Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA DE COMPUTADORES

Trabajo Fin de Grado

Demostrador IoT interactivo mediante Alexa

ESCUELA POLITECNICA SUPERIOR

Autor: Aarón Iglesias Chorro

Tutor/es: Antonio García Herraiz

2021-2022

Universidad de Alcalá

Escuela Politécnica Superior

GRADO EN INGENIERÍA DE COMPUTADORES

Tribunal calificador:

Presidente: Bernardo Alarcos Alcázar

Vocal 1º: Juan Antonio Rodrigo Yanes

Vocal 2º: Antonio García Herráiz

ESCUELA POLITECNICA SUPERIOR

Fecha: Septiembre 2022

Agradecimiento a mi familia y amigos por
haberme ayudado cuando los necesitaba

Índice

Resumen 5
Palabras Clave 5
Abstract 6
Key Words 6
Resumen extendido 7
1. Introducción 8
2. Descripción del trabajo 10
Base Teórica 10
ESP32 10
Mqtt 13
Alexa 14
QoS 16
Descripción Experimental 17
Hardware 17
Software 19
Código MQTT 28
Skill de Alexa 31
3. Arquitectura de la Aplicación 38
4. PRESUPUESTO 40
5. CONCLUSIONES 41
6. Bibliografía 42
7. Anexos y/o Apéndices 44
Instalación de software Arduino con módulo ESP32 44
Instalación de librerías para Python 45
Instalación de Mosquitto - MQTT 46
Muestras de Funcionamiento 47
Crear Skill de Alexa 51

Resumen

Hoy día resulta un desafío acercarse a un público joven cada vez más exigente en materia de nuevas tecnologías. En este contexto, en el presente trabajo se ha perseguido el desarrollo de una skill interactiva de Alexa en Python, la cual permite conocer datos en tiempo real de un demostrador loT, como son la temperatura, el sonido, la intensidad y el movimiento. El resultado de este procedimiento puede abrir un camino prometedor con múltiples aplicaciones en la vida cotidiana.

Palabras Clave

Sensor, Python, Arduino, Aplicación de Alexa, Servicio Web de Amazon, AWS.

Abstract

Today it is a challenge to approach a young audience that is increasingly demanding in terms of new technologies. In this context, in the present work the development of an interactive Alexa skill in Python has been pursued, which allows to know real-time data from an IoT demonstrator, such as temperature, sound, intensity and movement. The result of this procedure can open a promising path with multiple applications in everyday life.

Key Words

Sensor, Python, Arduino, Alexa´s Skill, Amazon Web Service, AWS.

Resumen extendido

La demanda constante de tecnología por parte de los jóvenes implica un rápido desarrollo en cuanto a aplicaciones para dispositivos de uso cotidiano.

Por esta razón, se ha buscado una manera de acercarse a este público objetivo con la idea de generar una nueva forma de aprendizaje a través de una aplicación o skill de Alexa desarrollada en Python. Para la obtención de datos que interaccionan con nuestro dispositivo Alexa, se ha diseñado una placa PCB con cuatro sensores encargados de tomar diferentes valores, como son la temperatura, el sonido, la intensidad de la luz y el movimiento. Para conectar estos sensores se ha utilizado un ESP32 o microcontrolador donde se programa el código de los mismos. El funcionamiento de este dispositivo está desarrollado en lenguaje Arduino y a diferencia de un chip normal, nos permite conexión Wi-Fi y Bluetooth.

Una vez obtenido el dato, este se recibe a través de un puerto COM del ordenador donde mediante un programa en Python se decodifica, se modifica el formato para su posterior lectura y se lanza a un bróker que almacenará la información deseada. Cada vez que se detecte movimiento, haya un cambio de temperatura o se genere una nueva entrada en uno de los sensores, este programa mandará de forma instantánea el valor actualizado al broker, para tener el último valor medido. Una vez allí, la skill de Alexa es la encargada de conectarse al bróker mediante MQTT, tomar la información del canal que se quiera leer y mediante un mensaje de voz devolver el dato que ha recogido el sensor. De esta forma se hace muy sencilla la interacción del ESP32 con Alexa, ya que de forma automática cada vez que se solicite, se puede obtener un dato en tiempo real del escenario en el que se encuentre la placa PCB, siendo este un despacho, una habitación de una vivienda, etc.

La aplicación resultante está pensada para ser utilizada por gente joven, acostumbrada a utilizar un dispositivo Alexa, pero por su sencillez a la hora de interactuar con la aplicación, cualquier persona puede hacer uso de ella. Además sirve como base para poder desarrollar multitud de aplicaciones y explorar todas las funcionalidades de Alexa que aún están por descubrir.

En vista a futuro, debido al gran nexo de unión entre Alexa, internet y el interés humano por seguir desarrollando nueva tecnología, todavía quedan muchos años para disfrutar de este tipo de aplicaciones. Con ello se quiere hacer hincapié en que puede llegar a mejorar significativamente la calidad de vida de la gente y así seguir adaptándose a los rápidos cambios que el mundo nos ofrece.

1. Introducción

El mundo contemporáneo evoluciona a ritmo vertiginoso. Grandes cambios han podido apreciarse en la manera de interactuar de las personas varias veces incluso dentro de la misma generación. Este avance es debido a diversas causas, siendo la más importante el avance científico/tecnológico y la inmensa cantidad de productos que llegan a las personas cada pocos años.

Las nuevas tecnologías de la información y el conocimiento (NTICs) han permitido estos cambios en la sociedad del mundo moderno, especialmente a partir de la difusión de importantes elementos como Internet. Los jóvenes han sido los primeros curiosos en abrazar las TICs, llevando su uso a un nuevo nivel de comunicación y relación entre ellos y el resto del mundo. Esto hace que el resto de colectivos se unan a la experiencia de probar estas tecnologías, de forma directa o indirecta a través de familiares y amigos.

Algo que hay que entender es la diferencia entre las TICs y las NTICs, ya que lo que ayer era una NTIC, hoy es una TIC. Es una realidad que todo cambia a un ritmo muy rápido, pero nuestra percepción de las cosas quizás nos haga dudar acerca de cuándo comenzó esta carrera de fondo. Aunque se piensa que la aparición de las NTICs tuvo lugar a mediados de los años 90 (Berrios y Buxarrais, 2005 [1]), lo cierto es que surgieron mucho antes. En Estados Unidos se empezó a hablar de ellas a partir de los 70, aunque en España se introdujeron oficialmente algo más tarde, en torno a principios de los años 80. Como prueba de ello, los primeros monográficos sobre las tecnologías de la juventud se publican en 1986, concretamente en el número 21 de la Revista de Estudios de Juventud.

Desde su aparición en la sociedad, el público joven ha sido en su mayoría usuaria de internet, como señala Zárraga en su libro de 1998 "Usos actuales de Internet por colectivos juveniles", un texto donde se deja una fuerte constancia de la relación entre ambos extremos. Otra publicación posterior por parte del Instituto de la Juventud (INJUVE), en 1999, analiza en profundidad las relaciones entre los jóvenes, las nuevas tecnologías, la telefonía, el entorno multimedia, el teletrabajo, etc. [2]

Si avanzamos un poco en el tiempo, más concretamente hasta el siglo XXI, múltiples artículos han ido apareciendo con el paso de los años en relación a como internet ha afectado a las vidas de nuestros jóvenes, aquí tenemos algunos ejemplos:

- En 2004 se publica "Jóvenes, relaciones familiares y tecnología de la información y de la comunicación"
- En 2006 en honor a las primeras redes sociales surge "Jóvenes y cultura Messenger"
- En 2011 "Adolescentes digitales"
- Un año después, en 2012, se publica "Juventud y educación ante las nuevas sociedades tecnológicas del siglo XXI". En el mismo año, el instituto de la Juventud en España publica "Informe Juventud en España"

Los dispositivos que se emplean hoy día son capaces de sincronizar gran cantidad de datos. Muchos de estos emplean aplicaciones escritas en lenguaje Python. En esta línea destacan actualmente la plataforma Alexa, el asistente virtual por voz ofrecido por el gigante Amazon. Alexa está disponible en varios idiomas y da la oportunidad de interactuar con el asistente facilitando tareas diarias de un individuo como pueden ser: informar de notificaciones de entregas de paquetes, consultar la previsión meteorológica o hacer preguntas de búsqueda de información puntual. Desde su lanzamiento al mercado en el año 2014, su uso cada vez es más habitual en los hogares de todas las familias.

Aprovechando la versatilidad y facilidad de uso de los dispositivos Alexa y poniendo nuestro foco en el público joven, el objetivo del presente trabajo ha consistido en generar una nueva plataforma de aprendizaje con una skill nueva desarrollada en Python. A partir del diseño de una placa PCB con cuatro sensores que miden valores de temperatura, sonido, intensidad de la luz y movimiento; junto con un microcontrolador en código Arduino, se pretende analizar datos en tiempo real. Esta skill puede interpretar estos datos y mostrarlos de forma simple al usuario de destino.

2. Descripción del trabajo

Base Teórica

El desarrollo del trabajo consiste en la lectura de múltiples valores en tiempo real mediante un microcontrolador programable que tomará datos del entorno en el que se encuentre el dispositivo. Tras esto, mediante el uso de Alexa se procederá a la lectura de los mismos siendo solicitados a través de comandos de voz.

ESP32

El ESP32 es una familia de microcontroladores diseñados por la marca china Espressif Systems. El ESP está compuesto por 3 tipos de productos. Los chips son los elementos de menor tamaño, donde se encuentra el cerebro del ESP. Después se encuentran los módulos, en ellos se encuentran los chips anteriores además de una memoria flash de 4 MB o una memoria SRAM de 8 MB. Lo más característico de estos dispositivos es que cuentan con una antena imprimida en la placa permitiendo la conexión por wifi y bluetooth. Los módulos facilitan la integración del ESP en otras placas, ya que incluyen los pines de entrada salida listos para soldaduras ASMD. Por último, las placas de desarrollo, son las más comunes ya que incluyen un módulo ya soldado y facilitan mucho su integración en prototipos. Esto es gracias a los pines diseñados para encajar en una protoboard, debido a que la distancia entre ellos es de 2,54 mm. Incluyen además su regulador de tensión y su controlador serie USB para poder dar alimentación y programarlos desde el mismo conector micro USB. El Wrover Kit, es el padre de los ESP y además de lo anteriormente mencionado, incluye ranura para microSD, una pantalla LCD, un led RGB y algunas cosas más.

Arduino y el ESP32 no pertenecen a la misma familia de microcontroladores. A pesar de las diferencias que podemos encontrar entre un microcontrolador de Arduino y un ESP, estos últimos fueron diseñados para adaptarse casi por completo al software de Arduino, por lo que la programación se realiza en el mismo lenguaje y entorno que un Arduino normal. A continuación se pueden apreciar las características físicas de ambos dispositivos y variantes de la misma familia, siendo notablemente superior en muchos aspectos el ESP.

Los más característicos son el número de capacidades incorporadas, la cantidad de memoria, la capacidad de procesamiento, precio, etc. [3]

	ESP8266 NodeMCU V2	NodeMCU V3	ESP32 NodeMCU	ESP8266 WeMos D1 Mini	Arduino NANO 3	Arduino UNO R3	Arduino UNO WIFI R2	Arduino Mega
Microcontrolador	ESP8266	ESP8266	ESP32	ESP8266	ATmega328p	ATmega328p	ATmega4809	ATmega2560
Tensión de funcionamiento	3,3V	3,3V	3. 3V	3,3V	5V	5V	5V	5V
Fuente de alimentación	7V – 12V	7V – 12V	7V – 12V	4V – 6V	7V – 12V	7V – 12V	7V – 12V	7V – 12V
Consumo de corriente	15 µA – 400 mA	15 µA – 400 mA	20 mA – 240 mA		19 mA – 180 mA	45 mA – 80 mA	50 mA – 150 mA	50 mA – 200 mA
Consumo de corriente Sueño profundo	0,5 µA	0. 5 µA	5 µA		23 µA (con ajustes especiales)	35 mA	35 mA	500 µA
Pines de E/S digitales	11 o 13	16	36	11	14	14	14	54
Pines de E/S digitales con PWM	11 o 13	16	36	11	6	6	5	15
Pines de entrada analógica	1	1	15	1	8	6	8	16
SPI/I2C/I2S/UART	2/1/2/2	2/1/2/2	4/2/2/2	1/1/1/1	1/1/1/1	1/1/1/1	1/1/1/1	1/1/1/4
Corriente CC por pin de E/S	12 mA	12 mA	20 mA		40 mA	40 mA	40 mA	20 mA
Corriente de CC para 3. 3V Pin			40 mA		50 mA	150 mA	150 mA	150 mA
Memoria flash	4 MB	4 MB	4 MB	4 MB	32 KB	32 KB	48 KB	256 KB
SRAM	n. A	64 KB	520 KB		2 KB	2 KB	6 KB	8 KB
EEPROM	512 bytes	512 bytes	–		1024 bytes	1024 bytes	256 bytes	4096 bytes
Velocidad de reloj	52 MHz	80 MHz	80 MHz / 160 MHz	80 MHz / 160 MHz	16 MHz	16 MHz	16 MHz	16 MHz
Longitud	48 mm	58 mm	52 mm	34	45 mm	69 mm	69 mm	102 mm
Ancho	26 mm	31 mm	31 mm	26	18 mm	53 mm	53 mm	53 mm
WIFI	sí	sí	sí	sí	no	no	sí	no
Bluetooth	no	no	sí	no	no	no	no	no
Sensor táctil	no	no	10	no	no	no	no	no
CAN	no	no	sí	no	no	no	no	no
Interfaz MAC Ethernet	no	no	sí	no	no	no	no	no
Sensor de temperatura	no	no	sí	no	no	no	no	no
Sensor de efecto de pared	no	no	sí	no	no	no	no	no
Toma de corriente	no	no	no	no	no	sí	sí	sí
conexión USB	sí	sí	sí	sí	sí	sí	sí	sí
Precio	6$	6$. 50	$11	$6	$22	$22	$44,90	$38,50

Ilustración 1 Características Arduino vs ESP

Para la asignación de variables globales del programa se ha de tener en cuenta el PINOUT del microcontrolador [4] , utilizando en algunos casos puertos específicos de un solo uso, ya sea solo de entrada, solo de salida o ambas. Según las especificaciones deseadas para este proyecto, en su mayoría se emplean pines de entrada/salida, ya que salvo casos especiales no se necesitarían de otro tipo, como analógicas o de memoria flash. Existen variaciones respecto a la colocación de los pines en función del modelo del ESP32. En este caso, el esquema o PINOUT del modelo utilizado, ESP32 dev KIT v.2, es el siguiente:

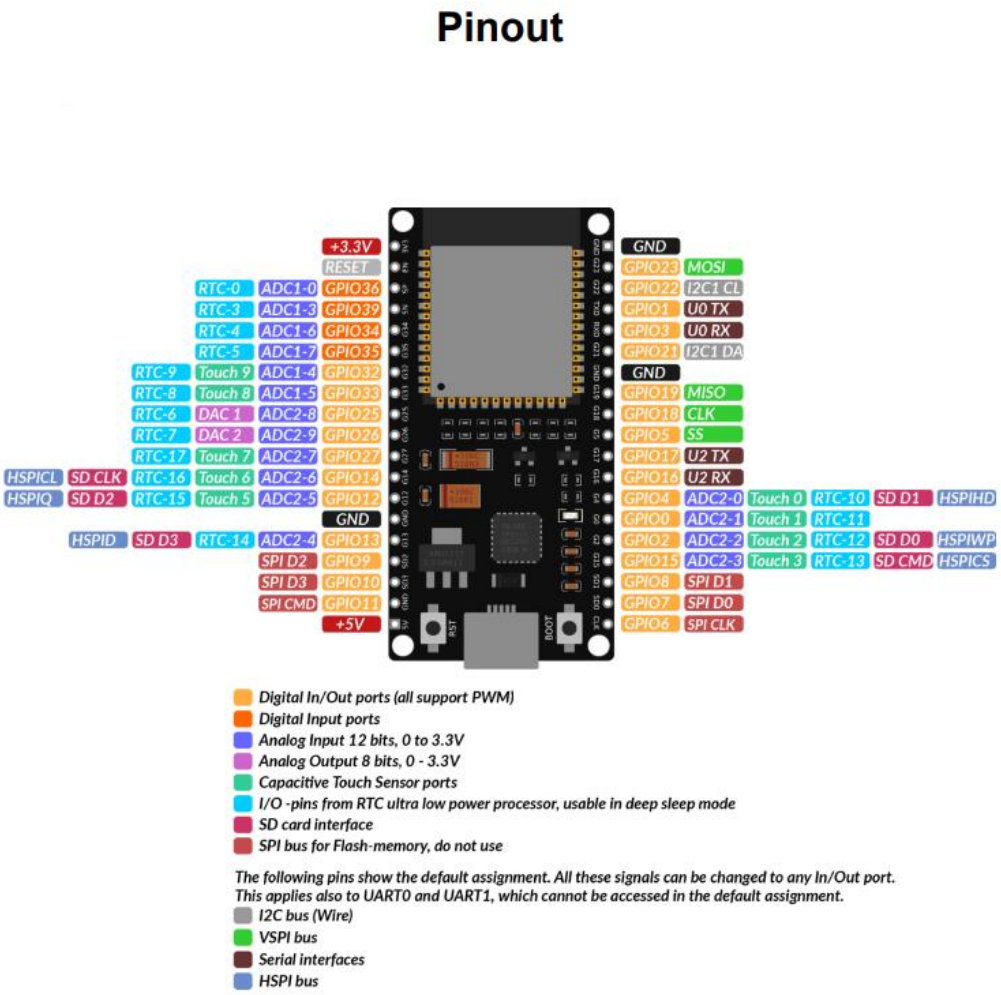

Ilustración 2 Pinout ESP32 dev Kit V2

Una vez conocido el diagrama del microcontrolador se definen las variables necesarias para el uso de los sensores en función del número de pin al que pertenece.

Mqtt

MQTT es el protocolo de mensajería más utilizado para el internet de las cosas (IoT). Las siglas significan transporte de telemetría MQ (Message Queuing). Este protocolo es un conjunto de reglas que describe como los dispositivos IoT pueden publicar y suscribirse a datos a través de Internet. Se utiliza principalmente para el intercambio de información entre dispositivos IoT y dispositivos IIoT (dispositivos de IoT industrial), como son dispositivos integrados, sensores, etc. Principalmente basa su uso en eventos y conecta dispositivos mediante el patrón de publicación y suscripción. El remitente y el receptor se comunican a través de Canales independientes entre sí con jerarquía de árbol y su conexión la gestiona el intermediario MQTT. Esto se hace ya que MQTT recibe como parámetro el canal que se desea escuchar y distribuye correctamente la información a los suscriptores.

En cuanto a nivel de seguridad, MQTT está diseñado para permitir una comunicación muy segura. Como protocolo de la capa de aplicación, posee amplias posibilidades de autorización y autenticación de dispositivos. El protocolo de transporte TCP/IP del que dispone, puede agregar seguridad adicional a través del cifrado TLS. TLS significa Transport Layer Security o Seguridad en la capa de transporte y es la versión actualizada y mejorada de SSL (Secure Socket Layer).

Es un protocolo bastante ligero, es decir, bajo consumo de CPU y de energía del aparato que lo esté utilizando. [5]

Dentro del servidor a utilizar, existen múltiples servidores en el mercado. Para el desarrollo del proyecto se ha decidido emplear Eclipse Mosquitto, servidor de uso libre que funciona de forma muy rápida tanto en Windows como en Linux. Al servidor MQTT se le denomina broker, ya que es un broker de mensajes, pudiéndose conectar a él desde teléfonos móviles, ordenadores, incluso neveras, tostadoras o cualquier dispositivo que disponga de un chip.

Como se ha comentado antes, los servidores MQTT están compuestos de canales o Topics, aquí tenemos una ilustración del funcionamiento básico del broker.

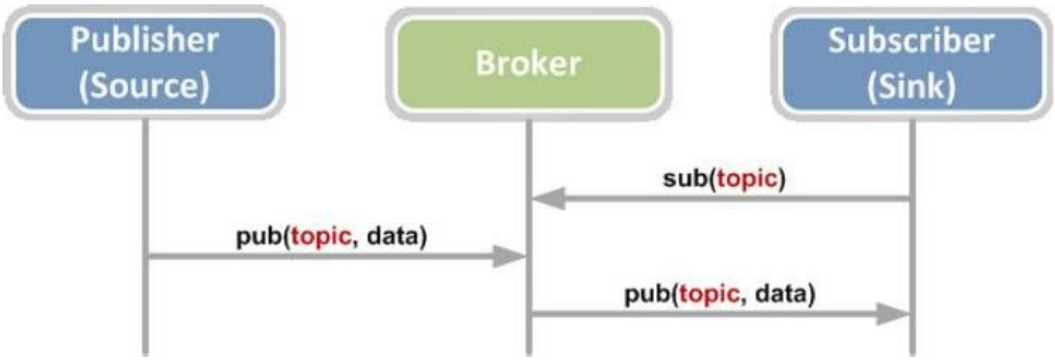

Ilustración 3 Publicador y Suscriptor Broker MQTT

Como muestra la imagen, el Publisher u origen hace una llamada al método pub, pasando como parámetros un "Topic" o canal y una "data" o información que se quiera enviar. Esto llega al broker, el cual almacenará la información recibida por el emisor de forma permanente o no, dependiendo del resto de parámetros que se quieran añadir. Después, el suscriptor realiza una llamada a la función sub, indicando el Topic al cual se quiere subscribir. El broker recibe la petición y envía un mensaje de vuelta al suscriptor a través de ese Topic con la información almacenada previamente.

Alexa

Alexa es un asistente virtual controlado por voz y creado por Amazon. Fue lanzado al mercado en 2014 junto con una serie de altavoces inteligentes marca Echo y su nombre fue elegido en honor a la biblioteca de Alejandría (el centro del saber del mundo antiguo), además de contener una x en su nombre para hacer que el dispositivo reconozca más fácilmente su nombre.

Como el resto de asistentes virtuales, su funcionamiento es muy similar a los más conocidos hoy en día, como son Siri, Cortana o Google Assistant. Se empieza diciendo su nombre, para que el dispositivo comience a escuchar a la espera de un comando de voz. Si el dispositivo consigue comprender lo que se ha dicho, dará una respuesta acorde a lo que se ha preguntado, en caso contrario responderá con un: -No lo sé, o -Lo siento, no te he escuchado, ¿puedes repetir?.

En un principio, Alexa se pensó para estar vinculado únicamente a los altavoces inteligentes, pero al poco tiempo se abrió su SDK (Software Development Kit) para que múltiples desarrolladores pudiesen empezar a trabajar libremente con él. Es así como ha llegado a formar parte de gran cantidad de dispositivos, desde un reloj hasta un microondas.

Las diversas funciones de Alexa distinguen dos funciones principales. Los comandos de voz integrados en el dispositivo a través de los cuales puedes realizar una amplia variedad de peticiones y las Skills o complementos, que puedes instalar para añadir aún más funcionalidad.

Respecto a los comandos por voz, se puede hacer una enorme cantidad de preguntas a Alexa, como son la meteorología, crear temporizadores, buscar información concreta relativa a personas y productos, además de incorporar funciones de compra por voz y seguimiento del pedido.

Las Alexa Skills son aplicaciones que son desarrolladas por terceros y con ellas se pueden ampliar las funciones del dispositivo exponencialmente. Por ejemplo, la skill Akinator es un juego desarrollado para adivinar una persona o personaje, tanto real como de ficción, a través de una serie de preguntas al usuario (Es cantante, Vive en España, tiene hijos, es un personaje de Marvel, etc.) hasta que finalmente, tras 10-20 preguntas consigue resolverlo con un alto porcentaje de acierto. De la misma forma en este proyecto se

desarrolla una skill con unas funcionalidades únicas y específicas para conseguir el objetivo deseado.

Ilustración 4 Dispositivos Alexa

Para comprender el uso de las librerías que se utilizan en el desarrollo de la skill de Alexa, antes hay que comprender que es un ASK (Alexa Skill Kit). Un ASK es un conjunto de herramientas, documentación, muestras de código y API de auto servicio con el que se pueden añadir Skills a Alexa de forma rápida y sencilla.

QoS

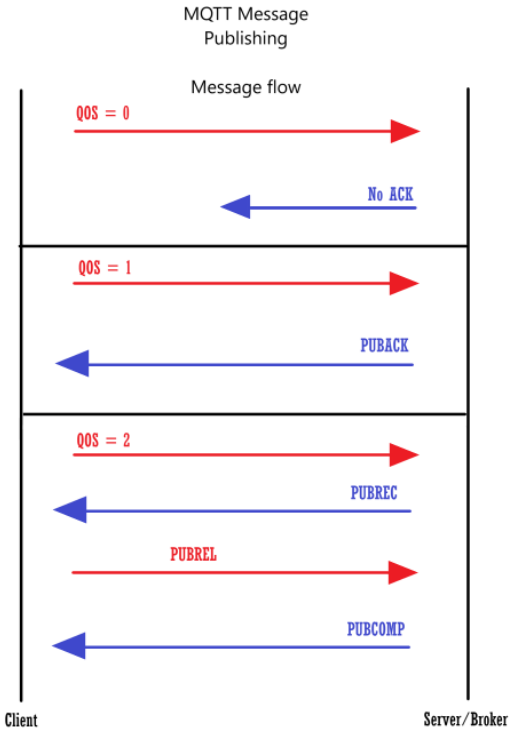

Ilustración 5 Mensajes MQTT en función de la QoS

La calidad del servicio (QoS) es un acuerdo entre el remitente de un mensaje y su receptor. Existen 3 niveles de QoS en MQTT, nivel 0, 1 y 2. [6]

- QoS = 0 es el nivel mínimo de calidad posible. No garantiza la entrega del mensaje, ya que como podemos apreciar en la ilustración, no se recibe acuse de recibo o ACK. Este nivel se denomina Activar y Olvidar y proporciona la misma garantía que el protocolo TCP subyacente.
- QoS = 1 es el nivel intermedio de calidad. Este nivel garantiza que el mensaje se entregue al menos una vez al receptor. El remitente almacena el mensaje hasta que recibe un PUBACK del receptor como ACK, además de permitir que un mensaje se entregue y se reciba varias veces.
- QoS = 2 es el nivel más alto del servicio en MQTT. Este nivel garantiza que cada mensaje sea recibido solo una vez por los destinatarios previstos. QoS 2 es el nivel de calidad de servicio más seguro y lento. La garantía de entrega del mensaje es proporcionada por al menos dos flujos de solicitud/respuesta entre el emisor y el receptor. Cuando el primero recibe un mensaje PUBLISH, el segundo responde con un paquete PUBREC. Si el paquete se ha recibido correctamente, el remitente responde con un paquete PUBREL y en caso contrario se seguirán enviando solicitudes publish hasta recibir el paquete PUBREC. Finalmente, el receptor enviará un paquete PUBCOMP al emisor. Durante la espera del mensaje PUBREL, el receptor almacena el identificador del mensaje original y cuando todo acaba, lo libera dejándolo disponible para su reutilización.

Descripción Experimental

Dentro de este apartado se recoge toda la parte de diseño y programación del trabajo presentado, dividiéndose en 4 partes principales: El montaje o Hardware, la programación del ESP32 o Software, el diseño MQTT y la Skill de Alexa.

Hardware

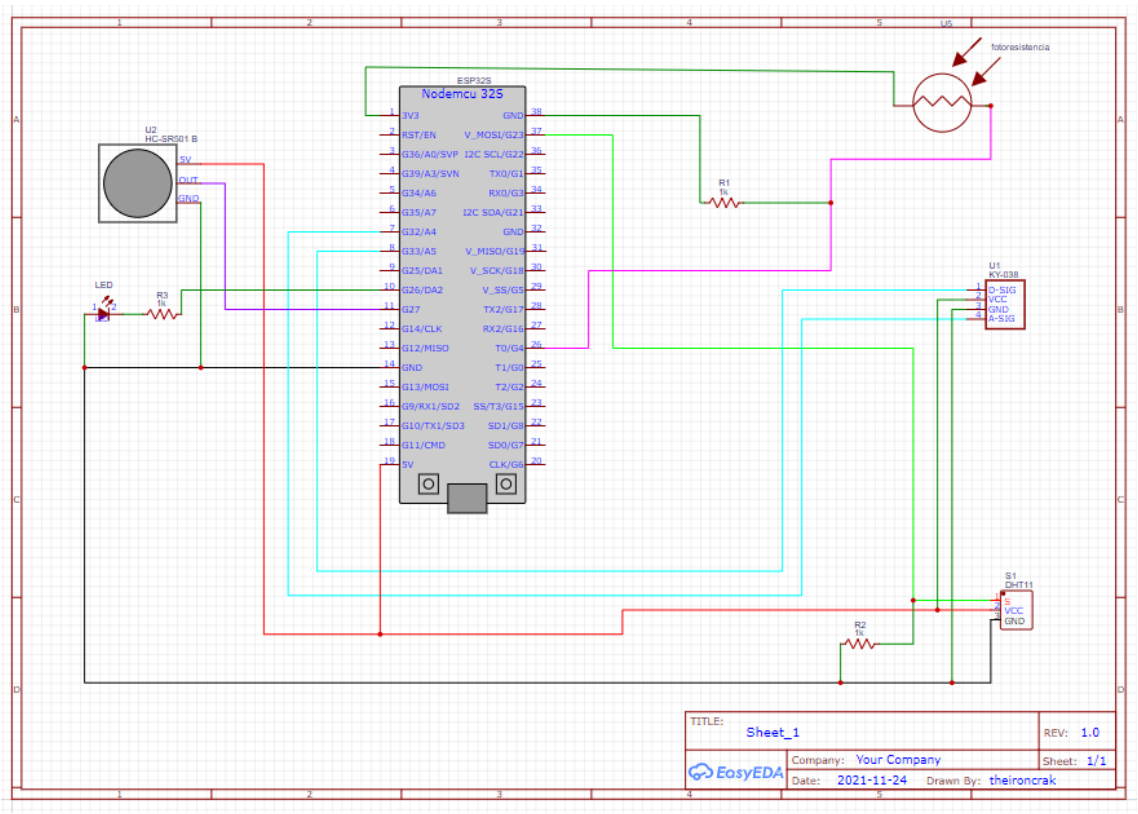

Ilustración 6 Diseño Circuito EasyEda

Para el diseño de la placa PCB, se ha utilizado un software de uso libre llamado EasyEDA. Este software permite crear un diseño real en un entorno simulado, donde cada componente y sensor se corresponde con uno físico. El diseño consta de 4 sensores, temperatura, sonido, intensidad y movimiento, además de varias resistencias para regular la potencia de entrada y salida y un led correspondiente al sensor de sonido.

El siguiente paso sería crear las conexiones sobre la placa PCB para mandarla a diseñar. Esta placa en concreto consta de dos capas, que se corresponden con las líneas rojas y azules de la siguiente imagen. Inicialmente se crearán unas conexiones automáticamente pero habrá que modificarlas para evitar cortocircuitos en la placa.

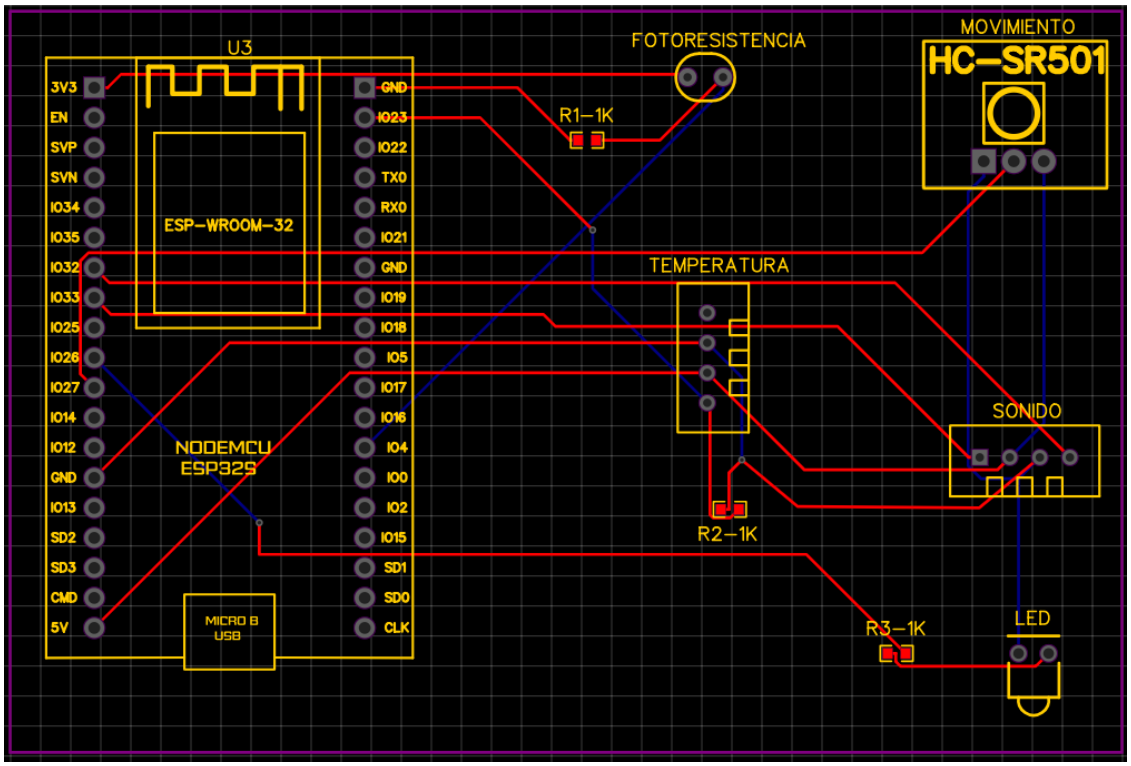

Ilustración 7 Diagrama a PCB

El resultado final de la PCB es el siguiente:

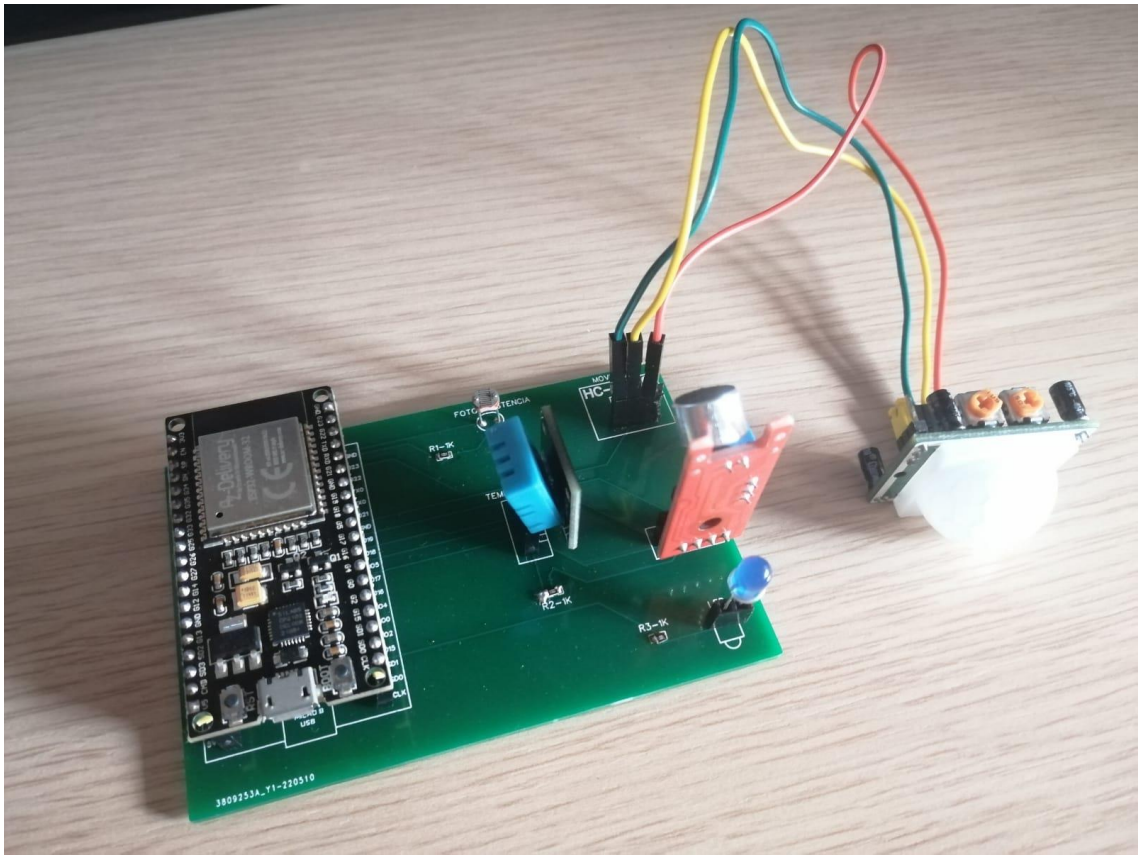

Ilustración 8 Diseño final PCB y Sensores

Software

En el proyecto realizado, se ha decidido utilizar 4 sensores para la toma y muestra de datos. Estos son, el sensor de Movimiento, el de Temperatura, el de Sonido y el de Intensidad.

Sensor de Movimiento

El sensor de movimiento es un sensor encargado de detectar movimiento por proximidad a unos cuantos metros de distancia. Consta de 3 patillas, una para VCC, otra para GND y la última para los datos.

Ilustración 9 Sensor de Movimiento

Las características físicas del modelo HC-SR501 son las siguientes:

- Hace uso de la tecnología infrarroja. Su funcionamiento es automático, solo necesita estar alimentado. Con 2 potenciómetros para un ajuste de la sensibilidad y la duración de conmutación. Es adecuado para una gran cantidad de funciones como detección automática de luz para pisos, baños, sótanos, almacén, garaje...
- Gracias a sus pequeñas dimensiones, se puede utilizar en una gran cantidad de equipos electrónicos de detección automática, sobre todo para productos controlados automáticamente con baterías. Ideal para proyectos Raspberry, PiC, AVR, STM32 Sensor infrarrojo con circuito de control. Sensibilidad y tiempo ajustables. Lente de 19 zonas, ángulo < 100º. Voltaje de funcionamiento: 4.5V-20V. Consumo en estado inactivo: <60uA. Voltaje de salida: 3.3V TTL output.
- Intervalo de tiempo: 0,5-200s (ajustable, por defecto 5s ±3%). Tiempo de bloque: 2.5s (por defecto). Rango de detección: 3 m a 7 m, ajustable mediante trimmer (Sx). Temperatura de trabajo:-20ºC a 70°C. Dimensiones: Sobre 3,2 x 2,4 x 1,8cm. Peso: Sobre 8 g.

Para el caso práctico, se ha decidido mostrar la última vez que se detectó movimiento mediante el acceso a la hora real, tomada de internet, en el que se realizó dicho movimiento cerca de la placa. Para hacer uso del sensor no se necesitarían librerías específicas, pero si para el trabajo propuesto:

- WiFi.h: Permite que una placa Arduino se conecte a internet. [7]
- Time.h: A través de esta librería se obtiene la fecha y hora actual.

```
• #include <WiFi.h> [8]
• #include "time.h"
```

Se indica el servidor NTP que se desea utilizar. En este caso, se hará uso de "pool.ntp.org", ya que es ideal para este tipo de requerimientos. Además, este servidor selecciona automáticamente los servidores de tiempo que se encuentran cerca geográficamente. No obstante, se puede elegir el servidor concreto si así se desea (europa.pool.ntp.org). Se define por tanto el "gtmOffset_sec", calculado mediante UTC +1.00 * 60 * 60 = 3600 y el "daylightOffset_sec" a 3600 también, ya que se aplica el horario de verano. [9]

A continuación, se definen como variables el nombre de la red Wifi (o SSID) y la contraseña para poder conectarse aunque por motivos de seguridad, se han eliminado de este documento. La variable "timeStringBuff" va a almacenar el resultado de la función printLocalTime, que será un String con el formato: El 25 del 05 de 1997 a las 03 horas 46 minutos y 30 segundos". Se ha transformado la función original para devolver el resultado como parámetro en la variable anteriormente mencionada por simplicidad en el desarrollo del código.

Para ello y como se muestra en la imagen posterior, en el paso previo al cálculo del tiempo, se han establecido unos parámetros base para la hora local en función de la franja horaria en la que se encuentre el dispositivo.

```
const char* ntpServer = "pool.ntp.org";
const long  gmtOffset_sec = 3600;
const int   daylightOffset_sec = 3600;
int sensor_pir = 27;

const char* ssid = "Nombre WiFi";
const char* pass = "Contraseña";

String aux_time = "";

char timeStringBuff[70];
String aux_mov = "hola";

char* printLocalTime()
```

```
{
  struct tm timeinfo;
  if(!getLocalTime(&timeinfo))
  {
    Serial.println("Failed to obtain time");
   //return;
  }else{

  strftime(timeStringBuff, sizeof(timeStringBuff), "El %d del %m de %Y
a las %H horas %M minutos y %S segundos", &timeinfo);
  //print like "const char*"
  //Serial.println("mov:"+ String(timeStringBuff));

  }return timeStringBuff;
}
```

La función Setup es la primera función en ejecutarse dentro de un programa en Arduino, es donde se "setean" las funciones que llevará cabo el microcontrolador. Aquí es donde se establecen algunos criterios que requieren una ejecución única.

Lo primero que hará será conectarse a la red Wi-Fi a través de los credenciales definidos anteriormente, dejando un tiempo de delay para que se pueda conectar sin problemas. En caso de no haberse conectado aun, se añade medio segundo más por cada intento hasta conseguirlo. Con "pinMode" se asigna el pin 27 del sensor definido al inicio y como tipo de entrada de datos "input_pullup", debido a las características físicas del dispositivo. Se hace la llamada a la función para configurar el tiempo, devolver el valor y una vez se obtenga, se procede a desconectarse de la red wifi. El valor del tiempo ya queda establecido y guardado a nivel local.

```
void setup() {
  WiFi.begin(ssid, pass);
  delay(2000);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
  pinMode(sensor_pir, INPUT_PULLUP);

  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
  printLocalTime();
  //disconnect WiFi as it's no longer needed
  WiFi.disconnect(true);
  WiFi.mode(WIFI_OFF);
}
```

Dentro de la función Main, se imprime el valor del momento en el que se ha detectado el movimiento, siendo como mínimo en intervalos de 1 segundo entre valor y valor, evitando crear bucles y congestión a la hora de imprimir los datos. Al terminar de recoger la información, se devuelve el valor del sensor a 0 para poder volver a medir el siguiente.

```
void loop() {
    PIR = digitalRead(sensor_pir);
    String mov = String(printLocalTime());
    if (PIR == 1 && aux_mov!= mov){
       aux_mov = mov;
       Serial.println("mov:"+mov);
       PIR = 0;
    }
}
```

Sensor de Temperatura

El sensor de temperatura, como dice su nombre, mide la temperatura del lugar donde se encuentre el sensor, en grados centígrados. Este sensor tiene 3 pines, uno para VCC, otro para GND y otro para datos.

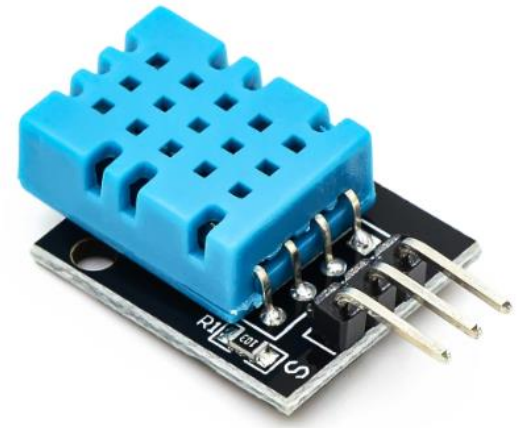

Ilustración 10 Sensor de Temperatura

El sensor digital de temperatura y humedad DHT11 es una salida de señal digital calibrada con un complejo de sensores de temperatura y humedad. Aplica la técnica de adquisición de datos del módulo digital específico y la tecnología de sensores de temperatura y humedad, para garantizar que el producto tenga una alta fiabilidad y una excelente estabilidad a largo plazo.

El sensor incluye un elemento sensible a la humedad de tipo resistivo y un elemento NTC, y se conecta un microordenador de un solo chip de 8 bits de alto rendimiento. Por lo tanto, es un producto con excelente calidad, respuesta rápida y una fuerte capacidad anti interferencias.

Consta de una interfaz serial de un solo cable, para que la integración del sistema se vuelva simple y rápida. Volumen Ultra pequeño y bajo consumo de energía La distancia de transmisión de la señal puede alcanzar los 20 metros, lo que hace que la aplicación de varios tipos, incluso en las aplicaciones más exigentes, sea la mejor opción.

Parámetro del producto:

1. Voltaje de la fuente de alimentación: 5V
2. Temperatura: 0-50
3. Rango de humedad: 20-90% RH

Para utilizar este sensor se necesita una librería específica para el DHT.

```
#include "DHT.h"
```

Se establece el tipo de DHT que se va a utilizar para el sensor de temperatura, DHT11 (existen variaciones como el DHT22) y se asigna el pin 23 para la entrada de datos de dicho sensor.

```
#define DHTPIN 23
#define DHTTYPE DHT11
```

Se declaran 4 variables para realizar cálculos posteriormente.

```
float temp = 0.0;
float aux_temp = 0.0;
unsigned long tiempo1 = 0;
unsigned long tiempo2 = 0;
```

Se crea la función medir temperatura, la cual almacena en "temperatura" el valor leído por el sensor y lo devuelve como parámetro.

```
float medir_temperatura(){
  float temperatura = dht.readTemperature();
  return temperatura;
}
```

Por último, en la función principal, se hace uso de "millis". Esto devuelve el número de milisegundos transcurridos desde que el microcontrolador ESP32 comienza a ejecutar el programa actual. Tiene un valor máximo de desbordamiento, aproximadamente transcurridos 50 días, valor inalcanzable por el programa actual, ya que el contador será reiniciado tras tomar una medida. [10]

Se toma el valor de la temperatura en la variable "temp" y se compara que este valor no sea igual al anterior medido, en cuyo caso no se mostrará. Además, aunque el valor de la temperatura cambie constantemente y de forma muy rápida, no se retornará un valor hasta pasados 4 segundos.

```
void loop() {
    tiempo2 = millis();
    temp = medir_temperatura();

    if (tiempo2 > (tiempo1+4000) && aux_temp!=temp){
      aux_temp = temp;
      Serial.println("temp:"+ String(temp));
      tiempo1 = millis();
    }

}
```

Sensor de Sonido

El sensor de sonido KY-038 es un sensor encargado de medir el sonido a través de la salida digital del dispositivo.

Ilustración 11 Sensor de Sonido

Consta de un led interno, el cual se activa cuando la salida digital se activa. Además incorpora un preset multivuelta, regulable para calibrar el valor que se desea medir (valor de resistencia). Con este preset lo que se establece es un valor umbral, el cual marca a partir de qué valor se va a considerar como "alto" el sonido captado.

Módulo de características:

1. Hay un orificio para Tornillo de montaje de 3mm
2. Fuente de alimentación de 5 VCC
3. Con salida analógica
4. Hay un giro de salida de nivel umbral
5. Micrófono de alta sensibilidad y alta sensibilidad.
6. Una luz indicadora de potencia
7. La salida del comparador es una luz

Para el caso práctico, cada vez que se detecte un sonido corto (ej.: palmada), se encenderá un led colocado en la placa. De la misma forma, este se mantendrá encendido hasta que se detecte una segunda palmada para volver a apagarlo, y así sucesivamente.

Los pines utilizados son, el 26 para el led, el 33 para la salida digital del sensor y una variable booleana para comprobar el estado del led.

```
int led = 26;
int sound_digital = 33;
bool estado;
```

En la parte de Setup, se declara la toma de datos como entrada y el led como salida.

```
void setup() {
  pinMode(led, OUTPUT);
  pinMode(sound_digital, INPUT);
}
```

En la función Main, la variable "X" almacenará el valor tomado mediante el sensor. Después, para comprobar si se ha detectado sonido o no, se compara la variable con HIGH, es decir, si el valor es "1", significando que hay sonido. Dentro de ese condicional, se comprueba si el estado es apagado o encendido para realizar la acción contraria cuando se detecte sonido, si el led ya estaba apagado, ahora se encenderá al dar una palmada y viceversa. Se deja un delay de 75 milisegundos para que reaccione a sonidos cortos. Finalmente se imprime el valor del led por el monitor serie de Arduino.

```
void loop() {
    int X = digitalRead(sound_digital);
    if (X == HIGH){
      if (estado == false){
        digitalWrite (led, HIGH);
        Serial.println("son:encendido");
        delay(75);
        estado = true;
      }else{
        digitalWrite (led, LOW);
        Serial.println("son:apagado");
        delay(75);
        estado = false;
      }
    }

}
```

Sensor de Intensidad

El sensor de intensidad o fotorresistencia mide la intensidad de la luz en lúmenes por metro cuadrado.

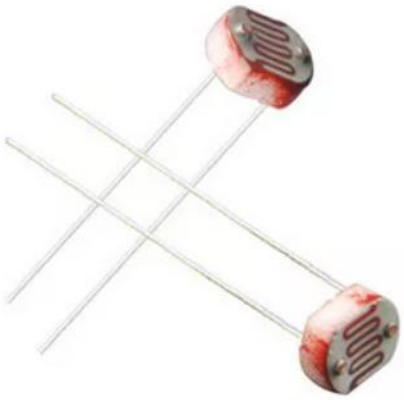

Ilustración 12 Sensor de Intensidad

Es un sensor con únicamente dos patas, una va al pin 3v3 del microcontrolador y la otra al pin 4. El valor de la intensidad oscila entre 0 y 4095 aproximadamente, no pudiendo superar este último valor, alcanzable con un flash de un teléfono móvil.

Se utiliza el pin 4 para leer los datos recibidos y el pin 3v3 para subministrar voltaje a la fotorresistencia.

```
#define LDR 4
```

Se establecen variables auxiliares para cálculos en el programa principal.

```
int luz = 0;
int aux_luz = 0;
unsigned long tiempo3 = 0;
unsigned long tiempo4 = 0;
```

Se define una función “medir_luz()”, esta toma el valor analógico de la fotorresistencia y lo devuelve como parámetro para su posterior uso.

```
int medir_luz(){
  int LDR_val = analogRead(LDR);
  return LDR_val;
}
```

Este último método hace también uso de “millis” para que, al igual que el sensor de temperatura se tomen medidas cada un tiempo determinado. En este caso, el sensor leerá un valor cada dos segundos mínimo, o hasta que aparezca un nuevo valor de intensidad en los dos segundos siguientes y así sucesivamente hasta que cambie.

```
void loop() {
    tiempo4 = millis();
    luz = medir_luz();


    if (tiempo4 > (tiempo3+2000) && aux_luz!=luz){
       aux_luz = luz;
       Serial.println("luz:"+String(luz));
       tiempo3 = millis();
    }
}
```

Código MQTT

Para empezar, se definen una serie de librerías necesarias para la ejecución del programa.

- Serial: Esta librería realiza la comunicación entre Python y el ESP32.
- Time: Librería necesaria para poner la ejecución del programa en espera
- Paho: La más importante, encargada de realizar todas las funciones relacionadas con MQTT. Con ella se instancia un cliente para la llamada a estas funciones. [5], [11]

```
import serial
import time
import paho.mqtt.client as paho
```

A continuación se define el servidor o broker en el que van a publicar los datos recibidos del ESP32. Como comentario, se indica la dirección IP del servidor, la conexión se puede realizar tanto a través de la Url como de la dirección IP.

```
#broker = "192.168.1.47"
broker="test.mosquitto.org"
port=1883
```

Se crea una instancia u objeto del tipo serial. A través de la variable "serialEsp32" se establece el puerto "COM5", a través del cual se va a recibir la información de la placa y la velocidad de comunicación serial a 115200 bits por segundo (baudios). Estos valores han de corresponder con los fijados en el programa de Arduino.

```
serialEsp32 = serial.Serial("COM5", 115200)
```

Tras la creación del objeto se añade un tiempo de espera de 2 segundos para que se establezca la conexión.

```
time.sleep(2)
```

Se define la función "on_publish", encargada de publicar al canal del servidor MQTT. Esta función es una función callback, es decir, no se va a ejecutar antes de que una tarea se complete, si no que se ejecutará justo después de que la tarea se haya completado. Esto ayuda a la realización de tareas asíncronas, como es el caso que se expone, ayudando a su vez a evitar errores y problemas.

```
def on_publish(client,userdata,result):             #create function
for callback
    #print("data published \n")
    pass
```

Esta es la función principal del programa, en ella se define un bucle infinito encargado de controlar la recepción de datos y el envió instantáneo de estos al servidor MQTT. Lo primero, se define un objeto cliente, que va a ser el que publique datos y se suscriba a los canales. Para iniciar la conexión con el servidor, se hará un "cielnt.connect" especificando el broker y puerto. Se define una variable llamada "cad", donde se recibirán todos los mensajes que mande el microcontrolador. El método readline().decode('ascii') descifrará los datos recibidos de la placa PCB. El siguiente fragmento de código cogerá la cadena recibida y traducida anteriormente y buscará por dos puntos pos=cad.index(":"). Una vez hecho eso se dividirá la cadena en dos y en la variable "label" se almacenará la primera parte de la cadena para poder identificar el sensor del que se está recibiendo el valor label=cad[:pos]. La segunda parte de la cadena quedará almacenada en la variable "value" guardando la información relevante y limpia que el sensor está captando value=cad[pos+1:].

La última parte del programa se encarga de distinguir en función del sensor del que se está recibiendo la información para ser publicado en un Topic u otro.

```
while True:
    #time.sleep(5)
    client= paho.Client("control1")                                    #create
client object
    client.connect(broker,port)
    cad = serialEsp32.readline().decode('ascii')

    pos=cad.index(":")
    label=cad[:pos]
    value=cad[pos+1:]
    if label == 'temp':
        client.on_publish = on_publish
        ret = client.publish("ALSW/temperatura", value, qos=1, retain
= True)
        print(cad)
        print("*********")
    if label == 'luz':
        client.on_publish = on_publish
        ret = client.publish("ALSW/intensidad", value, qos=1, retain =
True)
        print(cad)
        print("*********")
    if label == 'mov':
        client.on_publish = on_publish
        ret = client.publish("ALSW/movimiento", value, qos=1, retain =
True)
        print(cad)
        print("*********")
    if label == 'son':
```

```
        client.on_publish = on_publish
        ret = client.publish("ALSW/sonido", value, qos=1, retain =
True)
        print(cad)
        print("*********")
```

Una vez seleccionado el canal correspondiente, se hará una llamada a la función client.on_publish en la que se asigna la función al callback. La forma de publicar el mensaje es mediante la asignación a una variable, se podría entender como una variable de retorno. Para completar la llamada al método publish, se añadirán varios parámetros más.

El primero es el topic o canal, se ha elegido la ruta ALSW/temperatura para publicar el valor de temperatura. Después se cargará el mensaje que se quiera publicar. El parámetro retain = True hará que el mensaje se publique en el servidor de forma permanente (varias horas o días), de forma que cuando se quiera acceder al dato desde el dispositivo Alexa siempre habrá una publicación para que esta pueda leerla. El valor de QoS se establecerá a 1, ya que proporciona un mínimo de la calidad del nivel 2 pero devolviendo un ACK, a diferencia del QoS 0, que confirma que el mensaje ha sido recibido. Además, es más ligero y evita el envío de tantos mensajes entre emisor y receptor. Aun así, en función de la velocidad de reacción que necesite el servicio, la opción de QoS nivel 2 sigue siendo bastante factible.

```
if label == 'temp':
        client.on_publish = on_publish
        ret = client.publish("ALSW/temperatura", value, qos=1, retain
= True)
        print(cad)
        print("*********")
```

Skill de Alexa

Para el diseño de la Skill de Alexa, se ha hecho uso de 4 archivos de configuración y desarrollo necesarios para el correcto funcionamiento de la misma, estos son Requirements.txt, Utils.py, Test.py y Lambda_function.py .

Requirements.txt

Este archivo contiene las librerías necesarias para utilizar el servicio MQTT y las funciones de Alexa.

- Boto3: Incorpora tareas en espera, que sondean automáticamente cambios en los estados predefinidos de los recursos de AWS (Amazon Web Service). [12]
- Ask-sdk: Como se ha explicado anteriormente, esta librería proporciona un kit de desarrollo software a través del cual podemos hacer uso de la skill de Alexa.
- Paho-mqtt: Librería necesaria para las funciones de suscripción al servidor MQTT por parte del archivo Test.py

```
boto3==1.9.216
ask-sdk-core==1.11.0
paho-mqtt==1.6.1
```

Utils.py

Esta clase se genera automáticamente cuando se crea una nueva Skill. Se crea un objeto por defecto que se almacena en la variable "*response*" al instanciar el servicio de S3. Este servicio es proporcionado por el paquete boto3 que viene con el SDK.

```
import logging
import os
import boto3
from botocore.exceptions import ClientError


def create_presigned_url(object_name):
    """Generate a presigned URL to share an S3 object with a capped
expiration of 60 seconds

    :param object_name: string
    :return: Presigned URL as string. If error, returns None.
    """
    s3_client = boto3.client('s3',
```

```
region_name=os.environ.get('S3_PERSISTENCE_REGION'),

config=boto3.session.Config(signature_version='s3v4',s3={'addressing_s
tyle': 'path'}))
    try:
        bucket_name = os.environ.get('S3_PERSISTENCE_BUCKET')
        response = s3_client.generate_presigned_url('get_object',
                                                    Params={'Bucket':
bucket_name,
                                                            'Key':
object_name},
                                                    ExpiresIn=60*1)
    except ClientError as e:
        logging.error(e)
        return None

    # The response contains the presigned URL
    return response
```

Test.py

Las librerías que se utilizan son la de sleep, para detener la ejecución del programa y la de paho para el resto de servicios MQTT.

```
from time import sleep
import paho.mqtt.client as mqtt
```

La función on_message es una función callback en la que se define una variable "data" global, que almacenará el valor leído del broker.

```
def on_message(client, userdata, msg):
    global data
    data = msg.payload.decode("utf-8")
```

Se crea una función on_connect por cada sensor que se quiera utilizar, en la cual se define el topic al que se va a subscribir el objeto cliente. Al igual que la anterior, es una función callback que queda a la espera de recibir un dato por parte del servidor MQTT.

```
def on_connect_temperatura(client, userdata, flags, rc):
    client.subscribe("ALSW/temperatura")
```

De la misma forma, se va a definir una función por cada sensor implementado. Este es el ejemplo del sensor de temperatura, donde se define un objeto cliente MQTT y se hace una llamada tanto a la función on_connect (suscripción) como a la función on_message (extracción del dato y decodificación). Se define el servidor y se inicia un bucle que dura 2 segundos, tiempo suficiente para conectarse y leer la información. Finalmente, se hace un return de la función que devuelve y pasa el dato a la función Lambda.

```
def temperatura():
    client = mqtt.Client()
    client.on_connect = on_connect_temperatura
    client.on_message = on_message
    client.connect("test.mosquitto.org", 1883, 60)
    client.loop_start()
    sleep(2)
    client.loop_stop()
    return data
```

Lambda_function.py

Con la creación de la skill, la función lambda consta de unas librerías iniciales necesarias para el uso de la skill, creación de los IntentHandlers, uso de utils, etc. Además, se importa el archivo test.py para poder llamar a sus funciones y métodos.

```
import logging
import ask_sdk_core.utils as ask_utils

from ask_sdk_core.skill_builder import SkillBuilder
from ask_sdk_core.dispatch_components import AbstractRequestHandler
from ask_sdk_core.dispatch_components import AbstractExceptionHandler
from ask_sdk_core.handler_input import HandlerInput

from ask_sdk_model import Response

import test
```

LaunchRequestHandler es la primera clase que se va a ejecutar cuando se abra la skill. Para ello se ha definido una frase que Alexa dirá cada vez que se inicie la sesión. Esta frase puede escribirse tanto en inglés como en español, pero hay que tener en cuenta cierta configuración para hacer que funcione, ya que ahora la frase en inglés se podría leer pero con acento español.

```
class LaunchRequestHandler(AbstractRequestHandler):
    """Handler for Skill Launch."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return
ask_utils.is_request_type("LaunchRequest")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        speak_output = "Hola, bienvenido a la skill sensores test de
Alexa, ¿que parámetro de nuestra placa PCB te gustaría conocer?,
puedes decir temperatura, intensidad, sonido o quien anda ahi!!"
```

```
        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(speak_output)
                .response
        )
```

La clase *MovimientoIntentHandler* tiene en su interior un par de métodos como el resto de las clases *intentHandler* que son *can_handle* y *handle*. La primera es la que identifica el nombre de la función que se ha de activar según el comando de voz que se haga. Una vez detectado el método que se debe ejecutar, entrará a la función *handle* donde lo primero que se define es una variable "dato". Esta hará una llamada a la función correspondiente al sensor que se quiera leer (test.movimiento en este caso) y almacenará en "dato" el resultado de la llamada.

Ahora se define speak_output como variable string donde se establece la frase que va a decir Alexa y se concatena con el valor del dato del sensor.

En el return final de la función, al objeto llamado response_builder, se le índica lo que tiene que decir a través del método .speak y pasando como parámetro el string generado anteriormente. Una vez se tiene la respuesta, es posible que el usuario desee responder algo a la frase de Alexa. Esta respuesta ha de ser una nueva solicitud del valor de un sensor o una orden para que la sesión finalice. No obstante, para evitar que se cierre automáticamente después de decir la frase, se añade un .ask que hace que la sesión siga abierta a la espera de una nueva solicitud.

```
class MovimientoIntentHandler(AbstractRequestHandler):
    """Handler para consultar la última vez que se detectó
movimiento"""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return
ask_utils.is_intent_name("SensorMovimientoIntent")(handler_input)


    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        dato = test.movimiento()
        speak_output = "La última vez que se detectó movimiento fue "
+ dato
        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(speak_output)
                # .ask("add a reprompt if you want to keep the session
open for the user to respond")
                .response
        )
```

La clase CancelOrStopIntentHandler viene por defecto definida a la hora de crear una nueva skill, en ella se agrupan dos Intents o entradas para detener la aplicación, estas son CancelIntent y StopIntent.

La estructura es similar al resto de clases del archivo lambda_function.py, a diferencia de que en el método return, no interesa mantener la sesión abierta una vez que el usuario desea finalizar. En ese caso no se añadiría el .ask como en el resto de funciones handle. Se mostrará un mensaje de despedida y se cerrará la aplicación, teniendo que volver a iniciar la skill si se desea entrar de nuevo.

```
class CancelOrStopIntentHandler(AbstractRequestHandler):
    """Single handler for Cancel and Stop Intent."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return
(ask_utils.is_intent_name("AMAZON.CancelIntent")(handler_input) or

ask_utils.is_intent_name("AMAZON.StopIntent")(handler_input))

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        speak_output = "Hasta la proxima!"

        return (
            handler_input.response_builder
                .speak(speak_output)
                .response
        )
```

Al final del archivo Lambda se define un objeto del tipo SkillBuilder para construir todos los métodos que forman la skill. Se tendrá un método por cada sensor implementado, un método para iniciar la aplicación, dos para finalizar, otro para detectar fallos y un IntentReflectorHandler. Este último se utiliza para la prueba y depuración del modelo de interacción, simplemente repitiendo lo mismo que dijo el usuario.

```
sb = SkillBuilder()

sb.add_request_handler(MovimientoIntentHandler())
sb.add_request_handler(SonidoIntentHandler())
sb.add_request_handler(LuzIntentHandler())
sb.add_request_handler(TemperaturaIntentHandler())
sb.add_request_handler(LaunchRequestHandler())
sb.add_request_handler(HelpIntentHandler())
sb.add_request_handler(CancelOrStopIntentHandler())
sb.add_request_handler(FallbackIntentHandler())
sb.add_request_handler(SessionEndedRequestHandler())
sb.add_request_handler(IntentReflectorHandler()) # make sure
IntentReflectorHandler is last so it doesn't override your custom
intent handlers

sb.add_exception_handler(CatchAllExceptionHandler())

lambda_handler = sb.lambda_handler()
```

Carga de datos JSON

Para la estructura del fichero JSON, se establece el modelo de la ilustración anterior. Este determina el nombre con el que la función se relaciona con la petición por voz, pudiéndose dar el caso de varios nombres o distintas formas de solicitar el valor de un sensor. En caso de una Alexa configurada en otro idioma, este archivo también se vería modificado.

```
{
    "interactionModel": {
        "languageModel": {
            "invocationName": "sensores mqtt",
            "intents": [
                {
                    "name": "AMAZON.CancelIntent",
                    "samples": [
                        "Cancela",
                        "Cancelar"
                    ]
                },
                {
                    "name": "AMAZON.HelpIntent",
                    "samples": [
                        "me presta su ayuda preciosa dama",
                        "ayudame",
                        "necesito ayuda",
                        "Ayuda"
                    ]
                },
                {
                    "name": "SensorTemperaturaIntent",
                    "slots": [],
                    "samples": [
                        "temperatura",
                        "dime la temperatura",
                        "que temperatura hace",
                        "cuanto calor hace"
                    ]
                },
                {
                    "name": "SensorLuzIntent",
                    "slots": [],
                    "samples": [
                        "luz",
                        "dime la intensidad",
                        "cuanta luz hace",
                        "intensidad"
                    ]
                },
                {
                    "name": "SensorSonidoIntent",
                    "slots": [],
                    "samples": [
```

```
                    "sonido",
                    "como está el led",
                    "dime sonido",
                    "estado del led"
                ]
            },
            {
                "name": "SensorMovimientoIntent",
                "slots": [],
                "samples": [
                    "movimiento",
                    "ultimo movimiento",
                    "hay movimiento",
                    "se ha movido algo",
                    "quien anda ahí"
                ]
            },
            {
                "name": "AMAZON.StopIntent",
                "samples": [
                    "párate",
                    "Parar",
                    "Para",
                    "stop"
                ]
            },
            {
                "name": "AMAZON.NavigateHomeIntent",
                "samples": []
            },
            {
                "name": "AMAZON.FallbackIntent",
                "samples": []
            }
        ],
        "types": []
    }
}
```

3. Arquitectura de la Aplicación

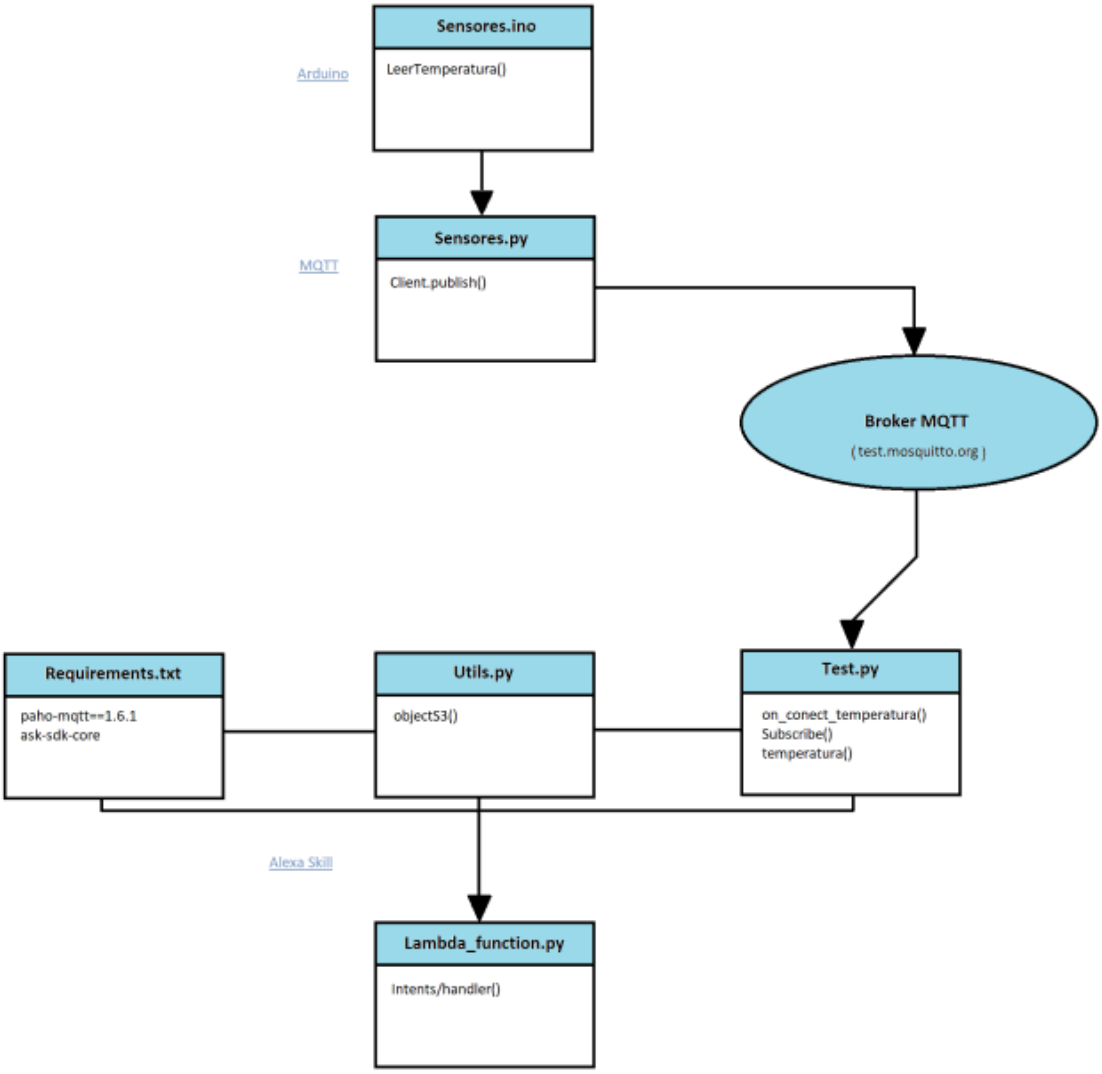

Ilustración 13 Arquitectura de la Aplicación

El diseño del proyecto está compuesto por 6 archivos, 2 de ellos estarán ejecutándose en el ordenador y los otros 4 en la nube de Amazon Web Service. Como se puede apreciar en la imagen, cada uno de los archivos o conjunto de ellos tiene a su lado el nombre de la parte del desarrollo a la que pertenece, siendo estos Arduino, MQTT y la Skill de Alexa.

El primer archivo es sensores.ino, el cual se encarga de controlar los sensores, leer su valor y mandarlo a través del puerto asignado para la salida (COM5). A continuación, sensores.py va a ser el encargado de recibir cada uno de los valores leídos por los sensores, decodificarlos para hacer el mensaje legible al ojo humano y publicar la información en el servidor MQTT o broker.

Una vez allí, la encargada de trabajar con esos datos es la skill de Alexa, formada por los siguientes ficheros: Requirements.txt, archivo que contendrá las librerías necesarias para el uso y creación de la skill, incluyendo la librería paho-mqtt para leer los datos del servidor. Utils.py va a crear una URL ya firmada, para compartir un objeto S3 con una caducidad limitada de 60 segundos. Test.py contendrá las funciones de suscripción al broker MQTT, es aquí cuando se leen los datos del canal concreto que se desee y se almacena la información en variables para luego tratarlas con la función Lambda.

En esta función se definen todos los Intents necesarios para el correcto funcionamiento de la Skill. Un Intent representa una acción que cumple con una solicitud hablada por parte del usuario. Opcionalmente pueden necesitar de argumentos para responder a la petición realizada, pero no en este caso. Dentro de cada función Intent se hace una llamada a otra función definida en Test.py (búsqueda y suscripción a Topic).

Con el dato ya dentro de la clase IntentHandler(), este se añade a un diálogo prefijado por el desarrollador como parámetro para ser leído por la Alexa. A la hora de solicitar un valor, por ejemplo Temperatura, Alexa estará lista para devolver el dato medido por el sensor correspondiente en el lugar en el que se encuentre la placa.

Una vez hecho esto, el programa Lambda quedará a la espera de una nueva petición por parte del usuario o en caso contrario de una orden para finalizar la ejecución de la skill.

4. PRESUPUESTO

Para poder hacer uso de la aplicación, es necesaria la adquisición de cierto material. A continuación, se muestra el precio de cada producto necesario para el desarrollo y uso de la aplicación.

1. Cables y Led ..... 2€
2. Placa PCB ..... 5€
3. Sensores (Temperatura, sonido, intensidad y movimiento) ..... 5€
4. ESP32 Dev kit v2 ..... 10€
5. Alexa (Echo Dot 3ª Gen) ..... 20€
6. Ordenador portátil ..... 400€

El presupuesto medio es aproximadamente de 442 euros, variando en función del dispositivo Alexa y portátil que se escoja. Es por ello que el presupuesto se puede reducir haciendo uso de un ordenador más básico.

Respecto al tiempo empleado para la realización del proyecto, en el siguiente diagrama de Gantt se puede apreciar la actividad realizada en el periodo acorde. [13]

Actividad	Periodos											
	01-abr	15-abr	30-abr	15-may	01-jun	15-jun	30-jun	15-jul	01-ago	15-ago	31-ago	15-sep
Conocer Bases del Proyecto												
Analisis del diseño												
Programación Arduino												
Montaje de la placa PCB												
Programación MQTT												
Programacion Alexa												
Unión de todos los elementos												
Pruebas												
Reunión para mostrar el resultado												
Documentación del Proyecto												

Ilustración 14 Diagrama de Gantt

5. CONCLUSIONES

El trabajo realizado nos permite no sólo encontrar una vía en las nuevas tecnologías a través de la cual podamos acercarnos más al interés de la población joven; sino conocer en mayor profundidad el uso de muchas herramientas que han facilitado el desarrollo del proyecto. La principal es AWS o Servicio Web de Amazon, en el que se ha podido desarrollar una skill, partiendo de la base de otras muchas que se encuentran ya en la nube de Amazon.

A nivel innovación y desarrollo se pueden abrir una infinidad de puertas para crear, aprender y en vista a futuro sacar partido de ello. No solo permite divertirse investigando nuevas funcionalidades con tu dispositivo Alexa, si no también es una forma de solucionar problemas y ayudar al aprendizaje de muchas personas.

Entrando en especificaciones del diseño presentado, como se mencionó en el anteproyecto había dos propuestas para el desarrollo de la aplicación. Una de ellas era que el dispositivo Alexa solicitase el valor de un sensor y la petición se realizase en el momento, necesitando tener la placa accesible en tiempo real. Viendo cómo ha evolucionado el proyecto, por eficiencia en la lectura de datos, se ha decidido tomar la otra vía. Esta manda de forma continua valores al servidor MQTT, de forma que siempre que se quiera solicitar información, Alexa leerá la última información publicada en *mosquitto*. Esto lo que permite es que, en caso de realizar una solicitud y no tener operativa la placa, Alexa devolverá el último valor que se almacenó en el servidor, pudiendo disponer de información, que aunque no es 100% precisa, es mejor que no disponer de ninguna por una pérdida de conexión con la placa o fallo de algún sensor. Además, dado que la permanencia de datos en el servidor MQTT es bastante alta, siempre habrá datos que leer.

Como trabajo futuro quedaría lanzar la *skill* a nivel internacional, es decir, que todo el mundo pueda hacer uso de ella desde su casa y con su propia Alexa, sin necesidad de nada más, ni portátil, ni placa. Para esto se necesitaría más tiempo, que sería empleado en mejorar el software de forma que sea aceptado por AWS y modificado para que de forma independiente, detecte diferentes dispositivos simultáneos y diversos idiomas.

6. Bibliografía

[1] “Las tecnologías de la información y la comunicación (TIC) y los adolescentes. Algunos datos.” https://cursa.ihmc.us/rid=1GN285S66-QLP75S-7JS0/reflexion05.htm (accessed Aug. 19, 2022).

[2] “Nº 21. Juventud y Nuevas Tecnologías | Injuve, Instituto de la Juventud.” http://www.injuve.es/ca/observatorio/infotecnologia/no-21-juventud-y-nuevas-tecnologias (accessed Aug. 14, 2022).

[3] “▷ Comparación de Arduino vs ESP8266 vs ESP32 (2022).” https://elosciloscopio.com/comparacion-arduino-vs-esp8266-vs-esp32/ (accessed Sep. 02, 2022).

[4] “(453) ¿QUÉ ES EL ESP32? - YouTube.” https://www.youtube.com/watch?v=9ouMjvVACnc&ab_channel=IngenieroenCasa (accessed Aug. 14, 2022).

[5] “Paho Python MQTT Client - Publish With Examples.” http://www.steves-internet-guide.com/publishing-messages-mqtt-client/ (accessed Aug. 14, 2022).

[6] “Quality of Service (QoS) 0,1, & 2 MQTT Essentials: Part 6.” https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/ (accessed Aug. 14, 2022).

[7] “xoseperez / fauxmoesp — Bitbucket.” https://bitbucket.org/xoseperez/fauxmoesp/src/master/ (accessed Aug. 14, 2022).

[8] “Source code beautifier / syntax highlighter – convert code snippets to HTML « hilite.me.” http://hilite.me/ (accessed Aug. 14, 2022).

[9] “Getting Date & Time From NTP Server With ESP32.” https://lastminuteengineers.com/esp32-ntp-server-date-time-tutorial/ (accessed Aug. 14, 2022).

[10] “millis() - Arduino Reference.” https://www.arduino.cc/reference/en/language/functions/time/millis/ (accessed Aug. 14, 2022).

[11] “MQTT 3. Python Libreria PAHO – MrubioDev.” https://roboticcode.wordpress.com/2020/04/27/mqtt-2-python-libreria-paho/ (accessed Aug. 14, 2022).

[12] “AWS Console - Signup.” https://portal.aws.amazon.com/billing/signup#/start/email (accessed Sep. 09, 2022).

[13] “TFG - Wrike.” https://www.wrike.com/workspace.htm?acc=5197335#folder/802066745/timeline3?spaceId=802064513&viewId=-4 (accessed Aug. 14, 2022).

[14] “Git - Downloading Package.” https://git-scm.com/download/win (accessed Aug. 19, 2022).

[15] “mosquitto_pub man page | Eclipse Mosquitto.” https://mosquitto.org/man/mosquitto_pub-1.html (accessed Aug. 14, 2022).

[16] “Amazon Alexa Console - Amazon Alexa Official Site.” https://developer.amazon.com/alexa/console/ask/build/custom/amzn1.ask.skill.605bbb4f-e38e-4fca-80e7-95afd521980c/development/es_ES/intents/HelloWorldIntent (accessed Aug. 14, 2022).

[17] “(453) Creando Skills Alexa desde Cero - YouTube.” https://www.youtube.com/watch?v=mqIk4-Ry-cg&ab_channel=AlexaDevelopers (accessed Aug. 14, 2022).

[18] “Crear una skill de Alexa usando Python 2/2 | by Francisco Rivas | Diseñando para la Voz | Medium.” https://medium.com/disenando-para-la-voz/crear-una-skill-de-alexa-usando-python-2-2-d3bb1bbf95b5 (accessed Aug. 14, 2022).

7. Anexos y/o Apéndices

Instalación de software Arduino con módulo ESP32

Lo primero que habrá que hacer será descargar la última versión de Arduino IDE para poder disponer de un entorno de trabajo (versión 1.9.19). Accedemos a ella a través del siguiente enlace: https://www.arduino.cc/en/Main/Software

Después se necesitará instalar el software de GitHub (Git Gui) para poder clonar repositorios en el disco local y así tener acceso a las librerías necesarias para que el microcontrolador funcione. Desde la siguiente referencia se instalará la versión que más se ajuste a las características del ordenador utilizado. [14]

El siguiente paso sería configurar Arduino para disponer del módulo de trabajo necesario, ya que de fábrica, al menos en la versión utilizada, todavía no se dispone de los módulos necesarios de ESP32. Para ello, se accede a la aplicación de Git Gui y habría que hacer "click" sobre Clone Existing Repository, copiar las direcciones de origen y destino que se indica en la página pero modificando en esta última la ruta donde se haya instalado Arduino. En el caso práctico presentado, si seguimos los pasos por defecto se instalará en: *C:\Users\their\Documents\Arduino*.

Por último habría que acceder a la ruta:

C:\Users\their\Documents\Arduino\hardware\espressif\esp32\tools, hacer doble "click" sobre el archivo get.exe. De esta forma el IDE de Arduino va a utilizar el compilador *xtensa-esp32-elf* para generar el código máquina entendible que después se va a transferir al ESP32.

Así comenzaría el proceso de descarga del compilador a la carpeta de herramientas. Una vez finalizado se reiniciará el programa y estarían disponibles, dentro del IDE de Arduino los módulos necesarios para trabajar con el microcontrolador y otras variaciones de los ESP. Dentro de la sección herramientas→placa→ seleccionar: esp32 Dev module.

Tras todo esto, se dispondría de un entorno listo y funcionando para poder trabajar con él.

Instalación de librerías para Python

Descargar del siguiente repositorio el instalador pip para la instalación de la librería paho-mqtt

```
D:\>curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 2548k  100 2548k    0     0  2548k      0  0:00:01 --:--:--  0:00:01 2767k
```

Utilizar *py* como abreviatura de Python para ejecutar el archivo .py

```
D:\>py get-pip.py
Collecting pip
  Downloading pip-22.0.3-py3-none-any.whl (2.1 MB)
     ---------------------------------------- 2.1/2.1 MB 26.3 MB/s eta 0:00:00
Collecting wheel
  Downloading wheel-0.37.1-py2.py3-none-any.whl (35 kB)
Installing collected packages: wheel, pip
  WARNING: The script wheel.exe is installed in 'C:\Users\their\AppData\Local\P
ocal-packages\Python310\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this war
  WARNING: The scripts pip.exe, pip3.10.exe and pip3.exe are installed in 'C:\U
qbz5n2kfra8p0\LocalCache\local-packages\Python310\Scripts' which is not on PATH
  Consider adding this directory to PATH or, if you prefer to suppress this war
Successfully installed pip-22.0.3 wheel-0.37.1
WARNING: You are using pip version 21.2.4; however, version 22.0.3 is available
You should consider upgrading via the 'C:\Users\their\AppData\Local\Microsoft\W
e -m pip install --upgrade pip' command.
```

Instalar la librería de paho-mqtt versión 1.6.1

```
D:\>pip install paho-mqtt
Collecting paho-mqtt
  Downloading paho-mqtt-1.6.1.tar.gz (99 kB)
     ---------------------------------------- 99.4/99.4 KB 5.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: paho-mqtt
  Building wheel for paho-mqtt (setup.py) ... done
  Created wheel for paho-mqtt: filename=paho_mqtt-1.6.1-py3-none-any.whl size=654
46238
  Stored in directory: c:\users\their\appdata\local\pip\cache\wheels\8b\bb\0c\794
Successfully built paho-mqtt
Installing collected packages: paho-mqtt
Successfully installed paho-mqtt-1.6.1
```

Instalar la librería *pyserial*.

```
D:\6 ano\TFG\python>py -m pip install pyserial-3.5-py3-none-any.whl
Processing d:\6 ano\tfg\python\pyserial-3.5-py3-none-any.whl
Installing collected packages: pyserial
  WARNING: The scripts pyserial-miniterm.exe and pyserial-ports.exe are inst
 not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this
Successfully installed pyserial-3.5

D:\6 ano\TFG\python>
```

Instalación de Mosquitto - MQTT

Para la realización de pruebas y test sobre el servicio MQTT, se puede hacer uso de la consola o terminal del ordenador.

Habrá que instalar mosquitto dentro del ordenador. Una vez hecho eso, la siguiente imagen muestra como se publicaría un mensaje a través del servidor de mosquitto.

La primera de las terminales, publicará el mensaje en el canal ALSW/temperatura

```
D:\6 ano\TFG\mosquitto>mosquitto_pub -h test.mosquitto.org -t ALSW/temperatura -m 36
```

Con el comando mosquitto_pub se publica, con el parámetro -h se indica el host, con -t el topic y con -m el mensaje.

Para poder recibir los datos al momento, la segunda terminal tendrá que estar abierta, a la espera de la información.

```
D:\6 ano\TFG\mosquitto>mosquitto_sub -h test.mosquitto.org -t ALSW/#
36
36
36
```

Si se quiere leer datos ya publicados en un canal habrá que añadir un parámetro más al comando, -r (retain), que indicará que se quiere mantener publicado y no desaparecer durante un tiempo.

```
D:\6 ano\TFG\mosquitto>mosquitto_pub -h test.mosquitto.org -t ALSW/temperatura -m 36º -r

D:\6 ano\TFG\mosquitto>mosquitto_pub -h test.mosquitto.org -t ALSW/sonido -m 100decibelios -r
```

Para leer mensajes publicados en canales diferentes, se añade al -t una "#" de forma que todos los mensajes (hijos) enviados a la ruta *ALWS/* se mostrarán.

```
D:\6 ano\TFG\mosquitto>mosquitto_sub -h test.mosquitto.org -t ALSW/#
100decibelios
36
```

Se puede ajustar el mensaje a lo deseado, para ello consultar el manual. [15]

Muestras de Funcionamiento

En este apartado se aprecian una serie de gráficas que muestran el progreso del desarrollo de la skill. Como se puede ver, el pico de actividad se da durante la primera semana de Mayo.

La primera de ellas proporciona datos sobre la confianza que proporcionan los Intents declarados en función del tiempo. Tras la primera implementación exitosa surgen algunos fallos debido a las pruebas realizadas contra la skill.

El segundo gráfico muestra la latencia de los servicios de Alexa. El número aquí indica el percentil del total de solicitudes.

- P50: el percentil de latencia 50, el 50% de las solicitudes serán más rápidas que el valor p50.
- P90: el percentil de latencia 90, el 90% de las solicitudes serán más rápidas que el valor p90.

Cuantos más bajos sean los valores de estas métricas, mejor será el rendimiento del servicio.

La última gráfica muestra las respuestas de la skill a las solicitudes realizadas. Como se puede apreciar al final de la gráfica, la tasa de acierto es del 100% y la de fallo del 0%, un dato bastante significativo, ya que da a entender que todas las peticiones se han resuelto correctamente.

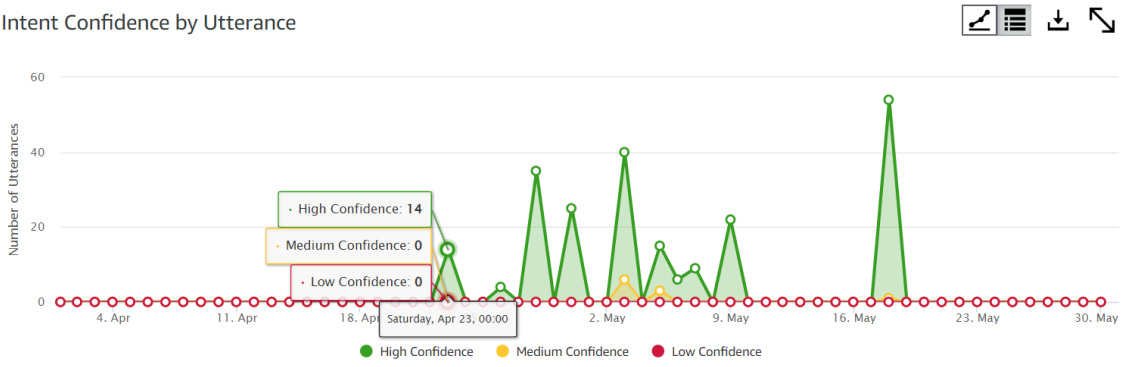

Ilustración 15 Confianza de los Intents

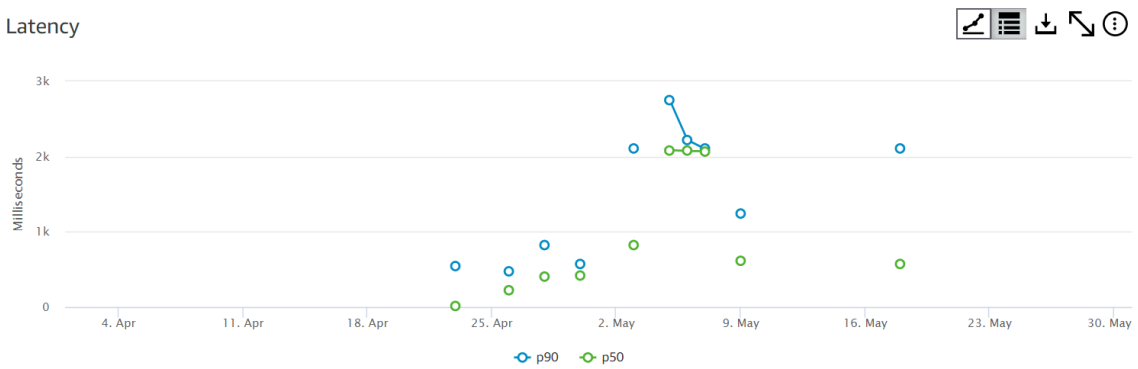

Ilustración 16 Latencia

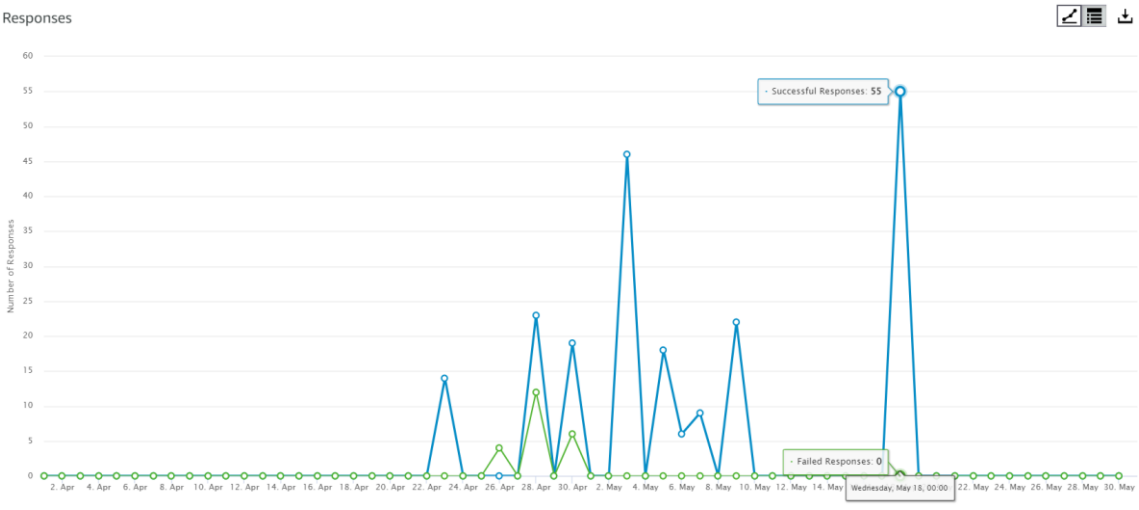

Ilustración 17 Respuestas

A continuación se muestran las imágenes desde el proceso de toma de datos hasta el de lectura por parte de Alexa.

La primera imagen se corresponde con la salida del serial de Arduino IDE. Este imprime por la consola los datos leídos por los sensores, aunque para poder tratar la información por el siguiente programa ha de estar cerrado, ya que ambos hacen uso del puerto COM5.

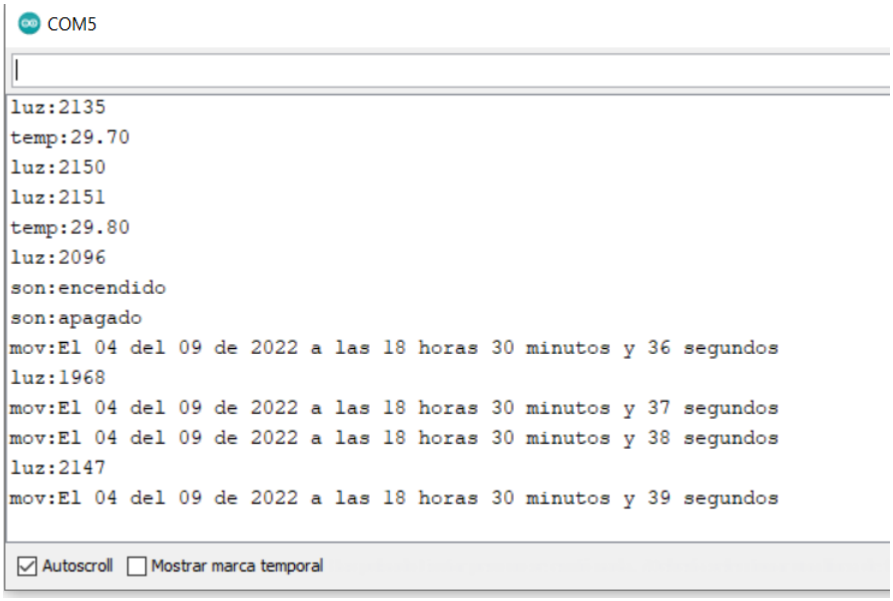

Ilustración 18 Serial Arduino IDE

En la siguiente imagen se muestra la ejecución del programa que publica los datos en el servidor MQTT. La salida por consola se hace para comprobar que los datos son los mismos que los leídos por el programa en Arduino.

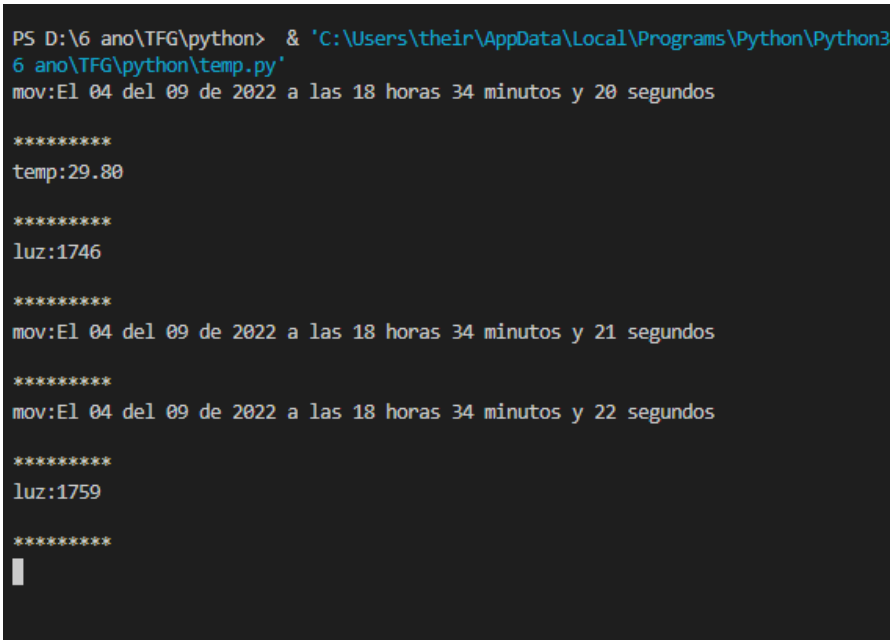

Ilustración 19 Programa publicador MQTT

En la última de las imágenes se muestra la respuesta de Alexa a las preguntas de la temperatura y del movimiento. Esto se explicará más a fondo en el siguiente apartado sobre la skill.

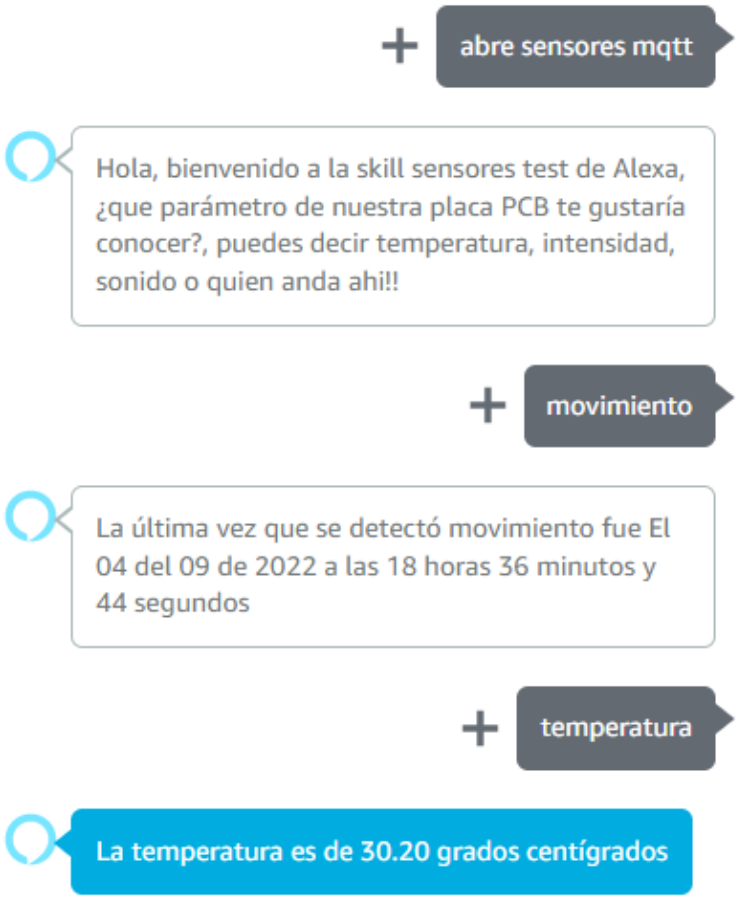

Ilustración 20 Lectura de datos Alexa

Crear Skill de Alexa

En la siguiente ilustración se muestra la ventana principal de la Consola de Desarrollo de Alexa. [16]

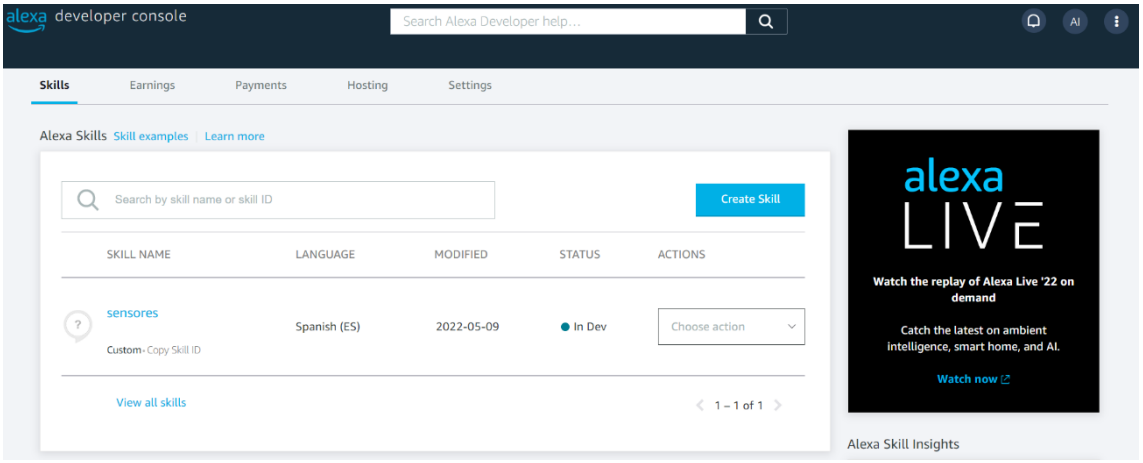

Ilustración 21 Ventana principal Skill

Aquí, aparece la skill que se ha desarrollado para el proyecto, llamada Sensores. A la hora de crear una nueva skill, se permiten varias opciones en cuanto al lenguaje de desarrollo, pudiendo elegir entre Node.js (entorno de ejecución de JavaScript) y Python, como es este caso.

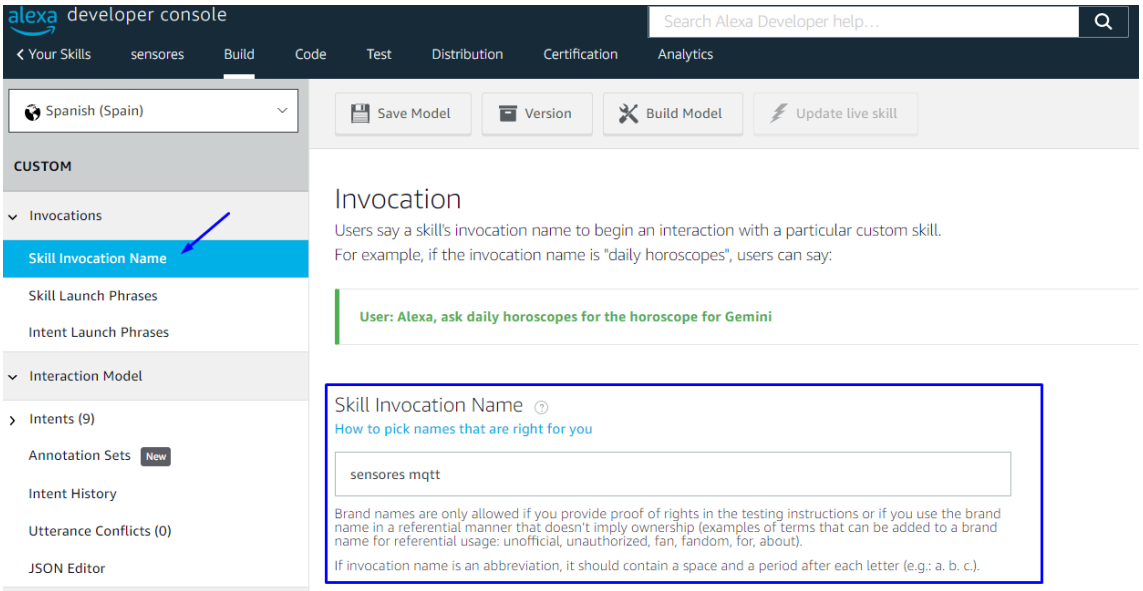

Ilustración 22 Nombre de Invocación

En el apartado Skill Invocation Name se define el nombre de la skill. La llamada se hará mediante: - Alexa, abre sensores mqtt.

Cada vez que se modifique algo en la skill, es necesario darle al botón de Save Model para guardar los cambios y también a Build Model en caso de modificar algo de código.

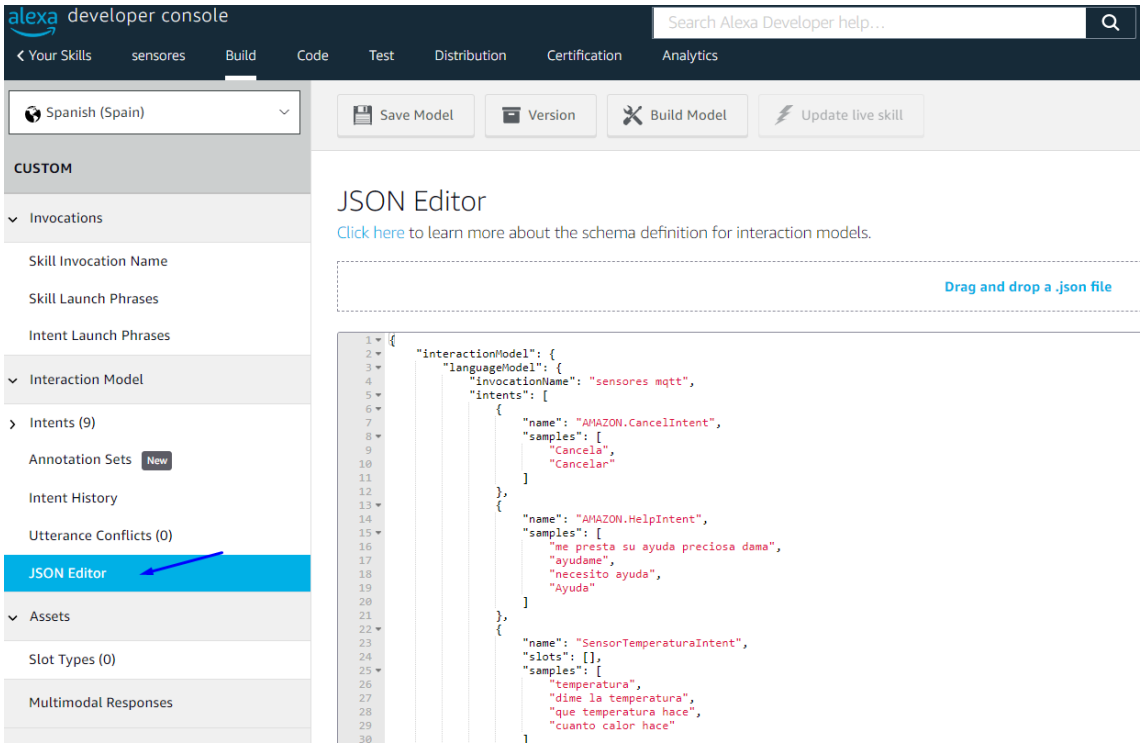

Ilustración 23 Fichero Json

Para personalizar los Intents, es decir, para determinar a través de que comando de voz la skill responderá con la información solicitada, se añadirán todos los posibles valores ya sea creándolos sobre el archivo que se muestra o desde un fichero externo, siempre con el mismo formato JSON.

Ilustración 24 Consola de Test Skill de Alexa

Este es el test de la consola de Alexa. Con él se pueden hacer solicitudes a la skill para detectar si esta responde correctamente con la información deseada. Este simulador permite realizar peticiones tanto escritas como por voz y como muestra la ilustración, se puede seleccionar el idioma de la solicitud. [17], [18]

En este ejemplo, se han realizado 3 peticiones. La primera inicia la skill y lanza un mensaje de bienvenida donde se indican unas posibilidades para interaccionar con la Alexa. La segunda es una petición sobre el sensor del sonido, que indica que el led está encendido. La última da el valor de la última temperatura medida por la placa.

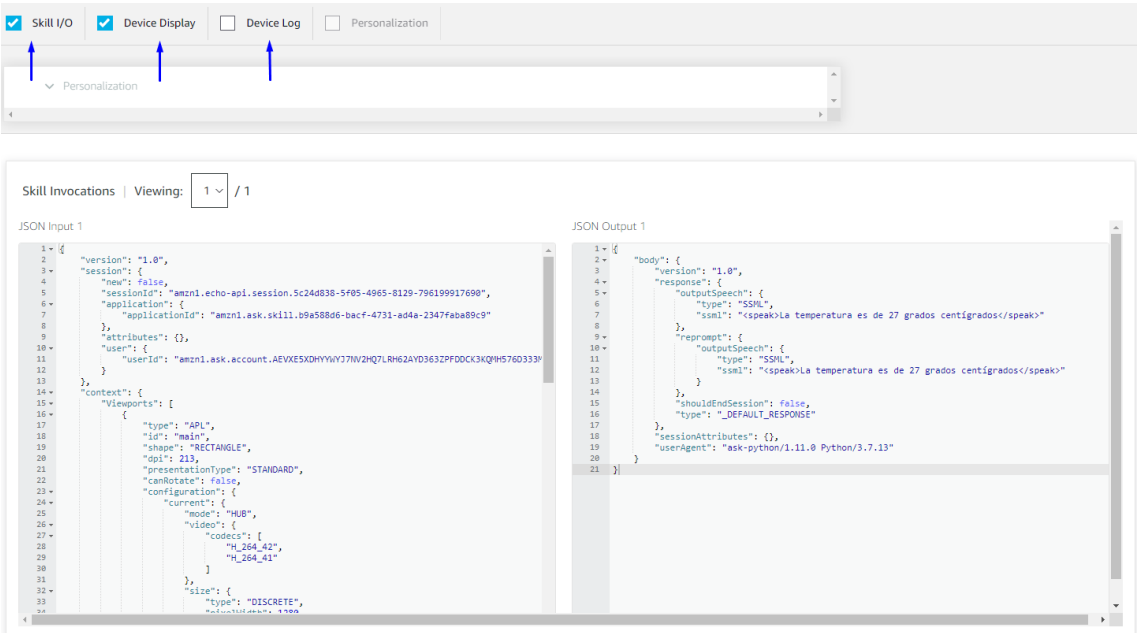

Ilustración 25 Respuesta JSON

Por último existe una salida de datos para ver si nuestras solicitudes son correctas, se muestra tanto la entrada como la salida. Además provee un fichero de log donde se mostrarán todas las peticiones y acciones realizadas.

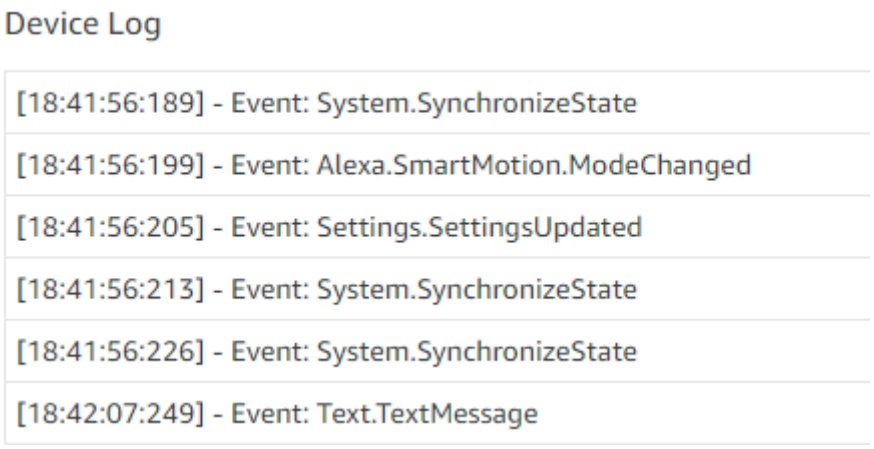

Ilustración 26 Log Skill Alexa

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR