Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

Universidad de Jaén

Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

DESARROLLO DE UN SISTEMA DE BAJO COSTE PARA CONTROL DE UN INVERNADERO

Alumno: **José Manuel Jiménez Judas**

Tutor: Juan Carlos Cuevas Martínez

Tutor: Antonio Jesús Yuste Delgado

Depto.: Ingeniería de Telecomunicación

Junio, 2024

Resumen

En el presente Trabajo Fin de Grado se ha desarrollado e implementado un sistema de control para un invernadero o vivero automatizado, utilizando plataformas conocidas como **Arduino**, **Android** y **Firebase**. El sistema se puede dividir en tres secciones: **IoT**, **Servidor** y **Aplicación**.

La parte **IoT** se encuentra desplegada físicamente y se pueden diferenciar en nodos sensores y actuadores. Los **nodos sensores** se encargan de la lectura de datos ambientales de la zona asociada y transmitirlos a la parte servidora mediante un nodo central o punto de acceso. Estos dispositivos son **autónomos** y poseen una batería que permite su libre colocación. Los nodos sensores tienen dos configuraciones posibles, una orientada a la medición de temperatura y humedad ambiental y otra orientada a la lectura de la humedad del suelo.

Los **nodos actuadores** se encargan de simular la parte de control de cada una de las variables ambientales utilizadas en la zona. Permiten un modo **manual** y otro **automático**. Este último, actuará dependiendo de varios factores como la franja horaria de activación o el valor objetivo.

La parte del **Servidor** o servidora se encarga de hacer de puente entre la sección IoT y la aplicación. Sus funciones principales son la **gestión**, **almacenamiento** y **procesamiento** de datos.

Para la interacción usuario-sistema, se ha desarrollado una **Aplicación** para terminales móviles Android que se encargará de ofrecer una **interfaz gráfica** amena al usuario donde pueda ver las zonas que posee el sistema, e interactuar con los nodos sensores y actuadores de forma **remota**.

Abstract

In this Thesis, a control system has been developed and implemented for an automated greenhouse or nursery, using well-known platforms such as **Arduino**, **Android**, and **Firebase**. The system can be divided into three sections: **IoT**, **Server**, and **Application**.

The **IoT** part is physically deployed and can be differentiated into sensor nodes and actuators. **Sensor nodes** are responsible for reading environmental data from the associated area and transmitting it to the server part through a central node or access point. These devices are **autonomous** and have a battery that allows for flexible

placement. Sensor nodes have two possible configurations: one focused on measuring ambient temperature and humidity, and another focused on soil moisture reading.

Actuator nodes simulate control of each of the environmental variables used in the area. They support both **manual** and **automatic** modes, with the latter operating based on factors such as activation time slots or target values.

The **Server** part acts as a bridge between the IoT section and the application. It's main functions include **data management**, **storage**, and **processing**.

For user-system interaction, an Android mobile **application** has been developed to provide a user-friendly **graphical interface** where users can view system zones an interact with sensor and actuator nodes **remotely**.

Índice

Índice de abreviaturas

« API: Application Programming Interface
« AP: Access Point
« BaaS: Backend as a Service
« CLI: Command Line Interface
« CoAP: Constrained Application Protocol
« CRUD: Create, Read, Update, Delete
« Deep-Sleep: Sueño profundo
« Gateway: Puerta de enlace
« HTTP: Hypertext Transfer Protocol
« HTTPs: Hypertext Transfer Protocol Secure
« I2C: Inter-Integrated Circuit
« IDE: Integrated Development Environment
« IaaS: Infrastructure as a Service
« ID: Identificación
« Li-Ion: Lithium-Ion
« LiFePo4: Litio-ferrofosfato
« LiPo: Polimero de lítio
« LOAD: Cargar
« MQTT: Message Queuing Telemetry Transport
« NiCd: Níquel-cadmio
« NiMH: Níquel-metalhidruro
« NoSQL: Not only SQL
« PaaS: Platform as a Service
« RAM: Random Access Memory
« RESET: Reiniciar
« RTC: Real-Time Clock
« SQL: Structured Query Language
« SPI: Serial Peripheral Interface
« SPIFFS: SPI Flash File System
« SSL: Secure Sockets Layer
« SSID: Service Set Identifier
« SO: Sistema Operativo
« TLS: Transport Layer Security

« UTC: Coordinated Universal Time
« WAN: Wide Area Network
« Wake-UP: Despertar
« WiFi: Wireless Fidelity
« Handshake: Saludo
« listeners: Funciones escucha
« triggers: Disparadores

Índice de figuras

Índice de tablas

1 INTRODUCCIÓN

En el presente Trabajo Fin de Grado (TFG) se ha creado un sistema de control y automatización de invernaderos. Se han utilizado dispositivos Arduino de bajo consumo a los que se han conectado sensores para obtener datos ambientales como temperatura, humedad y humedad de suelo.

En el presente la memoria, se llama **nodo sensor**, al conjunto de componentes de sensor, placa de desarrollo, batería y otra serie de componentes. Estos nodos sensores, tras autenticarse con el servidor, envían los datos obtenidos mediante un punto de acceso, utilizando comunicación WiFi. Cada nodo sensor tiene un identificador único.

Para simular el control de los distintos sistemas de actuación se ha utilizado un dispositivo Arduino y tres leds por cada variable ambiental. Al conjunto de dispositivo y leds se llama **nodo actuador**.

Cada uno de estos nodos está localizado en una **zona**. A su vez, cada **zona** está identificada con un identificador único.

Paralelamente se ha desarrollado una **aplicación móvil** para dispositivos Android, la cual permite ver el tipo de variables ambientales que se recogen en una zona y controlar el funcionamiento de los nodos actuadores y sensores interactuando con el servidor.

La parte del **servidor**, es la encargada de recibir los datos de los nodos sensores, procesarla para obtener las medias de cada variable ambiental y almacenarla en dos bases de datos. Una de ellos será utilizada para almacenar los datos que cambian con frecuencia y en la otra se guardaran históricos y otros datos.

El servidor permite informar de cambios en estas bases de datos, informando a los nodos y aplicación móvil de dichas actualizaciones. La notificación de estos cambios se manejan mediante una conexión persistente a través de ***WebSockets*** (1), utilizando HTTP/2 (2).

1.1 Objetivos

El presente TFG tiene como objetivo principal el uso de dispositivos de bajo coste que se puedan conectar a Internet para controlar distintos parámetros de un invernadero. Se realizará un desarrollo basado en la **Internet de las Cosas (IoT)** para la visualización de los distintos parámetros, recogidos de los sensores. Para ello se

usará una plataforma digital que tenga la posibilidad de guardar estos datos de forma gratuita o a coste mínimo. Una vez obtenidos los datos, se desarrollará también una aplicación para poder interactuar con otros dispositivos en el invernadero, bien de forma automática, o a través de la aplicación desarrollada en el TFG.

Los nodos del sistema serán sensores y actuadores de bajo coste y que puedan comunicarse con un nodo central para la recopilación/activación de estos a través de Internet. Como mínimo se visualizarán datos de temperatura y humedad, y se actuará sobre luces y ventiladores. La aplicación accesible vía Internet debe tener la posibilidad de activar/desactivar los actuadores de forma individual, además de los considerados como automáticos.

1.2 Estado del Arte

Las tecnologías de **Internet de las cosas** (**IoT**) han revolucionado la forma en que se recopilan y gestionan datos ambientales, como temperatura, humedad, calidad del aire, entre otros. Estos avances han permitido una monitorización más precisa y en tiempo real del entorno, lo que a su vez ha impulsado el desarrollo de sistemas de control más eficientes y la creación de servicios personalizados para los usuarios.

Empresas como **John Deere** (3) han desarrollado soluciones IoT para la agricultura inteligente que incluyen sensores de suelo, clima y cultivos, integrados en maquinaria agrícola. Estos sistemas recopilan datos ambientales en tiempo real y permiten controlar y optimizar el riego en función de la climatología y las necesidades hídricas de los cultivos.

En el ámbito de sensores se encuentran empresas como **Sensirion** y **Bosch Sensortec**, cuyos productos son ampliamente utilizados en aplicaciones IoT para el monitoreo ambiental en interiores y exteriores. **Autogrow** (4), una empresa que ofrece sistemas automatizados de monitoreo, riego y control de nutrientes, utiliza sensores de la empresa **Sensiron** para obtener datos de temperatura y humedad.

Gracias a plataformas como **AWS**, **Azure** y **Google Cloud**, es posible almacenar, procesar y analizar grandes volúmenes de datos ambientales de manera eficientes y en tiempo real. De las funcionalidades relacionadas con IoT destacamos los *broker* MQTT para la comunicación de los dispositivos IoT y la nube.

Un ejemplo es la empresa **CropX** (5) cuyo objetivo principal es la optimización del uso del agua y mejorar el rendimiento de los cultivos. Sus sensores IoT se conectan a la nube a través de AWS IoT Core, donde los datos son almacenados y procesados

con modelos predictivos de *machine learning*, ayudando a los agricultores a tomar decisiones eficientes y sostenibles basadas en datos precisos.

2 DISEÑO E IMPLEMENTACIÓN

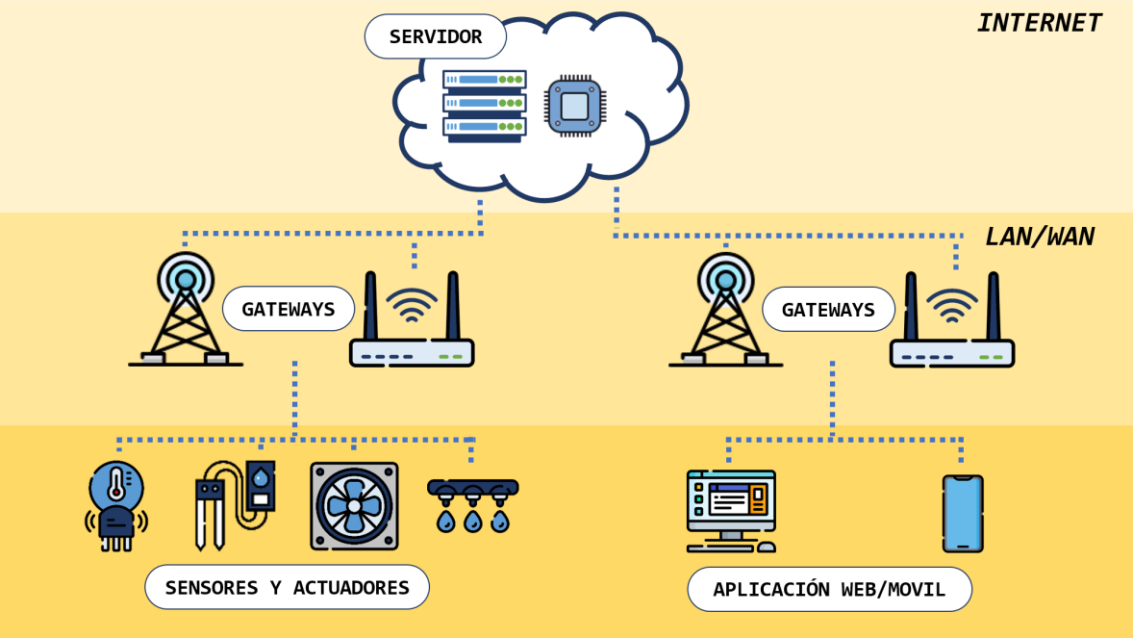


Ilustración 1 – Esquema del sistema.

La estructura del sistema se ha organizado en tres partes: Dispositivos IoT, Servidor y Aplicación web o móvil.

Los dispositivos IoT están formados por nodos sensores, nodos actuadores y un nodo central o punto de acceso. Los nodos sensores deben ser autónomos, con cierta autonomía, de bajo consumo y, como mínimo, medir una variable ambiental. Los nodos actuadores deben simular si el sistema de control (temperatura, humedad o riego) se encuentra en modo manual o automático y sus respectivos estados. Los nodos se comunicarán con el servidor mediante un nodo central o punto de acceso.

El servidor almacena los datos recibidos por los nodos sensores, calcula la media de cada variable ambiental e informa a los nodos y a la aplicación de los cambios.

La aplicación web/móvil permite controlar el tiempo de reposo de los nodos sensores y el funcionamiento de los nodos actuadores.

2.1 Elección de Tecnologías

En este apartado se examinan diferentes dispositivos, protocolos y tecnologías con el fin de seleccionar la opción que mejor se ajuste a los objetivos del presente TFG.

Para el diseño e implementación de los nodos IoT, se ha optado por el entorno de desarrollo de Arduino, por su bajo coste, una amplia gama de librerías y una

documentación extensa. Esta decisión se tendrá en cuenta en la selección de las diferentes tecnologías a emplear.

2.1.1 *COMUNICACIÓN INALÁMBRICA SENSOR-SERVIDOR*

Se va a hacer uso de una comunicación inalámbrica entre los nodos IoT (sensores y actuadores) y el nodo central, para transmitir los datos de los distintos parámetros ambientales al servicio en la nube, garantizando una cobertura amplia.

2.1.1.1 *WiFi 2.4GHz*

Tecnología de alta velocidad para redes locales que permite la conexión de dispositivos a internet.

Tabla 1 – Características de la tecnología WiFi 2.4GHz.

Alcance	90 metros en exteriores
Coste energético	Medio
Coste	Bajo
Velocidad de transferencia	600Mbps en condiciones ideales

2.1.1.2 *Bluetooth*

Estándar de comunicación para la transmisión de datos a corta distancia entre dispositivos.

Tabla 2 – Características de la tecnología Bluetooth.

Alcance	Clase 4 (0,5m) – Clase 1 (100m)
Coste energético	Bajo (0,5mW) – Alto (100mW)
Coste	Bajo
Velocidad de transferencia	Entre 1 Mbps y 3 Mbps

2.1.1.3 *ZigBee*

Protocolo de comunicación de bajo consumo diseñado para aplicaciones de automatización y redes de sensores.

Tabla 3 – Características de la tecnología ZigBee

Alcance	10 a 100 metros
Coste energético	Bajo
Coste	Bajo
Velocidad de transferencia	Entre 20 Kbps y 250 Kbps

2.1.1.4 *LoRaWAN*

Tecnología de red de área amplia de baja potencia que permite la comunicación de dispositivos IoT a largas distancias.

Tabla 4 – Características de la tecnología LoRaWAN.

Alcance	Hasta 20 Km en campo abierto
Coste energético	Bajo
Coste	Medio
Velocidad de transferencia	Entre 0,3 y 50 Kbps

2.1.1.5 *SigFox*

Red de comunicación de baja potencia y largo alcance optimizada para dispositivos IoT que envían pequeñas cantidades de datos.

Tabla 5 - Características de la tecnología SigFox.

Alcance	1,5 Km (Zonas Urbanas) – 20 Km (Campo abierto)
Coste energético	Bajo
Coste	Bajo
Velocidad de transferencia	10 a 1.000 bps

2.1.1.6 *Red de Telefonía Móvil*

Infraestructura de comunicación que permite la transmisión de voz y datos a través de redes celulares.

Tabla 6 - Características de la red de telefonía móvil.

Alcance	Varios Km
Coste energético	Alto
Coste	Alto
Velocidad de transferencia	100 Mbps (4G LTE) – 1Gbps (5G)

2.1.1.7 *Elección de la comunicación inalámbrica*

La red de telefonía móvil es la primera en ser descartada por su alto coste, seguido del Bluetooth por su alto consumo energético. LoRaWAN y SigFox, aunque ofrecen un bajo consumo de energía y largo alcance, dependen de la disponibilidad de cobertura a través de un Gateway LoRa cercano o una Antena SigFox, limitando su implementación. ZigBee es un gran candidato por su eficiencia energética, pero se ha descartado porque los módulos ZigBee, como el XBee, suelen ser más caros que los

módulos WiFi. Para conseguir una mayor versatilidad en el desarrollo, se ha optado por **WiFi 2.4GHz** como tecnología de comunicación inalámbrica.

2.1.2 *DISPOSITIVOS IOT*

Una vez elegida la comunicación inalámbrica, se pasará a elegir el dispositivo encargado de recoger la información de los sensores y transmitirlos. Existen dos módulos muy conocidos dentro del entorno de Arduino que tienen el estándar 802.11 b/g/n: **ESP32** y **ESP8266**. Antes de explicar los detalles más importantes de cada uno, en el diseño del sistema se utilizarán placas de desarrollo para facilitar la implementación, ya que los módulos abaratan costes, pero son difíciles de programar e implementar.

2.1.2.1 *ESP32*

Microcontrolador potente y versátil con conectividad WiFi y Bluetooth, ideal para proyectos IoT y aplicaciones embebidas.

Ilustración 2 – Módulo ESP-WROOM-32. Fuente: ESPRESSIF.

Tabla 7 – Características del módulo ESP-32.

Característica	ESP32
Consumo *Deep-Sleep*	<1 mA
Consumo Transmitiendo (WiFi)	180 a 240 mA
Consumo Recibiendo (WiFi)	95 a 100 mA
Memoria Flash	4 MB extensible a 16 MB
Entradas Analógicas	18 Canales 12 bits

2.1.2.2 *ESP8266*

Microcontrolador económico con conectividad WiFi, ampliamente utilizado en proyectos IoT para añadir capacidades de red a dispositivos.

Ilustración 3 – Módulo ESP8266. Fuente: ESPRESSIF.

Tabla 8 – Características del módulo ESP8266.

Característica	ESP8266
Consumo *Deep-Sleep*	< 1 mA
Consumo Transmitiendo (WiFi)	120 a 170 mA
Consumo Recibiendo (WiFi)	50 a 56 mA
Memoria Flash	512 KB a 4 MB
Entradas Analógicas	1 Canal 10 bits

2.1.2.3 *Elección del dispositivo IoT*

Se ha seleccionado el módulo ESP8266 como dispositivo IoT por sus consumos transmitiendo que son menores que el módulo ESP32.

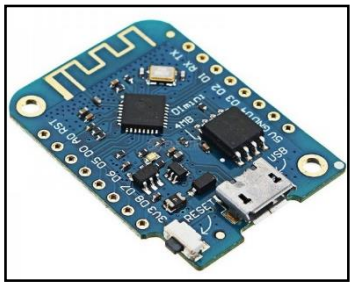


Ilustración 4 – LONIN D1 Mini v3. Fuente: Wemos.

Como placa de desarrollo se utilizará el LONIN D1 Mini v3, que pertenece a la familia D1 de la empresa WEMOS. Estos dispositivos están basados en los módulos ESP8266 y se suelen utilizar en proyectos IoT debido a su bajo coste y por su tamaño compacto. También se utilizará el *shield*[1] de batería para el D1 mini, una placa de

[1] *Shield*: Placa de expansión que se conecta sobre la placa base para añadir funcionalidades específicas.

expansión que permitirá alimentar a la placa mediante una batería y es compatible para todas las versiones.

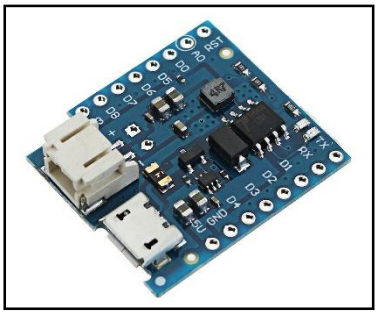


Ilustración 5 – Shield de batería para el LONIN D1 Mini. Fuente: Wemos.

2.1.3 *SENSORES Y MÓDULOS*

Son los encargados de leer los distintos datos ambientales.

2.1.3.1 *Temperatura y Humedad*

2.1.3.1.1 *DHT*

Sensores de temperatura y humedad, donde el DHT11 es básico y adecuado para proyectos sencillos, mientras que el DHT22 ofrece mayor precisión y un rango más amplio.

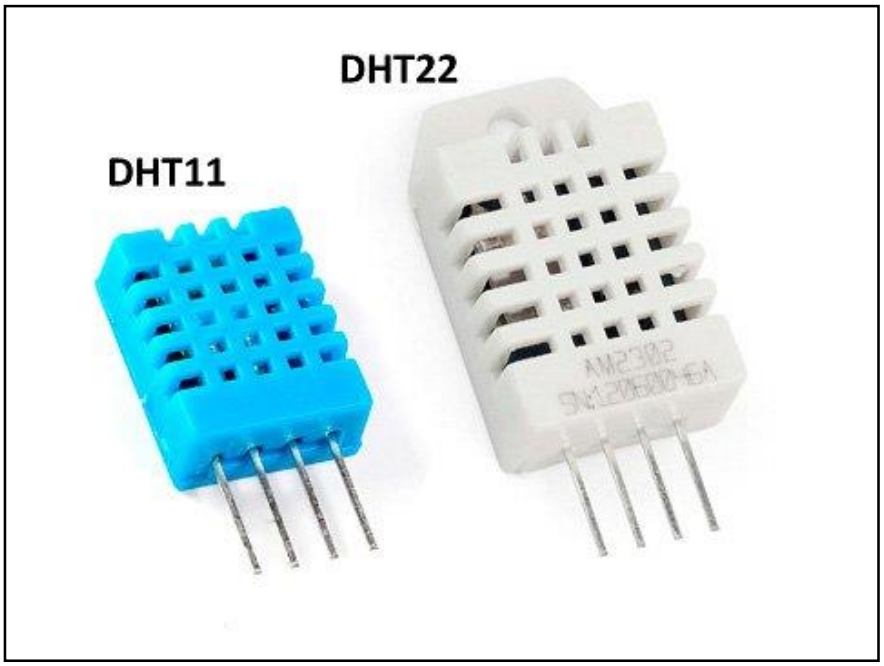


Ilustración 6 – Sensores de temperatura y humedad DHT11 y DHT22. Fuente: ASAIR.

Tabla 9 – Características de los sensores DHT11 y DHT22.

Modelo	DHT11	DHT22
Rango de temperatura	0 a 50 ºC	-40 a 80 ºC
Precisión de temperatura	± 2 ºC	± 0.5 ºC
Rango de humedad	20 a 90 %	0 a 100 %
Precisión de humedad	± 5 %	± 2 %
Consumo	0.3 mA	1.5 mA
Protocolo de comunicación	1 – Wire	1 – Wire

2.1.3.1.2 *BME*

Los sensores BME destacan por medir temperatura, humedad y presión. El BME 680 incluye un sensor de gases para el control de calidad del aire.

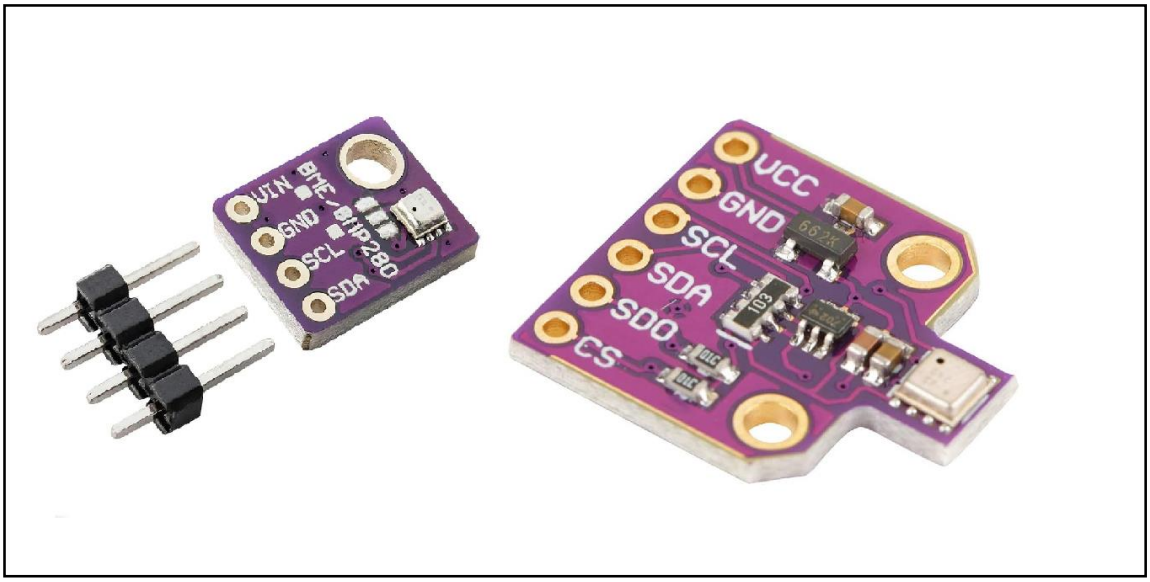


Ilustración 7 – Sensor BME280 (Izquierda) y BME680 (Derecha). Fuente: Bosch Sensortec.

Tabla 10 – Características de los sensores BME280 y BME680.

Modelo	BME280	BME680
Rango de temperatura	-40 a 85 ºC	-40 a 85 ºC
Precisión de temperatura	± 1 ºC	± 1 ºC
Rango de humedad	0 a 100 %	0 a 100 %
Precisión de humedad	± 3 %	± 3 %
Consumo	0.6 mA	3.1 mA
Protocolo de comunicación	I2C, SPI	I2C, SPI

2.1.3.1.3 *SHT*

La familia de SHT tiene una amplia gama de sensores cuya característica principal es su alta precisión. Por simplificación, se mostrarán los más relevantes:

Ilustración 8 – Generación SHT3x (Izquierda) y generación SHT4x (Derecha). Fuente: Sensirion AG.

Tabla 11 – Características de los distintos sensores SHT3x y SHT4x.

Modelo	SHT30	SHT31	SHT35	SHT40
Rango de temperatura	-40 a 125ºC	-40 a 125ºC	-40 a 125ºC	-40 a 125ºC
Precisión de temperatura	± 0.3 ºC	± 0.3 ºC	± 0.2 ºC	± 0.1 ºC
Rango de humedad	0 a 100 %	0 a 100 %	0 a 100 %	0 a 100 %
Precisión de humedad	± 3 %	± 2 %	± 1.5 %	± 1 %
Consumo	0.8 mA	0.8 mA	0.8 mA	0.32 mA
Protocolo de comunicación	I2C	I2C	I2C	I2C

2.1.3.1.4 *Elección de sensor de Temperatura y Humedad*

Se ha elegido el sensor **SHT40** como sensor de temperatura y humedad por su bajo consumo y gran precisión. El BME680 pese a incorporar dos variables ambientales (presión y gases), su precio triplica al SHT40. Y, por último, se ha descartado el DHT22 ya que su coste no es diferenciador y el consumo es superior.

2.1.3.2 *Humedad de Suelo*

Para la medición de la humedad del suelo se han utilizado sensores capacitivos y no sensores resistivos o volumétricos. Esto se debe a que los sensores resistivos son susceptibles a la corrosión y los sensores volumétricos son muy costosos y requieren una calibración compleja.

2.1.3.2.1 *Sensor Capacitivo*

El sensor capacitivo que se va a utilizar es el **Capacitive Soil Moisture Sensor v2**. No se han elegido opciones más precisas ya que la ventana de trabajo es bastante amplia y el coste de los sensores se disparan.

Ilustración 9 – Sensor capacitivo de humedad de suelo. Fuente: DFRobot.

Para controlar el instante temporal en el cual se toman las mediciones, se necesitarán módulos RTC (*Real Time Clock*). Se utilizará el sistema de Tiempo Unix o POSIX donde el tiempo se representa como la cantidad de segundos que han transcurrido desde la medianoche UTC del 1 de enero de 1970. Este sistema permite que los dispositivos y el servidor se encuentren sincronizados, independientemente de la zona horaria en la que se encuentren. Como módulo RTC se utilizará el RTC3231, que se caracteriza por su bajo consumo y alta precisión.

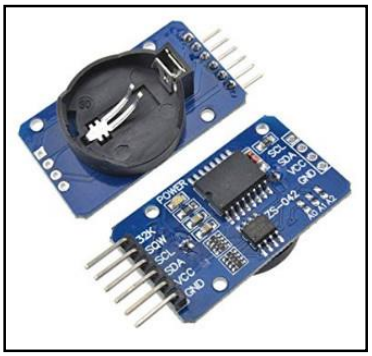


Ilustración 10 – Módulo RTC3231. Fuente: Analog Devices, Inc.

2.1.4 *Protocolo de Comunicación*

Será el encargado de encapsular los datos que se transmitan en la red de dispositivos IoT asegurando la comunicación.

2.1.4.1 *MQTT*

MQTT (*Message Queuing Telemetry Transport*) (6) es un protocolo de mensajería basado en estándares. Es eficiente en el uso de la red y tiene una baja latencia. Se basa en un modelo publicación/suscripción, donde los clientes (en este caso los nodos IoT) pueden publicar mensajes sobre un tema (ej. Temperatura) o suscribirse a uno (ej. Tiempo de *Deep Sleep*). Se necesita de un Broker[2] MQTT que gestione los mensajes entre los clientes.

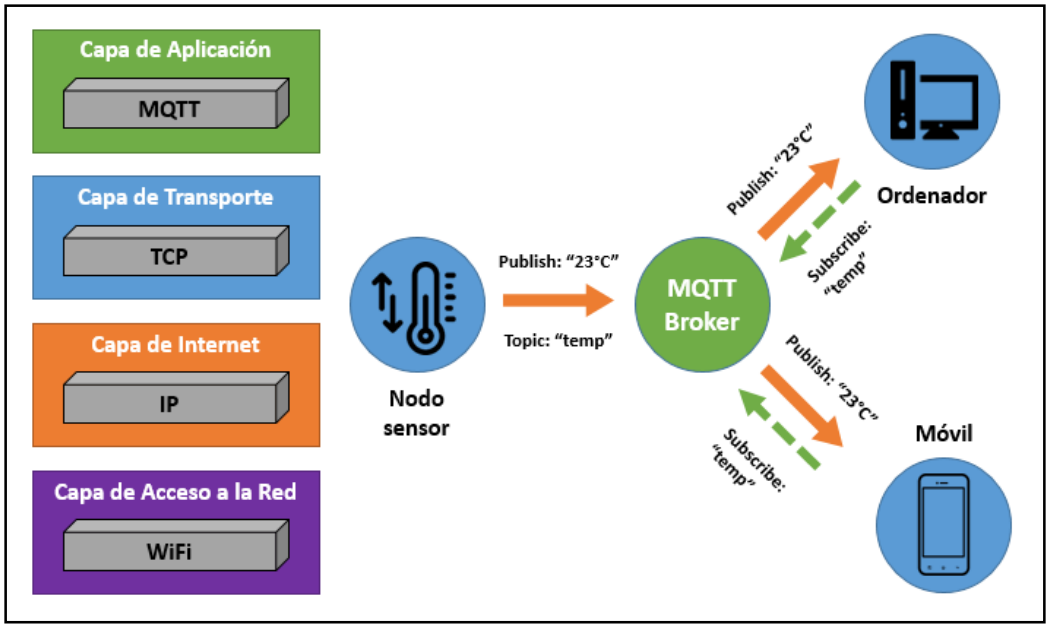


Ilustración 11 – Modelo TCP/IP que utiliza MQTT y flujo de información entre clientes y broker.

2.1.4.2 *HTTPS*

HTTPS (*HyperText Transfer Protocol Secure*) (7) significa que se emplea, en el protocolo HTTP (8), una capa de seguridad entre las capas de aplicación y transporte. Utiliza el protocolo TLS (*Transport Layer Security*) (9) para establecer una conexión segura entre el cliente y servidor, protegiendo la información a través del cifrado de los datos y la comprobación de la integridad de los mismos. Una vez establecida la

[2] Broker: servidor que funciona a modo de intermediario para facilitar la comunicación entre dispositivos IoT mediante la recepción, filtrado y distribución de mensajes publicados en temas específicos.

conexión, el cliente puede enviar peticiones con los métodos que se necesiten con el protocolo HTTP (GET, POST, etc.) con el servidor.

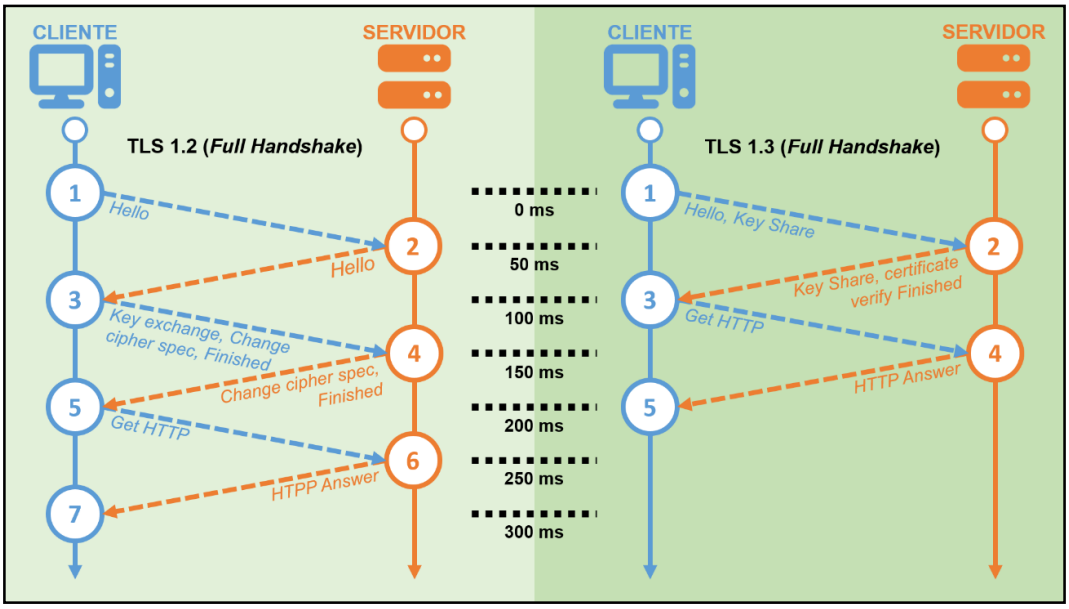


Ilustración 12 – Comparativa del establecimiento de conexión TLS 1.2 vs TLS 1.3.

2.1.4.3 *CoAP*

CoAP (*Constrained Application Protocol*) (10) es un protocolo cuyo funcionamiento es similar a HTTP ya que sigue un modelo RESTful. A diferencia de HTTP, CoAP está orientado a dispositivos con bajos recursos, por lo que su diseño consiste en ser simple y eficiente. Para conseguirlo, CoAP se ejecuta sobre UDP. Si se quiere interconectar un ecosistema que utiliza CoAP con Internet u otro ecosistema que no utilice CoAP, se necesitará un Proxy que haga de puente o traductor.

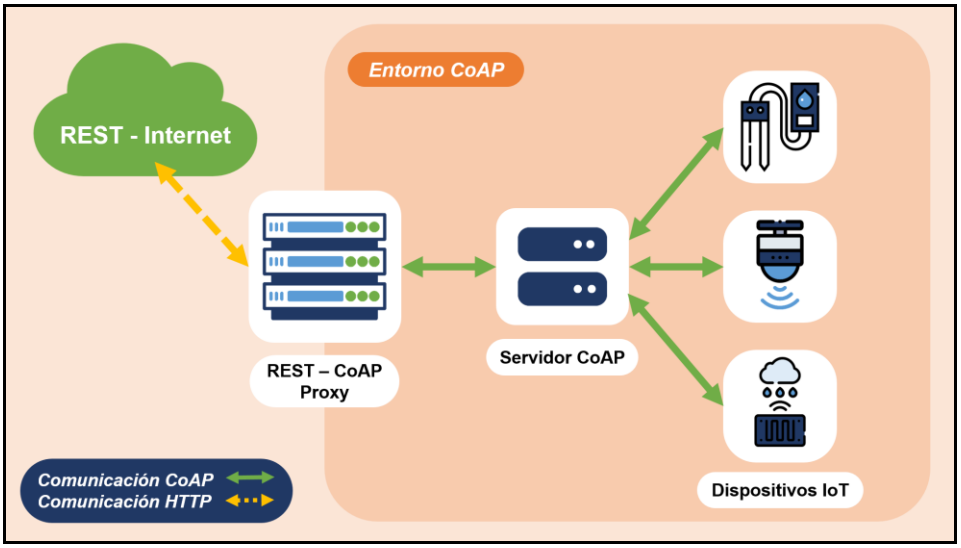


Ilustración 13 – Esquema de comunicación entre un ecosistema con CoAP y sin él.

2.1.4.4 *Elección del protocolo de comunicación*

Se ha elegido **HTTP sobre TLS (HTTPS)** como protocolo de comunicación porque CoAP necesita de un "traductor" para poder comunicarse con la parte servidora del sistema, dependiendo de un dispositivo o *proxy*, encareciendo los costes. Por otra parte, MQTT se ha descartado porque, como se verá en el apartado **Servidor**, Firebase carece de un Broker MQTT, por lo que se necesita un "traductor" entre el Broker y Firebase.

Otro motivo a favor del protocolo elegido es que los datos son sensibles a su manipulación y HTTPS ofrece cifrado a través de TLS, aumentando la seguridad en el ámbito IoT, aunque no se encuentre dentro de los objetivos que se desarrollan en este TFG. Se utilizará cifrado TLS v1.2 que es la versión que soportan las placas de desarrollo seleccionadas.

2.1.5 *SERVIDOR*

Será la parte encargada de almacenar, procesar y gestionar los datos que se generen en el sistema y de la cohesión y correcto funcionamiento entre las distintas partes del sistema.

2.1.5.1 *AWS*

Amazon Web Service (AWS), es una plataforma que ofrece servicios en la nube de computación, almacenamiento, bases de datos entre otros servicios. Ofrece soluciones escalables y un plan gratuito con más de 60 productos para experimentar sin incurrir en costes.

2.1.5.2 *AZURE*

Microsoft Azure es una plataforma de computación en la nube que ofrece servicios similares AWS. También tienen un plan gratuito con más de 55 productos con límites de uso mensuales.

2.1.5.3 *FIREBASE*

Firebase, desarrollada por Google, es una plataforma orientada al desarrollo de aplicaciones web y móviles. Soporta servicios de *backend* y de plataforma como servicio (hosting, …). Ofrecen un plan gratuito, y otro de prepago, cuyos costes están vinculados al uso tras superar un límite mensual.

2.1.5.4 *Elección del servidor*

Tanto AWS como AZURE ofrecen servicios bajo el modelo de distribución de Infraestructura como Servicio (IaaS) y Plataforma como servicio (PaaS). En el caso de Firebase, se enfoca en el modelo de PaaS con componentes considerados *Backend* como Servicio (BaaS). Para el desarrollo de la parte servidora se ha elegido la plataforma Firebase porque las soluciones que ofrecen están orientadas al desarrollo de aplicaciones Web o móviles y se cuenta con experiencia previa.

2.1.6 *APLICACIÓN*

Es la parte del sistema que permite la interacción del usuario con el sistema en su conjunto mediante una interfaz gráfica intuitiva.

2.1.6.1 *Aplicación Web*

Para que el usuario pueda interactuar con los actuadores y sensores, una posibilidad es una aplicación Web. Esta opción permite la interacción usuario-dispositivos IoT sin necesidad de tener una app instalada.

2.1.6.2 *Aplicación móvil nativa*

Otra opción es el desarrollo de una aplicación móvil nativa que ofrece ventajas como rendimiento óptimo, mayor seguridad (aprovecha características de seguridad integradas del SO) y permitir un mejor escalado (añadir funcionalidades, mejorar rendimiento, …).

2.1.6.3 *Elección de la aplicación*

Para el desarrollo de la aplicación se ha elegido una **aplicación móvil nativa** ya que ofrece un rendimiento superior, permite el acceso completo a funcionalidades del dispositivo (GPS, sensores, …) y una seguridad superior.

Se va a realizar la aplicación para dispositivos **Android** en lugar de iOS, porque durante el grado se forma en este tipo de aplicaciones y se cuenta con más experiencia. Como entorno de desarrollo se ha elegido **Android Studio** por su integración con Firebase y una documentación extensa.

2.1.7 *ACTUADORES*

Esta sección se encarga de recopilar las indicaciones del usuario o de operar de manera autónoma, según los datos obtenidos en la parte servidora. Llevará a cabo acciones en el entorno donde estén desplegados, en función de su variable ambiental asociada.

2.1.7.1 *LEDs*

Para simular los sistemas que regulan los distintos parámetros ambientales se utilizaran luces leds que simularán el apagado y encendido de los sistemas.

2.1.8 *BATERÍAS*

Los dispositivos IoT que se dediquen a medir datos ambientales, llevaran una batería para tener independencia de la red eléctrica. En el mercado existen distintas opciones según la química, capacidad, ciclos de carga, etc. Se van a estudiar las siguientes químicas de batería: LiPo, Li-Ion, LiFePo4, NiMH y NiCd.

2.1.8.1 *Características*

Tabla 12 – Características técnicas de las químicas de batería.

Modelo	LiPo	Li-Ion	LiFePo4	NiMH	NiCd
Densidad (Wh/kg)	Alta (150-200)	Alta (150-200)	Moderada (90-120)	Moderada (60-120)	Baja (45-80)
Ciclos	Moderado (300-500)	Moderado (500-1000)	Alto (2000-3000)	Moderado (500-1000)	Alto (1000-1500)
Coste	Alto	Alto	Moderado a Alto	Bajo a Moderado	Bajo
Tasa de Autodescarga	Baja (2-5% por mes)	Baja (2-3% por mes)	Muy baja (<2% por mes)	Alta (20-30% por mes)	Moderada (10-20% por mes)
Voltaje Nominal	3.7 V	3.6 - 3.7 V	3.2 V	1.2 V	1.2 V

2.1.8.2 *Elección de baterías*

Se han descartado las químicas de batería NiMH, NiCd y LiFePo4 ya que no llegan al voltaje operativo (3.3V – 4.2 V) del *shield* de batería del Wemos D1 mini. Entre las baterías tipo LiPo y las Li-Ion, se ha decidido el uso de las baterías Li-Ion por su número de ciclos. El formato de las mismas será el 18650.

Ilustración 14 – Batería de Litio 18650 utilizada en el TFG. Fuente: ANSMANN AG.

2.2 Esquema General

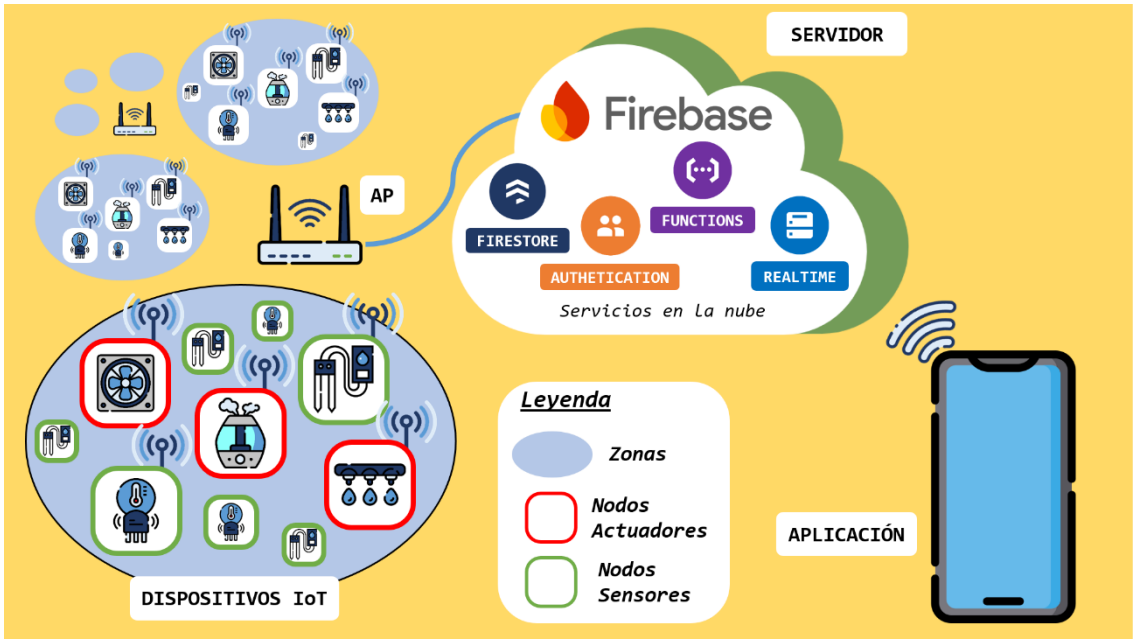


Ilustración 15 – Diseño del sistema.

En este apartado se detallan cada una de las partes que conforman el sistema y como han sido implementadas. Los nodos están desplegados en localizaciones que se llaman **zonas**. Dentro de una zona, se pueden establecer **subzonas** que permitan un control independiente de variables ambientales, separándolas de la gestión general de la zona principal. De esta forma, es posible cultivar diversas cosechas con variados niveles de humedad del suelo, aunque todas requieran condiciones ambientales similares de temperatura y humedad.

Por defecto, cada zona está formada por cuatro nodos sensores (dos de temperatura y humedad y dos de humedad de suelo) y tres nodos actuadores y obtendrán una conexión a internet mediante puntos de acceso. Es posible que una zona tenga unos nodos sensores específicos (por ejemplo, solo humedad de suelo) y otra zona tenga otros, lo que permite implementar diferentes tipos de nodos sensores en cada zona según sea necesario.

Los **nodos sensores** tienen dos configuraciones posibles, una para medir temperatura y humedad (TH) y una segunda para medir la humedad de suelo (RH). Estos nodos sensores se encuentran alimentados por una batería que les brindará una autonomía de al menos dos meses. Cada cierto tiempo, que puede ser variable, los nodos sensores saldrán del modo bajo consumo para obtener medidas del sensor y enviarlas al servidor en su ruta correspondiente. Una vez han enviado los datos, entran

en modo reposo durante un tiempo obtenido de la base de datos. Además, pueden informar de errores mediante unos códigos de error y enviar históricos antiguos en el caso de que se pierda la conexión durante un tiempo.

El **servidor** actuará de puente entre los distintos sistemas y se utilizarán cuatro servicios de Firebase: Realtime, Firestore, Functions y Authentication.

Realtime es una base de datos que es utilizada para almacenar variables que cambian con frecuencia como el tiempo de reposo de los nodos sensores, y otros parámetros como la media de una zona o la configuración de un nodo actuador. Su estructura tiene forma de árbol.

Firestore es otra base de datos que se utiliza para almacenar los históricos de cada una de las variables ambientales y otros datos como la localización de las zonas, descripciones, nombres, etc.

Functions es un servicio que detecta cuando un sensor actualiza información relacionada con una variable ambiental en Realtime. Según la variable que actualice, ejecutará un código que calculará la media con el resto de valores de los nodos sensores de la zona. Esta media es calculada con aquellos datos que no superan la media hora de antigüedad. Una vez calculada, se almacena en su histórico de Firestore y se actualiza el campo en Realtime.

Authentication es el servicio de autenticación de Firebase que restringe la escritura y lectura de las bases de datos a los nodos.

La **aplicación móvil** permite la configuración de cada uno de los nodos actuadores de forma individual y ajustar el tiempo en reposo de los nodos sensores de forma conjunta.

Por último, los **nodos actuadores** indican a Realtime los distintos campos que utilizan (media de la zona, el modo de funcionamiento, hora de inicio, etc.) y este informará cuando dichos valores se actualicen, similar a la subscripción a un *topic* en MQTT. Mediante una serie de leds, indicarán el modo de funcionamiento y los distintos estados posibles.

2.2.1 *NODOS SENSORES*

Los nodos sensores estarán formados por una placa de desarrollo LONIN D1 mini y su placa de expansión de batería, una batería de Ion-Litio 18650, un sensor (SHT40 o humedad de suelo), un módulo RTC3231 y otra serie de componentes que se detallan en el **Presupuesto**. La conexión entre los distintos componentes según su

configuración (temperatura y humedad ambiental o humedad de suelo) se detalla la Ilustración 16.

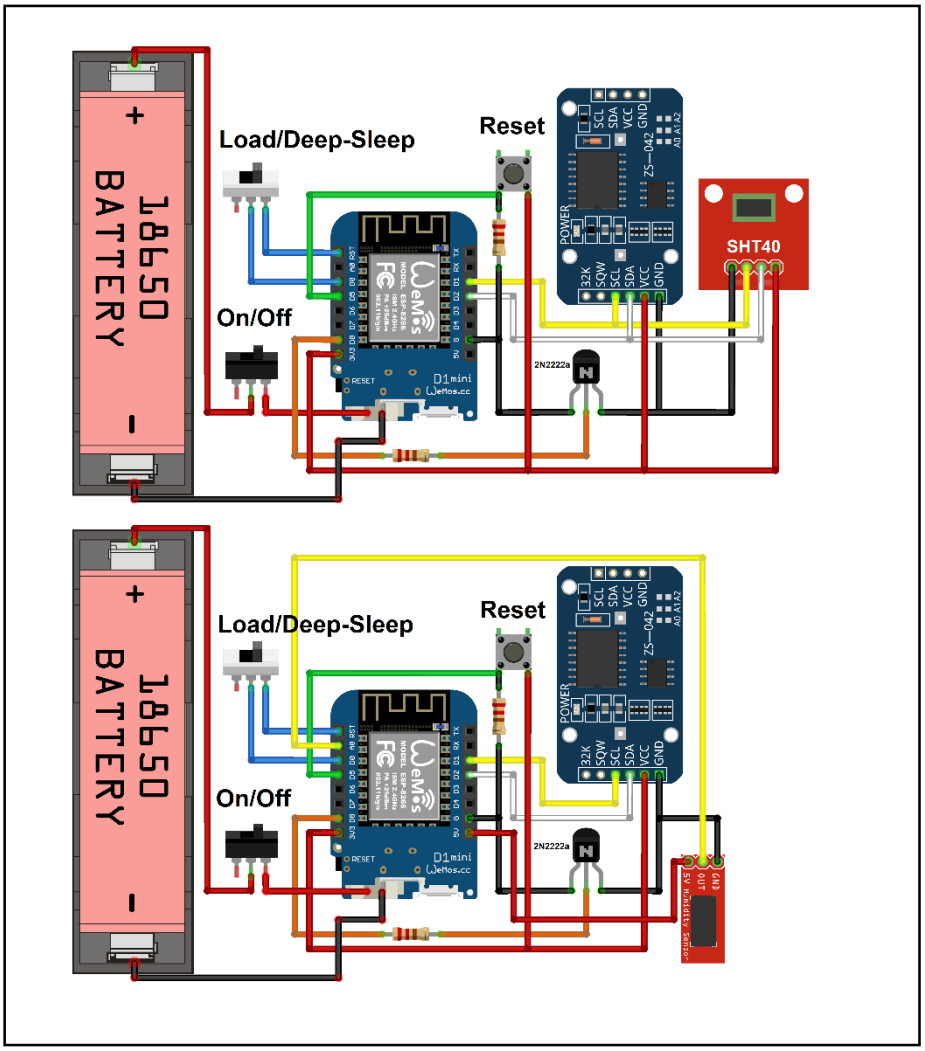


Ilustración 16 – Configuración de temperatura y humedad (superior) y humedad de suelo (inferior).

Los nodos sensores tienen dos interruptores, uno para controlar el encendido y apagado del mismo y otro para activar la carga del software o habilitar el "despertar" (*Wake-Up*) tras entrar en "sueño profundo" o "hibernación" (*Deep-Sleep*). Se ha instalado un transistor BJT NPN para activar/desactivar la alimentación de los módulos (sensor y RTC) para evitar consumos fantasmas.

2.2.1.1 *Configuración*

Para la configuración del nodo sensor, se realizará mediante el entorno de desarrollo Arduino IDE a través del puerto serial del LONIN D1 Mini v3. Se pedirán los siguientes datos:

- Identificador del sensor

- Identificador de zona
- SSID de la red WiFi
- Contraseña de la red WiFi
- Email
- Contraseña

Estos datos, si son correctos, los guarda en un fichero "**config.json**" y se reinicia. En cada reinicio comprueba si existe el fichero y en caso contrario, vuelve a pedir los datos al usuario. Para formatear (*RESET*) completamente el sistema, se ha colocado una resistencia Pull Down de 10 kΩ. Si se pulsa el botón durante un reinicio, se borrarán todos los archivos del SPIFFS (*Serial Peripheral Interface Flash File System*), incluidos los historiales.

2.2.1.2 *Conexión con Firebase*

Para la conexión con Firebase, se ha utilizado la librería Firebase ESP *Client* (11). Esta librería permite la autenticación y acceder a los distintos servicios de Firebase. En este caso, el nodo IoT utiliza solamente el servicio de Realtime Database y de Autenticación.

La autenticación ocurre de la siguiente manera: el nodo sensor establece una conexión segura mediante TLS/SSL y se autentica con las credenciales (email y contraseña). Una vez autenticado, Firebase envía al nodo sensor un *token* de acceso (**JSON Web Token**). Este *token* se incluye en la cabecera de las solicitudes HTTPS como un parámetro de autorización para que Firebase pueda verificar la identidad y los permisos del dispositivo.

La librería ofrece herramientas para realizar las operaciones básicas CRUD (Create, Read, Update y Delete), además de otras funcionalidades como añadir funciones de escucha (*listeners*[3]) o verificar si una ruta existe.

Con el objetivo de ser eficientes en la transmisión de datos, solo se envían y reciben datos una vez por ciclo, a excepción del primer encendido, que el nodo sensor establece la estructura necesaria para su funcionamiento en Realtime Database.

[3] L*isteners*: función que espera y responde a eventos específicos que se producen en la aplicación.

En el proceso de transmisión de datos (recepción y envío), se recibe la duración del tiempo en modo sueño profundo (en minutos) ya que puede cambiar. Después envía en formato JSON, los datos obtenidos con el instante temporal al cual fueron obtenidos. A este JSON se le añaden los datos históricos y un registro con los códigos de error en caso de que existan. Estos datos sobrescriben los que previamente existen en la ruta en Realtime.

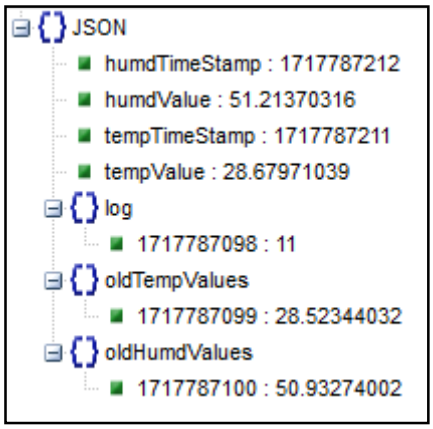


Ilustración 17 – Ejemplo de transmisión de datos en formato JSON.

Es necesario recalcar que antes de conectar el primer nodo sensor de una zona, se necesita crear la zona en la colección de zonas de Firestore.

2.2.1.3 *Históricos*

Con el objetivo de informar sobre errores que se puedan producir durante el proceso de *Wake-Up* y envío de datos, los nodos sensores están diseñados para guardar el código de error junto a su marca temporal en un archivo llamado “errors.txt”. Los códigos de error se recogen en la Tabla 13.

Tabla 13 – Códigos de error de los nodos IoT.

Módulo	Descripción	Tipo de Error	Código
WiFi	Error al conectarse a la red WiFi	Error al iniciar el módulo (Genérico)	10
		Error al conectar con el punto de acceso, no se encuentra el SSID	11
		Tiempo de espera excedido	12
		Contraseña incorrecta	13
Firebase	Error de Autenticación	Error al iniciar sesión en Firebase	20
	Tiempo de espera	Excedido el tiempo máximo de espera	30
	Error de transmisión	Error al enviar datos al servidor	40
		Error al recibir datos del servidor	41
	Datos incorrectos	Tiempo de *Deep-Sleep* erróneo	50
	Error de conexión	Conexión perdida con Firebase	60
Sensor	Error al iniciar el sensor	Sensor no se inicializa o no se reconoce	70
RTC	Error al iniciar el sensor	No se encuentra el módulo RTC o fallo al iniciar	80
		Batería del RTC agotada	81

Una vez se establece conexión con Firebase, el nodo sensor enviará todos los códigos de error si existiesen y, una vez enviados, los borrará del sistema SPIFFS.

En el caso de los parámetros ambientales, si ocurre algún error que no permita su envío a Firebase, se quedarán guardados en su histórico correspondiente. Cuando se produzca una conexión exitosa, se enviarán estos históricos y se borrarán.

2.2.1.4 *Modo sueño profundo*

El modo sueño profundo, o en inglés, *Deep-Sleep*, es un modo en el que se desactivan todos los periféricos y módulos del microcontrolador, a excepción del RTC interno. Dicho reloj se encarga de despertar al dispositivo mediante un pulso en el pin RST. Para que ocurra el “despertar” o *Wake-UP*, el interruptor *LOAD/Deep-Sleep* deberá estar pulsado (interconecta el pin de *Wake-UP* (D0) con el RST).

Al conjunto de procesos que realiza el nodo desde que se “despierta” hasta que entra en modo de sueño profundo se le ha denominado **ciclo**. El diagrama de flujo que siguen estos ciclos se puede encontrar en la Ilustración 18.

Si se produce un error en el programa o se ha completado un ciclo, el nodo entrará en el modo *Deep-Sleep*. Es importante saber que, en el modo de *Deep-Sleep*, los datos almacenados en la memoria RAM no se conservan, por lo que es esencial guardarlos en el sistema SPIFFS. Por último, en este estado, el consumo se reduce significativamente.

2.2.1.5 *Diagrama de Flujo*

En la Ilustración 18 se pueden observar todos los pasos o estados por los que pasa el nodo IoT, desde el *Wake-Up* hasta el siguiente *Deep-Sleep*.

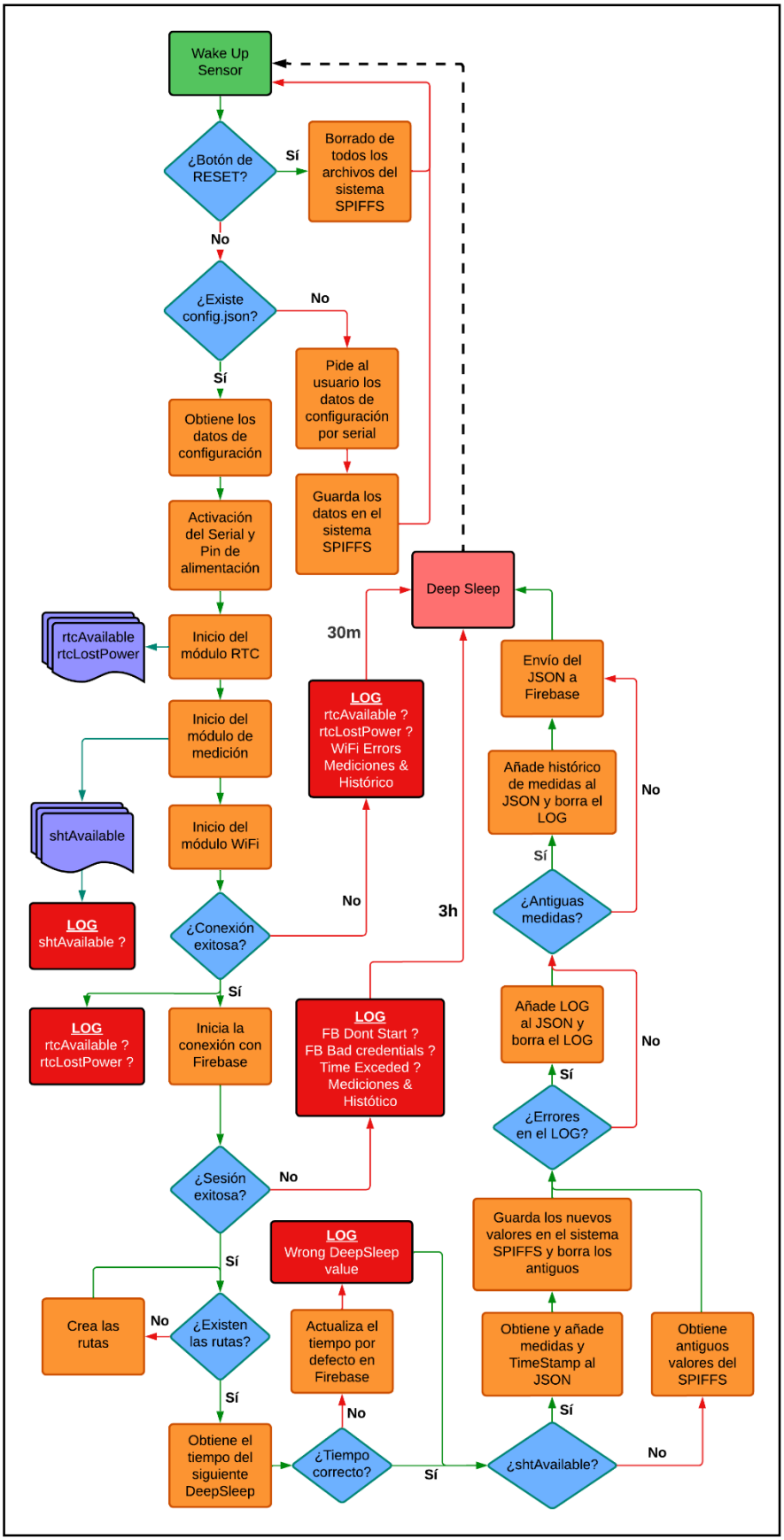


Ilustración 18 – Diagrama de Flujo del nodo IoT.

2.2.1.6 *Funciones*

Para evitar que todas las funciones y código se encuentren en un único fichero, se ha utilizado una práctica llamada organización modular, que consiste en dividir el código en módulos según su función principal. Estas funciones hacen uso de otras funciones y métodos de las librerías implementadas, con el objetivo de realizar ciertas tareas específicas en el funcionamiento del nodo. Las funciones más importantes y su descripción, agrupadas por módulos, se pueden encontrar en la Tabla 14:

Tabla 14 – Funciones utilizadas en el código de los nodos sensores.

Módulo	Función	Descripción
Values	Ninguna	Almacena las constantes utilizadas en el código.
WiFi	`bool initWiFi(String WIFI_SSID, String WIFI_PASS)`	Conecta el nodo con el AP y devuelve el estado de la conexión.
	`void disconnectWiFi()`	Desconecta el nodo del AP y ejecuta el proceso *Deep-Sleep*.
Firebase	`bool initFirebase()`	Establece una conexión con Firebase. En caso contrario, informa del motivo.
	`int getInt(String path)`	Devuelve un valor entero obtenido de una ruta en Firebase.
	`bool sendJSON(String path)`	Envía un fichero JSON a Firebase en una ruta.
	`void verifyPath()`	Verifica que exista el valor de duración del siguiente *Deep-Sleep*.
	`bool verifyDeepSleepValue (int min)`	Verifica si la duración del siguiente *Deep-Sleep* se encuentra entre 1 y un valor.
	`void defaultTime()`	Establece un valor por defecto de duración de *Deep-Sleep* en Firebase.

Tiempo	`bool initRTC()`	Inicia el módulo RTC3231 y verifica si la batería CR2032 se ha agotado.
	`unsigned long getUnixTimeFromServer()`	Devuelve el tiempo actual en formato Unix obtenido por el servidor.
	`unsigned long getUnixTime()`	Devuelve el tiempo actual en formato Unix del RTC si se encuentra disponible o del servidor.
	`uint64_t toMicroseconds(int minutes, int hours = 0)`	Devuelve el valor en microsegundos de un tiempo dado en horas y minutos. Utilizado en el proceso de *Deep-Sleep*.
Log	`void addData(float value, unsigned long time, String fileName, bool isError = false)`	Añade un valor tipo float y su marca temporal a un fichero, indicando si es un código de error o no.
	`void getData(String fileName, String valuePath, String timePath)`	Añade al JSON el ultimo valor y su marca temporal. Utilizado cuando el sensor falla.
	`void sendErrors()`	Añade al JSON todos los códigos de error producidos con su marca temporal.
	`bool eraseAllData()`	Borra todos los archivos del sistema SPIFFS.
	`bool eraseFile(String fileName)`	Borra un fichero dado su nombre.
	`void printError(int id)`	Imprime por serial el código de error y su descripción.
	`void WiFiErrors(int status, unsigned long time)`	Añade un error al fichero de errores según el estado de la conexión.

	`void sendOldValues(String fileName, String jsonName)`	Añade al JSON los históricos almacenados en un fichero.
SHT40	`bool initSTH()`	Inicia el módulo SHT40, establece la precisión y si necesita o no el precalentado. Si no se inicia, añade el error a su fichero.
	`bool getMeasures()`	Actualiza las variables de temperatura y humedad del código principal.
SoilSensor	`float getRHumd()`	Devuelve la humedad del suelo.

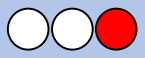
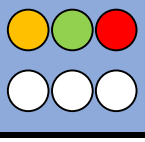
2.2.2 *NODOS ACTUADORES*

Ilustración 19 – Simulación del sistema de control de temperatura, humedad y riego.

Los nodos actuadores son los encargados de interactuar con los distintos sistemas que se implementen. Estos nodos actuadores dependerán del hardware que posea el invernadero y sus limitaciones. Un ejemplo claro es una bomba de agua o una válvula electrónica que permita el paso del agua para regar el suelo. Como la implementación de este hardware no es un objetivo en el TFG, se simularán estos

sistemas con nueve leds que indicarán, según su luminosidad (apagado o encendido), el estado de cada uno. Dichos estados y patrones leds se recogen en la Tabla 15.

Tabla 15 – Posibles estados de los sistemas y sus descripciones.

MODO	ESTADO	DESCRIPCIÓN
MANUAL	Todos los leds encendidos. 	El sistema de control se encuentra encendido.
	Todos los leds apagados. 	El sistema de control se encuentra apagado.
AUTOMÁTICO	LED amarillo encendido. 	El valor ambiental es inferior al valor mínimo.
	LED verde encendido. 	El valor ambiental se encuentra entre el valor mínimo y máximo.
	LED rojo encendido. 	El valor ambiental es superior al valor máximo.
	Cinco parpadeos y apagado. 	El sistema de control se encuentra apagado. Motivo: Fuera de tramo horario.

Para crear cada actuador se han utilizado dos LONIN D1 mini v2 y una LONIN D1 mini pro. Cada actuador se conecta a Firebase con sus credenciales y establece dos funciones *listeners* que escuchan en las siguientes rutas.

- zonas/{**zonaId**}/actuadores/{**parámetro**}
- zonas/{zonaId}/medias/{**media_del_parámetro**}

La **zonaId** indica el identificador de la zona, el **parámetro** indica la variable ambiental (T, H o RH) y la **media_del_parámetro** hace referencia a la media de la zona de esa variable.

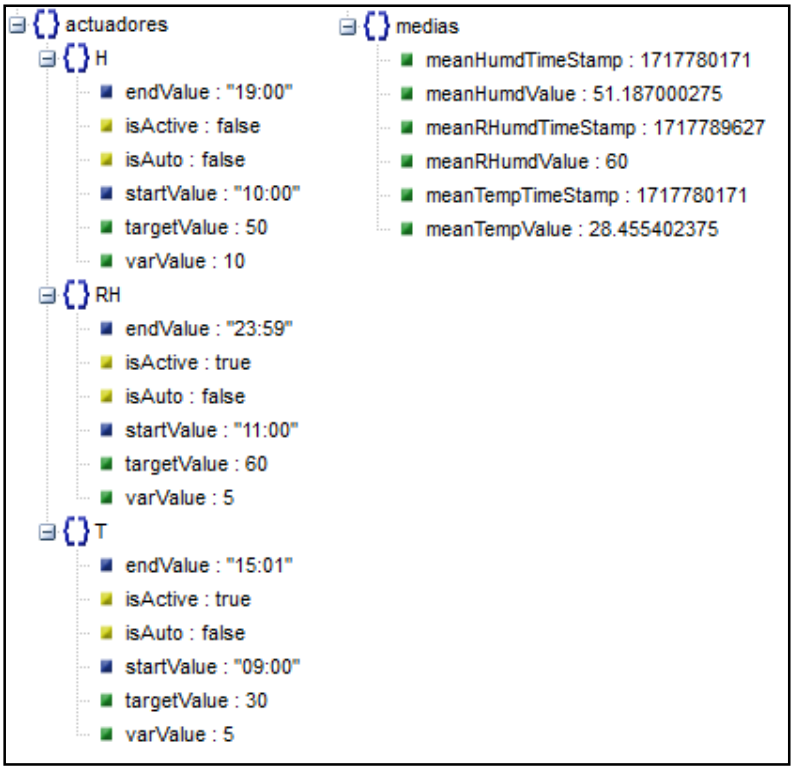


Ilustración 20 – Variables almacenadas en Realtime Database en las rutas …/actuadores y …/medias.

De la primera ruta se obtienen los siguientes parámetros:

- **startValue**: En el modo automático, indica la **hora inicial** en la que trabaja el actuador. Tiene formato **hh:mm** y es de tipo **String** o cadena de caracteres.
- **endValue**: En el modo automático, indica la **hora final** en la que trabaja el actuador. Tiene formato **hh:mm** y es de tipo **String** o cadena de caracteres.
- **isAuto**: Muestra si el actuador trabaja en modo **manual** o **automático**. Tipo **Bolean**.
- **isActive**: En el modo manual, señala si el actuador se encuentra **activo**. Ejemplo: El riego de la zona se encuentra activo. Tipo **Bolean**.
- **targetValue**: Indica el **valor objetivo** de la variable ambiental objeto de estudio. Tipo **Int**.
- **varValue**: Sirve para obtener el **valor mínimo** y **máximo** de la variable ambiental objeto de estudio. Tipo **Int**.

De la segunda ruta se obtienen las siguientes **medias**:

- **meanTempValue**: Media de **temperatura** de la zona. Tipo **Double**.
- **meanHumdValue**: Media de **humedad** de la zona. Tipo **Double**.

- **meanRHumdValue**: Media de **humedad de suelo** de la zona. Tipo **Double**.

Cuando se detecta un cambio en alguno de los parámetros en Realtime Database, Firebase notifica al actuador, indicando el nuevo valor, tipo de dato y la ruta del parámetro. Estos cambios se utilizan para actualizar los **LEDs** e indicar el nuevo estado. En la Ilustración 21 se puede ver la salida del actuador por puerto serial cuando Firebase notifica un cambio.

```
Camino de escucha: /zonas/Z0000001/actuadores/T
Camino de evento: /isAuto
Tipo de dato: boolean
Tipo de evento: put
Valor: false

Camino de escucha: /zonas/Z0000001/actuadores/T
Camino de evento: /isActive
Tipo de dato: boolean
Tipo de evento: put
Valor: false
```

Ilustración 21 – Salida serial del nodo actuador cuando Firebase notifica un cambio.

A diferencia de los nodos sensores, los nodos actuadores no generan su estructura en Realtime. Esta estructura es generada mediante la aplicación móvil accediendo a la sección de configuración de zona (pág. 85) de la variable ambiental que controle el nodo actuador.

2.2.3 *SERVICIO EN LA NUBE SOBRE FIREBASE*

Firebase es el encargado de unir la parte IoT del sistema con el usuario. En los siguientes apartados se detallan cada uno de los servicios utilizados en el diseño y su cometido.

2.2.3.1 *Autenticación*

Firebase Authentication permite que solamente los dispositivos que se encuentren dados de alta puedan acceder al resto de servicios. Como método de acceso se ha habilitado únicamente el **correo electrónico** y **contraseña**, aunque permite otros proveedores.

Cada nodo sensor y actuador tendrá un **email** y una **contraseña**. El correo electrónico en formato **ABNF** es el siguiente:

- **EMAIL** = **TIPO** "." **IDENTIFICADOR** "@" **DOMINIO**
- **TIPO** = "sensores" / "actuadores"

- **IDENTIFICADOR** = **PREFIX** 7DIGIT
- **PREFIX** = “sth” / “srh” / “atz” / “ahz” / “arhz”
- **DOMINIO** = **LABEL** “.” **TLD**
- **LABEL** = 1 * (ALPHA / DIGIT / “-”)
- **TLD** = 2 * (ALPHA)

Como **contraseña** se ha utilizado el propio identificador del nodo sensor/actuador para simplificar las pruebas, aunque es recomendable utilizar contraseñas de 12 o más caracteres con una combinación de letras mayúsculas, minúsculas, números y símbolos.

2.2.3.2 *Realtime Database*

Firebase ofrece dos bases de datos **NoSQL** y una de ellas es **Realtime Database** cuyas características consisten en una estructura de árbol JSON, los datos se sincronizan en tiempo real y se entregan a través de **WebSockets** entre el servidor y los clientes (nodos y usuario).

Esta base de datos será utilizada para almacenar aquellos datos que cambian con frecuencia, como la media de temperatura de una zona o la hora de inicio de la activación del riego. En este TFG se ha decidido no guardar los históricos en **Realtime Database** ya que, Firebase, cobra por espacio almacenado en este servicio, y no era imprescindible para el funcionamiento del servidor.

2.2.3.3 *Firestore Database*

La otra base de datos que se emplea es **Firestore Database**. A diferencia de **Realtime Database**, su estructura se caracteriza por estar formada por colecciones, documentos y consultas, permitiendo una mejor escalabilidad. Se utiliza Firestore para almacenar los históricos de cada nodo sensor y zona y de otros datos de interés como localización de la zona, nombre, capacidad de la batería de un sensor en específico, su dirección MAC, etc.

La estructura de los datos que se almacenan tanto en Realtime como en Firestore Database se especifican en el apartado **Estructura de datos**. Por último, el coste de almacenamiento de datos en Firestore es cinco veces inferior que al de Realtime Database.

2.2.3.4 *Functions*

Este servicio funciona de manera similar a los disparadores o "***triggers***" conocidos en bases de datos SQL. Permite definir en **JavaScript** funciones asociadas a una ruta y a una acción (crear, actualizar y borrar). Cuando se cumplen ambas condiciones, se ejecuta el código dentro de la función.

En este caso, se tienen tres funciones, una por cada variable ambiental, y se encargarán de calcular la media de la zona obteniendo los datos del resto de nodos sensores de la misma zona. Solo se tomarán en cuenta aquellos cuya marca temporal sea inferior a treinta minutos. Además, agregarán los valores a sus respectivos históricos en **Firestore** y actualizará el valor de la media en **Realtime**.

2.2.3.5 *Estructura de datos y formato*

2.2.3.5.1 *Formato de las variables*

El formato de las variables utilizas en Realtime y Firestore en formato ABNF son las siguiente:

- **actuadorID** = "ATZ" / "AHZ" / "ARH" 7DIGIT
- **deepSleepValue** = 1*DIGIT
- **endValue** = 2DIGIT ":" 2DIGIT
- **startValue** = 2DIGIT ":" 2DIGIT
- **isActive** = "true" / "false"
- **isAuto** = "true" / "false"
- **meanHumdTimeStamp** = 10DIGIT
- **meanHumdValue** = 1*DIGIT "." 1*DIGIT
- **meanRHumdTimeStamp** = 10DIGIT
- **meanRHumdValue** = 1*DIGIT "." 1*DIGIT
- **meanTempTimeStamp** = 10DIGIT
- **meanTempValue** = 1*DIGIT "." 1*DIGIT
- **rHumdTimeStamp** = 10DIGIT
- **rHumdValue** = 1*DIGIT "." 1*DIGIT
- **sensorID** = "STH" 7DIGIT / "SRH" 7DIGIT
- **startValue** = 2DIGIT ":" 2DIGIT
- **targetValue** = 1*DIGIT
- **tempTimeStamp** = 10DIGIT
- **tempValue** = 1*DIGIT "." 1*DIGIT

- **varValue** = 1*DIGIT
- **zonaID** = "Z" 7DIGIT

A continuación, se detallará la estructura de los datos de Realtime y Firestore Database.

2.2.3.5.2 *Estructura de Realtime Database*

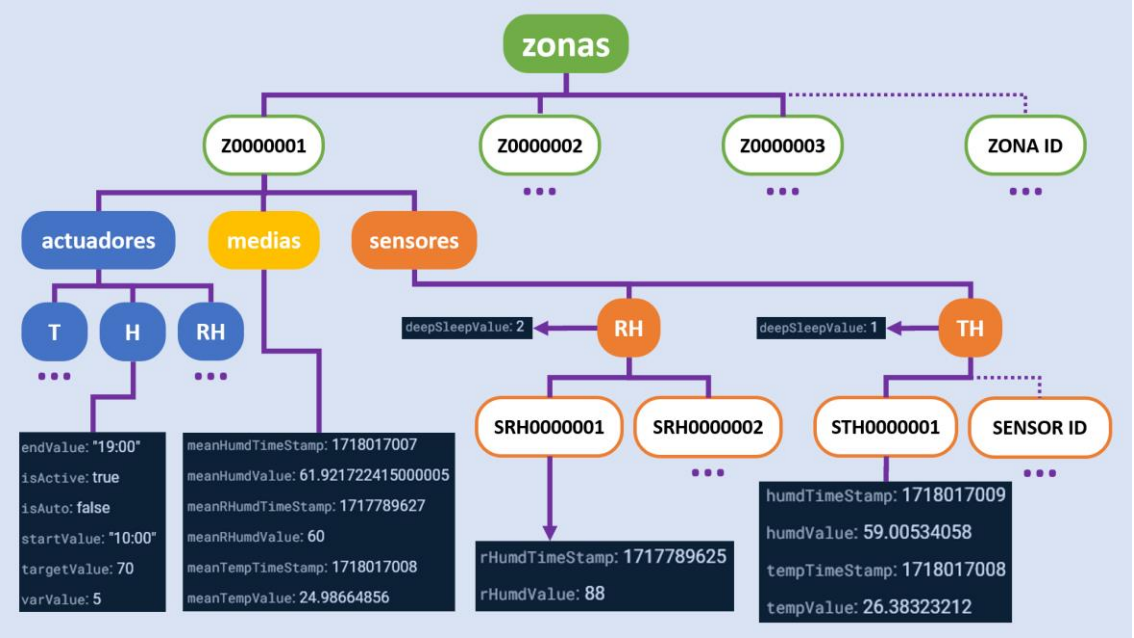


Ilustración 22 – Estructura en árbol JSON de Realtime Database.

Se parte de un nodo raíz llamado "**zonas**", del cual crecen más nodos cuyo nombre será el identificador de zona. Dentro de cada uno se encuentran los nodos de "**medias**", "**sensores**" y "**actuadores**".

En el nodo "**sensores**" (no confundir con los nodos físicos), existen el nodo "**RH**" (Humedad Relativa) y "**TH**" (Temperatura y Humedad), que coincide con el número de configuraciones posibles que tienen los nodos físicos de sensores. En el primer nodo ("**RH**") "nacen" todos los nodos sensores que midan la humedad relativa del suelo y todos ellos tienen un parámetro global llamado "**deepSleepValue**". Este parámetro, como su nombre indica, informa a dichos nodos físicos del tiempo de ***Deep-Sleep***.

El funcionamiento del nodo "**TH**" es similar, solo cambian los parámetros internos de cada uno de los nodos hijos, ya que miden variables ambientales distintas. Todos estos nodos hijos de "**RH**" y "**TH**" se llaman como el identificador del nodo físico.

Partiendo de nuevo de la ruta del identificador de zona, se encuentra el nodo "**medias**", que se encarga de almacenar las medias de cada variable ambiental junto a

su marca temporal. Su estructura es generada, de forma automática, por el servicio Functions cuando se reciben datos de los nodos sensores.

Por último, del nodo "**actuadores**" parten tres nodos hijos llamados "**T**" (temperatura), "**H**" (Humedad) y "**RH**" (Humedad Relativa). En cada uno de ellos se almacenan los parámetros asociados al control de cada uno de los nodos físicos actuadores. Para el caso de la estructura relacionada con los actuadores, es generada mediante la aplicación móvil cuando ya existen datos ambientales (se ha desplegado el primer nodo sensor de la zona).

Es importante recalcar que no es necesario que todas las zonas implementen todos estos nodos. Pueden existir zonas en las que se estudien menos variables ambientales. Tanto los dispositivos, como **Realtime Database**, se han diseñado de tal forma que su estructura se vaya generando de forma dinámica cuando se conecten los nodos sensores o se interactúe con la aplicación móvil. Con solo configurar un nodo sensor y conectarlo a **Firebase**, se generará la estructura necesaria para su correcto funcionamiento, ahorrando almacenamiento y facilitando su despliegue.

2.2.3.5.3 *Estructura de Firestore Database*

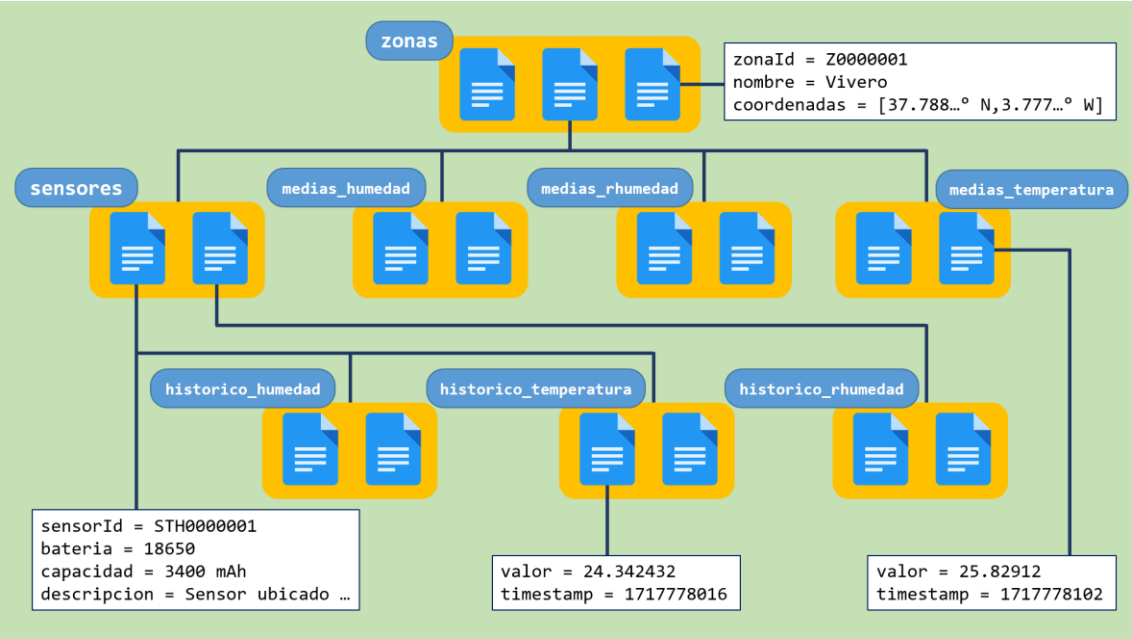


Ilustración 23 – Estructura de Firestore Database.

Firestore, como ya se ha mencionado, utiliza una estructura de colecciones y documentos. Para facilitar la comprensión, tiene una estructura similar a la creada en **Realtime**. Se parte desde el mismo punto, una colección llamada "**zonas**". En esta colección se guardan los documentos que pertenecerán, cada uno, a una zona. Dentro

de cada uno de los documentos se almacenan una serie de campos (coordenadas, nombre y “**zonaId**”) y otras colecciones.

Estas colecciones serán los históricos de la zona y los nodos sensores asociados a dicha zona. Dentro de la colección de históricos de cada parámetro, se encuentran documentos con dos parámetros, valor y la marca de tiempo. Con el uso de consultas, se puede ordenar los documentos por marca temporal descendiente, obteniendo los datos por orden cronológico.

Para la colección de “**sensores**”, se guarda un documento por cada nodo sensor activo en la zona, independientemente de su configuración. En cada uno se guardan datos como el formato de batería que utilizan, la capacidad, descripción y su identificación. Del mismo modo que con las “**zonas**”, cada documento de la colección “**sensores**” guarda una colección de históricos de cada variable ambiental que utilice. De esta forma, se consigue obtener tanto un histórico de la zona, como de los nodos sensores individualmente.

Actualmente, para el despliegue de una zona, los parámetros antes mencionados como descripción, nombre, etc. deben ser añadidos manualmente desde la plataforma de **Firebase**, aunque en las líneas de futuro se contempla la autoconfiguración de **Firestore** como lo hace **Realtime**.

2.2.4 *APLICACIÓN PARA MÓVILES ANDROID*

La aplicación móvil ofrece al usuario la visualización de las **medias ambientales**, el **tiempo meteorológico** de las zonas, **modificar** la duración del *Deep-Sleep* de los nodos sensores y los parámetros relacionados con el modo de funcionamiento de los nodos actuadores. Esta aplicación se ha desarrollado en Kotlin, ya que es interoperable con Java, lo que permite el uso de bibliotecas y dependencias de Java.

2.2.4.1 *Conexión con Firebase*

Para conectar Android Studio con Firebase, se necesitan vincular los servicios *Backend* de Firebase al proyecto, lo que simplificará funciones como autenticación, acceso a las bases de datos (Firestore y Realtime), entre otras.

Este proceso se realiza mediante el asistente de configuración, ubicado en: **Herramientas > Firebase**. Al seleccionarlo, se expandirá una sección en el lateral derecho del IDE y se debe hace clic en el botón “Conectarse a Firebase”. Esto abrirá una ventana en el navegador que llevará a Firebase, donde se iniciará sesión y se seleccionará el proyecto al que se desea vincular la aplicación.

Para añadir las librerías de cada servicio, se abrirá nuevamente el asistente y se pulsará el botón “Agregar” en el producto deseado. En este caso, se añadirán los servicios de Realtime y Firestore Database, ya que el servicio de autenticación no será utilizado. En las Líneas de futuro, se ampliará para que soporte varios usuarios.

Todas las funciones relacionadas con las operaciones básicas **CRUD** y las funciones de escucha, se pueden encontrar en la documentación oficial de Firebase (12).

2.2.4.2 *Actividad de visualización de zonas*

Partiendo de la *activity*[4] del menú principal, el usuario podrá acceder a la *activity* relacionada con cada variable ambiental (temperatura, humedad y riego) pulsando en su icono correspondiente.

Estas *activities* son similares entre ellas y son:

- TemperatureMenuActivity.kt
- HumidityMenuActivity.kt
- GroundHumidityMenuActivity.kt

Cuando se instancia una de estas *activities*, se realiza una petición a Firestore para obtener todas las zonas que existen. Para realizar esta consulta, se crea una instancia de la base de datos Firestore y se ejecuta la petición.

```
val result = fsdb.collection( collectionPath: "zonas").get().await()
```

Ilustración 24 – Consulta de zonas en Firestore en Android Studio.

Por cada zona se realiza una petición extra para obtener la última media de la variable ambiental que se desea ver.

```
val docs = fsdb.collection( collectionPath: "zonas").document(data.id) DocumentReference
    .collection( collectionPath: "medias_humedad") CollectionReference
    .orderBy( field: "timestamp", Query.Direction.DESCENDING).limit(1).get().await()
```

Ilustración 25 – Consulta para obtener la última media de humedad de una zona.

Los datos obtenidos de cada zona son almacenados en objetos de la clase “Zona.kt”. Las variables que almacena cada objeto de esta clase se pueden ver en la Ilustración 26.

[4] A*ctivity*: representa una pantalla o ventana de la interfaz de usuario de una aplicación Android.

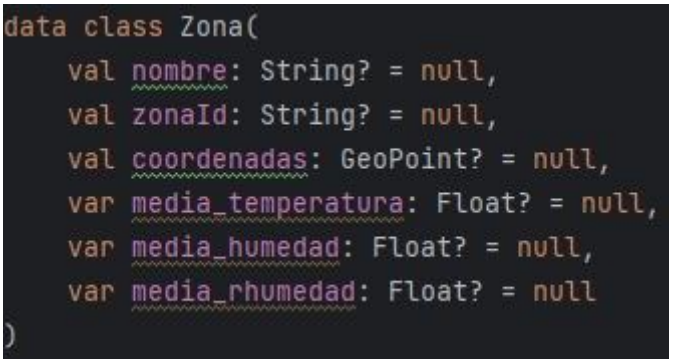


Ilustración 26 – Clase de datos Zona.kt en Android Studio.

Estas zonas se almacenan en un *ArrayList* que se pasará al adaptador del *recyclerview*[5]. El adaptador se encarga de representar cada zona con su media ambiental, añade la localización al botón que redirige a Google Maps, muestra el nombre e identificador y obtiene la meteorología. La obtención de la meteorología se detalla en el apartado **Conexión con la API del tiempo**.

El *recyclerview* tiene asociado un buscador que permite filtrar las zonas por nombre y un botón que oculta o hace visibles aquellas zonas que no tienen medias de la variable ambiental.

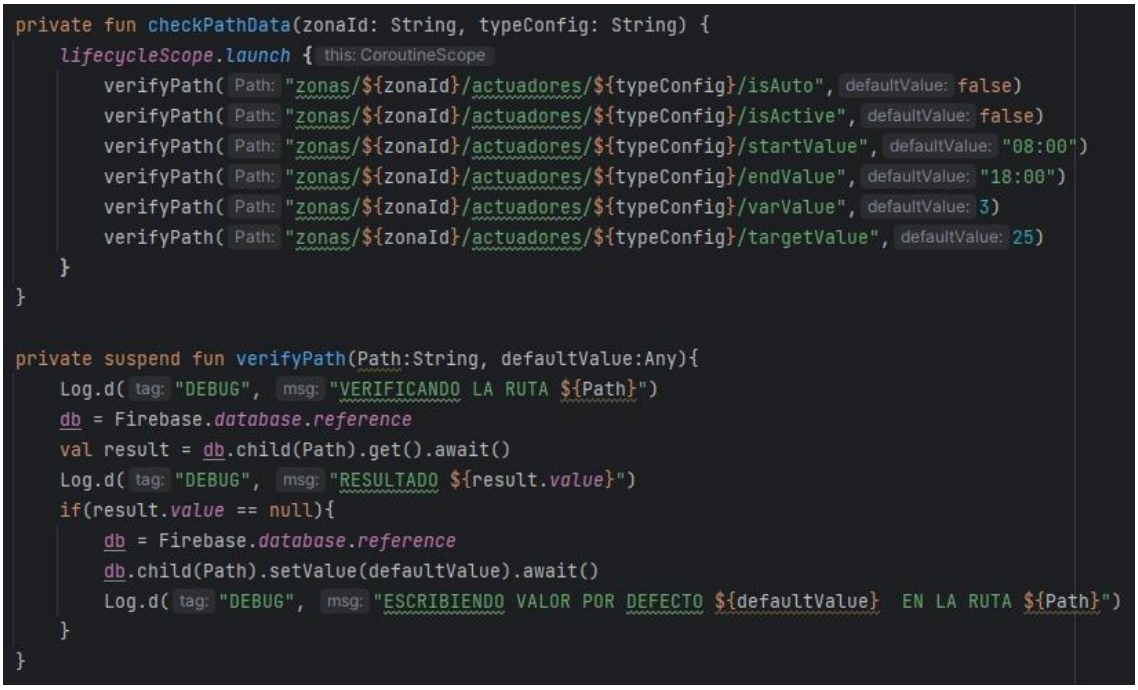
Cada ítem de la lista del *recyclerview* tiene un botón que inicia una instancia de la *activity* “ZoneDetailsActivity.kt” si existen medias. A esta instancia se le pasa el identificador de zona, el nombre y el tipo de configuración (temperatura, humedad o riego).

2.2.4.3 *Actividad de configuración de zona*

La *activity* “ZoneDetailsActivity.kt” es la encargada de obtener y mostrar al usuario la configuración actual del nodo actuador asociado a la zona y variable ambiental. También permite la modificación de la misma.

Cuando se instancia la actividad, se comprueba la existencia de la estructura necesaria en Realtime del nodo actuador asociado a la variable ambiental. Se realiza mediante unas funciones que verifican cada una de las rutas y, si no existen, las generan con un valor por defecto.

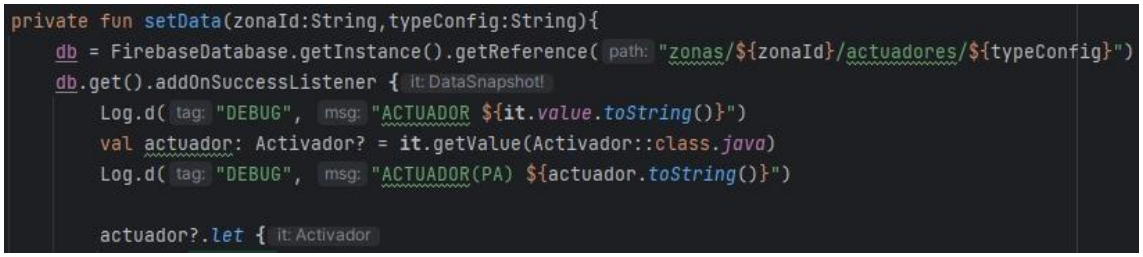
[5] *recyclerview*: elemento gráfico de Android que permite mostrar una lista de elementos mediante una disposición gráfica establecida.



```
private fun checkPathData(zonaId: String, typeConfig: String) {
    lifecycleScope.launch { this: CoroutineScope
        verifyPath( Path: "zonas/${zonaId}/actuadores/${typeConfig}/isAuto", defaultValue: false)
        verifyPath( Path: "zonas/${zonaId}/actuadores/${typeConfig}/isActive", defaultValue: false)
        verifyPath( Path: "zonas/${zonaId}/actuadores/${typeConfig}/startValue", defaultValue: "08:00")
        verifyPath( Path: "zonas/${zonaId}/actuadores/${typeConfig}/endValue", defaultValue: "18:00")
        verifyPath( Path: "zonas/${zonaId}/actuadores/${typeConfig}/varValue", defaultValue: 3)
        verifyPath( Path: "zonas/${zonaId}/actuadores/${typeConfig}/targetValue", defaultValue: 25)
    }
}

private suspend fun verifyPath(Path:String, defaultValue:Any){
    Log.d( tag: "DEBUG",  msg: "VERIFICANDO LA RUTA ${Path}")
    db = Firebase.database.reference
    val result = db.child(Path).get().await()
    Log.d( tag: "DEBUG",  msg: "RESULTADO ${result.value}")
    if(result.value == null){
        db = Firebase.database.reference
        db.child(Path).setValue(defaultValue).await()
        Log.d( tag: "DEBUG",  msg: "ESCRIBIENDO VALOR POR DEFECTO ${defaultValue}  EN LA RUTA ${Path}")
    }
}
```

Ilustración 27 – Funciones de verificación de ruta de Realtime en Android Studio.

En el caso de que exista la estructura o se haya generado, se obtiene cada uno de los valores mediante una petición a Realtime a la ruta del actuador de la zona.



```
private fun setData(zonaId:String,typeConfig:String){
    db = FirebaseDatabase.getInstance().getReference( path: "zonas/${zonaId}/actuadores/${typeConfig}")
    db.get().addOnSuccessListener { it: DataSnapshot!
        Log.d( tag: "DEBUG",  msg: "ACTUADOR ${it.value.toString()}")
        val actuador: Activador? = it.getValue(Activador::class.java)
        Log.d( tag: "DEBUG",  msg: "ACTUADOR(PA) ${actuador.toString()}")

        actuador?.let { it: Activador
```

Ilustración 28 – Parte de la función setData donde realiza la petición a Realtime para obtener la configuración del nodo actuador.

Si se obtiene la configuración, se hace visible el *layout* correspondiente al modo de funcionamiento actual (manual o automático) y se representan los datos obtenidos. Antes de representar los datos, adicionalmente se realiza una petición a Realtime para obtener la duración de *Deep-Sleep* de los nodos sensores asociados a dicha variable ambiental.

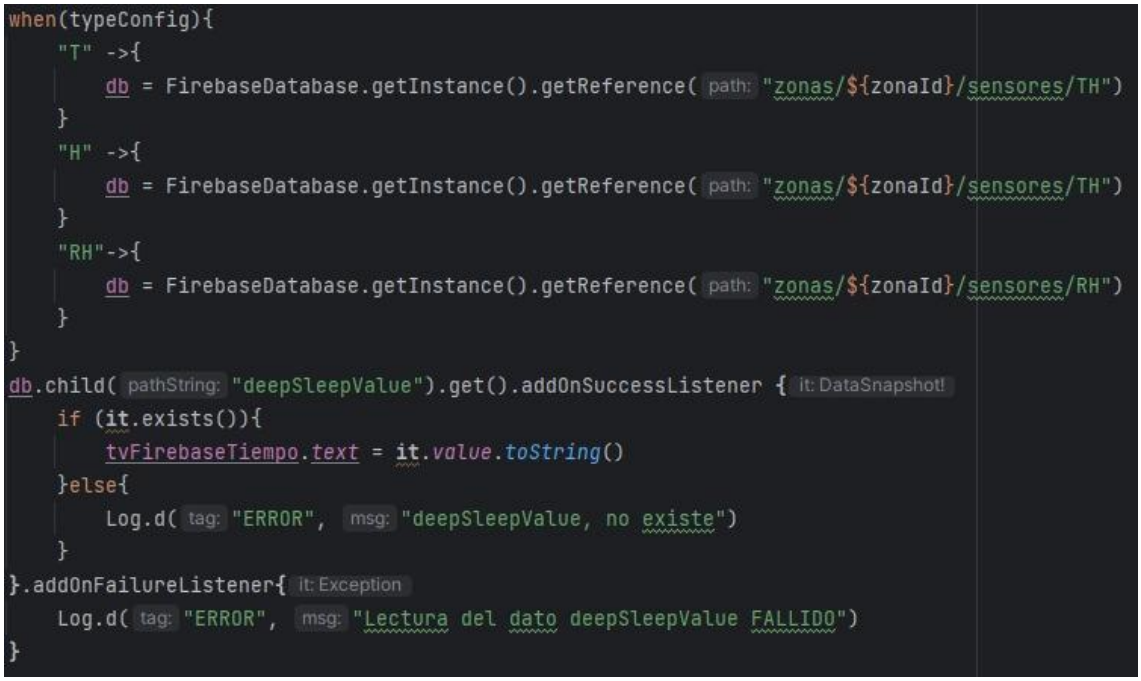


Ilustración 29 – Código para realizar la petición a Realtime del tiempo de Deep-Sleep de un conjunto de nodos sensores.

Una vez representada la configuración actual, se indica a Realtime que notifique de cualquier cambio en estas rutas:

- “zonas/${zonaId}/actuadores/${typeConfig}” – Configuración del nodo actuador de la zona “zonaId” asociado a la variable ambiental “typeConfig”.
- “zonas/${zonaId}/sensores/${typeConfig}/deepSleepValue” – Duración del *Deep-Sleep* de los nodos sensores de la zona “zonaId” asociados a la variable ambiental “typeConfig”.
- “zonas/${zonaId}/medias/${meanValue}” – Media actual de la zona “zonaId” de la variable ambiental.

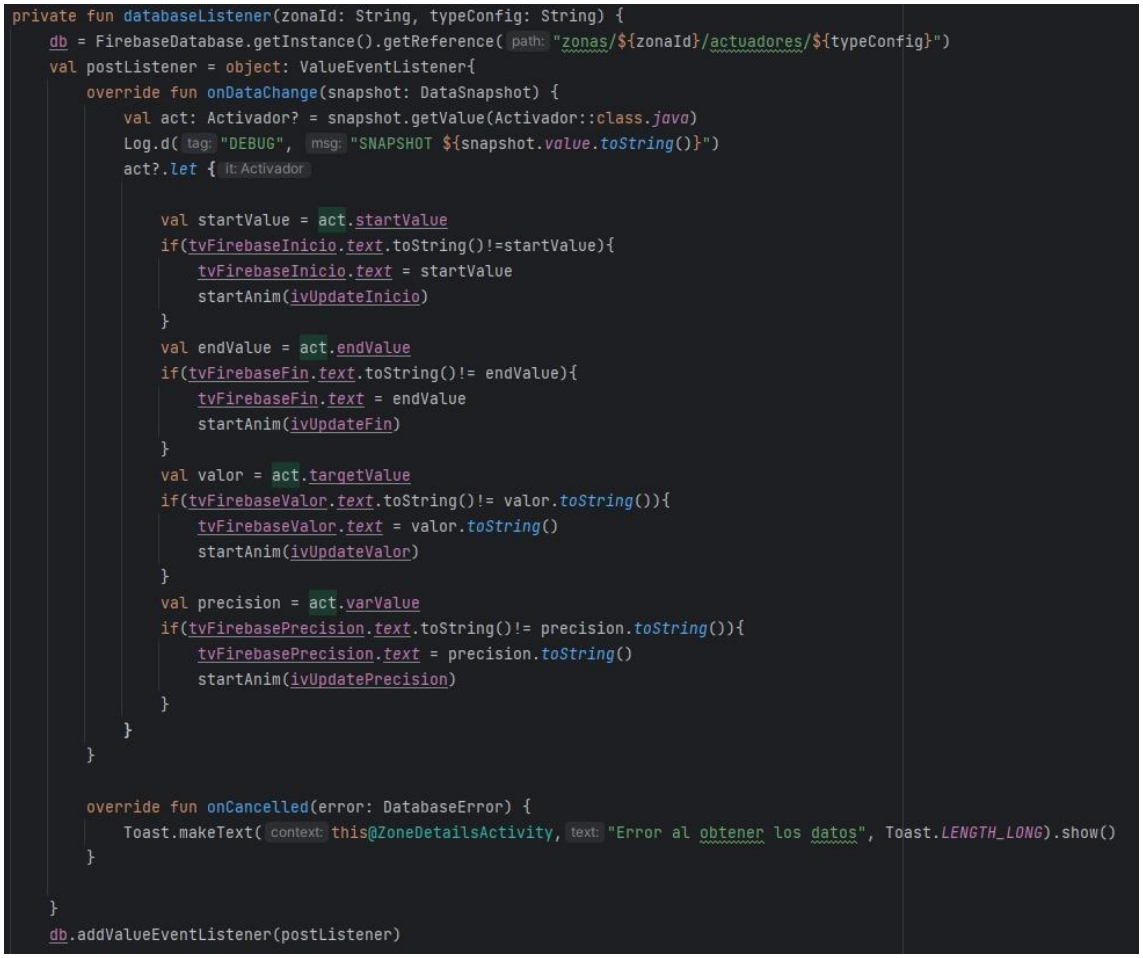


Ilustración 30 – Parte de la función databaseListener donde muestra la definición del listener asociada a la ruta de actuadores.

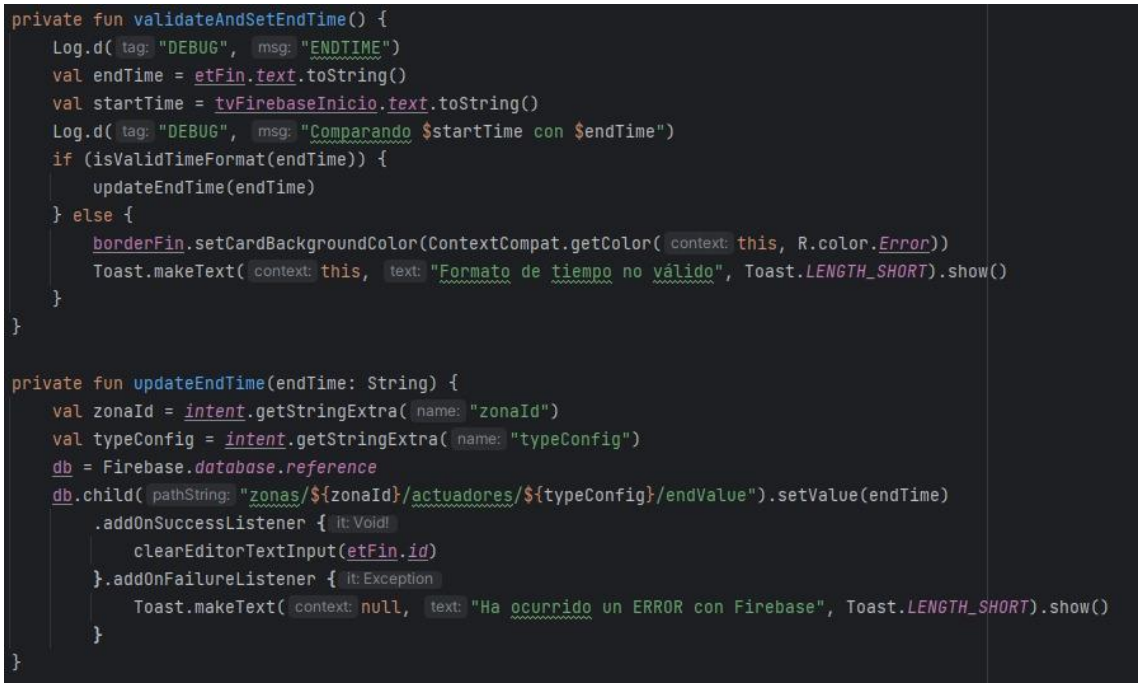
En el *layout* existe un *switch* que sirve para cambiar de un modo de configuración a otro (manual o automático). Este *switch* tiene una función que detecta cuando cambia el estado del *switch* (true o false) y lo envía a la ruta que se observa en la Ilustración 31. Para el *switch* del modo manual, que envía si el sistema se encuentra activo o no, la definición es similar, solo cambia la ruta.

```
private fun switchOnOffListener(zonaId: String,typeConfig: String){
    swOnOff.setOnCheckedChangeListener{_, isChecked ->
        updateIsActive(isChecked, zonaId, typeConfig)
        swOnOff.isEnabled = false
        Handler(Looper.getMainLooper()).postDelayed({
            swOnOff.isEnabled = true
        }, delaySwitch)

    }
}

private fun updateIsActive(checked: Boolean, zonaId: String,typeConfig: String){
    db = Firebase.database.reference
    db.child( pathString: "zonas/${zonaId}/actuadores/${typeConfig}/isActive").setValue(checked)
        .addOnSuccessListener { it: Void!
            if(checked){
                tvManualOn.setTypeface( tf: null, Typeface.BOLD)
                tvManualOff.setTypeface( tf: null, Typeface.NORMAL)
            }else{
                tvManualOff.setTypeface( tf: null, Typeface.BOLD)
                tvManualOn.setTypeface( tf: null, Typeface.NORMAL)
            }
        }
        .addOnFailureListener{ it: Exception
            Toast.makeText( context: null,  text: "Ha ocurrido un ERROR con Firebase", Toast.LENGTH_SHORT).show()
        }
}
```

Ilustración 31 – Función que detecta el cambio en el switch y función que envía el estado del switch a Realtime.

Por último, en el *layout* de la *activity*, existe un *editText*[6] por cada dato que interacciona la *activity*. Estos datos se encuentran en el apartado **Interacción con los nodos**. Cuando el usuario introduce un valor y pulsa para enviar los datos, se verifican y se envían a Realtime con su respectiva ruta.

[6] Elemento editText: campo de la interfaz de usuario donde se puede insertar texto.



```
private fun validateAndSetEndTime() {
    Log.d( tag: "DEBUG",  msg: "ENDTIME")
    val endTime = etFin.text.toString()
    val startTime = tvFirebaseInicio.text.toString()
    Log.d( tag: "DEBUG",  msg: "Comparando $startTime con $endTime")
    if (isValidTimeFormat(endTime)) {
        updateEndTime(endTime)
    } else {
        borderFin.setCardBackgroundColor(ContextCompat.getColor( context: this, R.color.Error))
        Toast.makeText( context: this,  text: "Formato de tiempo no válido", Toast.LENGTH_SHORT).show()
    }
}

private fun updateEndTime(endTime: String) {
    val zonaId = intent.getStringExtra( name: "zonaId")
    val typeConfig = intent.getStringExtra( name: "typeConfig")
    db = Firebase.database.reference
    db.child( pathString: "zonas/${zonaId}/actuadores/${typeConfig}/endValue").setValue(endTime)
        .addOnSuccessListener { it: Void!
            clearEditorTextInput(etFin.id)
        }.addOnFailureListener { it: Exception
            Toast.makeText( context: null,  text: "Ha ocurrido un ERROR con Firebase", Toast.LENGTH_SHORT).show()
        }
}
```

Ilustración 32 – Funciones de validación y envío a Realtime de la hora final del nodo actuador.

Cuando se envía el dato, se actualiza tanto en Realtime como en la aplicación móvil.

2.2.4.4 *Conexión con la API del tiempo*

Se ha integrado la funcionalidad para obtener la meteorología de la zona mediante una API gratuita llamada OpenWeather. Esta API permite realizar una petición GET indicando las coordenadas de la zona, y devuelve un JSON con el estado del tiempo. La estructura de la petición está detallada en su página web (13).

En su versión gratuita, OpenWeather permite 60 llamadas por minuto, aunque ofrecen distintos planes según las necesidades del usuario. Para obtener la API *key*, es necesario registrarse. Una vez registrado, en el perfil, se puede generar la API *key*.

Del JSON obtenido, se transforma a un objeto “WeatherResponse.kt” y se extrae el código del icono asociado al estado del tiempo, el cual es necesario para actualizar el icono que indica la meteorología.

```kotlin
data class WeatherResponse(
    val weather: List<Weather>,
    val main: Main
)

data class Weather(
    val icon: String
)

data class Main(
    val temp: Double
)
```

Ilustración 33 – Estructura de la clase WeatherResponse.

En el código se consigue mediante la librería Retrofit que permite la comunicación con APIs HTTP y transforma las respuestas en objetos.

RetrofitClient.kt

```kotlin
object RetrofitClient {
    private const val BASE_URL = "https://api.openweathermap.org/"

    val weatherApiService: WeatherApiService by lazy {
        Retrofit.Builder() Retrofit.Builder
            .baseUrl(BASE_URL) Retrofit.Builder
            .addConverterFactory(GsonConverterFactory.create())
            .build() Retrofit
            .create(WeatherApiService::class.java)
    }
}
```

WeatherApiService.kt

```kotlin
interface WeatherApiService {
    @GET("data/2.5/weather")
    fun getCurrentWeather(
        @Query("lat") lat: Double,
        @Query("lon") lon: Double,
        @Query("appid") apiKey: String,
        @Query("units") units: String = "metric"
    ): retrofit2.Call<WeatherResponse>
}
```

Ilustración 34 – Clase RetrofitClient.kt e interfaz WeatherApiService.kt.

Se ha implementado un objeto cliente y una interfaz asociada encargada de ejecutar la petición HTTP.

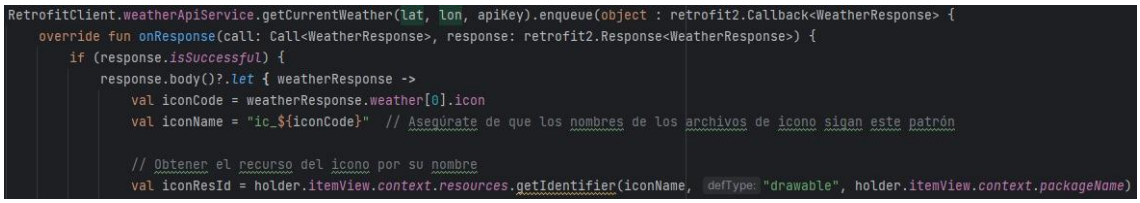


Ilustración 35 – Parte del código de la petición a la API de OpenWeather.

2.2.4.5 *Interacción con los nodos*

El usuario podrá interactuar con los nodos de cada una de las zonas. Para los nodos sensores, se puede cambiar el valor local del siguiente ***Deep-Sleep***.

En el caso de los nodos actuadores, se puede interactuar con los siguientes valores:

- **startValue** (hora inicial)
- **endValue** (hora final)
- **isAuto** (Modo manual/automático)
- **isActive** (Sistema apagado/activo)
- **targetValue** (Valor objetivo)
- **varValue** (Rango del valor objetivo)

El funcionamiento de la aplicación se detalla en profundidad en los **Anexos**.

3 PRUEBAS

3.1 Establecimiento TLS 1.2 de los nodos

Para realizar esta prueba se ha creado un punto de acceso con un ordenador portátil y se ha capturado mediante el programa Wireshark los paquetes que pasan por la interfaz WiFi que genera la red. Seguido, se ha alimentado un nodo actuador para que realice el proceso de autenticación. El resultado se puede observar en la Ilustración 36.

Source	Destination	Protocol	Length	Info
192.168.137.238	142.250.200.138	TLSv1.2	277	Client Hello (SNI=www.googleapis.com)
142.250.200.138	192.168.137.238	TLSv1.2	590	Server Hello
142.250.200.138	192.168.137.238	TLSv1.2	382	Certificate, Server Key Exchange, Server Hello Done
192.168.137.238	142.250.200.138	TLSv1.2	96	Client Key Exchange
192.168.137.238	142.250.200.138	TLSv1.2	60	Change Cipher Spec
192.168.137.238	142.250.200.138	TLSv1.2	91	Encrypted Handshake Message
142.250.200.138	192.168.137.238	TLSv1.2	97	Change Cipher Spec, Encrypted Handshake Message
192.168.137.238	142.250.200.138	TLSv1.2	370	Application Data
142.250.200.138	192.168.137.238	TLSv1.2	590	Application Data
142.250.200.138	192.168.137.238	TLSv1.2	590	Application Data
142.250.200.138	192.168.137.238	TLSv1.2	590	Application Data

Ilustración 36 – Establecimiento TLS 1.2 del nodo actuador en Wireshark.

Se ha filtrado por el protocolo TLS y se puede ver que el dispositivo es aquel que tiene dirección IP **192.168.137.238** y que el servidor es la **142.250.200.138**. El nodo inicia el *Handshake* con el mensaje *Client Hello*, indicando la versión de TLS que soporta, las suites de cifrado entre otras opciones.

Ilustración 37 – Mensaje Client Hello del Handshake de TLS 1.2.

El servidor responde a este mensaje con un *Server Hello* donde acepta los parámetros propuestos por el cliente e indica la suite de cifrado. Seguido de este

mensaje, el servidor envía su certificado seguido de un mensaje *Server Key Exchange* y finaliza el saludo inicial.

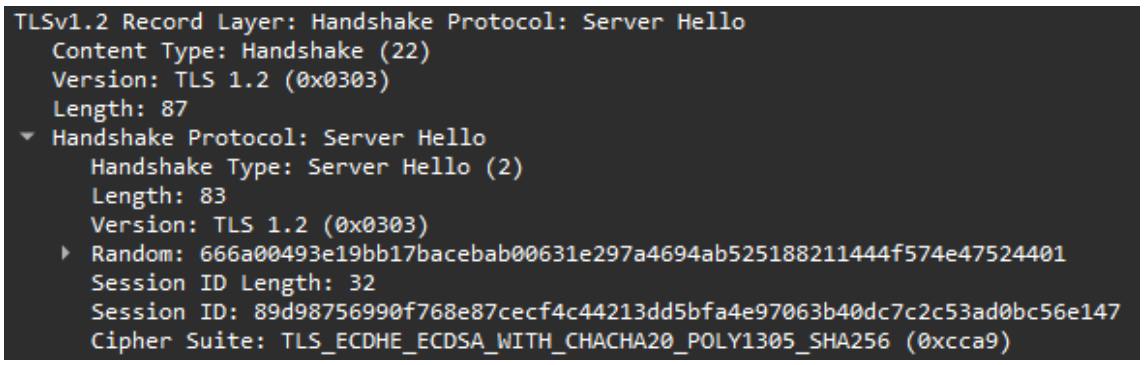


Ilustración 38 – Mensaje Server Hello del Handshake de TLS 1.2.

El nodo envía un mensaje *Client Key Exchange*, que contiene información para que el servidor derive las claves de sesión, confirma al servidor el uso de claves y algoritmos negociados y termina el proceso del *Handshake*. A partir de este mensaje, todas las comunicaciones entre el cliente y el servidor están cifradas, garantizando la confidencialidad e integridad de los datos.

3.2 Consumo

Se ha realizado un estudio con el objetivo de obtener el consumo de un nodo sensor durante varios ciclos ***Wake-UP Deep-Sleep***. Se ha utilizado una placa de desarrollo **LONIN WEMOS D1 Mini** y un sensor de corriente **INA226**.

Se ha diseñado un código en **Arduino** que envía por el puerto serial la información obtenida por el sensor de corriente **INA226**. Esta información es recogida por un script de **Python** que transforma los datos y los guarda en un documento de **Excel**.

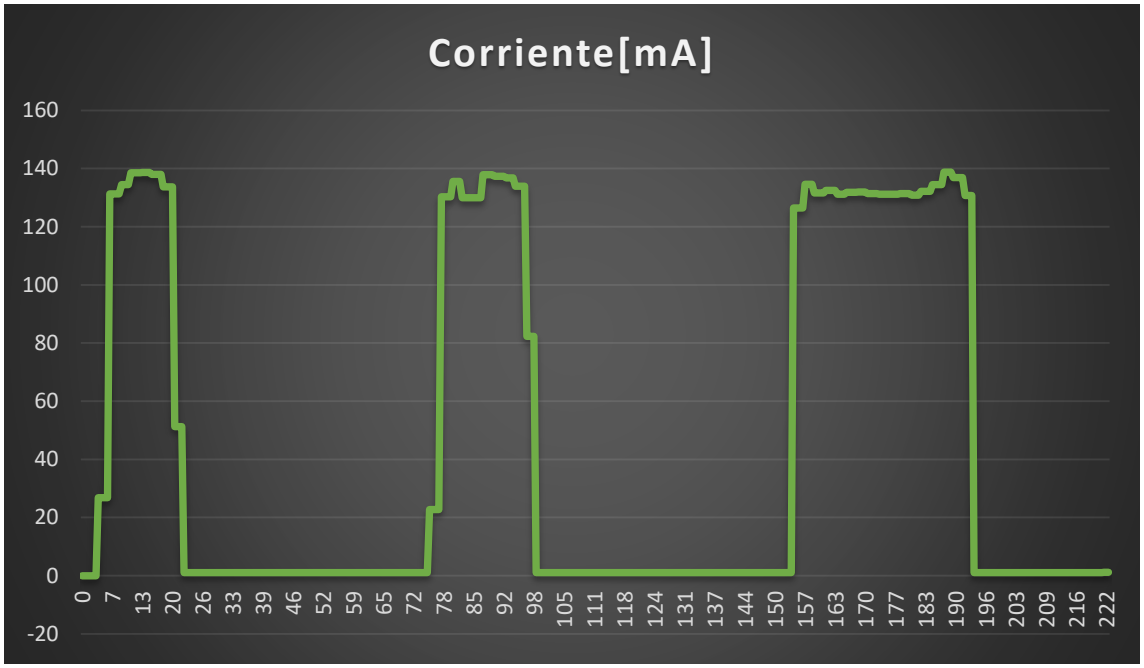


Ilustración 39 – Corriente del nodo sensor TH, obtenida con sensor INA226.

Con estos datos se puede calcular la duración máxima teórica de la batería utilizando los siguientes valores:

- Corriente activa: valor máximo obtenido (138,82 mA)
- Corriente reposo: valor mínimo obtenido (1 mA)
- Duración activa: Se ha utilizado una media de 15 segundos
- Duración reposo: 30 min
- Capacidad de la batería: 3400 mAh

Como valor mínimo no se han tenido en cuenta los obtenidos ya que rondan los 1 mA, el límite del sensor. Tras buscar estudios en internet (14) se encontró un estudio donde los consumos de este tipo de placas rondan los 156 µA.

Primero se calcula el consumo de un ciclo, que estará formada por el consumo en reposo y el consumo en activo.

$$Consumo\ en\ reposo = 0{,}156\ mA * 0{,}5\ h = 0{,}078\ mAh$$

$$Consumo\ en\ activo = 138.82\ mA * 15\ s * \frac{1\ h}{3600\ s} = 0{,}578\ mAh$$

$$Consumo\ de\ un\ ciclo = 0{,}078\ mAh + 0{,}578\ mAh = 0{,}656\ mAh$$

Ahora se calcula el número de ciclos (*Wake-Up Deep-Sleep*) que soportaría la batería.

$$Número\ de\ ciclos = \frac{3400\ mAh}{0{,}656\ mAh} = 5.179{,}63\ ciclos$$

Se multiplica por el número de ciclos para obtener la duración de la batería.

$$Duración = 5.179{,}63 * 1815\ s = 9.401.041\ s = 3\ meses, 2\ días\ y\ 4\ horas$$

Esta duración es teórica y para nada se asemeja a la realidad ya que las baterías se descargan un porcentaje mensualmente y no es recomendable descargarla por debajo del 20%. Si se añade un factor de seguridad del 30%, se obtiene una duración de 2 meses.

3.3 Envío de datos del nodo sensor

Para comprobar el funcionamiento del nodo sensor, se ha utilizado un nodo sensor de temperatura y humedad. Tras alimentarlo con la batería, se ha observado por la salida serial el proceso de autenticación, obtención de la duración del *Deep-Sleep* y el envío de los datos.

```
.................Conectado!
Token info: type = id token (GITKit token), status = on request
Token info: type = id token (GITKit token), status = ready
Duración del siguiente DeepSleep: 1m
Archivo eliminado... /templastValue.txt
Se han añadido los datos correctamente: 29.25,1718616913

Archivo eliminado... /humdlastValue.txt
Se han añadido los datos correctamente: 43.47,1718616915

Datos a enviar...
{
        "tempTimeStamp": 1718616913,
        "tempValue": 29.2511673,
        "humdTimeStamp": 1718616915,
        "humdValue": 43.46593475
}
Datos enviados...
No se encuentra el archivo /tempoldValue.txt
No se encuentra el archivo /humdoldValue.txt
No se encuentra el archivo /errors.txt
Entrando en deepsleep por 1m ...
```

Ilustración 40 – Salida serial de un nodo sensor de temperatura y humedad durante un ciclo.

Estos datos se han correlacionado con los campos en Realtime y que la media de la zona se ha actualizado.

Ilustración 41 – Datos actualizados en la base de datos Realtime.

Se comprobará que almacene datos antiguos si ocurre algún error en la conexión y realice correctamente el registro de los errores. Se desactivará el punto de acceso, lo que provocará el error 11(no se detecta el SSID) y almacenará los datos obtenidos en ese instante. En la Ilustración 42 se puede observar como genera el error, obtiene los datos del sensor y los almacena en los registros.

```
[ERROR] ID:11(No se encuentra el SSID: RED_Invernadero)
Se han añadido los datos correctamente: 11.00,1718664714

Se han añadido los datos correctamente: 28.62,1718664715

Se han añadido los datos correctamente: 37.84,1718664716

Archivo eliminado... /templastValue.txt
Se han añadido los datos correctamente: 28.62,1718664717

Archivo eliminado... /humdlastValue.txt
Se han añadido los datos correctamente: 37.84,1718664718

Entrando en deepsleep por 0h & 30m ...
```

Ilustración 42 – Salida del serial durante la simulación de generación de errores y guardado en los históricos.

Este proceso se ha realizado dos veces y en la tercera se ha activado el punto de acceso. En la salida serial se puede observar como el nodo sensor, aparte de los datos actuales de temperatura y humedad, añade al JSON los registros de error y valores ambientales antiguos y los envía a Realtime.

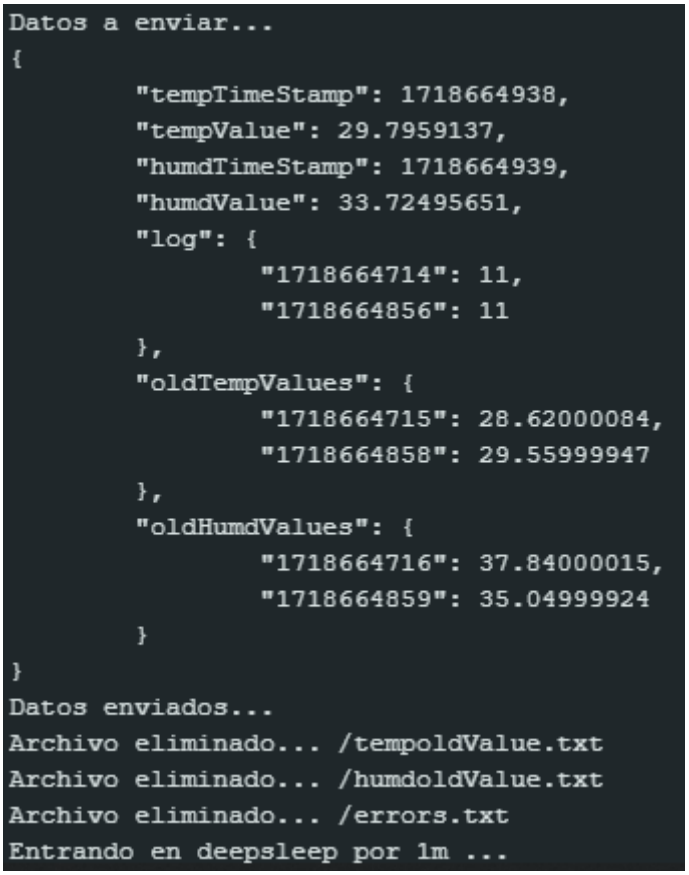


Ilustración 43 – Salida serial donde se observa el JSON enviado por el nodo sensor tras generar dos errores.

Actualmente, aunque se envían los registros de error e históricos antiguos a Realtime, no se almacenan en Firestore. Se tiene en consideración como línea futura en el apartado **Notificación de errores**.

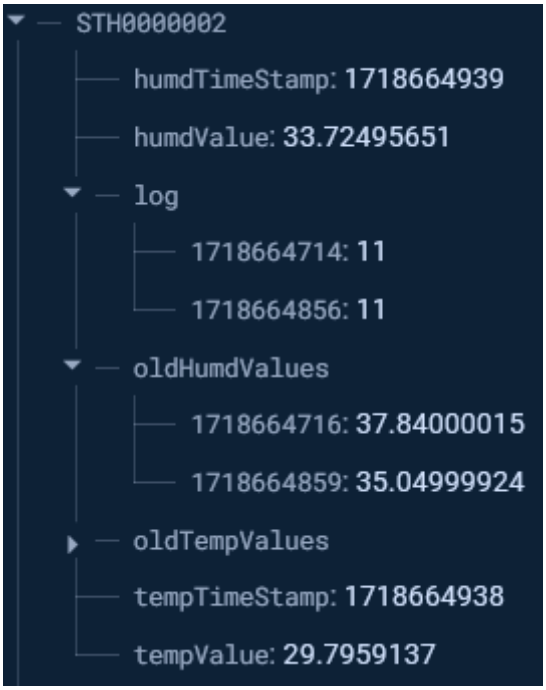


Ilustración 44 – Campos del nodo sensor en Realtime con los registros de error y datos ambientales.

3.4 Lectura de datos del nodo actuador

Del mismo modo que el nodo sensor, se va a comprobar que Realtime notifica los cambios correctamente en los nodos actuadores. Para esta prueba se ha utilizado el nodo actuador que actúa sobre el sistema de riego. Una vez alimentado y autenticado, el nodo sensor obtiene la configuración y la media de humedad de suelo actual.

```
Camino de evento: /
Tipo de dato: json
Tipo de evento: put
Valor: {"endValue":"23:59","isActive":true,"isAuto":false,"startValue":"11:00","targetValue":60,"varValue":5}

Camino de escucha: /zonas/Z0000001/medias/meanRHumdValue
Camino de evento: /
Tipo de dato: int
Tipo de evento: put
Valor: 60
```

Ilustración 45 – Valores de configuración iniciales del nodo actuador obtenido de la base de datos Realtime.

Una vez configurado, se mantiene en espera de que ocurran cambios. Se ha utilizado la aplicación móvil para cambiar el modo de funcionamiento de manual a automático y la franja horaria de inicio y fin de actuación. Realtime notifica al actuador el campo que ha cambiado, el tipo de dato y su valor.

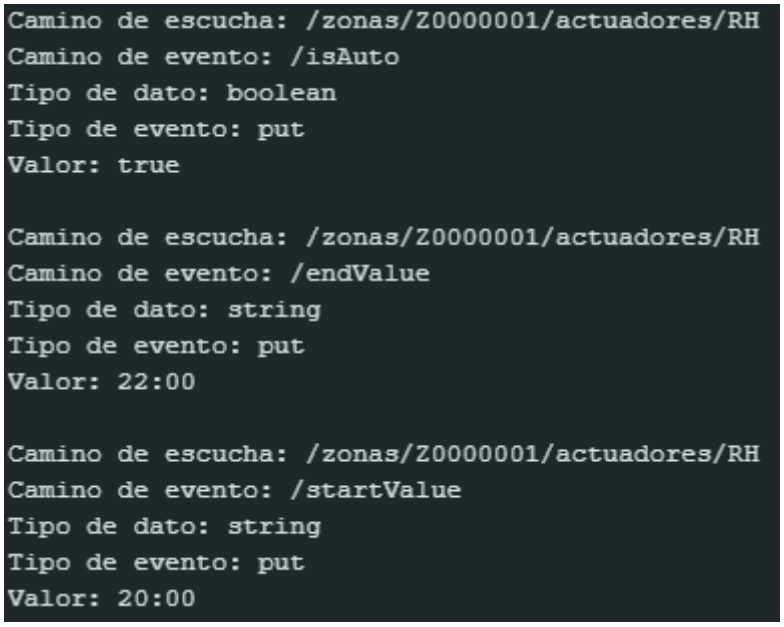


Ilustración 46 – Datos recibidos de la base de datos Realtime tras detectar cambios.

4 PRESUPUESTO

En este apartado, se muestra el coste, por capítulos, de los equipos y servicios necesarios para la puesta en marcha del sistema.

4.1 Capítulo 1 Dispositivos IoT

4.1.1 *PARTIDA 1.1 – NODO SENSOR TH*

En esta partida, se recogen los costes asociados al Nodo sensor TH, dispositivo encargado de medir temperatura y humedad ambiental y transmitirlos al servidor de forma periódica y autónoma.

Tabla 16 – Partida 1.1 del presupuesto.

Ud.	Concepto	P.Unitario	Subtotal
1	Placa Wemos Lonin D1 Mini v3	4,12 €	4,12 €
1	Wemos D1 mini Battery Shield	1,04 €	1,04 €
1	Sensor SHT40 + Carcasa + Cable 1m	5,61 €	5,61 €
1	Transistor BJT 2N2222 NPN	0,21 €	0,21 €
2	Resistencia 220Ω ¼ w ±5% 300V	0,07 €	0,14€
1	Mini pulsador PCB 6x6x5mm 2 o 4 pines	0,12 €	0,12 €
1	Modulo reloj DS3231 RTC + Batería	2,47 €	2,47 €
1	Interruptor ON/OFF 21x15mm Negro	0,49 €	0,49 €
1	Interruptor ON/OFF 15x10mm Rojo	0,29 €	0,29 €
2	Conector XH 2,54mm 2 pines macho y hembra	0,08 €	0,16 €
1	Conector XH 2,54mm 4 pines macho y hembra	0,08 €	0,08 €
1	Placa PCB 7x9cm perforada doble cara 2,54mm	0,70 €	0,70 €
1	Portapilas 1x batería 18650 con cable	0,57 €	0,57 €
1	Pila recargable 18650 Ion-Litio 3,6V 3400mAh	23,45 €	23,45 €
1	Cinta de cable plano AWG28 40 cables 20cm	0,90 €	0,90 €
1	Prensaestopas PA6 6-8mm Negro	0,41 €	0,41 €
1	Carcasa impresa en 3D	12 €	12 €
		Total 1.1	**52,67 €**

4.1.2 *PARTIDA 1.2 – NODO SENSOR RH*

En esta partida, se recogen los costes asociados al Nodo sensor RH, dispositivo encargado de medir la humedad del suelo y transmitirlos al servidor de forma periódica y autónoma.

Tabla 17 – Partida 1.2 del presupuesto.

Ud.	Concepto	P.Unitario	Subtotal
1	Placa Wemos Lonin D1 Mini v3	4,12 €	4,12 €
1	Wemos D1 mini Battery Shield	1,04 €	1,04 €
1	Sensor capacitivo de humedad	0,88 €	0,88 €
1	Transistor BJT 2N2222 NPN	0,21 €	0,21 €
1	Resistencia 220Ω ¼ w ±5% 300V	0,07 €	0,07 €
1	Resistencia 10kΩ ¼ w ±5% 300V	0,07 €	0,07 €
1	Mini pulsador PCB 6x6x5mm 2 o 4 pines	0,12 €	0,12 €
1	Modulo reloj DS3231 RTC + Batería	2,47 €	2,47 €
1	Interruptor ON/OFF 21x15mm Negro	0,49 €	0,49 €
1	Interruptor ON/OFF 15x10mm Rojo	0,29 €	0,29 €
2	Conector XH 2,54mm 2 pines macho y hembra	0,08 €	0,16 €
1	Conector XH 2,54mm 3 pines macho y hembra	0,08 €	0,08 €
1	Placa PCB 7x9cm perforada doble cara 2,54mm	0,70 €	0,70 €
1	Portapilas 1x batería 18650 con cable	0,57 €	0,57 €
1	Pila recargable 18650 Ion-Litio 3,6V 3400mAh	23,45 €	23,45 €
1	Cinta de cable plano AWG28 40 cables 20cm	0,90 €	0,90 €
5	Cable 28AWG 3 hilos	0,81 € / m	4,05 €
1	Prensaestopas PA6 6-8mm Negro	0,41 €	0,41 €
1	Carcasa impresa en 3D	12 €	12 €
		Total 1.2	**49,91 €**

4.1.3 *PARTIDA 1.3 – NODO ACTUADOR*

En esta partida se recogen los costes asociados al Nodo actuador, dispositivo encargado de interaccionar con el sistema de control de temperatura, humedad o riego.

Tabla 18 – Partida 1.3 del presupuesto.

Ud.	Concepto	P.Unitario	Subtotal
1	Placa Wemos Lonin D1 Mini v3	4,12 €	4,12 €
3	Diodo LED	0,17 €	0,51 €
1	Placa Protoboard 56x165mm	1,31 €	1,31 €
3	Resistencia 220Ω ¼ w ±5% 300V	0,07 €	0,21 €
		Total 1.2	**6,15 €**

4.1.4 *PARTIDA 1.4 – NODO CENTRAL/PUNTO DE ACCESO*

En esta partida se recogen los costes asociados al Nodo central o Punto de acceso, dispositivo encargado de ofrecer una red WLAN y conexión a internet a los nodos sensores TH y RH.

Tabla 19 – Partida 1.4 del presupuesto.

Ud.	Concepto	P.Unitario	Subtotal
1	Punto de Acceso que permita una red WiFi Mesh escalable de alto rango (100 m2 mínimo por AP)	82,64 €	82,64 €
20	Cable de 4 pares UTP Cat.5e	0,29 € /m	5,80 €
2	Conectores macho RJ-45	0,19 €	0,38 €
2	Instalación del punto de acceso y conexión al router proveedor de servicio de internet.	25 €	4,12 €
		Total 1.4	**92,94 €**

4.1.5 *Resumen Capítulo 1*

Para el desarrollo del Trabajo Fin de Grado, se han utilizado dos nodos sensores TH, dos nodos sensores RH y tres nodos actuadores.

Tabla 20 – Resumen del Capítulo 1 del presupuesto.

Ud.	Capítulo 1 - Dispositivos IoT	P. Partida	Subtotal
2	**Partida 1.1 – Nodo sensor TH**	52,67 €	105,34 €
2	**Partida 1.2 – Nodo sensor RH**	49,91 €	99,82 €
3	**Partida 1.3 – Nodo actuador**	6,15 €	18,45 €
1	**Partida 1.4 – Nodo Central / Punto de Acceso**	92,94 €	92,94 €
	TOTAL CAPITULO 1		**316,55 €**

4.2 Capítulo 2 Servicios

4.2.1 *Partida 2.1 – Servicios de Firebase*

En esta partida se recogen los costes asociados al servicio encargado de gestionar y almacenar los datos transmitidos por los nodos sensores, como de la autenticación e interacción de las distintas piezas del sistema. Firebase cobra al usuario según el uso de sus servicios, que en este caso es nulo porque no se supera el límite. Los costes de cada servicio se pueden encontrar desglosados en su página web (15).

Tabla 21 – Partida 2.1 del presupuesto.

Ud.	Concepto	P.Unitario	Subtotal
1	Servicio de Autenticación	0 €	0 €
1	Servicio de Realtime Database	0 €	0 €
1	Servicio de Firestore Database	0 €	0 €
1	Servicio de Functions	0 €	0 €
		Total 2.1	**0 €**

4.2.2 *Resumen Capítulo 2*

Tabla 22 – Resumen del Capítulo 2 del presupuesto.

Capítulo 2 - Servicios	
Partida 2.1 – Servicios de Firebase	0 €
TOTAL CAPITULO 2	**0 €**

4.3 Capítulo 3 Implementación del TFG

Este capítulo detalla las horas y su coste asociado para el desarrollo e implementación del TFG agrupados según la plataforma.

4.3.1 *Partida 3.1 – Plataforma IoT*

En esta partida se recogen los costes asociados al diseño e implementación de los nodos sensores y actuadores.

Tabla 23 – Partida 3.1 del presupuesto.

Ud.	Concepto	P.Unitario	Subtotal
40 h	Estudio previo	25 € / h	1.000 €
80 h	Diseño y programación	25 € / h	2.000 €
25 h	Pruebas y test	25 € / h	625 €
5 h	Implementación	25 € / h	125 €
150 h		**Total 4.1**	**3.750 €**

4.3.2 *Partida 3.2 – Plataforma Firebase*

En esta partida se recogen los costes asociados al diseño e implementación de los servicios Firebase utilizados en el TFG.

Tabla 24 – Partida 3.2 del presupuesto.

Ud.	Concepto	P.Unitario	Subtotal
20 h	Estudio previo	25 € / h	500 €
25 h	Diseño y programación	25 € / h	625 €
4 h	Pruebas y test	25 € / h	100 €
1 h	Implementación	25 € / h	25 €
50 h		**Total 3.2**	**1.250 €**

4.3.3 *Partida 3.3 – Plataforma Android*

En esta partida se recogen los costes asociados al diseño e implementación de la aplicación nativa en el sistema operativo Android.

Tabla 25 – Partida 3.3 del presupuesto.

Ud.	Concepto	P.Unitario	Subtotal
30 h	Investigación	25 € / h	750 €
60 h	Diseño y programación	25 € / h	1.500 €
10 h	Pruebas y test	25 € / h	250 €
100 h		**Total 4.3**	**2.500 €**

4.3.4 *Resumen Capítulo 3*

Tabla 26 – Resumen del Capítulo 3 del presupuesto.

Capítulo 3 – Implementación del TFG		**Horas**
Partida 3.1 – Plataforma IoT	3.750 €	150 h
Partida 3.2 – Plataforma Firebase	1.250 €	50 h
Partida 3.3 – Plataforma Android	2.500 €	100 h
TOTAL CAPITULO 3	**7.500 €**	**300 h**

4.4 Resumen del presupuesto

Tabla 27 – Resumen del presupuesto.

RESUMEN	Precio	Tiempo
TOTAL CAPÍTULO 1 – DISPOSITIVOS IOT	316,55 €	
TOTAL CAPÍTULO 2 – SERVICIOS	0 €	
TOTAL CAPÍTULO 3 – IMPLEMENTACIÓN DEL TFG	7.500 €	300 h
SUBTOTAL	**7.816,55 €**	**300 h**
I.V.A (21%)	**1.641,48 €**	
TOTAL PROYECTO	**9.458,03 €**	**300 h**

El importe total del presupuesto del presente del Trabajo Fin de Grado asciende a la cantidad de **NUEVE MIL CUATRO CIENTOS CINCUENTA Y OCHO EUROS Y TRES CÉNTIMOS (9.458,03 €)**.

5 CONCLUSIONES Y LINEAS DE FUTURO

5.1 Conclusiones

En este apartado, se verificará el cumplimiento de los objetivos del TFG. A continuación, se presentan los objetivos alcanzados.

- Se ha desarrollado un sistema basado en dispositivos IoT para el control y visualización de varios parámetros medioambientales.
- Se han utilizado tres variables ambientales: temperatura, humedad y humedad del suelo.
- Se ha empleado Firebase como plataforma digital para almacenar, procesar y gestionar los datos obtenidos por los dispositivos IoT.
- Se ha creado una aplicación móvil que facilita el control y la configuración individual y conjunta de los dispositivos IoT.
- Los nodos actuadores controlan LEDs para indicar su estado.

5.2 Líneas de futuro

5.2.1 *AÑADIR OPCIÓN DE TEMPORIZACIÓN AL RIEGO*

Un temporizador de riego, evitaría que el usuario deje encendido el sistema por error. Además, permitiría programar horarios específicos y optimizar el riego.

5.2.2 *NODOS ACTUALIZABLES VÍA OTA*

La librería utilizada en los dispositivos IoT proporciona una función para que el dispositivo pueda comprobar si existe una nueva versión del software. Si se detecta una nueva versión, el dispositivo puede descargar automáticamente el nuevo archivo sin intervención presencial. Además, se añadiría la configuración inicial a los nodos actuadores.

5.2.3 *DISEÑO DE UN SISTEMA FOTOVOLTAICO AUTÓNOMO*

Con un Sistema Fotovoltaico Autónomo (SFA) diseñado a medida, permitirá incrementar el número de mediciones y evitar la recarga de baterías.

5.2.4 *APP MULTIUSUARIO Y APP WEB*

Añadir un inicio de sesión y asociar las zonas a los usuarios para evitar el uso del sistema por personas no autorizadas. El desarrollo de una aplicación web para convertirla en multiplataforma.

5.2.5 *VISUALIZACIÓN DE DATOS*

Añadir una opción en la ventana de visualización de zonas que permita ver en una gráfica desplegable los valores obtenidos de esa variable ambiental en un determinado periodo de tiempo.

5.2.6 *EDITOR DE LOCALIZACIÓN Y NOMBRE DE ZONAS*

Incluir la posibilidad de editar la localización de la zona por parte del usuario y el nombre de zona. Además, permitir a los usuarios asignar descripciones detalladas a cada zona para facilitar su identificación y gestión.

5.2.7 *NOTIFICACIÓN DE ERRORES Y GUARDADO DE REGISTROS*

El sistema guarda los errores generados por los sensores en un histórico. Añadiendo una función en Firebase se puede conseguir que avise al usuario si se reciben errores mediante una notificación, indicando la zona, el código de error y el sensor que lo ha producido.

6 REFERENCIAS Y BIBLIOGRAFÍA

1. **IETF.** The WebSocket Protocol. [En línea] Diciembre de 2011. https://datatracker.ietf.org/doc/html/rfc6455.

2. —. HTTP/2. [En línea] Junio de 2022. https://datatracker.ietf.org/doc/html/rfc9113.

3. **JOHN DEERE.** John Deere impulsa una innovación en España. [En línea] 18 de Octubre de 2023. https://www.deere.es/es/nuestra-compa%C3%B1%C3%ADa/noticias-y-medios-de-comunicaci%C3%B3n/notas-de-prensa/2023/john-deere-impulsa-una-innovacion-en-espana.html.

4. **Autogrow.** MultiGrow - Greenhouse Climate Controller. [En línea] https://autogrow.com/products/multigrow.

5. **CropX.** CropX Agronomic Farm Management System. [En línea] https://cropx.com/.

6. **IETF.** RFC 9431 - Message Queuing Telemetry Transport (MQTT) and Transport Layer Security (TLS) Profile of Authentication and Authorization for Constrained Environments (ACE) Framework. [En línea] Julio de 2023. https://datatracker.ietf.org/doc/html/rfc9431.

7. —. HTTP Over TLS. [En línea] Mayo de 2000. https://datatracker.ietf.org/doc/html/rfc2818.

8. —. RFC 9112 - HTTP/1.1. [En línea] Junio de 2022. https://datatracker.ietf.org/doc/html/rfc9112.

9. —. RFC 8446 - The Transport Layer Security (TLS) Protocol Version 1.3. [En línea] Agosto de 2018. https://datatracker.ietf.org/doc/html/rfc8446.

10. —. RFC 7252 - The Constrained Application Protocol (CoAP). [En línea] Junio de 2014. https://datatracker.ietf.org/doc/html/rfc7252.

11. **mozbizt.** Firebase ESP Client. [En línea] https://github.com/mobizt/Firebase-ESP-Client.

12. **Firebase.** Documentación de Firebase. [En línea] https://firebase.google.com/docs?hl=es.

13. **OpenWeather.** One Call API: weather data for any geographical coordinate. [En línea] https://openweathermap.org/api/one-call-api.

14. **del Valle Hernández, Luis.** ESP8266 Deep Sleep, cuánto consumen NodeMCU y Wemos D1 Mini. [En línea] Programarfacil, 2022. https://programarfacil.com/esp8266/esp8266-deep-sleep-nodemcu-wemos-d1-mini/.

15. **GOOGLE.** Firebase Pricing. [En línea] Firebase, Febrero de 2024. https://firebase.google.com/pricing?hl=es-419#blaze-calculator.

16. **EIZAGIRRE.** ¿Cuál es el rango de alcance medio de una red Wi-Fi? [En línea] 22 de Mayo de 2023. https://elandroidefeliz.com/rango-de-alcance-medio-de-una-red-wi-fi/.

17. **Aguirre, Austin.** 2.4 GHz vs. 5 GHz Wi-Fi: Which One Should You Use for Better Range and Speed? [En línea] HighSpeedInternet.com, 20 de Octubre de 2022. https://www.highspeedinternet.com/resources/2-4-ghz-vs-5-ghz-wi-fi.

18. **Garcia, Alberto.** ¿A cuántos metros llega el WiFi, el Bluetooth, el NFC o los infrarrojos? [En línea] ADSLZone, 20 de Enero de 2022. https://www.adslzone.net/noticias/redes/alcance-wifi-bluetooth-nfc-cobertura-movil/.

19. **Teel, John.** Comparison of Wireless Technologies: Bluetooth, WiFi, BLE, Zigbee, Z-Wave, 6LoWPAN, NFC, WiFi Direct, GSM, LTE, LoRa, NB-IoT, and LTE-M. [En línea] Predictable Designs, 17 de Julio de 2022. https://predictabledesigns.com/wireless_technologies_bluetooth_wifi_zigbee_gsm_lte_lora_nb-iot_lte-m/.

20. **DSET Energy.** Sigfox: ventajas y funcionalidades de la tecnología IoT - Productos IoT. [En línea] 2018. http://productos-iot.com/sigfox-3/.

21. —. Lorawan - Tecnología y distribución de productos LoraWan. [En línea] 2018. http://productos-iot.com/lorawan-3/.

22. **Wang, Aaron.** LoRa vs Zigbee: la mejor tecnología para la conectividad IoT. [En línea] Mokolora, 25 de Agosto de 2022. https://www.mokolora.com/lora-vs-zigbee-which-is-better/.

23. **RCARRILLO.** Qué es LoRa, cómo funciona y características principales. [En línea] Venco Electronica, 25 de Agosto de 2022. https://www.vencoel.com/que-es-lora-como-funciona-y-caracteristicas-principales/.

24. **ALLDATASHEET.COM.** DHT22 Datasheet (PDF). [En línea] https://pdf1.alldatasheet.com/datasheet-pdf/view/1132459/ETC2/DHT22.html.

25. —. DHT11 Datasheet (PDF). [En línea] https://pdf1.alldatasheet.com/datasheet-pdf/view/1440068/ETC/DHT11.html.

26. —. BME680 Datasheet (PDF). [En línea] https://pdf1.alldatasheet.com/datasheet-pdf/view/1132061/BOSCH/BME680.html.

27. **SENSIRON.** Sensirion_Humidity_Sensors_SHT3x_Datasheet_digital-971521.pdf. [En línea] Mouser, Agosto de 2016. https://www.mouser.com/datasheet/2/682/Sensirion_Humidity_Sensors_SHT3x_Datasheet_digital-971521.pdf.

28. —. HT_DS_Datasheet_SHT4x.pdf. [En línea] SENSIRON, Abril de 2024. https://sensirion.com/media/documents/33FD6951/662A593A/HT_DS_Datasheet_SHT4x.pdf.

29. **DFRobot.** Capacitive Soil Moisture Sensor . [En línea] https://rajguruelectronics.com/Product/5538/Capacitive%20Soil%20Moisture%20Sensor%20V2(1).0.pdf.

30. **IETF.** RFC 9110 - HTTP Semantics. [En línea] Junio de 2022. https://datatracker.ietf.org/doc/html/rfc9110.

31. —. RFC 6101 - The Secure Sockets Layer (SSL) Protocol Version 3.0. [En línea] Agosot de 2011.

32. **ALLDATASHEET.COM.** ESP32 Datasheet (PDF). [En línea] 2019. https://pdf1.alldatasheet.com/datasheet-pdf/view/1148023/ESPRESSIF/ESP32.html.

33. **ESPRESSIF Systems.** ESP8266 Datasheet (PDF). [En línea] 2023. https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf.

34. **INWG.** Augmented BNF for Syntax Specifications: ABNF. [En línea] Enero de 2008. https://datatracker.ietf.org/doc/html/rfc5234.

35. **ALLDATASHEET.COM.** BME280 Datasheet (PDF). [En línea] https://pdf1.alldatasheet.com/datasheet-pdf/view/1132060/BOSCH/BME280.html.

36. **Info Barcelona.** El Smart City Expo World Congress 2023 muestra una Barcelona puntera en innovación urbana. [En línea] 20 de Noviembre de 2023. https://www.barcelona.cat/infobarcelona/es/el-smart-city-expo-world-congress-2023-muestra-una-barcelona-puntera-en-innovacion-urbana_1342940.html.

37. **Texas Instruments.** INA226 datasheet (PDF). [En línea] Agosto de 2015. https://www.ti.com/lit/ds/symlink/ina226.pdf.

38. **Sacristán, Laura.** El ojo que todo lo ve en la Fórmula 1: así funcionan los F1 Insights de Amazon Web Services . [En línea] Xataka, 11 de Junio de 2023. https://www.xataka.com/movilidad/ojo-que-todo-ve-formula-1-asi-f1-insights-aws.

39. **IETF.** JSON Web Token (JWT). [En línea] Mayo de 2015. https://datatracker.ietf.org/doc/html/rfc7519.

40. —. The Base16, Base32, and Base64 Data Encodings. [En línea] Octubre de 2006. https://datatracker.ietf.org/doc/html/rfc4648.

41. **Nodejs.** Node.js - Run JavaScript Everywhere. [En línea] https://nodejs.org/en.

7 ANEXOS

7.1 Anexo I – Manual de despliegue

7.1.1 *INICIAR PROYECTO EN FIREBASE*

El primer paso para el despliegue es configurar el proyecto en **Firebase**. Se entra en la consola de Firebase (Firebase Console) y se selecciona "**crear proyecto**". Firebase pedirán un nombre para el proyecto y si se desea compartir los datos para utilizar el servicio de Google Analytics. En este caso, se rechaza esta última opción.

El siguiente paso será mejorar el plan gratuito al plan **Blaze**. Es necesario realizar esta mejora para acceder a los servicios de **Functions** y **Firestore Database**. Dado que el sistema no supera los límites gratuitos, el gasto de los servicios es inexistente.

7.1.2 *INICIO DE FIRESTORE DATABASE*

Una vez mejorado el proyecto al plan **Blaze**, se iniciará **Firestore Database**. Dentro de este, se creará una colección llamada "**zonas**". Es crucial seguir la misma nomenclatura, ya que la aplicación móvil utiliza este nombre para obtener los datos durante las consultas. Firestore pedirá añadir el primer documento, cuyos requisitos serán un identificador y un campo como mínimo. El identificador será el identificador de zona (Por ejemplo, Z0000001) y en los campos se añadirán los siguientes:

- "zonaId" (Tipo String)
- "nombre" (Tipo String)
- "coordenadas" (Tipo Geopoint)

Una vez añadido el primer documento, se repetirá este proceso para todas las zonas restantes.

7.1.3 *CONFIGURACIÓN DEL SERVICIO DE AUTENTICACIÓN*

Dentro de este servicio, se solicitará añadir un proveedor nuevo al proyecto. Para el despliegue del TFG, se seleccionará **Correo electrónico/contraseña**. Una vez habilitado el método de acceso, se procederá a añadir los usuarios necesarios. En este caso, se creará un correo y una contraseña para cada uno de los nodos sensores y actuadores que se utilicen en la implementación del sistema. El formato utilizado de correo se puede encontrar en el apartado Autenticación.

7.1.4 *DESPLIEGUE DE LAS FUNCIONES*

Para el despliegue de las funciones, se necesitará **Node.js** y las herramientas **CLI de Firebase**. En el enlace (12) se muestran los pasos necesarios. Una vez obtenidas las herramientas, se deberá navegar por una ventana de comandos hasta la carpeta donde se encuentran los archivos relacionados con el proyecto. Allí se ejecutará el comando "**firebase login**" para iniciar sesión con la cuenta con la que se creó el proyecto.

Una vez iniciada la sesión, se descargarán las herramientas y emuladores de los servicios que se utilizan. Utilizando el comando "**firebase init**", se iniciará la guía de configuración para configurar los archivos del proyecto. Dado que ya se ha creado el proyecto, se seleccionará un proyecto ya existente.

La guía de configuración preguntará que servicios se utilizan y se seleccionarán **Authentication**, **Database** (Realtime), **Firestore** y **Functions**. También se marcarán los emuladores, aunque para el despliegue no son esenciales (sí lo son para el desarrollo).

Tras finalizar la guía de configuración, se tendrá como resultado una carpeta con los siguientes archivos:

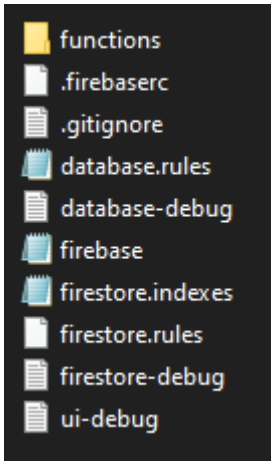


Ilustración 47 – Archivos de configuración generados por Firebase CLI.

Se copiará el archivo **index.js** en la carpeta **functions** y se reemplazará el archivo existente. Para desplegar las funciones, se ejecutará el comando "**firebase deploy --only functions**".

7.1.5 *INSTALACIÓN Y CONFIGURACIÓN DE LOS NODOS SENSORES*

Para la configuración de los nodos sensores, primero se debe obtener la dirección de la **Realtime Database** y la **API *key*** del proyecto. La dirección se puede encontrar en el servicio de Realtime y la API *key* en la configuración del proyecto en Firebase. Se reemplazará la dirección y la API *key* en el fichero **Value.h** en el Arduino IDE.

Una vez realizado el cambio, se cargará el archivo con extensión **.ino** en la placa seleccionando la placa **LONIN(WEMOS) D1 mini (clone)** y el puerto serial al cual esté conectada. Una vez cargado el archivo, se solicitará el identificador del sensor, el identificador de la zona, el SSID y la contraseña de la red WiFi, email y contraseña de **Firebase Authentication**. Si se desea borrar el archivo de configuración **config.json**, solo se tiene que mantener el botón de ***RESET*** y mientras se conecta la placa al ordenador.

Es importante destacar que existen dos archivos de configuración dependiendo del nodo sensor que se esté utilizando. Una vez completados los pasos, el nodo quedará configurado.

7.1.6 *INSTALACIÓN DE LOS CÓDIGOS ACTUADORES*

Al no tener una guía de configuración como los nodos actuadores, será necesario modificar directamente en el código principal los parámetros de configuración mencionados en el apartado anterior. La carga del archivo se efectúa de la misma forma.

7.2 Anexo II – Manual de Usuario

7.2.1 *INTRODUCCIÓN DE LA APP*

Esta aplicación permite al usuario interactuar de formar remota con los nodos actuadores y sensores de las distintas zonas. Se requiere una versión Android 5.0 (Lolipop) o superior.

7.2.2 *MENÚ PRINCIPAL*

El menú principal es la primera pantalla que se visualiza tras iniciar la aplicación. Desde aquí se puede acceder a las principales funcionalidades principales de control de temperatura, humedad y riego.

Ilustración 48 – Menú principal de la aplicación.

A continuación, se describen cada botón del menú principal:

- **Temperatura**: Su icono es un sol con un termómetro de color naranja. Al presionar este botón, se abrirá la pantalla de vista de zonas para monitorear y ajustar las configuraciones relacionadas con la temperatura.
- **Humedad**: Su icono es una gota de agua con un porcentaje de color azul claro. Al presionar este botón, se abrirá la pantalla de vista de zonas para monitorear y ajustar las configuraciones relacionadas con la humedad.
- **Riego**: Su icono es una regadera de color morado. Al presionar este botón, se abrirá la pantalla de vista de zonas para monitorear y ajustar las configuraciones relacionadas con el riego.

7.2.3 *VISTA DE ZONAS POR VARIABLE*

La vista de zonas es una interfaz clave de la aplicación que facilita la monitorización y gestión de las zonas para cada variable ambiental (temperatura,

humedad y riego). Esta vista es similar para las tres categorías y permite observar rápidamente el estado de cada zona y acceder a su configuración.

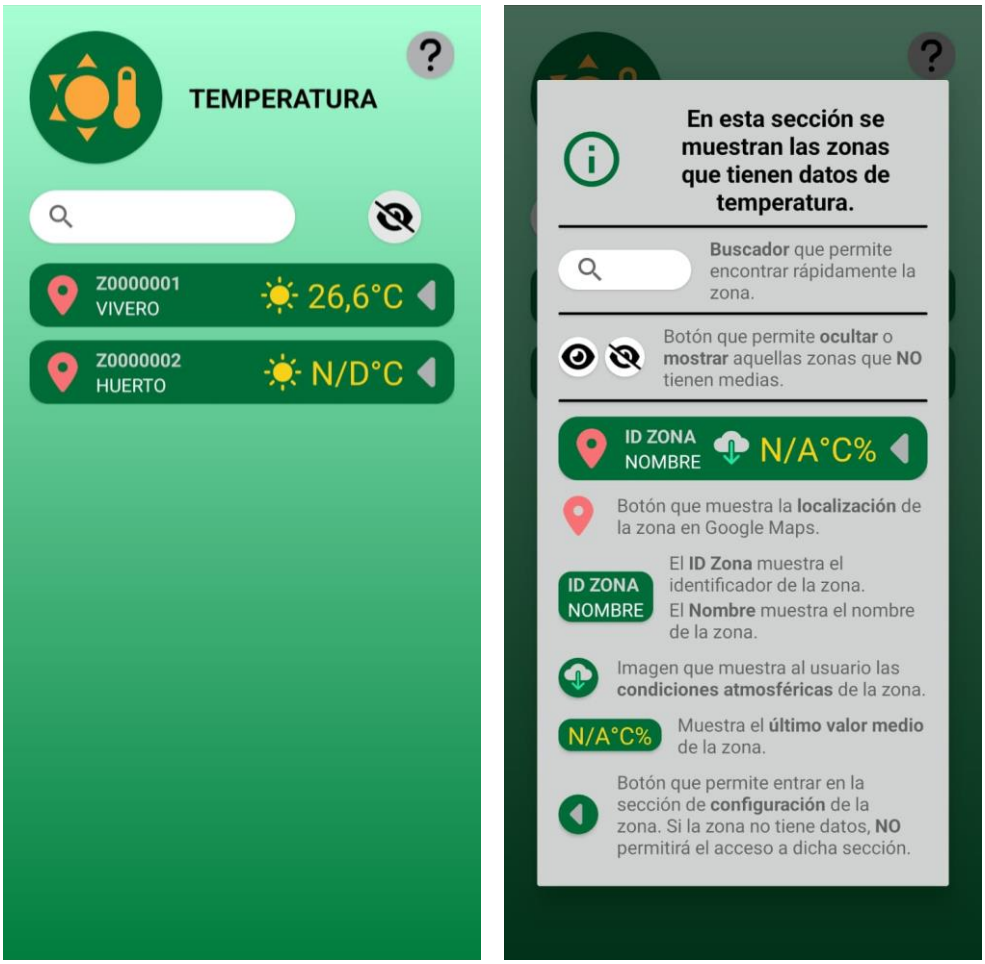


Ilustración 49 – Vista de zonas de temperatura y su ayuda desplegable.

A continuación, se detalla la vista de zonas:

- **Ícono y Título**: Indican la variable seleccionada.
- **Botón de ayuda**: Al pulsarlo, desplegará una ventana que indica el funcionamiento de la vista de zonas.
- **Barra de búsqueda**: Permite encontrar una zona determinada según su nombre.
- **Botón ocultar**: Permite ocultar o mostrar aquellas zonas que no tienen medias de la variable ambiental que se desea configurar. Por defecto, se encuentran ocultas.
- **Listado de zonas**: Las zonas se muestran de forma vertical.
    - **Botón de localización**: Un icono de ubicación que, al pulsarlo, abrirá la localización en Google Maps.

- **Identificador** y **Nombre** de zona: Muestra el identificador de zona y debajo un nombre descriptivo.
- **Estado del tiempo**: Un icono que representa el estado actual del tiempo en la zona.
- **Último valor medio**: El ultimo valor medio de la zona para la variable seleccionada (temperatura, humedad o humedad de suelo). Si no existen datos, se mostrará “N/C”.
- **Botón de acceso a configuración**: Permite acceder a la sección de configuración de variable ambiental y zona seleccionada. Si no existen datos (N/C), no permitirá el acceso.

7.2.4 *CONFIGURACIÓN DE ZONA*

Esta vista facilita la configuración e interacción con los nodos sensores y actuadores asociados a la variable ambiental y la zona que se desea configurar.

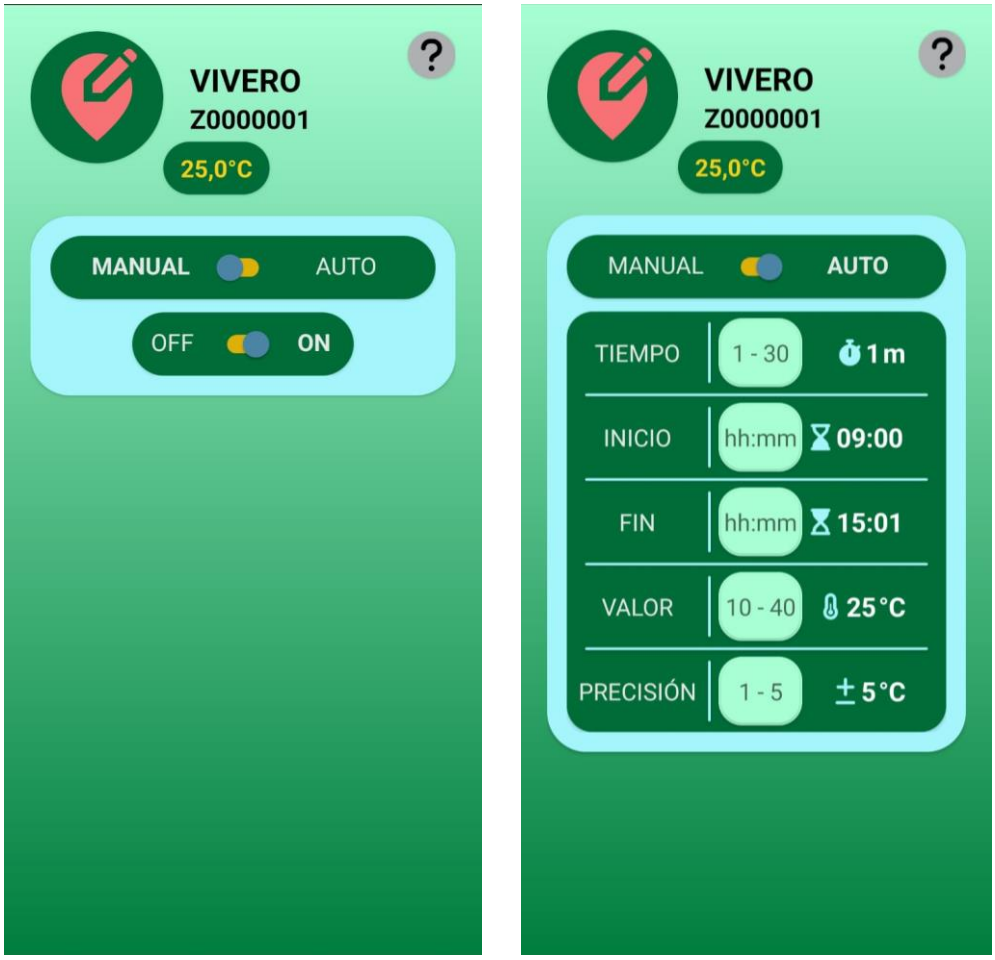


Ilustración 50 – Pantalla de configuración de zona (vista manual y vista automático).

A continuación, se detalla la vista de configuración de zona:

MODO MANUAL

- Switch **Manual / Auto**: Permite cambiar entre la vista **manual** y **automático**.
- Switch **Off / On**: Indica al nodo actuador asociado a la variable ambiental, el **apagado** o **encendido** del sistema.

MODO AUTOMÁTICO

- Switch **Manual / Auto**: Permite cambiar entre la vista **manual** y **automático**.
- **Tiempo**: Permite especificar la **duración** del siguiente ***Deep-Sleep*** para los nodos sensores asociados a la variable ambiental. Limita la entrada de un valor dentro de un rango mínimo y máximo.
- **Inicio**: Indica la **hora** del día en la que el nodo actuador puede **comenzar a operar**. Requiere que el usuario utilice un formato específico de entrada.
- **Fin**: Indica la **hora** del día en la que el nodo actuador debe **cesar** sus operaciones. Requiere que el usuario utilice un formato específico de entrada.
- **Valor**: Entrada que indica al nodo actuador el **valor objetivo** para la variable ambiental. Por ejemplo, 24 ºC.
- **Precisión**: Entrada que especifica al nodo actuador el **rango** dentro del cual puede variar el valor objetivo. Por ejemplo, ±3 ºC (entre 21 y 27 ºC).

Cada una de las entradas, tiene un campo a la derecha que se actualiza en tiempo real cuando una de estas variables cambia en Realtime Database. Para notificar al usuario sobre el cambio, el icono de Firebase parpadeará brevemente junto a ella.

Ilustración 51 – Ayuda desplegable de la vista configuración de zona.

Del mismo modo que la sección anterior, existe un **botón de ayuda** que muestra información del uso de la sección de configuración.