Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Desarrollo de aplicaciones para IoT con el módulo ESP32

Autor: Álvaro Benito Herranz

Director: José Manuel Villadangos Carrizo

2019

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

Trabajo Fin de Grado

Desarrollo de aplicaciones para IoT con el módulo ESP32

Autor: Álvaro Benito Herranz

Director: José Manuel Villadangos Carrizo

TRIBUNAL:

Presidente: Luciano Boquete Vázquez

Vocal 1º: Francisco Javier Rodríguez Sánchez

Vocal 2º: José Manuel Villadangos Carrizo

2019

Índice

Índice de figuras

Índice de tablas

Índice de códigos

Resumen

El presente TFG tiene como objetivo documentar información técnica sobre el módulo ESP32, y explicar el desarrollo de una sencilla aplicación orientada a Internet de las Cosas (IoT) en el que se utilice el dispositivo.

Se trata de un microcontrolador que integra tecnologías WiFi y Bluetooth, que le proporcionan conectividad con internet u otros dispositivos. Se describirán sus principales características y funcionalidades, así como también los diferentes entornos de desarrollo y lenguajes de programación que permiten programar el dispositivo.

Respecto a la parte de desarrollo de la aplicación, se medirán datos ambientales como la temperatura y la humedad, que podrán ser visualizados a través de diferentes plataformas como un servidor Web, ThingSpeak, Adafruit.io y Twitter. Además, como actuador se tiene un array de relés que podrá ser controlado desde el servidor Web, desde Adafruit.io y mediante comandos de voz de Google Assistant con el servicio IFTTT. Estos dos últimos se llevan a cabo mediante MQTT, que se trata de un protocolo orientado a Internet de las Cosas.

Palabras clave

ESP32, IoT, ThingSpeak, MQTT, IFTTT.

Abstract

The objective of this proyect is to document technical information about the ESP32 module, and also to explain the development of a simple application based on Internet Of Things (IoT), in which the device is used.

The ESP32 is a microcontroller with Integrated WiFi and Bluetooth technologies, which provides conectivity with Internet or other devices. Its main features and functionalities will be described, as well as the different development frameworks and programming languages that allow programming the device.

About the development of the application, environmental data such as temperature or humedity will be measured, which can be visualized through different platforms such as a Web server, ThingSpeak, Adafruit.io and Twitter. In addition, there is a relay array that can be controlled from the Web server, from Adafruit.io and through Google Assistant voice commands with the IFTTT service. These last two services use MQTT, which is a protocol oriented to the Internet of Things.

Keywords

ESP32, IoT, ThingSpeak, MQTT, IFTTT.

Resumen extendido

El presente TFG tiene como objetivo documentar información técnica sobre el módulo ESP32, y explicar el desarrollo de una sencilla aplicación orientada a Internet de las Cosas (IoT) en el que se utilice el dispositivo.

Se trata de un microcontrolador que integra tecnologías WiFi y Bluetooth, que le proporcionan conectividad con internet u otros dispositivos. El hecho de implementar tecnología Bluetooth le hace destacar sobre sus rivales, al tratarse del módulo más barato que integra dichas tecnologías, que, junto al doble procesador y al modo de bajo consumo, lo hacen un rival muy fuerte en el mercado de los módulos de desarrollo orientados a IoT.

Se mencionarán los principales chips de la familia ESP32, así como también los módulos y las tarjetas de desarrollo que utilizan dicho chip, junto a sus principales características, como el tamaño, el tipo de antena o el tamaño de memoria. A continuación, se describirá el módulo ESP32 de forma general donde se verán sus especificaciones detalladas, como los procesadores, la memoria, los bloques de radiofrecuencia y los principales periféricos y sensores que monta el dispositivo. También se describirá la funcionalidad WiFi y Bluetooth del dispositivo y la funcionalidad ESP-MESH, basada en el protocolo MESH.

Posteriormente se hará un análisis de los diferentes entornos de desarrollo y lenguajes de programación que permiten programar el dispositivo, como por ejemplo el entorno Arduino, el entorno ESP-IDF o lenguajes como Python o Javascript.

El módulo ESP32 es el sucesor de otro módulo anterior que le dio fama a la compañía creadora de estos dispositivos. Se trata del módulo ESP8266. Se verán las principales diferencias entre ambos dispositivos.

Respecto a la parte de desarrollo de la aplicación, se medirán datos ambientales como la temperatura y la humedad con el sensor BME280, que podrán ser visualizados a través de diferentes plataformas: desde un servidor Web, desde ThingSpeak, que se trata de una base de datos orientada a IoT, desde Adafruit.io, que se trata de un MQTT-Broker que mediante un panel de control permite interactuar con el módulo ESP32, y desde la plataforma IFTTT, que mediante la conexión al servicio de Twitter, permitirá visualizar los datos desde dicho servicio.

Además, como actuador se tiene un array de relés que podrá ser controlado desde el mismo servidor Web, desde Adafruit.io y mediante comandos de voz de Google Assistant gracias a la plataforma IFTTT. Estos dos últimos se llevan a cabo mediante MQTT, que se trata de un protocolo orientado a Internet de las Cosas.

El dispositivo se conectará a un punto de acceso WiFi al que se podrá conectar cualquier dispositivo con WiFi integrado, de forma que permita visualizar remotamente los datos de los sensores, así como también controlar el estado del relé.

En la parte práctica se describirán los diferentes sensores y dispositivos utilizados para la ejecución de la aplicación y posteriormente se explicarán las principales funciones de código que se han desarrollado en la plataforma Arduino, junto a la configuración de los servicios web que se han utilizado.

En el apartado de resultados se realizará un análisis de los datos obtenidos en ThingSpeak con Matlab, y se visualizarán diferentes capturas del funcionamiento de las distintas plataformas.

Introducción

Internet de las Cosas es un concepto basado en la conexión de dispositivos electrónicos entre sí o mediante Internet, que lleva años generando expectación ya que se prevé que sea un gran impulsor de la transformación digital en hogares y ciudades, así como también en empresas.

El rápido crecimiento de aplicaciones con dispositivos interconectados en el mercado de consumo, junto a asistentes del hogar como Amazon Echo o Google Home, está suponiendo un auge en la producción de dispositivos orientados a Internet de las Cosas, de modo que cada vez es mayor el conocimiento de la población sobre este término, y mayor el número de dispositivos comerciales que se fabrican para cubrir dicho mercado.

Igualmente, el crecimiento en el sector empresarial es notable, ya que permite junto al Big Data y las tecnologías en la nube, capturar, almacenar y analizar datos de interés para las empresas, de cara a ampliar su competitividad en el mercado o mejorar sus procesos internos.

Existen diversos dispositivos orientados a IoT, pero en el caso específico de este TFG se ha optado por el ESP32.

El ESP32 es un microcontrolador de bajo coste creado por Espressif Systems. Destaca debido a que posee tecnologías WiFi y Bluetooth integradas, dos núcleos de procesamiento y tecnología de bajo consumo, que, en relación con su precio, lo hacen un dispositivo muy recomendable para desarrollar aplicaciones enfocadas a Internet de las Cosas.

El objetivo de la parte teórica de este documento es definir las características principales del dispositivo, así como sus funcionalidades y periféricos. Además, se describirán los principales entornos de desarrollo y lenguajes de programación que permiten programar el dispositivo.

En cuanto a la parte práctica del proyecto, la idea es implementar una sencilla aplicación que use dicho dispositivo. Dado el enfoque del proyecto, se implementará un sistema orientado a Internet de las Cosas, especialmente para el hogar inteligente o *Smart Home.*

En definitiva, este TFG trata de dar a conocer al dispositivo ESP32, y a los distintos servicios web existentes que pueden implementarse junto al dispositivo, de cara a favorecer su interconectividad en el contexto del Internet de las Cosas.

Capítulo 1

Descripción técnica del módulo ESP32

1.1 Antecedentes

ESP32 es el nombre de una serie de microcontroladores diseñados por Espressif Systems, una empresa china situada en Shanghái. Dicha empresa comenzó a sacar al mercado módulos WiFi fabricados por Ai-Thinker, que permitían a otros microcontroladores conectarse a redes inalámbricas y realizar conexiones TCP/IP usando comandos AT.

Por aquel entonces, era el Arduino MKR1000 el dispositivo que estaba revolucionando el mercado con su chip ATSAMW25, que disponía de conectividad WiFi, pero tenía un elevado precio, lo que hizo que la comunidad de desarrolladores se inclinase hacia otras opciones mucho más asequibles como los microcontroladores de Espressif, haciendo de shield o puente WiFi para otros microcontroladores.

El primer dispositivo que salió al mercado fue el ESP-01, en agosto de 2014. Por entonces, la única información del dispositivo estaba en chino por lo que no se demostró su potencial hasta que la comunidad empezó a crear firmwares y manuales traducidos para programar el dispositivo. La comunidad posteriormente liberó SDKs (Software Development Kits) de código abierto basados en el toolchain GCC. Esto permitió la programación con el Arduino Core, que fue la catapulta que llevó a la fama a este tipo de dispositivos. Posteriormente, Espressif generó un SDK con licencia que dio soporte a los usuarios (ESP-IDF).

El SoC (System on Chip) que integraba el dispositivo era el ESP8266, y las distintas evoluciones que tuvo el módulo variaban básicamente en la memoria flash disponible. Así siguieron saliendo módulos con diferentes funcionalidades desde el ESP-02 hasta el ESP-14, y otros como el ESP-201, basado en los anteriores. Pero el módulo más característico de la familia ESP8266 es el NodeMCU, basado en el ESP-12, que incluía un adaptador serie/USB y se alimentaba a través de micro USB. Mediante firmware permitía programar el módulo en lenguajes como C, LUA o Python.

Finalmente, a mediados de 2016, salió una mejora del ESP8266 con 1MB de flash integrada en el propio chip, llamado ESP8285, que fue el último modelo que desarrolló el fabricante antes de la salida al mercado del chip ESP32, en septiembre de 2016.

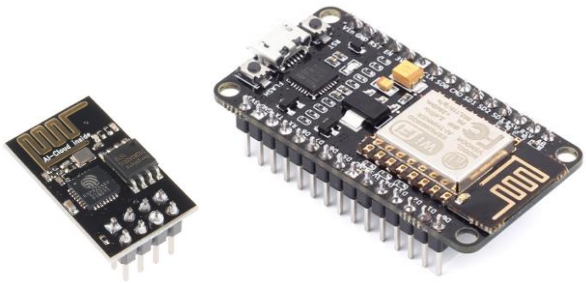

Figura 1. Módulo ESP-01 y NodeMCU. Ambos con el SoC ESP8266.

1.2 Dispositivos ESP32

El ESP32 es un System On Chip igualmente diseñado por Espressif Systems, pero fabricado por TSMC. Al igual que el chip ESP8266 dispone de varios modelos con diferentes características.

La propia empresa define esta serie como una solución para microcontroladores que no dispongan de conectividad, ya que podrían utilizar la familia ESP32 como puente para el acceso a la red o a las soluciones IoT. Además, la serie ESP32 es capaz de ejecutar sus propias aplicaciones de tiempo real, lo que le hace un dispositivo muy interesante.

Es importante diferenciar entre chip, módulo y tarjeta de desarrollo, aunque comúnmente se le pueda llamar módulo ESP32 o chip ESP32 indistintamente. El módulo suele llevar al chip integrado, más un cristal de 40Mhz, memoria flash, y una antena dependiendo del modelo. La tarjeta de desarrollo suele integrar al módulo en un PCB que permite conexión serie/USB, alimentación por USB, botones de *boot* y *reset* y pines soldados a la placa. A continuación, se detallan los chips, módulos y tarjetas de desarrollo principales del ESP32 del fabricante. [3]

En primer lugar, existen los siguientes chips en la familia ESP32:

Ordering code	Core	Embedded flash	Connection	Package
ESP32-D0WDQ6	Dual core	No embedded flash	Wi-Fi b/g/n + BT/BLE Dual Mode	QFN 6*6
ESP32-D0WD	Dual core	No embedded flash	Wi-Fi b/g/n + BT/BLE Dual Mode	QFN 5*5
ESP32-D2WD	Dual core	16-Mbit embedded flash (40 MHz)	Wi-Fi b/g/n + BT/BLE Dual Mode	QFN 5*5
ESP32-S0WD	Single core	No embedded flash	Wi-Fi b/g/n + BT/BLE Dual Mode	QFN 5*5

Tabla 1. Chips de la familia ESP32.

La nomenclatura de los chips anteriores define sus características y sigue el siguiente esquema:

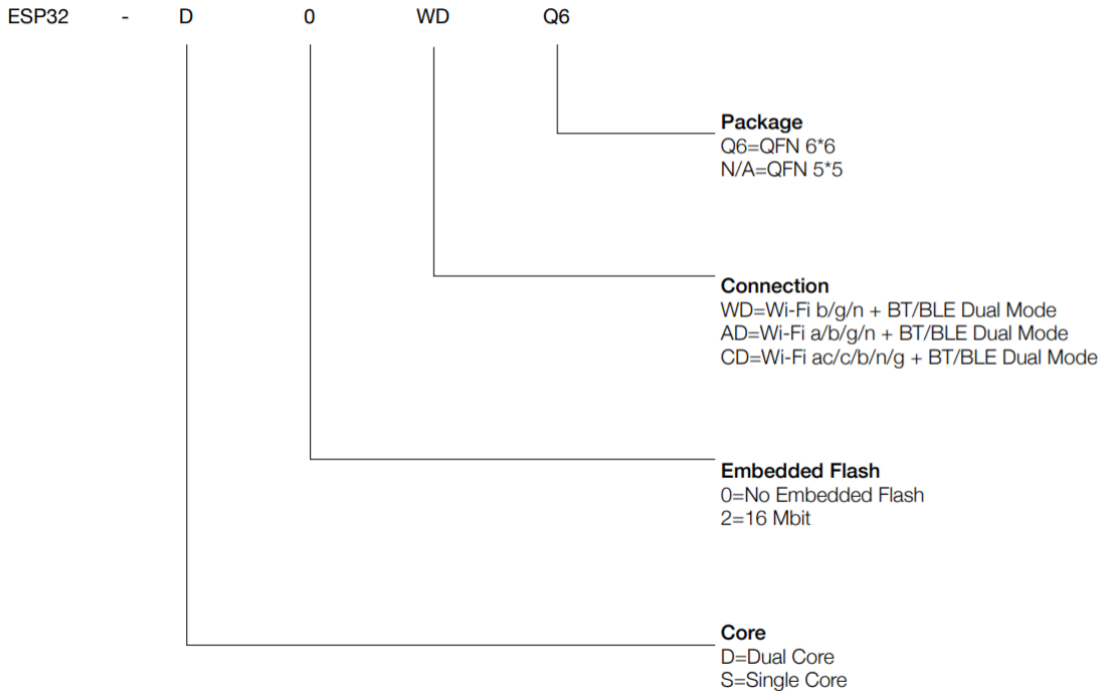

Figura 2. Nomenclatura de los chips ESP32.

En donde se pueden diferenciar entre chips con uno o dos núcleos, si montan memoria flash embebida, el tipo de conexión WiFi que soportan y el tamaño del encapsulado.

Una vez vistos los diferentes tipos de chip se describen a continuación los principales módulos que han sido desarrollados por Espressif Systems, que se listan en la siguiente tabla.

–	Key Components				Dimensions [mm]		
Module	**Chip**	**Flash**	**RAM**	**Ant.**	**L**	**W**	**D**
ESP32-WROOM-32	ESP32-D0WDQ6	4MB	–	MIFA	25.5	18	3.1
ESP32-WROOM-32D	ESP32-D0WD	4MB	–	MIFA	25.5	18	3.1
ESP32-WROOM-32U	ESP32-D0WD	4MB	–	U.FL	19.2	18	3.2
ESP32-SOLO-1	ESP32-S0WD	4MB	–	MIFA	25.5	18	3.1
ESP32-WROVER	ESP32-D0WDQ6	4MB	4MB	MIFA	31.4	18	3.2
ESP32-WROVER-I	ESP32-D0WDQ6	4MB	4MB	U.FL	31.4	18	3.5

Tabla 2. Principales módulos ESP32.

1.2.1 ESP-WROOM-32

Se trata del primer módulo de la familia ESP32 que salió al mercado. Por defecto tiene 4MB de memoria flash, aunque es ampliable a 8 ó 16MB. Además, lleva el chip ESP32-DOWDQ6 y antena MIFA.

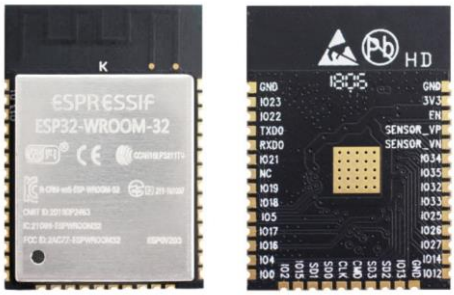

Figura 3. Módulo ESP-WROOM-32.

1.2.2 ESP-WROOM-32D

En general es prácticamente igual al modelo anterior. La única diferencia es que utiliza el chip D0WD, con encapsulado interior del chip de 5x5mm.

Figura 4. Módulo ESP-WROOM-32D.

1.2.3 ESP-WROOM-32U

Como en el caso anterior, monta el chip ESP32-D0WD. Lleva el encapsulado QFN de 5x5mm, pero en este caso monta un conector de antena IPEX/U.FL, lo que hace reducir sus dimensiones respecto a los módulos anteriores.

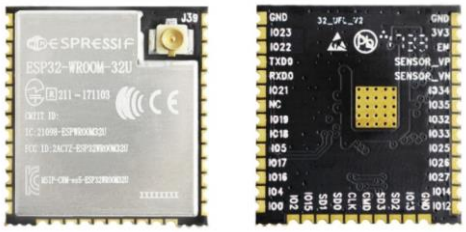

Figura 5. Módulo ESP-WROOM-32U.

1.2.4 ESP32-SOLO-1

Se trata de una versión simplificada del módulo ESP-WROOM-32D, ya que cuenta con un solo núcleo con una frecuencia de hasta 160MHz, inferior a los 240MHz del doble procesador.

Figura 6. Módulo ESP32-SOLO-1.

1.2.5 ESP-WROVER y ESP-WROVER-I

Es el módulo más avanzado. Contiene una SPI PSRAM de 4MB además de la memoria flash externa de 4 MB. Tiene dos versiones, una con antena MIFA y otra con conector IPEX/U.FL. Por sus componentes adicionales es el de mayor tamaño.

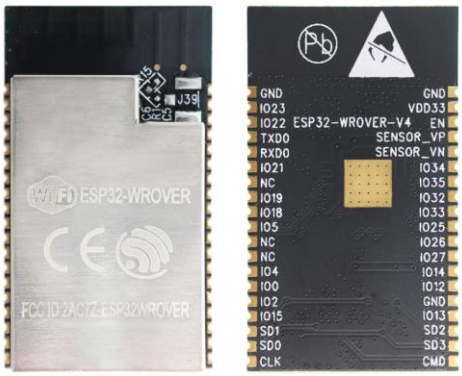

Figura 7. Módulo ESP-WROVER.

Para finalizar este apartado, se van a describir a continuación las tarjetas de desarrollo, que consisten en uno de los módulos ya citados junto a componentes adicionales como pines adaptados, conexión USB, pulsadores y componentes para la interfaz serie/USB. Existen los siguientes modelos, actualizados a su versión más reciente a fecha de la realización del documento:

1.2.6 ESP32-PICO-KIT v4.1

Es la tarjeta de desarrollo más pequeña. Incluye antena, LDO (Low Dropout Regulator), soporte USB-UART y dos botones para reset y modo boot. Monta un módulo especial dedicado llamado ESP-PICO-D4 con 4MB flash. Las dimensiones del dispositivo son 20 x 52mm. Debido a su pequeño tamaño suele utilizarse en aplicaciones portátiles introduciendo una pila en su parte inferior para tener alimentación portátil.

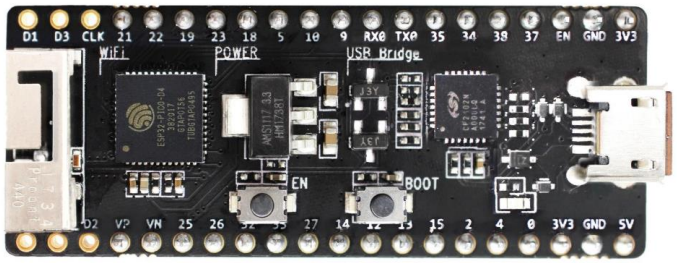

Figura 8. ESP32-PICO-KIT v4.1.

1.2.7 ESP-WROVER-KIT v3

Es la tarjeta de desarrollo más potente ya que cuenta con el módulo ESP-WROVER-32. Incluye doble puerto USB, interfaz JTAG, dos pulsadores, espacio para insertar una memoria microSD, espacio para insertar una pantalla LCD de 3.2 pulgadas y una interfaz con pines para conexión de dispositivos de video o cámara. Es la más completa y versátil que ha comercializado el fabricante.

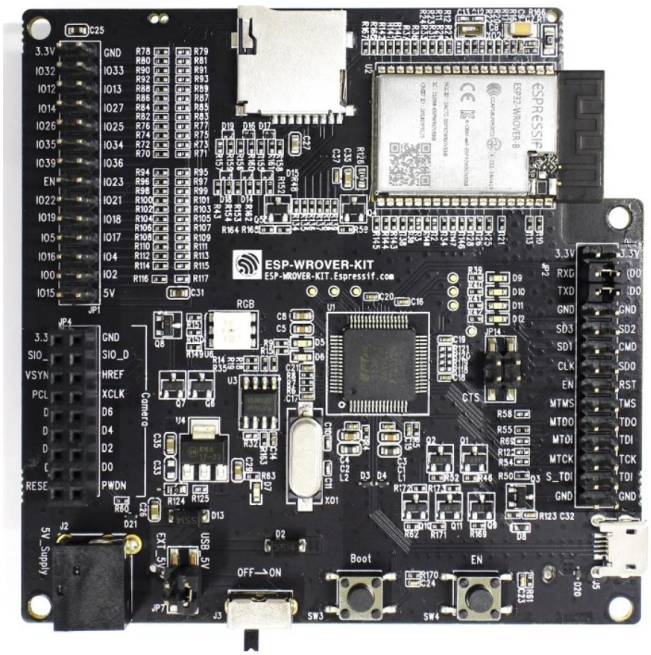

Figura 9. ESP32-WROVER-KIT v3.

1.2.8 ESP32-LyraT

Esta tarjeta de desarrollo monta también el chip ESP-WROVER-32. Especialmente dedicada para desarrollo de aplicaciones de audio, ya que posee reconocimiento por voz, botones para reproducción de sonido y salidas de audio Jack 3.5mm. Además, se puede introducir una microSD para almacenamiento de audio. La tarjeta ESP32-LyraT ha sido especialmente diseñada para altavoces inteligentes y aplicaciones Smart Home.

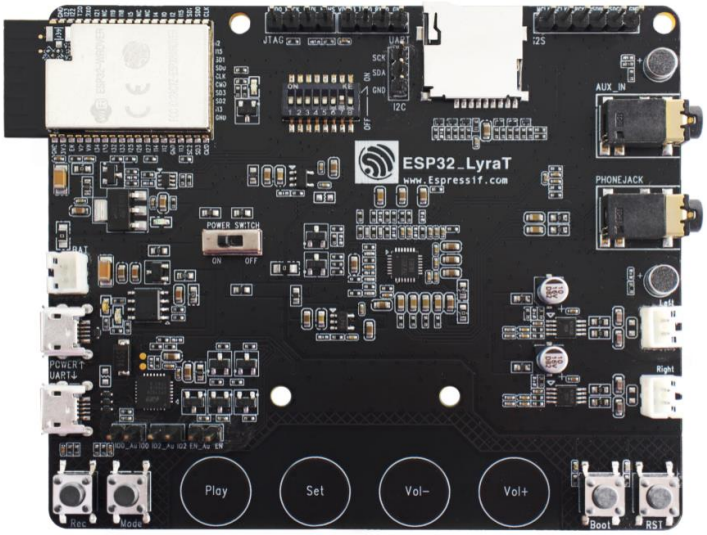

Figura 10. ESP32-LyraT.

1.2.9 ESP32 DevKitC v4

Por último, se muestra la tarjeta de desarrollo con la que se desarrolla la aplicación. Puede incorporar tanto el módulo ESP-WROOM-32 como el ESP-WROOM-32D. Incorpora pines adaptados e incluye interfaz USB, que también sirve de alimentación a la placa. También incluye el chip CP2102N que soporta velocidades de transmisión de hasta 3Mbit/s en el modo USB/UART. Por último, incorpora dos pulsadores con funciones de reset y boot. En el siguiente capítulo se ampliará la información sobre esta tarjeta, ya que se utilizará para el desarrollo de la parte práctica.

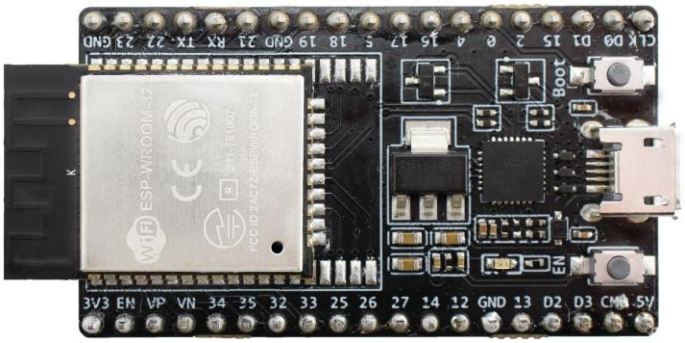

Figura 11. ESP32 DevKitC v4.

Unos meses después del lanzamiento de los chips ESP32 (finales de 2016), otros fabricantes utilizaron los microcontroladores de Espressif para adaptarlos a sus propias necesidades, creando distintas tarjetas de desarrollo, e incluyendo baterías o pantallas integradas en dichas tarjetas de desarrollo.

1.3 Especificaciones detalladas del SoC ESP32

A continuación, se va a proceder a la descripción interna del SoC ESP32, sus funcionalidades y características. En la siguiente imagen se muestran los principales bloques del dispositivo.

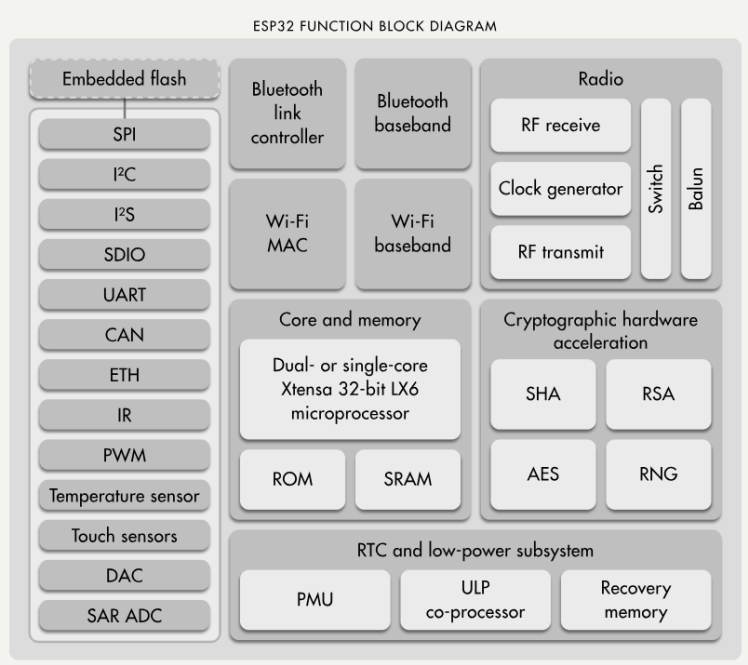

Figura 12. Diagrama de bloques del SoC ESP32. [4]

Procesadores

El ESP32 monta uno o dos microprocesadores Tensilica Xtensa 32-bit LX6, dependiendo del modelo, que consiguen obtener una frecuencia de reloj de hasta 240MHz y un rendimiento de hasta 600 DMIPS.

Además, incorpora un procesador ULP (Ultra Low Power) que es capaz de trabajar cuando la CPU entra en el modo deep-sleep de ahorro de energía. Puede ejecutar conversiones ADC, operaciones computacionales y verificar el estado de los pines con un consumo minúsculo. El procesador ULP y la memoria RTC permanecen encendidas durante el modo Deep-sleep. Es útil para el diseño de aplicaciones donde la CPU necesita despertarse por un evento o interrupción, manteniendo en el resto del tiempo un consumo mínimo.

Memoria

En cuanto a la memoria, el chip dispone de las siguientes memorias internas:

- Memoria ROM de 448 KiB, para funciones de núcleo y boot.
- Memoria SRAM de 520 KiB, para datos e instrucciones.
- Memoria RTC fast SRAM de 8 KiB, para almacenamiento de datos. Puede ser usada por el núcleo principal durante el boot desde el modo deep-sleep.
- Memoria RTC slow SRAM de 8KiB, para que el coprocesador acceda durante el modo deep-sleep.
- Memoria flash embebida, que, dependiendo del modelo, tiene un tamaño de 0 a 4MiB.

Además, se puede acoplar una memoria flash externa de hasta 16MiB, o una SRAM de hasta 8MiB, dependiendo también del modelo.

Conectividad

Como se ha mencionado en la introducción, el SoC ESP32 posee soporte para tecnologías WiFi y Bluetooth. Respecto al WiFi, soporta las tecnologías estándar Wifi 802.11 b/g/n. (802.11n a 2.4 GHz hasta 150 Mbit/s), mientras que la versión de Bluetooth soportada es la v4.2 BR/EDR y dispone además de BLE (Bluetooth Low Energy).

Timers

El SoC dispone de cuatro temporizadores de propósito general embebidos de 64 bits, basados en prescalers de 16 bits (de 2 a 65536). Los Timers son configurables en cuenta ascendente o descendente y pueden ser generados como interrupción por nivel o por flanco.

Watchdog

Dispone de tres Watchdog Timers, uno en cada núcleo, llamados Main Watchdog Timer (MWDT) y otro en el RTC, llamado RTC Watchdog Timer (RWDT). Están destinados a recuperar al dispositivo ante un fallo imprevisto, pudiendo desencadenar distintas acciones como el reinicio de la CPU, del núcleo o del sistema. Se puede programar un temporizador diferente para cada tipo de reinicio. Solo el RWDT puede ejecutar el reinicio del sistema y del propio RTC (Real Time Clock).

Relojes del sistema

El dispositivo cuenta con diferentes tipos de relojes del sistema. En primer lugar, el reloj de la CPU, cuya fuente de reloj es por defecto un reloj de cristal externo de 40MHz que se conecta a un PLL para generar una frecuencia de 160Mhz. Además, cuenta con un oscilador interno de 8MHz. El programa puede elegir que reloj utilizar en el sistema, dependiendo de la aplicación.

En segundo lugar, un reloj de tiempo real (RTC), cuya fuente puede ser un reloj de un oscilador RC interno de 150KHz, de un oscilador interno de 8MHz, o del cristal externo al dispositivo.

Radiofrecuencia:

EL módulo ESP32 consta de los siguientes bloques de radiofrecuencia:

Un receptor de 2.4 GHz que demodula la señal de radiofrecuencia y la convierte al dominio digital por medio de dos ADCs de alta resolución. Para adaptarse a las diferentes condiciones de la señal, el chip monta filtros de RF, control automático de ganancia, cancelación de continua y filtros de banda base integrados.

Un transmisor de 2.4Ghz que modula la señal en banda base a señal de RF a 2.4Ghz, que posteriormente sale por la antena gracias a un amplificador de potencia basado en CMOS. Incluye calibraciones integradas que permiten cancelar las imperfecciones de ruido.

Un generador de reloj que produce una señal en cuadratura de 2.4Ghz para transmisión y recepción de WiFi con todos los componentes integrados y calibración automática.

Periféricos y sensores

A continuación, se van a tratar los diferentes periféricos y sensores del dispositivo.

En primer lugar, los pines de propósito general. El módulo ESP32 tiene 34 pines GPIO a los que se le pueden asignar distintas funciones mediante la programación de los registros apropiados. Hay pines de distintos tipos: digitales, digitales y analógicos, y con función touch. La mayoría de los pines GPIO se pueden configurar como pull-up, pull-down, o alta impedancia.

El ESP32 dispone de dos conversores analógico-digital (ADC) de 12-bit SAR, que pueden soportar entre ambos hasta 18 canales de entrada analógica. Además, es capaz de funcionar cuando el dispositivo entra en modo de ahorro de energía, gracias al coprocesador de bajo consumo. También posee un conversor digital-analógico (DAC) de 8 bits que permite convertir dos señales digitales en analógicas.

El dispositivo cuenta con una serie de sensores internos como un sensor de temperatura, un sensor de efecto Hall, que permite detectar si existe campo magnético cerca del dispositivo, o sensores tipo touch existentes en algunos pines que pueden detectar variaciones al acercar o tocar el pin con el dedo o con un objeto.

También se dispone de un controlador SD/SDIO/MMC que soporta traspaso de datos desde dispositivos externos de hasta 80Mhz con diferentes buses: de 1bit, de 4bits y de 8bits.

Para la comunicación da datos serie el dispositivo cuenta con tres interfaces UART que proporcionan comunicación asíncrona de hasta 5Mbps.

Otros protocolos de comunicación como I2C, I2S y SPI son soportados por el dispositivo. Soporta dos interfaces I2C, que pueden servir como maestro o como esclavo a una velocidad de hasta 5Mhz. Por otra parte, soporta comunicaciones I2S por dos interfaces con controladores DMA dedicados. Respecto al SPI, se proporcionan cuatro modos de funcionamiento dependiendo de la polaridad y la fase del reloj del SPI, con una velocidad de hasta 80MHz y hasta cuatro interfaces simultáneas.

Otra de las características del SoC es que posee un controlador de infrarrojos de ocho canales para la transmisión y envío de datos, que soporta varios protocolos de infrarrojos. Para finalizar el apartado de periféricos, el ESP32 también puede generar hasta dieciséis canales PWM (pulse width modulation) y posee una interfaz para la comunicación por bus CAN 2.0.

Seguridad

Para la conexión vía WiFi el dispositivo soporta los estándares de seguridad de 802.11 como WFA, WPA/WPA2 y WAPI. Además, permite la encriptación de la memoria flash del dispositivo y aceleración criptográfica hardware como AES (Advanced Encryption Standard), SHA-2 (Secure Hash Algorithm 2), RSA, ECC (Elliptic Curve Cryptography) y RNG (Random Number Generator).

Modos de operación

El SoC puede estar en distintos modelos de operación en función de la energía utilizada y son los siguientes:

- Activo: todo encendido, el chip puede recibir, transmitir y escuchar.
- Modem-sleep: la CPU está operativa y el reloj es configurable. WiFi y Bluetooth están desactivados.
- Light-sleep: la CPU está pausada. La memoria y los periféricos RTC están activos junto al coprocesador ULP de bajo consumo. La CPU se activará ante un evento que requiera su uso.
- Deep-sleep: solo la memoria y los periféricos RTC están encendidos. Los datos de conexión WiFi o Bluetooth se almacenan en la memoria RTC.
- Hibernación: el oscilador interno de 8MHz y el coprocesador ULP están deshabilitados. El RTC Timer o un evento generado por los pines GPIO pueden despertar al chip del modo hibernación.

1.4 Comparativa ESP8266 vs ESP32

Anteriormente se ha visto que el ESP8266 tenía ciertas restricciones al estar destinado inicialmente a ser un adaptador de serie a WiFi por lo que el fabricante se puso a desarrollar un nuevo producto que posteriormente conoceríamos como ESP32, un chip con potencia y versatilidad para el mundo del IOT. Las principales diferencias del ESP32 con su antecesor, el SoC ESP8266, son las siguientes:

- Presencia de doble núcleo hasta 240MHz.
- Disponibilidad de un coprocesador de bajo consumo.
- Integración de tecnología Bluetooth en el dispositivo.
- Disponibilidad de más canales para el conversor analógico digital (ADC).
- Disponibilidad de un conversor digital analógico (DAC).
- Mayor número de puertos para comunicaciones I2C o SPI.
- Presencia de sensores internos como sensores de temperatura y de efecto Hall.
- Posibilidad de encriptación hardware y de la memoria flash.

A continuación, se muestra en la siguiente tabla las principales diferencias entre ambos dispositivos:

Características	ESP8266	ESP32
Procesador	Tensilica LX106 32 bit a 80Mhz (hasta 160MHz)	Tensilica Xtensa LX6 32 bit Dual Core a 160Mhz (hasta 240MHz)
Memoria RAM	80 KB (40 kB disponibles)	520 KB
Memoria Flash	Hasta 4MB	Hasta 16MB
ROM	No	448 KB
Alimentación	3.0v a 3.6v	2.2v a 3.6v
Rango temperaturas	-40ºC a 125ºC	-40ºC a 125ºC
Consumo de corriente	80 mA (promedio) 225 mA máximo	80 mA (promedio) 225 mA máximo
Consumo en deep-sleep	20uA (RTC + memoria RTC)	2.5uA (RTC +memoria RTC)
Coprocesador ULP	No	Si. Consumo inferior a 150uA
WiFi	802.11 b/g/n (hasta +20dBm) WEP, WPA	802.11 b/g/n (hasta +20dBm) WEP, WPA
Soft-AP	Si	Si
Bluetooth	No	v4.2 BR/EDR y BLE
UART	2 puertos	3 puertos
I2C	1 interfaz	2 interfaces
SPI	2 interfaces	4 interfaces
GPIO (utilizables)	32 pines	11 pines
PWM	8 canales	16 canales
ADC	1(10bit)	2 (Hasta 18 canales) (12 bit)
ADC preamplificador	No	Si. Bajo ruido hasta 60dB
DAC	No	2 canales (8bit)
1-wire	Vía software	Vía software
I2S	1 interfaz	2 interfaces
CAN 2.0	No	1 bus
Ethernet	No	10/100Mbps MAC
Sensor temperatura	No	Si
Sensor efecto hall	No	Si
Infrarrojos	No	Si
Timers	3	4(64bits)
Encriptación	No (TLS por software)	Si (AES, SHA, RSA, ECC)
RNG	No	Si
Encriptación flash	No	Si
Secure Boot	No	Si

Tabla 3. Comparativa ESP8266 vs ESP32.[5]

1.5 WiFi

Como se ha especificado anteriormente el dispositivo ESP32 dispone del estándar 802.11 b/g/n con una velocidad máxima de hasta 150Mbit/s a 2.4GHz. Además, dispone de varios modos de seguridad como WPA o WPA2, escáner de puntos de acceso activo y pasivo, y un modo promiscuo para monitorizar el envío de paquetes.

Cuando se trabaja con un dispositivo WiFi, es importante tener una noción básica de cómo funciona el dispositivo. En general, se trata de la comunicación TCP/IP sobre un enlace inalámbrico.

Durante las operaciones como dispositivo WiFi, se producen eventos que tienen que ver con el estado de la conexión y que el ESP32 tiene que tratar con un manejador de eventos y son los siguientes:

SYSTEM_EVENT_SCAN_DONE	Escáner completo de AP cercanos.
SYSTEM_EVENT_STA_START	Establecimiento como estación.
SYSTEM_EVENT_STA_STOP	Finalización como estación.
SYSTEM_EVENT_STA_CONNECTED	Conectado a un AP como estación.
SYSTEM_EVENT_STA_DISCONNECTED	Desconectado del AP como estación.
SYSTEM_EVENT_STA_AUTHMODE_CHANGE	Cambio en el modo de autenticación.
SYSTEM_EVENT_STA_GOT_IP	Asignar una IP al dispositivo desde el AP.
SYSTEM_EVENT_AP_START	Comienzo como punto de acceso.
SYSTEM_EVENT_AP_STOP	Fin como punto de acceso.
SYSTEM_EVENT_AP_STACONNECTED	Una estación se conectó al AP creado.
SYSTEM_EVENT_AP_STADISCONNECTED	Una estación se desconectó del AP creado.
SYSTEM_EVENT_AP_PROBEREQRECVED	Recibo de petición de prueba al AP creado.

Tabla 4. Eventos WiFi en ESP32.

A continuación, se detallan los modos WiFi que permite el ESP32:

- Modo estación (STA): el dispositivo se conecta a un punto de acceso existente, al igual que otros terminales compartiendo dicha red. Cuando se establece la conexión llegan dos eventos al ESP32, SYSTEM_EVENT_STA_CONNECTED indicando que se ha conectado a un punto de acceso y SYSTEM_EVENT_STA_GOT_IP que indica que se ha asignado una IP con el servidor DHCP del punto de acceso.

- Modo punto de acceso (Soft AP): el dispositivo crea su propio punto de acceso y son los terminales los que se conectan a dicha red. Para este caso hay que definir el SSID y la contraseña a utilizar para crear el AP. Llegará el evento SYSTEM_EVENT_AP_START que indicará que el punto de acceso a sido creado. Cuando una estación se conecte al AP se generará el evento SYSTEM_EVENT_AP_STACONNECTED y se le asignará una IP con un servidor DHCP interno. Es importante destacar que el dispositivo realiza conexiones directas con las estaciones que se conecten, por lo que, en el caso de conectar más de una estación, estas no se podrían comunicar entre ellas, sino a través del punto de acceso.

- Modo combinado (AP + STA): el dispositivo actúa como punto de acceso y a su vez está conectado a otra red como estación. Es un conjunto de los dos modos anteriores.

1.6 Bluetooth

El dispositivo también dispone de comunicaciones inalámbricas vía Bluetooth mediante dos estándares, Bluetooth clásico y Bluetooth LE de bajo consumo. Para poder comunicar al dispositivo con otro que también posea tecnología Bluetooth, hay que tener en cuenta la seguridad de la conexión, mediante la generación de claves únicas que solo son compartidas entre ambos dispositivos. A este proceso se le conoce como *pairing*.

Existen dos pasos para implementar el protocolo Bluetooth:

- El perfil de acceso genérico (GAP) es el encargado de determinar qué dispositivos interactúan entre sí, controlando las conexiones y los anuncios. Posee dos formas de implementar la conexión Bluetooth, enviando tramas de tipo broadcast que serán escuchados por otros clientes cercanos, o mediante conexión directa, en la que después de que el servidor envíe broadcast el otro dispositivo establece directamente la conexión.

- El perfil de atributos genérico (GATT) describe cómo los datos son transferidos una vez se establezca la conexión.

Dentro del GATT se pueden diferenciar tres campos diferentes:

- Servicio: contiene información almacenada en características o referencias a otros servicios, como por ejemplo el servicio Current Time, que indica la fecha actual.

- Características: el valor del dato que se quiere enviar por comunicación Bluetooth. Se suelen utilizar características predefinidas.

- Descriptores: provee información adicional sobre una característica y se puede utilizar para activar notificaciones cada vez que el valor de la característica cambie.

Para cada uno de los servicios, características y descriptores se utilizan UUIDs (Universally Unique Identifier) para describir su naturaleza, de modo que cada servicio tenga un identificador único.

Como en el caso de WiFi, el ESP32 dispone de una serie de eventos sobre Bluetooth, que nos permiten controlar el broadcast y la recogida de información, los principales son los siguientes:

- ESP_GAP_BLE_ADV_DATA_SET_COMPLETE_EVT
- ESP_GAP_BLE_ADV_DATA_RAW_SET_COMPLETE_EVT
- ESP_GAP_BLE_ADV_START_COMPLETE_EVT
- ESP_GAP_BLE_SCAN_PARAM_SET_COMPLETE_EVT
- ESP_GAP_BLE_SCAN_RESULT_EVT
- ESP_GAP_BLE_SCAN_RSP_DATA_RAW_SET_COMPLETE_EVT
- ESP_GAP_BLE_SCAN_RSP_DATA_SET_COMPLETE_EVT
- ESP_GAP_BLE_SCAN_START_COMPLETE_EVT

1.7 ESP-MESH

ESP-MESH [6] es un protocolo de red construido sobre el protocolo WiFi y permite a un gran número de dispositivos distribuidos en una gran área física conectarse bajo una sola WLAN o red de área local inalámbrica. Una de las ventajas es que la red se puede construir y mantener de forma autónoma.

La manera normal de operar con WiFi es con una conexión de punto a multipunto donde existe un nodo central o punto de acceso al que se conectan los diferentes dispositivos, y es este punto de acceso es el encargado de dirigir las transmisiones entre los dispositivos conectados a la red. El problema es que todos estos dispositivos tienen que estar lo suficientemente cerca del punto de acceso para garantizar la conexión, y además el número máximo de dispositivos conectados está limitado a la capacidad que soporte al punto de acceso.

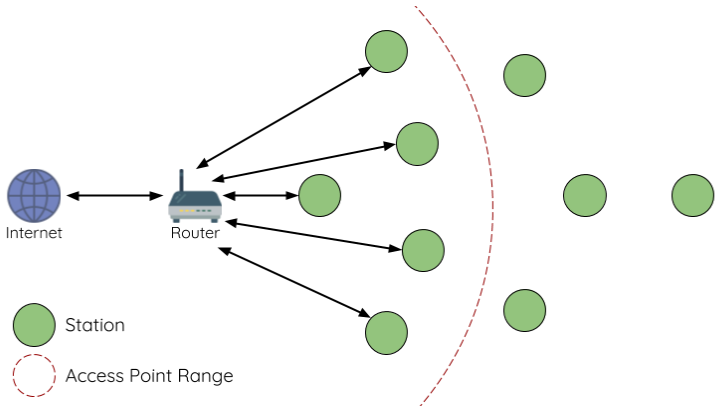

Figura 13. Infraestructura tradicional WiFi.

El protocolo MESH difiere de la infraestructura tradicional ya que los nodos no tienen por qué estar conectados a un nodo central, permitiendo conectarse a nodos vecinos. Los mismos nodos son responsables de transmitir las conexiones de los demás funcionando como puntos de acceso o como estaciones. Lo que permite que el área de cobertura sea mucho mayor y el número de nodos que permite la red ya no está limitado a la capacidad del nodo central.

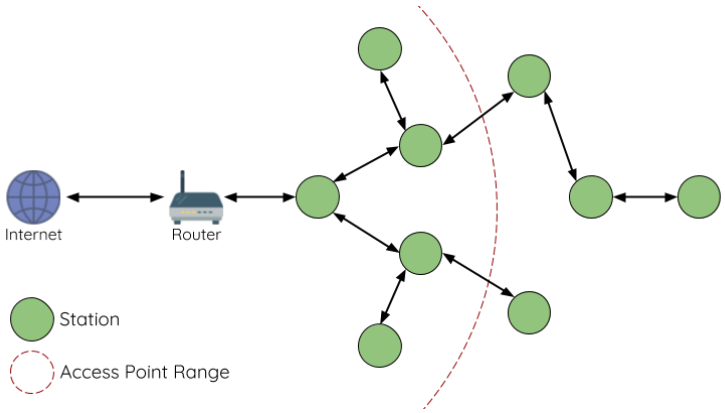

Figura 14. Infraestructura MESH.

Dichas características dan lugar a que se establezca una topología de red tipo árbol y en la que se diferencian las siguientes capas:

- Nodo root: es el nodo de mayor nivel. Posee la única interfaz hacia el exterior de la subred. En el protocolo ESP-MESH solo se puede definir un único nodo root.

- Nodo intermedio: pueden comportarse como estaciones conectadas al nodo raíz o a otros nodos intermedios, o también como puntos de acceso a otros nodos.

- Nodo leaf: se trata de nodos que no pueden ser puntos de acceso para otros nodos, es decir, puede enviar y recibir paquetes como estación, pero no puede procesar paquetes hacia otros nodos. Normalmente se designan este tipo de nodos para limitar el número de capas o niveles de la topología de tipo árbol.

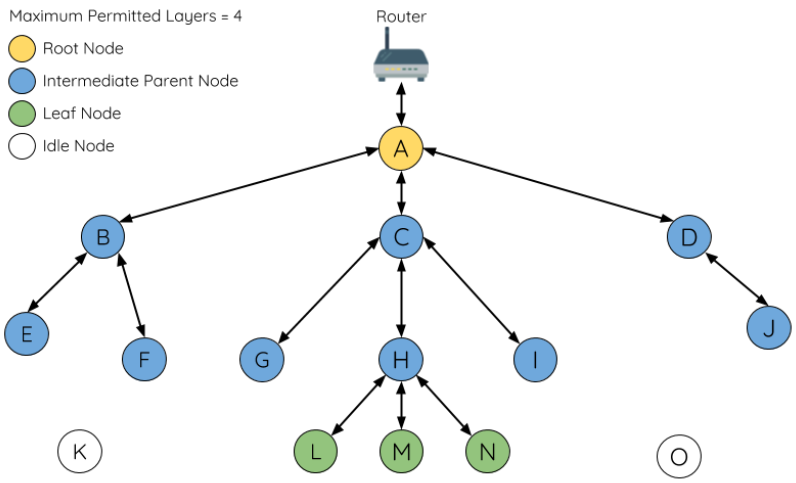

Figura 14. Niveles de la topología MESH.

Para detectar la presencia de nodos cercanos se utilizan tramas beacon. El ESP32 permite configurar en la red MESH cuál es el nodo root por defecto, o establecerlo dependiendo del nodo que proporcione una mejor señal RSSI, calculada con el envío de dichas tramas beacon.

El funcionamiento para la interconexión de nodos se basa en la topología árbol, dando prioridad a la conexión al nodo root, y en caso de no ser posible porque la señal sea débil, se dará prioridad a la siguiente capa de nodos intermedios. En caso de poder conectarse a varios nodos intermedios el nodo elegirá el que posea en el momento de conexión menos nodos conectados.

Respecto al enrutamiento, cada nodo tiene una tabla con las rutas a los nodos de los que es padre. Cuando se quiere enviar un paquete a un nodo que pertenece a su subred, se envía el mensaje al nivel inferior, y el nodo de la capa inferior procesa el mensaje. Cuando se quieran comunicar dos estaciones del mismo nivel, no se hace directamente, ya que la tabla de rutas solo tiene los nodos de su subred, por lo que se enviaría el mensaje al nodo padre, que es el que redirigiría el tráfico hacia al nodo destino.

Como se ha comentado al principio sobre la autonomía del dispositivo, el protocolo también es capaz de actuar ante fallos, como por ejemplo la pérdida de uno de los nodos intermedios. En este caso los nodos hijo de ese nodo intentarían reconectarse a otros nodos cercanos para no quedarse desconectados de la red MESH mediante el envío de tramas beacon a los nodos cercanos. Igual sucedería en el caso de que fallara el nodo root.

1.8 Entornos de desarrollo y lenguajes de programación

A continuación, se van a describir los principales entornos de desarrollo y lenguajes de programación que permiten programar el ESP32.

ESP-IDF (Espressif IoT Development Framework)

Se trata del entorno de desarrollo producido por Espressif Systems. Es uno de los entornos de desarrollo analizados que posee mayor número de recursos a la hora de desarrollar aplicaciones para el ESP32, ya que posee el soporte del fabricante. Está basado en lenguaje C/C++. Tiene una gran cantidad de funciones definidas para facilitar la programación del dispositivo y es posible acceder a extensa documentación en su página web. Además, se basa en el código nativo de FreeRTOS para el desarrollo de sus funciones. Otros entornos de desarrollo que utilizan C/C++ suelen trabajar directamente sobre las funciones creadas por Espressif Systems, como se verá posteriormente.

Los proyectos basados en ESP-IDF pueden desarrollarse desde un terminal con la instalación del toolchain necesario, pero no es muy recomendable debido a la dificultad de manejo del terminal y a la lentitud de la compilación y el posterior flash del código en el dispositivo. La parte positiva del uso del terminal es que se puede acceder con el comando make menuconfig a un menú donde se pueden configurar distintos aspectos del dispositivo como la tabla de particiones y funciones de seguridad como encriptación entre otras funcionalidades.

Aun así, se recomienda instalar un software de terceros, llamado Eclipse. Es un entorno de código abierto para hospedar herramientas de desarrollo. Configurándolo correctamente (hay un tutorial en su página web), permite desarrollar, compilar y flashear el código desde la plataforma, facilitando el desarrollo.

Además del entorno ESP-IDF, Espressif Systems ha desarrollado otros entornos de desarrollo como ESP-ADF (Audio Development Framework) especializado para aplicaciones de audio, y ESP-MDF (Mesh Development Framework) para aplicaciones que hagan uso del protocolo MESH. [10]

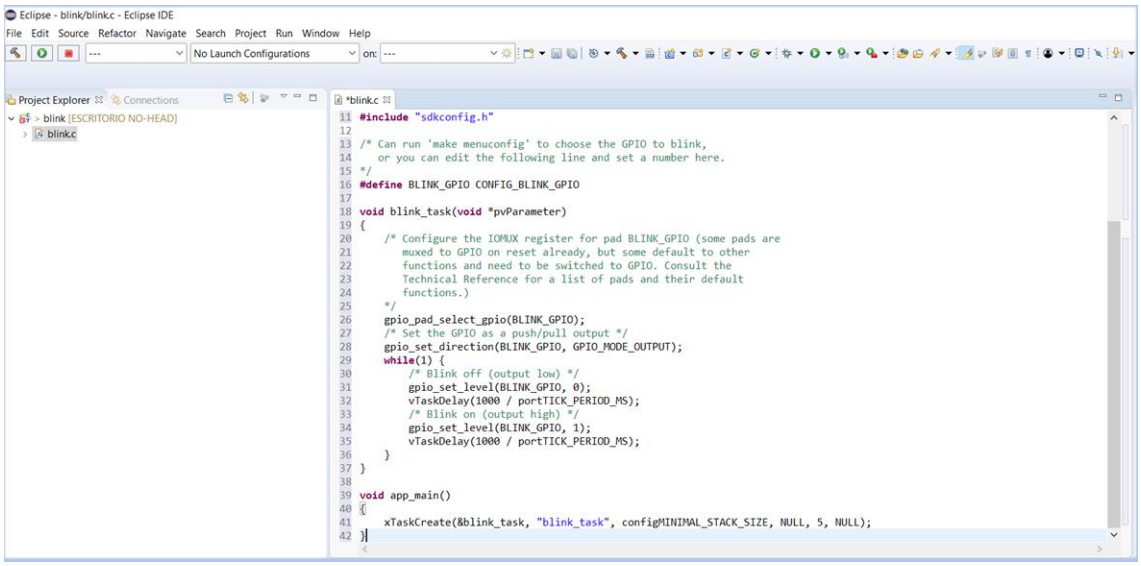

Figura 15. Entorno ESP-IDF con Eclipse.

Arduino IDE (Arduino Integrated Development Enviroment)

Una de las ventajas de Arduino es su facilidad de uso tanto en su interfaz como en la programación. Junto a ESP-IDF son los dos entornos más utilizados para el desarrollo de aplicaciones con el ESP32.

Además, tiene una amplia comunidad que está continuamente mejorando la implementación del módulo ESP32 en el entorno Arduino, facilitando la creación de funciones de alto nivel para adquirir una mayor abstracción del código. El código se desarrolla en C/C++.

Como se ha comentado anteriormente, las funciones que se definen en ESP-IDF son totalmente compatibles con el entorno Arduino, ya que ha sido el mismo fabricante Espressif Systems el que ha desarrollado Arduino Core para el ESP32, por lo que es una gran ventaja de cara a disponer de todas las funcionalidades que posee el dispositivo.

Una vez dentro del entorno de desarrollo se permite configurar aspectos como el puerto por el que flashear el dispositivo, su frecuencia, el tipo de partición y la velocidad del monitor serie entre otras opciones. Existen dos botones en el entorno para compilar y flashear el código desarrollado.

Figura 16. Entorno Arduino.

Otra de las ventajas es que dispone de un repositorio llamado gestor de librerías con todas las librerías existentes donde se puede descargar e instalar el código asociado a dicha librería fácilmente y utilizarse posteriormente.

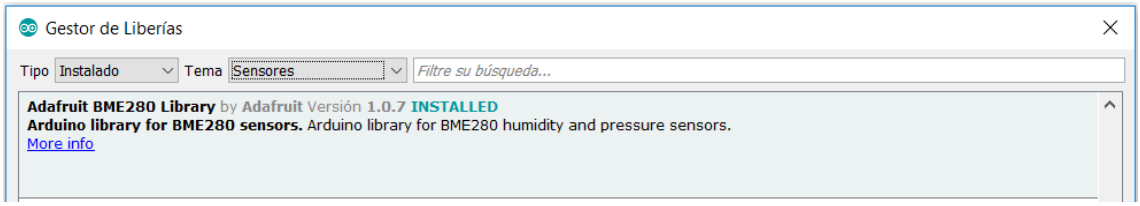

Figura 17. Gestor de librerías Arduino.

Mongoose IDE (Mongoose Integrated Development Enviroment)

Se trata de un IDE que permite trabajar con el sistema operativo Mongoose OS. Es un sistema operativo de código abierto para IoT, con librerías especializadas para trabajar con soluciones cloud como AWS IoT y Google Cloud IoT. Soporta como lenguajes de programación Javascript y una utilidad para portar código en C a Javascript. Se accede al IDE a través del navegador web.[8]

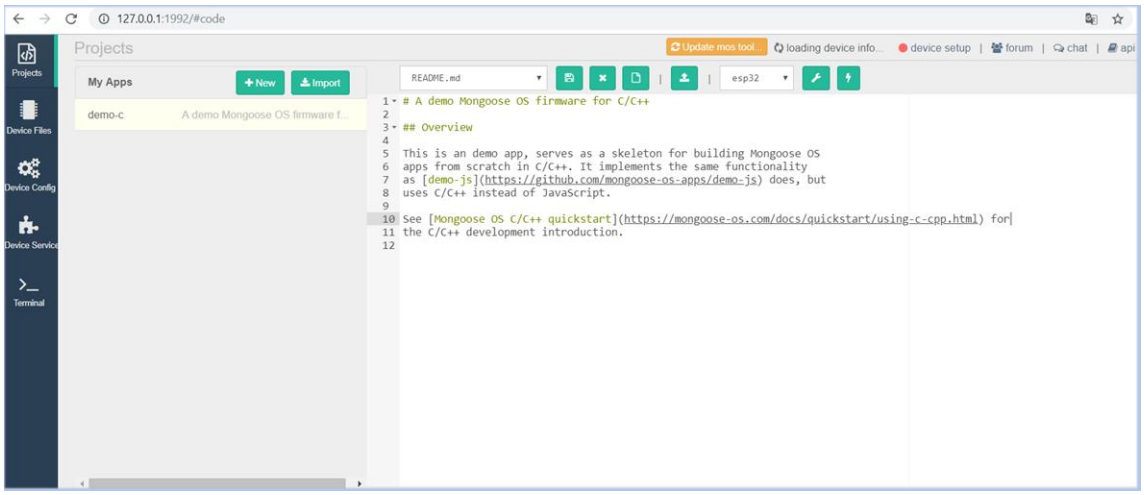

Figura 18. Entorno Mongoose IDE.

Javascript Engines

Se trata de una serie de entornos que corren motores JavaScript embebidos. Los más conocidos son Duktape, Espruino y JerryScript [9].

Duktape es una implementación de Javascript desarrollada exclusivamente en C, en la que se pueden llamar a funciones nativas en C para ejecutar parte del código de Javascript. Al igual que Mongoose, el acceso al IDE es a través de una página web. Espruino permite además instalar el entorno como una extensión del navegador en Google Chrome, y su código se puede representarse como bloques tipo Scratch. JerryScript no tiene un entorno de desarrollo propio y se suele utilizar con el entorno Eclipse.

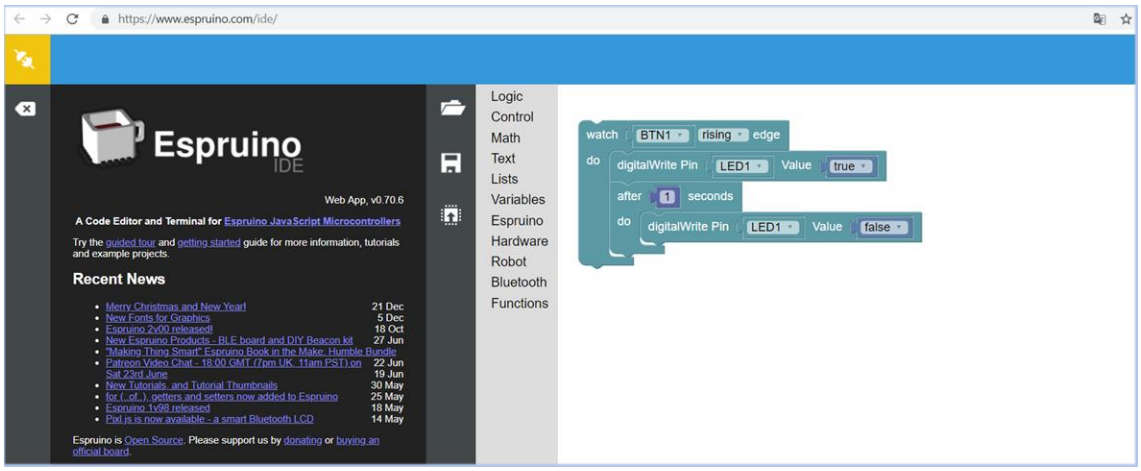

Figura 19. Entorno Espruino.

MicroPython IDE

Es un entorno de desarrollo basado en el lenguaje MicroPython. Este lenguaje no posee todas las librerías estándar que lleva Python debido a que está especialmente diseñado para trabajar con microcontroladores y sistemas embebidos. La ventaja de usar MicroPython es la facilidad del lenguaje. Se pueden utilizar distintos IDE como uPyCraft PyCharm o EsPy.[11]

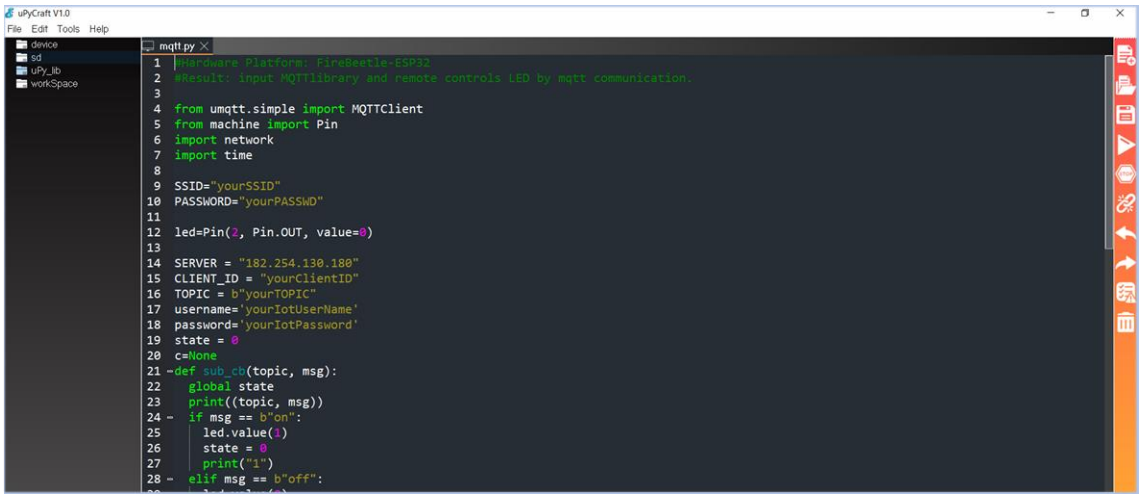

Figura 20. Entorno uPyCraft para MicroPython.

LUA-NodeMCU

Basado en LUA-RTOS, un sistema operativo de tiempo real basado en el lenguaje LUA, que ha sido diseñado para sistemas embebidos con requerimientos mínimos de flash y RAM. Puede ser programado de dos maneras, usando el lenguaje LUA directamente, o usando programación en bloques, que posteriormente serán traducidos a LUA. Puede programarse desde entornos como WhiteCat IDE que se accede a través del navegador web y permite la programación en bloques o en código, o desde el entorno Eclipse.[12]

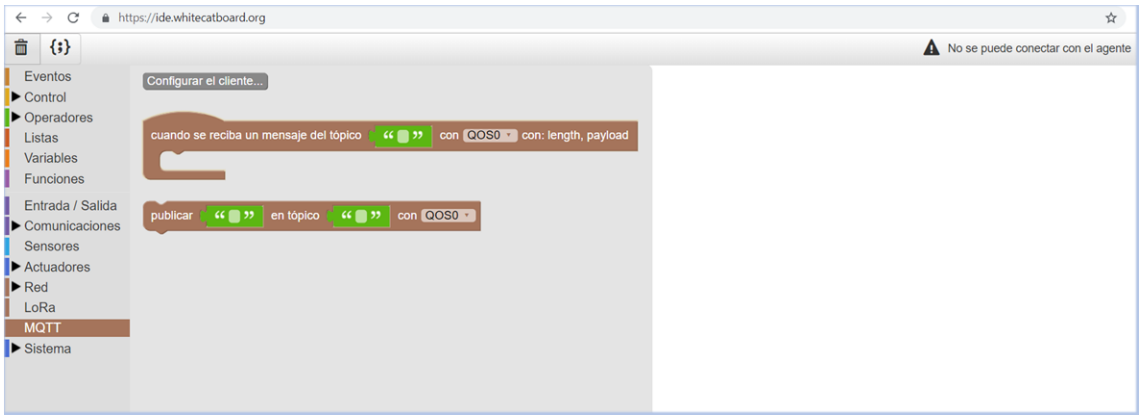

Figura 21. Entorno WhiteCat IDE.

Zerynth

Otro entorno de desarrollo a tener en cuenta es Zerynth. Se trata de un entorno de desarrollo basado en Python y un lenguaje híbrido entre C y Python. Proporciona un IDE en el que se desarrollan las aplicaciones y posee una APP para dispositivos móviles que permite controlar dispositivos IoT remotamente.

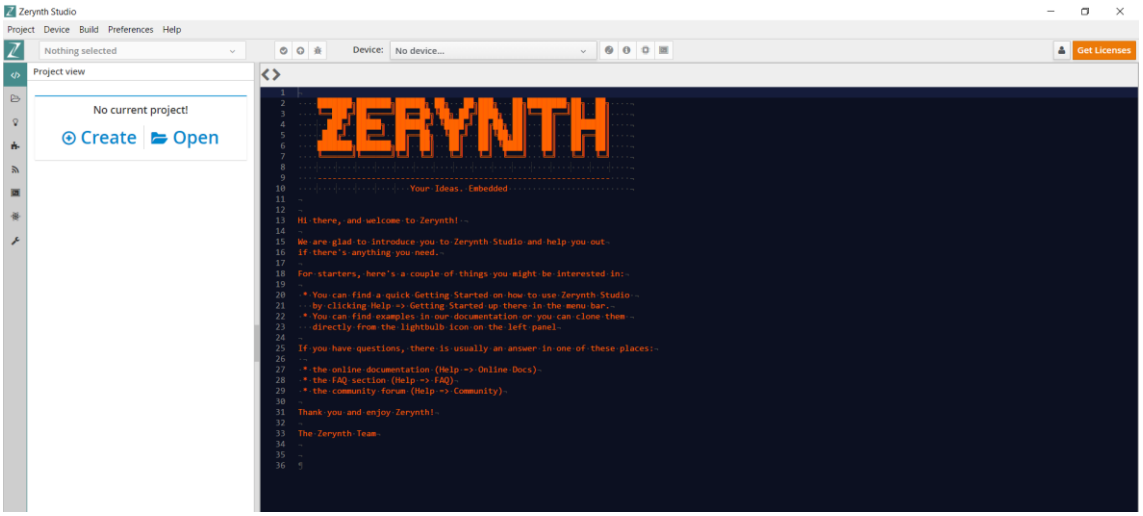

Figura 22. Entorno Zerynth.

De los entornos expuestos se han probado los tres primeros, ESP-IDF, Arduino IDE y Mongoose, y finalmente para el desarrollo de la aplicación se ha decidido utilizar el entorno Arduino por la gran comunidad que lo soporta, por el gran número de ejemplos desarrollados, por su facilidad de uso tanto en la interfaz como en la programación, y porque además permite usar todas las funcionalidades del entorno ESP-IDF, debido a que Espressif liberó el Arduino Core para el ESP32.

Capítulo 2

Desarrollo de la aplicación

2.1 Descripción general de la aplicación

La parte práctica del TFG está dirigida al desarrollo de una aplicación con la tarjeta de desarrollo ESP32 DevKitC de Espressif Systems. Dicha aplicación está orientada al campo “Smart Home” de Internet of Things.

Se trata de desarrollar una aplicación que tome datos de un sensor para monitorizar la temperatura y la humedad del hogar. Se empleará el protocolo I2C para la comunicación entre el módulo ESP32 y el sensor. Otro sensor nos indicará si una persona está presente en el lugar mediante un sensor de presencia. Como actuadores se dispone de un array de relés que irá conectado a bombillas LED de bajo consumo. Estas bombillas podrían simular la iluminación de una sala. También se podría conectar cualquier dispositivo que funcione con corriente alterna.

Los datos podrán verse a través de una pantalla OLED de 0.96 pulgadas que se comunica con el módulo a través también del protocolo I2C. Además, como parte del desarrollo de la aplicación sobre Internet of Things, mediante conexión WiFi se desarrollarán las siguientes aplicaciones:

- Servidor Web:
  Con la ayuda de un navegador web e introduciendo la IP del servidor web en la url, se podrán visualizar los datos recogidos por los sensores e interactuar con las luces LED que irán conectadas al relé. Para ello se ha desarrollado un servidor web que atiende las peticiones GET del cliente. La IP del servidor web podrá verse en la pantalla OLED.

- Adafruit.io + IFTTT:
  Utilizando el protocolo MQTT (Message Queue Telemetry Transport) se tendrá comunicación con la plataforma de Adafruit.io que actuará como un MQTT-Broker, y permitirá la visualización de los sensores e interactuar con el relé. Además, con la herramienta externa IFTTT, se podrá controlar el relé mediante comandos de voz con el servicio de Google Assistant y los datos de la temperatura y la humedad se podrán observar en un canal de Twitter.

- ThingSpeak:
  Se trata de una herramienta externa orientada a IoT con la que se podrán almacenar datos y ver gráficas de la temperatura y la humedad, y analizarlas posteriormente con Matlab, ya que Matlab posee una API para interactuar con los datos almacenados en ThingSpeak.

2.2 Descripción del material utilizado

2.2.1 Sensor Ambiental BME280

En primer lugar, para la toma de datos de la temperatura y la humedad se ha utilizado el sensor BME280. Se trata de un sensor que permite medir temperatura, humedad y presión atmosférica.

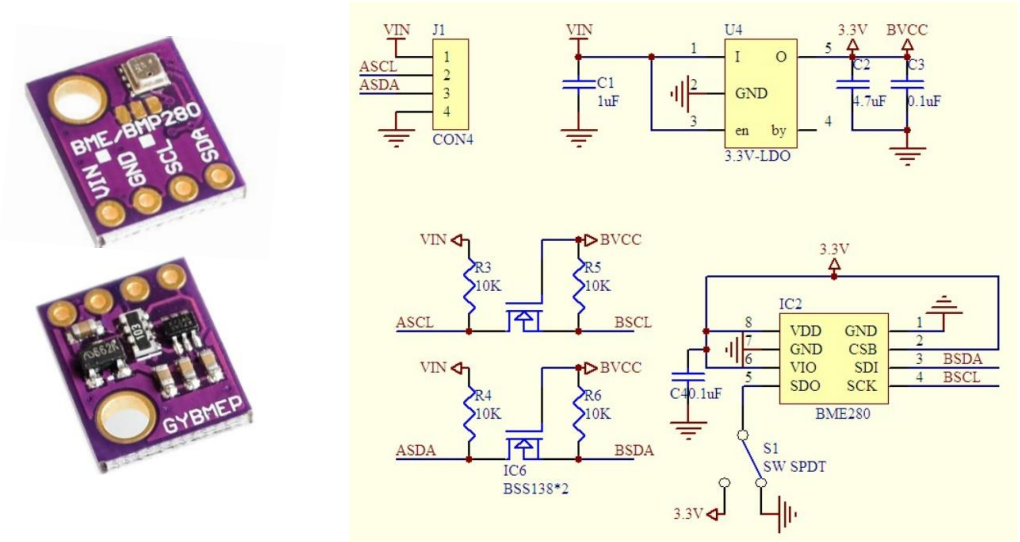

Figura 23. Sensor BME280.

Destaca por su gran precisión en la toma de medidas, como se puede observar en la siguiente tabla de características:

Voltaje de suministro	1.8 - 5V DC
Interfaz I2C	Hasta 3.4MHz
Temperatura (Rango)	-40 a +85ºC
Temperatura (Resolución)	0.01ºC
Temperatura (Exactitud)	+ - 1ºC
Humedad (Rango)	0-100%
Humedad (Resolución)	0.008%
Humedad(Exactitud)	+ - 3%
Presión (Rango)	300-1100 hPa
Presión (Resolución)	0.18 Pa
Presión (Exactitud)	+ - 1Pa

Tabla 5. Características del sensor BME280.

Dispone de cuatro pines (Vin, GND, SCL y SDA) y puede implementar protocolos como SPI e I2C

Con este sensor se obtendrá la temperatura y la humedad ambiente, mediante el protocolo I2C (Inter Integrated Circuit). La dirección de envío del byte START del protocolo I2C es 0x76. [13]

2.2.2 Sensor de presencia HC-SR501

El otro sensor que se va a utilizar para el desarrollo de la aplicación es el sensor de presencia HC-SR501. Con él se podrá observar si se encuentra alguien en la sala que se decida colocar el sensor. Nos puede servir para colocar un Led en la entrada del hogar para que cuando entre alguien detecte presencia y active el Led. [15]

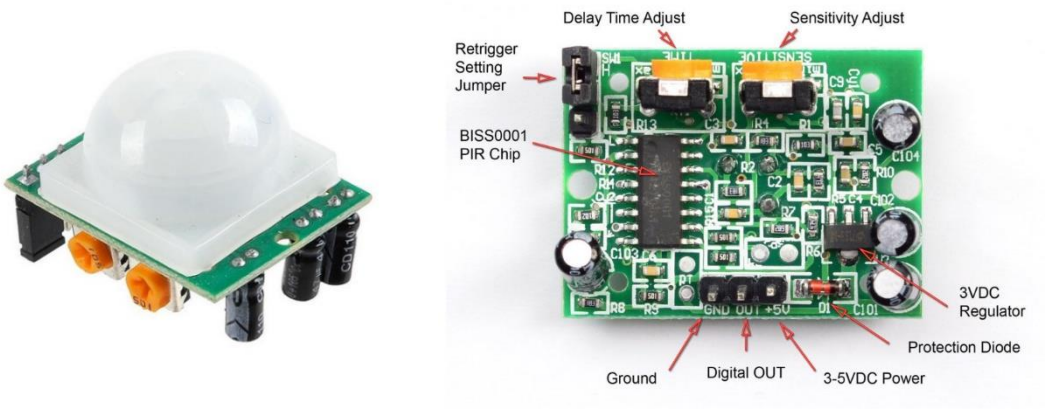

Figura 24. Sensor HC-SR501. [14]

Se trata de un sensor PIR (Passive Infra Red), por lo que es capaz de detectar radiación infrarroja. Para ello utiliza un sensor piro eléctrico que es capaz de detectar la radiación. Además, posee una lente de Fresnel que le permite recibir radiación de múltiples direcciones.

Dispone de tres pines (VCC, GND y OUTPUT). La alimentación recomendada está entre 4 y 12V. El pin de salida tiene un voltaje de 3v3 ya que el sensor dispone de un regulador de voltaje interno a 3v3. En cuanto al modo de funcionamiento, cuando el sensor detecte movimiento, pondrá la salida a nivel alto.

El sensor posee una serie de parámetros que son configurables en los dos potenciómetros del sensor:

- Ajuste de sensibilidad: con el ajuste de sensibilidad se puede modificar el rango de detección máximo que se desee medir. Admite un rango de detección máximo ajustable de 3m a 7m de distancia.

- Ajuste de retardo: con el ajuste de retardo se puede modificar el valor que permanecerá la salida a nivel alto tras la detección de presencia. Este tiempo se puede ajustar entre 0.3 segundos a 5 minutos.

- Modo de disparo:
    - Modo H: Salida a nivel alto y tras un periodo de tiempo (ajuste de retardo) se pone a nivel bajo. Si después la persona sigue en la sala la salida volverá a ponerse a nivel alto.
    - Modo L: Salida a nivel alto hasta que la persona este fuera del alcance del sensor.

2.2.3 Pantalla OLED

Para la presentación de los datos obtenidos por el sensor BME280 de temperatura y humedad se ha utilizado una pantalla OLED de 0.96 pulgadas. El controlador de la pantalla es el SSD1306.

Se trata de una pantalla monocroma de 128x64 píxeles. Su rango de alimentación oscila entre 3 y 5V. Para la comunicación entre el módulo ESP32 y la pantalla se ha optado por el protocolo I2C. Posee cuatro pines: VCC, GND y los pines para la comunicación I2C que son SDA (señal de datos) y SCL (señal de reloj). La dirección de envío del byte START es 0x3C.[16]

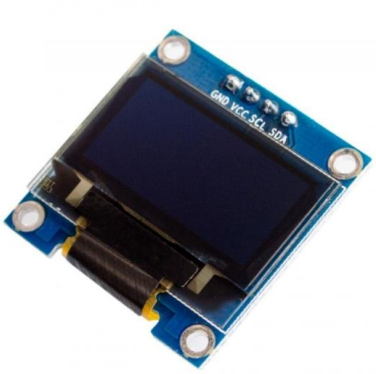

Figura 25. Pantalla OLED de 0.96".

2.2.4 Array de relés de cuatro canales

Se trata de un dispositivo capaz de conmutar mediante entrada electrónica la posición del conmutador interno que estará conectado a corriente alterna. La alimentación del relé es de 5v. Las salidas de los pines actuadores irán conectados al relé, de forma que le permita circular corriente alterna cuando el pin está a nivel alto, y no dejará circular corriente cuando el pin de salida esté a nivel bajo.[17]

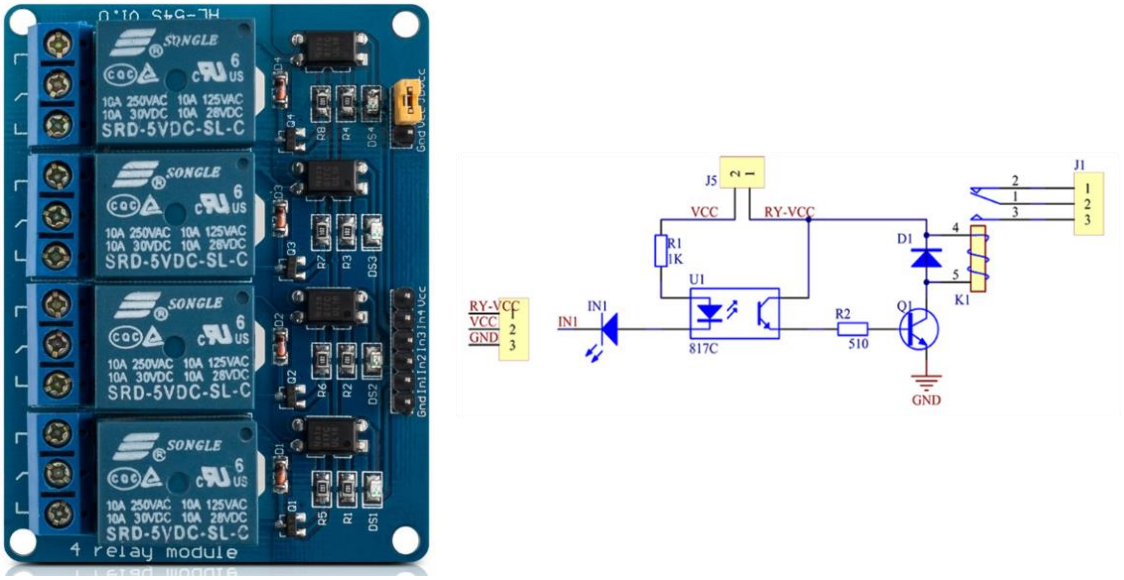

Figura 26. Array de relés de cuatro canales.

2.2.5 Bombillas LED de bajo consumo

Al relé anteriormente citado se le añadirán una serie de bombillas LED de bajo consumo como actuadores, que podrán simular la iluminación de una sala, o un pasillo. La marca utilizada es Lighting EVER.[18]

Figura 27. Bombillas Led de bajo consumo.

Algunas de las características que poseen dichas bombillas son:

- Alta eficiencia. Poseen un consumo de 4W, produciendo una luz blanca de 350 lúmenes.
- Brillo instantáneo. Llegarán al brillo máximo en 0.5 segundos.
- Ángulo de haz de 120º.
- Ecológicas: sin plomo ni mercurio. No emiten luz UV ni IR.
- Voltaje de 220V-240V AC y casquillos GU10.

2.2.6 ESP32 DevKit4C

Se trata del módulo ESP32 original de Espressif Systems. Como ya se ha mencionado en el anterior capítulo posee tecnologías WiFi y Bluetooth que lo hacen muy atractivo para el desarrollo de aplicaciones orientadas a IOT.

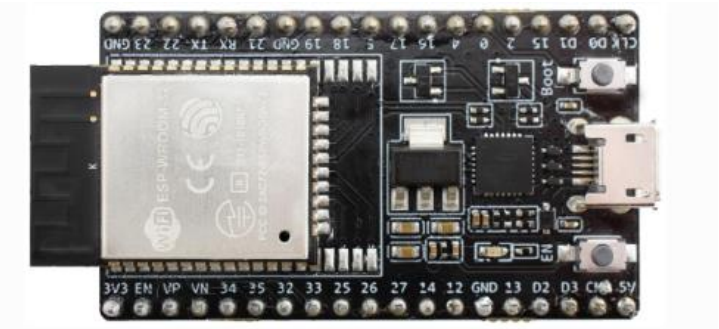

Figura 28. ESP-DevKitC v4.

El resumen del pinout del dispositivo es el siguiente:

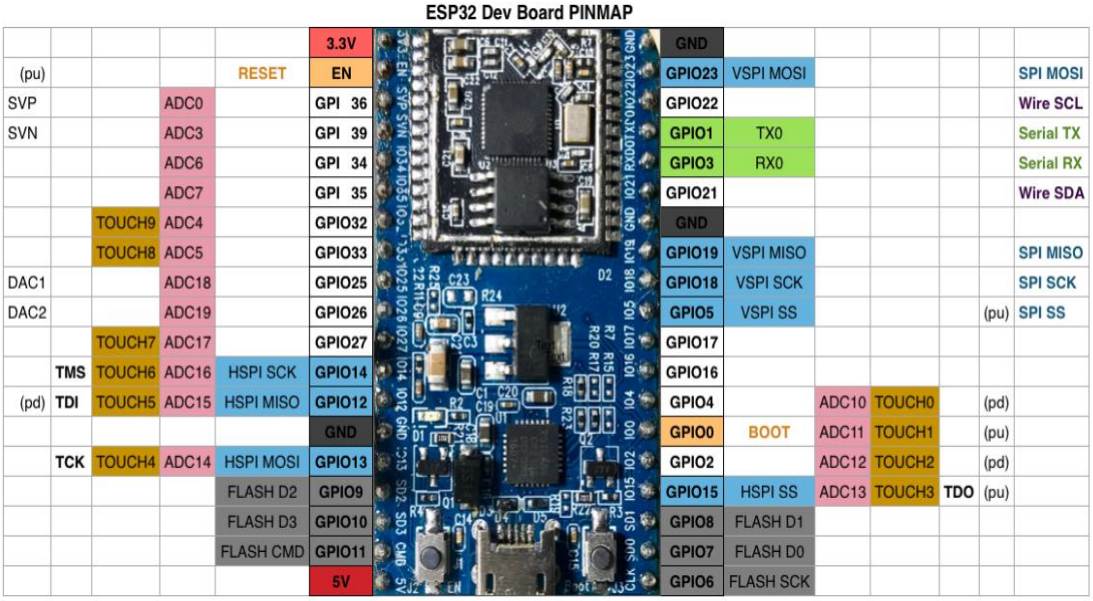

*Figura 29. Pinout ESP32.**[21]***

En esta tarjeta de desarrollo en concreto, hay que tener especial cuidado con algunos pines, como se ve a continuación:

- Pines GPIO 34 y 39: pines solamente de entrada. No pueden generar PWM ni actuar como salidas GPIO.

- Pines GPIO6 hasta GPIO11: conectados a la flash SPI interna del chip ESP-WROOM-32, por lo que no se recomienda su uso ya que puede fallar al cargar el código en el dispositivo.

- Pines para el ADC: no todos los pines están conectados a los ADC internos, solo los siguientes: 0,2,4,12,13,14,15,25,26,27,32,33,34,35,36,39.

- Pines GPIO25 y GPIO26: pines que pueden usar el DAC.

- Pines GPIO21 y GPIO22: pines para la comunicación I2C. Aunque como se verá posteriormente en el apartado 2.3 se puede abrir otro puerto para una segunda comunicación I2C.

- Pines GPIO23 (MOSI), GPIO19(MISO), GPIO18(CLK), GPIO5(CS): pines por defecto para la comunicación por SPI.

La tarjeta de desarrollo también posee alimentación a 3V3 y 5V, tres pines conectados a GND (tierra) y dos pulsadores que se corresponden con los pines EN (reset) y GPIO0 (boot o gpio)

Para el desarrollo de la aplicación se han utilizado los siguientes pines:

PIN	Función
GPIO 19	SDA en el sensor BME280
GPIO23	SCL en el sensor BME280
GPIO21	SDA en la pantalla SSD1306
GPIO22	SCL en la pantalla SSD1306
GPIO 0	Salida LED1
GPIO 18	Salida LED2
GPIO 5	Salida LED3
GPIO 17	Salida LED4
GPIO 16	Salida LED5
GPIO 2	Salida LED6
GPIO 26	Salida LED sensor PIR
GPIO 27	Entrada detección sensor PIR

Tabla 6. Pines utilizados para la aplicación.

2.3 Diagramas de la aplicación

En primer lugar, se va a mostrar un diagrama de bloques de la aplicación, visualizando los componentes descritos en el apartado anterior:

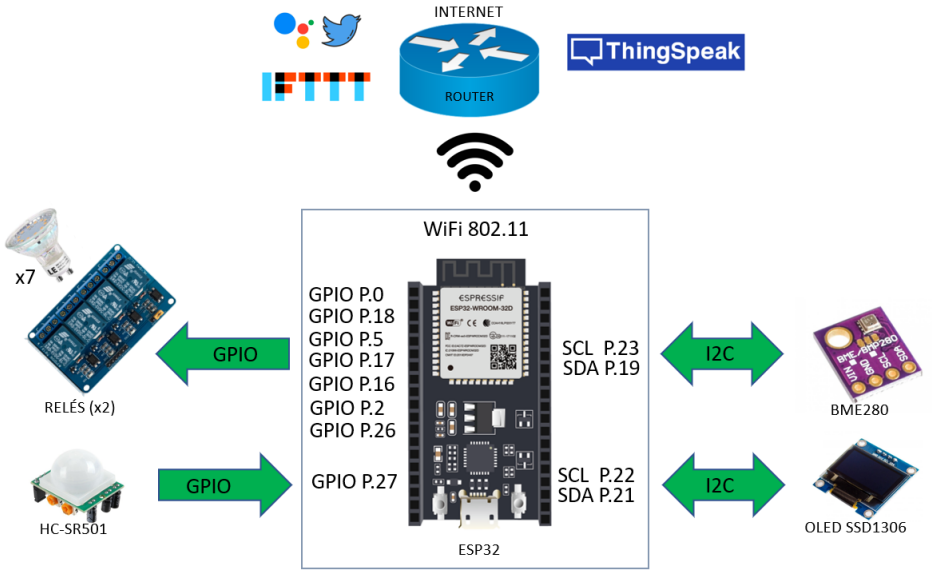

Figura 30. Diagrama de bloques de la aplicación.

En segundo lugar, se va a mostrar el conexionado de la aplicación. Se ha utilizado la herramienta Fritzing.

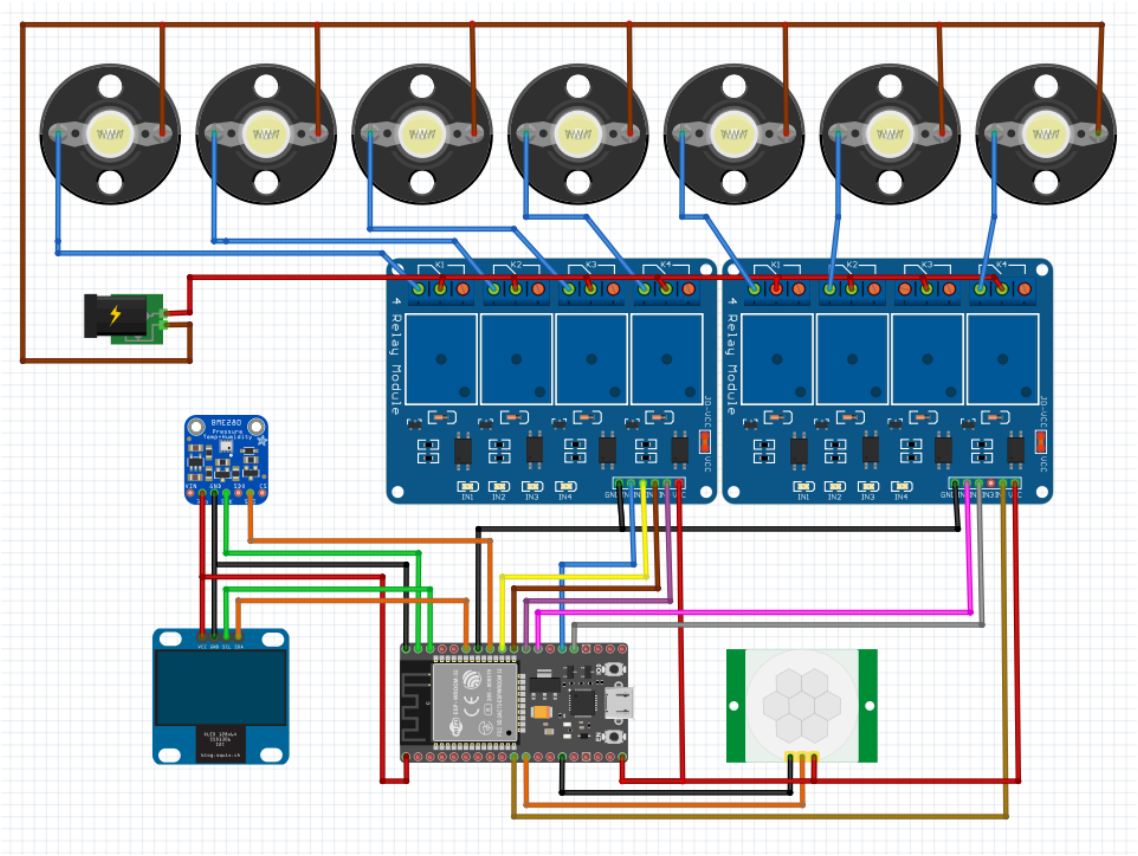

Figura 31. Diagrama de conexiones.

Los pines que se han utilizado se muestran en la tabla anterior (tabla 6). La alimentación de la pantalla OLED y del sensor BME280 es de 3v3. La alimentación del sensor PIR y de cada uno de los módulos de relés es de 5V.

Los seis primeros relés se asocian con cada una de las salidas de los pines GPIO y el relé octavo se conecta a la salida que provoca el sensor de presencia. En cuanto a las bombillas Led, la alimentación es de 220V AC y se conecta de la siguiente manera:

- Cable rojo: desde uno de los cables de alimentación de alterna se conectan entre sí todos los terminales comunes (C-Common) de cada relé.

- Cable marrón: el otro cable de alimentación se conecta en todas las bombillas a uno de sus contactos.

- Cable azul: el otro contacto de la bombilla se conecta al terminal normalmente abierto (NO-Normally Open) de cada relé.

2.4 Descripción interna de la aplicación

A continuación, se explicarán las partes más importantes del código desarrollado.
En primer lugar, hay que mencionar que para el desarrollo software se ha elegido el entorno de Arduino, ya que es más sencillo de usar que la otra alternativa que se barajaba, el SDK de Espressif Systems.

Para el uso del módulo ESP32 con Arduino hay que utilizar Arduino Core, basado en el sistema operativo FreeRTOS. Se trata de un sistema operativo en tiempo real de código abierto.

Este sistema operativo ofrece una serie de funciones que permiten la multitarea. El siguiente gráfico nos ayuda a ver los diferentes estados en los que puede estar una tarea en el sistema operativo FreeRTOS.

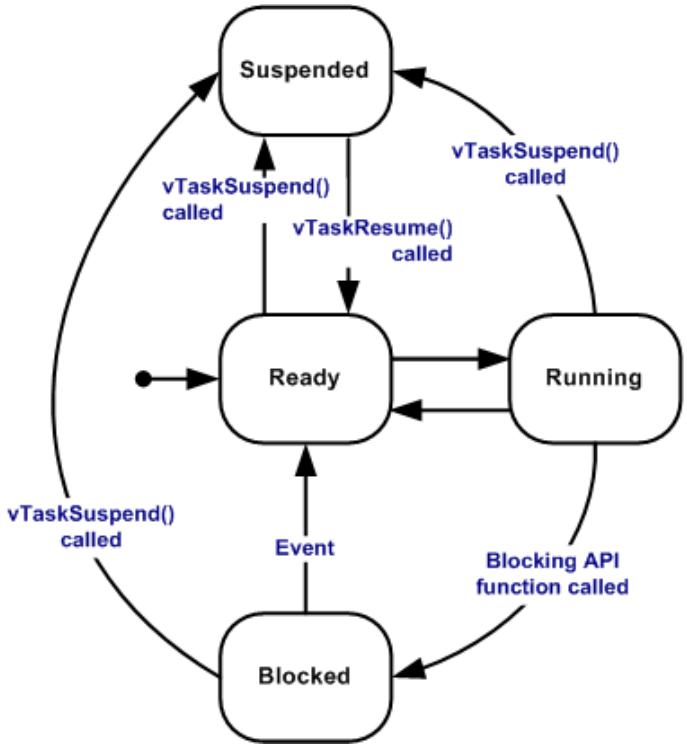

Figura 32. Gráfico de tareas en FreeRTOS.[19]

En nuestra aplicación la multitarea funcionará de la siguiente forma:

- Cuando una tarea esté en estado preparado (*ready)*, y no haya otra tarea ejecutándose, pasará a estado de ejecución (*runnning)*.
- Cuando la tarea acabe su parte del código se bloqueará mediante la llamada a la función *vTaskDelay(Ticks)* que permitirá pasar la tarea al estado bloqueado (*blocked)* durante un tiempo *Ticks*, establecido en milisegundos.
- En el caso de que haya dos tareas preparadas para pasar a ejecución, la que tenga una menor prioridad pasará a estado de ejecución.
- Si una tarea está ejecutándose y llega otra de mayor prioridad, la de mayor prioridad pasará a ejecución.

Para crear las tareas y asignar las prioridades se dispone de una función de FreeRTOS llamada *xTaskCreatePinnedToCore,* que nos permite definir los siguientes parámetros:

- Puntero a la función de la tarea.
- Nombre descriptivo de la tarea.
- Tamaño reservado en tamaño palabra para la tarea.
- Parámetro de entrada de la tarea.
- Prioridad de la tarea.
- Identificador de la tarea.
- Núcleo en el que debe ejecutarse la tarea.

Como se ha podido observar en el capítulo teórico de este trabajo, el módulo ESP32 tiene doble núcleo, por lo que se puede asignar con el uso de esta función tareas por separado a los diferentes núcleos. Esto es una gran ventaja ya que alivia el uso de la CPU y permite tener más de una tarea ejecutándose al mismo tiempo, cada una en un núcleo diferente.

A continuación, se muestran las tareas que se han creado:

```
xTaskCreatePinnedToCore
(INIT_Task,"INIT_Task",10000, NULL,3,NULL,1);

xTaskCreatePinnedToCore
(WEB_Task,"WEB_Task",10000, NULL,4,NULL,1);

xTaskCreatePinnedToCore
(OLED_Task,"OLED_Task",10000, NULL,3,NULL,1);

xTaskCreatePinnedToCore
(BME_Task,"BME_Task",10000, NULL,3,NULL,0);

xTaskCreatePinnedToCore
(PIR_Task,"PIR_Task",10000, NULL,3,NULL,1);

xTaskCreatePinnedToCore
(IFTTT_Task,"IFTTT_Task",10000, NULL,3,NULL,1);

xTaskCreatePinnedToCore
(ThingSpeak_Task,"ThingSpeak_Task",10000, NULL,3,NULL,0);
```

Código 1. Creación de las tareas.

En los siguientes apartados se explicarán las distintas tareas en detalle. Se ha decidido establecer la misma prioridad a las tareas, menos para el caso del servidor web, ya que un cliente web espera que el tiempo de respuesta del servidor sea reducido.

Respecto al reparto de núcleos, se ha decidido que la recogida de datos del sensor BME280 se realice en un núcleo diferente, ya que la transferencia de datos I2C no debería ser interrumpida, mientras que el resto de las tareas se asignan en el otro núcleo.

En la función setup() de Arduino se crearán todas las tareas sucesivamente, mientras que la función loop() de Arduino se le asigna el estado de tarea bloqueada continuamente.

2.4.1 Tarea BME_Task

La tarea BME_Task recopila información sobre los datos ambientales de temperatura y humedad a través del sensor BME280. Para la implementación del protocolo I2C en el sensor BME280 se han utilizado los pines SDA (22) y SCL (23) de la tarjeta de desarrollo ESP32 DevKit4C.

Además, se ha usado la librería "Adafruit_BME280_Library" [24] para la obtención de los valores de temperatura y humedad. Para ello hay que crear una clase de tipo *Adafruit_BME280* llamada *bme.* A continuación, se muestra parte del código utilizado:

```
void BME_Task( void * parameter)
   {

    Wire2.begin(SDA2,SCL2);
    status = bme.begin(0x76,&Wire2);
    if (!status) { Serial.println("sensor BME280 no conectado"); }

        bme.setSampling(Adafruit_BME280::MODE_NORMAL,
                    Adafruit_BME280::SAMPLING_X1,  // temperature
                    Adafruit_BME280::SAMPLING_X1, // pressure
                    Adafruit_BME280::SAMPLING_X1,  // humidity
                    Adafruit_BME280::FILTER_X16,
                    Adafruit_BME280::STANDBY_MS_0_5 );

    for(;;){

    Serial.print("Temperatura = ");
    temperatura=bme.readTemperature();
    Serial.print(temperatura);
    Serial.println(" *C");

    Serial.print("Humedad = ");
    humedad=bme.readHumidity();
    Serial.print(humedad);
    Serial.println(" %");

    vTaskDelay(10000);}
  }
```

Código 2. Tarea BME_Task.

Como se han utilizado dos líneas de comunicación I2C, es necesario abrir un segundo puerto para I2C con la línea de código TwoWire Wire2 = TwoWire(1) de la librería Wire.h de Arduino.

Para iniciar la comunicación I2C se inicializa el bus I2C para los pines SDA2 y SCL2 y a continuación comienza el envío del byte START (0x76). Para acabar la configuración inicial se llama a la función setSampling, que tiene diversos modos de medida en función de si el dispositivo se encuentra en interior o al aire libre. Para el desarrollo del proyecto se ha escogido la configuración Indoor.

Una vez que se entra en el bucle de la tarea se procede a la lectura del sensor. Posteriormente la tarea será bloqueada y se le asignará una espera de diez segundos, hasta que vuelva a tomar nuevos valores de temperatura y humedad.

2.4.2 Tarea PIR_Task

Esta tarea permite detectar cuando una persona entra en la sala donde está instalado el módulo ESP32, encendiendo uno de los Leds que están conectados al relé. El código de la tarea se muestra a continuación:

```
void PIR_Task(void *parameter)
{
   pinMode(sensor_PIR, INPUT_PULLUP);
   attachInterrupt(digitalPinToInterrupt(sensor_PIR), detectaPIR,
RISING);
   pinMode(PIR, OUTPUT);
   digitalWrite(PIR, LOW);
   for(;;)
   {
      if(wait && (millis() - lasttime > (10*1000)))
      {
        digitalWrite(PIR, LOW);
        wait = 0;
      }
      vTaskDelay(2000);
      }
}
```

Código 3. Tarea PIR_Task.

Para la implementación del sensor PIR se ha utilizado como entrada a la tarjeta ESP32 el pin 27 como pull-up y como salida hacia el actuador (relé) el pin 26. Posteriormente se ha creado una función llamada detectaPIR (). Dicha función se ejecutará en RAM cada vez que se detecte nivel alto en la salida del sensor PIR, poniendo a nivel alto la salida del relé. A la función se le asigna prioridad por interrupción en flanco de subida.

Cada dos segundos se verifica si se ha activado la interrupción viendo el valor de la variable "wait". En caso de haberse habilitado habrá que esperar diez segundos para apagar el LED conectado al relé.

2.4.3 Tarea INIT_Task

Es una tarea que usa la pantalla OLED SSD1306. Sirve para iniciar la pantalla con el mensaje: "Loading…". Esta tarea se borra del manejador de tareas con la función vTaskDelete (NULL), por lo que sólo se ejecuta una única vez al iniciar la aplicación.

```
void INIT_Task(void *parameter)
{
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.println("");
  display.println("");
  display.println("       LOADING...");
  display.display();
  vTaskDelete( NULL );
}
```

Código 4. Tarea INIT_Task.

2.4.4 Tarea OLED_Task

La tarea OLED_Task permite visualizar los datos tomados del sensor BME280 por la pantalla SSD1306, además de la dirección IP que se asigne al servidor web y la intensidad de señal (RSSI). Para la implementación se ha utilizado la librería "Adafruit_SSD1306".[26]

Primeramente, habrá que crear una clase de tipo Adafruit_SSD1306. A continuación, se muestra el extracto del código al que se hace referencia en esta tarea:

```
void OLED_Task(void *parameter)
  {
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  display.clearDisplay();
  display.setTextColor(WHITE);
  display.setRotation(0);
  display.setTextWrap(false);
  display.dim(0);

  for(;;){
    Serial.println("\nActualiza display");
    display.clearDisplay();
    display.setTextSize(0);
    display.setCursor(0,0);
    display.println("Temperatura:" + String(temperatura) +
(char)247 + "C");
    display.println("Humedad:" + String(humedad)+ "%");
    display.print("IP:");display.println(WiFi.localIP());
    display.println("RSSI:" + String(WiFi.RSSI()) + "dBm");
    display.display();
    vTaskDelay(10000);}
   }
```

Código 5. Tarea OLED_Task.

Para la implementación del protocolo I2C se envía la dirección del byte START, que en el caso de la pantalla SSD1306 es la dirección 0x3C.

Las funciones que implementa la librería hacen muy sencilla la representación de los valores por la pantalla. Es posible cambiar el tamaño del texto o seleccionar la posición del cursor y moverse por los 128x64 píxeles.

Para el caso de la aplicación se mostrará en la pantalla la temperatura, la humedad, y la IP del dispositivo, que posteriormente nos permitirá conectarnos al servidor web, y el nivel de señal RSSI. La tarea se repite en intervalos de diez segundos, mandando la tarea a estado bloqueado en cada ciclo, igual que la tarea de toma de medidas del sensor BME280.

2.4.5 Tarea WEB_Task

La tarea WEB_Task permite crear un servidor web que muestra los datos de temperatura y humedad y permite interactuar con los Leds conectados al relé, accediendo a dicho servidor escribiendo su dirección IP asignada en cualquier navegador que esté conectado a la misma red que el ESP32. Para implementar el funcionamiento del servidor web se ha basado en un ejemplo de la librería wifi de Arduino-esp32. [25]

Es en esta tarea donde se configura la tecnología WiFi del dispositivo, de modo que el establecimiento de la conexión sirva para las tareas posteriores (MQTT y ThingSpeak). A continuación, se muestran partes del código desarrollado para la tarea WEB_Task:

```
    WiFiServer server(80);
    WiFi.begin(ssid, pass);
    Serial.print("Conectando WiFi...");
    while (WiFi.status() != WL_CONNECTED)
    {
      Serial.println(".");
      delay(1000);
    }
```

Código 6. Establecimiento de conexión WiFi.

En primer lugar, para la creación del servidor web hay que indicar el puerto de destino, que en este caso es el puerto 80 (http). Se ha decidido que el módulo ESP32 se conecte a una red WiFi cercana de modo que se tenga acceso a Internet, por lo que el modo Wifi elegido en modo estación.

A continuación, se establece la conexión con la línea WiFi.begin introduciendo el usuario y la contraseña de la red a la que se quiera conectar, permaneciendo hasta que el flag de Wifi.status() se active, lo que indicará que la conexión se ha realizado con éxito. Una vez realizada la conexión, habrá que observar la entrada de peticiones en caso de existir un cliente web. El código que se ejecuta es el siguiente:

```
if (client)
    {
      Serial.println("New Client.");
      String html="";
      while (client.connected())
      {
         if (client.available())
         {
            char ultimo_byte = client.read();
            if (ultimo_byte == '\n')
            {
              if (html.length() == 0)
              {
                client.println("HTTP/1.1 200 OK");
                client.println("Content-type:text/html");
                client.println();
                client.println("<!DOCTYPE html><html>");
```

Código 7. Tarea WEB_Task

Si se detecta un cliente, se lee la petición byte a byte hasta que se llega al carácter “/n”, que indica el fin de la petición HTTP, ya que el protocolo HTTP escribe “/r/r/n/n” cuando acaba dicha petición HTTP.

A continuación, se manda la respuesta al cliente y con ella el código en HTML, que mostrará los datos en la interfaz web del usuario, como la temperatura, la humedad y el estado de los Leds, junto a una serie de botones que permitirán interactuar con el relé.

Una vez que tengamos la petición completa será posible comparar la cadena de la url (petición GET) con las páginas disponibles en el servidor y ver que página pide el cliente:

```
if (html.indexOf("GET /1/on") >=0)
 {
   led1="ON";
   digitalWrite(pin0, HIGH);
 }
else if (html.indexOf("GET /1/off") >=0)
 {
   led1="OFF";
   digitalWrite(pin0, LOW);
 }
```

Código 8. Petición GET.

La estructura para identificar que Led quiere encender o apagar el cliente web es la siguiente: GET/Número_de_Led/Estado, por lo que para el ejemplo del Led1 se encenderá si llega la url GET /1/on y se apagará si llega GET/1/off, e igualmente para el resto de Leds.

Finalmente, se cierra la conexión, y la tarea se bloquea durante un segundo, hasta que vuelve a comprobar si hay un cliente disponible.

Esta tarea está bloqueada durante menos tiempo ya que es necesario tener cierta recursividad para que la interfaz de usuario del servidor web no se haga lenta para el usuario.

2.4.6 Tarea IFTTT_Task

Esta tarea incluye la implementación del protocolo MQTT, la conexión a la página Adafruit.io y la herramienta IFTTT.

Para la implementación del servidor MQTT es necesario conocer su funcionamiento. MQTT es un protocolo ligero de transporte de mensajería de publicación y suscripción que está diseñado para que sea fácil de implementar.

El protocolo no implementa la arquitectura tradicional de cliente-servidor, sino que es un tercero el que gestiona ambas partes, llamado MQTT-Broker, que es el encargado de enviar y recibir mensajes, por lo que cliente y servidor no necesitan conocerse.

Principio de funcionamiento: un dispositivo puede enviar información sobre un topic al MQTT-Broker mediante publicación. Otro dispositivo podrá subscribirse a ese topic, de modo que cada

vez que el primer dispositivo envíe información, ésta llegará al segundo dispositivo. El encargado de realizar el proceso es el MQTT-Broker.
Para la programación del servidor MQTT se ha recurrido a la librería Adafruit_MQTT_Library [27]. La siguiente Figura muestra el comportamiento del protocolo en el caso de nuestra aplicación:

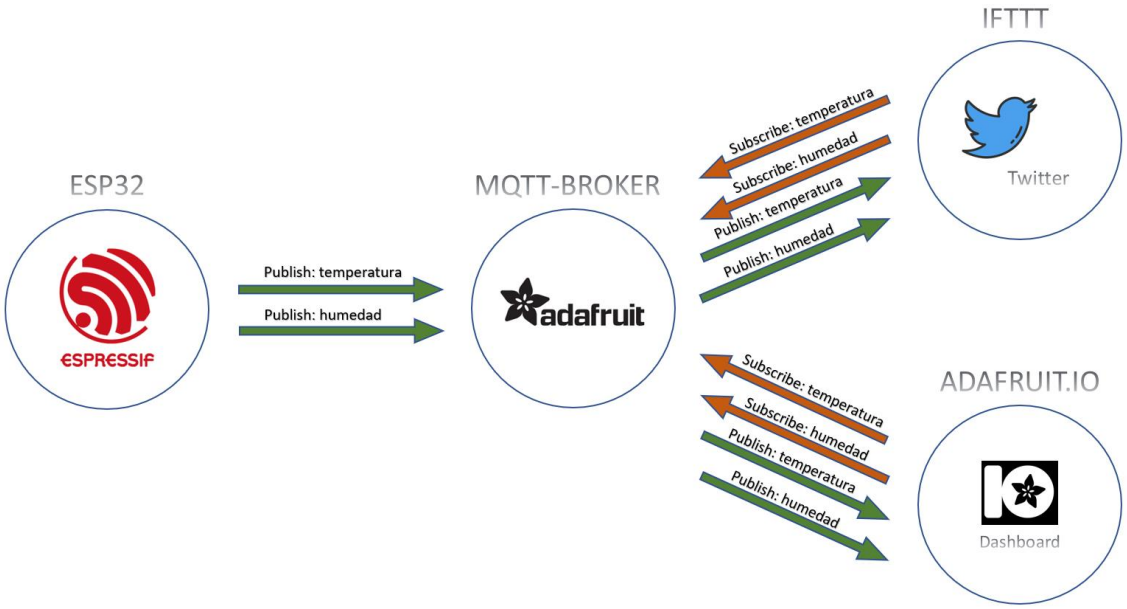

Figura 33. Publish/Subscribe temperatura y humedad.

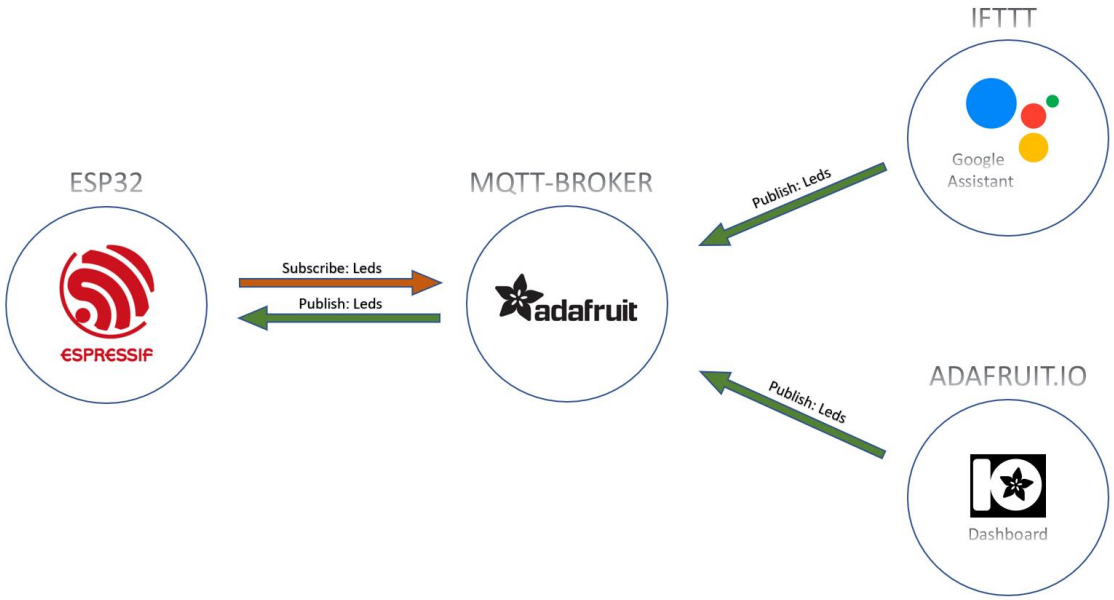

Figura 34 Publish/Subscribe estado de los Leds

El MQTT-Broker de nuestro sistema es adafruit.io, que será el encargado de enviar los mensajes que se publiquen de un topic determinado a los servicios o dispositivos que se hayan suscrito a dicho topic.

El dispositivo ESP32 publicará la temperatura y la humedad recogidos cada veinte segundos. Por otra parte, el ESP32 se suscribirá al topic Leds, para que cuando uno de los servicios publique un mensaje sobre Leds, el MQTT-Broker se lo envíe al dispositivo ESP32.

La interfaz de usuario de adafruit.io (dashboard), se comportará como un cliente MQTT que podrá publicar el estado de los Leds y recibir mediante suscripción los valores de temperatura y humedad.

Finalmente, el cliente IFTTT, posee dos servicios. Uno de ellos es Twitter, que, mediante la suscripción a la temperatura y la humedad publicados por el ESP32, escribirá automáticamente un tweet con esos valores cada veinte segundos. Por otra parte, el servicio de Google Assistant permite introducir por comandos de voz si se quiere apagar o encender los Leds, en cuyo caso se enviará una publicación al MQTT-Broker, y este, enviará al dispositivo ESP32 dicho mensaje para interactuar con el estado de los Leds.

A continuación, se muestran las partes más importantes del código de la tarea:

```
WiFiClient client;

Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT,
AIO_USERNAME, AIO_KEY);

Adafruit_MQTT_Subscribe Led1 = Adafruit_MQTT_Subscribe(&mqtt,
AIO_USERNAME"/feeds/LED1");

Adafruit_MQTT_Subscribe Led2 = Adafruit_MQTT_Subscribe(&mqtt,
AIO_USERNAME "/feeds/LED2");

Adafruit_MQTT_Subscribe Led3 = Adafruit_MQTT_Subscribe(&mqtt,
AIO_USERNAME "/feeds/LED3");

Adafruit_MQTT_Subscribe Led4 = Adafruit_MQTT_Subscribe(&mqtt,
AIO_USERNAME "/feeds/LED4");

Adafruit_MQTT_Subscribe Led5 = Adafruit_MQTT_Subscribe(&mqtt,
AIO_USERNAME "/feeds/LED5");

Adafruit_MQTT_Subscribe Led6 = Adafruit_MQTT_Subscribe(&mqtt,
AIO_USERNAME "/feeds/LED6");

Adafruit_MQTT_Subscribe TODO = Adafruit_MQTT_Subscribe(&mqtt,
AIO_USERNAME "/feeds/TODO");

Adafruit_MQTT_Publish Temperatura = Adafruit_MQTT_Publish(&mqtt,
AIO_USERNAME "/feeds/Temperatura");

Adafruit_MQTT_Publish Humedad = Adafruit_MQTT_Publish(&mqtt,
AIO_USERNAME "/feeds/Humedad");
```

Código 9. Definición de clases MQTT.

En primer lugar, es necesario crear un cliente WiFi. Después se crea el cliente MQTT con los siguientes parámetros:

- AIO_SERVER: se trata del MQTT-Broker que actuará de intermediario, en este caso será io.adafruit.com
- AIO_SERVERPORT: el puerto al que se conecta MQTT, por defecto 1883.
- AIO_USERNAME: el nombre de usuario de la cuenta Adafruit.
- AIO_KEY: la clave Adafruit IO.

Posteriormente se definen como clases de tipo Adafruit_MQTT_Publish las publicaciones de la temperatura y la humedad recogidas por el sensor BME280 en los topic /feeds/Temperatura y /feeds/Humedad respectivamente, y se definen como clases de tipo Adafruit_MQTT_Subscribe para suscribirse a los seis Leds del relé en los topic /feeds/LEDX.

Para suscribirse a los topic que envíe el usuario al MQTT-Broker hay que usar la siguiente función:

```
mqtt.subscribe(&Led1);
mqtt.subscribe(&Led2);
mqtt.subscribe(&Led3);
mqtt.subscribe(&Led4);
mqtt.subscribe(&Led5);
mqtt.subscribe(&Led6);
mqtt.subscribe(&TODO);
```

Código 10. Suscribirse al estado de los Led.

Una vez dentro del bucle de la tarea este es el código que se ejecuta continuamente:

```
MQTT_connect();
    Temperatura.publish(temperatura);
    Humedad.publish(humedad);
    Adafruit_MQTT_Subscribe *subscription;

    while ((subscription = mqtt.readSubscription(20000))) {
      if (subscription == &Led1) {
        Serial.print(F("Got: "));
        Serial.println((char *)Led1.lastread);
        int Led1_State = atoi((char *)Led1.lastread);
        digitalWrite(pin0, Led1_State);
      }
```

Código 11. Tarea IFTTT_Task.

Donde *MQTT_connect ()* es una función externa que comprueba si hay conexión MQTT y en caso de no haberla intentar reconectarse al servicio MQTT.

A continuación, lo que se hace es publicar la temperatura y la humedad al MQTT-Broker. Después se leen todas las suscripciones y se actúa para cada caso. En la captura se ha reflejado el caso del Led1. Lo que se recibe es la publicación por parte del cliente del estado del Led1, que puede ser uno o cero, y en consecuencia se encenderá o apagará el Led correspondiente.

La tarea hace un envío de datos de temperatura y humedad por medio de publicación cada veinte segundos, y se queda esperando posibles modificaciones en sus suscripciones el resto del tiempo.

2.4.7 Tarea ThingSpeak_Task

Esta tarea permite subir los datos de la temperatura y la humedad a la herramienta ThingSpeak, que nos permite almacenar dichos valores y analizarlos mediante Matlab.

Gracias a la librería ThingSpeak-Arduino la comunicación con la herramienta ThingSpeak es muy sencilla. [28]

```
void ThingSpeak_Task(void *parameter)
{

  WiFiClient client2;
  unsigned long channelID = 574014;
  const char * APIKEY = "GZMBK2LK2QUPECX2";
  ThingSpeak.begin(client2);
  for(;;)
   {
      ThingSpeak.setField(1,temperatura);
      ThingSpeak.setField(2,humedad);
      ThingSpeak.writeFields(channelID, APIKEY);
      vTaskDelay(20000);
   }
}
```

Código 12. Tarea ThingSpeak_Task.

Los pasos de la tarea son los siguientes:

- Crear un cliente de tipo WiFiClient.
- Seleccionar el canal y la APIKEY proporcionados en la cuenta de ThingSpeak.
- Comenzar la comunicación con ThingSpeak.
- En el bucle de la tarea seleccionar el primer campo para la temperatura y el segundo campo para la humedad.
- Posteriormente se bloquea la tarea durante veinte segundos para que pasado ese tiempo volver a escribir nuevos datos en ThingSpeak.

Se toman datos cada veinte segundos porque es una limitación de la cuenta gratuita de ThingSpeak, por lo que la tarea pasa a estado bloqueado hasta que vuelve a enviar los datos pasados los veinte segundos.

2.5 Configuración de los servicios web

2.5.1 Configuración de adafruit.io

Para la configuración de adafruit.io primeramente es necesario crear una cuenta. Una vez creada habrá que moverse a la pestaña *Dashboards* y en *Actions* se crea un nuevo panel de visualización.

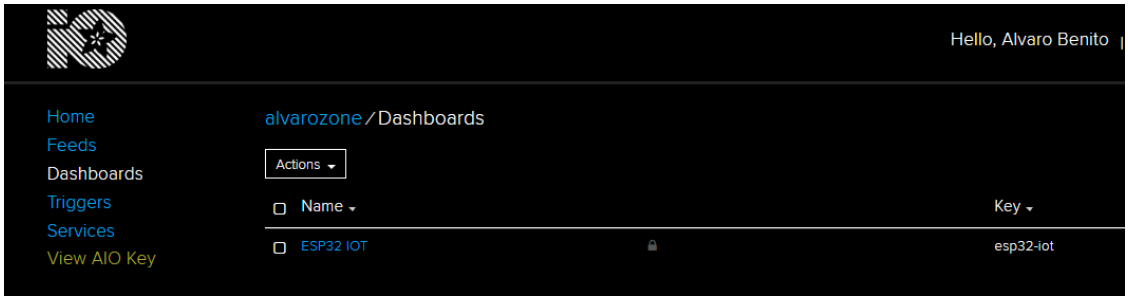

Figura 35. Creación del panel de control en adafruit.io

Después es necesario crear los *topics o feeds* para que luego sean visibles en el panel de control. Pulsando en el botón *Create* se podrán crear los distintos feeds, a los que se les darán los mismos nombres que en el código desarrollado en Arduino en el apartado 2.3.6.

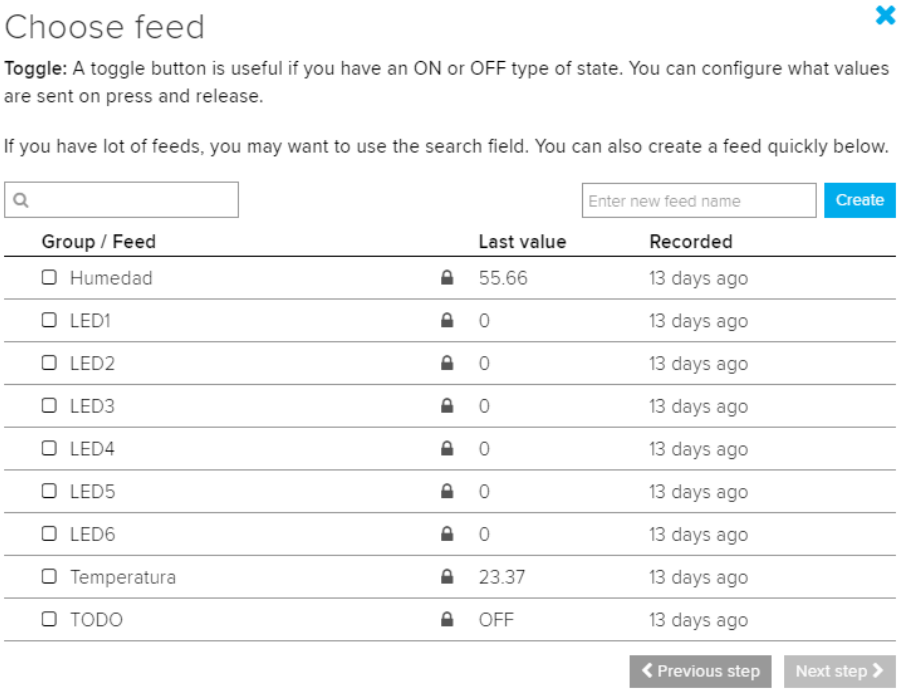

Figura 36 Creación de los feeds en adafruit.io

Posteriormente se crearán los bloques para la visualización por el panel de control. Se pueden añadir nuevos bloques en la parte superior derecha en el botón *Create New Block.* Para el caso de los Leds es importante configurar 1 o 0 en los campos que se muestran en la imagen, de cara a ser reconocidos por IFTTT o por el MQTT-Broker, como se muestra en la siguiente figura:

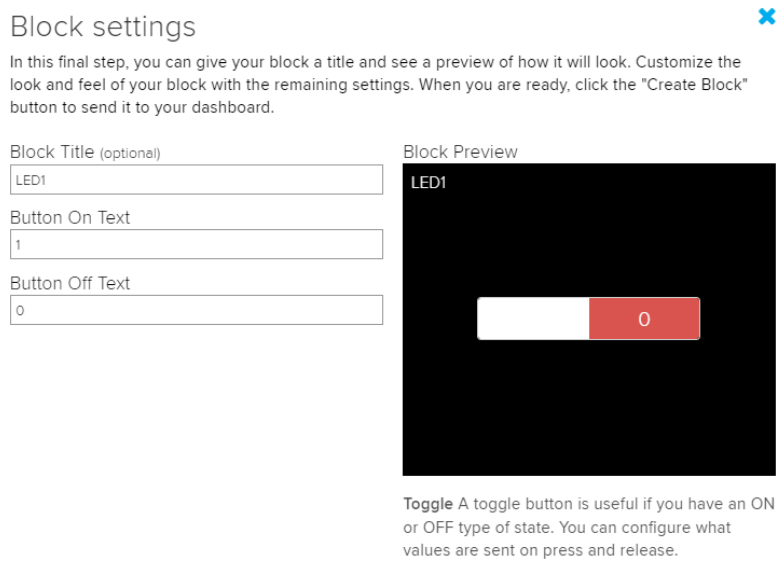

Figura 37 Creación del bloque toggle en adafruit.io

La siguiente figura hace referencia a la creación de un bloque para visualización de los valores de temperatura y humedad en el panel de control. Para configurarlo es necesario crear un bloque de tipo gauge. Es posible modificar ajustes como el valor mínimo y máximo y el ancho del objeto.

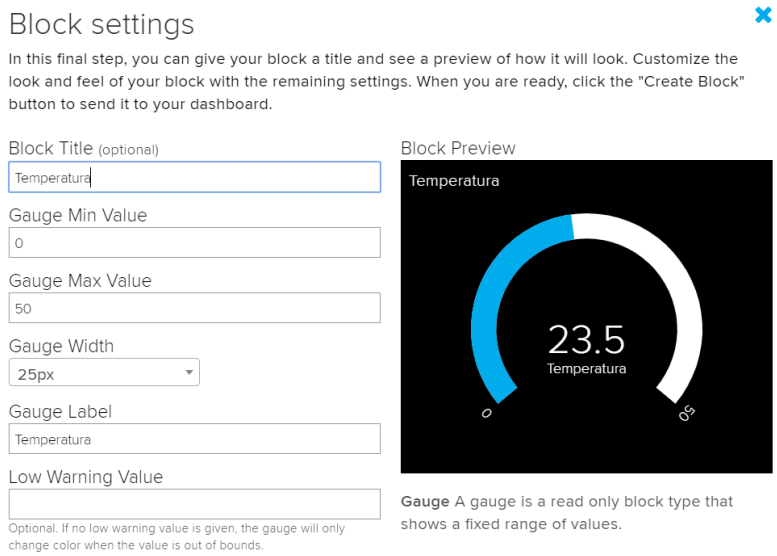

Figura 38 Creación del bloque gauge en adafruit.io

Por último, una última zona en la que se visualizan los valores de temperatura y humedad en una gráfica. Data es un array con los valores de temperatura y humedad. Como en el anterior caso se pueden modificar algunos parámetros como los valores límite o la cantidad de datos a mostrar en función del tiempo, como se muestra en la siguiente figura:

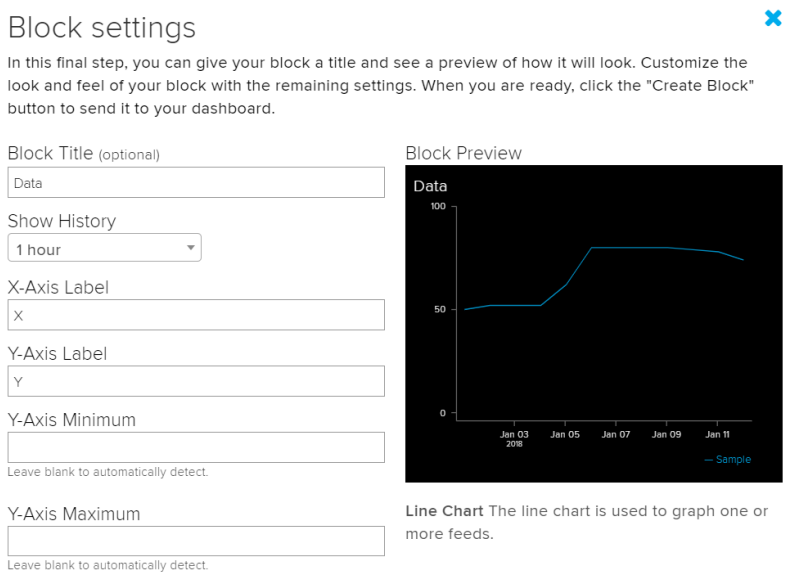

Figura 39. Creación del bloque line chart en adafruit.io

Una vez configurados todos los feeds podrán verse en un panel de control o dashboard, que es totalmente modificable a gusto del usuario.

2.5.2 Configuración de IFTTT

Al igual que en la configuración anterior, es necesario crear una cuenta para poder acceder a las funciones que ofrece IFTTT. Una vez creada la cuenta hay que entrar en MyApplets -> Applets -> NewApplet, en donde aparece la siguiente figura:

Figura 40. Creación de nuevo servicio en IFTTT.

Para el control de los Leds mediante comandos de voz de Google Assistant se pulsa sobre *this, y* aparecerán una serie de servicios. Escribiendo Google Assistant se puede encontrar el servicio que se pretende utilizar para el control del relé por comandos de voz.

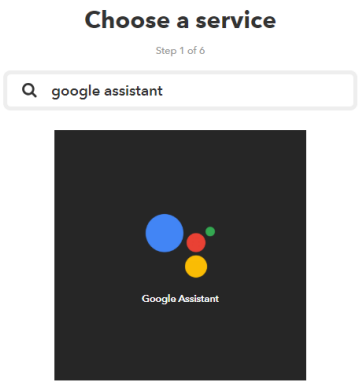

Figura 41. Selección del servicio Google Assistant.

Posteriormente se pulsa sobre *that*, y se escoge Adafruit como servicio de respuesta. A continuación, hay que configurar los parámetros como se muestra en la imagen. Para el caso de los demás Leds tanto encendido como apagado se repetiría el proceso. La configuración permite definir distintos comandos de voz y como se indica en la figura, asignar un valor a un feed determinado, como que el Led1 se le asigne un 1, lo que provoca que se encienda el led.

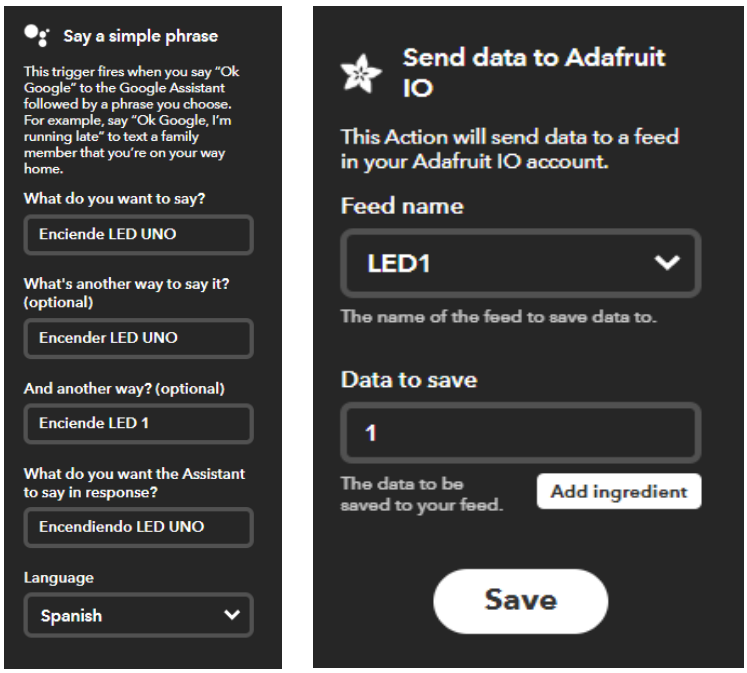

Figura 42 Configuración del servicio Google Assistant.

Para la visualización de los datos de la temperatura y la humedad se ha creado un usuario de Twitter que permite visualizar los datos a través de la plataforma cada veinte segundos con un nuevo servicio basado en Adafruit como parámetro *this* y Twitter como parámetro *that.* Se repetiría el proceso para la humedad.

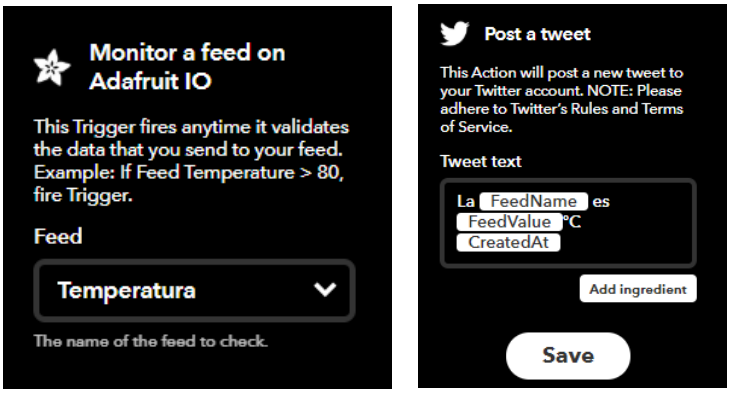

Figura 43. Configuración del servicio de Twitter.

También se han configurado alertas que serán enviadas por correo electrónico en caso de que la temperatura sobrepase un umbral inferior y superior. Finalmente se muestra una captura con todos los servicios incluidos:

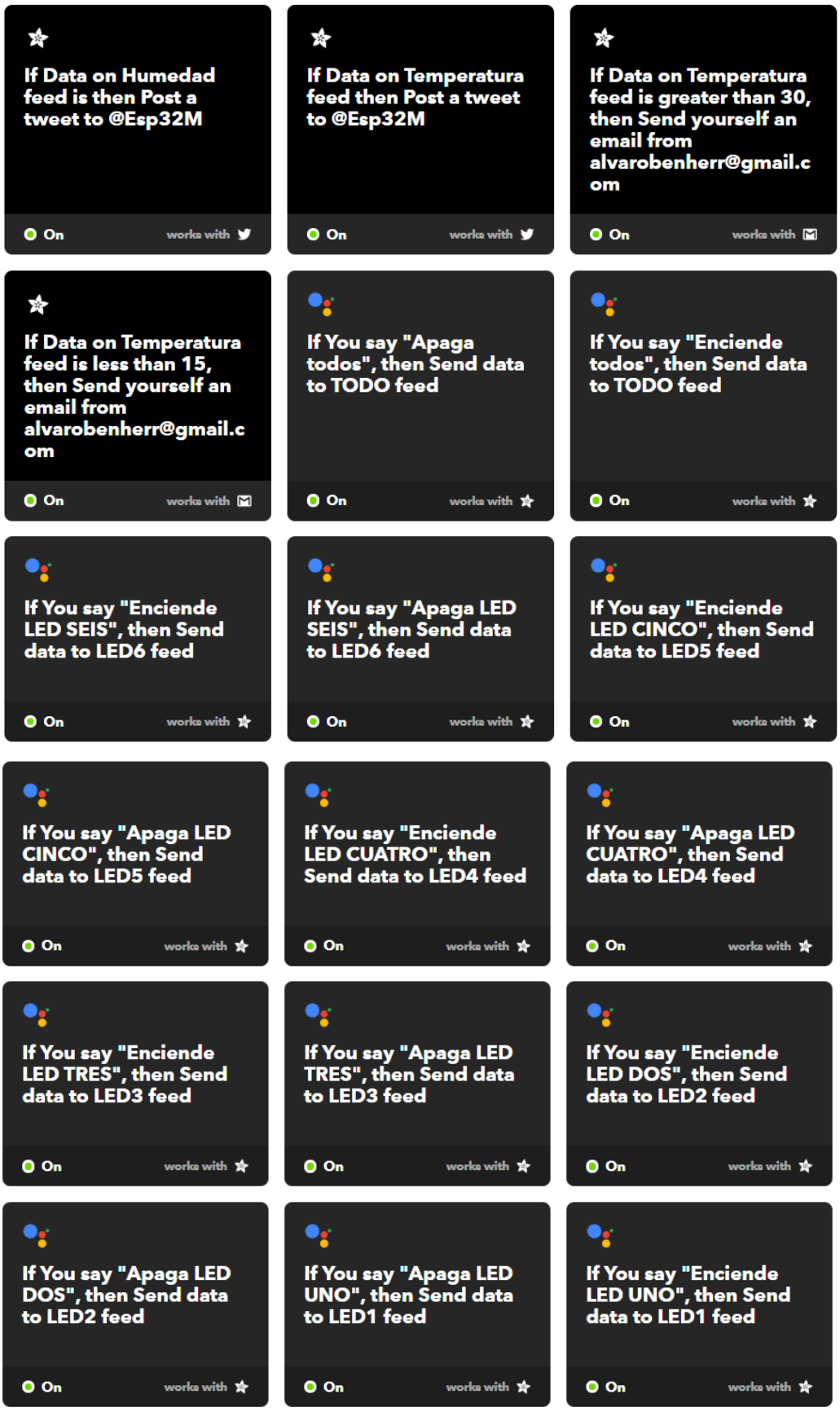

Figura 44. Servicios activados en IFTTT.

2.5.3 Configuración de ThingSpeak

Para el uso de ThingSpeak primeramente se creará una cuenta en dicha plataforma. Tras la creación de la cuenta habrá que dirigirse a la sección de canales y seleccionar la creación de un nuevo canal.

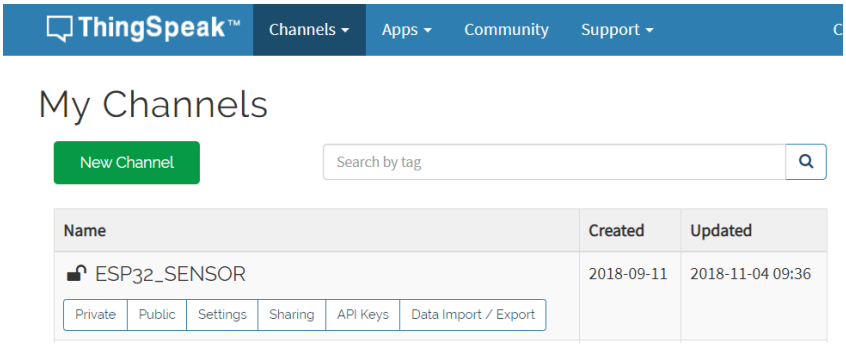

Figura 45. Creación de un canal en ThingSpeak.

La creación del canal se llevará a cabo rellenando los campos uno y dos con los valores que se quieran mostrar en ThingSpeak, que en nuestro caso son la temperatura y la humedad.

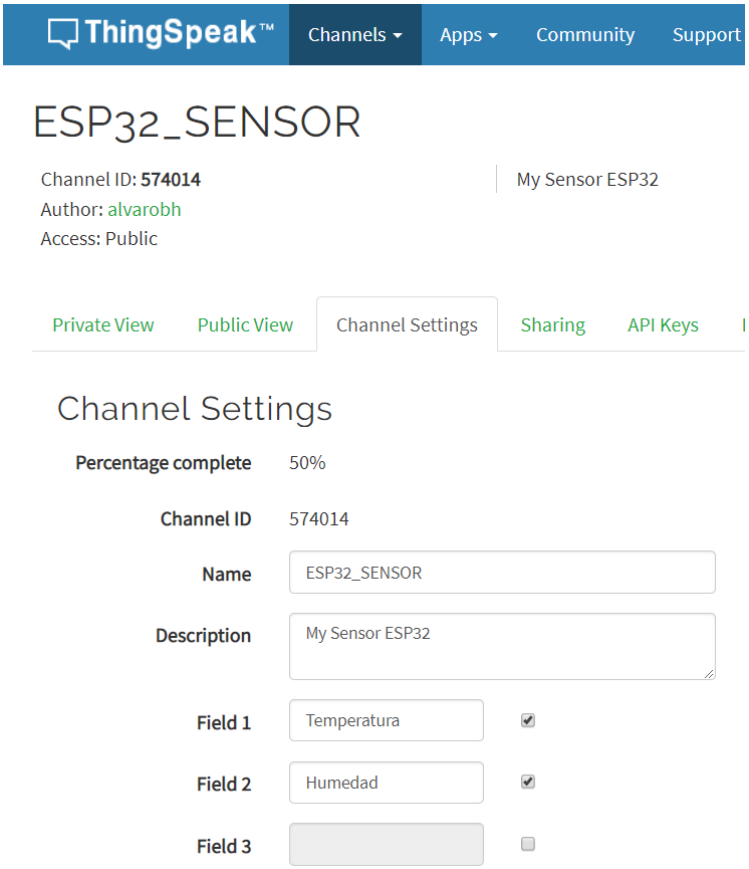

Figura 46. Creación de los campos del canal de ThingSpeak.

En el apartado *API Keys* se mostrarán las claves privadas para poder escribir valores en ThingSpeak.

Capítulo 3

Resultados de la aplicación

A continuación, se van a describir los diferentes resultados en cada una de las plataformas web en las que se ha basado el desarrollo de la aplicación.

En primer lugar, mediante el acceso web a la plataforma adafruit.io, se pueden visualizar los feeds configurados en el capítulo anterior tal y como se muestra en la siguiente figura:

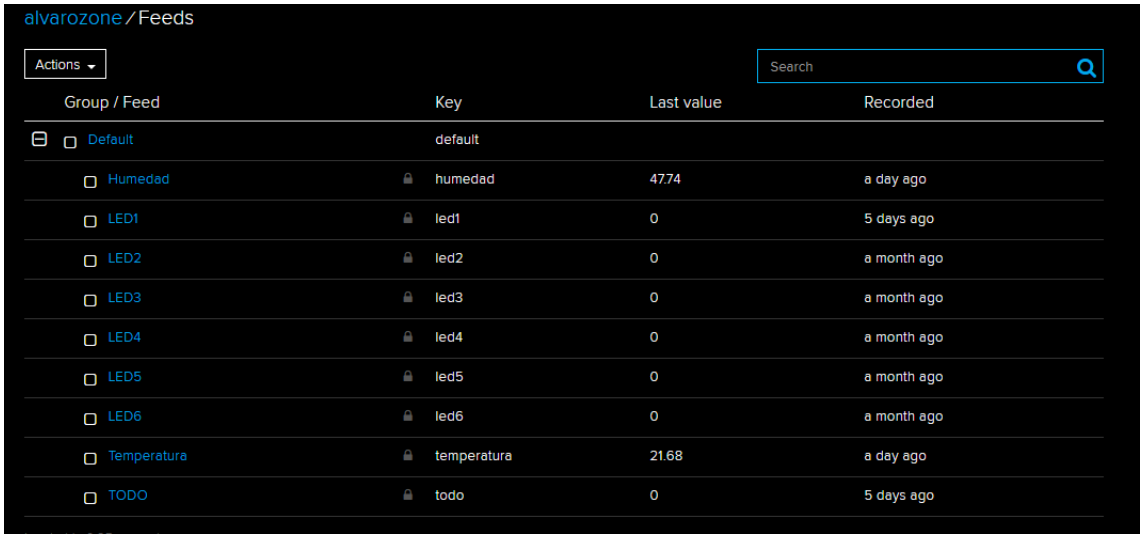

Figura 47. Feeds disponibles en adafruit.io

Cuando se pulsa sobre uno de ellos aparece información sobre el feed, como, por ejemplo, en el caso del LED1 aparece información sobre el estado del relé y la fecha en las que se conmutó el relé. Igualmente proporciona información sobre el estado de la temperatura o de la humedad si se entra dentro de su respectivo feed.

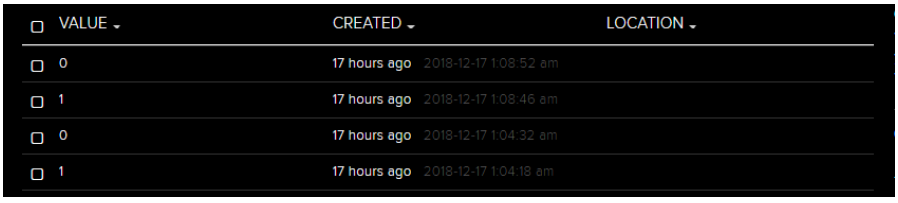

Figura 48. Feed del LED1

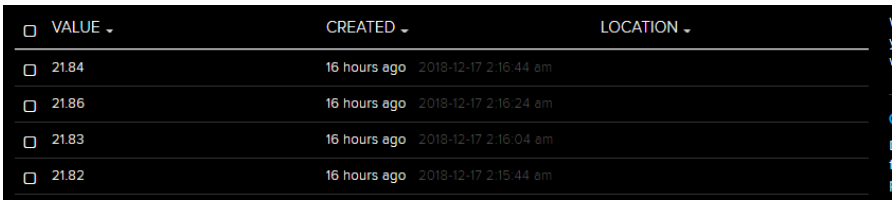

Figura 49. Feed de la temperatura.

Además, desde la visión del *Dashboard* o panel de control que se ha creado es posible apagar o encender los Leds, y visualizar los valores de temperatura y humedad. Estos valores se actualizan cada veinte segundos. En la siguiente figura se puede observar el panel de control desarrollado.

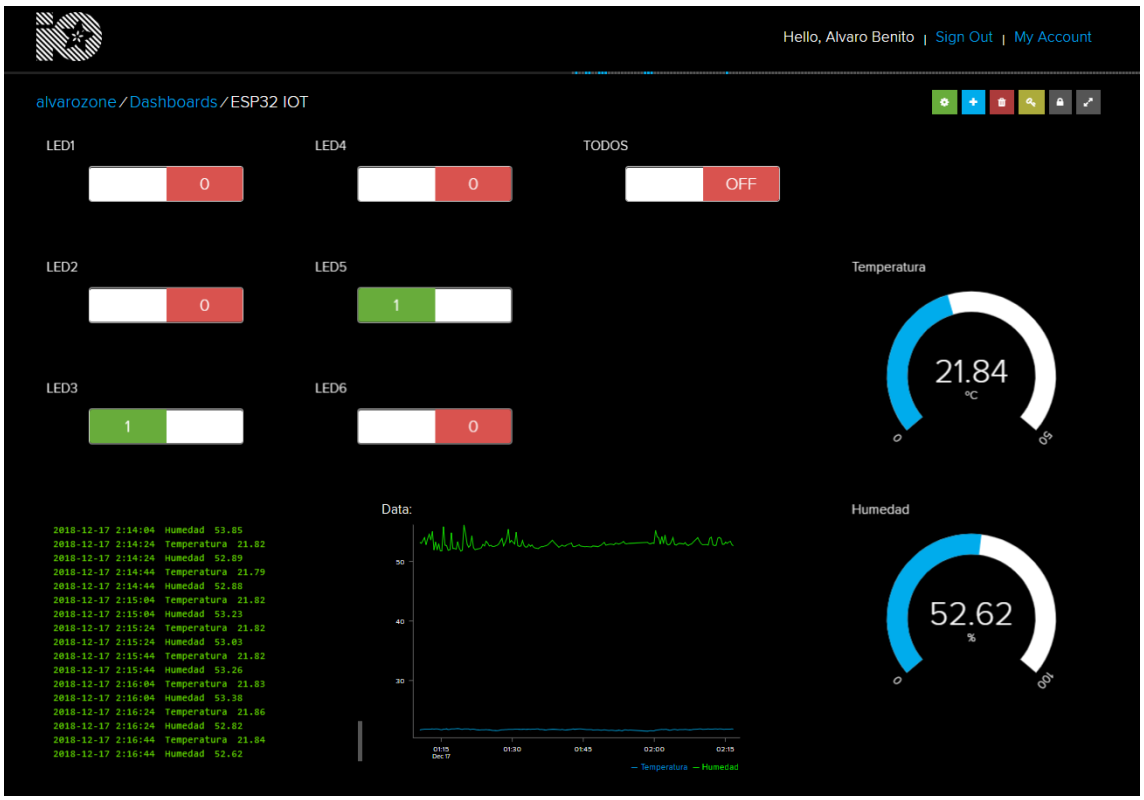

Figura 50. Panel de control de adafruit.io

Se han creado diferentes bloques para visualizar los datos de temperatura y humedad por medio de un bloque tipo gauge y de una gráfica en el que se dibujan los valores en función del tiempo. Además, existe un registro de la actividad de ambos valores en la parte inferior izquierda. Por otra parte, los bloques de tipo toggle permiten cambiar el estado del relé haciendo clic en los mismos.

Como se ha mencionado en el capítulo anterior, se utiliza la herramienta IFTTT para comunicarse con el dispositivo ESP32 mediante el protocolo MQTT, utilizando el MQTT-Broker de adafruit.io De esta forma se puede tener acceso a los diferentes servicios que ofrece IFTTT. Entre todos los posibles servicios se han escogido dos. El primero de ellos es Twitter, donde se subirán los datos de temperatura y humedad al canal de Twitter @Esp32M cada veinte segundos. Hay que decir que la herramienta establece un límite diario de cien tweets por usuario. En la siguiente figura se puede visualizar un ejemplo de la temperatura y humedad obtenidas:

Figura 51. Ejemplo de monitorización utilizando Twitter.

El otro servicio que se ha implementado mediante IFTTT es capaz de controlar el relé que está conectado al dispositivo ESP32 por comandos de voz. Con la aplicación móvil *Google Assistant* se pueden introducir comandos escritos o por voz de manera que por el protocolo MQTT se publiquen nuevos valores del estado del relé. En la siguiente figura se puede observar un ejemplo de la interfaz de *Google Assistant* en la que primeramente se apaga el Led1 que había sido encendido previamente y a continuación se encienden y se apagan todos los Leds, con los comandos que se habían visto en el capítulo anterior:

Figura 52. Ejemplo de uso de los comandos de voz.

Otro de los servicios web implementados es la creación de un servidor web sencillo que permita visualizar el valor de la temperatura, la humedad y el estado de los Leds, pudiendo a su vez modificar el estado del relé pulsando sobre los botones que aparecen en la siguiente figura, que enviarán una petición GET como ya se vio en el capítulo anterior. La interfaz es la siguiente:

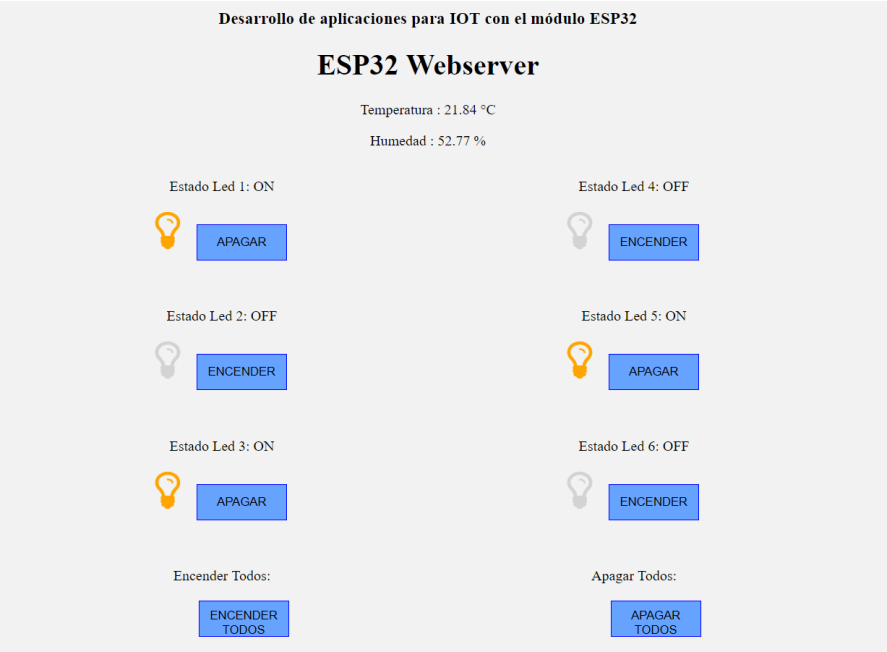

Figura 53. Interfaz del servidor web.

Para el acceso al servidor web es necesario conocer su IP. Esta información la podemos obtener en la pantalla OLED junto a la temperatura, humedad y nivel de señal RSSI, como se ve en la imagen siguiente:

Figura 54. Pantalla OLED

A continuación, se va a mostrar un análisis de los datos de temperatura y humedad obtenidos por la herramienta ThingSpeak, que incluyen gráficas proporcionadas por ThingSpeak y gráficas obtenidas por un análisis en Matlab. Esto es debido a que ambas herramientas pertenecen a la misma organización (MathWorks) y Matlab dispone de una API para comunicarse con ThingSpeak.

El estudio se ha realizado mandando datos a ThingSpeak sobre la temperatura y la humedad, obteniendo muestras cada veinte segundos. En total se han recogido 1200 muestras lo que supone un intervalo de casi siete horas. El estudio se ha llevado a cabo en el interior de una habitación.

En primer lugar, se va a mostrar la distribución de la temperatura y de la humedad de las 1200 muestras obtenidas:

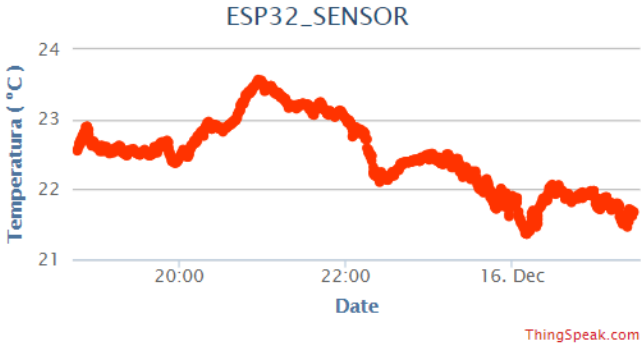

Figura 55. Gráfica de la temperatura con 1200 muestras.

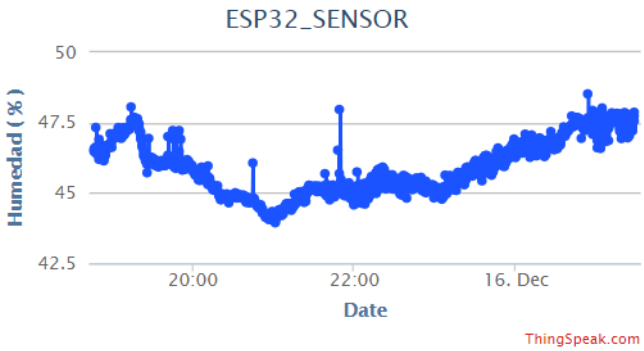

Figura 56. Gráfica de la humedad con 1200 muestras.

Como se puede observar el valor de temperatura varía entre los 21.5ºC y los 23.5ºC aproximadamente. La humedad varía entre el 44% y el 48% aproximadamente.

También se muestra a continuación las gráficas para un número de muestras menor. En este caso serán las últimas 30 muestras medidas, donde se puede apreciar la toma de muestras cada veinte segundos.

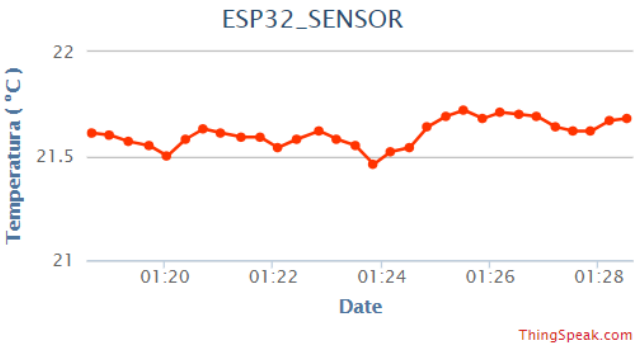

Figura 57. Gráfica de la temperatura con 30 muestras.

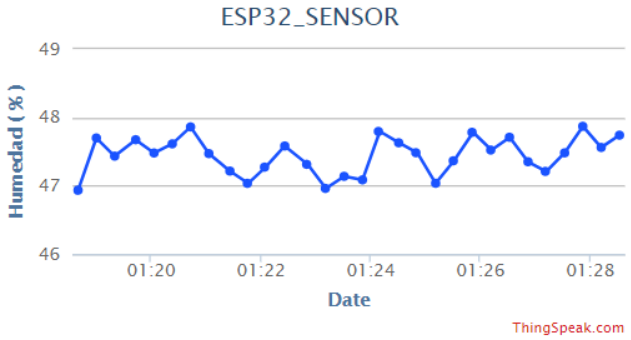

Figura 58. Gráfica de la humedad con 30 muestras.

Debido a la relación con Matlab, ThingSpeak tiene una serie de códigos ejemplo que pueden ser ejecutados desde la interfaz web y que simulan la herramienta Matlab. Dicho código puede ser ejecutado por igual desde la interfaz web como desde el software de Matlab, pudiendo modificarse como se quiera, y añadiendo el código que se considere necesario.

La siguiente gráfica representa los datos correlados de temperatura y humedad, produciendo un diagrama de dispersión que nos permite visualizar los valores típicos de temperatura en función de la humedad y viceversa, durante las 1200 muestras tomadas. También se adjunta el código utilizado en Matlab para la representación de la gráfica.

```
%channel ID to read data from:
readChannelID = 574014;
%Field ID to read data from:
fieldID1 = 1;
%Field ID to read data from:
fieldID2 = 2;
% Channel Read API Key
readAPIKey = 'XC0269GV1UJHZPKB';
% Read first data variable
data1 = thingSpeakRead(readChannelID, 'Field', fieldID1,...
            'NumPoints', 1200, 'ReadKey', readAPIKey);
% Read second data variable
data2 = thingSpeakRead(readChannelID, 'Field', fieldID2,...
            'NumPoints', 1200, 'ReadKey', readAPIKey);
% Visualize Data %%
scatter(data1, data2, 30, 'green', 'filled');
xlabel('Temperatura ( ºC )');
ylabel('Humedad ( %)');
```

Código 13. Correlación Temperatura/Humedad.

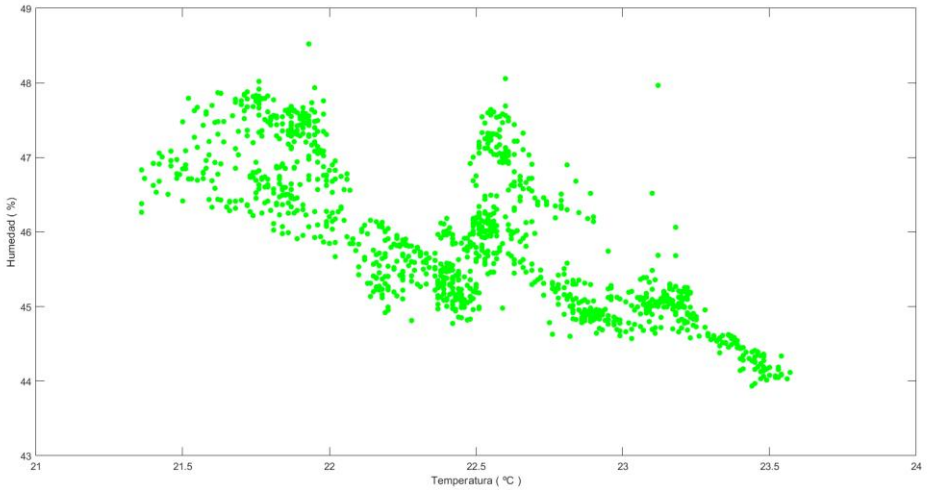

Figura 59. Gráfica de correlación de temperatura y humedad.

La siguiente figura muestra la representación en la misma gráfica de la temperatura y de la humedad durante las 1200 muestras empleadas. A continuación, se adjunta además el código utilizado.

```
% Channel ID to read data from:
readChannelID = 574014;
% Field ID to read data from:
fieldID1 = 1;
% Field ID to read data from:
fieldID2 = 2;
% Channel Read API
readAPIKey = 'XC0269GV1UJHZPKB';
% Read first data variable
[data1, time1] = thingSpeakRead(readChannelID, 'Field',fieldID1,...
        'NumPoints', 1200, 'ReadKey', readAPIKey);
% Read second data variable
[data2, time2] = thingSpeakRead(readChannelID, 'Field',fieldID2,...
        'NumPoints', 1200, 'ReadKey', readAPIKey);
% Read second data variable
[data3, time3] = thingSpeakRead(readChannelID, 'Field',...
        [fieldID1,fieldID2], 'NumPoints', 1200, 'ReadKey',...
        readAPIKey,'OutputFormat','timetable');
% Visualize Data %
yyaxis left;
plot(time2, data2,'blue');
ylim([43 49])
ylabel('Humedad ( % )');

yyaxis right;
plot(time1, data1,'red')
ylim([21 24])
ylabel('Temperatura ( ºC )');
legend({'Humedad','Temperatura'});
```

Código 14. Datos en la misma gráfica

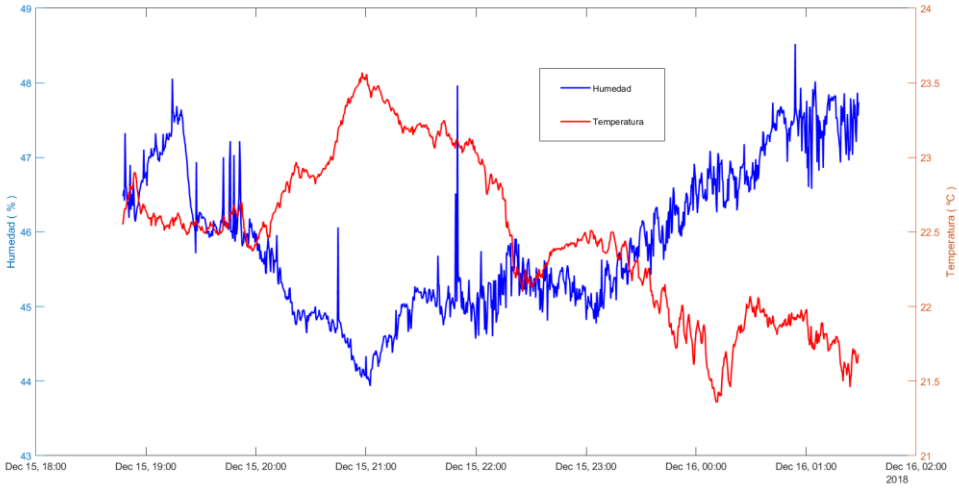

Figura 60. Temperatura y humedad en la misma gráfica.

De la figura de puede extraer a grandes rasgos que la temperatura en este análisis es inversamente proporcional a la humedad, ya que se observa que cuando la temperatura aumenta disminuye la humedad y viceversa.

Los datos obtenidos en el código ejecutado en Matlab se guardan de la siguiente manera, permitiendo la visualización de la fecha en la que se tomaron los datos, el valor de la temperatura y el valor de la humedad, lo que permite seleccionar por días, horas o incluso minutos.

	Timestamps	1 Temperatura	2 Humedad
1	15-Dec-2018 18:47:16	22.5500	46.4941
2	15-Dec-2018 18:47:36	22.5700	46.4844
3	15-Dec-2018 18:47:56	22.6100	46.5635
4	15-Dec-2018 18:48:16	22.6500	46.4190
5	15-Dec-2018 18:48:36	22.6600	47.3262
6	15-Dec-2018 18:49:02	22.6500	46.6309
7	15-Dec-2018 18:49:22	22.6800	46.5801
8	15-Dec-2018 18:49:42	22.7100	46.4053
9	15-Dec-2018 18:50:02	22.7100	46.3633
10	15-Dec-2018 18:50:23	22.7400	46.4082
11	15-Dec-2018 18:50:43	22.7700	46.1904
12	15-Dec-2018 18:51:03	22.7300	46.4600
13	15-Dec-2018 18:51:23	22.8100	46.9004
14	15-Dec-2018 18:51:43	22.8100	46.2988
15	15-Dec-2018 18:52:04	22.7900	46.4238
16	15-Dec-2018 18:52:24	22.7900	46.5088
17	15-Dec-2018 18:52:46	22.7900	46.3506
18	15-Dec-2018 18:53:06	22.8400	46.6816
19	15-Dec-2018 18:53:26	22.8900	46.5186
20	15-Dec-2018 18:53:46	22.9000	46.2031

Figura 61. Forma de almacenamiento de datos en Matlab.

El siguiente código permite visualizar un histograma de la temperatura y la humedad, pudiendo observar en que tramos de temperatura y humedad se han obtenido un número mayor de muestras.

```
% Channel ID to read data from
readChannelID = 574014;
% Temperature Field ID
TemperatureFieldID = 1;
% Channel Read API Key
readAPIKey = 'XC0269GV1UJHZPKB';
% Read Data
temp = thingSpeakRead(readChannelID,'Fields',TemperatureFieldID,...
'NumPoints', 1200, 'ReadKey',readAPIKey);
histogram(temp,100,'FaceColor','r');
xlabel('Temperature (ºC)');
ylabel('Numero de medidas\newline por cada valor de temperatura');
```

Código 15. Creación de un histograma.

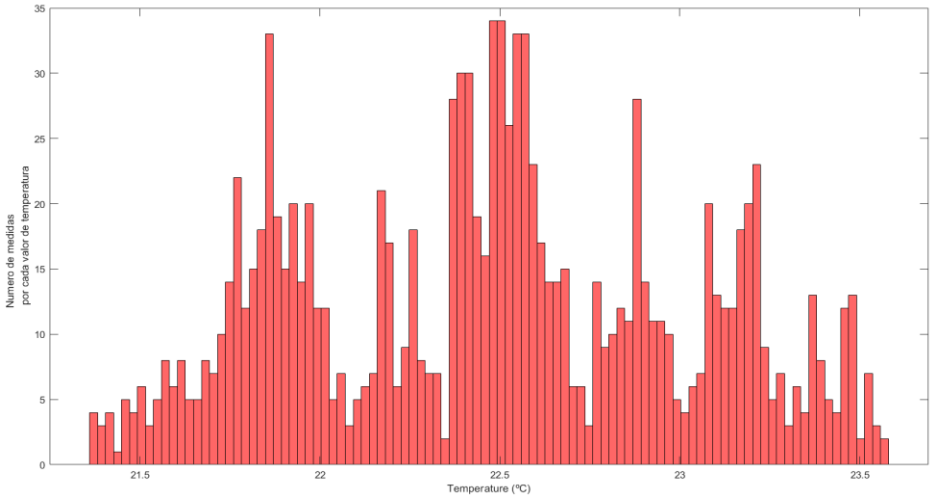

Figura 62. Histograma de la temperatura.

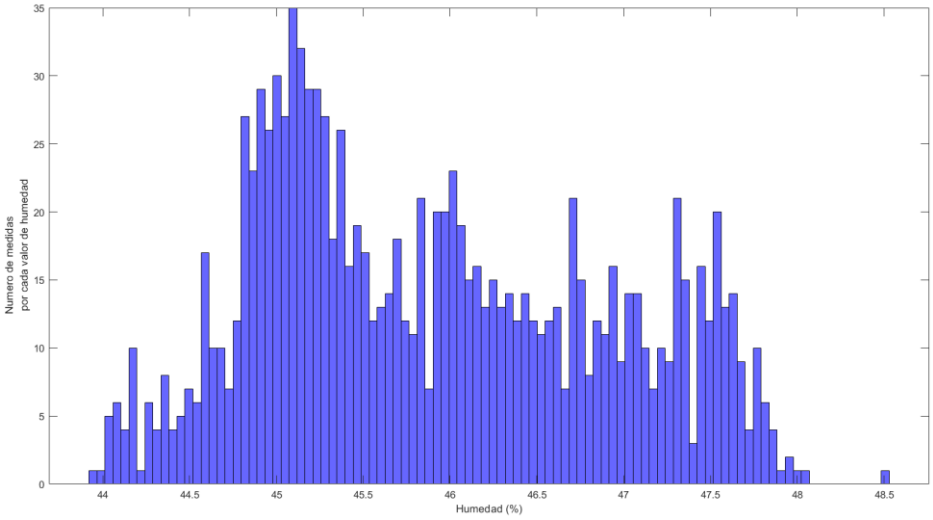

Figura 63. Histograma de la humedad.

Para terminar el análisis se ha desglosado la temperatura y la humedad en tramos de una hora y se han comparado tanto la temperatura como la humedad en cinco horas diferentes. Se adjuntan el código utilizado en Matlab y ambas gráficas:

```
% Channel ID to read data from
readChannelID = 574014;
% Temperature Field ID
TemperatureFieldID = 1;
% Channel Read API Key
readAPIKey = 'XC0269GV1UJHZPKB';
% Read Temperature Data
temperatureDay1 = thingSpeakRead(readChannelID,'Fields',...
        TemperatureFieldID,'dateRange', [datetime('today')-hours(1),...
        datetime('today')],'ReadKey',readAPIKey);

temperatureDay2 = thingSpeakRead(readChannelID,'Fields',...
        TemperatureFieldID,'dateRange', [datetime('today')-hours(2),...
        datetime('today')-hours(1)],'ReadKey',readAPIKey);

temperatureDay3 = thingSpeakRead(readChannelID,'Fields',...
        TemperatureFieldID,'dateRange', [datetime('today')-hours(3),...
        datetime('today')-hours(2)],'ReadKey',readAPIKey);

temperatureDay4 = thingSpeakRead(readChannelID,'Fields',...
        TemperatureFieldID,'dateRange', [datetime('today')-hours(4),...
        datetime('today')-hours(3)],'ReadKey',readAPIKey);

temperatureDay5 = thingSpeakRead(readChannelID,'Fields',...
        TemperatureFieldID, 'dateRange', [datetime('today')-hours(5),...
        datetime('today')-hours(4)], 'ReadKey',readAPIKey);

% Create the array of temperatures
allData=[temperatureDay1(1:177),temperatureDay2(1:177),...
temperatureDay3(1:177),temperatureDay4(1:177),temperatureDay5(1:177)];

% Visualize the data
plot((1:177),allData);
legend({'Hour(0)','Hour(-1)','Hour(-2)','Hour(-3)','Hour(-4)'});
xlabel('Numero de muestras/Hora');
```

Código 16. Desglose por horas.

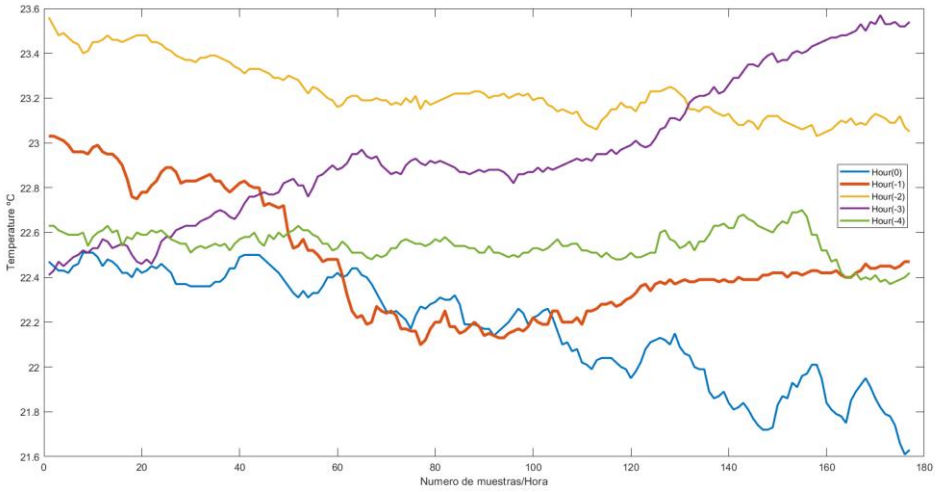

Figura 64. Gráfica de la temperatura en cinco horas distintas.

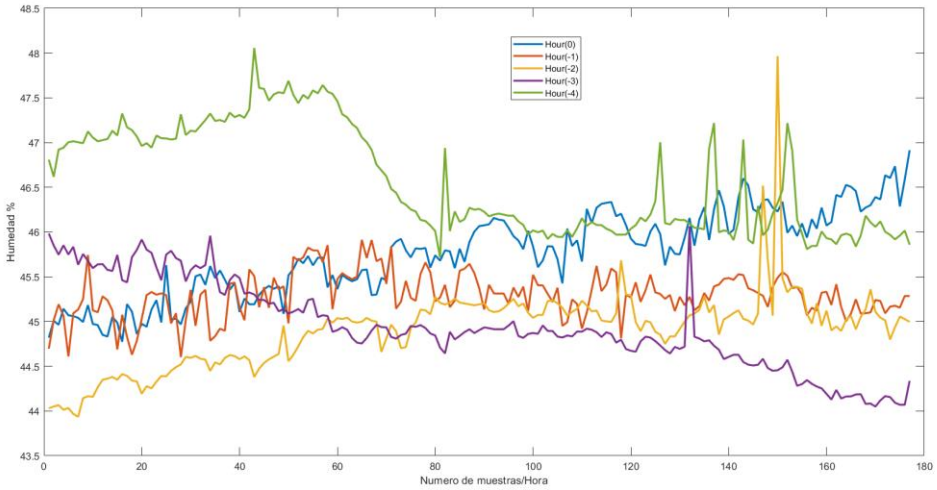

Figura 65. Gráfica de la humedad en cinco horas distintas.

Por último, se muestra una imagen general del sistema completo, incluyendo todos los componentes utilizados.

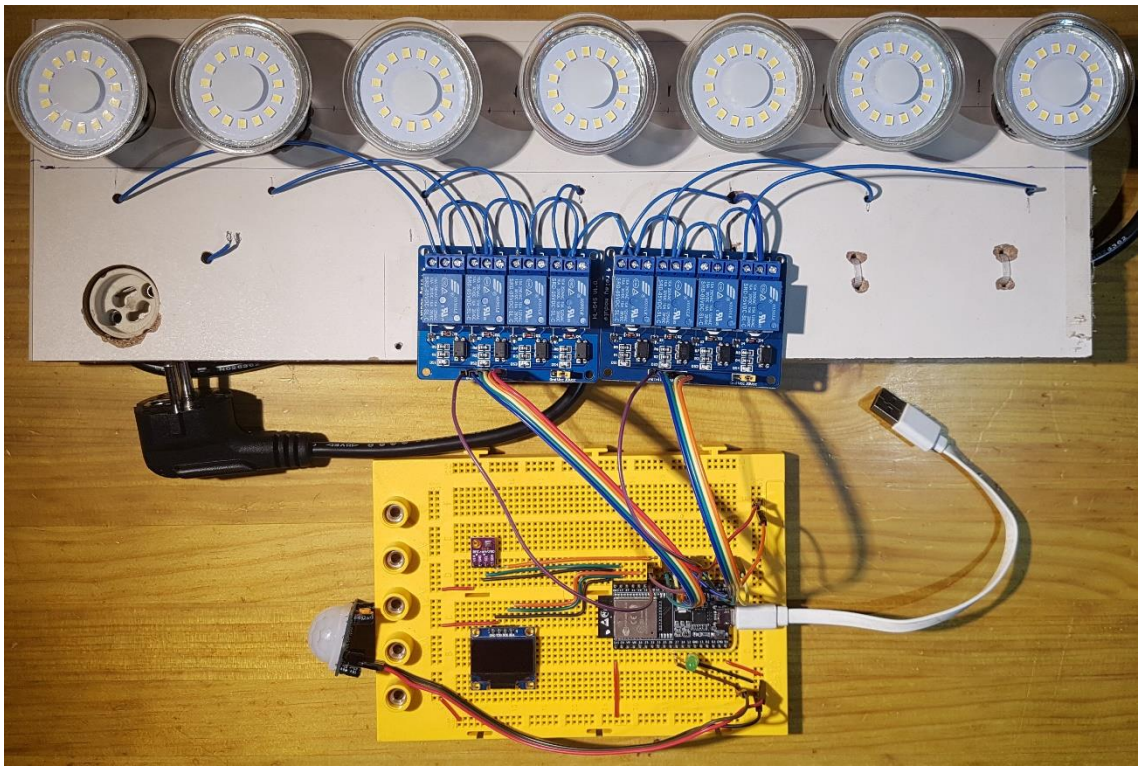

Figura 66. Captura del sistema completo.

Conclusiones y líneas futuras

A modo de resumen hemos podido conocer las diferentes características del dispositivo ESP32, sus funcionalidades y periféricos, de modo que tenemos en líneas generales un mejor conocimiento del dispositivo.

Por otra parte, hemos desarrollado una aplicación que nos ha permitido recordar conceptos vistos durante el grado y adquirir nuevos, como el protocolo MQTT.

Además, nos ha permitido conocer diversas herramientas enfocadas a IoT, que facilitan la interacción entre el usuario y el dispositivo, como ThingSpeak e IFTTT.

Por último, nos hemos familiarizado con distintos entornos de desarrollo como ESP-IDF, Mongoose OS y Arduino, que nos permiten adquirir experiencia en caso de usarse en el futuro.

En cuanto a las posibles mejoras que se podrían implementar al desarrollo de la aplicación destacan las siguientes:

- Aumento de los sensores que intercambian información con el dispositivo.
- Utilizar el ADC y el DAC del ESP32.
- Utilizar el PWM del dispositivo, por ejemplo, con un servomotor.
- Comunicación máquina-maquina con el protocolo ESP-MESH.
- Utilizar la tecnología Bluetooth que integra el dispositivo.
- Utilizar el modo de ahorro de energía del ESP32.

Pliego de condiciones

El proyecto se ha desarrollado con un PC portátil MSI GL63 con sistema operativo Windows 10 Home. El desarrollo software se ha programado desde la plataforma Arduino en la versión 1.8.5 y se han analizado datos con Matlab 2017b.

Presupuesto

Para el desarrollo de la aplicación se han utilizado los siguientes recursos:

Material	Coste
ESP32 original Espressif Systems	14.50€
Sensor BME280	9.50€
Sensor PIR HC-SR501	1.40€
2x Módulo Relé de 4 canales	8.80€
Resistencias 1KOhm	0.05€
Pantalla OLED SSD1306	5.85€
Bombillas Led 4W Lighting Ever (x10)	19.90€
Otros materiales	3€
Total (recursos materiales)	63€

Tabla 7. Recursos materiales de la aplicación.

El presupuesto material podría reducirse considerablemente de cara a replicar el proyecto si se escoge un módulo ESP32 de otro fabricante o si los diferentes productos se adquieren a proveedores como Aliexpress o similares. A continuación, se ha calculado el coste del equipamiento y del software utilizado:

Equipamiento y software	Precio	Uso (meses)	Coste
Ordenador portátil (36 meses) + S.O.	979€	6	163.33€
Microsoft Office 365 (12 meses)	69€	6	34.50€
Software Arduino	0€	--	0€
Software Git GUI	0€	--	0€
Software Matlab (licencia universitaria)	0€	--	0€
Total (equipamiento y software)			197.83€

Tabla 8. Recursos de equipamiento y software.

Posteriormente se calcula el coste de la mano de obra:

Mano de obra	€/hora	Horas	Coste
Ingeniero	20	240	4800€
Total (mano de obra)			4800€

Tabla 9. Coste de la mano de obra.

El presupuesto de ejecución material se calcula sumando los costes anteriores:

Descripción	Coste
Materiales	63€
Equipamiento	197.83€
Mano de obra	4800€
Presupuesto de ejecución material	5060.83€

Tabla 10. Presupuesto de ejecución material.

También hay que tener en cuenta para el presupuesto un margen de gastos no previstos y un margen de beneficio.

Descripción	Coste
Coste de ejecución material	5060.83€
Gastos no previstos (10%)	506.08€
Beneficio (6%)	303.65€
Presupuesto de contrata	5870.56€

Tabla 11. Presupuesto de contrata.

Finalmente, el presupuesto total se calcula añadiendo el impuesto sobre el valor añadido:

Descripción	Coste
Presupuesto de contrata	5870.56€
I.V.A. (21%)	1232.82€
Presupuesto TOTAL	7103.38€

Tabla 12.Presupuesto total.

Anexo: Instalación del entorno Arduino para el ESP32

Tutorial para la instalación del entorno Arduino para el ESP32 en Windows 10.
Para la instalación de Arduino primeramente es necesario descargar el entorno de desarrollo Arduino IDE [29]. Posteriormente habrá que instalar Git GUI[30].
A continuación, entrar dentro de la aplicación Git GUI y seleccionar *Create Existing Repository*

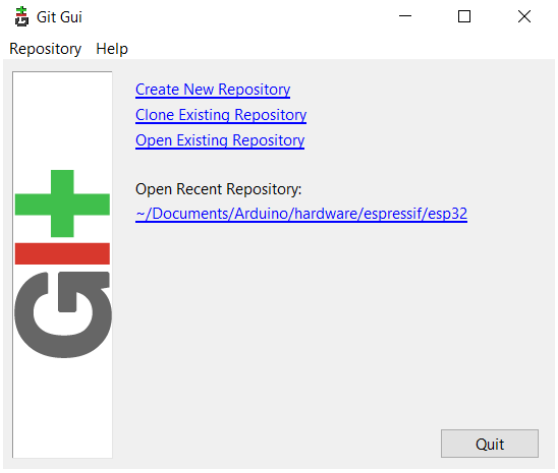

Figura 67. Git GUI.

Posteriormente, se introduce el repositorio de GitHub en el campo *Source* y en *Target* donde se desee colocar el entorno (se creará una carpeta nueva).

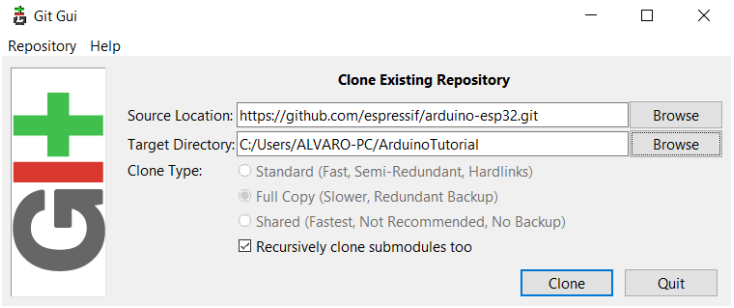

Figura 68. Creación del entorno Arduino.

Seguidamente abrir la aplicación GIT BASH, moverse al directorio creado y ejecutar el código que aparece en la siguiente imagen:

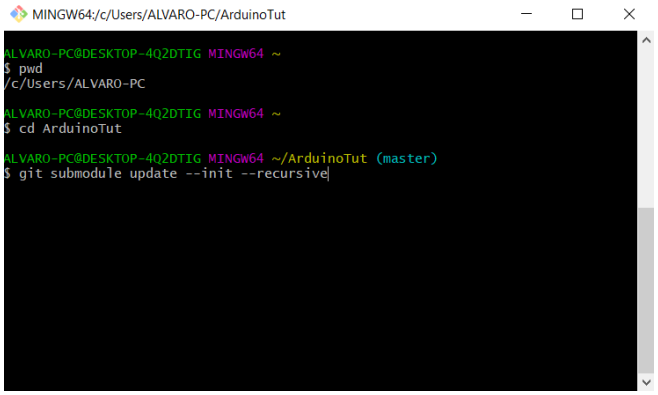

Figura 69. Ejecutar código en GitBash.

El siguiente paso es ejecutar el archivo *get.exe* que se encuentra en el directorio creado.

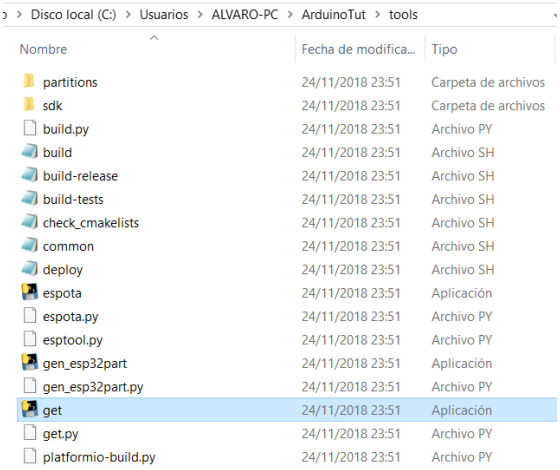

Figura 70. Localización de archivo get.exe

Como se puede ver en la imagen se descargará el repositorio al ejecutar el archivo get.exe

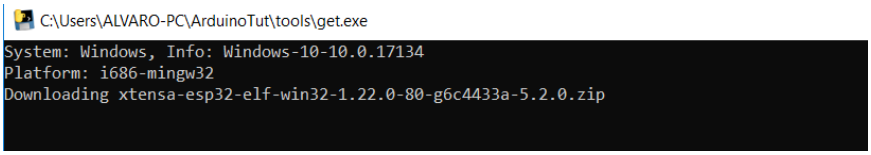

Figura 71. Ejecución del archivo get.exe

Finalmente se abrirá el entorno de desarrollo Arduino IDE y se verificará que se ha instalado correctamente, comprobando que en *Herramientas -> Placa* se encuentran los diferentes modelos de ESP32

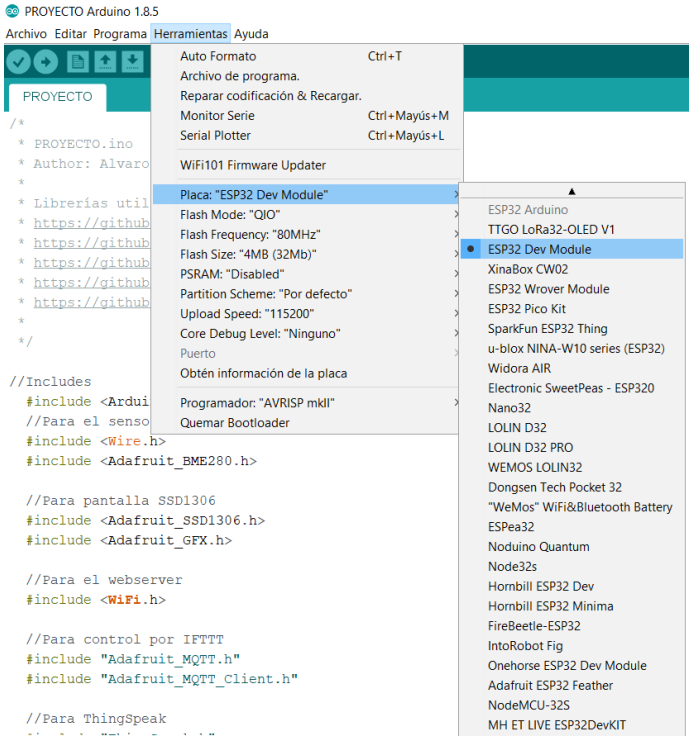

Figura 72. Arduino Core instalado para el ESP32.

Bibliografía

[1] «Datasheet ESP32,» [En línea]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[2] «Technical Reference Manual ESP32,» [En línea]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf.

[3] «ESP32 Modules and Boards,» [En línea]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/hw-reference/modules-and-boards.html.

[4] «The Internet of Things with ESP32,» [En línea]. Available: http://esp32.net.

[5] O. Gonzalez, «Comparativa y análisis completo de los módulos Wifi ESP8266 y ESP32,» [En línea]. Available: https://blog.bricogeek.com/noticias/electronica/comparativa-y-analisis-completo-de-los-modulos-wifi-esp8266-y-esp32/.

[6] «ESP-MESH — ESP-IDF Programming Guide,» [En línea]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/mesh.html.

[7] «ESP32-LyraT DebugDump Forum,» [En línea]. Available: https://debugdump.com/t_946.html.

[8] «Connecting the ESP32-DevKitC and AWS IoT Using Mongoose OS,» [En línea]. Available: https://medium.com/@gomaketeam/connecting-the-esp32-devkitc-and-aws-iot-using-mongoose-os-part-i-11fecd2d86d5.

[9] «Listing of JavaScript engines for embedded systems/IoT applications,» [En línea]. Available: https://acassis.wordpress.com/2017/05/03/listing-of-javascript-engines-for-embedded-systemsiot-applications/.

[10] «Software | Espressif Systems,» [En línea]. Available: https://www.espressif.com/en/products/software.

[11] «Getting Started with MicroPython on ESP32 and ESP8266,» [En línea]. Available: https://randomnerdtutorials.com/getting-started-micropython-esp32-esp8266/.

[12] «ESP8266 (Parte 1) : Programación en Lua y Arduino | B105 lab,» [En línea]. Available: http://elb105.com/esp8266-parte-1-programacion-en-lua-y-arduino/.

[13] «BME280 Digital Barometer,» [En línea]. Available: https://cdn-shop.adafruit.com/datasheets/BST-BME280_DS001-10.pdf.

[14] «HC-SR501,» [En línea]. Available: https://learn.adafruit.com/assets/13829.

[15] «Manual sensor PIR HC-SR501 – CR Electrons,» [En línea]. Available: https://puntoflotante.net/MANUAL-DEL-USUARIO-SENSOR-DE-MOVIMIENTO-PIR-HC-SR501.pdf.

[16] «SSD1306,» [En línea]. Available: https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf.

[17] «MODULO RELÉ 5V 4 canales» [En línea]. Available: https://www.amazon.es/kwmobile-M%C3%B3dulo-rel%C3%A9-Arduino-Canales/dp/B01H2D2RXA/ref=sr_1_1?ie=UTF8&qid=1546194159&sr=8-1.

[18] «Bombillas Lightning Ever,» [En línea]. Available: https://www.amazon.es/gp/product/B00LN0RAUY/ref=oh_aui_detailpage_o02_s01?ie=UTF8&psc=1.

[19] «FreeRTOS task states and state transitions described,» [En línea]. Available: https://www.freertos.org/RTOS-task-states.html.

[20] «MQTT API | Adafruit IO | Adafruit Learning System,» [En línea]. Available: https://learn.adafruit.com/adafruit-io/mqtt-api#.

[21] «ESP32 Pinout Reference: Which GPIO pins should you use?,» [En línea]. Available: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/.

[22] «ESP32 Web Server - Arduino IDE,» [En línea]. Available: https://randomnerdtutorials.com/esp32-web-server-arduino-ide/.

[23] «espressif/arduino-esp32,» GitHub, [En línea]. Available: https://github.com/espressif/arduino-esp32.

[24] «adafruit/Adafruit_BME280_Library,» GitHub, [En línea]. Available: https://github.com/adafruit/Adafruit_BME280_Library.

[25] «espressif/arduino-esp32/tree/master/libraries/WiFi,» GitHub, [En línea]. Available: https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi.

[26] «adafruit/Adafruit_SSD1306,» GitHub, [En línea]. Available: https://github.com/adafruit/Adafruit_SSD1306.

[27] «adafruit/Adafruit_MQTT_Library,» GitHub, [En línea]. Available: https://github.com/adafruit/Adafruit_MQTT_Library.

[28] «mathworks/thingspeak-arduino,» GitHub, [En línea]. Available: https://github.com/mathworks/thingspeak-arduino.

[29] «Arduino - Software,» [En línea]. Available: https://www.arduino.cc/en/Main/Software.

[30] «Git - Downloading Package,» [En línea]. Available: https://git-scm.com/download/win.

[31] «Adafruit IO,» [En línea]. Available: https://io.adafruit.com.

[32] «IFTTT helps your apps and devices work together,» [En línea]. Available: https://ifttt.com.

[33] «ESP32_SENSOR - ThingSpeak IoT,» [En línea]. Available: https://thingspeak.com/channels/574014.

[34] «API Reference — ESP-IDF Programming Guide,» [En línea]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/index.html.

[35] «Get Started — ESP-IDF Programming Guide,» [En línea]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/get-started/index.html.

[36] «IoT Sharing,» [En línea]. Available: http://www.iotsharing.com/search/label/ESP32.

[37] «ESP32 – techtutorialsx,» [En línea]. Available: https://techtutorialsx.com/category/esp32/.

[38] L. Llamas, «ESP8266, la alternativa a Arduino con Wifi,» [En línea]. Available: https://www.luisllamas.es/esp8266/.

[39] N. Kolban, «Kolban´s Book on ESP32,» https://leanpub.com/kolban-ESP32, 2018.

[40] A. S.L., «A FONDO: ¿Qué es IoT (el Internet de las Cosas)?,» [En línea]. Available: https://www.domodesk.com/221-a-fondo-que-es-iot-el-internet-de-las-cosas.html.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR