ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Máster

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA EMBEBIDO DE TELECONTROL BASADO EN ARDUINO CLOUD

(Design and implementation of a telecontrol embedded system based on Arduino Cloud)

Para acceder al Título de

Máster Universitario en Ingeniería de Telecomunicación

Autora: Raquel Martín Turrión

Junio - 2024

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO DE FIN DE MÁSTER

Realizado por: Raquel Martín Turrión
Director del TFM: Roberto Sanz Gil
Título: "Diseño e implementación de un sistema embebido de telecontrol basado en Arduino Cloud"
Title: "Design and implementation of a telecontrol embedded system based on Arduino Cloud"

Presentado a examen el día: 28 de junio de 2024

para acceder al Título de

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): García Arranz, Marta
Secretario (Apellidos, Nombre): García García, José Ángel
Vocal (Apellidos, Nombre): García Gutiérrez, Alberto Eloy

Este Tribunal ha resuelto otorgar la calificación de: ……

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFM
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Trabajo Fin de Máster Nº
(a asignar por Secretaría)

Agradecimientos

A mi tutor, Roberto Sanz Gil, que ha sabido guiarme en el transcurso de este Trabajo de Fin de Máster, ayudándome con gran disposición con cada piedra en el camino.

A mis compañeros más cercanos, quienes han convertido este camino de dos años en un aprendizaje más llevadero gracias al apoyo mutuo prestado durante el máster.

A mis amigos, por creer en mis capacidades para completar sin problemas la formación que empecé hace ya seis años en la Universidad de Cantabria.

A mi familia, por ayudar a levantarme cada vez que me venía abajo y ser mi fuente constante de inspiración y motivación.

Resumen

Este proyecto consistirá en el desarrollo de un sistema de telecontrol con hardware embebido de bajo coste y acceso a los datos proporcionados por sensores a través de la plataforma Arduino Cloud, la cual permite conectividad global a través de Internet.

En concreto, el diseño partirá de las placas Arduino Nano 33 IoT y ESP32-WROOM-32, compatibles con Arduino Cloud, a las cuales se les conectará varios sensores para una aplicación genérica de monitorización del entorno doméstico del usuario.

Además, la interacción con el sistema desarrollado integrará un servicio de notificaciones a través de alguna de las plataformas de mensajería más populares actualmente. Estas son el correo electrónico a través de Gmail o mensajes instantáneos mediante aplicaciones como Telegram y WhatsApp.

Abstract

This project will involve the development of a remote control system with low cost embedded hardware and access to data provided by sensors through the Arduino Cloud platform, which enables global connectivity via the Internet.

Specifically, the design will be based on the Arduino Nano 33 IoT and ESP32-WROOM-32 boards, both compatible with Arduino Cloud, to which various sensors will be connected for a generic home environment monitoring application.

Furthermore, interaction with the developed system will integrate a notification service through one of the most popular messaging platforms currently available. These are email through Gmail or instant messaging applications such as Telegram and WhatsApp.

Índice de contenidos

Lista de figuras

Lista de tablas

Lista de acrónimos

ADC	Analog-to-Digital Converter
API	Application Programming Interface
CO	Monóxido de carbono
CC	Corriente Continua
CPU	Central Processing Unit
DAC	Digital-to-Analog Converter
DHT	Digital Humidity Temperature
EPA	Electrostatic Discharge Protected Area
ESP	Espressif Systems
GND	Ground
GPIO	General Purpose Input Output
HTTP	HyperText Transfer Protocol
IMU	Inertial Measurement Unit
IoT	Internet of Things
LED	Light Emitting Diode
LPG	Liquefied Petroleum Gas
MQTT	Message Queing Telemetry Transport
NTC	Coeficiente de Temperatura Negativa
PCB	Printed Circuit Board
PWM	Pulse Width Modulation
REST	Representational State Transfer
SoC	System-on-Chip
SoM	System-on-Modules
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter

USB	Universal Serial Bus
Wi-Fi	Wireless Fidelity

Capítulo 1. Introducción

1.1. Objetivos del proyecto

El objetivo de este Trabajo Fin de Máster es realizar el diseño y posterior implementación de un sistema embebido de telecontrol de bajo coste basado en Arduino Cloud, controlado a través de dispositivos embebidos, como son Arduino Nano 33 IoT y la placa ESP32-WROOM-32.

La elección de los sensores ha sido de propósito general, queriendo monitorizar el entorno del hogar del usuario, en este caso la temperatura, humedad y varios gases. El sensor encargado de la detección de gases (LPG, CO y humo) estará conectado al Arduino Nano 33 IoT, mientras que el sensor DHT11 de temperatura y humedad se conectará al ESP32-WROOM-32.

Ambas placas transmitirán los valores leídos de los sensores a través de la plataforma Arduino Cloud, en la que se podrán monitorizar los datos mediante el dashboard.

Además, se buscará interactuar con estos datos a través de distintas plataformas, como puede ser mediante Gmail, Telegram y WhatsApp, estudiando asimismo la viabilidad a la hora de comunicarse de manera bidireccional con el sistema de telemetría.

1.2. Motivación personal

La motivación detrás de este proyecto radica en la creación de un sistema de telecontrol que aproveche el potencial de la tecnología embebida y la conectividad global ofrecida por la plataforma Arduino Cloud. Al utilizar las placas Arduino Nano 33 IoT y ESP32-WROOM-32, compatibles con Arduino Cloud, se pretende diseñar un sistema robusto y versátil que permita la supervisión del entorno doméstico del usuario de manera efectiva.

La integración de diversos sensores en este sistema brindará la oportunidad de recopilar datos precisos y detallados sobre factores clave del entorno, como temperatura, humedad y presencia de gases nocivos para la salud. Esto permitirá una aplicación genérica de monitoreo que pueda adaptarse a diferentes necesidades y contextos de los usuarios.

Además, la inclusión de un servicio de notificaciones a través de plataformas de mensajería populares como Gmail, Telegram y WhatsApp agregará un valor significativo al trabajo. Esto facilitará la comunicación instantánea de eventos y datos importantes, permitiendo una respuesta rápida y eficiente ante cualquier situación.

En resumen, este proyecto busca aprovechar la tecnología de hardware embebido y la conectividad global para crear un sistema de telecontrol avanzado y fácil de usar, que brinde a los usuarios la capacidad de controlar y gestionar su entorno doméstico de manera inteligente y efectiva.

1.3. Estructura del documento

La estructura de este documento se presenta de la siguiente manera:

- En el capítulo 2 se describen los componentes que han constituido el hardware del sistema, habiendo explicado previamente y de manera concisa qué es el IoT y la domótica.
- A continuación, en el capítulo 3, se detallan las herramientas software utilizadas, así como la configuración tanto del Arduino Nano 33 IoT como del ESP32-WROOM-32.
- Posteriormente, en el capítulo 4 se detalla la integración del sistema junto con las formas de notificar al usuario los valores de los sensores.
- Seguidamente, en el capítulo 5 se presenta el presupuesto del proyecto.
- Finalmente, en el capítulo 6, se exponen las conclusiones y posibles líneas futuras que puede ofrecer el sistema de telemetría desarrollado en este trabajo.

1.4. Diferencia entre IoT y Domótica

1.4.1. Internet of Things

El Internet de las Cosas (IoT, por sus siglas en inglés, Internet of Things) es una red de dispositivos físicos, vehículos, electrodomésticos y otros objetos equipados con electrónica, software, sensores, actuadores y conectividad, que les permite conectarse y compartir datos. [1]

El IoT posibilita que estos dispositivos inteligentes se comuniquen entre sí y con otros equipos que cuenten con acceso a Internet, como smartphones y puertas de enlace, estableciendo una extensa red de dispositivos interconectados que logran intercambiar datos y ejecutar varias tareas de forma autónoma. Esto abarca desde la monitorización de las condiciones ambientales en granjas hasta la gestión del tráfico con automóviles y otros dispositivos automotrices inteligentes, el control de máquinas y procesos en fábricas, y el seguimiento de inventarios y envíos en almacenes. [2]

Un sistema típico de IoT opera mediante la recopilación y el intercambio de datos en tiempo real y consta de tres componentes principales: [3]

- *Dispositivos inteligentes*: Estos son dispositivos como televisores, cámaras de seguridad o equipos de ejercicio que han sido dotados de capacidades de computación. Recopilan datos de su entorno, de las entradas de los usuarios o de los patrones de uso, y comunican estos datos a través de Internet hacia y desde su aplicación de IoT.
- *Aplicación de IoT*: Es un conjunto de servicios y software que integra los datos recibidos de varios dispositivos de IoT. Utiliza tecnologías como el machine learning o la inteligencia artificial (IA) para analizar estos datos y tomar decisiones informadas. Estas decisiones se comunican al dispositivo de IoT, que responde de manera inteligente a las entradas.

- *Interfaz de usuario gráfica*: Los dispositivos de IoT o la flota de dispositivos pueden gestionarse a través de una interfaz de usuario gráfica. Ejemplos comunes incluyen aplicaciones móviles o sitios web que permiten registrar y controlar dispositivos inteligentes.

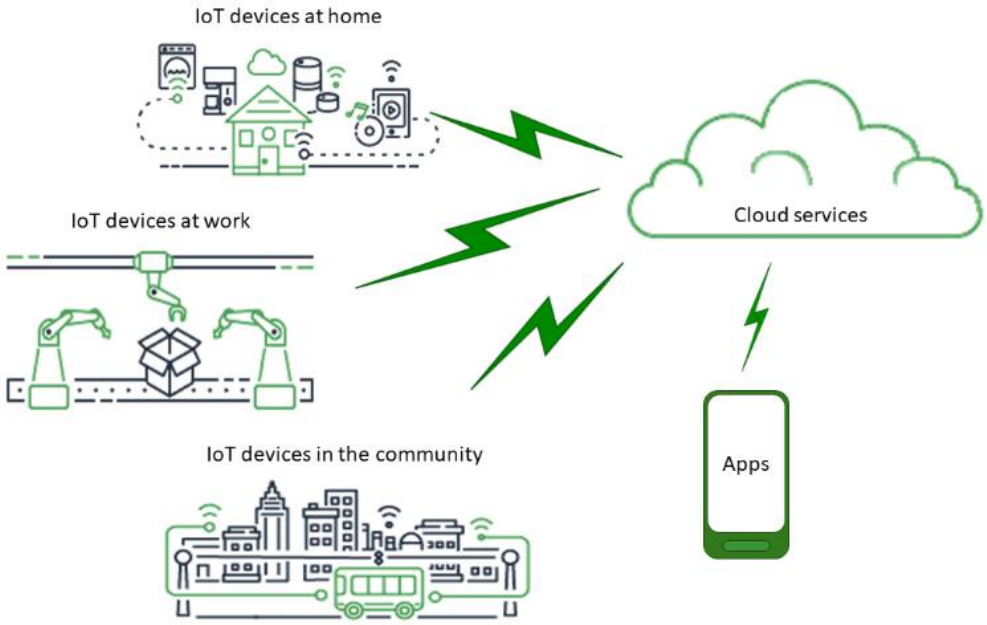


Figura 1.1. Ejemplo de sistema IoT

1.4.2. Domótica

Etimológicamente, el término "domótica" proviene del latín y del griego. Se forma a partir de las palabras "Domus", que significa "casa" en latín, y "Autónomo", que significa "que se gobierna a sí mismo" en griego. [4]

La domótica, también denominada automatización del hogar o ingeniería domótica, se refiere al empleo de la tecnología automatizada para supervisar y controlar las funciones y características de un edificio o vivienda, generalmente a través de sensores y actuadores inalámbricos. [1]

La domótica posibilita la automatización de las tareas vinculadas con la seguridad, el bienestar y el confort mediante un sistema inteligente implementado en una vivienda o edificio. En esencia, implica la integración de la tecnología en el diseño de un espacio habitable. [4]

Algunas aplicaciones domésticas son: [5]

- *Iluminación inteligente*: Permite el control y regulación de la intensidad y el color de las luces, la programación de escenas y horarios específicos, así como su activación mediante sensores de movimiento.

- *Climatización automatizada*: Controla la temperatura y la humedad de una vivienda de manera inteligente, adaptándose a las condiciones externas y a los hábitos de los residentes. Esto incluye el control de persianas y cortinas motorizadas para la gestión del aislamiento térmico.
- *Seguridad y vigilancia*: Incluye cámaras de seguridad, sensores de movimiento, sistemas de alarma y control de accesos mediante cerraduras inteligentes o reconocimiento facial.
- *Gestión energética*: Monitoriza y optimiza el consumo energético del hogar, permitiendo programar y controlar electrodomésticos para un uso más eficiente de la energía.

Figura 1.2. Ejemplo de domótica [4]

1.4.3. Conclusión

Aunque el Internet de las Cosas y la domótica están estrechamente relacionados y comparten ciertos aspectos, como por ejemplo un robot conectado a Internet que puede ser utilizado para la domótica, existen diferencias fundamentales entre ambos conceptos.

Mientras que el Internet de las Cosas se enfoca en conectar objetos y dispositivos físicos a la red, la domótica tiene como objetivo principal proporcionar a los usuarios un control avanzado sobre su hogar, sus funciones y así mejorar su calidad de vida. [1]

Este proyecto implica tanto aspectos de IoT como de domótica.

- *IoT*: Se utiliza Arduino Cloud como plataforma para la conectividad y gestión remota de los dispositivos. Los dispositivos embebidos (Arduino Nano 33 IoT y ESP32-WROOM-32) están conectados a Internet y transmiten los datos de los sensores al Arduino Cloud para su posterior visualización y control a través del dashboard.

- *Domótica*: El objetivo principal del proyecto es monitorizar y controlar el entorno del hogar del usuario, lo cual es característico de la domótica. Se emplean sensores (temperatura, humedad y detección de gases) para recopilar información ambiental dentro del hogar. Además, se menciona la posibilidad de notificar estos datos al usuario a través de servicios como Gmail, Telegram o WhatsApp, lo cual también es una función típica de la domótica para mejorar la seguridad y la comodidad en el hogar.

En resumen, este proyecto combina elementos de IoT para la conectividad y gestión remota de dispositivos, con aspectos de domótica para la monitorización y control del entorno del hogar.

Capítulo 2. Hardware del sistema

En este capítulo se expondrá el hardware empleado en el proyecto, mostrando tanto las placas utilizadas como los sensores que se conectarán a las mismas.

2.1. Arduino Nano 33 IoT

Para la realización de este proyecto, se ha empleado la placa Arduino Nano 33 IoT (Figura 2.1). Es una placa compacta, robusta y potente que incluye conectividad Wi-Fi y Bluetooth. Gracias a su arquitectura de bajo consumo, se presenta como una solución práctica y rentable para proyectos IoT. Además, es totalmente compatible con Arduino IoT Cloud. El criptochip, que integra ATECC608A, almacena las claves criptográficas en hardware, proporcionando un alto nivel de seguridad para estos dispositivos. [6]

Figura 2.1. Arduino Nano 33 IoT [6]

En la Figura 2.2 se muestran los pines de los que dispone esta placa:

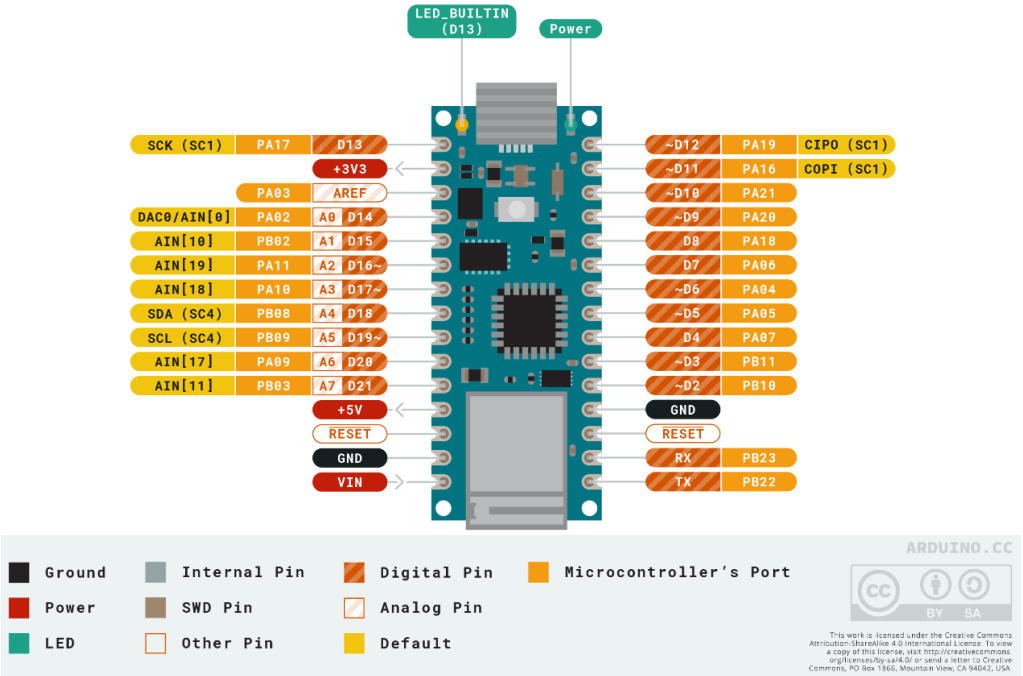


Figura 2.2. Pinout del Arduino Nano 33 IoT [7]

El procesador principal de la placa es un SAMD21 Arm® Cortex®-M0 de 32 bits y bajo consumo. La conectividad Wi-Fi y Bluetooth se gestiona mediante un módulo u-blox NINA-W10, un chipset de bajo consumo que opera en la banda de 2.4 GHz. Cabe destacar que la comunicación segura está garantizada gracias al chip criptográfico Microchip® ATECC608. Además, la placa incluye una IMU (Unidad de Medición Inercial) de 6 ejes, lo que la hace ideal para aplicaciones como sistemas de alarma por vibración, podómetros y posicionamiento relativo de robots, entre otros (Figura 2.3). [7]

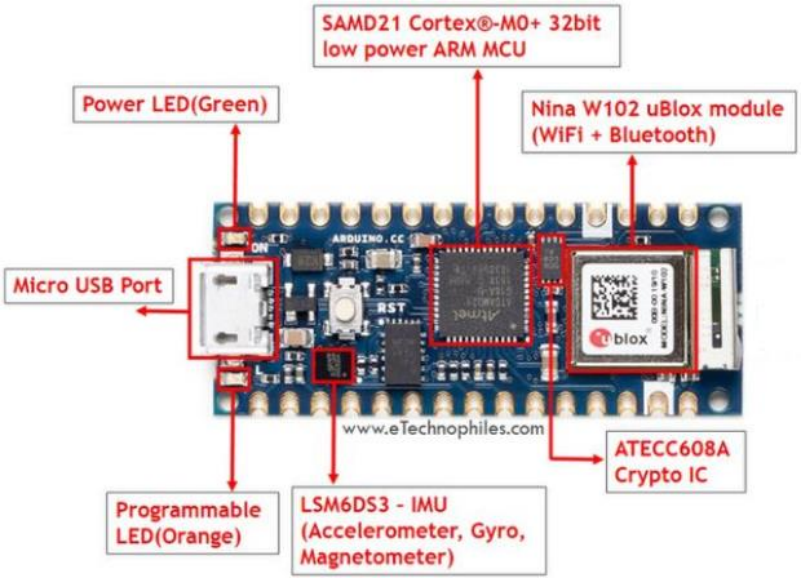


Figura 2.3. Esquema del Arduino Nano 33 IoT [8]

Las especificaciones técnicas se pueden observar en la Tabla 2.1:

Microcontrolador	MCU ARM de baja potencia SAMD21 Cortex®-M0+ de 32 bits
Módulo de radio	u-blox NINA-W102
Elemento seguro	ATECC608A
Tensión de funcionamiento	3.3 V
Voltaje de entrada (límite)	21 V
Corriente CC por pin de E/S	7 mA
Velocidad de reloj	48 MHz
Memoria flash de la CPU	256 KB
SRAM	32 KB
EEPROM	Ninguno
Pines de E/S digitales	14
Pines PWM	11 (2, 3, 5, 6, 9, 10, 11, 12, 16/A2, 17/A3, 19/A5)
UART	1
SPI	1

I2C	1
Pines de entrada analógica	8 (ADC 8/10/12 bits)
Pines de salida analógica	1 (DAC de 10 bits)
Interrupciones externas	Todos los pines digitales (todos los pines analógicos también se pueden usar como pines de interrupción, pero tendrán números de interrupción duplicados)
LED_BUILTIN	13
USB	Nativo en el procesador SAMD21
IMU	LSM6DS3
Longitud	45 mm
Ancho	18 mm
Peso	5 gr. (con cabeceras)

Tabla 2.1. Características técnicas del Arduino Nano 33 IoT [7]

Cabe destacar que hay que soldar el pin de 5V para que pueda alimentar al sensor que esté conectado a esta placa, ya que no viene soldado por defecto. Esto se realiza por la parte inferior de la placa, en el jumper etiquetado como VUSB (Figura 2.4).

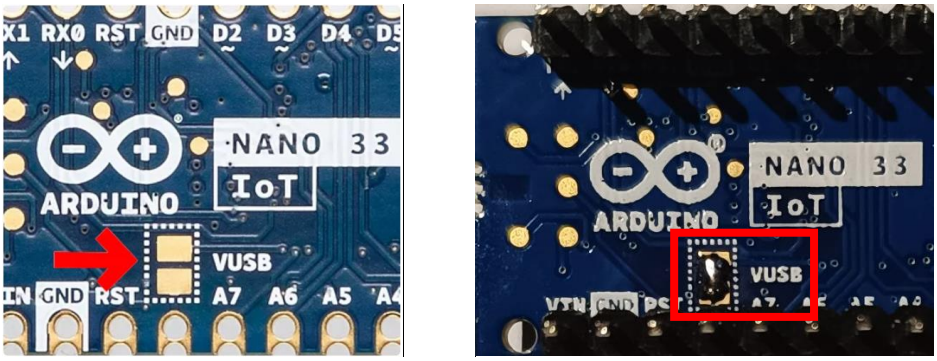


Figura 2.4. Jumpler sin soldar (izquierda) y soldado (derecha) del Arduino Nano 33 IoT [9]

2.2. ESP32-WROOM-32

Esta es una placa genérica que dispone de conectividad Wi-Fi basada en el conocido microprocesador ESP32 (WROOM32). Es totalmente compatible y programable con Arduino, ofreciendo la ventaja de ser mucho más potente, operando a 240 MHz gracias a su procesador Tensilica LX6. Cuenta con un conector microUSB para alimentación y programación a través de la UART, y puede ser programada con el IDE de Arduino. [10]

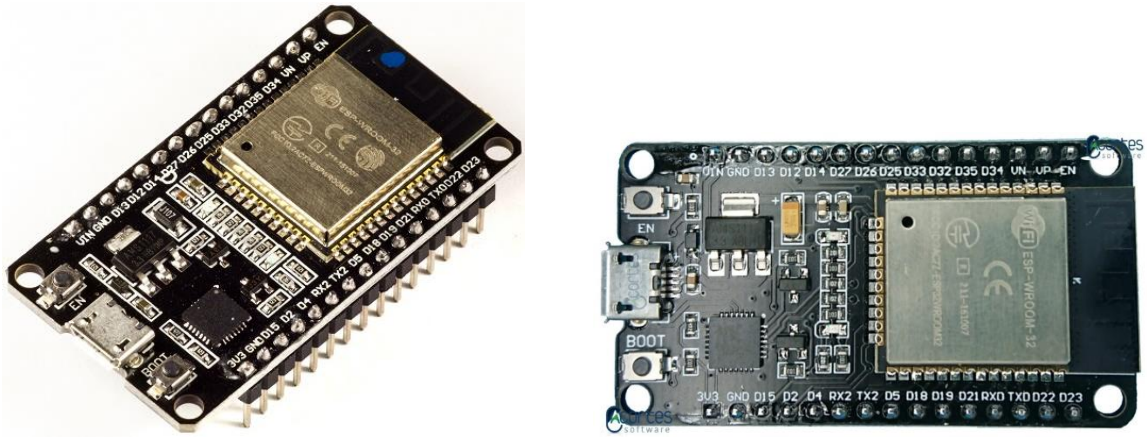


Figura 2.5. ESP32-WROOM-32 [11] [12]

Esta placa cuenta con los pines presentados en la Figura 2.6:

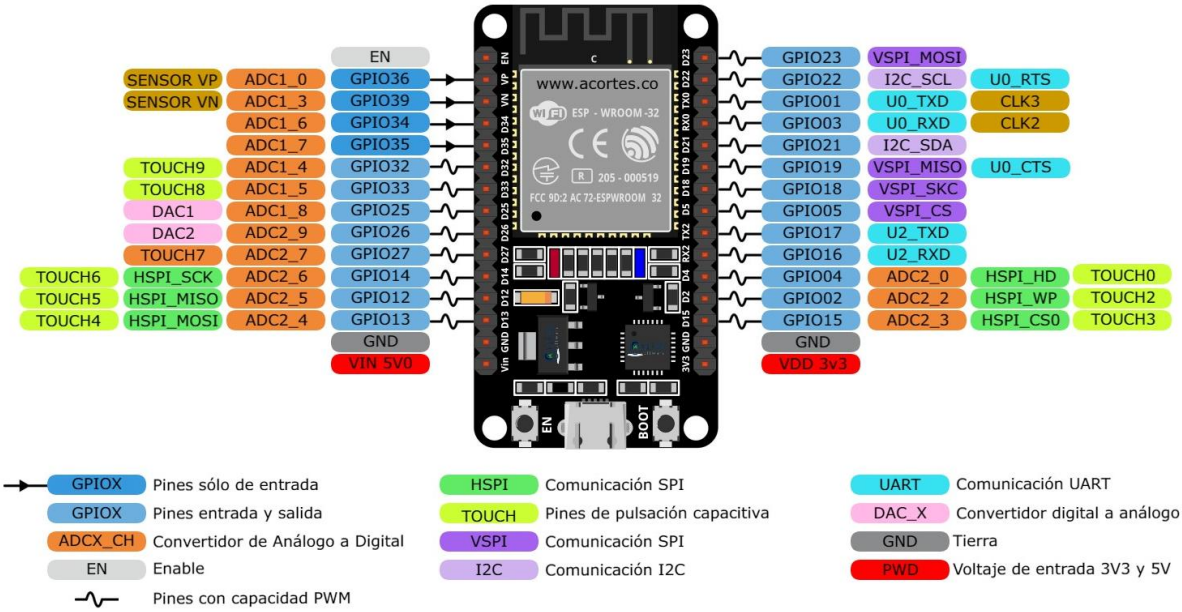


Figura 2.6. Pinout del ESP32-WROOM-32 [12]

Las características técnicas son las siguientes (Tabla 2.2):

Voltaje de Alimentación	3.3 V DC (2.7 ~ 3.6V)
Voltaje lógico entradas/salidas (GPIO)	3.3 V
Corriente de Operación	~80mA (fuente superior a 500mA)
SoM (System-On-Module)	ESP-WROOM-32
SoC (System on a Chip)	ESP32 (ESP32-D0WDQ6)
CPU	Dual core Tensilica Xtensa LX6 (32 bit)
Frecuencia de reloj	240 MHz

Memoria SRAM		520 KB
Memoria FLASH Externa		4 MB
Pines Digitales GPIO		34 (incluyendo todos los periféricos)
UART		2
SPI		3
I2C		2
Capacitive touch sensors		10
Timers		3 (16-bit)
PWM Led		16 canales independientes (16-bits)
ADC (Analog-to-Digital Converter)		2 (12-bit)
DAC (Digital-to-Analog Converter)		2 (8-bit)
Wi-Fi	Protocolos	802.11 b/g/n/e/i (802.11n hasta 150 Mbps)
	Certificación RF	FCC/CE/IC/TELEC/KCC/SRRC/NCC
	Rango de Frecuencia	2.4 ~ 2.5 GHz
	Modo	Station/SoftAP/SoftAP+Station/P2P
	Seguridad	WPA/WPA2/WPA2-Enterprise/WPS
Protocolos de Red		IPv4, IPv6, SSL, TCP/UDP/HTTP/FTP/MQTT
Bluetooth	Protocolos	V4.2 BR/EDR y especificación BLE
	Radios	Receptor NZIF con -97 dBm de sensibilidad; transmisor clase 1, 2 y 3; AFH
	Audio	CVSD and SBC
Interfaz SD y Stack de Protocolo TCP/IP integrado en hardware		Sí
Dimensiones		18 x 25.5 x 3.1 mm
Peso		3 gr.

Tabla 2.2. Características técnicas del ESP32-WROOM-32 [10]

2.3. Sensores

En los siguientes subapartados se mostrarán los sensores utilizados en el proyecto.

2.3.1. Sensor de gas SEN0132 DFRobot

Este sensor de gas es un modelo MQ-7 de los sensores de gases que sirve para medir monóxido de carbono (CO).

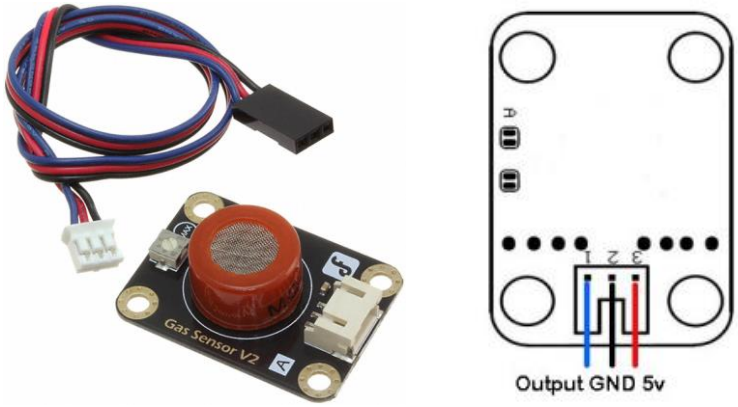


Figura 2.7. Sensor de gas analógico MQ-7 [13] [14]

En la Tabla 2.3 se muestran las especificaciones técnicas de este sensor:

Voltaje de operación del circuito	5 V
Tipo de interfaz	Analógica
Voltaje de calentamiento (alto)	5 V
Voltaje de calentamiento (bajo)	1.4 V
Tiempo de calentamiento (alto)	60 segundos
Tiempo de calentamiento (bajo)	90 segundos
Potencia de consumo	350 mW
Dimensiones	40 mm x 20 mm
Peso	15 gr.

Tabla 2.3. Características técnicas del sensor de gas analógico MQ-7 [15] [16]

Según los datos de la tabla, se puede ver que hay que alternar en bucle entre dos tensiones de alimentación:

- Primero, aplicar una tensión alta de 5V durante 60 segundos para calentar el sensor. Durante este tiempo no hay que medir o utilizar las medidas que se realicen, ya que serán incorrectas.
- Seguidamente, aplicar una tensión baja 1.4V durante 90 segundos. Estas medidas son las que hay que tener en cuenta.

El principal problema encontrado es conseguir una tensión de 1.4V e ir alternándola con la de 5V, por lo que se ha intentado realizar el código fuente para medir el valor de CO empleando únicamente 5V para calentar y 0V para medir durante el tiempo de enfriamiento. Al probar este código, se han obtenido los resultados de la Figura 2.8.

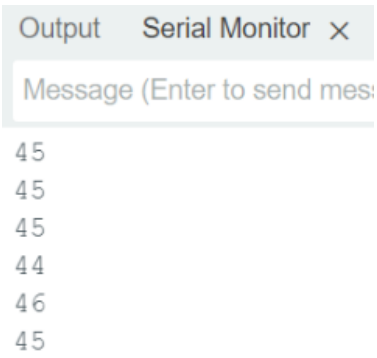


Figura 2.8. Medidas del sensor de gas analógico MQ-7

Las primeras 2 mediciones se han realizado en aire limpio. Las siguientes mediciones “45” y “44” se han obtenido echando gas con un mechero durante el tiempo de calentamiento. Por último, los valores “46” y “45” se han obtenido echando gas con un mechero al final del tiempo de enfriamiento.

Se puede observar claramente que las medidas no son nada fiables ya que apenas hay cambios, incluso cambiando la sensibilidad del sensor con el potenciómetro. Por lo tanto, se ha decidido cambiar este sensor por uno de tipo MQ-2, que se describirá en el siguiente apartado.

2.3.2. Sensor de gas analógico MQ-2

El sensor de gas analógico MQ-2 (Figura 2.9) es empleado para detectar fugas de gas en equipos tanto del mercado de consumo como en la industria. Es especialmente útil para la detección de gases como gas LPG (Gas Licuado del Petróleo, compuesto por butano y propano), metano, alcohol, hidrógeno, monóxido de carbono y humo. Destaca por su alta sensibilidad y su rápido tiempo de respuesta. Además, la sensibilidad puede ajustarse mediante un potenciómetro, incrementando la sensibilidad girando en el sentido de las agujas del reloj. [17]

En este proyecto, únicamente se utilizará este sensor para obtener datos sobre el gas LPG, CO y humo medidos.

Figura 2.9. Sensor de gas analógico MQ-2 [18]

Las siguientes especificaciones técnicas de este sensor se muestran en la Tabla 2.4:

Voltaje de operación	5 V
Corriente de operación	150 mA
Potencia de consumo	800 mW
Temperatura de operación	-10ºC ~ 50º C
Humedad de operación	< 95% RH

Tabla 2.4. Características técnicas del sensor de gas analógico MQ-2 [17]

En la Figura 2.10 se puede ver la conexión de este sensor con el Arduino Nano 33 IoT:

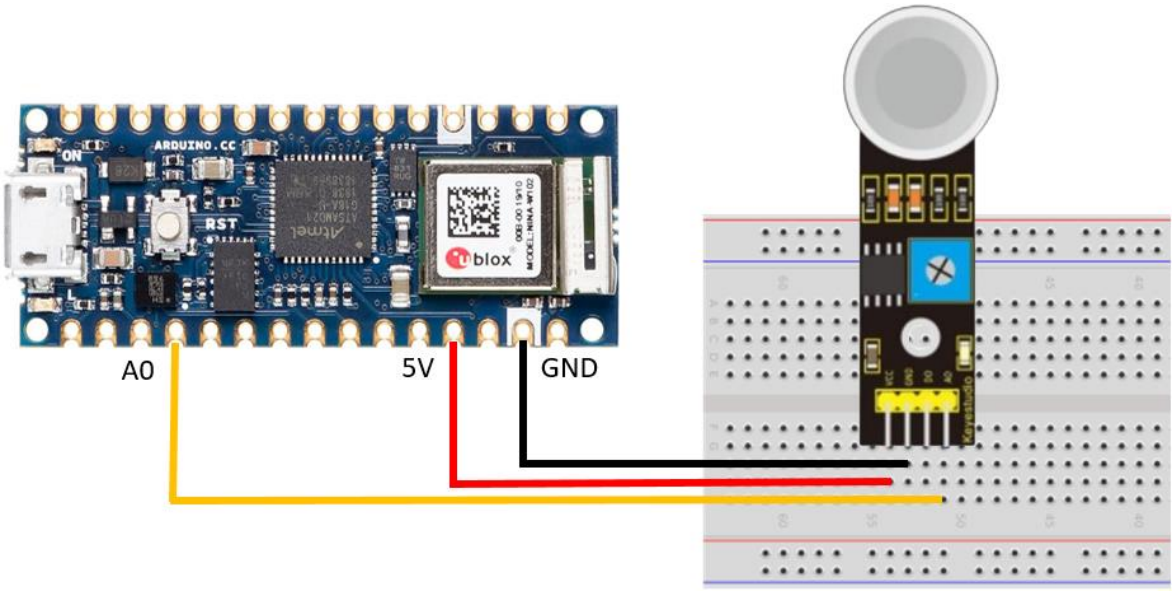


Figura 2.10. Conexión entre el sensor de gas analógico MQ-2 y el Arduino Nano 33 IoT

Como se puede ver, este sensor se conecta al pin de alimentación de 5V del Arduino y al pin de GND (tierra). Por último, se conecta el pin analógico de datos del sensor al pin analógico A0 del Arduino para la recepción de los valores medidos.

Cabe destacar que este sensor hay que precalentarlo manteniéndolo conectado a la fuente de alimentación durante aproximadamente 24 horas antes de su primer uso, ya que de no hacerlo las medidas variarían demasiado, por lo que muchas serían confusas y/o incorrectas. [19]

2.3.3. Sensor de temperatura y humedad DHT11 Módulo KY-015

El DHT11 (Figura 2.11) es un sensor digital de temperatura y humedad relativa, económico y fácil de usar. Combina un termistor y un sensor capacitivo de humedad para medir las condiciones del aire circundante, entregando los datos a través de una señal digital utilizando su pin de datos (pin S), sin disponer de una salida analógica.

En este caso, se utiliza un módulo KY-015, que es un modelo del DHT11 en el que cambia únicamente la notación al ser un sensor propio del fabricante Keyestudio. [20]

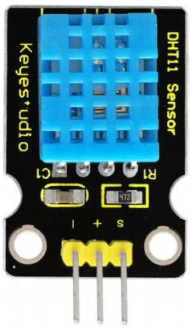


Figura 2.11. Sensor DHT11 [18]

El sensor DHT11 mide la humedad relativa, que es la cantidad de vapor de agua en el aire en comparación con la cantidad que podría contener antes de saturarse. Emplea un método para detectar el vapor de agua, basado en la medición de la resistencia eléctrica entre dos electrodos.

El componente del sensor de humedad consiste en un sustrato que retiene la humedad, con electrodos aplicados en su superficie. Cuando el sustrato absorbe vapor de agua, libera iones que incrementan la conductividad entre los electrodos. Este cambio en la resistencia entre los electrodos es directamente proporcional a la humedad relativa: una humedad relativa más alta reduce la resistencia, mientras que una humedad relativa más baja la incrementa.

En lo que respecta a la medición de la temperatura, el DHT11 utiliza un sensor termistor NTC (Coeficiente de Temperatura Negativo) integrado en su mismo ensamblaje. [21]

Las características técnicas de este sensor son las siguientes (Tabla 2.5):

Voltaje de operación	3-5 V
Rango de medición de temperatura	0 a 50º C con ±2ºC de error
Rango de medición de humedad relativa	20%-90% con ±5% de error
Frecuencia de muestreo	< 1 Hz (una vez por segundo)
Interfaz digital	Single-bus (bidireccional)
Dimensiones	16 mm x 12 mm x 5 mm
Peso	1 gr.

Tabla 2.5. Características técnicas del módulo sensor DHT11 [20]

La versión empleada en este proyecto está montada en una PCB (Placa de Circuito Impreso) e incorpora una resistencia de 10 kΩ soldada en superficie para la línea de señal. Esto elimina la necesidad de añadir resistencias externas en el pin de datos.

En la Figura 2.12 se puede ver la conexión de este sensor con el ESP32-WROOM-32:

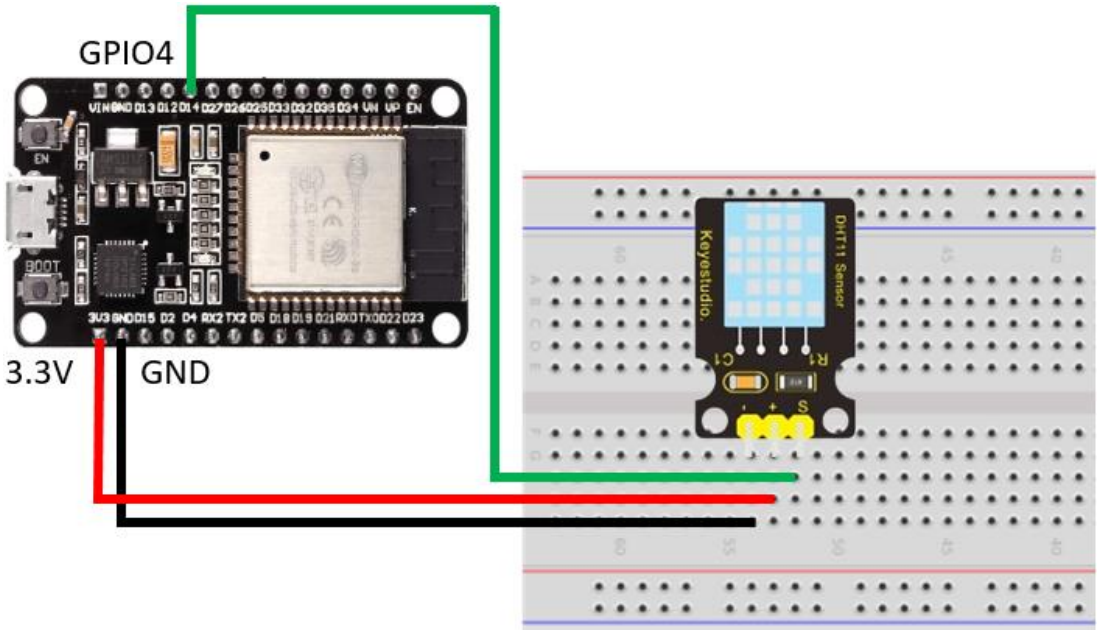


Figura 2.12. Conexión entre el sensor DHT11 y el ESP32-WROOM-32

Como se puede observar, el sensor se conecta al pin de alimentación de 3.3V del ESP32 y al pin de GND (tierra). Por último, se conecta el pin de datos del sensor al pin GPIO 4 del ESP32 para la transmisión de datos.

2.4. Conclusión

En este punto, cada placa microcontroladora está conectada de forma individual a un sensor. En los siguientes apartados se explicará cómo se configuran, se programan y se integran en un único sistema de telemetría.

Capítulo 3. Software del sistema

En este capítulo se describirá qué es la plataforma “Arduino Cloud” y su aplicación para móvil asociada “IoT Remote”. Además, se comentarán los Things creados para ambas placas en esta plataforma.

3.1. Arduino Cloud

Arduino es conocido por fabricar placas y proporcionar un entorno de desarrollo (Arduino IDE) que facilita enormemente la configuración, programación e integración de todo tipo de dispositivos electrónicos.

Sin embargo, Arduino es mucho más que eso. Dispone de Arduino Cloud (Figura 3.1), una plataforma de gestión de dispositivos IoT en la nube. Permite crear paneles de control para monitorizar y controlar los dispositivos de forma remota.

Lo más destacado es que Arduino Cloud no solo es compatible con proyectos creados en Arduino, sino también con hardware ESP y dispositivos basados en Linux, como Raspberry Pi. [22]

Además, ofrece varios métodos de interacción, incluyendo protocolos como HTTP, API REST, MQTT, herramientas de línea de comandos, Websockets y JavaScript, entre otros.

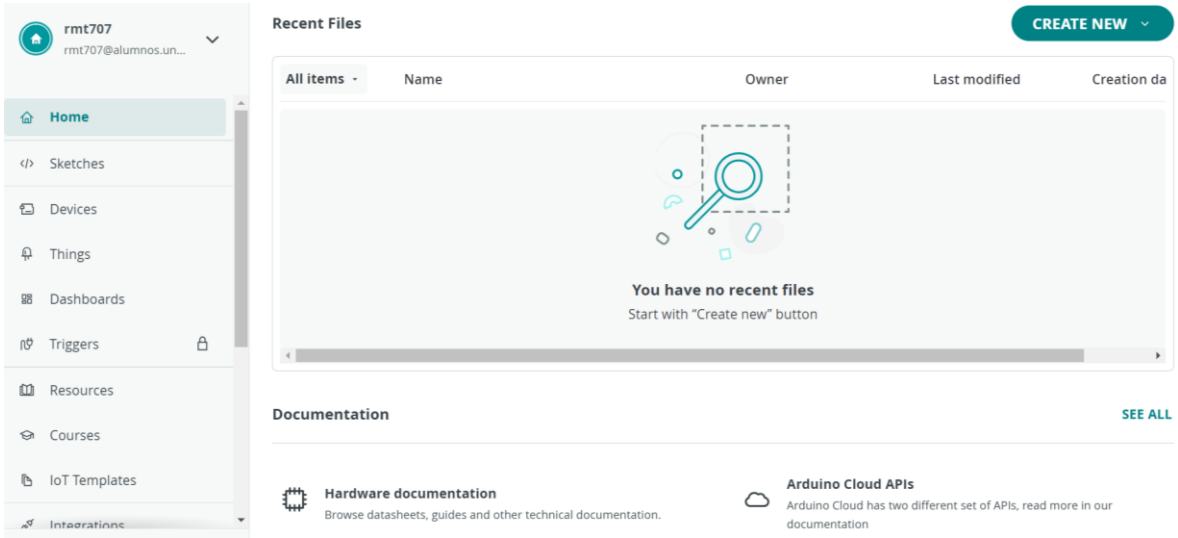


Figura 3.1. Plataforma Arduino Cloud

Dentro de esta plataforma, se encuentran distintas secciones:

- Al pinchar en el nombre de la cuenta (Figura 3.2), se puede acceder a la configuración, facturación y otras configuraciones.

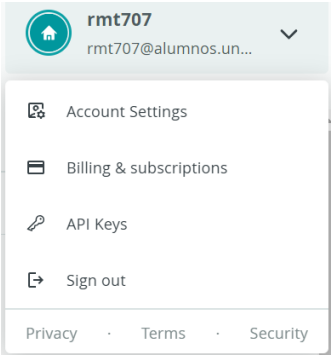


Figura 3.2. Cuenta en Arduino Cloud

- En “Home” se encuentra el centro de archivos recientes, estado del dispositivo y un enlace directo a la documentación de Arduino.
- En “Sketches” se muestran los códigos creados en el editor de Arduino online asociado a la cuenta de Arduino Cloud.
- Seguidamente, aparece el entorno de IoT (Figura 3.3), donde se pueden agregar dispositivos, crear *Things* (gemelo virtual del dispositivo de hardware), paneles y definir activadores.

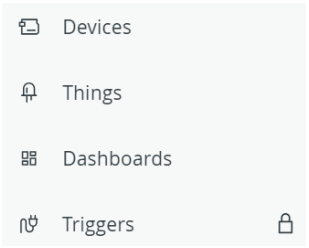


Figura 3.3. Entorno de IoT en Arduino Cloud

- Debajo de esta sección, se encuentran materiales de aprendizaje para Arduino e IoT (Figura 3.4), entre los que se encuentran cursos y plantillas.

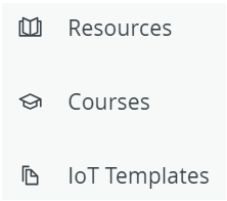


Figura 3.4. Materiales de aprendizaje en Arduino Cloud

En los apartados 3.2 y 3.3 se explicará cómo se han creado los Things para cada placa.

3.1.1. Arduino Create Agent

Es importante saber que, para conectar los dispositivos a IoT Cloud, es necesario instalar la última versión disponible de un software adicional llamado “Arduino Create Agent”. Este software permite que la plataforma reconozca las placas conectadas al puerto USB, lo que posibilita cargar los programas y ver la información transmitida a través del puerto serie directamente. [23]

Los pasos para la instalación de este agente de Arduino se muestra en el Anexo I.

3.2. Arduino Nano 33 IoT

3.2.1. Thing “TFM_rmt”

En los siguientes subapartados se explicará cómo se ha creado el Thing enlazado con el Arduino Nano 33 IoT. Para ello, se crearán variables para almacenar los valores de cada gas medido, se desarrollará un código para realizar la medición de estos datos y se representarán en el Dashboard de Arduino Cloud.

3.2.2. Setup

Primero, se crea un Thing desde el menú “Home” pulsando en “Create new” y seguidamente en “Thing” (Figura 3.5).

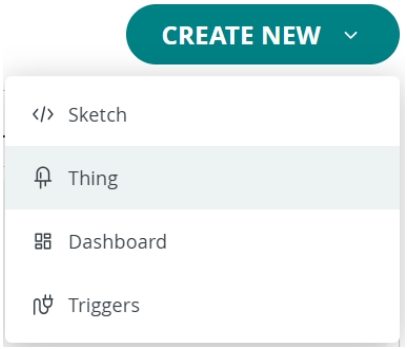


Figura 3.5. Menú “Home” en Arduino Cloud

En la Figura 3.6 se muestra el panel de Thing con los parámetros a configurar.

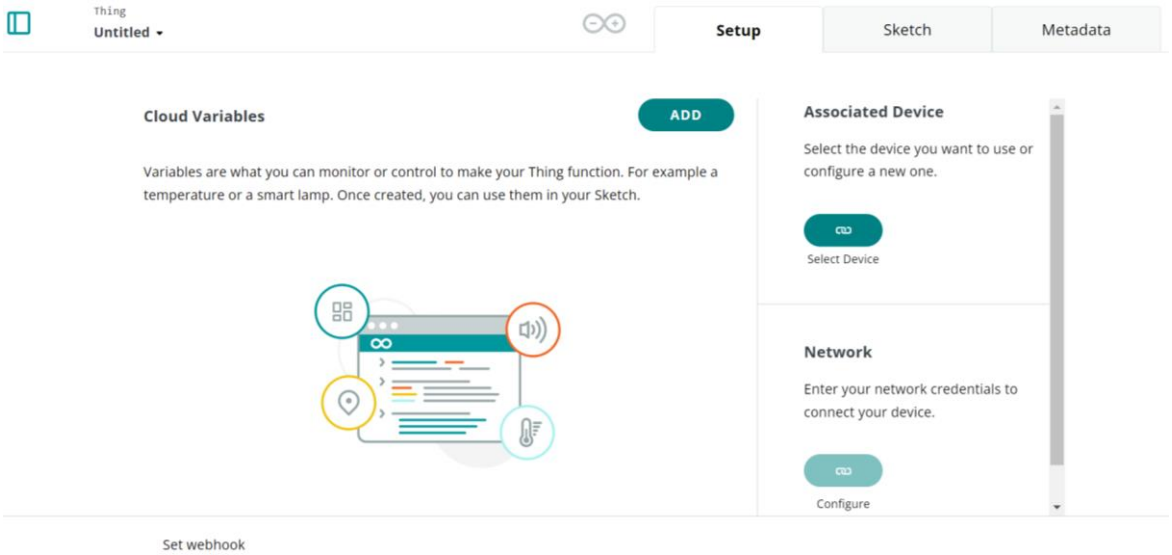


Figura 3.6. Panel “Thing” en Arduino Cloud

Por lo tanto, se empieza cambiando el nombre del Thing a “TFM_rmt”. Seguidamente, se selecciona el dispositivo a asociar, conectando el Arduino Nano 33 IoT al puerto serie para que pueda ser detectado (Figura 3.7).

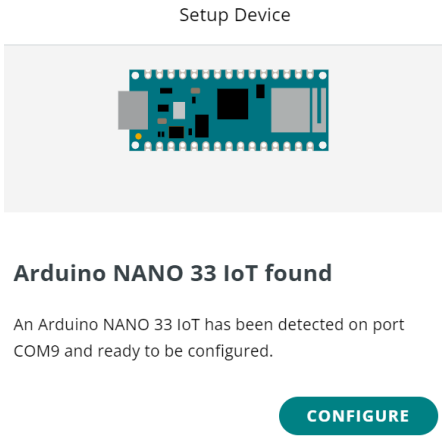


Figura 3.7. Detección del Arduino Nano 33 IoT

A continuación, se le configurará dándole el nombre “TFM” (Figura 3.8).

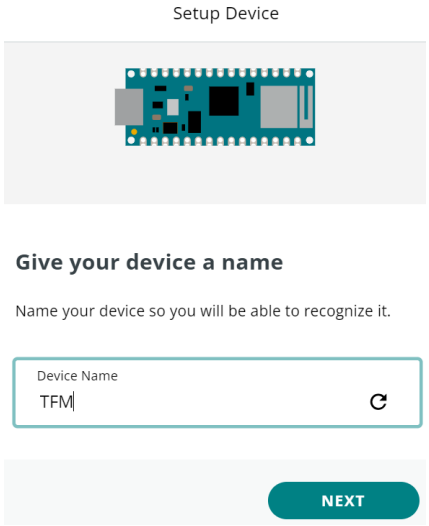


Figura 3.8. Nombre del Arduino Nano 33 IoT

Tras ello, aparece en el panel principal como dispositivo asociado (Figura 3.9).

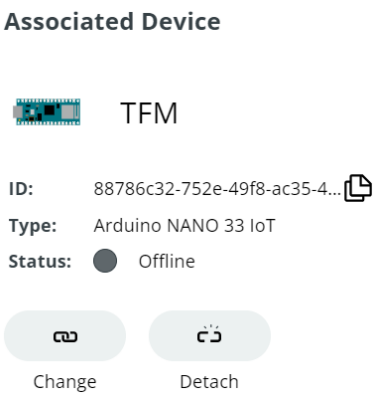


Figura 3.9. Arduino Nano 33 IoT como dispositivo asociado al Thing “TFM_rmt”

Después, se configuran las credenciales de la red Wi-Fi a las que se conectará el dispositivo (Figura 3.10).

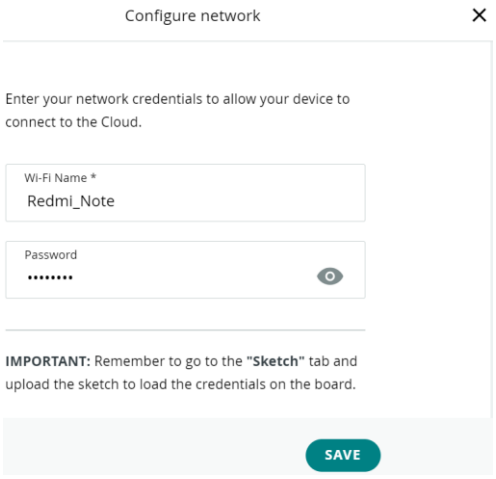
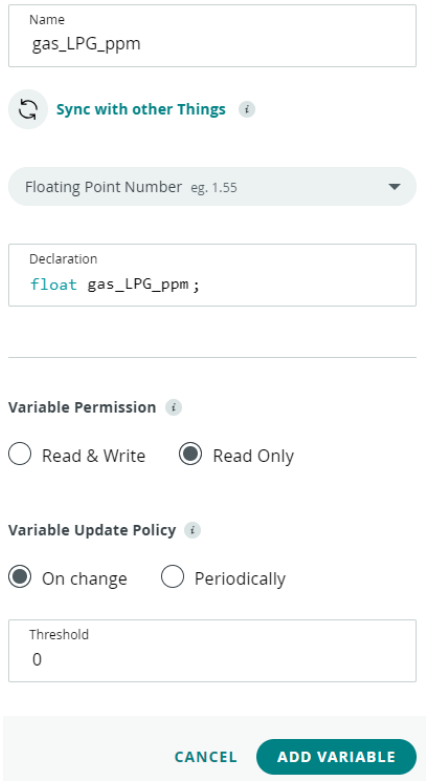
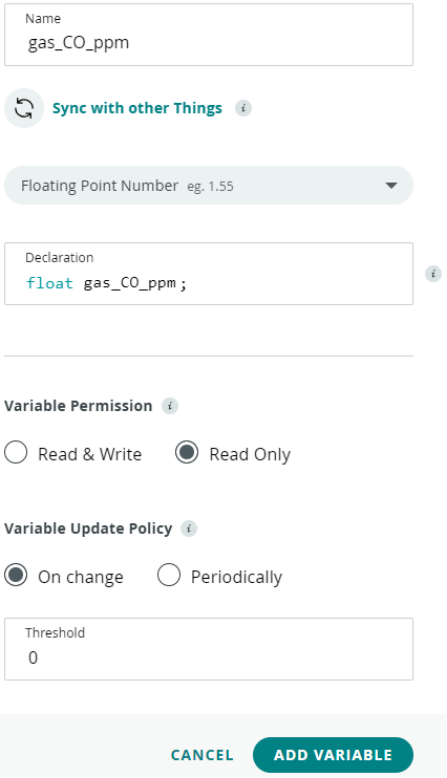


Figura 3.10. Credenciales de la red Wi-Fi del Thing "TFM_rmt"

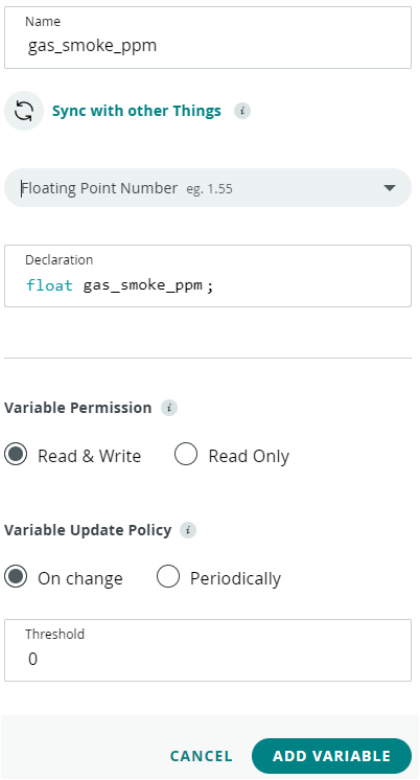
Por último, se deben crear las variables cuyos valores se quieren almacenar en la nube. En el caso de esta placa, como va a tener conectado el sensor de detección de gases MQ-2, se crearán las variables "gas_LPG_ppm", "gas_CO_ppm" y "gas_smoke_ppm" (Figura 3.11). Estas variables se configuran para que sean de solo lectura y que se actualicen solo cuando cambie el valor medido, en lugar de actualizarse periódicamente.



a) Variable para LPG



b) Variable para CO



c) Variable para humo

Figura 3.11. Variables del Thing "TFM_rmt"

Por lo tanto, el "Setup" del Thing "TFM_rmt" quedaría como se muestra en la Figura 3.12.

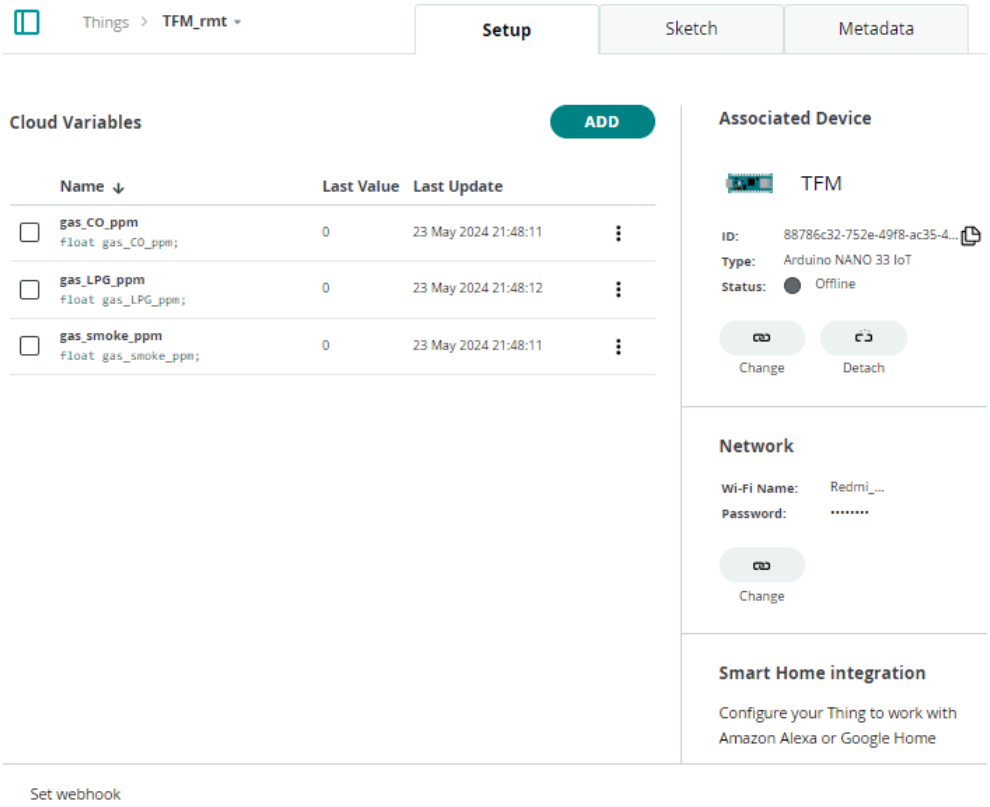


Figura 3.12. Setup final del Thing "TFM_rmt"

3.2.3. Código fuente para el sensor de gas MQ-2

A continuación, se explicará el código fuente empleado para medir los valores de LPG, CO y humo de este sensor guardándolos en variables globales, cuyos valores se mostrarán en el Dashboard de Arduino Cloud. Aunque antes habrá que realizar dos códigos para calcular el valor de dos variables necesarias que se utilizarán en el código final, como se explica a continuación.

En este código se tendrá en cuenta el diagrama de la Figura 3.13 extraído del datasheet. En este, se tendrá en cuenta que R0 es la resistencia del sensor en una concentración conocida (aire ambiente) y Rs es la resistencia del sensor que varía dependiendo de la concentración de un determinado gas. [24]

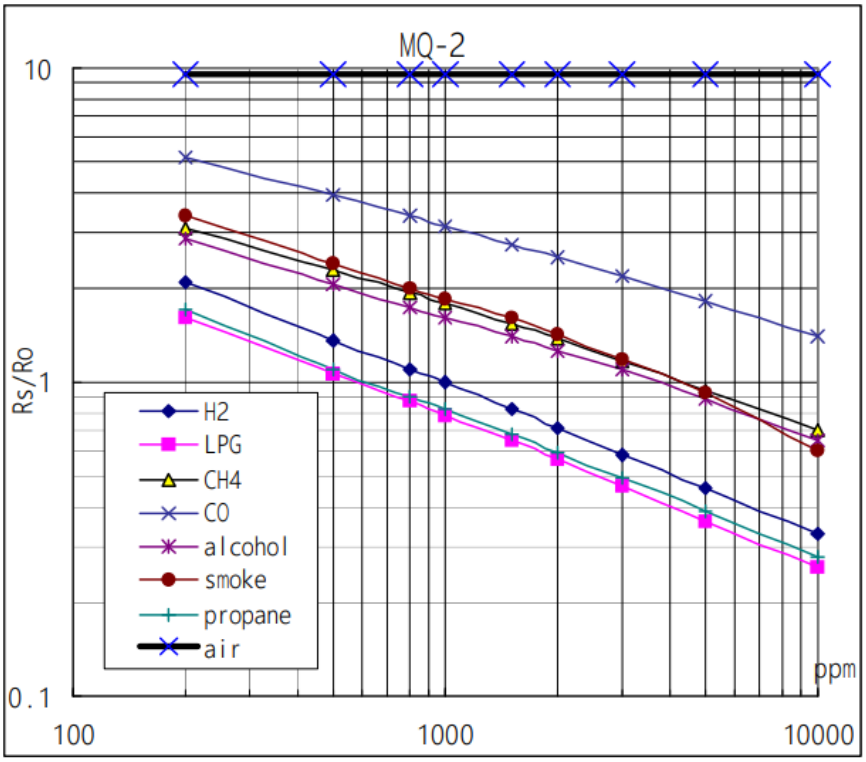


Figura 3.13. Características de sensibilidad del MQ-2 [19]

Obteniendo de la gráfica la relación de Rs/R0 = 9.8 para el aire. Este valor se utilizará para obtener la resistencia R0 para aire limpio, sabiendo que la resistencia de carga del módulo es RL = 1 kΩ. El código para calcular esta resistencia se puede ver en la Figura 3.14.

```
1   const int MQ2_PIN = A0;
2   const float CLEAN_AIR_RATIO = 9.80; //Grafica datasheet RS/R0
3
4   void setup() {
5     Serial.begin(9600);
6   }
7
8   void loop() {
9     float MQ2_volt;
10    float RS_air; //kohm, en aire limpio
11    float R0; //kohm
12    float MQ2_value;
13
14    //Dato promedio (100 veces)
15    for(int x = 0; x < 100; x++){
16      MQ2_value = MQ2_value + analogRead(MQ2_PIN);
17    }
18    MQ2_value = MQ2_value / 100.0; //valor 0-1023
19
20    //Se transforma a voltaje
21    MQ2_volt = MQ2_value / 1024 * 5.0; //Se omite RL porque RL = 1kohm
22    RS_air = (5.0 - MQ2_volt) / MQ2_volt; //RL*(1023 - analogRead(MQ2_PIN))/analogRead(MQ2_PIN)
23    R0 = RS_air / CLEAN_AIR_RATIO;
24
25    Serial.print("MQ2_volt = ");
26    Serial.print(MQ2_volt);
27    Serial.println("V");
28
29    Serial.print("R0 = ");
30    Serial.println(R0);
31    delay(1000);
32  }
```

Figura 3.14. Código fuente para calcular el valor de R0 en aire limpio [24]

En ese código se ve que se inicializan las constantes del pin del sensor y del ratio Rs/R0 (líneas 1 y 2). Tras ello, se inicializa el monitor serie en la función setup (líneas 4-6).

Luego, se definen las variables que se van a calcular a lo largo del código dentro del loop. Para ello, se realizará un promedio de las medidas del sensor de gas y se transformarán a voltaje (líneas 15-21). Con ello, se obtiene Rs (línea 22) con la siguiente fórmula:

$$\frac{RS}{RL} = \frac{V - VS}{VS}$$

Por último, se obtiene el valor de R0 en aire limpio (línea 23), ya que se ha calculado el valor de RS en aire limpio y se conoce el valor de la relación de RS/R0.

Al subir el sketch a la placa, hay que esperar a que se estabilice el valor de R0, el cual se usará en el siguiente sketch. En la Figura 3.15 se puede observar cómo varía el valor de R0, por lo que se elegirá R0 = 0.69 kΩ al ser el valor más repetido.

```
MQ2_volt = 0.64V
R0 = 0.69
MQ2_volt = 0.66V
R0 = 0.67
MQ2_volt = 0.64V
R0 = 0.70
MQ2_volt = 0.65V
R0 = 0.69
MQ2_volt = 0.64V
R0 = 0.70
MQ2_volt = 0.65V
R0 = 0.68
MQ2_volt = 0.64V
R0 = 0.69
MQ2_volt = 0.64V
R0 = 0.70
```

Figura 3.15. Obtención del valor de R0 en aire limpio

Seguidamente, hay que implementar otro código para calcular la relación de Rs_gas/R0, utilizando el valor de R0 calculado anteriormente.

Como se puede observar en la Figura 3.16, el código es muy parecido al anterior. La única diferencia es que se sustituye el valor de R0 en lugar de utilizar como referencia el ratio Rs/R0 obtenido en la gráfica (Figura 3.13).

```
1   const int MQ2_PIN = A0;
2   const float CLEAN_AIR_RATIO = 9.80; //Grafica datasheet RS/R0
3   const float R0 = 0.69; //kohm, valor obtenido en el anterior sketch
4
5   void setup() {
6     Serial.begin(9600);
7   }
8
9   void loop() {
10    float MQ2_volt;
11    float RS_gas; //kohm, Obtencion con gas
12    float ratio; //Obtencion de ratio RS_GAS/RS_air
13    int MQ2_value = analogRead(MQ2_PIN);
14
15    MQ2_volt = (float)MQ2_value/1024*5.0;
16    RS_gas = (5.0 - MQ2_volt) / MQ2_volt; //Se omite RL porque RL = 1kohm
17
18    ratio = RS_gas / R0; //ratio = RS/R0
19
20    Serial.print("MQ2_volt = ");
21    Serial.println(MQ2_volt);
22    Serial.print("RS_ratio = ");
23    Serial.println(RS_gas);
24    Serial.print("RS/R0 = ");
25    Serial.println(ratio);
26    delay(1000);
27    Serial.print("\n\n");
28  }
```

Figura 3.16. Código fuente para calcular la relación de Rs_gas/R0 [24]

Se sube el sketch a la placa y se espera a que se estabilice el valor de Rs/R0. En la Figura 3.17 se puede observar que el valor de este ratio es de 9.79, el cual se aproxima bastante al obtenido en la gráfica en el caso del aire limpio, ya que era de 9.8.

```
MQ2_volt = 0.64
RS_ratio = 6.76
RS/R0 = 9.79


MQ2_volt = 0.64
RS_ratio = 6.76
RS/R0 = 9.79


MQ2_volt = 0.64
RS_ratio = 6.76
RS/R0 = 9.79
```

Figura 3.17. Obtención del valor de Rs/R0

Queda realizar un sketch para calcular las partes por millón (ppm), que es la relación de un gas hacia otro gas. Se puede ver un ejemplo de esta relación en la Figura 3.18.

Figura 3.18. Ejemplo de partes por millón

Por lo tanto, el código final utilizado en Arduino Cloud se muestra en las Figuras 3.19, 3.20, 3.21 y 3.22.

Antes del setup (Figura 3.19), se incluyen las variables globales (líneas 9-11) y se sustituye el valor del ratio calculado en el anterior código (línea 23). Tras ello, se declaran unas variables para el tiempo de muestreo (líneas 26-30), que se utilizarán en las funciones de cálculo y calibración. Luego, se definen índices para cada gas (líneas 34-36). Seguidamente, se declaran datos de las curvas de los gases que se obtienen de la gráfica presentada en la Figura 3.13, el formato es {x, y, pendiente} (líneas 40-42).

```
1   /*
2     Sketch generated by the Arduino IoT Cloud Thing "Untitled"
3     https://create.arduino.cc/cloud/things/295a7d83-e1d6-4305-b0a1-b54cc008db87
4
5     Arduino IoT Cloud Variables description
6
7     The following variables are automatically generated and updated when changes are made to the Thing
8
9     float gas_CO_ppm;
10    float gas_LPG_ppm;
11    float gas_smoke_ppm;
12
13    Variables which are marked as READ/WRITE in the Cloud Thing will also have functions
14    which are called when their values are changed from the Dashboard.
15    These functions are generated with the Thing and added at the end of this sketch.
16  */
17  #include "thingProperties.h"
18
19  // HARDWARE
20  const int calibrationLed = 13; // Se encendera durante la calibracion
21  const int MQ2_PIN = A0;
22  int RL_value = 1; //kohm
23  float R0_CLEAN_AIR_FACTOR = 9.79; // Ratio calculado en el anterior sketch
24
25  // SOFTWARE
26  int CALIBRATION_SAMPLE_TIMES = 50;
27  int CALIBRATION_SAMPLE_INTERVAL = 500;
28
29  int READ_SAMPLE_INTERVAL = 50;
30  int READ_SAMPLE_TIMES = 5;
31
32  #include <WiFiNINA.h>
33
34  #define GAS_LPG 0
35  #define GAS_CO 1
36  #define GAS_SMOKE 2
37
38  // Se toman 2 puntos de la curva de la grafica y se pone el formato:
39  // {x,y,pendiente}
40  float LPG_curve[3] = {2.3, 0.21, -0.47};
41  float CO_curve[3] = {2.3, 0.72, -0.34};
42  float Smoke_curve[3] = {2.3, 0.53, -0.44};
43
44  float R0 = 10; //R0 se inicializa a 10kohm
45
46  WiFiClient client; // Crea el cliente para Wi-Fi
```

Figura 3.19. Código fuente para calcular el valor de los gases en ppm – Parte 1 [24]

En el setup (Figura 3.20), se calcula R0 con la función de calibración “MQCalibration” (línea 55). Esta función calcula el valor de R0 llamando a la función “MQResistanceCalculation” (línea 108). Esta función toma múltiples lecturas analógicas del sensor, calcula la resistencia correspondiente para cada lectura llamando a la función “MQResistanceCalculaion”, promedia las resistencias calculadas y ajusta el promedio utilizando el factor de aire limpio.

```
48  void setup() {
49    Serial.begin(9600);
50
51    pinMode(calibrationLed, OUTPUT);
52    digitalWrite(calibrationLed, HIGH);
53    Serial.print("Calibrando...");
54
55    R0 = MQCalibration(MQ2_PIN);
56    digitalWrite(calibrationLed, LOW);
57    Serial.println("Terminado!");
58    Serial.print("R0 = ");
59    Serial.print(R0);
60    Serial.println("kohm\n");
61
62    delay(1500);
63
64    // Defined in thingProperties.h
65    initProperties();
66
67    // Connect to Arduino IoT Cloud
68    ArduinoCloud.begin(ArduinoIoTPreferredConnection);
69    setDebugMessageLevel(2);
70    ArduinoCloud.printDebugInfo();
71  }
```

```
98  float MQResistanceCalculation(int raw_adc){
99    return ( ((float)RL_value*(1023-raw_adc)/raw_adc));
100 }
101
102
103 float MQCalibration(int mq_pin){
104   int i;
105   float val = 0;
106
107   for(i = 0; i < CALIBRATION_SAMPLE_TIMES; i++){
108     val += MQResistanceCalculation(analogRead(mq_pin));
109     delay(CALIBRATION_SAMPLE_INTERVAL);
110   }
111   val = val / CALIBRATION_SAMPLE_TIMES;
112   val = val / R0_CLEAN_AIR_FACTOR;
113
114   return val;
115 }
```

Figura 3.20. Código fuente para calcular el valor de los gases en ppm – Parte 2 [24]

En el loop, se obtienen los porcentajes de los gases (líneas 76-78) utilizando la función “MQGetGasPercentage”, que a su vez llama a la función “MQRead”. Esta función se utiliza para leer la resistencia del sensor, tomando múltiples muestras y calculando el promedio de las resistencias medidas para obtener un valor más preciso.

```
73  void loop() {
74    ArduinoCloud.update();
75
76    gas_LPG_ppm = MQGetGasPercentage(MQRead(MQ2_PIN)/R0, GAS_LPG);
77    gas_CO_ppm = MQGetGasPercentage(MQRead(MQ2_PIN)/R0, GAS_CO);
78    gas_smoke_ppm = MQGetGasPercentage(MQRead(MQ2_PIN)/R0, GAS_SMOKE);
79
80    Serial.println("********** CONCENTRACION DE GASES **********");
81    Serial.print("LPG: ");
82    Serial.print(gas_LPG_ppm);
83    Serial.println(" ppm");
84
85    Serial.print("CO: ");
86    Serial.print(gas_CO_ppm);
87    Serial.println(" ppm");
88
89    Serial.print("Smoke: ");
90    Serial.print(gas_smoke_ppm);
91    Serial.println(" ppm");
92    Serial.println("***********************************************");
93    Serial.println();
94    delay(500);
95  }
```

```
118 float MQRead(int mq_pin){
119   int i;
120   float rs = 0;
121
122   for(i = 0; i < READ_SAMPLE_TIMES; i++){
123     rs += MQResistanceCalculation(analogRead(mq_pin));
124     delay(READ_SAMPLE_INTERVAL);
125   }
126   rs = rs / READ_SAMPLE_TIMES;
127
128   return rs;
129 }
130
131
```

Figura 3.21. Código fuente para calcular el valor de los gases en ppm – Parte 3 [24]

La función “MQGetGasPercentage” (Figura 3.22) se encargará de identificar el gas para enviar los valores correctos a la función “MQGetPercentage” (Figura 3.22), la cual utilizará los datos de la curva de calibración de cada gas para convertir la relación de resistencias en una concentración de gas en ppm.

```
132  long MQGetGasPercentage(float rs_ro_ratio, int gas_id){
133    if(gas_id = GAS_LPG){
134      return MQGetPercentage(rs_ro_ratio, LPG_curve);
135    } else if(gas_id = GAS_CO){
136      return MQGetPercentage(rs_ro_ratio, CO_curve);
137    } else if(gas_id = GAS_SMOKE){
138      return MQGetPercentage(rs_ro_ratio, Smoke_curve);
139    }
140
141    return 0;
142  }
143
144
145  //Calculo de PPM
146  long MQGetPercentage(float rs_ro_ratio, float *pcurve){
147    //ppm = 10^[(log(Rs/R0) - y)/pendiente + x]
148    return (pow(10,(((log(rs_ro_ratio)-pcurve[1])/pcurve[2])+pcurve[0])));
149  }
```

Figura 3.22. Código fuente para calcular el valor de los gases en ppm – Parte 4 [24]

En la Figura 3.23 se puede ver resumido el funcionamiento de este código en un diagrama de flujo.

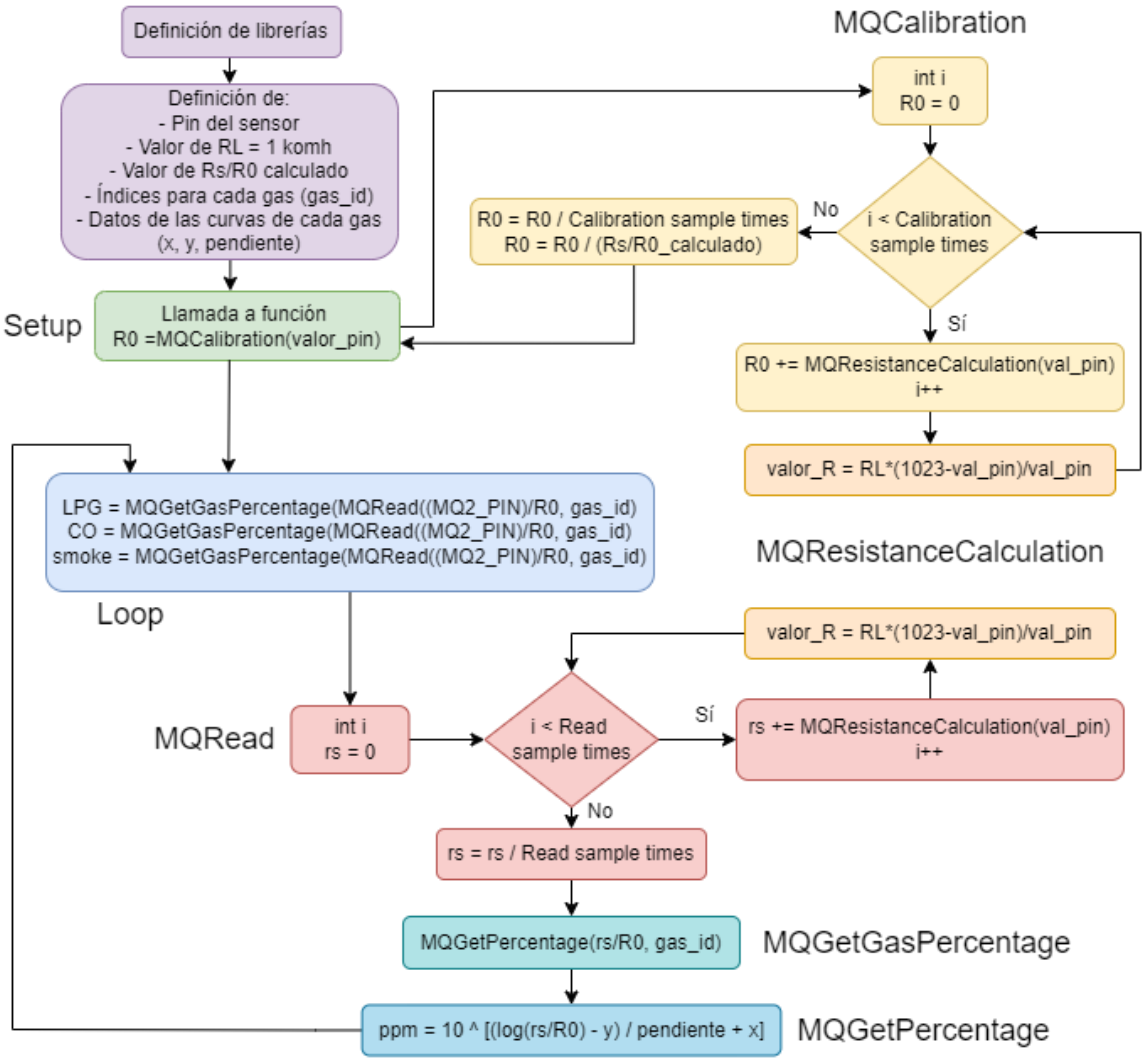


Figura 3.23. Diagrama de flujo del código fuente para calcular el valor de los gases en ppm

Al subir el código al Arduino, se puede comprobar que funciona correctamente para los casos en los que no hay gas (Figura 3.24) y cuando se echa gas, por ejemplo, utilizando un mechero (Figura 3.25).

```
*********** CONCENTRACION DE GASES ***********
LPG: 0 ppm
CO: 0 ppm
Smoke: 0 ppm
********************************************

*********** CONCENTRACION DE GASES ***********
LPG: 0 ppm
CO: 0 ppm
Smoke: 0 ppm
********************************************

*********** CONCENTRACION DE GASES ***********
LPG: 0 ppm
CO: 0 ppm
Smoke: 0 ppm
********************************************
```

Figura 3.24. Medición de los valores de los gases en aire limpio

```
*********** CONCENTRACION DE GASES ***********
LPG: 0 ppm
CO: 0 ppm
Smoke: 0 ppm
********************************************

*********** CONCENTRACION DE GASES ***********
LPG: 501 ppm
CO: 3365 ppm        (1)
Smoke: 6602 ppm
********************************************

*********** CONCENTRACION DE GASES ***********
LPG: 7823 ppm
CO: 5703 ppm
Smoke: 4145 ppm
********************************************

*********** CONCENTRACION DE GASES ***********
LPG: 1917 ppm
CO: 990 ppm
Smoke: 646 ppm
********************************************

*********** CONCENTRACION DE GASES ***********
LPG: 244 ppm
CO: 209 ppm
Smoke: 152 ppm
********************************************

*********** CONCENTRACION DE GASES ***********
LPG: 38 ppm
CO: 51 ppm
Smoke: 28 ppm
********************************************
```

```
*********** CONCENTRACION DE GASES ***********
LPG: 18 ppm
CO: 22 ppm
Smoke: 16 ppm
********************************************

*********** CONCENTRACION DE GASES ***********
LPG: 4 ppm
CO: 5 ppm
Smoke: 4 ppm
********************************************

*********** CONCENTRACION DE GASES ***********
LPG: 3 ppm
CO: 2 ppm
Smoke: 1 ppm
********************************************

*********** CONCENTRACION DE GASES ***********
LPG: 0 ppm
CO: 1 ppm
Smoke: 0 ppm
********************************************

*********** CONCENTRACION DE GASES ***********
LPG: 0 ppm
CO: 0 ppm        (2)
Smoke: 0 ppm
********************************************

*********** CONCENTRACION DE GASES ***********
LPG: 0 ppm
CO: 0 ppm
Smoke: 0 ppm
********************************************
```

Figura 3.25. Evolución de la medición de los valores de los gases en aire contaminado

En la Figura 3.25, teniendo en cuenta que cada medida emplea 750 milisegundos en realizarse, se puede ver que, en este caso, tarda unos 7.5 segundos en evaporarse todo el gas nocivo desde (1) hasta (2), por lo que se puede seguir correctamente la evolución de la presencia de los gases en el ambiente.

3.2.4. Dashboard

Por último, en el Dashboard de Arduino Cloud se creará un widget de tipo “Chart” para visualizar los valores de cada variable (Figura 3.29) en un diagrama en el que se puede elegir la antigüedad de evolución de los valores. Para ello, se enlazarán las variables a cada widget seleccionando el Thing al que pertenecen (Figuras 3.26, 3.27 y 3.28).

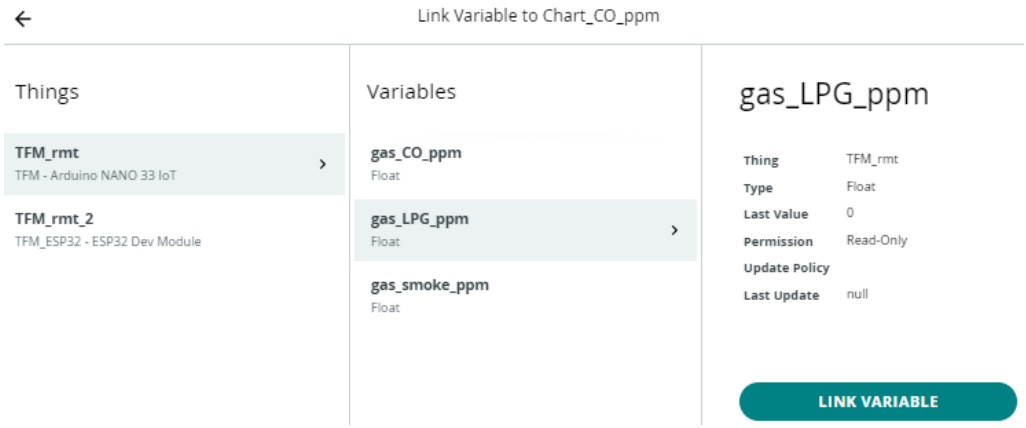


Figura 3.26. Variable enlazada del widget de gas LPG

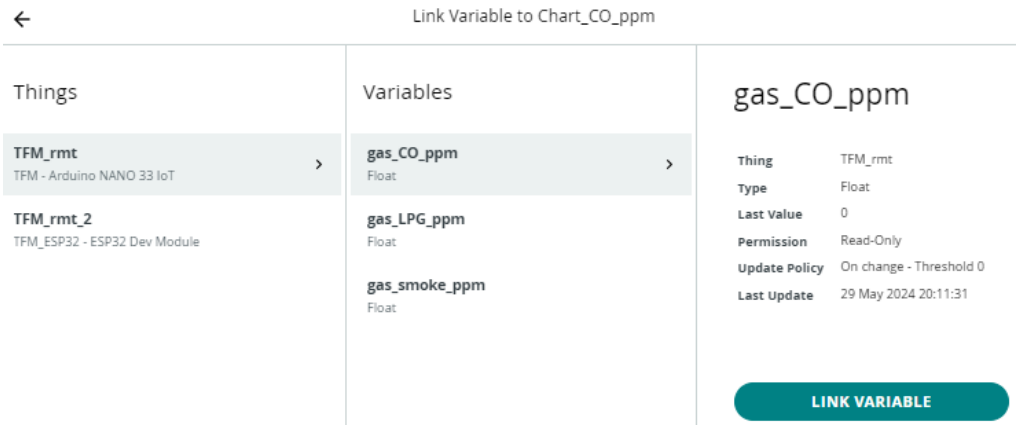


Figura 3.27. Variable enlazada del widget de gas CO

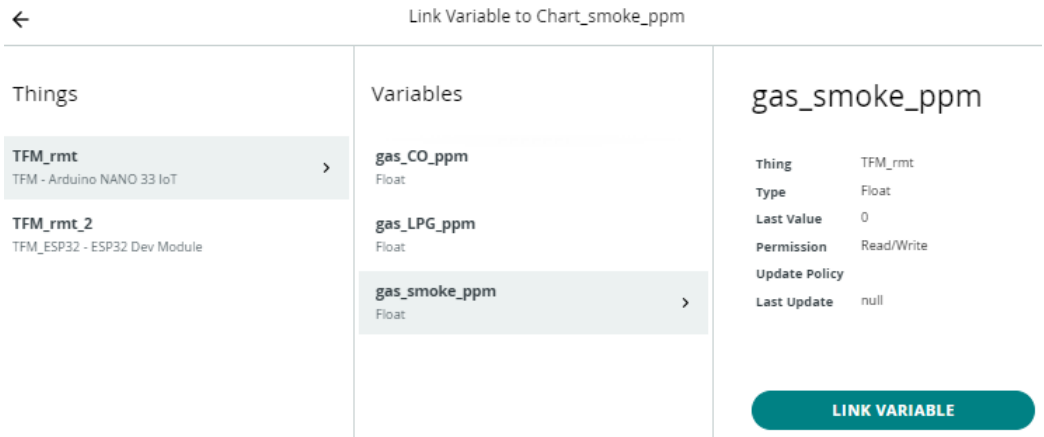


Figura 3.28. Variable enlazada del widget de humo

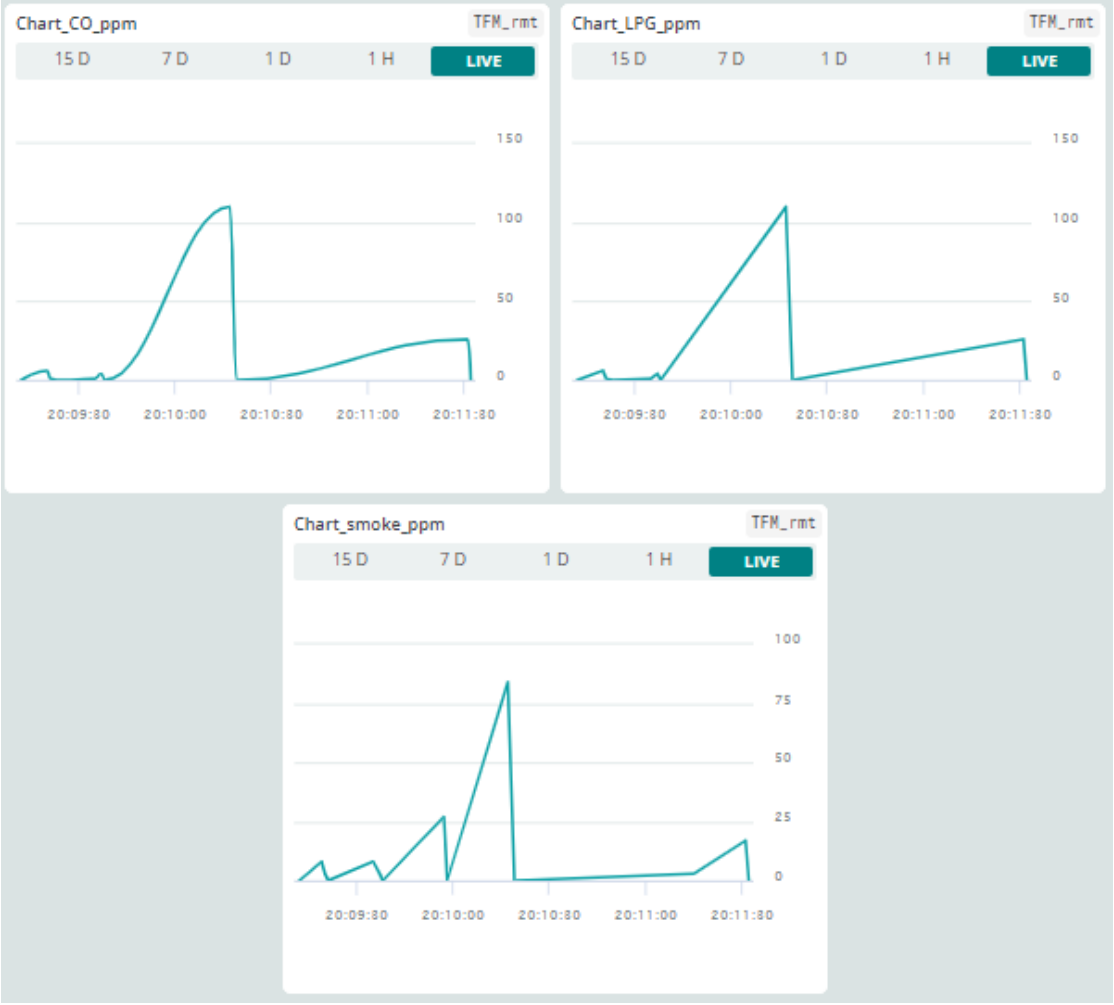


Figura 3.29. Dashboard con valores del sensor de gas MQ-2

3.3. ESP-32-WROOM-32

3.3.1. Thing “TFM_rmt_2”

En los siguientes subapartados se explicará cómo se ha creado el Thing enlazado con el ESP-32-WROOM-32. Para ello, se crearán variables para almacenar los valores de la humedad y la temperatura. Seguidamente, se desarrollará un código para realizar la medición de estos datos y representarlos en el Dashboard de Arduino Cloud.

3.3.2. Setup

Lo siguiente es crear otro Thing que se encargue de las medidas de la humedad y temperatura del sensor DHT11. Para ello, se creará un nuevo dispositivo seleccionando “+ DEVICE” desde el menú “Devices” (Figura 3.30).

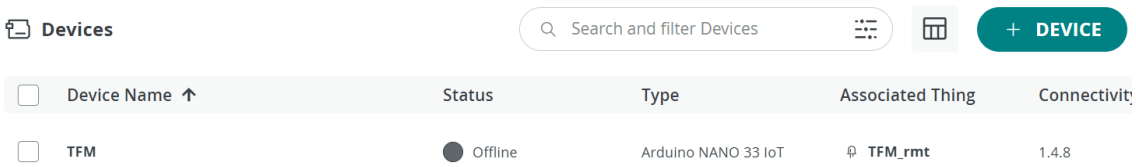


Figura 3.30. Inclusión de otro device

En la pantalla que aparece a continuación, se debe seleccionar “Third party device” (Figura 3.31), ya que se quiere incluir el ESP32, que es un dispositivo que no pertenece a la familia oficial de Arduino.

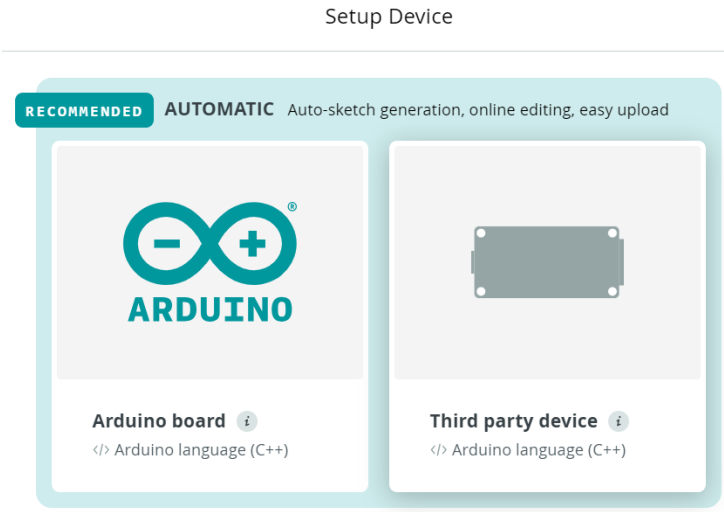


Figura 3.31. Selección de tipo de device

Seguidamente, se selecciona el modelo tal y como se ve en la Figura 3.32.

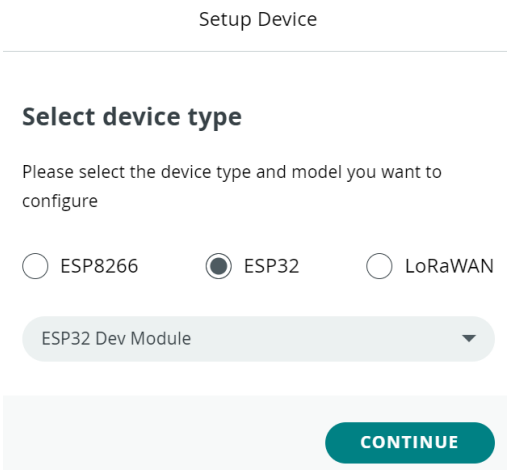


Figura 3.32. Selección de modelo de device

Por último, se le da el nombre de “TFM_ESP32”. Tras ello, se nos proporcionará un “Device ID” y una “Secret key”. Se debe guardar esta última, ya que se utilizará para conectarse a la red Wi-Fi que se seleccione en el Thing.

Después de haber creado el nuevo dispositivo, se creará un nuevo Thing llamado “TFM_rmt_2”. Para ello, se asociará el dispositivo creado anteriormente (Figura 3.33).

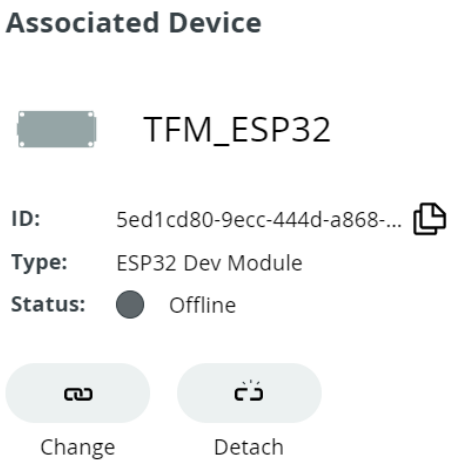


Figura 3.33. ESP32-WROOM-32 como dispositivo asociado al Thing "TFM_rmt_2"

Después, se configuran las credenciales de la red, que es la misma que el Thing anterior pero añadiendo la Secret key del dispositivo por no ser una placa de Arduino oficial (Figura 3.34).

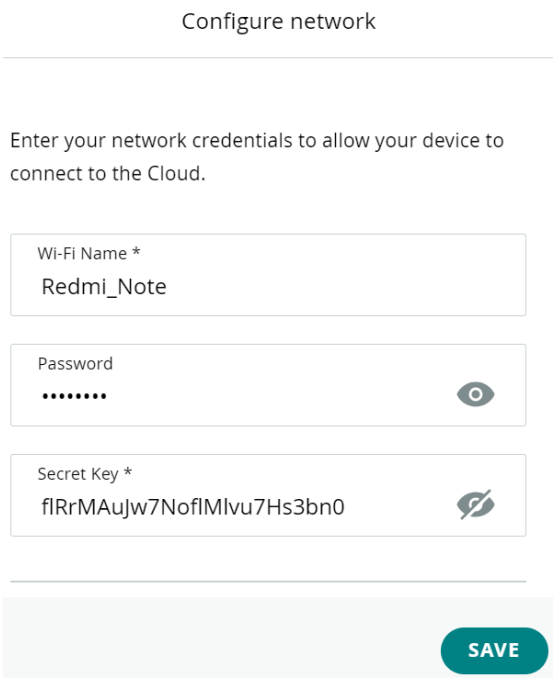


Figura 3.34. Credenciales de la red del Thing "TFM_rmt_2"

Por último, se crearán las variables de temperatura y humedad en el "Setup" del Thing (Figura 3.35), las cuales servirán para almacenar los valores medidos por el DHT11 en la nube. Estas variables se configuran para que sean de solo lectura y que se actualicen solo cuando cambie el valor medido, en lugar de actualizarse continuamente.

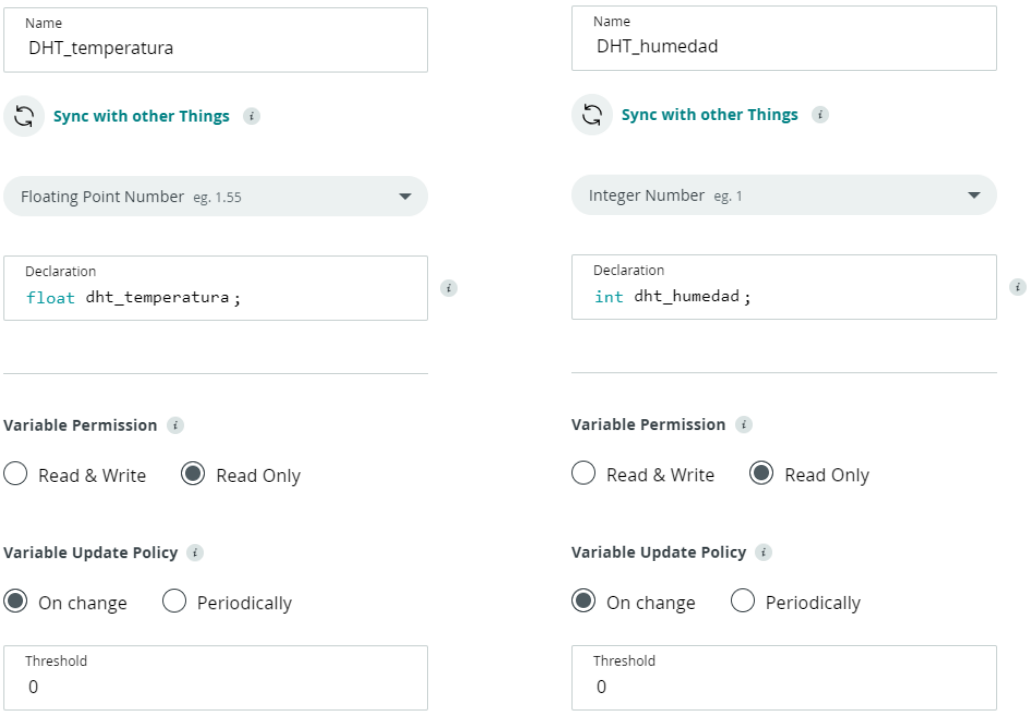


Figura 3.35. Variables del Thing "TFM_rmt_2"

Por lo tanto, el "Setup" del Thing "TFM_rmt_2" quedaría como se muestra en la Figura 3.36.

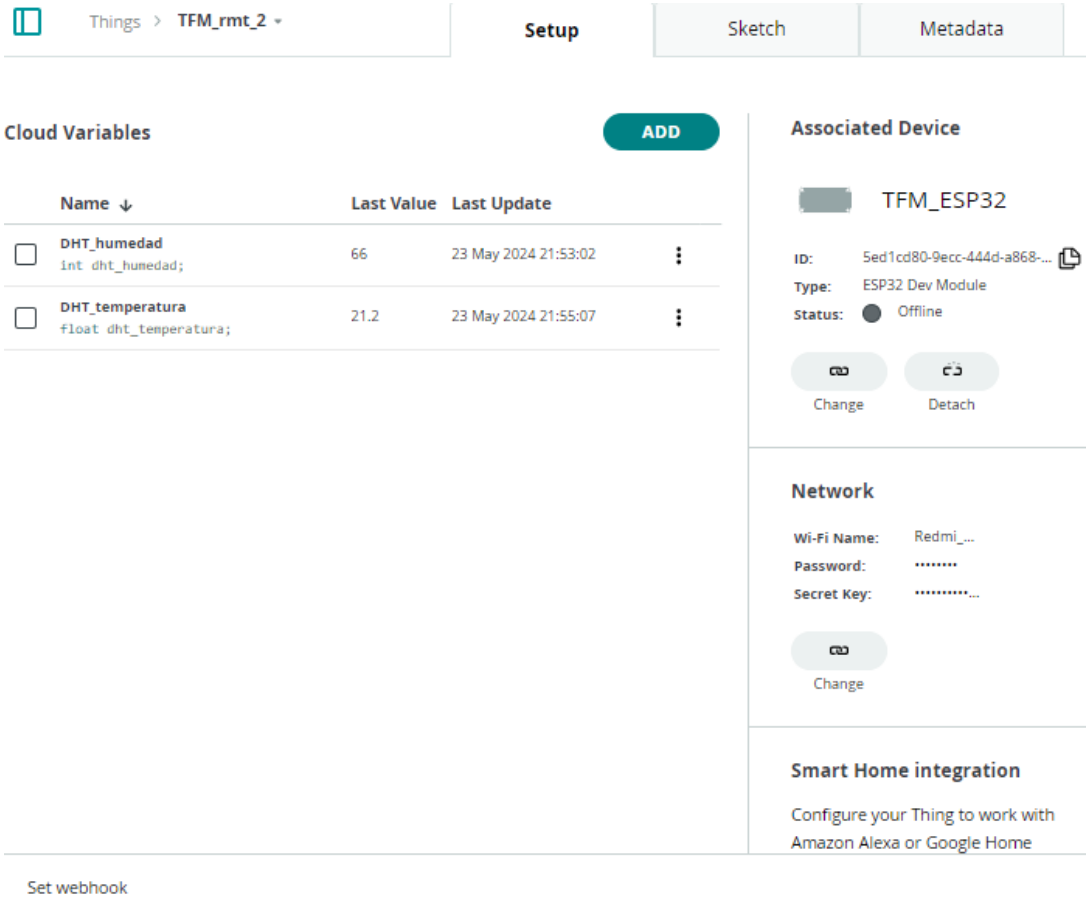


Figura 3.36. Setup del Thing "TFM_rmt_2"

3.3.3. Código fuente para el sensor DHT11

Lo siguiente es crear el código fuente que se empleará para medir la humedad y la temperatura del sensor DHT11 y almacenar los valores en las variables globales creadas en Arduino Cloud para poder visualizarlas en el Dashboard (Figuras 3.37, 3.38 y 3.39).

Antes del setup (Figura 3.37), se incluyen las variables globales (líneas 9-10) y las librerías especiales del sensor (líneas 20-21). También, se define que el tipo de sensor que se utiliza es el DHT11 (línea 24), ya que también existe el DHT22. Seguidamente, se define el sensor (línea 26).

```
1  /*
2    Sketch generated by the Arduino IoT Cloud Thing "Untitled"
3    https://create.arduino.cc/cloud/things/7315d4bd-10d6-4628-8406-f85c37c42891
4
5    Arduino IoT Cloud Variables description
6
7    The following variables are automatically generated and updated when changes are made to the Thing
8
9    float dht_temperatura;
10   int dht_humedad;
11
12   Variables which are marked as READ/WRITE in the Cloud Thing will also have functions
13   which are called when their values are changed from the Dashboard.
14   These functions are generated with the Thing and added at the end of this sketch.
15 */
16
17 #include "thingProperties.h"
18
19 // DHT sensor library - Version: Latest
20 #include <DHT.h>
21 #include <DHT_U.h>
22
23 #define DHTPIN 4
24 #define DHTTYPE DHT11
25
26 DHT dht(DHTPIN,DHTTYPE);
```

Figura 3.37. Código fuente para calcular los valores de humedad y temperatura – Parte 1

En el setup (Figura 3.38), únicamente se inicializa el monitor serie, el sensor y las propiedades de inicio de Arduino Cloud.

```
29 void setup() {
30   Serial.begin(9600);
31   delay(1500);
32
33   // Defined in thingProperties.h
34   initProperties();
35
36   dht.begin();
37
38   // Connect to Arduino IoT Cloud
39   ArduinoCloud.begin(ArduinoIoTPreferredConnection);
40   setDebugMessageLevel(2);
41   ArduinoCloud.printDebugInfo();
42 }
```

Figura 3.38. Código fuente para calcular los valores de humedad y temperatura – Parte 2

En el loop (Figura 3.39), se obtienen los valores de la humedad y la temperatura del sensor utilizando instrucciones que pertenecen a las librerías del DHT11 declaradas al principio del programa.

```
44  void loop() {
45    ArduinoCloud.update();
46
47    delay(2000); //For the DHT sensor
48
49    dht_humedad = dht.readHumidity();
50    Serial.println(dht_humedad);
51    dht_temperatura = dht.readTemperature(); //Celsius
52    Serial.println(dht_temperatura);
53  }
```

Figura 3.39. Código fuente para calcular los valores de humedad y temperatura – Parte 3

En la Figura 3.40 se puede ver cómo sería el simple diagrama de flujo para este sensor.

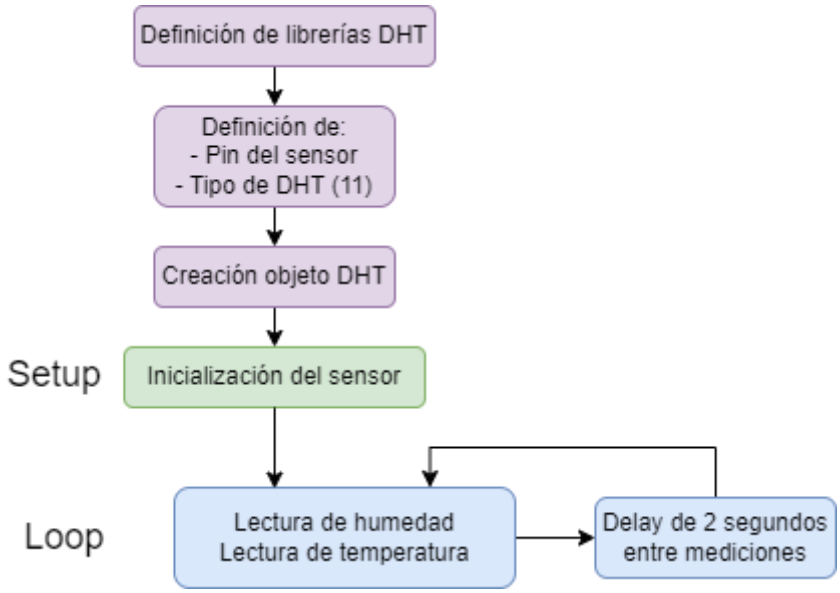


Figura 3.40. Diagrama de flujo del código fuente para calcular los valores de humedad y temperatura

3.3.4. Dashboard

Por último, se crean widgets para las variables de temperatura y humedad. La configuración de cada widget será la siguiente:

- El de la temperatura será de tipo “Gauge” (Figura 3.42) para poder ver el valor que se está midiendo a tiempo real. Lo primero que hay que hacer es enlazar la variable que se quiere mostrar, siendo en este caso “DHT_temperatura” del Thing “TFM_rmt_2” (Figura 3.41). Luego, se configura que el rango de valores sea de 0 a 50.

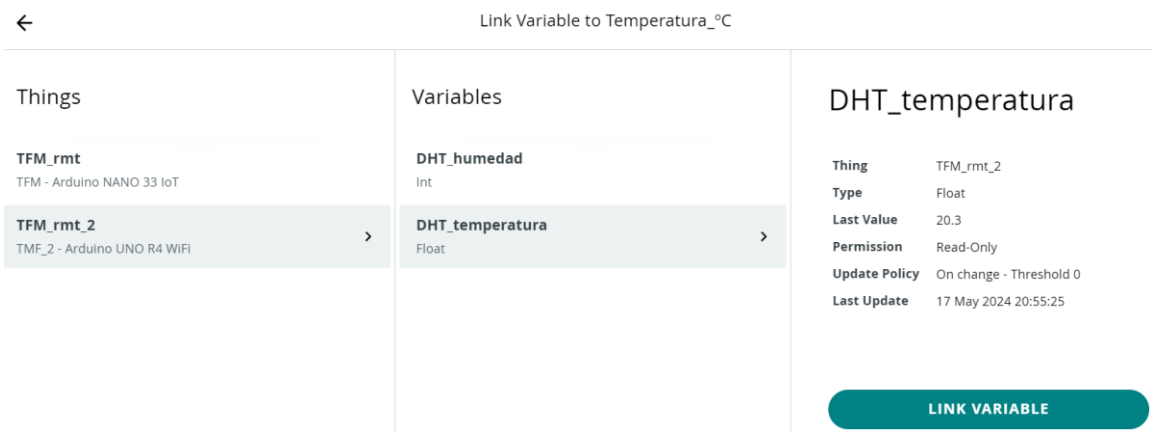


Figura 3.41. Variable enlazada del widget de temperatura

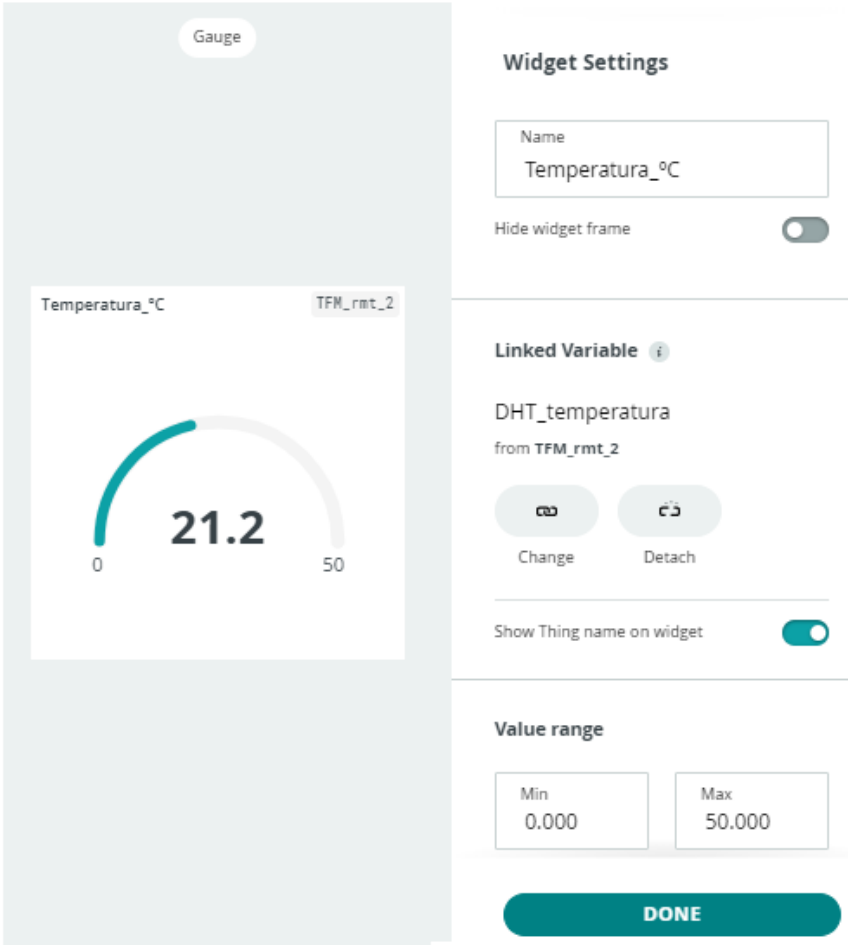


Figura 3.42. Rango de valores del widget de temperatura

- El widget de la humedad será de tipo “Percentage” (Figura 3.44) ya que el valor recibido es el porcentaje de humedad en el ambiente. En este se enlaza la variable “DHT_humedad” del Thing “TFM_rmt_2” (Figura 3.43). Seguidamente, se configura que a partir del valor 70, se cambie de color azul a naranja para indicar de forma visual una advertencia de que el valor de la humedad es elevado (Figura 3.44).

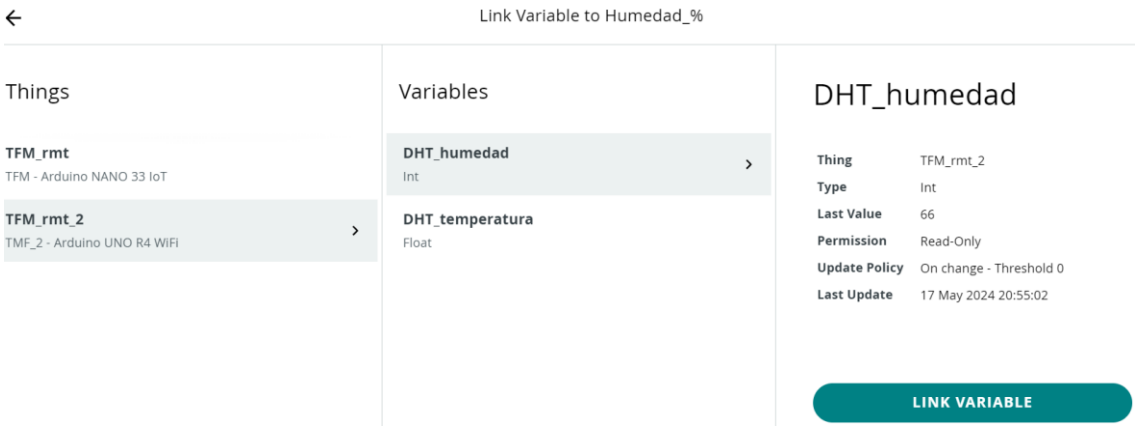


Figura 3.43. Variable enlazada del widget de humedad

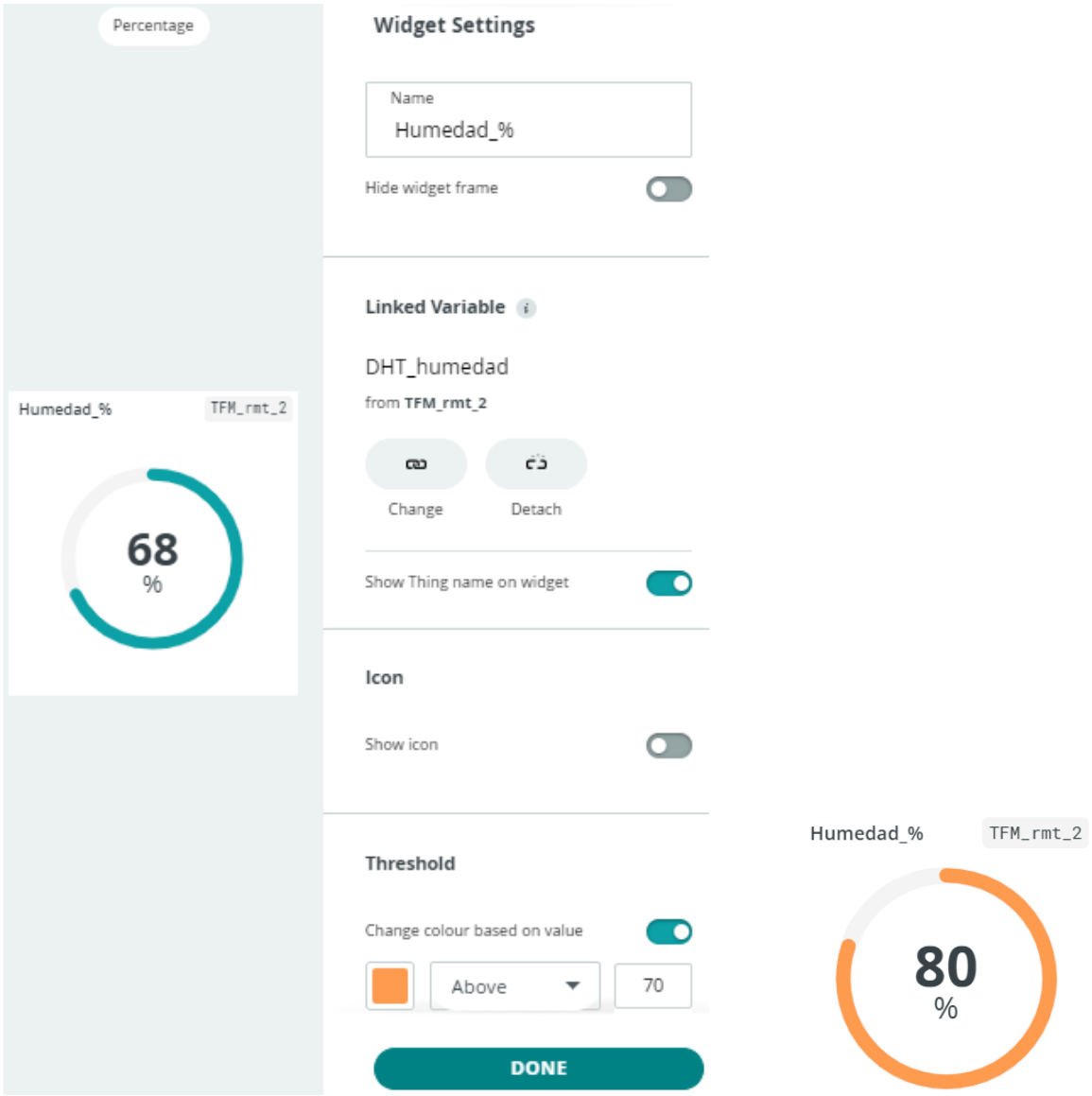


Figura 3.44. Configuración del widget de humedad (izquierda) y comprobación (derecha)

Por lo tanto, en la Figura 3.45 se puede ver cómo queda el dashboard con estos widgets.

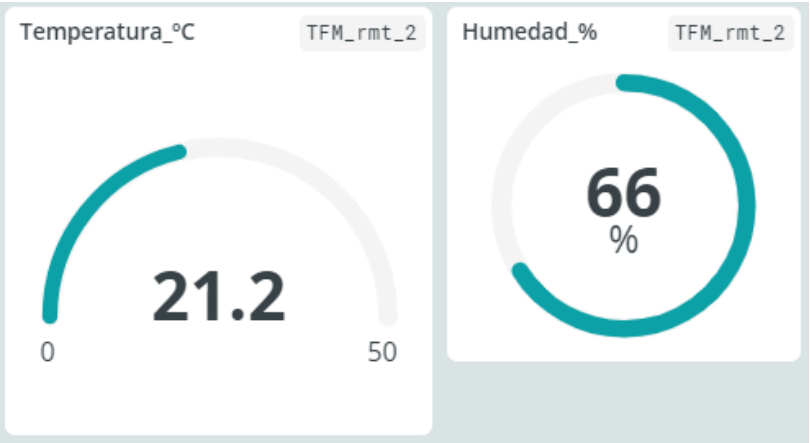


Figura 3.45. Dashboard con valores del sensor DHT11

Capítulo 4. Integración y validación del sistema

En este capítulo se mostrará el diseño completo del sistema, la integración de los diferentes módulos y las formas de notificación que se ofrecen de cara al usuario.

4.1. Diseño completo del sistema

En la Figura 4.1 se puede ver un diagrama simplificado del funcionamiento del sistema diseñado.

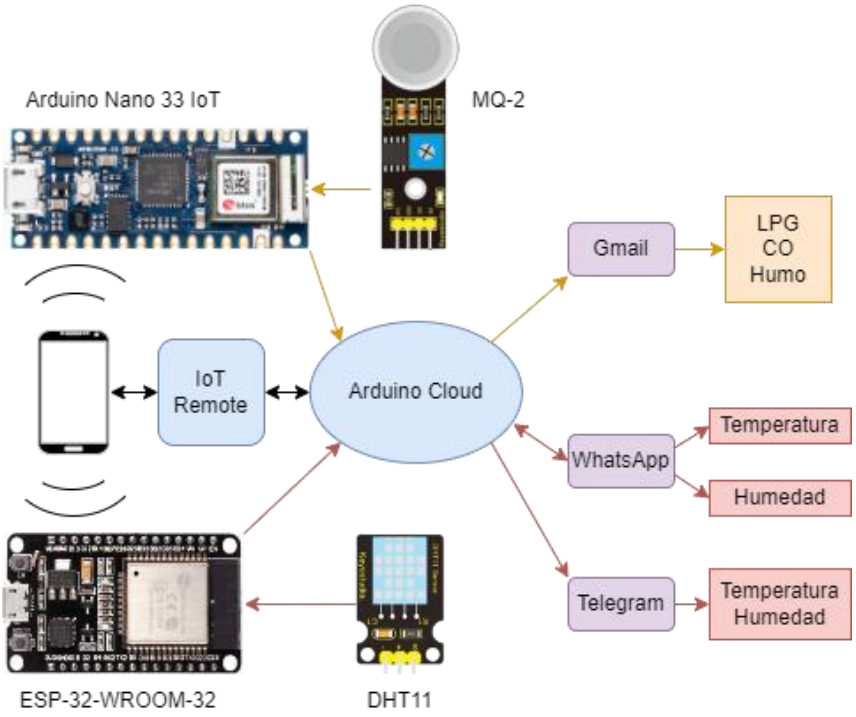


Figura 4.1. Diagrama del funcionamiento del sistema

Según el diagrama, se puede ver que, en este caso, el móvil es el punto de acceso para ambas placas, además de tener la aplicación “IoT Remote”. Como se explicará en el apartado 4.3, esta aplicación está directamente conectada a “Arduino Cloud”.

Además, se puede ver que existen 2 partes diferenciadas en el sistema, que coinciden con los Things creados en Arduino Cloud:

- Por una parte, el Arduino Nano 33 IoT está conectado al sensor de gas, del que recibirá los valores de los gases LPG, CO y humo del ambiente. Seguidamente, estos datos se almacenarán en las variables globales definidas en Arduino Cloud, concretamente del Thing “TFM_rmt”. Tras ello, si los valores de estas variables sobrepasan los umbrales definidos en el código, se enviará un email utilizando la aplicación de Gmail. Por lo tanto, el usuario recibirá una alerta de gases elevados, recibiendo el valor de los tres gases medidos.
- Por otra parte, el ESP-32-WROOM-32 se encuentra conectado al sensor DHT11, que enviará los valores de la temperatura y humedad. Estos datos se guardarán en

las variables globales definidas en Arduino Cloud, concretamente del Thing “TFM_rmt_2”. Teniendo en cuenta esto, existen dos formas de comunicar estos valores al usuario:

- Mediante la aplicación “WhatsApp”, el usuario puede pedir cualquiera de los dos valores que mide el sensor. Esto es escribiendo “temperatura” o “humedad” para recibir el valor que elija.
- A través de “Telegram” recibirá alertas si los datos medidos sobrepasan algún umbral, ya sean inferiores o superiores, obteniendo los dos valores medidos junto a un mensaje que indique el umbral que se ha sobrepasado.

En el apartado 4.5 se encuentran los diagramas de flujo completos de los Things “TFM_rmt” y “TFM_rmt_2”.

4.2. Arduino Cloud

Integrando las variables creadas de los Things del capítulo 3 en un solo Dashboard, toda la información se puede mostrar fácilmente de cara al usuario utilizando “Arduino Cloud” en un ordenador (Figura 4.2).

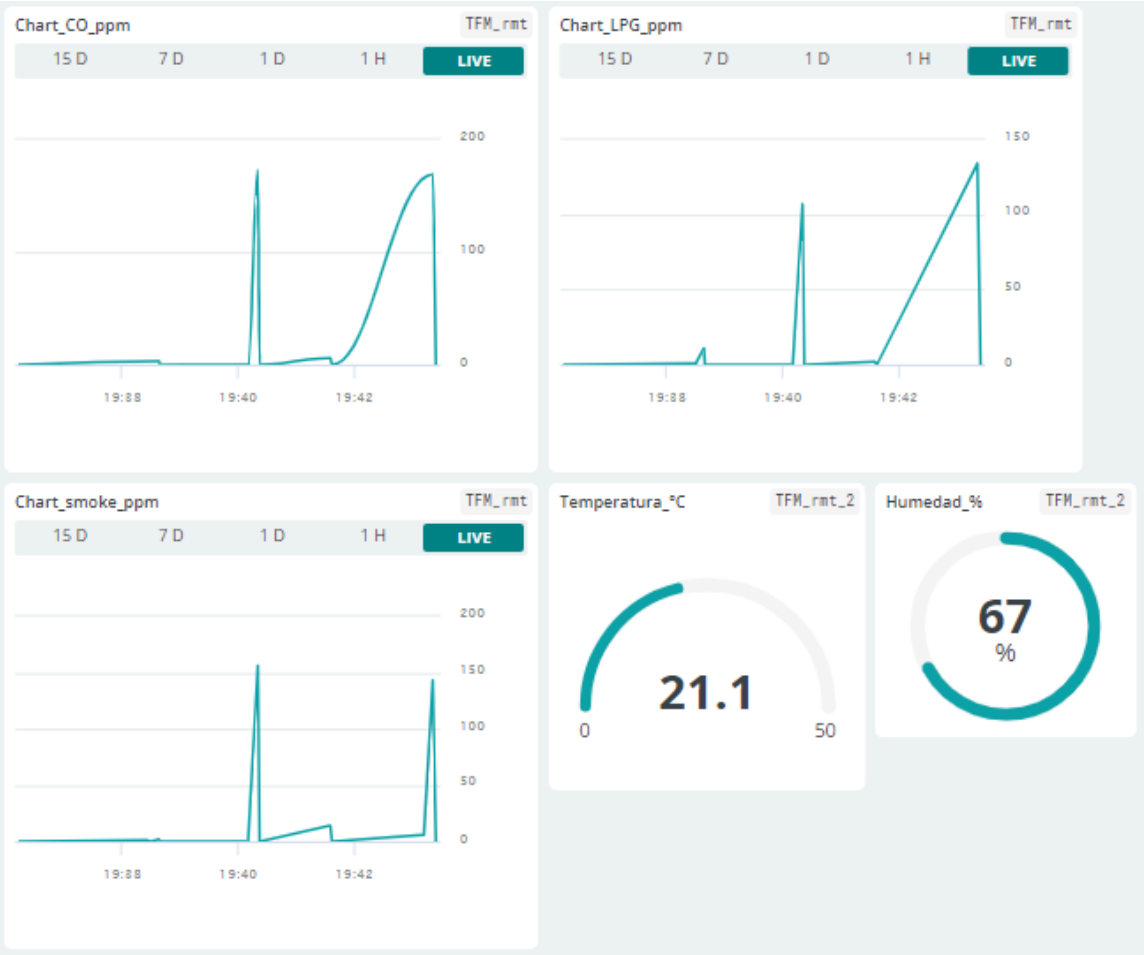


Figura 4.2. Dashboard final en ordenador (Arduino Cloud)

Desde este Dashboard se puede observar la evolución de los valores de los gases eligiendo el periodo desde el que se quiera observar, así como los valores a tiempo real de la humedad y la temperatura. Además, en la esquina superior derecha de cada widget viene indicado a qué Thing pertenece cada variable representada.

4.3. IoT Remote

“IoT Remote” es una aplicación para smartphones que se puede descargar a través de Google Play y la App Store. Esta aplicación complementa perfectamente a Arduino Cloud.

Permite desarrollar soluciones IoT en línea desde un ordenador, crear paneles optimizados para teléfonos móviles y luego monitorizar y controlar estos paneles de forma remota utilizando la aplicación móvil.

Esta herramienta permite monitorizar y controlar en tiempo real el estado de dispositivos y sensores mediante widgets intuitivos. Ofrece acceso a datos históricos para obtener información valiosa, además de permitir encender y apagar dispositivos de forma remota, ajustar niveles y modificar valores, proporcionando un control total sobre la solución IoT.

La función de Dispositivo Telefónico sincroniza los datos de los sensores del teléfono con la nube, transformando el móvil en un dispositivo IoT que se integra con proyectos en Arduino Cloud. [25]

En la Figura 4.3 se puede comprobar que en “IoT Remote” se ve el mismo dashboard que en Arduino Cloud, aunque en esta aplicación de móvil no se ha conseguido ver los datos de las gráficas a tiempo real (LIVE), ya que tiene problemas para cargar en la aplicación.

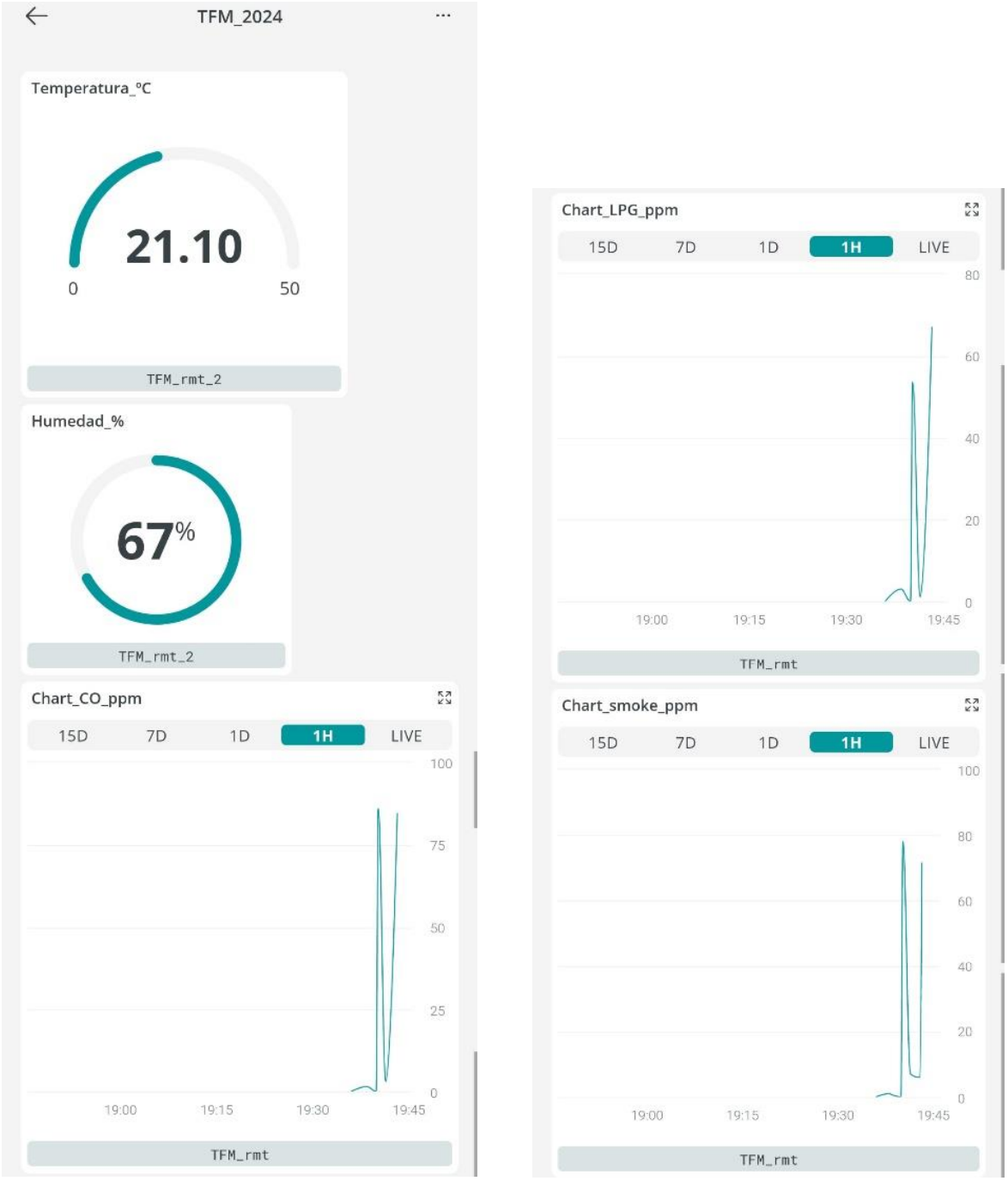


Figura 4.3. Dashboard final en smartphone (IoT Remote)

4.4. Notificaciones

En este apartado se describirán las aplicaciones de mensajería utilizadas para enviar notificaciones al usuario y, además, pueda realizar consultas de algunos datos.

4.4.1. Gmail

En este subapartado se describirá brevemente qué es Gmail y cómo se ha empleado para enviar notificaciones del valor de los gases medidos al usuario.

4.4.1.1. Introducción

Gmail (Figura 4.4) es el servicio de correo electrónico gratuito de Google para usuarios con una cuenta de dominio *gmail.com*. En el ámbito del correo electrónico, Gmail compite con servicios como Outlook (antiguo Hotmail) de Microsoft, uno de los pioneros en internet.

Esta aplicación permite enviar y recibir mensajes casi en tiempo real, aunque no es un servicio de mensajería instantánea, como Telegram o WhatsApp. Además, Gmail permite organizar los mensajes en carpetas independientes mediante filtros configurables. Esto significa que, si un correo cumple ciertas condiciones definidas por el usuario, se moverá automáticamente a la carpeta correspondiente o realizará otra acción específica. [26]

Figura 4.4. Icono Gmail [27]

Gmail permite enviar y recibir correos electrónicos a través de internet. Su gestor de correo almacena y organiza los mensajes de forma que sean fácilmente localizables, por ejemplo, según su remitente. Una de las características más útiles de Gmail es la opción de configurar respuestas automáticas por un periodo determinado. Además de enviar correos, Gmail permite adjuntar documentos, facilitando la transferencia de archivos directamente en los mensajes.

Asimismo, si se gestionan cuentas de correo electrónico con diferentes dominios, lo ideal es redirigir todos a una sola o administrarlos desde un único cliente. Por esta razón, Gmail es compatible con SMTP y se puede configurar para acceder al correo desde un cliente distinto a la propia web o desde la aplicación de Gmail. [26]

4.4.1.2. App password

El primer paso para poder configurar el envío de un correo electrónico mediante Gmail desde Arduino Cloud es obtener la “app password” de Google, que es lo que se utilizará para poder acceder a la cuenta de correo desde aplicaciones y servicios externos a Google.

Para ello, desde la cuenta de correo en la que se quieran enviar los mensajes, hay que pulsar en “Manage your account” y buscar “app passwords”. Tras pulsar en la opción, aparecerá una ventana en la que se pedirá dar nombre a la aplicación que se quiere crear (Figura 4.5, izquierda). Pulsando “Crear”, se indicará la contraseña de aplicación generada, que debe contener 16 letras (Figura 4.5, derecha).

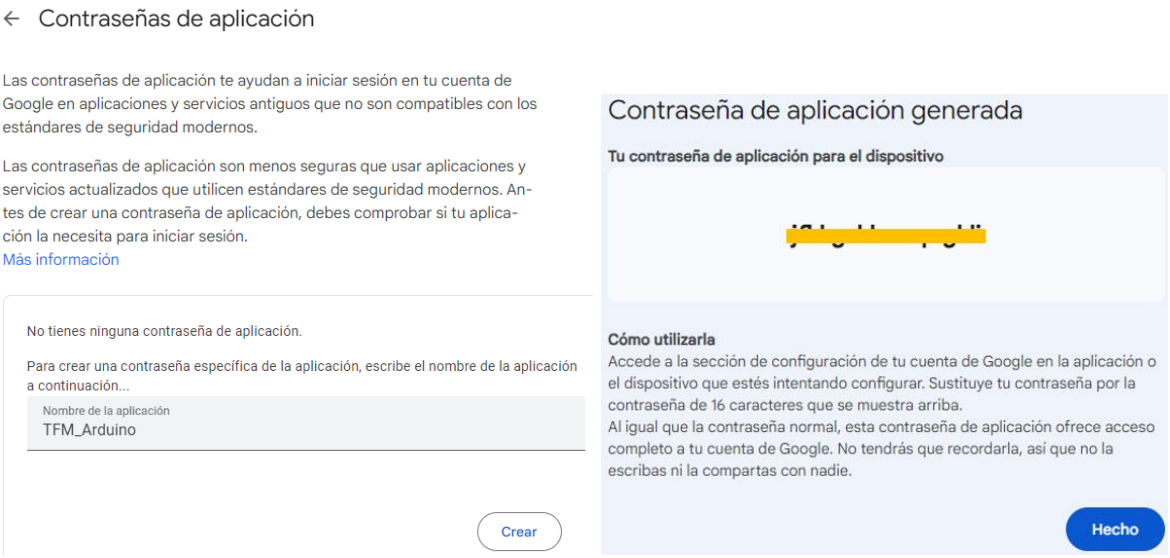


Figura 4.5. App password – Gmail

4.4.1.3. Código fuente

Al código utilizado en Arduino Cloud para el Thing “TFM_rmt” (Figuras 3.19, 3.20, 3.21 y 3.22) se le añadirán las siguientes líneas de código explicadas a continuación.

Antes del setup, se incluirán las líneas de la Figura 4.6, donde se incluye la librería especial para enviar emails y se define el email desde el que se manda (línea 36), la app password para poder acceder a este (línea 37) y el email receptor (línea 38). Seguidamente, se definen las variables del cuerpo del email (línea 40), que se rellenará en la función “sendEmail”, y el asunto del correo (línea 41).

```
33  //GMAIL
34  #include <WiFiNINA.h>
35  #include <EMailSender.h>
36  char eMailUser[] = "rmtpruebas707@gmail.com";
37  char eMailPass[] = "[redacted]";  // App password
38  char eMailRecipient[] = "rmtpruebas707@gmail.com";  // Receptor
39
40  String TEXT; // Cuerpo del email
41  const char SUBJ[] = "Peligro - Altos niveles de gases detectados"; // Asunto del email
```

Figura 4.6. Código fuente para enviar un email mediante Gmail – Parte 1 [28]

Seguidamente, se define la función “sendEmail” (Figura 4.7). Esta función:

- Rellena la variable creada para el cuerpo del mensaje con los valores de los gases medidos por el sensor (líneas 166-169).
- Crea un objeto “EmailSender” (línea 172) utilizando el nombre de usuario del correo electrónico (eMailUser), la contraseña (eMailPass), el remitente (eMailUser) y el nombre del remitente (senderName).
- Crea un objeto de mensaje de correo electrónico (línea 173).
- Crea un objeto para almacenar la respuesta del servidor de correo (línea 174).
- Asigna el asunto del mensaje a “SUBJ” (línea 176).
- Asigna el cuerpo del mensaje a “TEXT” (línea 177).

- Envía el correo electrónico al destinatario (eMailRecipient) con el mensaje (msg) y almacena la respuesta en “resp” (línea 178).

```
162  void sendEmail(){
163      // Nombre del emisor del email
164      char senderName[] = "Arduino Nano 33 IoT";
165
166      TEXT = "Se han detectado los siguientes niveles de gases:<br>"
167        "LPG = " + String(gas_LPG_ppm) + "ppm <br>" +
168        "CO = " + String(gas_CO_ppm) + "ppm <br>" +
169        "Humo = " + String(gas_smoke_ppm) + "ppm <br>";
170
171      // Se envia el email
172      EMailSender emailSend(eMailUser, eMailPass, eMailUser, senderName);
173      EMailSender::EMailMessage msg;
174      EMailSender::Response resp;
175      Serial.println("Sending email...");
176      msg.subject = SUBJ;
177      msg.message = TEXT;
178      resp = emailSend.send(eMailRecipient, msg);
179
180      Serial.println("Sending status: ");
181      Serial.print(resp.status);
182      Serial.println(resp.code);
183      Serial.println(resp.desc);
184      Serial.println("");
185      Serial.print("FROM: ");
186      Serial.println(eMailUser);
187      Serial.print("TO: ");
188      Serial.println(eMailRecipient);
189      Serial.print("SUBJECT: ");
190      Serial.println(SUBJ);
191      Serial.print("DATA:");
192      Serial.println(TEXT);
193      Serial.println("");
194  }
```

Figura 4.7. Código fuente para enviar un email mediante Gmail – Parte 2 [28]

Por último, al final del loop se añade la llamada a la función “sendEmail” (Figura 4.8), la cual se efectuará si el valor de alguno de los gases sobrepasa el umbral estipulado.

```
106  if (gas_LPG_ppm > 1000.0 || gas_CO_ppm > 9.0 || gas_smoke_ppm > 10.0){
107    sendEmail();
108  }
```

Figura 4.8. Código fuente para enviar un email mediante Gmail – Parte 3 [28]

Estos umbrales se han definido de la siguiente forma:

- El gas licuado del petróleo es peligroso cuando se sobrepasa el umbral de 1000 ppm. [29]
- La Agencia de Protección Ambiental de los Estados Unidos (EPA, por sus siglas en inglés) ha establecido una norma federal para la calidad del aire ambiental en relación con el CO, fijando un límite de 9 ppm para una exposición de 8 horas y de 25 ppm para una exposición a corto plazo de 1 hora [EPA 1991a]. Por lo tanto, se ha elegido la opción más restrictiva para definirla como umbral en el código, es decir, 9 ppm. [30]

- No se ha encontrado un valor exacto para definir un umbral de peligro en cuanto al humo, por lo que se ha definido de 10 ppm como aproximación al valor de CO comentado anteriormente.

En la Figura 4.9 se puede observar un diagrama de flujo del funcionamiento del código.

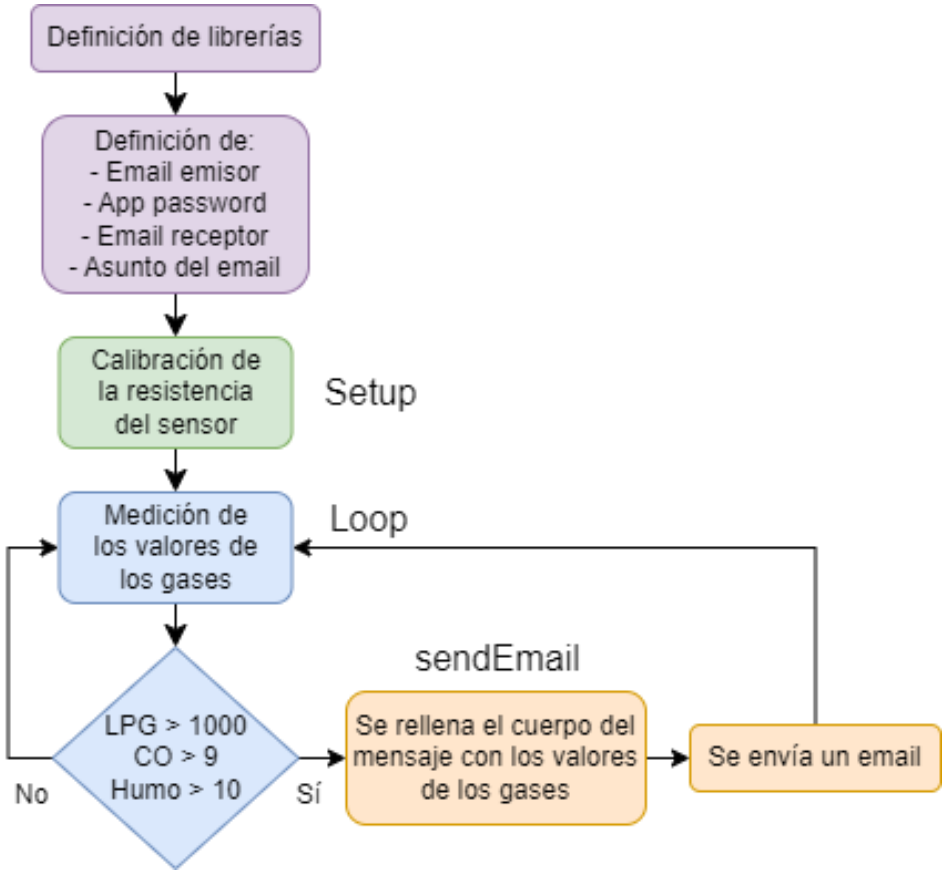


Figura 4.9. Diagrama de flujo para enviar un email mediante Gmail

4.4.1.4. Comprobación

Para comprobar el correcto funcionamiento de las alertas por Gmail, basta con echar algo de gas con un mechero directamente al sensor. En la Figura 4.10 se ve que se ha echado gas en distintos momentos, los cuales han sido notificados al usuario. De igual forma, en la Figura 4.11 se observan los valores de los gases enviados por el Arduino.

Figura 4.10. Alertas mediante Gmail

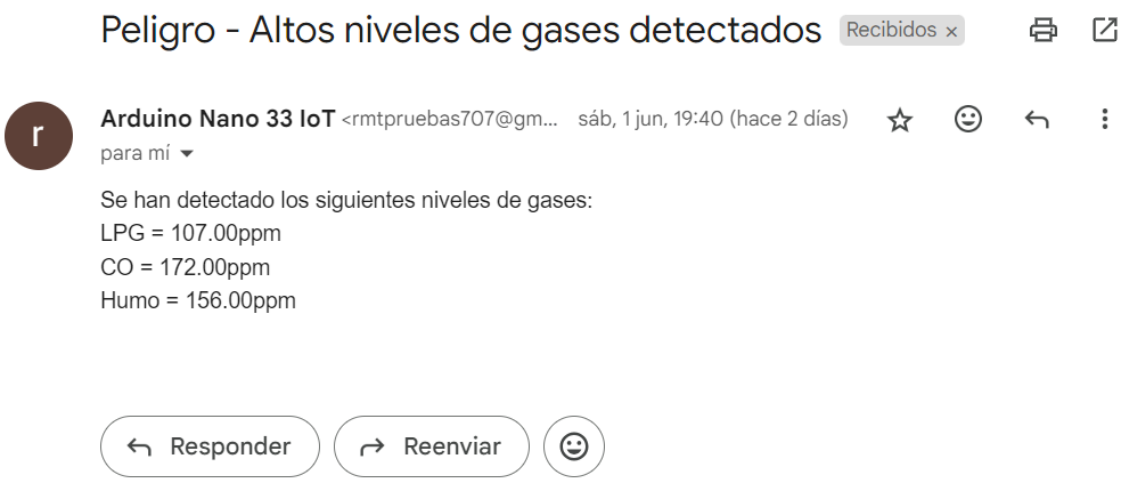


Figura 4.11. Valores de los gases medidos – Gmail

4.4.2. Telegram

4.4.2.1. Introducción

Telegram (Figura 4.12) es una aplicación de mensajería que prioriza la velocidad y la seguridad, siendo extremadamente rápida, gratuita y simple. Esta aplicación puede ser utilizada en varios dispositivos simultáneamente, con sincronización de mensajes entre teléfonos, tablets y computadoras.

Con Telegram, es posible enviar mensajes, fotos, videos y archivos de cualquier tipo (doc, zip, mp3, etc.), además de crear grupos de hasta 200.000 personas y canales para transmisiones a audiencias ilimitadas. Asimismo, es posible comunicarse con los contactos telefónicos y encontrar personas mediante sus nombres de usuario. En este sentido, Telegram combina las funcionalidades de los SMS y el correo electrónico, pudiendo ser utilizado para mensajería personal o profesional.

Además, esta aplicación ofrece llamadas de voz y videollamadas con cifrado de extremo a extremo, y chats de voz en grupos que pueden incluir miles de participantes, proporcionando una solución completa para la comunicación segura y eficiente. [31]

Figura 4.12. Icono Telegram [31]

En el caso de este proyecto, se creará un bot de Telegram para poder recibir notificaciones a través de él. Para ello, se utilizará la API de bots de Telegram.

4.4.2.2. Creación de un bot de Telegram

El primer paso es buscar el bot “BotFather” en Telegram (Figura 4.13), que es el bot principal utilizado para crear nuevas cuentas de bots y gestionar los del usuario existentes.

Figura 4.13. BotFather – Telegram

Al entrar en la conversación, aparecerá información sobre este bot además de un manual para utilizar la API para los bots. Para iniciar la conversación, se escribe “/start”, tras lo que mostrará los diferentes comandos que se le puede mandar para realizar distintas funcionalidades (Figura 4.14).

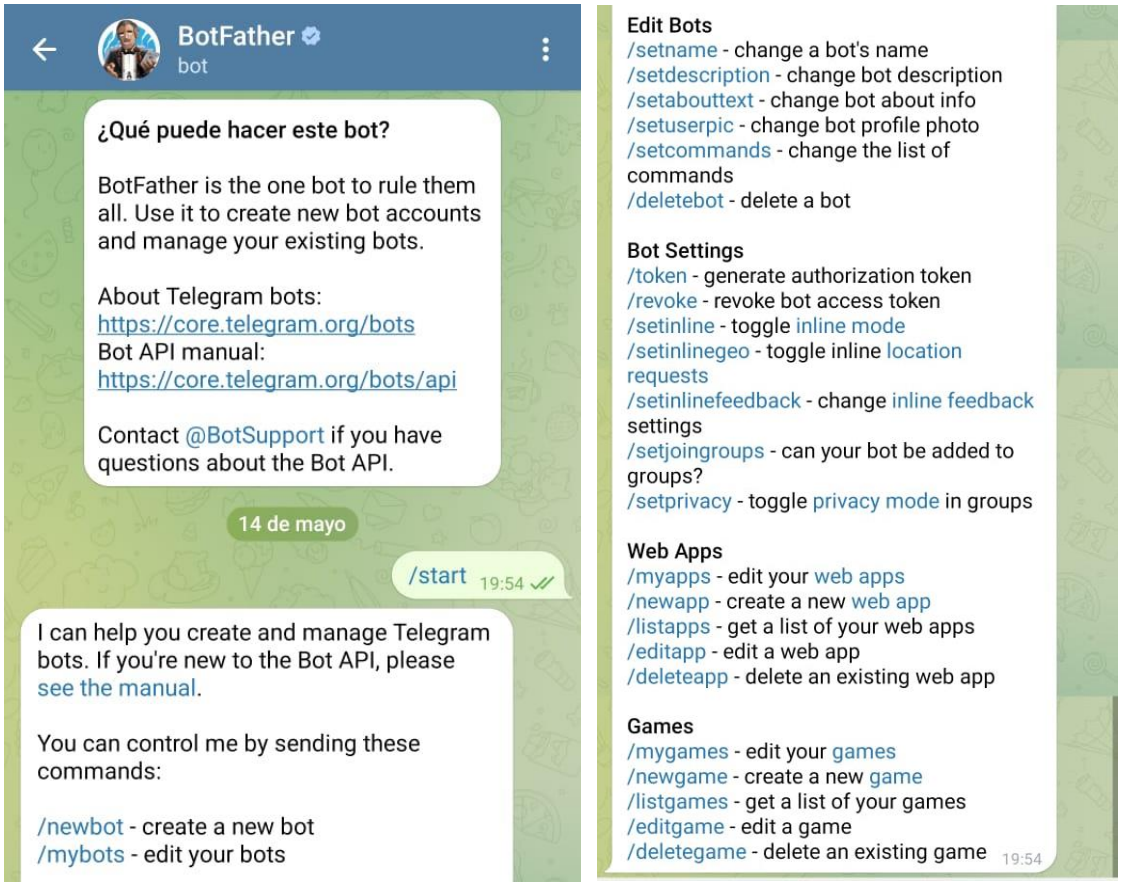


Figura 4.14. Comandos del BotFather – Telegram

Por lo tanto, para crear un nuevo bot se le manda el comando “/newbot”. Tras ello, hay que definir un nombre para este bot, que será el que se vea como nombre de contacto, y un nombre de usuario único. Este último nombre debe terminar en “Bot” o “_bot”, tal y como

indica el BotFather. Por último, recibimos el token que se utilizará para acceder al API HTTP.

Este procedimiento se puede ver en la Figura 4.15.

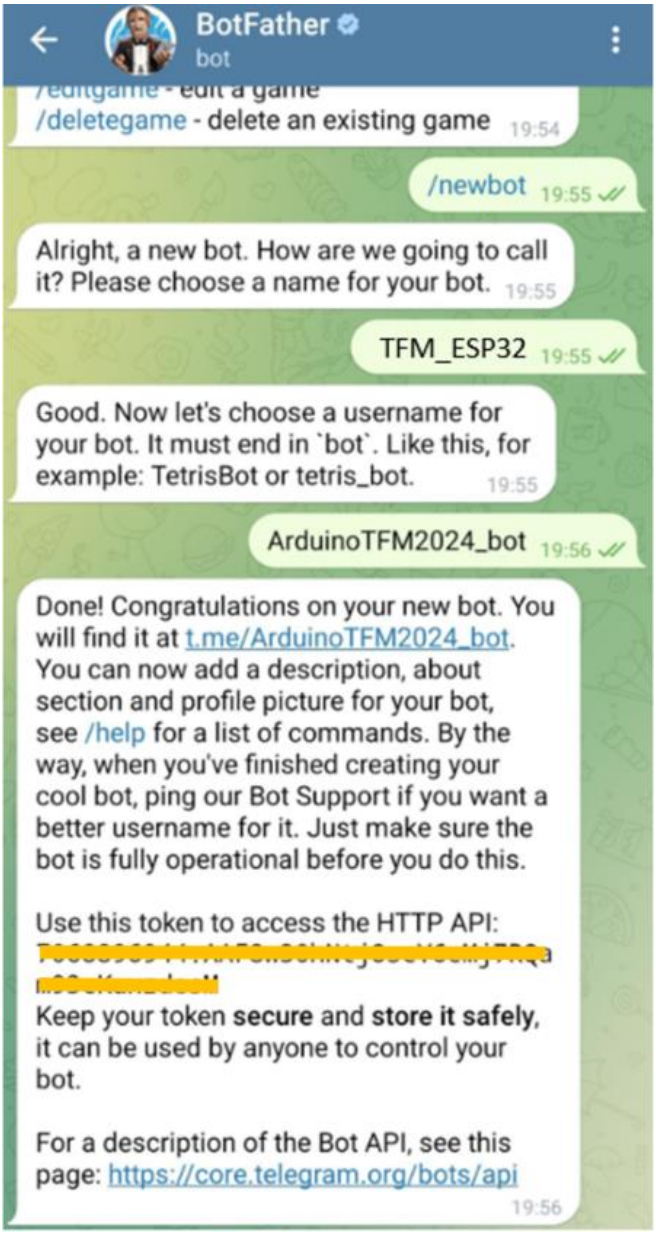


Figura 4.15. Creación del bot "TFM_ESP32" – Telegram

A continuación, se manda el mensaje "/start" en la conversación del nuevo bot (Figura 4.16) para iniciarlo.

Para obtener el ID del chat del bot, hay que escribir en un navegador web:

```
api.telegram.org/bot[token HTTP API]/getUpdates
```

Al cargar la página, aparecerá la respuesta al mensaje "/start" (Figura 4.16). Desde aquí se puede ver el ID del chat, el usuario que le ha escrito, la fecha y el ID del mensaje.

Figura 4.16. Obtención del ID del chat del bot

4.4.2.3. Código fuente

Al código utilizado en Arduino Cloud para el Thing “TFM_rmt_2” (Figuras 3.37, 3.38 y 3.39) se le añadirán las siguientes líneas de código explicadas a continuación.

Antes del setup, se incluirán las líneas de la Figura 4.17, donde se incluye la librería especial de Telegram (línea 18). Tras ello, se define el token del bot (línea 30), el ID del chat (línea 33) y se crea el objeto del bot (38). Seguidamente, se define el intervalo de tiempo mínimo que tiene que haber pasado para enviar el siguiente mensaje de Telegram, en este caso de 20 segundos, para evitar que se bloquee la conexión con el chat del bot (líneas 41-42). Por último, se definen umbrales para enviar alertas por Telegram cuando los valores de la temperatura o la humedad los superen (líneas 45-48).

```
17  #include "thingProperties.h"
18  #include <UniversalTelegramBot.h>
19
20  // DHT sensor library - Version: Latest
21  #include <DHT.h>
22  #include <DHT_U.h>
23
24  #define DHTPIN 4
25  #define DHTTYPE DHT11
26
27  DHT dht(DHTPIN,DHTTYPE);
28
29  // Inicializacion del Telegram BOT
30  #define BOTtoken "[redacted]"
31
32  // Bot Token (Obtenido del Botfather)
33  #define CHAT_ID "[redacted]"
34
35  int status = WL_IDLE_STATUS;
36
37  WiFiClientSecure client;
38  UniversalTelegramBot bot(BOTtoken, client);
39
40  // Variables para controlar el tiempo entre envíos
41  unsigned long lastTelegramTime = 0;
42  const unsigned long telegramInterval = 20000; // 20 segundos
43
44  // Umbrales para alertas
45  const float TEMP_ALTA_UMBRAL = 28.0;
46  const float TEMP_BAJA_UMBRAL = 5.0;
47  const int HUMEDAD_ALTA_UMBRAL = 70;
48  const int HUMEDAD_BAJA_UMBRAL = 50;
```

Figura 4.17. Código fuente para enviar un mensaje de Telegram – Parte 1

En el setup (Figura 4.18), únicamente se incluye la instrucción para activar el certificado de Telegram (línea 67).

```
64    // Connect to Arduino IoT Cloud
65    ArduinoCloud.begin(ArduinoIoTPreferredConnection);
66
67    client.setCACert(TELEGRAM_CERTIFICATE_ROOT);
68    setDebugMessageLevel(2);
69    ArduinoCloud.printDebugInfo();
```

Figura 4.18. Código fuente para enviar un mensaje de Telegram – Parte 2

A continuación, se crea la función “enviarAlerta” (Figura 4.19), que servirá para comprobar si han pasado 20 segundos desde el último mensaje enviado a través de Telegram (líneas 85-87). También, comprobará si los valores medidos de humedad y temperatura han superado los umbrales (líneas 90-101) definidos antes del setup y definir así un mensaje indicando el umbral que se ha superado. En este caso llamará a la función “sendTelegram” (línea 103) y guardará el tiempo en el que se ha enviado el mensaje de Telegram (línea 104).

```
84  void enviarAlerta(float temperature, int humidity) {
85    unsigned long currentMillis = millis();
86
87    if (currentMillis - lastTelegramTime >= telegramInterval) {
88      String alerta;
89
90      if (temperature >= TEMP_ALTA_UMBRAL) {
91        alerta = "Temperatura alta detectada\n";
92      } else if (temperature <= TEMP_BAJA_UMBRAL) {
93        alerta = "Temperatura baja detectada\n";
94      } else if (humidity >= HUMEDAD_ALTA_UMBRAL) {
95        alerta = "Humedad alta detectada\n";
96      } else if (humidity <= HUMEDAD_BAJA_UMBRAL) {
97        alerta = "Humedad baja detectada\n";
98      } else {
99        // Si no hay alerta, no se envia nada
100       return;
101     }
102
103     sendTelegram(alerta, temperature, humidity);
104     lastTelegramTime = currentMillis; // Actualiza el tiempo del último mensaje enviado
105   }
106 }
```

Figura 4.19. Código fuente para enviar un mensaje de Telegram – Parte 3

Seguidamente, se crea la función “sendTelegram” (Figura 4.20), la cual recibirá los datos de temperatura y humedad además del mensaje personalizado por la anterior función, los cuales se concatenarán para crear un solo mensaje que será enviado por el bot al chat con el usuario (línea 112, 114, 116).

```
109 void sendTelegram(String alerta, float temperatura, int humedad)  {
110   Serial.println("SENDING A TELEGRAM MESSAGE...");
111   alerta.concat("\nTemperatura: ");
112   alerta.concat(temperatura);
113   alerta.concat("ºC\nHumedad: ");
114   alerta.concat(humedad);
115   alerta.concat("%");
116   bot.sendMessage(CHAT_ID, alerta);
117
118   Serial.println("Mensaje enviado");
119 }
```

Figura 4.20. Código fuente para enviar un mensaje de Telegram – Parte 4

Por último, al final del loop (Figura 4.21) se añade la llamada a la función “enviarAlerta” (línea 80).

```
72  void loop() {
73    ArduinoCloud.update();
74    delay(2000); // Espera 2 segundos entre medidas
75    dht_humedad = dht.readHumidity();
76    dht_temperatura = dht.readTemperature(); // Celsius
77    Serial.println(dht_humedad);
78    Serial.println(dht_temperatura);
79
80    enviarAlerta(dht_temperatura, dht_humedad);
81  }
```

Figura 4.21. Código fuente para enviar un mensaje de Telegram – Parte 5

En la Figura 4.22 se puede observar un diagrama de flujo del funcionamiento del código.

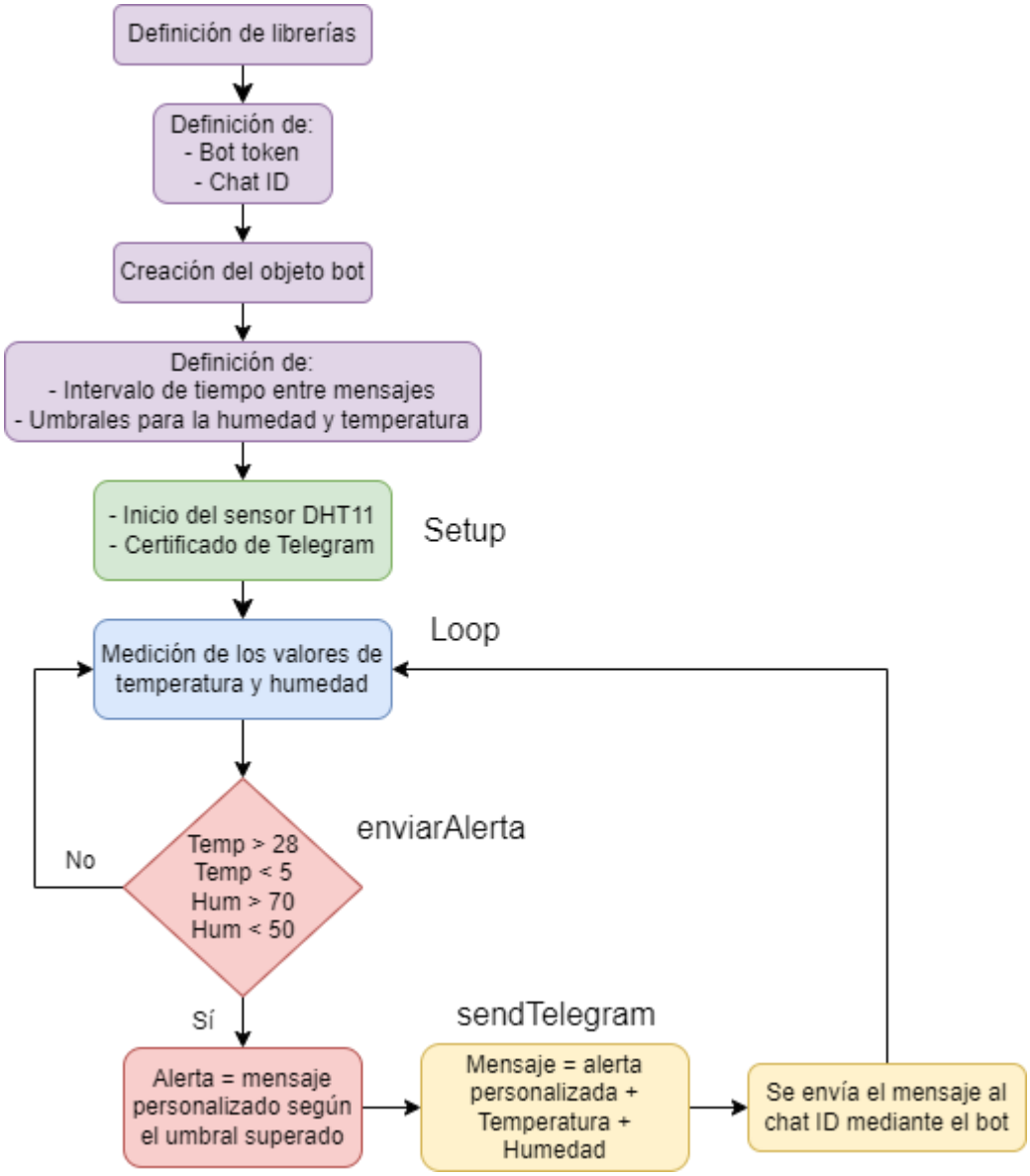


Figura 4.22. Diagrama de flujo para enviar un mensaje mediante Telegram

4.4.2.4. Comprobación

Para comprobar el correcto funcionamiento de las alertas mediante Telegram (Figura 4.23), se ha probado a incrementar la temperatura del ambiente poniendo algo con temperatura elevada al lado del sensor, a la vez que se incrementaba la humedad echando agua al aire con un pulverizador.

Figura 4.23. Alertas mediante Telegram

Viendo las alertas, se comprueba que la temperatura se ha notificado cuando ha superado los 28ºC, mientras que la humedad se ha seguido notificando mientras sobrepasa el 70%.

Además, se puede observar que únicamente se envía una alerta, aunque cada variable sobrepase uno de sus umbrales. Esto es porque la elección de qué alerta se notifica viene dada por el orden descrito en el código.

A lo largo de estas notificaciones se puede ver la evolución de estos valores cada 20 segundos mientras superen el umbral definido para las alertas.

4.4.3. WhatsApp

4.4.3.1. Introducción

WhatsApp (Figura 4.24) es una plataforma rápida, fácil de usar y confiable para conectarse con personas en cualquier parte del mundo que se originó como una alternativa a los SMS. Esta aplicación permite enviar y recibir una variedad de archivos multimedia, como texto, fotos, videos, documentos y ubicaciones, además de realizar llamadas.

Debido a que los usuarios también comparten datos muy personales, se incorporó el cifrado de extremo a extremo en la aplicación. WhatsApp, además de ser gratuita, es compatible con una amplia gama de dispositivos móviles y funciona en áreas con baja conectividad. [32]

Figura 4.24. Icono WhatsApp [32]

En los siguientes subapartados se explicará cómo se ha conseguido enviar información a través de una cuenta de WhatsApp externa utilizando las plataformas ThingESP, Twilio y Arduino Cloud.

4.4.3.2. ThingESP

La plataforma ThingESP (Figura 4.25) facilita la operación de una amplia gama de dispositivos IoT, como ESP32, ESP8266, Raspberry Pi, NodeMCU y otros, todo ello con la comodidad de la integración de Twilio WhatsApp. [33]

Figura 4.25. Icono ThingESP [33]

Por lo tanto, tras iniciar sesión en la aplicación, aparecerá la pantalla principal que se muestra en la Figura 4.26.

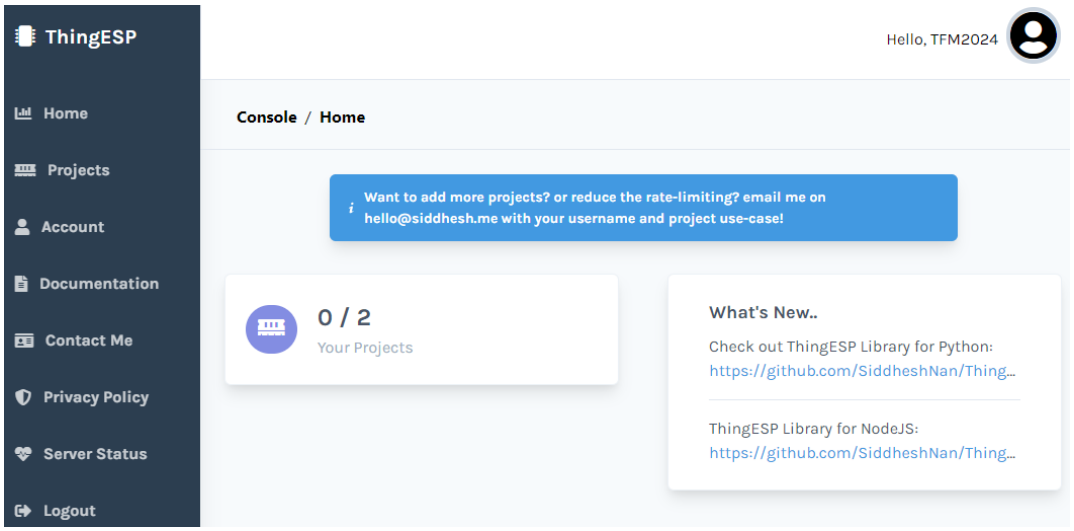


Figura 4.26. Menú principal – ThingESP

Desde el menú que aparece en la izquierda, se pincha en “Projects” y, seguidamente, en “Add New Project” (Figura 4.27).

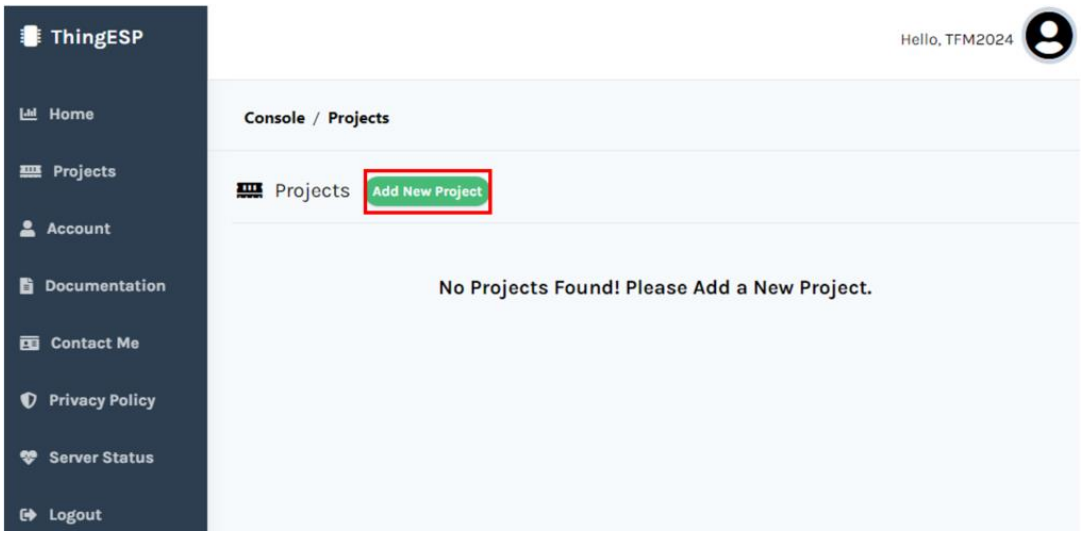


Figura 4.27. Menú Projects – ThingESP

A continuación, se define que el nombre del proyecto sea “HUMTEMP” y que las credenciales del dispositivo sean los caracteres “NOTIFICACION” (Figura 4.28, izquierda). Tras lo cual, habrá que guardar el link que se genera asociado al proyecto (Figura 4.28, derecha).

Figura 4.28. Configuración del proyecto – ThingESP

En la Figura 4.29 se puede ver cómo queda el menú principal del proyecto.

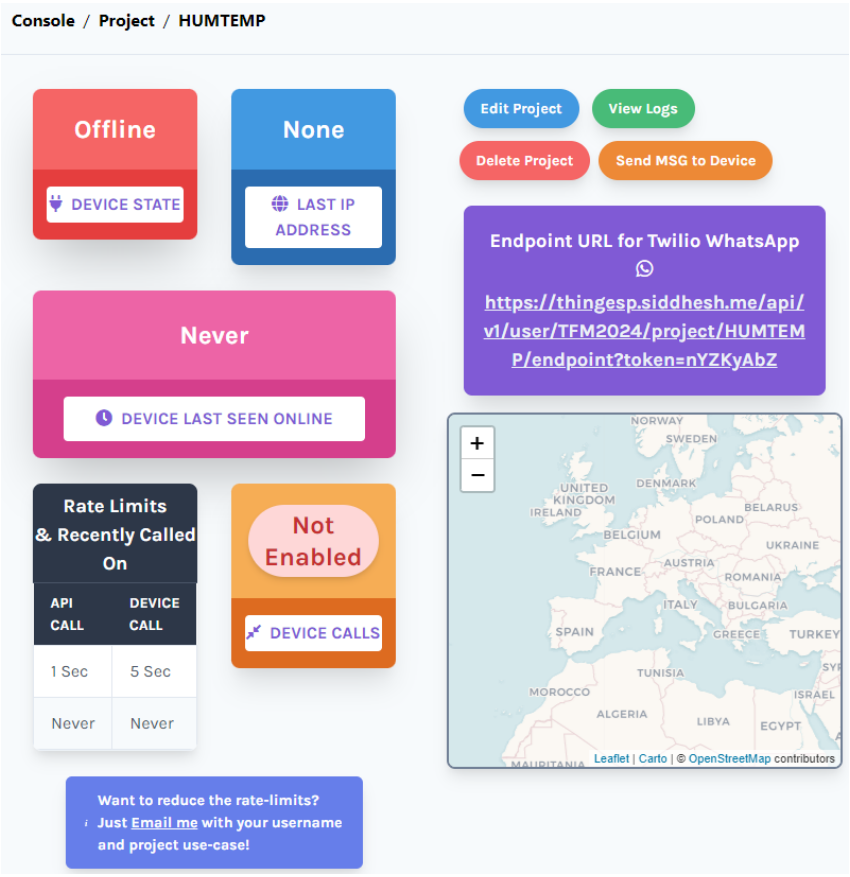


Figura 4.29. Menú principal del proyecto – ThingESP

Los siguientes pasos se realizarán en la aplicación de Twilio, tras lo que habrá que volver a la configuración del proyecto en ThingESP.

4.4.3.3. Twilio

Esta aplicación se utiliza para gestionar varios aspectos de las comunicaciones de un software, desde el call center en la nube y las llamadas dentro de la aplicación, hasta los SMS automatizados, la autenticación en dos pasos y los mensajes a los usuarios a través de WhatsApp. [34]

Figura 4.30. Icono Twilio [35]

El primer paso tras iniciar sesión es pinchar en “Get Started with Twilio” para conseguir un número de teléfono de Twilio. Además, en la parte de abajo aparece el ID de la cuenta y el Token de autentificación (Figura 4.31), que se utilizarán en el código para el ESP32.

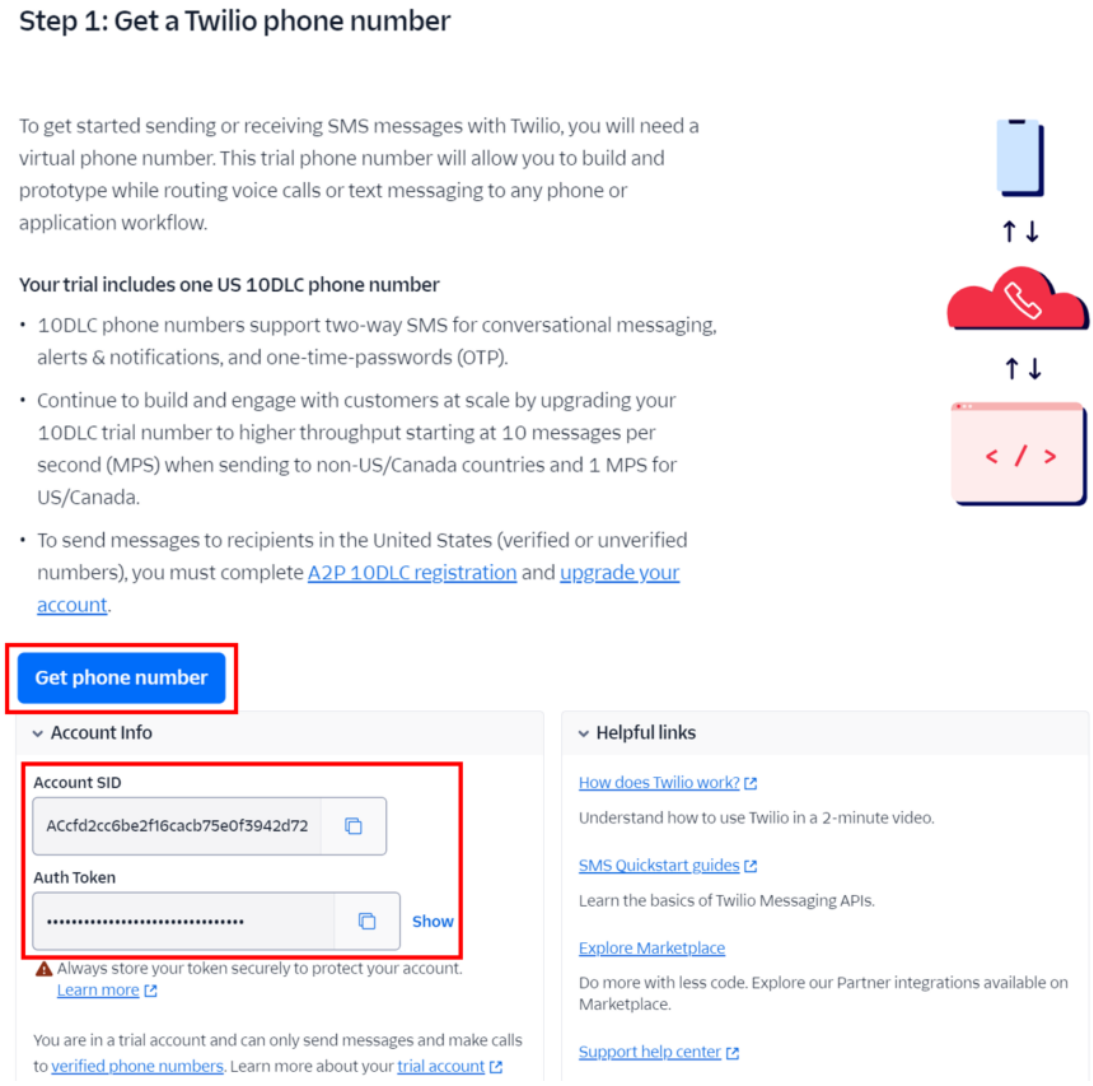


Figura 4.31. Información de la cuenta – Twilio

Por lo tanto, al pinchar en “Get phone number” aparece el número que se asociará a la cuenta (Figura 4.32).

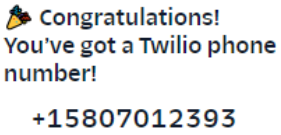


Figura 4.32. Número de teléfono asociado a la cuenta – Twilio

En la Figura 4.33 se puede ver información sobre el número de teléfono proporcionado.

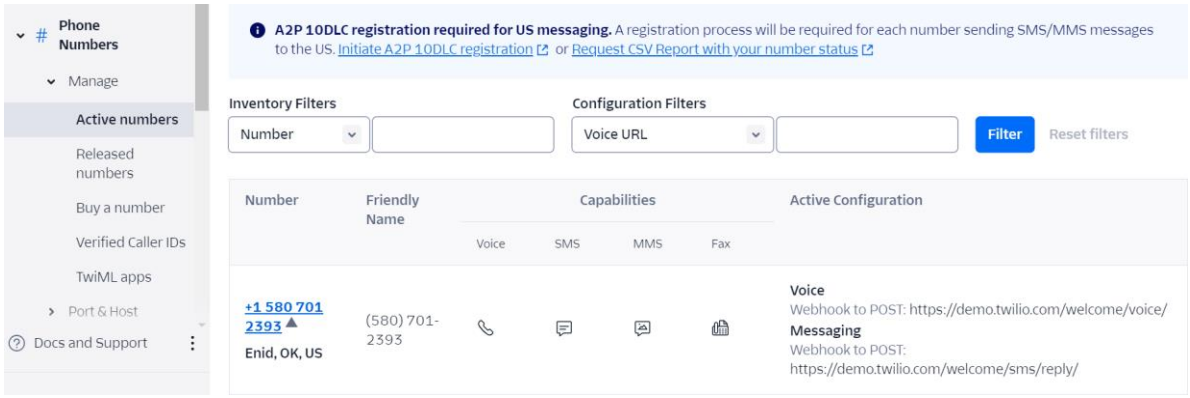


Figura 4.33. Información del número de teléfono asociado a la cuenta – Twilio

Tras haber configurado la cuenta con un número de teléfono, se selecciona en el menú principal “Send a WhatsApp message” (Figura 4.34).

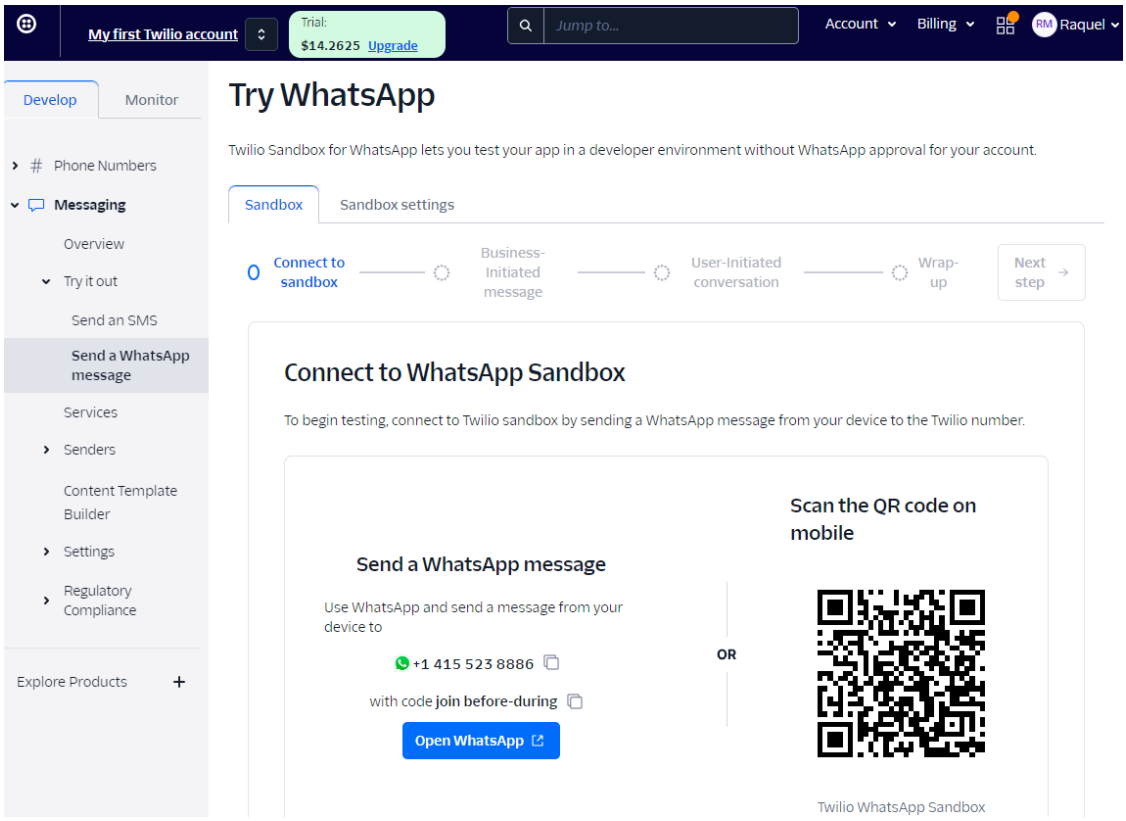


Figura 4.34. Conexión al Sandbox de WhatsApp – Twilio

Al escanear el QR con el teléfono móvil aparece la conversación con el contacto de Twilio, en la que se manda el mensaje “join main-there” (Figura 4.35) para poder empezar a recibir mensajes desde el número asociado en Twilio. Cabe destacar que la conexión al Sandbox de Twilio dura 72 horas, tras las que habrá que volver a enviar el mensaje.

Figura 4.35. Mensaje para la conexión al Sandbox de Twilio – WhatsApp

A continuación, se copia el link de la API del proyecto de ThingESP y se pega en el recuadro de “When a message comes in” del “Sandbox settings” de Twilio (Figura 4.36).

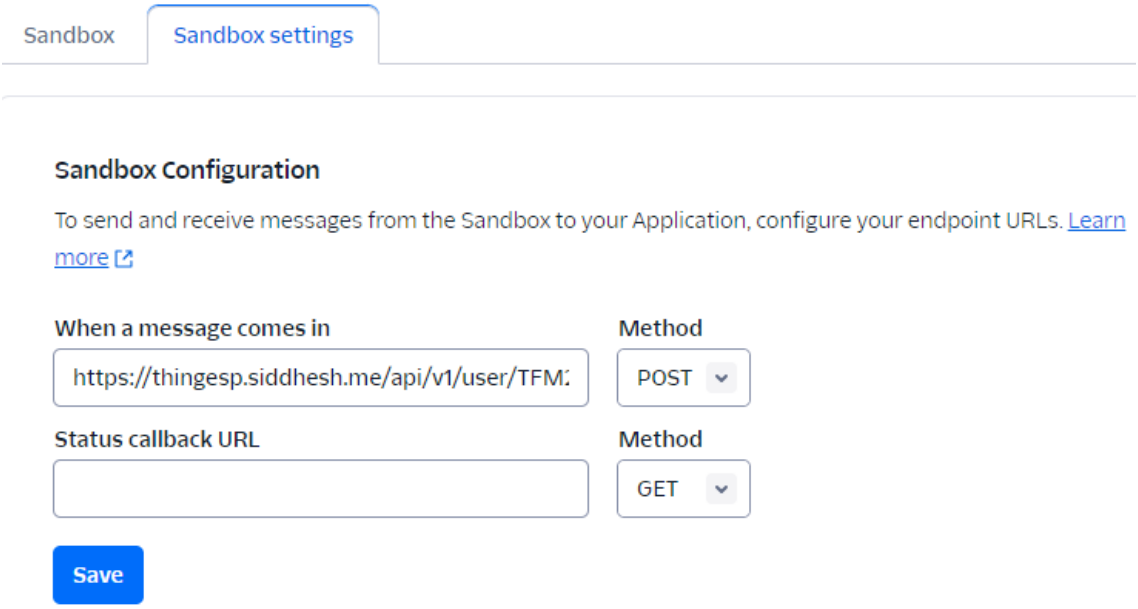


Figura 4.36. Conexión de ThingESP con Twilio – Twilio

Tras ello, se vuelve a la plataforma “ThingESP” para editar el proyecto, al cual se le habilitarán las “Device Calls” (Figura 4.37). Para ello, se rellena con la ID de la cuenta de Twilio y el AuthToken. El “Random Endpoint Token” se genera y rellena automáticamente.

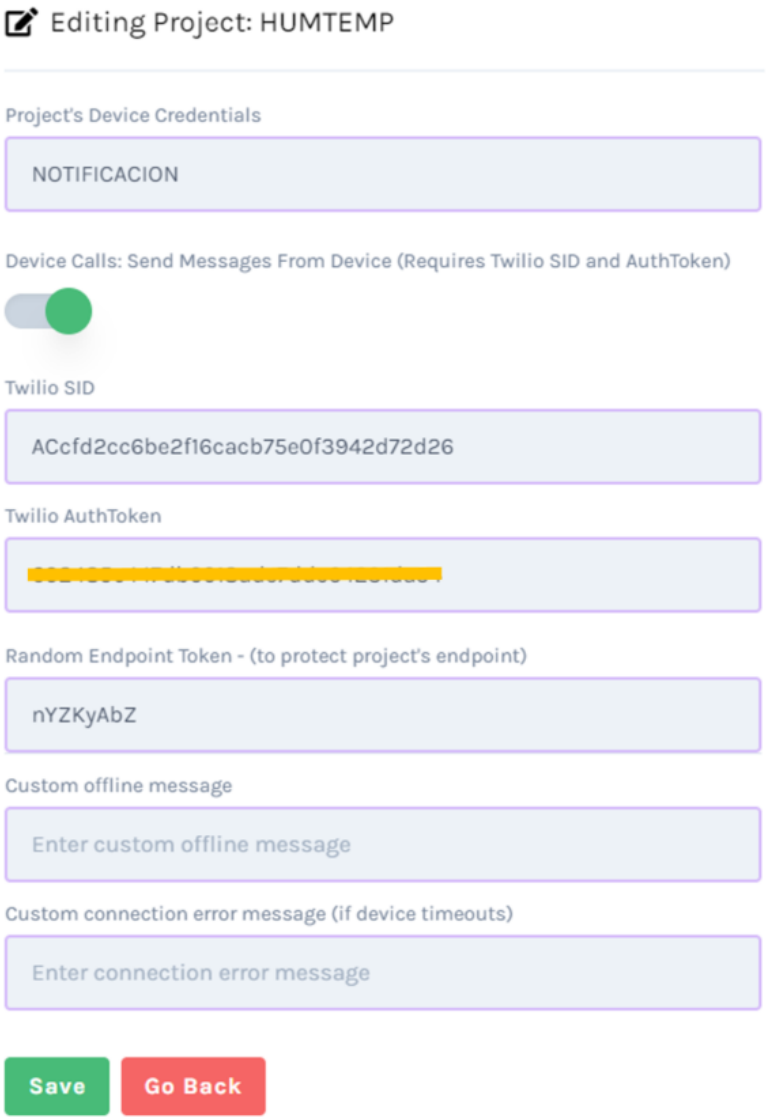


Figura 4.37. Habilitación de las “Device calls” – ThingESP

Tras completar estos pasos, únicamente falta el código en Arduino Cloud que se encargará de realizar la conexión con Twilio.

4.4.3.4. Código fuente

Al código utilizado en Arduino Cloud para el Thing “TFM_rmt_2” con Telegram (4.17, 4.18, 4.19, 4.20 y 4.21) se le añadirán las siguientes líneas de código explicadas a continuación.

Antes del setup, se incluirán las líneas de la Figura 4.38, donde se incluyen las librerías (líneas 17-22), en especial la de “Arduino_ConnectionHandler” para poder realizar la conexión con Twilio. Tras ello, se configura el objeto “ThingESP32” para conectar el dispositivo con el proyecto de la plataforma ThingESP (línea 25).

```
17  #include "thingProperties.h"
18  #include <ThingESP.h>
19  #include <Arduino_ConnectionHandler.h>
20  #include <WiFi.h>
21  #include <ArduinoIoTCloud.h>
22  #include <UniversalTelegramBot.h>
23
24  // Conexion del dispositivo con la plataforma
25  ThingESP32 thing("TFM2024", "HUMTEMP", "NOTIFICACION");
26
27  // DHT sensor library - Version: Latest
28  #include <DHT.h>
29  #include <DHT_U.h>
30
31  #define DHTPIN 4
32  #define DHTTYPE DHT11
```

Figura 4.38. Código fuente para recibir/enviar un mensaje de WhatsApp – Parte 1

En el setup (Figura 4.39), se inicializa el dispositivo “ThingESP32” (línea 97).

```
91    // Connect to Arduino IoT Cloud
92    ArduinoCloud.begin(ArduinoIoTPreferredConnection);
93
94    client.setCACert(TELEGRAM_CERTIFICATE_ROOT);
95    setDebugMessageLevel(2);
96    ArduinoCloud.printDebugInfo();
97    thing.initDevice();
```

Figura 4.39. Código fuente para recibir/enviar un mensaje de WhatsApp – Parte 2

Seguidamente, en el loop (Figura 4.40) se añade la instrucción “thing.Handle()” (línea 110), que servirá para gestionar la transferencia de datos entre el dispositivo y la plataforma Arduino Cloud. Esto hará que Twilio envíe el mensaje recibido a la URL de ThingESP para manejar las consultas y que esta plataforma se comunique con la placa para identificar la consulta.

```
100 void loop() {
101   ArduinoCloud.update();
102   delay(2000); // Espera 2 segundos entre medidas
103   dht_humedad = dht.readHumidity();
104   dht_temperatura = dht.readTemperature(); // Celsius
105   Serial.println(dht_humedad);
106   Serial.println(dht_temperatura);
107
108   // Gestion de la transferencia de datos entre el dispositivo
109   // y la plataforma IoT Cloud
110   thing.Handle();
111
112   enviarAlerta(dht_temperatura, dht_humedad);
113 }
```

Figura 4.40. Código fuente para recibir/enviar un mensaje de WhatsApp – Parte 3

Por último, se crea la función “HandleResponse” (Figura 4.41), la cual gestionará la respuesta de una consulta realizada por el usuario desde WhatsApp. En esta función, se indica que:

- Si se recibe un mensaje de “temperatura”, se le enviará el valor de la temperatura actual medido por el sensor DHT11.
- Si se recibe un mensaje de “humedad”, se le enviará el valor de la humedad actual medido por el sensor DHT11.
- Por el contrario, si se recibe un mensaje con cualquier otro valor, se le responderá con un “invalido”, indicando así al usuario que la consulta que intenta realizar no es válida.

Por lo tanto, esta función solo retornará datos cuando el usuario los pida, así como eligiendo los datos retornados según la consulta del usuario. El mensaje generado por esta función se enviará a través de Twilio al número de WhatsApp del usuario.

```
116  String HandleResponse(String consulta) {
117
118    String message_WP;
119
120    if (consulta == "temperatura") {
121      message_WP = "Temperatura: ";
122      message_WP.concat(dht_temperatura);
123      message_WP.concat("ºC");
124      return message_WP;
125    }
126    else if (consulta == "humedad") {
127      message_WP = "Humedad: ";
128      message_WP.concat(dht_humedad);
129      message_WP.concat("%");
130      return message_WP;
131    }
132    else{
133      message_WP = "Invalido";
134      return message_WP;
135    }
136  }
```

Figura 4.41. Código fuente para recibir/enviar un mensaje de WhatsApp – Parte 4

En la Figura 4.42 se puede observar un diagrama de flujo del funcionamiento del código en cuanto al envío y recepción de mensajes por WhatsApp.

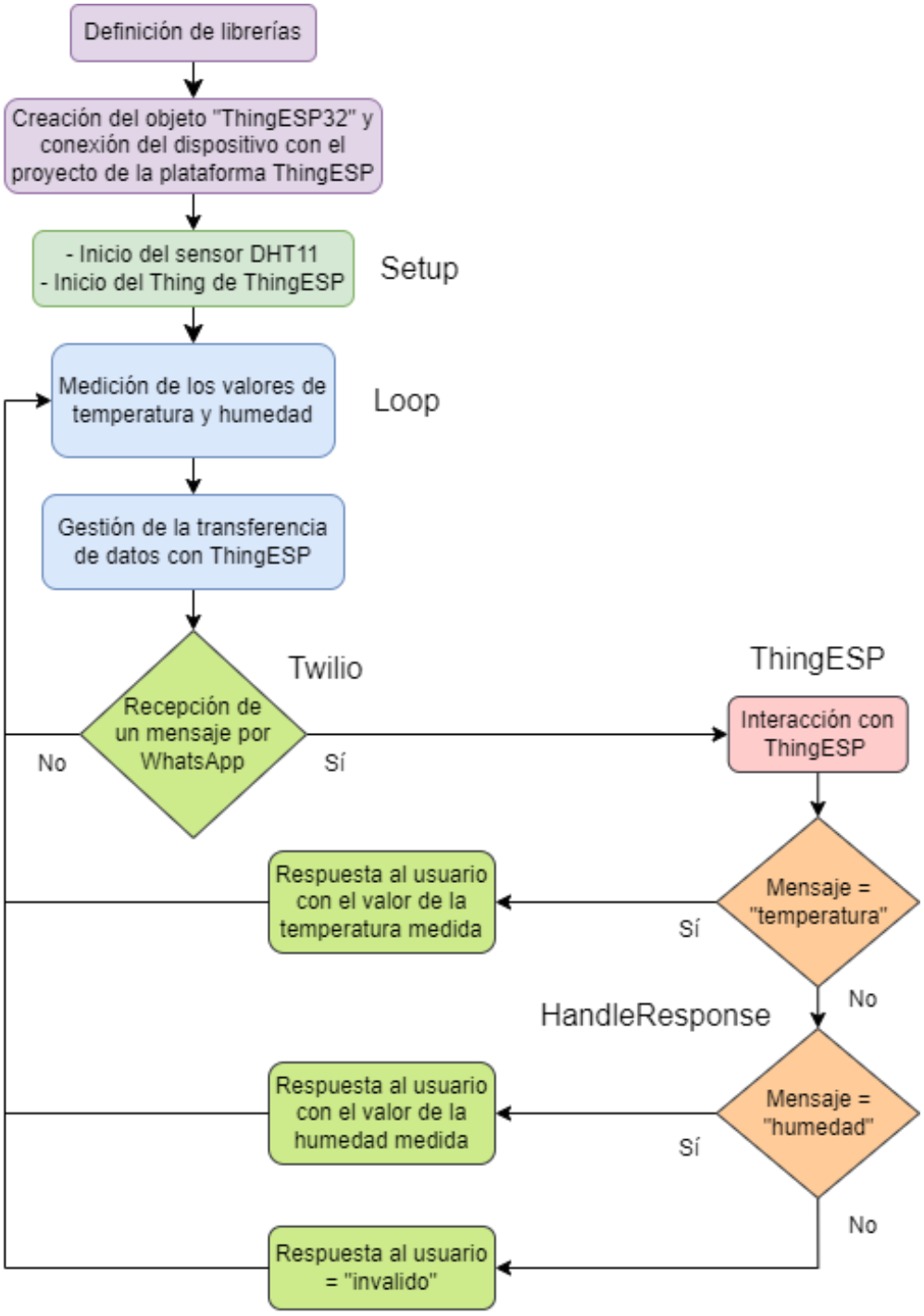


Figura 4.42. Diagrama de flujo para recibir/enviar un mensaje mediante WhatsApp

4.4.3.5. Comprobación

Para comprobar las peticiones por WhatsApp, basta con realizar los casos configurados en el código, como se puede ver en la Figura 4.43:

- Al enviar el mensaje “temperatura” en distintos momentos, se comprueba que envía la temperatura en tiempo real.
- Al enviar el mensaje “humedad”, se ve que se envían distintos valores correspondientes a las medidas realizadas en el momento.
- Por último, al enviar cualquier otro mensaje como “Hola”, responde con “invalido” sin enviar ningún otro dato.

Figura 4.43. Consultas mediante WhatsApp

4.5. Diagramas de flujo de los Things

Los códigos fuente completos de los Things “TFM_rmt” y “TFM_rmt_2” se encuentran en el Anexo II y en el Anexo III, respectivamente.

Lo que se muestra a continuación son los diagramas de flujo completos del funcionamiento de ambas placas, el Arduino Nano 33 IoT y el ESP32-WROOM-32.

4.5.1. Thing “TFM_rmt” – Arduino Nano 33 IoT

En la Figura 4.44 se puede observar el diagrama de flujo completo del Thing “TFM_rmt”.

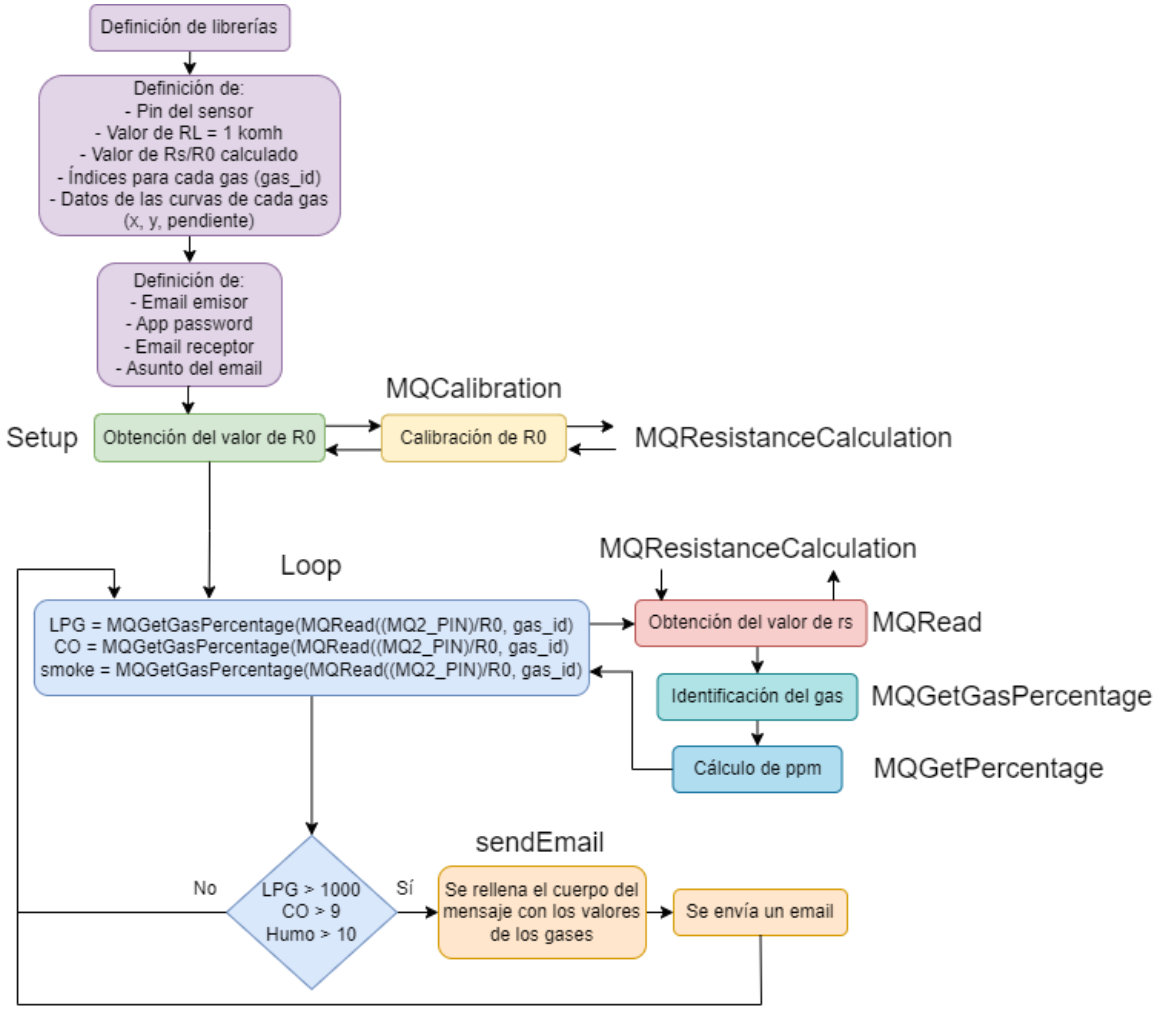


Figura 4.44. Diagrama de flujo completo del Thing “TFM_rmt”

4.5.2. Thing “TFM_rmt_2” – ESP32-WROOM-32

En la Figura 4.45 se puede observar el diagrama de flujo completo del Thing “TFM_rmt_2”.

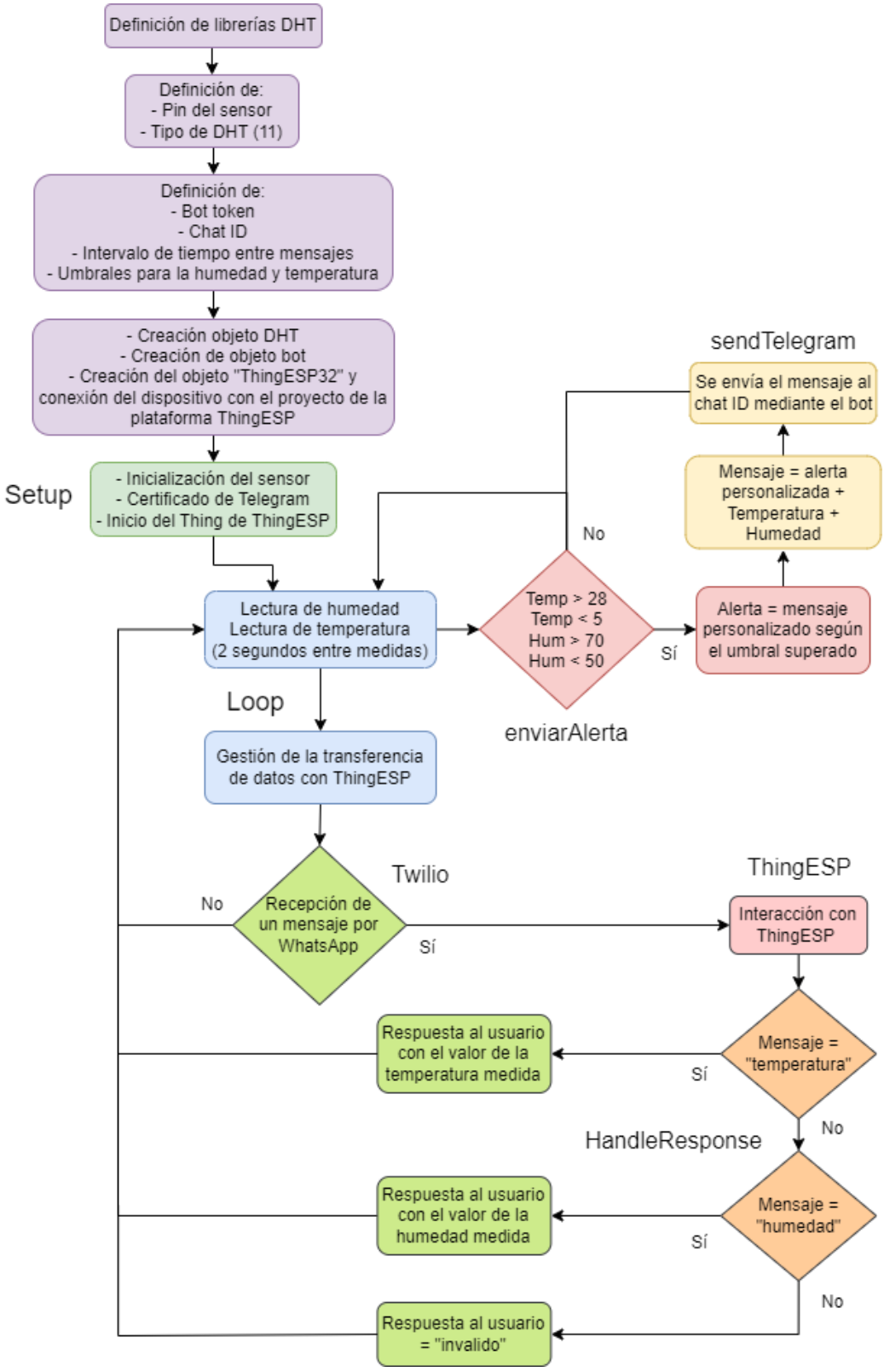


Figura 4.45. Diagrama de flujo completo del Thing “TFM_rmt_2”

Capítulo 5. Presupuesto

En este capítulo, únicamente se va a mostrar el presupuesto de este proyecto. En primer lugar, en la Tabla 5.1, se presenta el coste total del mismo con precios obtenidos de tiendas dedicadas específicamente a la venta de este tipo de componentes electrónicos.

		Precio
Placas	Arduino Nano 33 IoT [36]	22.80€
	ESP32-WROOM-32 [37]	7.87€
Sensores	Sensor de temperatura y humedad DHT11 (Keyestudio) [38]	8.53€
	Sensor de gas analógico MQ-2 (Keyestudio) [39]	7.00€
TOTAL		46.20€

Tabla 5.1. Presupuesto del proyecto

Se puede observar que es un proyecto barato, aunque se podría reducir aún más el coste si se utiliza otra placa del tipo ESP32 en lugar de una oficial de Arduino o comprarla en una tienda que no sea la oficial, ya que la mitad del presupuesto se destina a dicho componente. Además, se podrían comprar sensores de marcas más económicas, aunque seguramente disminuiría la fiabilidad de estos.

En la Tabla 5.2, se puede ver un presupuesto bastante reducido pero menos seguro en cuanto al funcionamiento de las partes del hardware, debido a que los elementos se obtienen de tiendas que ofrecen bajo coste y calidad cuestionable.

		Unidades	**Precio/ unidad**
Placas	ESP32-WROOM-32 [40]	2	2.27€
Sensores	Sensor de temperatura y humedad DHT11 [41]	1	2.33€
	Sensor de gas analógico MQ-2 [42]	1	1.89€
TOTAL			8.76€

Tabla 5.2. Presupuesto alternativo más barato

Por lo tanto, el usuario deberá realizar una solución de compromiso entre el coste y la fiabilidad de los materiales.

Capítulo 6. Conclusiones y líneas futuras

6.1. Conclusiones

En la actualidad, la automatización en la recopilación, inserción y análisis de datos es crucial para ahorrar retardos y mejorar los resultados obtenidos. En este proyecto, un sistema de medición del entorno del usuario, es importante minimizar el retardo en la detección de gases perjudiciales para la salud.

En concreto, el sensor que se ha elegido para ser utilizado por el Arduino Nano 33 IoT ha sido el sensor de gas analógico MQ-2, mientras que el sensor seleccionado para estar conectado al ESP32-WROOM-32 ha sido el sensor de temperatura y humedad DHT11.

Ambas placas, conectadas a la misma red Wi-Fi, transmiten los datos medidos a la plataforma de Arduino Cloud, la cual los almacena y los muestra al usuario mediante el uso de widgets en un dashboard común.

De forma individual, la placa de Arduino realizará alertas mediante correos electrónicos a través de Gmail comunicando valores anormales de los gases medidos. Por otra parte, la placa de ESP32 enviará alertas mediante mensajes instantáneos de Telegram cuando los valores de temperatura y humedad sobrepasen unos umbrales, además de poder consultar los valores medidos mediante mensajes de WhatsApp.

Finalmente, se han realizado las correspondientes pruebas de validación del prototipo para comprobar que, tanto los sensores, como la programación de las dos placas (Arduino y ESP32) y las notificaciones funcionaran correctamente, logrando así los objetivos previamente establecidos en este trabajo.

6.2. Líneas futuras

Una mejora podría ser integrar lo realizado en el proyecto con plataformas de hogar inteligente, como pueden ser Google Home, Amazon Alexa o Apple HomeKit para permitir un control centralizado y por voz.

También, se podrían implementar protocolos como MQTT o el más reciente Matter para comparar las posibilidades que éstos ofrecerían frente al sistema desarrollado en este proyecto.

Asimismo, se podrían automatizar acciones basadas en los datos de los sensores, como activar ventiladores o sistemas de purificación de aire, o se podrían implementar notificaciones basadas en la ubicación del usuario, activadas cuando el usuario entra o sale de una zona específica.

Por último, se podrían realizar los cambios necesarios en la electrónica para poder usar el sensor MQ-7 correctamente, especialmente la fase de 1.4V.

Referencias

Todas las referencias web han sido visitadas y están online a fecha de 20 de junio de 2024.

[1] «Diferencias entre el Internet de las Cosas y Domótica», Tutoriales Dongee. Accedido: 20 de mayo de 2024. [En línea]. Disponible en: https://www.dongee.com/tutoriales/internet-de-las-cosas-y-domotica/

[2] «¿Qué es Internet de las cosas? | IBM». Accedido: 20 de mayo de 2024. [En línea]. Disponible en: https://www.ibm.com/es-es/topics/internet-of-things

[3] «¿Qué es IoT? - Explicación del Internet de las cosas - AWS», Amazon Web Services, Inc. Accedido: 20 de mayo de 2024. [En línea]. Disponible en: https://aws.amazon.com/es/what-is/iot/

[4] «¿Qué es la domótica y para qué sirve?», REPSOL. Accedido: 20 de mayo de 2024. [En línea]. Disponible en: https://www.repsol.com/es/energia-futuro/tecnologia-innovacion/que-es-la-domotica/index.cshtml

[5] «Domótica, ¿qué es y cómo funciona?», PcComponentes. Accedido: 20 de mayo de 2024. [En línea]. Disponible en: https://www.pccomponentes.com/que-es-domotica

[6] «Arduino NANO 33 IoT | Arduino.cl - Compra tu Arduino en Línea». Accedido: 26 de mayo de 2024. [En línea]. Disponible en: https://arduino.cl/producto/arduino-nano-33-iot/

[7] «Arduino Nano 33 IoT», Arduino Official Store. Accedido: 26 de mayo de 2024. [En línea]. Disponible en: https://store.arduino.cc/products/arduino-nano-33-iot

[8] «Arduino Na]no 33 IoT Board layout | Arduino, Iot, Iot projects», Pinterest. Accedido: 26 de mayo de 2024. [En línea]. Disponible en: https://www.pinterest.com/pin/arduino-nano-33-iot-pinoutpin-diagram--802344489872655397/

[9] «Guía de modelos Arduino y sus características | Arduino Nano 33 IoT | BricoGeek Lab», Modelos de Arduino. Accedido: 27 de mayo de 2024. [En línea]. Disponible en: https://lab.bricogeek.com/tutorial/guia-de-modelos-arduino-y-sus-caracteristicas/arduino-nano-33-iot

[10] «Módulo ESP-WROOM-32 ESP32 WiFi», Naylamp Mechatronics - Perú. Accedido: 26 de mayo de 2024. [En línea]. Disponible en: https://naylampmechatronics.com/espressif-esp/382-modulo-esp-wroom-32-esp32-wifi.html

[11] «Introduction to ESP32». Accedido: 5 de junio de 2024. [En línea]. Disponible en: https://www.tutorialspoint.com/esp32_for_iot/esp32_for_iot_introduction.htm

[12] A. Cortés, «Tutorial ESP32», Acortes Software. Accedido: 5 de junio de 2024. [En línea]. Disponible en: https://acortes.co/tutorial-esp32/

[13] «SEN0132», DigiKey Electronics. Accedido: 4 de junio de 2024. [En línea]. Disponible en: https://www.digikey.es/es/products/detail/dfrobot/SEN0132/6588532

[14] «sen0132.pdf». Accedido: 4 de junio de 2024. [En línea]. Disponible en: https://www.application-datasheet.com/pdf/dfrobot/sen0132.pdf

[15] «SEN0132 DFRobot | Mouser», Mouser Electronics. Accedido: 4 de junio de 2024. [En línea]. Disponible en: https://www.mouser.es/ProductDetail/DFRobot/SEN0132?qs=Zcin8yvlhnMRSldE57IDmQ%3D%3D

[16] «MQ_7-2487908.pdf». Accedido: 4 de junio de 2024. [En línea]. Disponible en: https://www.mouser.es/datasheet/2/830/MQ_7-2487908.pdf

[17] «Sensor de gas MQ2 MQ-2 para Arduino», CREATEC4 S.C.A. Accedido: 27 de mayo de 2024. [En línea]. Disponible en: https://createc3d.com/es/modulos-y-sensores/660-comprar-sensor-de-gas-mq2-mq-2-para-arduino-precio-oferta.html

[18] «Kit de sensores 48 en 1 Keyestudio — Documentación del kit de sensores 48 en 1». Accedido: 27 de mayo de 2024. [En línea]. Disponible en: https://docs.keyestudio.com/projects/KS0522/en/latest/KS0522.html

[19] «605-00008-MQ-2-Datasheet-370464.pdf». Accedido: 30 de mayo de 2024. [En línea]. Disponible en: https://www.mouser.com/datasheet/2/321/605-00008-MQ-2-Datasheet-370464.pdf

[20] «Sensor de temperatura y humedad relativa DHT11», Naylamp Mechatronics - Perú. Accedido: 27 de mayo de 2024. [En línea]. Disponible en: https://naylampmechatronics.com/sensores-temperatura-y-humedad/57-sensor-de-temperatura-y-humedad-relativa-dht11.html

[21] «Como utilizar el sensor de humedad DHT11 con un Arduino. – Electrónica Especializada». Accedido: 27 de mayo de 2024. [En línea]. Disponible en: https://www.aladuino.com.mx/blog/como-utilizar-el-sensor-de-humedad-dht11-con-un-arduino/
[22] «Introducción a Arduino Cloud», Maker Space Zaragoza. Accedido: 28 de mayo de 2024. [En línea]. Disponible en: https://zaragozamakerspace.com/event/arduino-iot-cloud/
[23] «¡Vamos a conectarnos! ... | Librería CATEDU». Accedido: 28 de mayo de 2024. [En línea]. Disponible en: https://libros.catedu.es/books/microcontroladores-vestibles-y-conectados-a-internet/page/vamos-a-conectarnos-parte-3
[24] Jadsa Tech, *SERIE SENSORES Y MODULOS #17: SENSOR GAS MQ - CALCULO DE PPM - CALIBRACION - MQ2*, (25 de abril de 2019). Accedido: 30 de mayo de 2024. [En línea]. Disponible en: https://www.youtube.com/watch?v=ihxuKCD-zFw
[25] «IoT Remote App by Arduino Cloud». Accedido: 1 de junio de 2024. [En línea]. Disponible en: https://cloud.arduino.cc/iot-remote-app/
[26] «¿Qué es Gmail y para qué sirve? - Definición», GEEKNETIC. Accedido: 2 de junio de 2024. [En línea]. Disponible en: https://www.geeknetic.es/Gmail/que-es-y-para-que-sirve
[27] «Gmail: Correo electrónico gratuito, privado y seguro | Google Workspace». Accedido: 2 de junio de 2024. [En línea]. Disponible en: https://www.google.com/intl/es-419/gmail/about/
[28] eveningstem, *Sending an Email or Text using an Arduino Nano 33 IoT*, (17 de junio de 2021). Accedido: 2 de junio de 2024. [En línea]. Disponible en: https://www.youtube.com/watch?v=GFDTMFBPCn4
[29] «FDS-Gas-LP-V-3-2020_1.pdf». Accedido: 2 de junio de 2024. [En línea]. Disponible en: https://www.recope.go.cr/wp-content/uploads/2020/10/FDS-Gas-LP-V-3-2020_1.pdf
[30] «CDC - Publicaciones de NIOSH - Prevención de envenenamiento con monóxido de carbono producido por herramientas y equipos con motores pequeños de gasolina (96-118)». Accedido: 2 de junio de 2024. [En línea]. Disponible en: https://www.cdc.gov/spanish/niosh/docs/96-118_sp/default.html
[31] «Preguntas frecuentes», Telegram. Accedido: 5 de junio de 2024. [En línea]. Disponible en: https://telegram.org/faq
[32] «WhatsApp», WhatsApp.com. Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://www.whatsapp.com/join
[33] «ThingESP». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://thingesp.siddhesh.me/#/
[34] «¿Qué es Twilio? ¿Cómo funciona esta plataforma? | SEIDOR». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://www.seidor.com/blog/twilio-que-es
[35] «¿Qué hace Twilio?», Twilio. Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://www.twilio.com/es-mx/blog/que-hace-twilio
[36] «Arduino Nano 33 IoT», Arduino Official Store. Accedido: 17 de junio de 2024. [En línea]. Disponible en: https://store.arduino.cc/products/arduino-nano-33-iot
[37] «NodeMCU ESP32 Wroom WIFI + Bluetooth BricoGeek | BricoGeek.com». Accedido: 17 de junio de 2024. [En línea]. Disponible en: https://tienda.bricogeek.com//arduino-compatibles/1274-nodemcu-esp32-wroom-wifi-bluetooth.html?srsltid=AfmBOor7j1OG1_arZVZtvAZ220AVJiXbjyZO1hkBXsFN31T57hMc1P5XdxE
[38] «Módulo sensor de temperatura y humedad DHT11 para Arduino Keyestudio», ABC Escolar. Accedido: 17 de junio de 2024. [En línea]. Disponible en: https://abcescolar.pt/es/products/modulo-sensor-de-temperatura-e-humidade-dht11-para-arduino-keyestudio
[39] «Módulo Sensor de Gas Analógico (MQ2) para Arduino (Conexión Fácil) Key – ABC Escolar». Accedido: 17 de junio de 2024. [En línea]. Disponible en: https://abcescolar.pt/es/products/modulo-sensor-de-gas-analogico-mq2-para-arduino-ligacao-easy-keyestudio?srsltid=AfmBOoqm2uiPSziLJYdJekuOb1_EItMcXRxBEIPbS3H4yXQ27U_tDS0lV_w
[40] «Placa de desarrollo ESP32 de 1 piezas, WiFi + Bluetooth, consumo de energía ultrabajo, doble núcleo, ESP-32S, ESP32-WROOM-32D, ESP32-WROOM-32U, ESP 32», aliexpress. Accedido: 17 de junio de 2024. [En línea]. Disponible en: https://es.aliexpress.com/item/?src=ibdm_d03p0558e02r02&sk=&aff_platform=&aff_trace_key=&af=&cv=&cn=&dp=

[41] «Módulo de Sensor de temperatura y humedad, de 3 pines Dispositivo inteligente, piezas DHT11 Digital, PCB para Arduino, Kit DIY, 37 en 1, 1/5 KY-015», aliexpress. Accedido: 17 de junio de 2024. [En línea]. Disponible en: https://es.aliexpress.com/item/?src=ibdm_d03p0558e02r02&sk=&aff_platform=&aff_trace_key=&af=&cv=&cn=&dp=
[42] «Módulo Detector de Gas MQ-2 MQ2 LPG, butano, hidrógeno, para arduino, 1 ud.», aliexpress. Accedido: 17 de junio de 2024. [En línea]. Disponible en: https://es.aliexpress.com/item/?src=ibdm_d03p0558e02r02&sk=&aff_platform=&aff_trace_key=&af=&cv=&cn=&dp=
[43] «Arduino Create Plugin - Getting Started». Accedido: 28 de mayo de 2024. [En línea]. Disponible en: https://create.arduino.cc/getting-started/plugin/welcome

Anexo I. Instalación del Agente de Arduino

Primero, se accede a la página web oficial que se puede ver en la Figura I.1. [43]

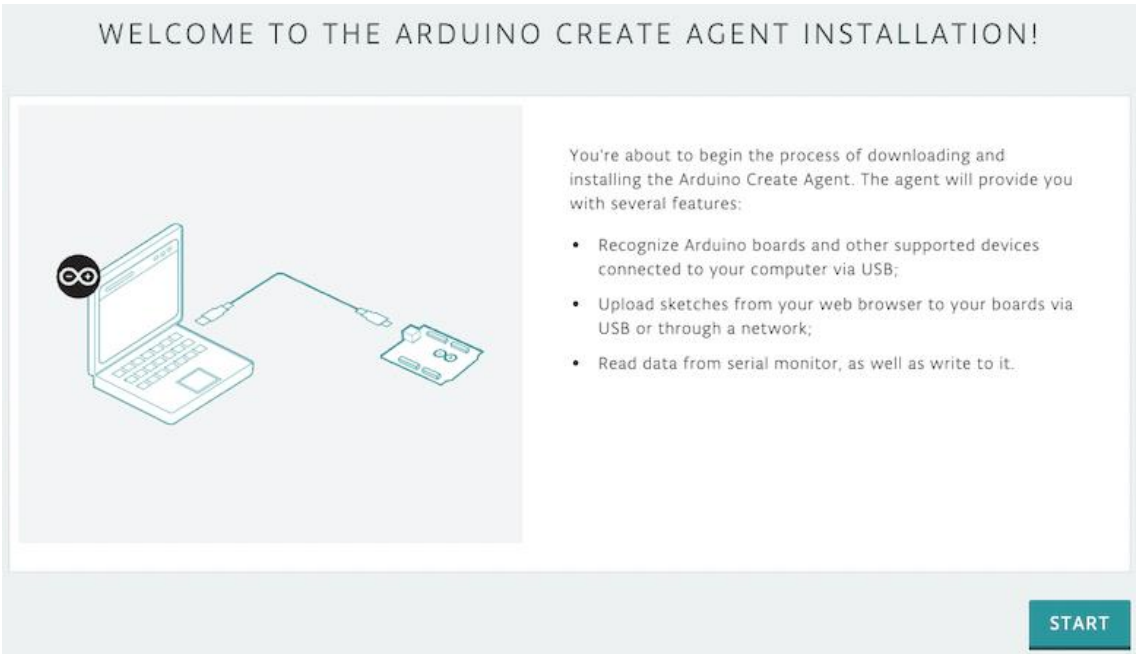


Figura I.1. Página de descarga del Arduino Create Agent [43]

Seguidamente, se selecciona “START” para poder empezar a descargar el agente de Arduino (Figura I.2). En este caso, se descarga para “WIN64”.

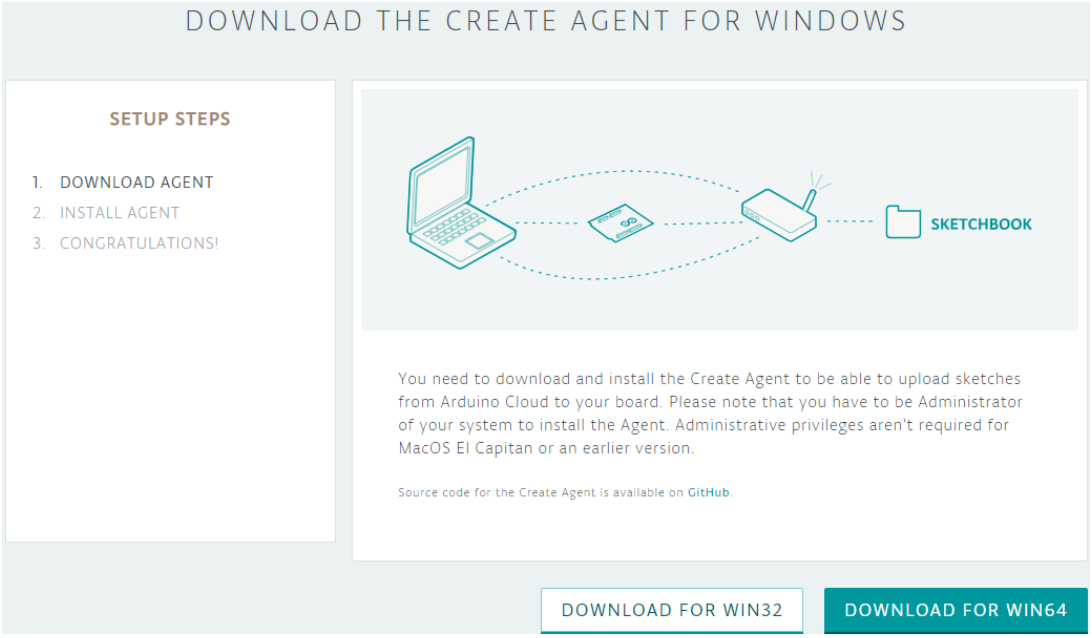


Figura I.2. Descarga del Arduino Create Agent [43]

Tras ello, se habrá completado la descarga. Mientras, en la página web aparecerá que se está buscando el agente de Arduino para comprobar si se ha instalado correctamente (Figura I.3).

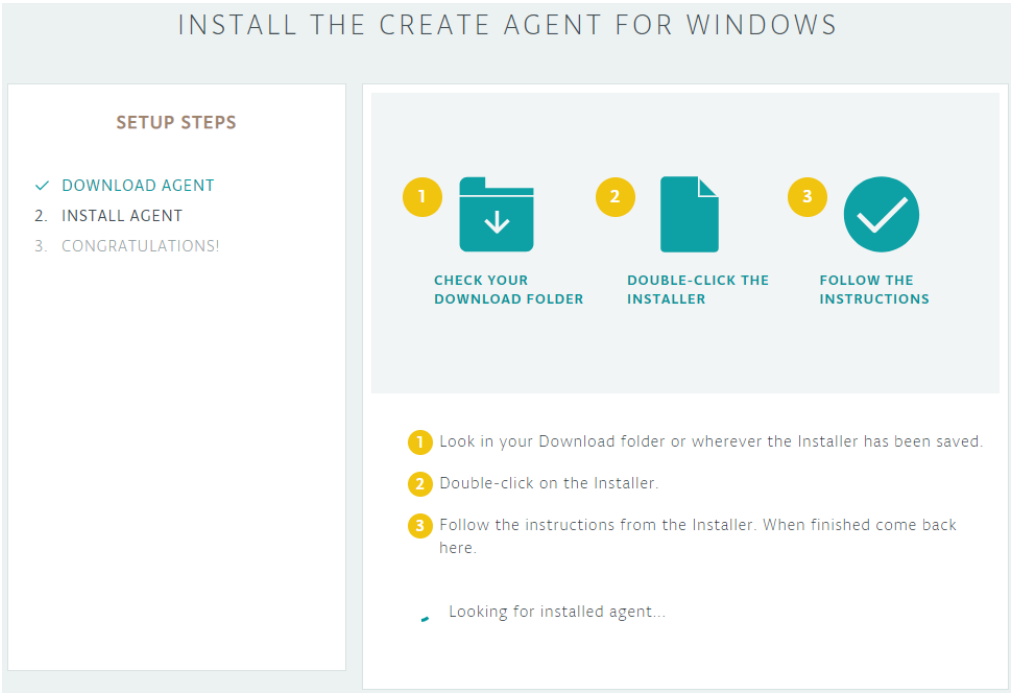


Figura I.3. Búsqueda del agente de Arduino [43]

Por lo tanto, se procede a instalar este agente pulsando doble click en el instalador y seleccionando “Next” además de aceptar los términos y condiciones.

Una vez instalado, en la página web de descarga aparecerá que se ha detectado y que está correctamente instalado (Figura I.4). Además, se podrá pulsar en un botón que lleva directamente al editor de Arduino web.

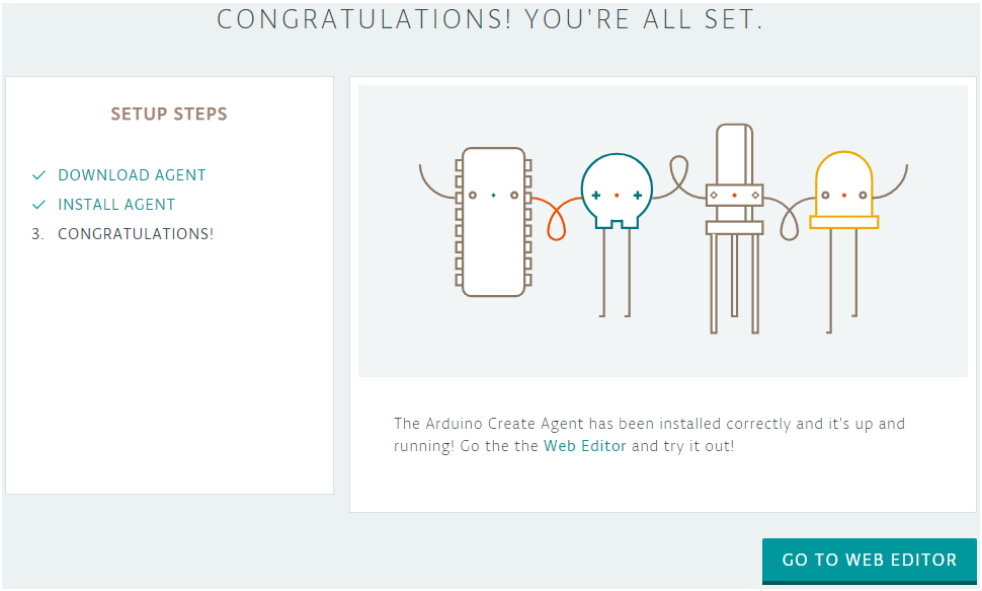


Figura I.4. Agente de Arduino instalado [43]

Anexo II. Código fuente completo del Thing “TFM_rmt” – Arduino Nano 33 IoT

```
/*
  Sketch generated by the Arduino IoT Cloud Thing "Untitled"
  https://create.arduino.cc/cloud/things/0000xxxx-0000-0000-0000-x00x00x00000

  Arduino IoT Cloud Variables description

  The following variables are automatically generated and updated when changes
are made to the Thing

  float gas_CO_ppm;
  float gas_LPG_ppm;
  float gas_smoke_ppm;

  Variables which are marked as READ/WRITE in the Cloud Thing will also have
functions
  which are called when their values are changed from the Dashboard.
  These functions are generated with the Thing and added at the end of this
sketch.
*/
#include "thingProperties.h"

//HARDWARE
const int calibrationLed = 13; //Se encendera durante la calibracion
const int MQ2_PIN = A0;
int RL_value = 1; //kohm
float R0_CLEAN_AIR_FACTOR = 9.79; //ratio calculado en el anterior sketch

//SOFTWARE
int CALIBRATION_SAMPLE_TIMES = 50;
int CALIBRATION_SAMPLE_INTERVAL = 500;

int READ_SAMPLE_INTERVAL = 50;
int READ_SAMPLE_TIMES = 5;


//GMAIL
#include <WiFiNINA.h>
#include <EMailSender.h>
char eMailUser[] = "emisor@gmail.com";
char eMailPass[] = "xxxx xxxx xxxx xxxx";  // App password
char eMailRecipient[] = "receptor@gmail.com";  // Receptor

String TEXT; // Cuerpo del email
const char SUBJ[] = "Peligro - Altos niveles de gases detectados"; // Asunto
del email


#define GAS_LPG 0
#define GAS_CO 1
#define GAS_SMOKE 2
```

```
//Se toman 2 puntos de la curva de la grafica y se pone el formato:
//{x,y,pendiente}
float LPG_curve[3] = {2.3, 0.21, -0.47};
float CO_curve[3] = {2.3, 0.72, -0.34};
float Smoke_curve[3] = {2.3, 0.53, -0.44};

float R0 = 10; //R0 se inicializa a 10kohm

WiFiClient client; // Crea el cliente para Wi-Fi

void setup() {
  Serial.begin(9600);

  pinMode(calibrationLed, OUTPUT);
  digitalWrite(calibrationLed, HIGH);
  Serial.print("Calibrando...");

  R0 = MQCalibration(MQ2_PIN);
  digitalWrite(calibrationLed, LOW);
  Serial.println("Terminado!");
  Serial.print("R0 = ");
  Serial.print(R0);
  Serial.println("kohm\n");

  delay(1500);

  // Defined in thingProperties.h
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();
}

void loop() {
  ArduinoCloud.update();

  gas_LPG_ppm = MQGetGasPercentage(MQRead(MQ2_PIN)/R0, GAS_LPG);
  gas_CO_ppm = MQGetGasPercentage(MQRead(MQ2_PIN)/R0, GAS_CO);
  gas_smoke_ppm = MQGetGasPercentage(MQRead(MQ2_PIN)/R0, GAS_SMOKE);

  Serial.println("*********** CONCENTRACION DE GASES ***********");
  Serial.print("LPG: ");
  Serial.print(gas_LPG_ppm);
  Serial.println(" ppm");

  Serial.print("CO: ");
  Serial.print(gas_CO_ppm);
  Serial.println(" ppm");

  Serial.print("Smoke: ");
  Serial.print(gas_smoke_ppm);
  Serial.println(" ppm");
  Serial.println("*********************************************");
  Serial.println();
```

```
  delay(500);

  if (gas_LPG_ppm > 1000.0 || gas_CO_ppm > 9.0 || gas_smoke_ppm > 10.0){
    sendEmail();
  }
}


float MQResistanceCalculation(int raw_adc){
  return ( ((float)RL_value*(1023-raw_adc)/raw_adc));
}


float MQCalibration(int mq_pin){
  int i;
  float val = 0;

  for(i = 0; i < CALIBRATION_SAMPLE_TIMES; i++){
    val += MQResistanceCalculation(analogRead(mq_pin));
    delay(CALIBRATION_SAMPLE_INTERVAL);
  }
  val = val / CALIBRATION_SAMPLE_TIMES;
  val = val / R0_CLEAN_AIR_FACTOR;

  return val;
}

float MQRead(int mq_pin){
  int i;
  float rs = 0;

  for(i = 0; i < READ_SAMPLE_TIMES; i++){
    rs += MQResistanceCalculation(analogRead(mq_pin));
    delay(READ_SAMPLE_INTERVAL);
  }
  rs = rs / READ_SAMPLE_TIMES;

  return rs;
}

long MQGetGasPercentage(float rs_ro_ratio, int gas_id){
  if(gas_id = GAS_LPG){
    return MQGetPercentage(rs_ro_ratio, LPG_curve);
  } else if(gas_id = GAS_CO){
    return MQGetPercentage(rs_ro_ratio, CO_curve);
  } else if(gas_id = GAS_SMOKE){
    return MQGetPercentage(rs_ro_ratio, Smoke_curve);
  }

  return 0;
}

//Calculo de PPM
long MQGetPercentage(float rs_ro_ratio, float *pcurve){
  return (pow(10,(((log(rs_ro_ratio)-pcurve[1])/pcurve[2])+pcurve[0])));
}
```

```
void sendEmail(){
    // Nombre del emisor del email
    char senderName[] = "Arduino Nano 33 IoT";

    TEXT = "Se han detectado los siguientes niveles de gases:<br>"
      "LPG = " + String(gas_LPG_ppm) + "ppm <br>" +
      "CO = " + String(gas_CO_ppm) + "ppm <br>" +
      "Humo = " + String(gas_smoke_ppm) + "ppm <br>";

    // Se envia el email
    EMailSender emailSend(eMailUser, eMailPass, eMailUser, senderName);
    EMailSender::EMailMessage msg;
    EMailSender::Response resp;
    Serial.println("Sending email...");
    msg.subject = SUBJ;
    msg.message = TEXT;
    resp = emailSend.send(eMailRecipient, msg);

    Serial.println("Sending status: ");
    Serial.print(resp.status);
    Serial.println(resp.code);
    Serial.println(resp.desc);
    Serial.println("");
    Serial.print("FROM: ");
    Serial.println(eMailUser);
    Serial.print("TO: ");
    Serial.println(eMailRecipient);
    Serial.print("SUBJECT: ");
    Serial.println(SUBJ);
    Serial.print("DATA:");
    Serial.println(TEXT);
    Serial.println("");
}
```

Anexo III. Código fuente completo del Thing “TFM_rmt_2” – ESP32-WROOM-32

```
/*
  Sketch generated by the Arduino IoT Cloud Thing "Untitled"
  https://create.arduino.cc/cloud/things/0000xxxx-0000-0000-0000-x00x00x00000

  Arduino IoT Cloud Variables description

  The following variables are automatically generated and updated when changes
are made to the Thing

  float dht_temperatura;
  int dht_humedad;

  Variables which are marked as READ/WRITE in the Cloud Thing will also have
functions
  which are called when their values are changed from the Dashboard.
  These functions are generated with the Thing and added at the end of this
sketch.
*/

#include "thingProperties.h"
#include <ThingESP.h>
#include <Arduino_ConnectionHandler.h>
#include <WiFi.h>
#include <ArduinoIoTCloud.h>
#include <UniversalTelegramBot.h>

// Conexion del dispositivo con la plataforma
ThingESP32 thing("TFM2024", "HUMTEMP", "NOTIFICACION");

// DHT sensor library - Version: Latest
#include <DHT.h>
#include <DHT_U.h>

#define DHTPIN 4
#define DHTTYPE DHT11

DHT dht(DHTPIN,DHTTYPE);

// Inicializacion del Telegram BOT
#define BOTtoken "0000000000:xxx0x00xxxxOxxx0xxx0xxxx00xxxxxxxxx"

// Bot Token (Obtenido del Botfather)
#define CHAT_ID "0000000000"

int status = WL_IDLE_STATUS;

WiFiClientSecure client;
UniversalTelegramBot bot(BOTtoken, client);

// Variables para controlar el tiempo entre envíos
```

```
unsigned long lastTelegramTime = 0;
const unsigned long telegramInterval = 20000; // 20 segundos

// Umbrales para alertas
const float TEMP_ALTA_UMBRAL = 28.0;
const float TEMP_BAJA_UMBRAL = 5.0;
const int HUMEDAD_ALTA_UMBRAL = 70;
const int HUMEDAD_BAJA_UMBRAL = 50;


void setup() {
  delay(5000);
  // Inicializacion del puerto serie
  Serial.begin(9600);
  while (!Serial) {
    ; // Espera para que se conecte
  }

  dht.begin();

  // Defined in thingProperties.h
  initProperties();

  // Connect to WiFi
  WiFi.begin(SECRET_SSID, SECRET_OPTIONAL_PASS);

  int attempt = 0;
  while (WiFi.status() != WL_CONNECTED && attempt < 20) {
    delay(500);
    Serial.print(".");
    attempt++;
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\nConnected to WiFi");
    Serial.print("IP Address: ");
    Serial.println(WiFi.localIP());
  } else {
    Serial.println("\nFailed to connect to WiFi");
    return; // No continuar si no hay conexión WiFi
  }


  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);

  client.setCACert(TELEGRAM_CERTIFICATE_ROOT);
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();
  thing.initDevice();
}

void loop() {
  ArduinoCloud.update();
  delay(2000); // Espera 2 segundos entre medidas
  dht_humedad = dht.readHumidity();
  dht_temperatura = dht.readTemperature(); // Celsius
```

```
  Serial.println(dht_humedad);
  Serial.println(dht_temperatura);

  // Gestion de la transferencia de datos entre el dispositivo
  // y la plataforma IoT Cloud
  thing.Handle();

  enviarAlerta(dht_temperatura, dht_humedad);
}


String HandleResponse(String consulta) {

  String message_WP;

  if (consulta == "temperatura") {
    message_WP = "Temperatura: ";
    message_WP.concat(dht_temperatura);
    message_WP.concat("ºC");
    return message_WP;
  }
  else if (consulta == "humedad") {
    message_WP = "Humedad: ";
    message_WP.concat(dht_humedad);
    message_WP.concat("%");
    return message_WP;
  }
  else{
    message_WP = "Invalido";
    return message_WP;
  }
}


void enviarAlerta(float temperature, int humidity) {
  unsigned long currentMillis = millis();

  if (currentMillis - lastTelegramTime >= telegramInterval) {
    String alerta;

    if (temperature >= TEMP_ALTA_UMBRAL) {
      alerta = "Temperatura alta detectada\n";
    } else if (temperature <= TEMP_BAJA_UMBRAL) {
      alerta = "Temperatura baja detectada\n";
    } else if (humidity >= HUMEDAD_ALTA_UMBRAL) {
      alerta = "Humedad alta detectada\n";
    } else if (humidity <= HUMEDAD_BAJA_UMBRAL) {
      alerta = "Humedad baja detectada\n";
    } else {
      // Si no hay alerta, no envíes nada
      return;
    }

    sendTelegram(alerta, temperature, humidity);
    lastTelegramTime = currentMillis; // Actualiza el tiempo del último mensaje
enviado
  }
```

```
}


void sendTelegram(String alerta, float temperatura, int humedad)  {
  Serial.println("SENDING A TELEGRAM MESSAGE...");
  alerta.concat("\nTemperatura: ");
  alerta.concat(temperatura);
  alerta.concat("ºC\nHumedad: ");
  alerta.concat(humedad);
  alerta.concat("%");
  bot.sendMessage(CHAT_ID, alerta);

  Serial.println("Mensaje enviado");
}
```