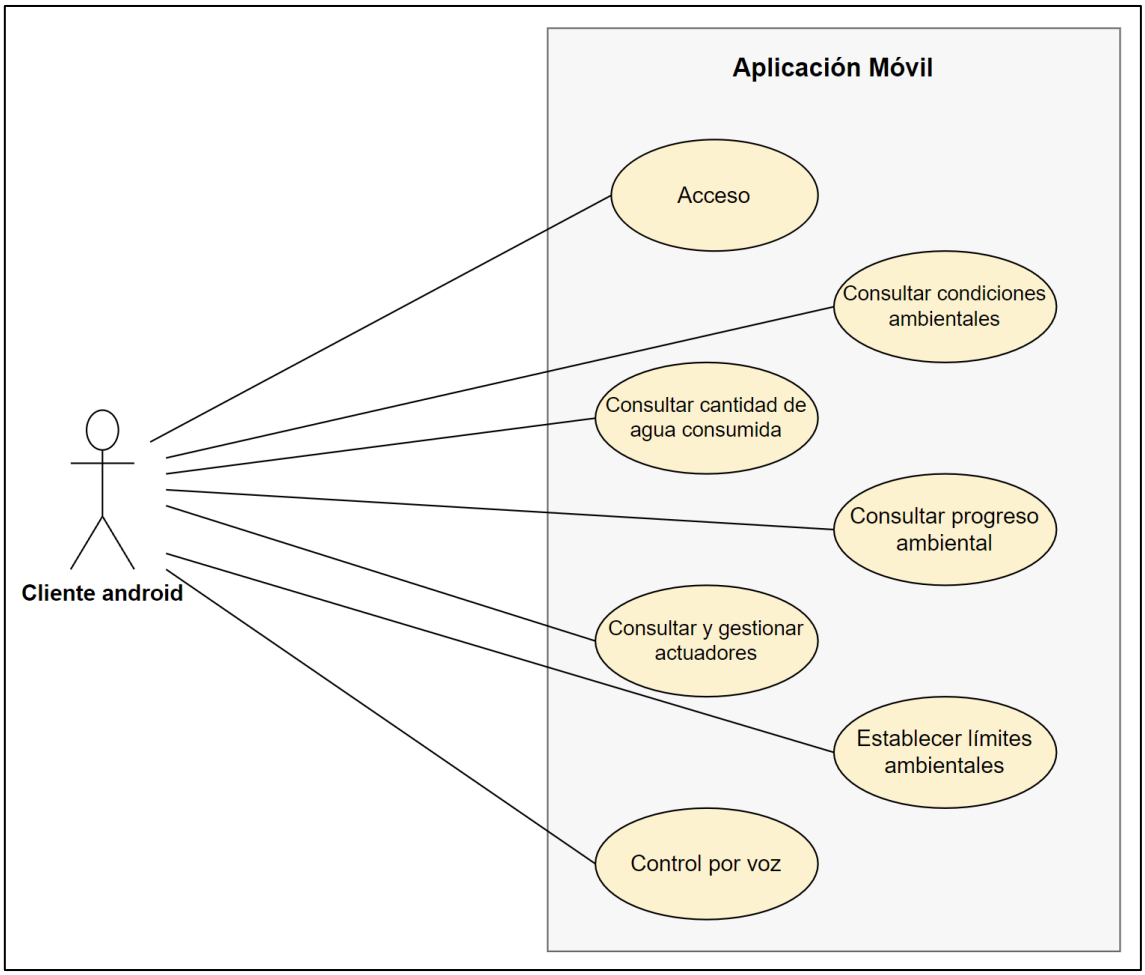

Figura 5-16: Diagrama de casos de uso

5.3.1 Organización

La aplicación consta principalmente de seis actividades por las que el cliente se puede desplazar a través de un menú. Sin embargo, antes de interactuar con cualquiera de éstas, se muestra en el arranque de la app una actividad en la que se inicia sesión. En ella, se introducirán los datos de usuario y contraseña correspondientes y pulsando posteriormente en el botón, se enviará una petición POST al servidor REST para comprobar las credenciales introducidas.

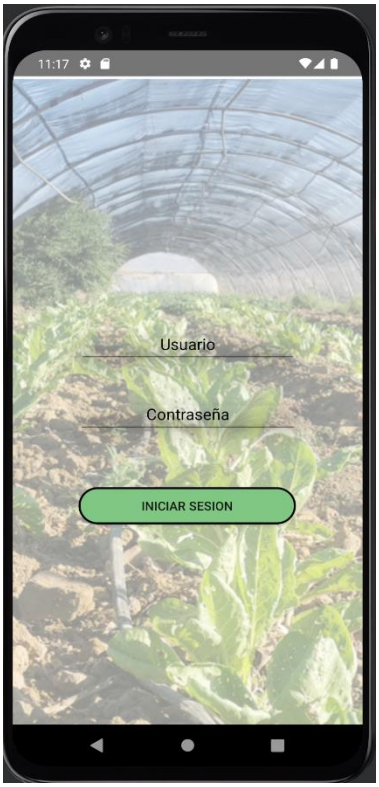

Figura 5-17: Actividad MainActivity

Una vez iniciada la sesión, la primera actividad en ejecución, llamada “Inicio”, mostrará un resumen de todas las condiciones ambientales actuales del invernadero. Cada uno de estos datos se obtienen del servidor al realizar una petición de tipo GET.

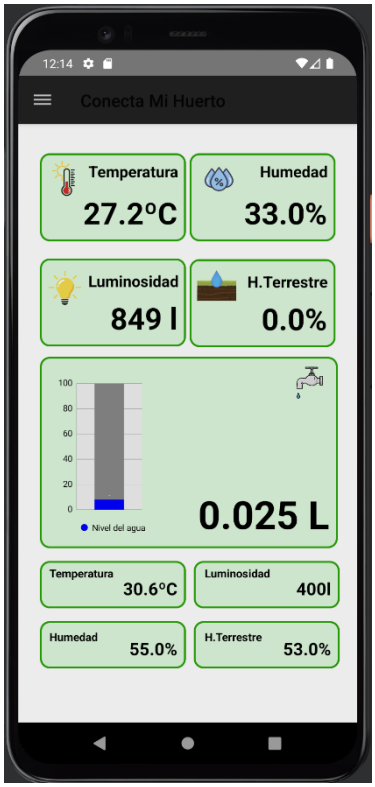

Figura 5-18: Actividad Inicio

Esta actividad no está pensada para que el usuario gestione el sistema desde ella. Para ello, existe otra llamada “Gestión” que permite, no solo obtener los valores límites ambientales establecidos actualmente en el invernadero, sino que, además, admite la posibilidad de establecer unos límites personalizados según las condiciones óptimas de cada cultivo, entre otras funcionalidades.

Para manejarse entre todas las actividades, existe un menú ubicado en la esquina superior izquierda, accesible en cualquier momento de ejecución de la aplicación una vez iniciada sesión con el servidor.

Figura 5-19: Menú desplegable

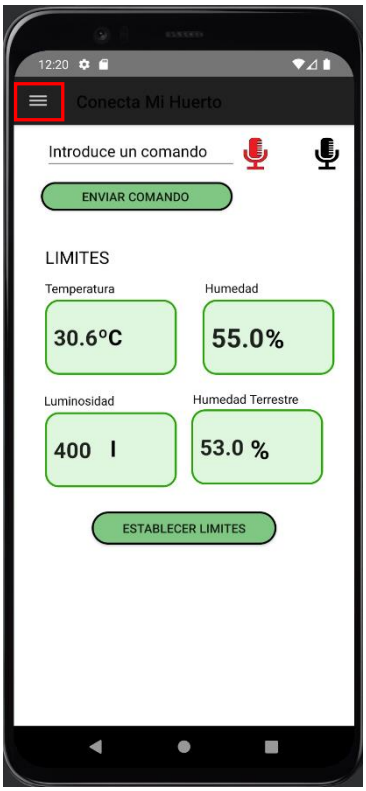

Figura 5-20: Actividad Gestión

Como se puede observar, desde la actividad "Gestión" se envían las ordenes al servidor para que estas sean atendidas. Estas órdenes se pueden decidir mediante dos formas:

- Utilizando una orden por defecto, mediante el campo de escritura situado en la parte superior de la actividad. Cada vez que se empiece a escribir una orden, aparecerá una lista con todos los comandos disponibles que mejor coincidan con las letras/palabras escritas en este campo. Se muestra un ejemplo a continuación:

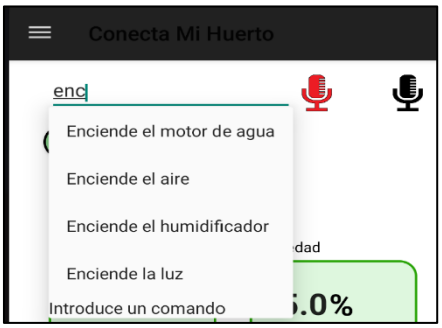

Figura 5-21: Comandos predefinidos

- Utilizando el micrófono del dispositivo móvil, para enviar un comando por voz. Esta forma se realiza pulsando sobre el micrófono de color negro situado en la esquina superior derecha. Al pulsarlo, este icono se coloreará en rojo, indicando el comienzo de la grabación. El micrófono

ubicado a la izquierda del anterior, permite escuchar el comando de voz una vez terminada la grabación.

El envío tanto del comando de voz como del escrito se realiza a través de una petición POST al servidor para que sean tratados de la manera oportuna, según se explicó en el apartado 5.2. Sin embargo, la forma empleada para enviar el comando de voz es distinta a la utilizada en el resto de las peticiones POST.

Para el envío de los audios, se hace uso de la librería 'Retrofit'. Esta permite crear un cliente REST seguro para Android, que se encarga de la serialización, deserialización, concurrencia y manejo de las respuestas. Se trata de una alternativa a 'HttpUrlConnection' y 'Volley', utilizada porque permite una gran simplificación de código además de poder realizar el envío de estos archivos de audio de forma sencilla.

Para su uso, es necesario implementar las dependencias necesarias en app/build.gradle:

```
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'com.squareup.retrofit2:converter-scalars:2.9.0'
```

Figura 5-22: Implementación librerías Retrofit

Debido a que la aplicación necesita conectarse a internet para su funcionamiento, es necesario concederle este permiso. Para ello, se le solicita al usuario, mediante una ventana emergente, el uso de internet, además de acceso al almacenamiento interno del dispositivo, utilizado para conservar el fichero de audio grabado.

Para la construcción del mensaje, se crea un objecto de la clase 'Retrofit' al que se le indica la URL del servidor, y a la que se le asocia una interfaz java que contiene la estructura de los posibles tipos de peticiones. En dicha interfaz, debido a que solo se usa esta librería para enviar el comando por voz, solo se específica un método para una operación de tipo POST.

```
public interface InterfazAPI {
    //Post request
    1 usage
    @Multipart
    @POST("subir")
    Call<ResponseBody> subirAudio(@Part MultipartBody.Part audioFile);
}
```

Figura 5-23: InterfazAPI

En el cuerpo del mensaje se incluye el fichero de audio grabado que ha sido almacenado en el dispositivo y seleccionando el anterior método encargado de la petición, es enviado al servidor. Al mandarlo, esta librería también permite tanto gestionar las respuestas recibidas como tratar los fallos que pudieran aparecer.

La otra forma utilizada para el envío de comandos en texto plano se realiza a través de la librería Volley. Los mensajes se construyen indicando la URL del servidor, el tipo de método HTTP (POST) y el cuerpo del mensaje que contiene el comando. Esta librería, al mandar la petición, también permite gestionar tanto las respuestas recibidas como los fallos ocasionados durante la transmisión.

La actividad “Graficas”, como su propio nombre indica, se utiliza para mostrar visualmente la evolución que ha tenido el invernadero desde distintas condiciones ambientales. En ellas se pueden observar el progreso que ha tenido la temperatura, la humedad ambiental, la luminosidad o la humedad terrestre en diferentes periodos de tiempo. Los tipos de gráficas utilizados han sido: gráfica de barras vertical, para la representación de las humedades y la gráfica de puntos, para la representación de la temperatura y de la luminosidad.

En las siguientes figuras se pueden observar a modo de ejemplo las evoluciones de la temperatura y humedad ambiental en el último día.

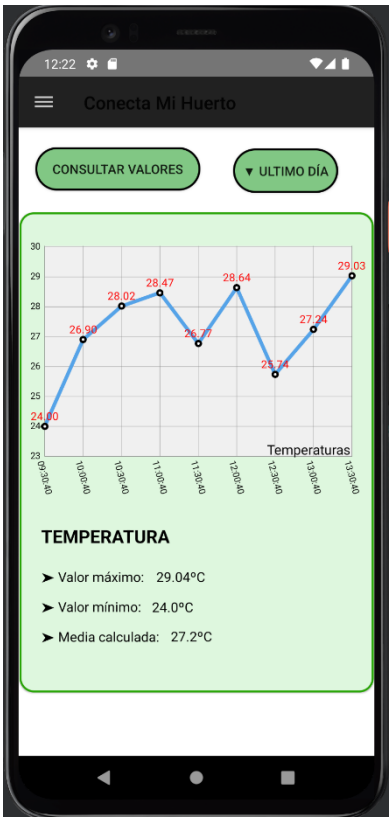

Figura 5-24: Gráfica temperatura

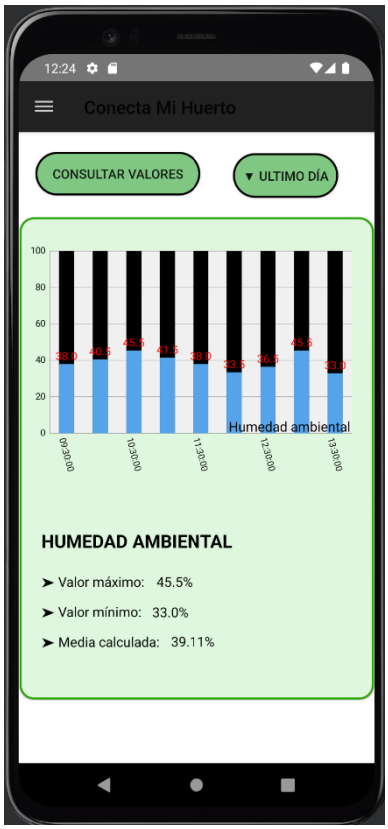

Figura 5-25: Gráfica humedad ambiental

La librería ‘MPAndroidChart’ es la encargada del desarrollo de estas gráficas. Al tratarse de una biblioteca externa, es necesario implementarla en el archivo build:gradle del proyecto:

```
implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'
```

Figura 5-26: Implementación librerías MPAndroidChart

Antes de que la actividad dibuje las gráficas, solicita previamente los datos al servidor, utilizando la URL: http://192.168.1.62:9000/servidor/consulta?seleccion='tabla'&opcion='opcionElegida', siendo ‘tabla’, el nombre de la tabla de la base de datos de la que se quiere obtener la información y ‘opcionElegida’ el rango temporal. Estos datos serán los valores enviados por los sensores y los tiempos en los que han sido almacenados en la base de datos y ambos conjuntos serán representados en el eje OY y en el eje OX respectivamente.

En todas las gráficas se muestran nueve puntos, de los cuales se calcula el máximo, el mínimo y la media entre ellos para mostrarlos en la parte inferior de la actividad.

La pestaña “Información” del menú lleva a una lista en la que se detalla, a modo de curiosidad, cuáles son las condiciones óptimas de diferentes cultivos en el invernadero.

La pestaña “Configuración” permite dirigirse a una actividad en la que se puede cambiar tanto la dirección del servidor como el puerto a la que van dirigidas todas las peticiones usadas en la aplicación.

Estas variables son almacenadas en una clase get/set de Java y al pulsar sobre el botón “Guardar Cambios”, son accesibles desde cualquier actividad de la app.

Figura 5-27: Actividad Información

Figura 5-28: Actividad Configuración

Por último, la pestaña “Salir” permite cerrar y abandonar la aplicación.

5.3.2 Conexión entre componentes

A continuación, se muestra en un diagrama de secuencia el paso de mensajes que tiene lugar cuando se pretende encender la bomba de agua mediante una orden por voz. Se suponen que todas las conexiones están iniciadas.

Este proceso comienza cuando el usuario ejecuta la aplicación móvil. Lo primero que se le pedirá será que se autentifique para poder acceder. Una vez introducido el usuario y la contraseña, la app envía una petición HTTP POST al servidor REST. Este, en función del nombre del usuario introducido, realizará una consulta a la base de datos para obtener, si existe, dicho usuario y la contraseña correspondiente. Si las credenciales introducidas y las almacenadas en la base de datos coinciden, el

servidor envía como respuesta una variable booleana con valor true, permitiendo que la aplicación cambie de actividad.

La nueva actividad mostrada será la de 'Inicio', y mediante el menú, el usuario podrá dirigirse a la actividad 'Gestión', en la que se configura el invernadero. Desde ella, el usuario, pulsando sobre el icono del micrófono, grabará una orden que, una vez terminada, recibirá el servidor presionando sobre el botón "Enviar comando". La grabación será enviada en un archivo de audio de tipo 3gp en el cuerpo de una petición POST.

Una vez que el servidor recibe este mensaje, extrae de él el fichero de audio y mediante la herramienta de línea de comandos ffmpeg, realiza la conversión de 3gp a wav.

Posteriormente, el servidor, al tener el archivo en formato wav, lo envía al programa Voice2JSON para que realice la transcripción de audio a texto. Al acabar este proceso, el resultado JSON se escribe en un fichero y se almacena en un directorio.

Cuando el script "enviar_comando.sh" detecta la existencia de este fichero, lee su contenido para enviarlo, como productor, al bróker de Kafka a través del topic "Thing1" y seguidamente será reenviado al programa Python, configurado como suscriptor de este topic. Por último, el script será el encargado de eliminar el fichero consiguiendo de esta manera estar preparado para la recepción de nuevos comandos.

El programa Python, en función de la orden recibida, enviara una petición POST al servidor REST de OpenHAB con el valor data establecido a ON reflejando la voluntad del encendido del motor. Al recibirlo, OpenHAB envía un mensaje que contiene un entero 1, de tipo PUBLISH, al servidor MQTT a través del topic "Entrada0/1". Esta petición no tiene una respuesta por parte del servidor, al ser enviada con el nivel QoS establecida a 0.

Cuando la placa ESP32 recibe el mensaje PUBLISH reenviado por el servidor MQTT, comprueba el topic del cual proviene y el valor que transporta el mensaje. Al proceder de "Entrada0/1" y con un valor 1, enviará la orden LOW al pin conectado al relé, ya que éste funciona a nivel bajo. Cuando el relé se desactive, el motor recibirá la tensión suficiente de las cuatro pilas instaladas y bombeará el agua.

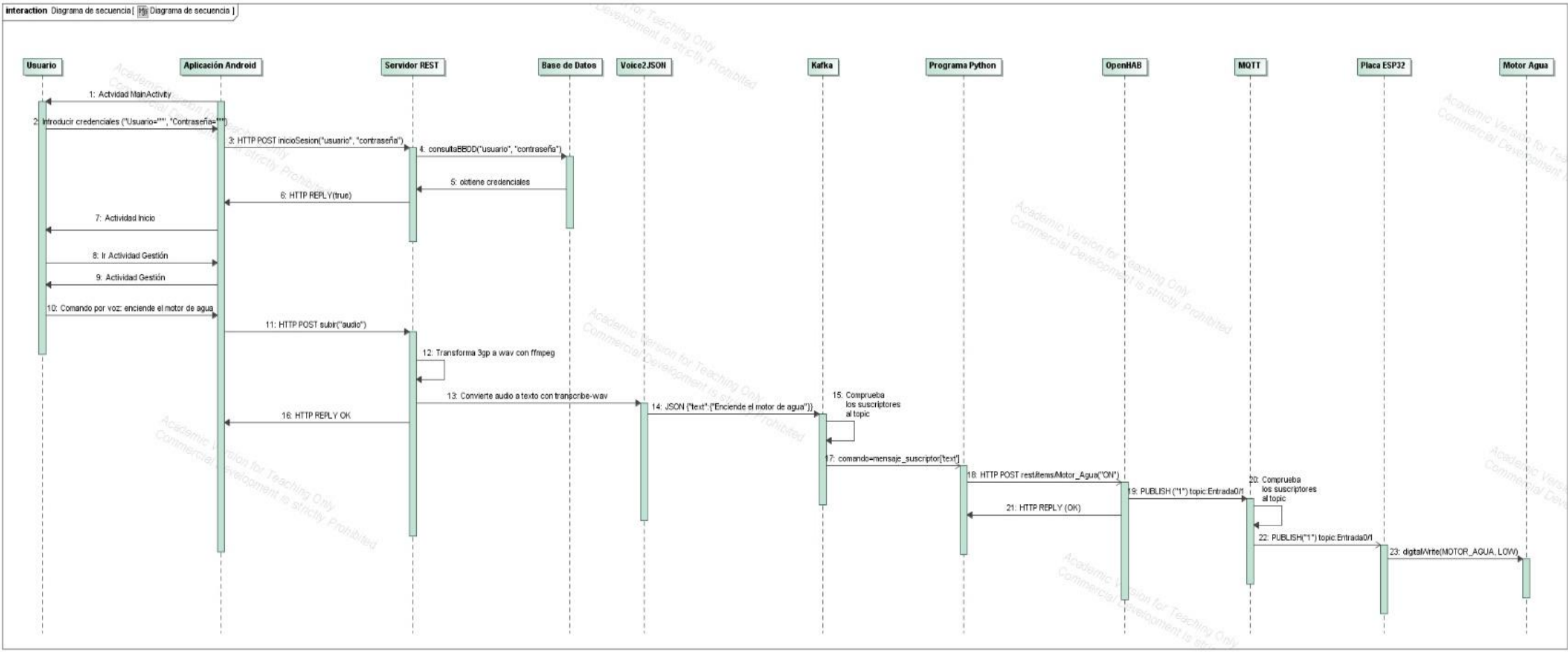

Figura 5-29: Diagrama de secuencia

6 CONCLUSIONES Y FUTURAS AMPLIACIONES

Gracias al internet de las cosas, el mundo tal y como se conoce actualmente, está evolucionando digitalmente. Multitud de avances afirman este cambio, como pueden ser el nacimiento de las ciudades inteligentes, en las que los servicios públicos están controlados a través de internet, o la aparición de las monedas virtuales, utilizadas para realizar cualquier tipo de transacción.

Este cambio se ha visto altamente respaldado por la reciente pandemia vivida como consecuencia de la Covid-19. Durante este tiempo, el uso de internet permitió que el mundo siguiera comunicado a pesar de las estrictas restricciones, evitando así una fuerte involución que hubiera afectado, no solo a las actuales generaciones, también a las venideras.

La aparición de nuevas tecnologías, como la red 5G o la inteligencia artificial, incentivarán aún más el uso de dispositivos IoT conectados a Internet. Ámbitos como la educación, vigilancia o la sanidad se verán beneficiados por el uso de dichas tecnologías, ya que otorgarán una mayor eficiencia y productividad en el desarrollo de sus procesos y operaciones.

Sin embargo, este cambio también puede suponer riesgos para la seguridad o intimidad de las personas, al abrir espacios privados hacia el exterior o la publicación de datos confidenciales en caso de que los sistemas IoT sean vulnerados.

En este trabajo se ha conseguido administrar y gestionar un pequeño invernadero simulado y gracias a la gran escalabilidad que ofrecen los programas con los que se ha desarrollado, se podría emplear para el control de invernaderos a gran escala. Se ha diseñado y configurado poniendo de manifiesto su bajo coste, teniendo en cuenta únicamente los distintos tipos de sensores, actuadores utilizados y la placa ESP32.

Además, se ha demostrado que la interconexión entre Kafka y OpenHAB es posible, a pesar de no haber en este último bindings específicos que lo faciliten, como puede ser el usado para la comunicación con MQTT. Esto tiene lugar gracias a que OpenHAB dispone de su propia API REST, a través de la cual se han podido controlar todos los elementos físicos existentes en el invernadero. Así mismo, las reglas configuradas en OpenHAB permiten la automatización del invernadero.

Por otro lado, el programa Voice2JSON ofrece un software gratuito de código abierto utilizado para la conversión de audio a texto y que ha permitido la interacción entre la voz y el invernadero. Con él, se han resuelto los inconvenientes ocasionados al trabajar en estas instalaciones, como pueden ser la manipulación de herramientas húmedas o el contacto con la tierra. Sin embargo, su uso no garantiza completamente que los procesos de transcripción se realicen correctamente ya que, al ser de código abierto, los conjuntos de datos utilizados para las conversiones no son tan extensos como los de las entidades privadas.

Finalmente, la aplicación Android otorga al usuario la posibilidad de poder gestionar el invernadero de acuerdo con sus necesidades. A través de ella, se observan las condiciones ambientales actuales

del invernadero, se configuran límites ambientales para accionar los distintos actuadores de forma automática cuando el estado del clima no se adapte a estos valores. Además, mediante toda la información proporcionada de forma gráfica, se puede obtener la máxima rentabilidad del suelo cultivado, aumentar la producción y calidad de las cosechas y optimizar el consumo del agua.

Este Trabajo Fin de Grado es un proyecto que empezó con la idea de conectar dos programas, y fue ampliándose hasta llegar al trabajo descrito en la presente memoria. Sin embargo, muchas han sido las ideas que no han sido implementadas para no extender en exceso el contenido del mismo. Por ello, en futuras versiones de este proyecto se podrían incorporar nuevas mejoras en campos como el de la seguridad, el rendimiento o el diseño:

6.1 Líneas futuras: Seguridad

- Encriptar los pares usuario:contraseña almacenados en la base de datos para el acceso en la app. Actualmente, el servidor realiza la comprobación de las credenciales sin utilizar ningún tipo de programa que encripte las claves.
- Envío de las peticiones a través del protocolo HTTPS.

6.2 Líneas futuras: Diseño

- Ofrecer la posibilidad de poder cambiar la frecuencia a la que se recogen los datos. De esta forma, la gestión del invernadero se ajusta a las necesidades propias del usuario.
- Ofrecer la capacidad de mostrar los datos que se vayan recogiendo del invernadero en tiempo real, utilizando para ello, por ejemplo, WebSockets.
- Instalar todos los servidores y programas en la nube, de forma que sea posible acceder a la gestión del invernadero desde cualquier lugar y no estar necesariamente en la misma red local.
- Utilizar sensores que sean más precisos a la hora de recoger datos, con menos margen de error. Por ejemplo, sustituir el sensor de temperatura y humedad DHT11 por el DHT22 o utilizar un sensor de ultrasonidos en lugar de uno resistivo para conocer el nivel del agua.
- Utilizar otro bróker MQTT con una mayor disponibilidad, ya que el que se ha usado en el trabajo, a pesar de ser de uso público, en ocasiones no se podía utilizar en ciertos periodos del día.
- Mejorar el proceso de conversión audio a texto, de forma que se implementara algún procedimiento para que se verificase si la transcripción realizada se corresponde con la orden por voz. Se podría realizar instalando el programa Voice2JSON en la aplicación Android.

6.3 Líneas futuras: Funcionalidad

- Crear un acceso exclusivo a administradores para que puedan cambiar la configuración del sistema, como añadir nuevos usuarios o borrar datos de la base de datos.
- Incluir la posibilidad de eliminar los datos que están almacenados en la base de datos, ya sea por decisión del usuario o de forma automática.
- Crear notificaciones o alertas de forma que cuando se supere un límite establecido, la aplicación lo muestre, aunque el usuario no la esté ejecutando en primer plano.

- Permitir al usuario la posibilidad de cancelar el accionamiento automático de los actuadores.
- Permitir al usuario actualizar el contador del agua consumida a través de la app.

REFERENCIAS

[1] Oracle: What is IoT. https://www.oracle.com/es/internet-of-things/what-is-iot/

[2] Redhat: What is IoT. https://www.redhat.com/es/topics/internet-of-things/what-is-iot

[3] OpenSistemas: Dispositivos IoT. https://opensistemas.com/dispositivos-iot/

[4] Xiomi Token Extractor for Mi Smart LED Bulb Essential (White and Color). https://gadget-freakz.com/xiaomi-token-extractor/

[5] Software Arduino. https://www.arduino.cc/en/software

[6] Página oficial Spring. https://spring.io/

[7] Página oficial PostgreSQL. https://www.postgresql.org/

[8] Página oficial Android Studio. https://developer.android.com/studio

[9] USB to UART bridge vcp drivers by Silicon Labs. https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers

[10] Página oficial de Microsoft Windows. https://www.microsoft.com/es-es/windows/?r=1

[11] Amazon: Motor de Agua RUNCCI-YUN: https://www.amazon.es/RUNCCI-YUN-Sistema-Autom%C3%A1tico-Arduino%EF%BC%8CKit-Suelo%EF%BC%8Cpara/dp/B088T64ZT2

[12] Amazon: Set de placa y sensores ELEGOO. https://www.amazon.es/gp/product/B01MQPT9OD

[13] Amazon: ESP32 ESP-WROOM-32 NodeMCU de AZDelivery. https://www.amazon.es/gp/product/B071P98VTG?th=1

[14] Gramática para Voice2JSON. https://www.w3.org/TR/jsgf/

[15] Página oficial de Voice2JSON. http://voice2json.org/

[16] Página oficial de Voice2JSON: How it Works. http://voice2json.org/whitepaper.html

[17] Página oficial de Kafka. https://kafka.apache.org/

[18] Página oficial de Kafka: Documentación. https://kafka.apache.org/documentation/

[19] Página oficial de OpenHAB. https://www.openhab.org/

[20] Página oficial de OpenHAB: Documentación. https://www.openhab.org/docs/

[21] Página oficial de OpenHAB: Instalación en Windows. https://www.openhab.org/docs/installation/windows.html

[2] OASIS. MQTT v3.1.1. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf

[23] Libro electrónico ESP32. ESP32 NodeMCU ? Libro electrónico gratuito de AZ-Delivery

[24] Grandes Empresas: IoT en Invernaderos https://hablemosdeempresas.com/grandes-empresas/iot-en-invernaderos/

[25] Geeksforgeeks: Formato de los mensajes Kafka https://www.geeksforgeeks.org/how-kafka-

producers-message-keys-message-format-and-serializers-work-in-apache-kafka/

[26] Expresiones Cron. https://www.quartz-scheduler.org/documentation/quartz-2.2.2/tutorials/tutorial-lesson-06.html

[27] Librería Kafka para Python. https://kafka_python.readthedocs.io/en/master/apidoc/KafkaConsumer.html

[28] Enlace para la descarga del fichero libffi6_3.2.1-8_amd64.deb. http://mirrors.edge.kernel.org/ubuntu/pool/main/libf/libffi/libffi6_3.2.1-8_amd64.deb

[29] JSON con las librerías necesarias para la configuración del ESP32 a través del Arduino IDE. https://dl.espressif.com/dl/package_esp32_index.json

[30] MQTT público de EMQX Cloud. https://www.emqx.com/en

[31] Página oficial de Kafka: Descarga. https://kafka.apache.org/downloads

ANEXO A: INSTALACIÓN DE VOICE2JSON

En este anexo se explica los pasos que se han seguido para instalar el programa Voice2JSON en la máquina virtual que proporciona la Universidad de Sevilla.

El software de descarga se encuentra en la página oficial de Voice2JSON, en el apartado "Getting Started", concretamente en el subapartado "Install" voice2json [15].

Se descargará el archivo .deb correspondiente, según el tipo de arquitectura del dispositivo en el que se vaya a instalar. En mi caso, para conocer la arquitectura de la máquina virtual, se introduce el siguiente comando con el que obtenemos esta información:

dpkg-architecture | grep DEB_BUILD_ARCH=

obteniendo como resultado una arquitectura del tipo amd64:

```
salas@localhost:~$ dpkg-architecture | grep DEB_BUILD_ARCH=
DEB_BUILD_ARCH=amd64
```

Figura A-1: Arquitectura máquina virtual

Por lo que, se descargará el paquete correspondiente:

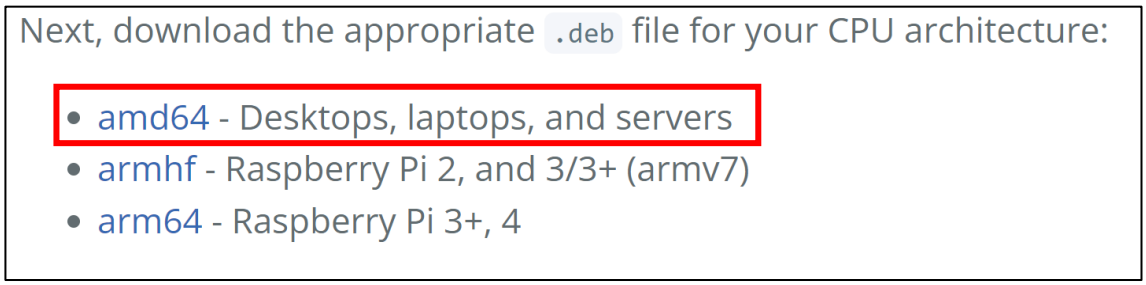

Figura A-2: Paquetes para diferentes arquitecturas

Posteriormente, se instalará el paquete .deb descargado haciendo uso del comando:

sudo apt install /path/to/voice2json_<VERSION>_<ARCH>.deb

donde,

· VERSION es la versión del paquete Voice2JSON (en mi caso será la v2.1)

· ARCH es el tipo de arquitectura (amd64)

Una vez instalado, ya se tendrá disponible el programa para convertir voz a JSON. Sin embargo,

para poder ejecutar el software de forma correcta, en mi caso tuve que realizar dos pasos adicionales:

- Otorgar permiso de lectura al fichero libssl.so.1.1, necesario porque este es importado por archivos Python de Voice2JSON para su funcionamiento, utilizando el comando:

```
salas@localhost:~$ sudo chmod +r /usr/local/lib/libssl.so.1.1
```

Figura A-3: Comando para otorgar permiso de lectura

- Voice2JSON requería un fichero llamado libffi.so.6. Sin embargo, mi sistema disponía de este fichero en otra versión (libfii.so.8). Para solucionarlo, se descargó el paquete libffi6_3.2.1-8_amd64.deb del enlace [28]. Una vez descargado, se instaló usando el comando:

```
salas@localhost:~$ sudo apt install /home/salas/Descargas/libffi6_3.2.1-8_amd64.deb
```

Figura A-4: Comando para instalar el paquete libffi6_3.2.1-8_amd64.deb

ANEXO B: COMANDOS PARA LA EJECUCIÓN DEL SERVICIO KAFKA

En este anexo se explican cuáles han sido los pasos para poner en funcionamiento el servicio Kafka. Todos los comandos mostrados a continuación permiten la ejecución de scripts que se encuentran en un directorio comprimido al descargase [31]. En este trabajo, la ubicación de la carpeta raíz de Kafka se ha colocado en la el directorio 'Descargas' de la máquina virtual.

Ejecución del zookeeper

El zookeeper permite coordinar cada uno de los brókers existentes en el clúster. Utilizando una terminal, y ubicandose en el directorio raíz de Kafka, se ejecuta el siguiente comando:

```
salas@localhost:~/Descargas/kafka_2.13-3.0.0$ ./bin/zookeeper-server-start.sh config/zookeeper.properties
```

Figura B-1: Comando para ejecutar el zookeeper

Permite ejecutar el zookeeper con las propiedades definidas en el fichero de configuración zookeeper.properties.

Ejecución servidor Kafka

Con el siguiente comando se ejecuta el bróker con las configuraciones establecidas en el fichero server.properties. Este se ejecutará localmente en el puerto 9092.

```
salas@localhost:~/Descargas/kafka_2.13-3.0.0$ ./bin/zookeeper-server-start.sh config/zookeeper.properties
```

Figura B-2: Comando para ejecutar el servidor

Creación del topic

Los topics en Kafka son las categorías en las que se clasifican los mensajes, de modo que estos se pueden entender como si fuera un flujo de datos. Para la creación de un topic, se ejecuta el siguiente comando:

```
salas@localhost:~/Descargas/kafka_2.13-3.0.0$ ./bin/kafka-topics.sh --create thing1
--bootstrap-server localhost:9092 --replication-factor 1 --partitions 1
```

Figura B-3: Comando para crear el topic

Como ya se comentó en el apartado 3.2: Kafka, para que este servicio pueda funcionar, es necesario que haya tanto un productor como un consumidor. En el proyecto, el productor es el programa Voice2JSON a través del script "enviar_comando.sh" cuando hay un comando por voz o el servidor REST a través de este mismo script cuando es un comando sin voz, y el consumidor el propio fichero Python.

ANEXO C: INSTALACIÓN Y CONFIGURACIÓN DE ARDUINO Y ESP32

En este anexo se detallan los pasos seguidos para la instalación de Arduino IDE así como las librerías necesarias para que la placa ESP32 pudiera ser configurada a través de ella:

Instalación Arduino IDE

Se descarga el software de la página oficial de Arduino [5], concretamente el ejecutable MSI Installer para Windows:

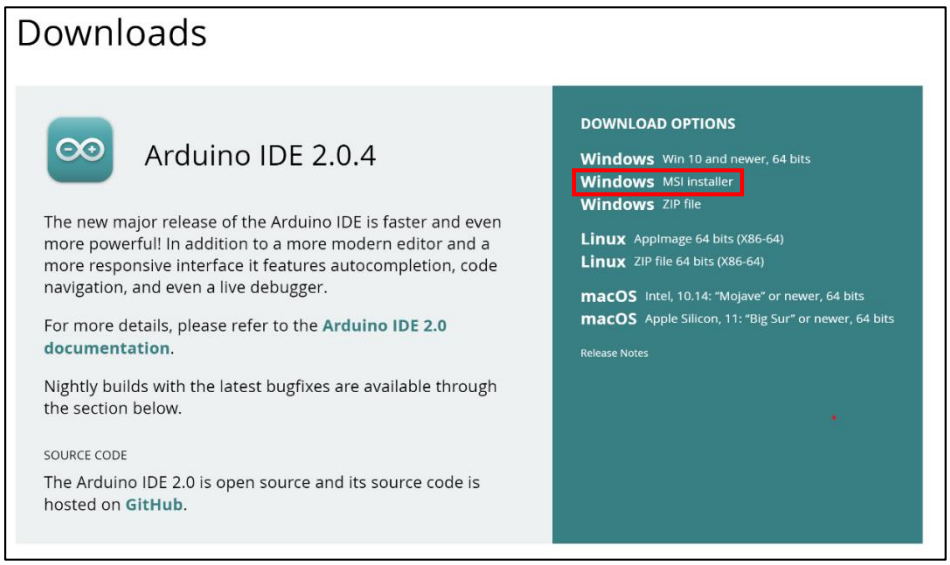

Figura C-1: Descarga del software Arduino

Una vez instalado y ejecutado, se pedirá la instalación de otros softwares adicionales:

- Software Arduino USB Driver, necesario para que la computadora pueda reconocer y comunicarse correctamente con las placas Arduino conectadas a través del puerto USB.

Figura C-2: Software Arduino USB Driver

- Software de Adafruit específico utilizado como controlador para poder conectarse con placas desarrolladas por esta empresa.

Figura C-3: Software Adafruit

- Software Genuino USB Driver, la cual se trata de otro controlador que permite la comunicación entre placas y la computadora.

Figura C-4: Software Genuino USB Driver

Instalación librerías ESP32

Para poder comunicarse y programar la placa ESP32 desde Arduino IDE es necesario instalar unas librerías adicionales. Para ello, desde el menú del IDE de Arduino, en Archivo → Preferencia, en el cuadro de texto llamado *"URLs adicionales de gestor de placas"* se añade el enlace [29].

Este contiene un fichero JSON con las direcciones de todas las librerías necesarias para poder gestionar la placa ESP32.

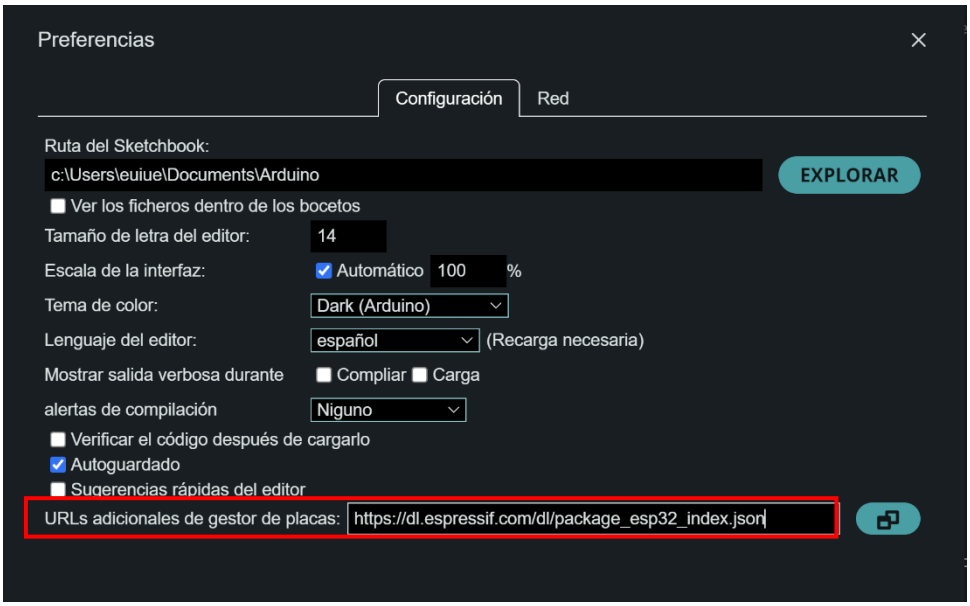

Figura C-5: Configuración del IDE de Arduino

Posteriormente, en Herramientas → Placas → Gestor de placas, se busca la nueva placa descargada y se instala para que pueda ser configurada a través del IDE.

Figura C-6: Instalación de la placa ESP32 en el IDE de Arduino

En caso de que el controlador encargado de la comunicación USB-UART no se haya instalado automáticamente, será necesario descargarlo a través del enlace [9].

Una vez instalado, se podrá comprobar que en el puerto USB del PC se reconoce la placa a través del menú Windows → Administrador de dispositivos:

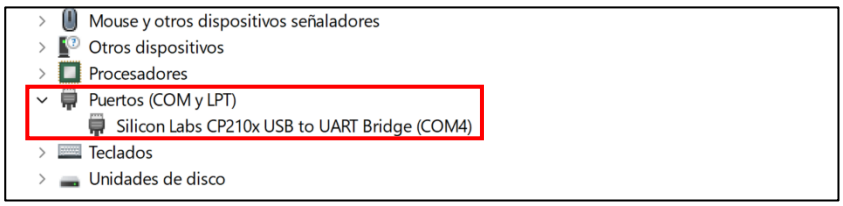

Figura C-7: Software reconocible en el PC

Por último, hay informar que se ha tenido que utilizar un cable USB-micro USB de pequeña longitud para establecer la comunicación entre el host y la placa. De esta forma se asegura que los datos no se atenúan y no hay problemas de conexión.

ANEXO D: CONFIGURACIÓN DE LA BASE DE DATOS

En este anexo se explicará cuáles han sido los pasos necesarios para la configuración de la base de datos utilizada en el proyecto. No ha sido preciso su instalación en la máquina virtual puesto que ya venía implementada.

Al instalarse el SGBD PostgreSQL, se crea dentro del sistema operativo de Linux el superusuario 'postgres' del servidor de la base de datos. Sin embargo, por motivos académicos, la contraseña se modificó a 'pgsqlroot' mediante el siguiente comando con privilegios de superusuario (necesario haber accedido mediante el usuario root):

```
root@localhost:/home/salas# passwd postgres
```

Figura D-1: Comando para cambiar de contraseña al usuario 'postgres'

Posteriormente, para crear el nuevo usuario 'openhab' dentro del SGBD, hay que ejecutar los siguientes comandos:

```
salas@localhost:~$ su postgres
Contraseña:
postgres@localhost:/home/salas$ start-postgresql
postgres@localhost:/home/salas$ psql
psql (14.8 (Ubuntu 14.8-0ubuntu0.22.04.1))
Type "help" for help.

postgres=# CREATE USER openhab WITH ENCRYPTED PASSWORD 'openhab' CREATEDB;
```

Figura D-2: Creación del nuevo usuario 'openhab'

Con el comando "*start-postgresql"* iniciamos el servidor PostgreSQL, y con "psql" ejecutamos la base de datos.

Para la creación de la base de datos 'openhabdb' se utiliza la siguiente orden:

```
postgres@localhost:/home/salas$ createdb openhabdb
```

Figura D-3: Creación de la base de datos 'openhabdb'

Una vez creado el nuevo usuario y la base de datos, es necesario cambiar la configuración correspondiente a la autenticación por contraseñas de los usuarios del SGSB. Para ello, en el fichero pg_hba.conf se incluye la siguiente sentencia:

local all openhab md5

De esta forma, se consigue el acceso desde el usuario 'openhab' a la base de datos 'openhabdb', introduciendo el comando

psql -U openhab openhabdb