ArduSmartHome.

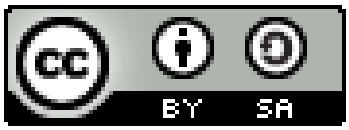
Diseño e implementación de red de sensores inalámbricos para el control domótico de una vivienda basado en Arduino.

Autor: Miguel Ángel Sánchez Muñoz
Plan de Estudios: Grado Ingeniería Informática
Área del trabajo final: Arduino

Consultor: José López Vicario
Profesor: Pere Tuset Peiró

11 de Junio del 2.017

Licencia



A Sara

FICHA DEL TRABAJO FINAL

Título del trabajo:	***ArduSmartHome*** ***Diseño e implementación de red de sensores inalámbricos para el control domótico de una vivienda basado en Arduino.***
Nombre del autor:	*Miguel Angel Sánchez Muñoz*
Nombre del consultor/a:	*José López Vicario*
Nombre del PRA:	*Pere Tuset Peiró*
Fecha de entrega (mm/aaaa):	06/2017
Titulación::	*Grado Ingeniería Informática*
Área del Trabajo Final:	*Arduino*
Idioma del trabajo:	***Castellano***
Palabras clave	*Arduino, MQTT, Node-RED*

Resumen del Trabajo:

El proyecto ArduSmartHome consiste en la elaboración de un sistema de control domótico que permite obtener datos ambientales de diferentes sensores, distribuidos en diferentes espacios dentro de una vivienda familiar.

Con la ayuda de una plataforma web basada en Node-RED se monitoriza la información de los diferentes sensores.

El sistema electrónico se compone de varios dispositivos Arduino con sensores para medir condiciones ambientales de iluminación, temperatura, humedad y ruido, mas un dispositivo Arduino que realiza las funciones de nodo central, estableciendo y coordinando las comunicaciones del sistema.

El sistema establece su propia red Wifi local a la que se conecta cada dispositivo Arduino ubicado en un espacio diferente de la vivienda. Esto consigue una independencia en conectividad y operatividad, no interfiriendo directamente en las redes del cliente final.

Abstract:

The ArduSmartHome project consists of the elaboration of a domotic control system that allows to obtain environmental data of different sensors, distributed in different spaces within a familiar home.

With the help of a web platform based on Node-RED, the information of the different sensors is monitored.

The electronic system consists of several Arduino devices with sensors to measure ambient conditions of illumination, temperature, humidity and noise, plus an Arduino device that performs the functions of central node, establishing and coordinating the communications of the system.

The system establishes its own local Wifi network to which it connects each Arduino device located in a different space of the home. This achieves an independence in connectivity and operability, not interfering directly in the networks of the final client.

Índice

Lista de ilustraciones

Lista de tablas

1. Introducción

1.1. Contexto y justificación del Trabajo

Se define a la Domótica como un grupo de sistemas relacionados, que al trabajar en conjunto, son capaces de automatizar una vivienda. Los ámbitos de actuación de un sistema domótico en términos generales se agrupan en cinco: ahorro energético, confort, seguridad, comunicaciones y accesibilidad.

Cada ámbito merece un tratamiento particular ya que las aplicaciones son diversas, dentro del ámbito de este proyecto se va a abarcar principalmente el ahorro energético y el confort dentro de una vivienda familiar, no cerrando el proyecto a futuras mejoras incluyendo otros posibles ámbitos de actuación.

Siendo una tecnología de futuro, el mercado de la Domótica esta copado por soluciones cerradas que dan poco margen de modificación y adaptabilidad. Aunque se establecen estándares de operabilidad y comunicación, estas soluciones cerradas producen una inoperabilidad entre ellas, causada principalmente por los propios fabricantes.

Desde hace unos años, y en gran parte influenciados por el movimiento Maker, han surgido una serie de alternativas libres, que buscan hacerse un hueco dentro del mercado domótico. Estas opciones intentan integrar bajo una misma interfaz diversos elementos que tiene cualquier instalación domótica haciendo uso de APIs de Internet y gran parte de los dispositivos inteligentes. Algunas de estas alternativas libres son Domoticz, Jeedom o OpenHAB.

Tras analizar las diferentes soluciones ya disponibles, se llego a la conclusión del bajo o inexistente número de sistemas domóticos capaces de funcionar sin el uso de equipos informáticos adicionales. La gran mayoría de los nodos receptores de información requieren de ser instalados en un ordenador para poder interpretar los datos, por esta razón se planteó la posibilidad de desarrollar un sistema autónomo basado en la plataforma Arduino[1], totalmente operativo he independiente de otras plataformas como el PC.

1.2. Objetivos del Trabajo

El objetivo principal del proyecto consiste en el diseño e implementación de una red de comunicaciones Machine-to-Machine (M2M) entre distintos sensores/actuadores utilizando el protocolo MQTT[2] (Message Queue Telemetry Transport) y permitiendo su monitorización/interacción mediante una interfaz web.

Esta idea principal se desglosa en los siguientes objetivos que trazan el desarrollo del proyecto:

- Integrar varios sensores/actuadores y dotar de conectividad inalámbrica a una placa de desarrollo Arduino[1] UNO R3.
- Instalación del Broker MQTT[2] Mosquitto[3] en una placa de desarrollo Arduino[1] YUN o compatible.

- Establecer una red de comunicaciones Machine-to-Machine (M2M) utilizando el protocolo MQTT[2] entre todos los dispositivos.
- Implementar una plataforma de acceso web que permita la interpretación de los diferentes datos obtenidos de los sensores y la interacción con el usuario.

1.3. Enfoque y método seguido

El enfoque del proyecto ha sido desde sus comienzos una idea basada en el aprendizaje y estudio de la tecnología Arduino[1]. Para poder llevar acabo el mismo, se ha intentado utilizar en la totalidad del proceso todos los recursos que proporciona la plataforma Arduino[1], tales como: librerías, ejemplos, códigos o ideas existentes dentro de la comunidad Arduino[1].

La metodología de desarrollo e implementación ha sido durante todo el proceso un esquema basado en etapas con el seguimiento del diagrama de Gantt desarrollado en planificación del proyecto durante la PEC 1. Con ello se ha conseguido un trabajo equilibrado, en el cual las dificultades abordadas han sido solucionadas en cada momento. Para la constitución de la memoria ha resultado de gran utilidad hacer una recopilación de todas las fuentes utilizadas durante el proceso de investigación y desarrollo del TFG.

La programación ha sido desarrollada siguiendo un esquema estructural, utilizando funciones que modularizan el trabajo dentro del código y facilitan la depuración de errores además de la implantación de nuevas funciones. Esta modularidad permite la activación y desactivación de funciones dentro del código en tiempo de compilación, así ofreciendo una mayor adaptación de los Sensores inalámbricos Clientes MQTT[2] a la ubicación final en la que se sitúe cada uno. No se ha contemplado el desarrollo de clases o librerías específicas ya que el código existente en cada función no resulta demasiado extenso.

1.4. Planificación del Trabajo

La planificación de trabajo del proyecto se desarrollara principalmente en tres fases diferenciadas, finalizando cada una con la entrega de cada PEC:

- Primera fase, PEC 2: Diseño y desarrollo del hardware. Diseño y desarrollo del software empotrado en los Sensores inalámbricos. Test hardware y software.
- Segunda fase, PEC 3: Diseño y desarrollo de todos los componentes que forman la red MQTT[2]. Diseño y desarrollo de la Web Dashboard[7]. Test global del sistema.
- Tercera fase: Documentación de la memoria y presentación final del proyecto.

1.4.1 Diagrama de Gantt

En el siguiente diagrama de Gantt se realiza la representación temporal de las tareas llevadas a cabo en el desarrollo del proyecto [*Ilustración 1*]. Las tareas están distribuidas en función a las necesidades de cada PEC, utilizando cada entrega como punto de control.

Como podemos observar, algunas de las tareas necesitan de trabajos o de otras tareas previas para ser iniciadas, por lo que no se podrán realizar hasta que sus predecesoras sean finalizadas. Del mismo modo, algunas de las tareas son solapadas ya que pueden ser ejecutadas al mismo tiempo, lo cual genera un mayor aprovechamiento del tiempo.

No se han detectado grandes retrasos en cada una de las partes. Cada hito fijado en la planificación ha sido alcanzado en el tiempo estimado para ello, manteniendo una concurrencia temporal correcta, no siendo reseñable ninguna variación dentro del esquema propuesto.

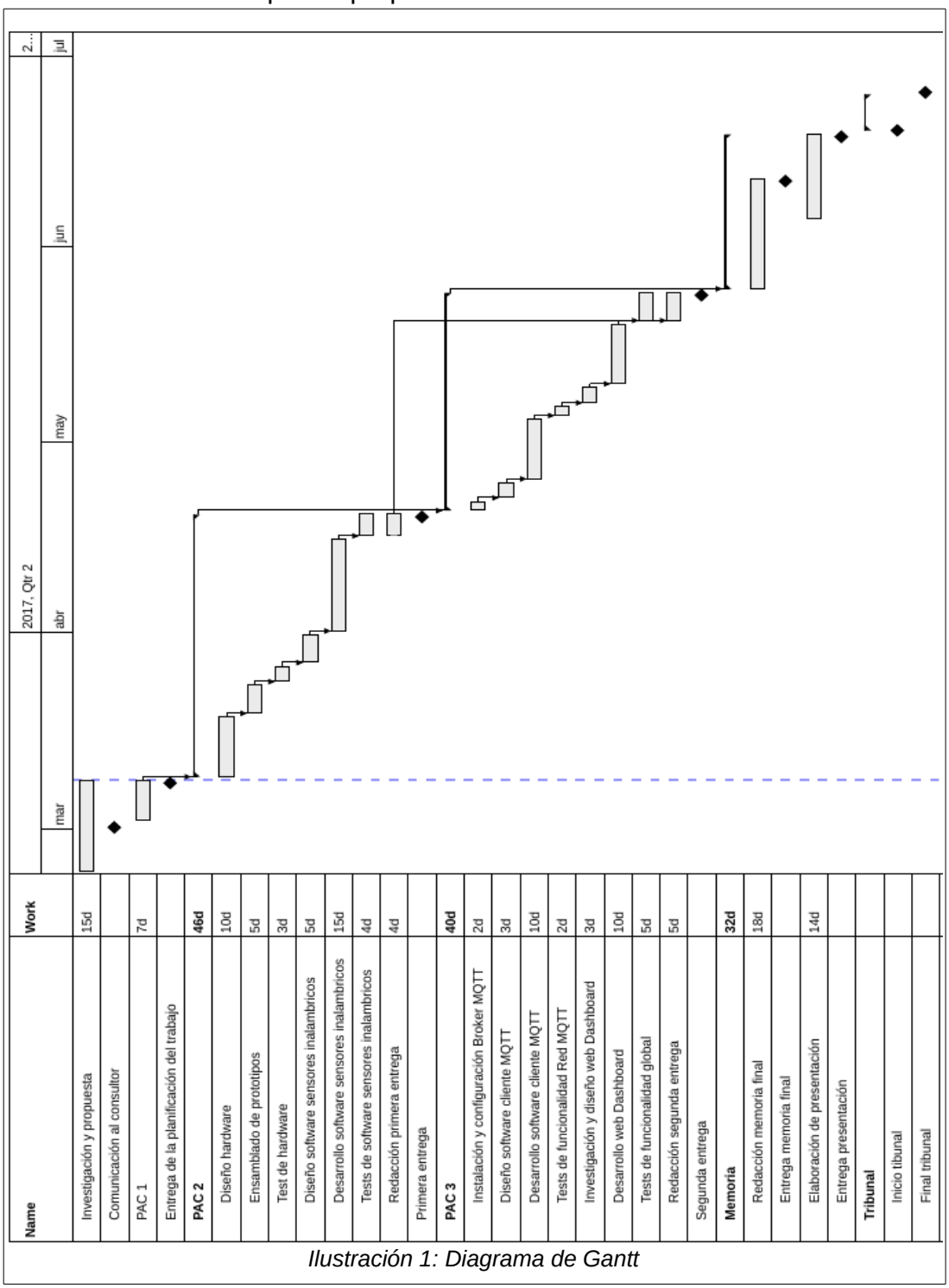


Ilustración 1: Diagrama de Gantt

WBS	Name	Start	Finish	Duration
1	Investigación y propuesta	feb 22	mar 8	15d
2	Comunicación al consultor	mar 1	mar 1	N/A
3	PAC 1	mar 2	mar 8	7d
4	Entrega de la planificación del trabajo	mar 8	mar 8	N/A
5	**PAC 2**	**mar 9**	**abr 19**	**42d**
5.1	Diseño hardware	mar 9	mar 18	10d
5.2	Ensamblado de prototipos	mar 19	mar 23	5d
5.3	Test de hardware	mar 24	mar 26	3d
5.4	Diseño software sensores inalambricos	mar 27	mar 31	5d
5.5	Desarrollo software sensores inalambricos	abr 1	abr 15	15d
5.6	Tests de software sensores inalambricos	abr 16	abr 19	4d
5.7	Redacción primera entrega	abr 16	abr 19	4d
6	Primera entrega	abr 19	abr 19	N/A
7	**PAC 3**	**abr 20**	**may 24**	**35d**
7.1	Instalación y configuración Broker MQTT	abr 20	abr 21	2d
7.2	Diseño software cliente MQTT	abr 22	abr 24	3d
7.3	Desarrollo software cliente MQTT	abr 25	may 4	10d
7.4	Tests de funcionalidad Red MQTT	may 5	may 6	2d
7.5	Investigación y diseño web Dashboard	may 7	may 9	3d
7.6	Desarrollo web Dashboard	may 10	may 19	10d
7.7	Tests de funcionalidad global	may 20	may 24	5d
7.8	Redacción segunda entrega	may 20	may 24	5d
8	Segunda entrega	may 24	may 24	N/A
9	**Memoria**	**may 25**	**jun 18**	**25d**
9.1	Redacción memoria final	may 25	jun 11	18d
9.2	Entrega memoria final	jun 11	jun 11	N/A
9.3	Elaboración de presentación	jun 5	jun 18	14d
9.4	Entrega presentación	jun 18	jun 18	N/A
10	**Tribunal**	**jun 19**	**jun 25**	**6d**
10.1	Inicio tibunal	jun 19	jun 19	N/A
10.2	Final tribunal	jun 25	jun 25	N/A

Ilustración 2: Planificación del trabajo.

1.5. Breve sumario de productos obtenidos

Como resultado del proyecto se ha obtenido como producto un sistema de control domótico autónomo formado por dos tipos de equipos o subproductos:

- Servidor Broker MQTT[2]. Nodo central del sistema, encargado de las comunicaciones y de la interfaz de usuario Dashboard[7] [*Ilustración 3*].

Ilustración 3: Instalación de Servidor Broker MQTT junto a switch domestico

- Sensor inalámbrico Cliente MQTT[2]. Para la demo del proyecto se han desarrollado dos nodos que montan varios tipos de sensores junto con un modulo de comunicaciones ESP8266[4] y una batería de alimentación externa. [*Ilustración 4*] [*Ilustración 5*].

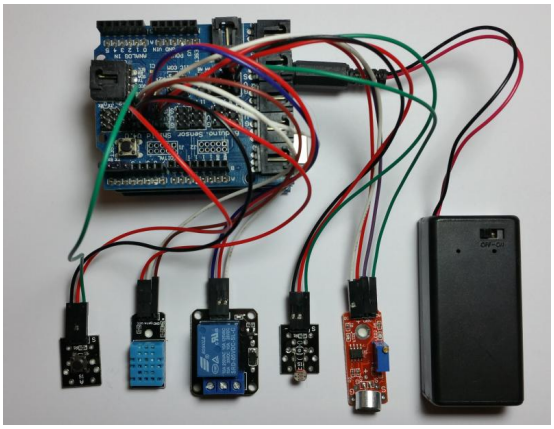


Ilustración 4: Sensor inalambrico

Ilustración 5: Instalación de sensor inalambrico en caldera.

1.6. Breve descripción de los otros capítulos de la memoria

Los capítulos de esta memoria se encuentran distribuidos de la siguiente manera:

- Hardware. Describe el diseño y el desarrollo del hardware que compone los prototipos que forman el sistema.
- Software. Describe el diseño y el desarrollo del software que compone todo el sistema.
- Valoración económica. Valoración del coste económico del proyecto y de los materiales empleados.
- Conclusiones. Alberga las conclusiones obtenidas del desarrollo del proyecto, propuesta de mejoras al proyecto y una valoración personal.
- Anexos. Información complementaria a la memoria.

2. Hardware.

El diseño del hardware que compone el proyecto se ha basado en la idea principal de la creación de una plataforma compuesta en su totalidad por hardware Arduino[1] y compatible, totalmente operativa he independientemente de otras plataformas, como puede ser la necesidad de utilizar un ordenador personal como servidor MQTT[2].

La base hardware de los prototipos esta formada, tanto para la parte servidor como para los clientes remotos, de una placa Arduino[1] UNO R3 [*Ilustración 6*]. Las especificaciones técnicas de la placa Arduino[1] UNO R3 se exponen en el Anexo 1**.**

Ilustración 6: Arduino UNO R3

En los capítulos que integran este apartado se muestra las distintas partes del hardware que componen cada prototipo y la presentación de un posible ensamblado final.

2.1. Servidor Broker MQTT.

El hardware del prototipo de servidor Broker MQTT[2] esta compuesto por dos niveles dentro de la plataforma de prototipado Arduino[1], Estos niveles se componen por una placa Arduino[1] UNO R3 como base, junto con un Dragino[5] Yun Shield v2.4.

Básicamente, la unión de Arduino[1] UNO R3 junto con el Dragino[5] Yun Shield igualan a la placa Arduino[1] Yun oficial [*Ilustración 7*], pero la independencia que ofrece el uso del Dragino[5] Yun Shield lo convierten en una solución más flexible, ya que este Shield puede acoplarse y trabajar con otras placas de la familia Arduino[1] como la Arduino[1] Duemilanove, Arduino[1] Leonardo o Arduino[1] Mega.

Ilustración 7: Comparativa Arduino[1] Yun

El Dragino[5] Yun Shield extiende las características de la placa Arduino[1] con el poder de un sistema basado en Linux que permite conexiones y aplicaciones de red avanzadas.

La configuración del Dragino[5] Yun Shield es sencilla gracias al Yun Web Panel. El panel Web le permite administrar sus preferencias del Shield y cargar el programa diseñado (en formato .hex) en la placa Arduino[1]. Dragino[5] Yun Shield utiliza la librería Bridge y, por lo tanto, amplía las capacidades de la placa utilizando el procesador Linux, de la misma manera que la placa Arduino[1] Yun original.

Como todas las placas que forman el ecosistema Arduino[1], cada elemento de la plataforma como son el hardware, software y documentación, está libremente disponible y de código abierto. Esto significa que usted puede aprender exactamente cómo se hace y utilizar su diseño como el punto de partida para sus propios proyectos.

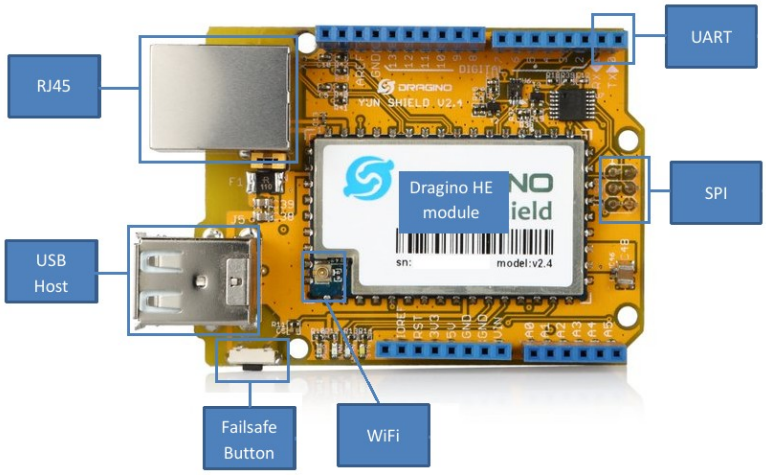


Ilustración 8: Dragino Yun Shield v2.4

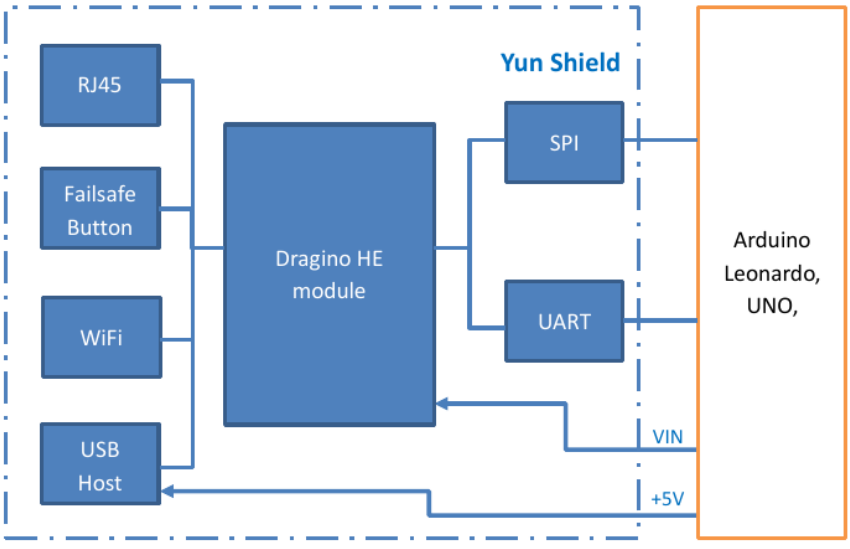


Ilustración 9: Diagrama de bloques Dragino Yun Shield

A parte de la mayor flexibilidad que se obtiene del uso de este Shield, se ha elegido el Dragino[5] Yun Shield en el desarrollo del proyecto por ser una solución mas económica a la placa Arduino[1] Yun original. Esta diferencia económica es bastante sustancial, ya que solo la placa Arduino[1] Yun duplica el precio del Dragino[5] Yun Shield junto con una placa Arduino[1] UNO R3.

Ilustración 10: Montaje Servidor Broker MQTT

Las características y especificaciones técnicas del Dragino[5] Yun Shield v2.4 se exponen en el Anexo 1**.**

2.2. Sensor inalambrico, Cliente MQTT.

El hardware de cada uno de los Sensores inalambricos Cliente MQTT[2] esta compuesto por tres niveles de la plataforma de prototipado Arduino[1]. Estos niveles los componen una placa Arduino[1] UNO R3 como base, junto con un Arduino[1] ESP8266[4] WiFi Shield v1.0 y un Arduino[1] Sensor Shield v4.0.

2.2.1 Arduino ESP8266 WiFi Shield v1.0

El Arduino[1] ESP8266[4] WiFi Shield dispone de un modulo ESP-12 ESP8266[4] que permite conectar la placa Arduino[1] UNO, o cualquier otro microcontrolador, a una red Wifi estandard IEEE 802.11b/g/n domestica, sin necesidad de establecer otros protocolos de comunicación como puede ser el caso de los dispositivos Zigbee.

La ventaja de usar este Shield en lugar de otras soluciones del modulo ESP8266[4], es que integra la conversion de voltage a 3.3v TTL, ya que esta es la tension de trabajo del modulo ESP8266[4], tanto de alimentación como de comunicación UART.

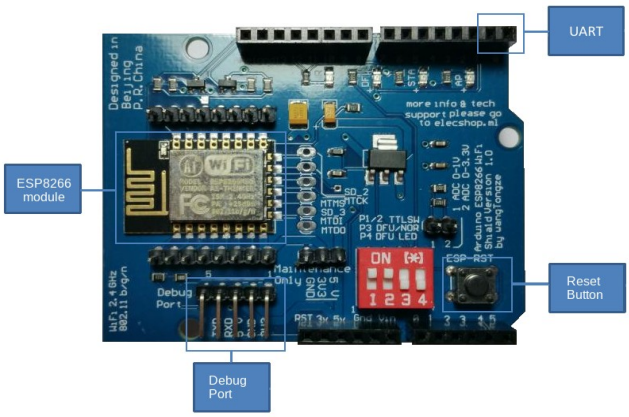


Ilustración 11: Arduino ESP8266 WiFi Shield v1.0

La comunicación entre la placa Arduino[1] UNO y el modulo ESP8266[4] se realiza, por defecto, mediante el puerto UART (RX0, TX1).

Esta conexión UART impide una correcta programación de nuestro microcontrolador, ya que al mantener el puerto UART en uso por parte del modulo ESP8266[4], fuerza la necesidad de una continua conexión y desconexión el Arduino[1] ESP8266[4] WiFi Shield, fisicamente o mediante los switch de los que dispone el propio Shield, cada vez que se graba el programa en la memoria del microcontrolador Arduino[1] UNO.

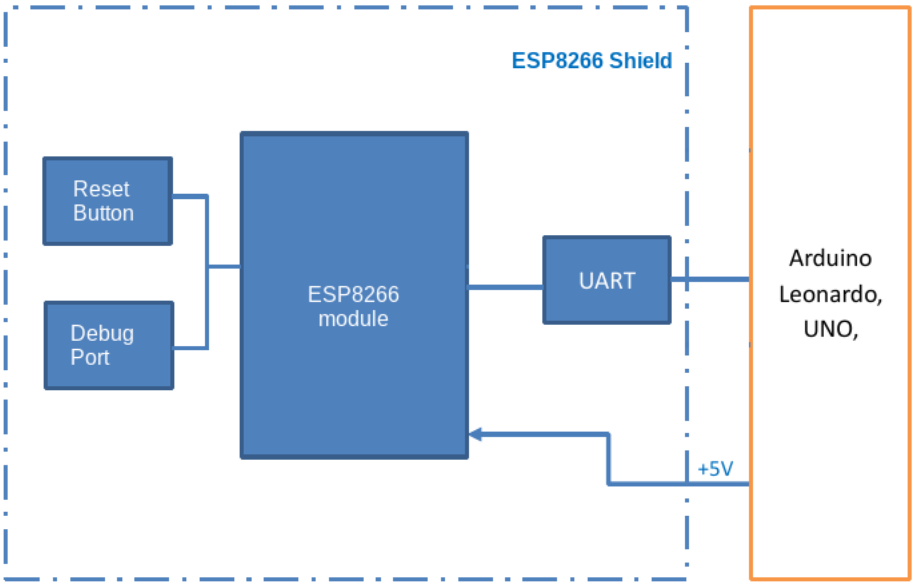


Ilustración 12: Diagrama de bloques ESP8266 Wifi Shield

Para evitar dicho trastorno continuo durante el desarrollo, se ha eliminado la conexión directa entre el Shield y el Arduino[1] UNO, estableciendo un puente entre el pin 0 del Shield con el pin 2 de Arduino[1] y el pin 1 del Shield con el pin 3 de Arduino[1].

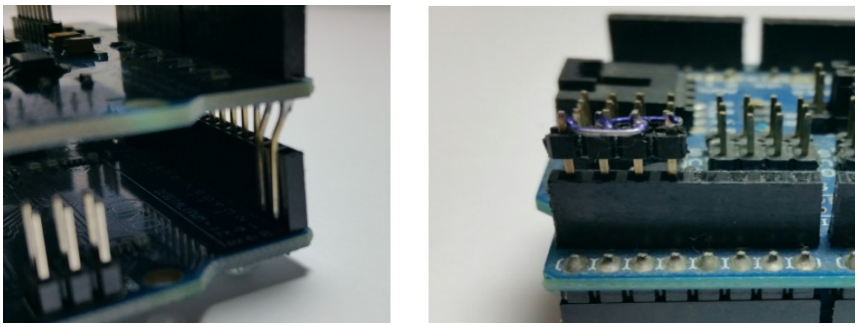


Ilustración 13: Puente UART, SoftwareSerial

Una vez realizado estas modificaciones en el diseño, la comunicación UART con el Shield se ha de realizar mediante la librería *SoftwareSerial*[10] disponible en la plataforma Arduino[1]. Esta libreria utiliza los pines digitales 2 y 3 de la placa Arduino[1] UNO como pines de comunicación UART.

2.2.2 Arduino Sensor Shield v4.0.

El Arduino[1] Sensor Shield es una placa que facilita el conexionado de los sensores a la hora de realizar el prototipado. Simplemente es una extenxión de las conexiones de entrada/salida de las que dispone el Arduino[1] UNO. [*lustración 14*].

Todas las conexiones externas con los diferentes sensores se realizaran a traves de este Shield mediante cablecillos de conexionado hembra/hembra.

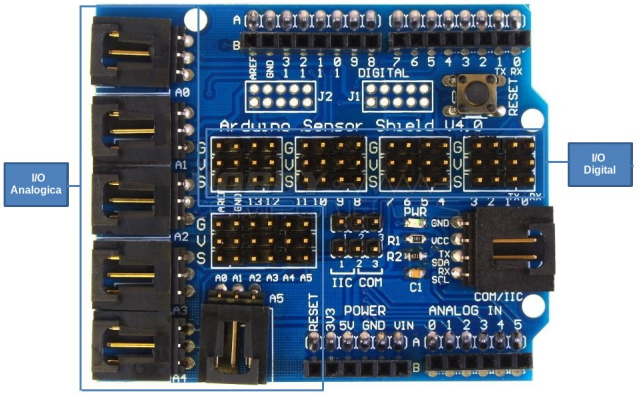


Ilustración 14: Arduino Sensor Shield v4.0

En la *Ilustración 15* se muestra el resultado del montaje de tres niveles de la placa Arduino[1] UNO R3 junto a los dos Shields.

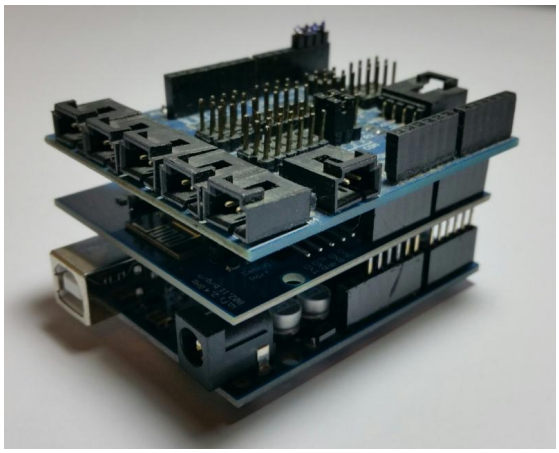
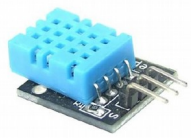
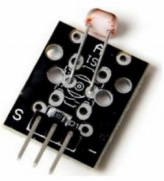


Ilustración 15: Cliente Inalambrico MQTT (sin sensores)

2.2.3 Sensores y actuadores.

En la siguiente tabla se muestran los diferentes sensores y actuadores que se han integrado en el sistema. Las caracteristicas tecnicas y de conexionado de cada sensor se describe en el Anexo 1**.**

	Sensor de Temperatura y Humedad DHT11.
	Sensor LDR.

	Sensor de Micrófono Sensible.
	Interruptor tipo Botón.
	Modulo Relé.

En la *ilustración 16* se muestra una representación del montaje final del prototipo del Sensor inalámbrico Cliente MQTT[2], una vez realizado el ensamblado y conexionado a los sensores y actuadores.

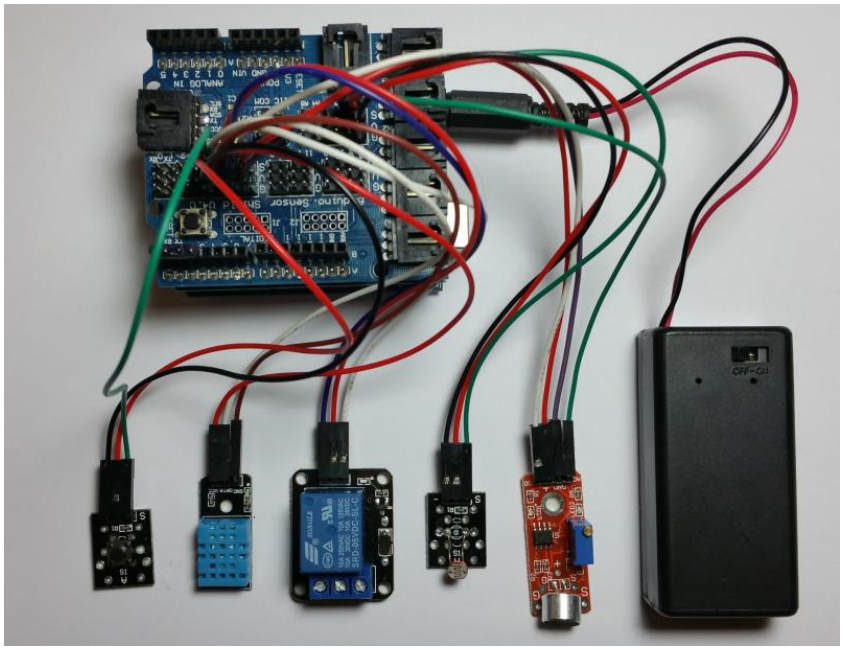


Ilustración 16: Sensor inalambrico

En esta representación del montaje se le ha añadido una batería externa para dotarlo de mayor flexibilidad y movilidad. No siendo esta necesaria si en el montaje final se dispone de una toma de corriente, en la que se realizaría la conexión eléctrica haciendo uso de un adaptador de tensión, o si el dispositivo que se quiere "domotizar" dispone que una fuente de alimentación que permita la alimentación del Sensor inalámbrico Cliente MQTT[2].

3. Software.

3.1. Servidor Broker MQTT.

El servidor Broker MQTT[2] es el nodo central de todo el sistema que forma el proyecto. Es el encargado de las siguientes cinco tareas principalmente:

- Punto de acceso Wifi. Establece una red inalámbrica en la que se conecten los sensores inalambricos.
- Puente Ethernet. Mediante la conexion Ethernet diponible en el Dragino[5] Yun Shield, interconecta el sistema con la red domestica del cliente.
- Broker MQTT[2]. Encargado de gestionar la red MQTT y de transmitir los mensajes.
- Cliente Node-RED[6]. Implementa la logica que gestiona los diferentes datos obtenidos de los sensores.
- Dashboard[7] de interaccion con el usuario. Sevidor Web que permite la interaccion del usuario final con el sistema.

3.1.1 Red de comunicaciones inalámbrica.

El sistema se ha diseñado con la idea de dotarlo de una conectividad independiente, pudiendo operar si la necesidad de que exista una red de comunicaciones previa en la ubicación donde se quiera instalar. El servidor Broker MQTT[2] es el nodo central del sistema encargado de aportar la conectividad necesaria para el funcionamiento global del sistema.

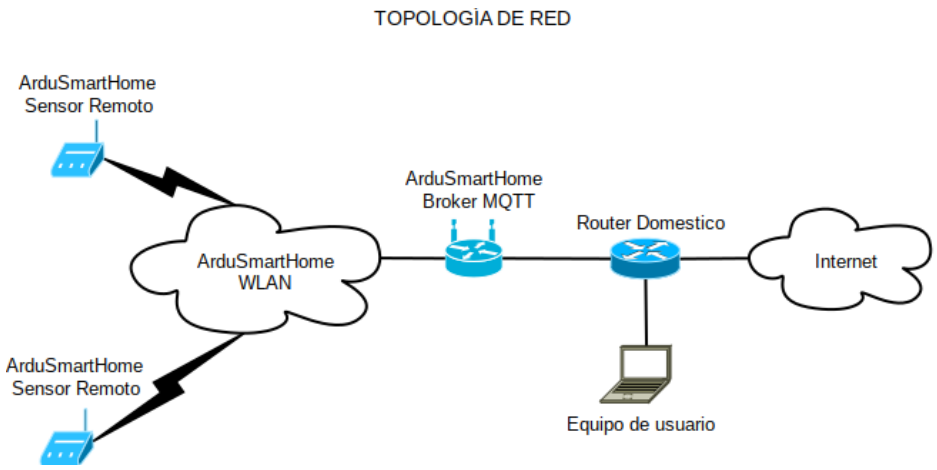


Ilustración 17: Topología de red.

Como se muestra en la *Ilustración 17*, para conseguir la topología de red deseada en el proyecto, se requiere de la configuración de dos redes en el Servidor Broker MQTT[2], una red WLAN interna del sistema para la comunicación de los Sensores inalámbricos Clientes MQTT[2] con el Broker MQTT[2] y una red Ethernet para la comunicación del conjunto del sistema con la red domestica del cliente final.

El Dragino[5] Yun Shield dispone de un entorno Web Yun integrado que permite realizar la configuración de las redes de comunicación de una forma rápida y sencilla. [*Ilustración 18*].

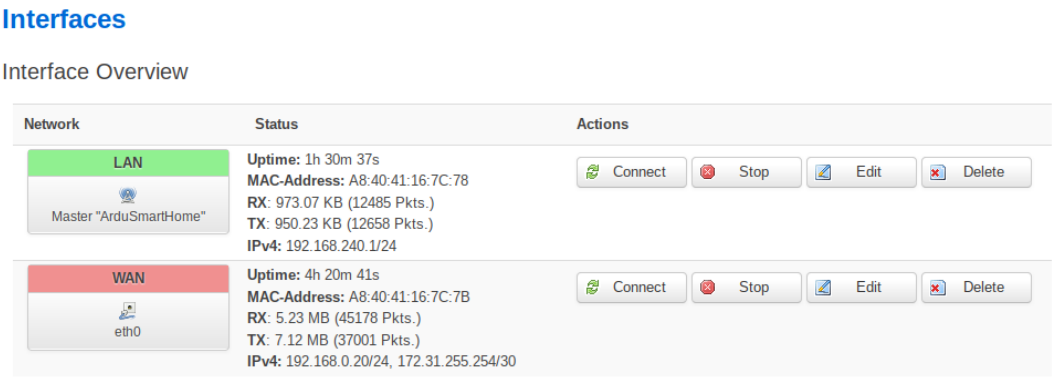


Ilustración 18: Configuración de Red

3.1.2 Software Broker MQTT, servidor Mosquitto.

MQTT[2] es un protocolo creado a finales de los años 90 por Andy Stanford-Clark de IBM y Arlen Nipper de Arcom Control Systems. Fue diseñado para la comunicación de máquina a máquina (M2M), y se ejecuta a través de TCP/IP. La versión actual de MQTT[2] es la 3.1.1. Sus objetivos principales son:

- Evitar el continuo sondeo de sensores, permitiendo que los datos sean enviados a las partes interesadas en el momento en el que esté listo.

- Ligero, para que pueda ser utilizado en conexiones de que disponen de un ancho de banda muy bajo.

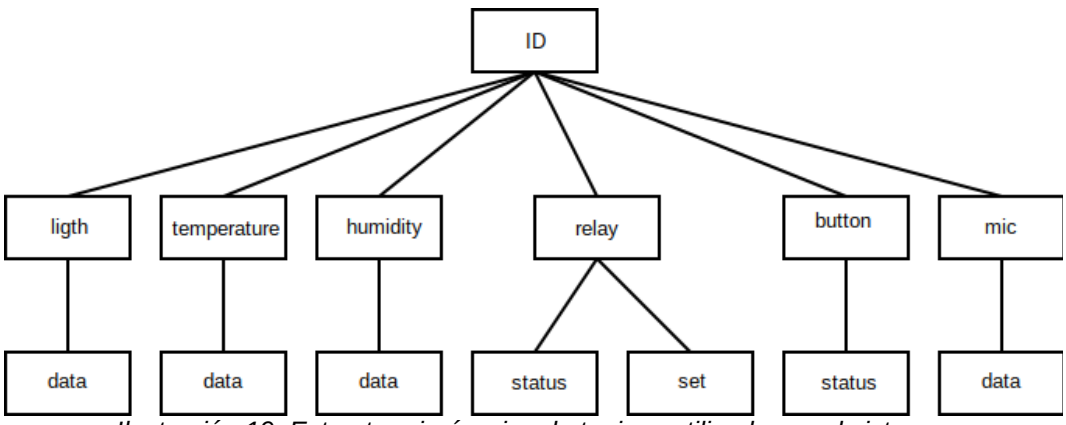


Ilustración 19: Estructura jerárquica de topicos utilizada por el sistema.

MQTT[2] es un protocolo cliente-servidor, para el cual se necesita un servidor para distribuir mensajes entre las aplicaciones cliente. También se utiliza el paradigma publicar-suscribirse, en lugar de poner en cola, una aplicación receptora se suscribe a temas/tópicos de interés y la aplicación de envío publica mensajes en los temas/tópicos. El editor está desvinculado de cualquier suscriptor no tiene conocimiento de si alguna aplicación está recibiendo sus mensajes.

Mosquitto[3] proporciona una implementación de servidor ligero del protocolo MQTT[2], escrita en C. La razón para escribirlo en C es permitir que el servidor funcione en máquinas que ni siquiera tienen capacidad para ejecutar una JVM. Los sensores y actuadores, que a menudo son las fuentes y destinos de los mensajes MQTT[2], pueden ser muy pequeños y carentes de potencia. Esto también se aplica a las máquinas embebidas a las que están conectados, que es donde se puede ejecutar Mosquitto[3] como el Dragino[5] Yun Shield.

Normalmente, la implementación actual de Mosquitto[3] de Roger Light tiene un ejecutable del orden de 120kB que consume alrededor de 3MB de RAM con 1.000 clientes conectados. Ha habido informes de exitosas pruebas con 100.000 clientes conectados a tasas de mensajes modestas.

El servidor Mosquitto[3] tiene las siguientes características, que no se describen en la especificación MQTT[2]:

- Un puente MQTT[2], para permitir que Mosquitto[3] se conecte a otros servidores MQTT[2].
- La capacidad de asegurar las comunicaciones con el uso de SSL/TLS.
- Autorización de usuarios, capacidad de restringir el acceso de los usuarios a los tópicos MQTT[2].

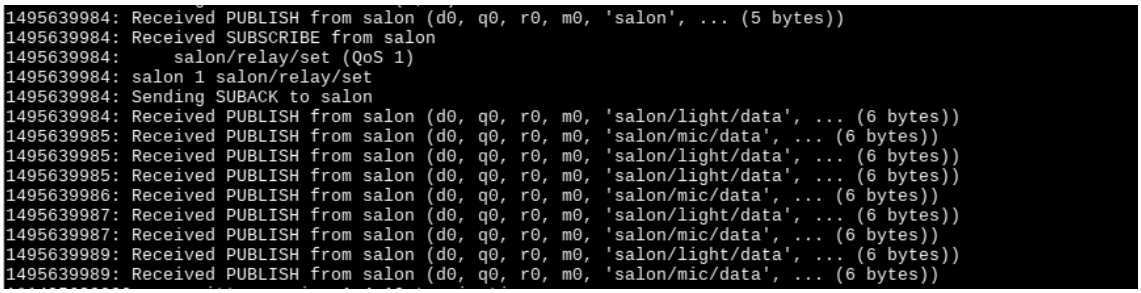


Ilustración 20: Muestra de funcionamiento del servidor Mosquitto

La instalación del software que compone el servidor Mosquitto[3] se describe en el Anexo 2.

3.1.3 Node-RED. Web Dashboard.

El desarrollo de la interfaz de usuario Web o Dashboard[7] se ha realizado con el motor de flujos Node-RED[6]. Este motor permite definir gráficamente flujos de servicios a través de protocolos estándares como REST, MQTT[2], Websocket, AMQP. Se trata de una herramienta visual muy ligera, programada en NodeJS y que puede ejecutarse desde dispositivos tan limitados como en este caso un Dragino[5] Yun Shield de Arduino[1].

El editor de flujos de Node-RED[6] consiste en una sencilla interfaz Web, en la que arrastrando y conectando nodos entre sí es posible definir un flujo que ofrezca un servicio final.

Ilustración 21: Nodos Node-RED

Node-RED[6] se integra en la red MQTT[2] como un cliente más mediante los nodos MQTT[2] de los que dispone la aplicación. Para realizar la suscripciones MQTT[2] se utilizan los nodos input MQTT[2], para subscribirse a los tópicos en los que los Sensores inalámbricos Cliente MQTT[2] publican los datos y las publicaciones MQTT[2] se realizan mediante los nodos output MQTT[2], publicando en los tópicos a los que se encuentran suscritos los Sensores inalámbricos Clientes MQTT[2]. La publicación de tópicos permite la interacción con los actuadores instalados en los Sensores inalámbricos Clientes MQTT[2]. La instalación de Node-RED[6] se detalla en el Anexo 3.

Para la demo de muestra del proyecto se han diseñado tres flujos de configuración que corresponden con tres supuestas zonas de una vivienda.

- **Zona de control global de temperatura**. [*Ilustración 22*] Este flujo de configuración gestiona el control del sensor inalámbrico instalado en la caldera de calefacción de la vivienda, gobernando su encendido y apagado remoto. El calculo de la temperatura global se realiza dentro del flujo mediante la media de las temperaturas obtenidas en el resto de las zonas de la vivienda que disponen de Sensores inalámbricos Clientes MQTT.

  La implementación de las funciones que forman parte del flujo de configuración Node-RED[6] del control global de temperatura se exponen en el Anexo 4.

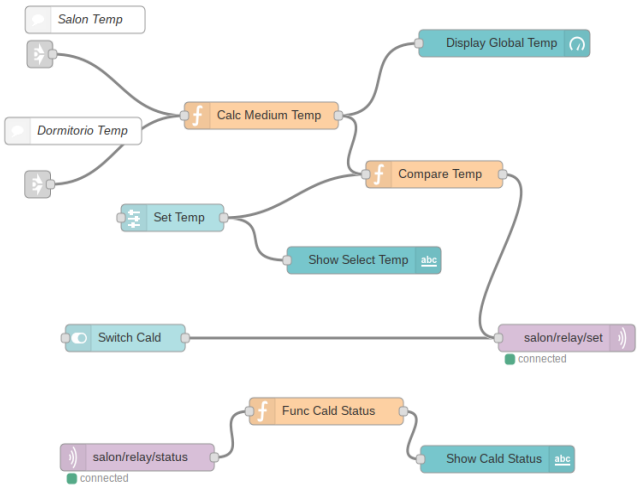


Ilustración 22: Flujo de configuración del control global de temperatura

- **Zona de salón**. [*Ilustración 23*] Este flujo de configuración gestiona el control del Sensor inalámbrico Cliente MQTT[2] instalado en el salón de la vivienda.

  Permite la visualización en el Dashboard[7] de los valores obtenidos por los sensores instalados en el Sensor inalámbrico Cliente MQTT[2] y el encendido/apagado remoto de la iluminación mediante el actuador instalado en el.

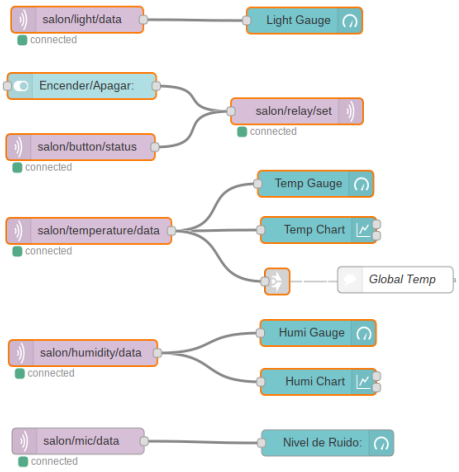


Ilustración 23: Flujo de configuración de la zona Salón

- **Zona de dormitorio**. [*Ilustración 24*] Este flujo de configuración gestiona el control del Sensor inalámbrico Cliente MQTT[2] instalado en uno de los dormitorios de la vivienda.

  Permite la visualización en el Dashboard[7] de los valores obtenidos por los sensores instalados en el Sensor inalámbrico Cliente MQTT[2] y el encendido/apagado remoto de la iluminación mediante el actuador instalado en el.

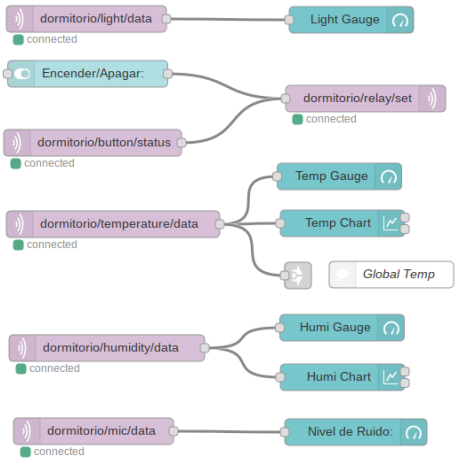


Ilustración 24: Flujo de configuración de la zona Dormitorio

El potencial de esta aplicación es realmente inmenso, quedando en su mayoría fuera del alcance de este proyecto. Para el desarrollo de nuestro sistema solo hemos necesitado crear flujos de complejidad baja/media. Sin embargo este programa permite realizar un gran número de tareas de manera sencilla e intuitiva, lo que lo hacen perfecto para este tipo de proyectos.

3.2. Sensor inalámbrico, Cliente MQTT.

El software empotrado en los Sensores inalámbricos Clientes MQTT[2] se ha diseñado teniendo en mente la posibilidad de aplicar la multitarea y la modularidad al diseño y ejecución dentro del microcontrolador Arduino[1] UNO. Esto permite dividir en tareas cada parte de la ejecución del programa, alcanzando un mayor rendimiento ya que el microcontrolador se encuentra en un estado de mayor ocupación. La modularidad nos permite la activación y desactivación de funciones dentro del código en tiempo de compilación, así ofreciendo una mayor adaptación de los Sensores inalámbricos Cliente MQTT[2] a la ubicación final en la que se sitúen dentro de la vivienda o en el dispositivo que se desea "domotizar".

En esta parte del diseño se ha encontrado el problema de no poder utilizar un sistema operativo en tiempo real (RTOS) como FreeRTOS, ya que las librerías MQTT[2] mas utilizadas en la plataforma Arduino[1] no se encuentran soportadas por este. Tras bastante tiempo de investigación, se desistio la idea optando por la utilización de la libreria TaskScheduler[12] para el propósito de la multitarea.

3.2.1 Codigo fuente que compone el proyecto.

El desarrollo del software empotrado en los Sensores inalámbricos Cliente MQTT[2] se a realizado utilizando como herramienta principal el IDE de desarrollo Eclipse[8] Neón junto con el AVR Eclipse Plugin[9].

El AVR Eclipse Plugin[9] es una extensión del C/C++ Development Toolkit para soportar el desarrollo de la serie de procesadores embebidos AVR de Atmel.

El IDE Ecplise dispone de un mayor numero de características mas avanzadas que el IDE de Arduino[1], como son la integración de Subversion/GIT o la terminación de código en el editor de código fuente. Estas caracteristicas lo convierten en una herramienta mas potente y flexible que el IDE de Arduino[1].

Todo el desarrollo del proyecto se ha realizado sobre la versión 1.8.1 del Arduino[1] Core.

El Workspace del IDE Eclipse[8] para este Proyecto está compuesto por dos proyectos C++:

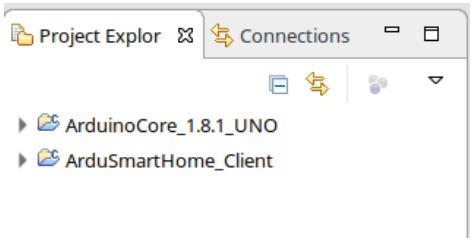


Ilustración 25: Eclipse Neon Workspace

- **ArduinoCore_1.8.1_UNO**. Es un proyecto C++ de tipo *AVR Cross Target Static Library,* configurado para un target MCU ATmega328P con una frecuencia de reloj de 16Mhz.

El codigo fuente de este proyecto lo componen el código fuente del Core de Arduino[1] versión 1.8.1 junto con la librerías utilizadas en el proyecto.

La compilación de este proyecto genera una librería estática que es utilizada por la aplicación de software empotrado en los Sensores inalámbricos Cliente MQTT[2].

```
Building target: libArduinoCore_1.8.1_UNO.a
Invoking: AVR Archiver
avr-ar -r  "libArduinoCore_1.8.1_UNO.a"  ./Libraries/WiFiEsp/src/utility/EspDrv.o ./Libraries/WiFiEsp/src/utility/RingBuffer.o
avr-ar: creating libArduinoCore_1.8.1_UNO.a
Finished building target: libArduinoCore_1.8.1_UNO.a

23:44:01 Build Finished (took 3s.347ms)
```

Ilustración 26: Compilación proyecto ArduinoCore_1.8.1_UNO

- **ArduSmartHome_Client**. Es un proyecto C++ de tipo *AVR Cross Target Application,* configurado para un target MCU ATmega328P con una frecuencia de reloj de 16Mhz.

  El codigo fuente del proyecto lo componen los ficheros *main.cpp* y *config.h*.

  - ***main.cpp***. Contiene el programa principal.

  - ***config.h***. *Contiene la configuración del programa.*

  Estos dos ficheros componen el código fuente del software empotrado en los Sensores inalámbricos Cliente MQTT[2] y puede ser consultado en el Anexo 6.

  La compilacion de este proyecto genera el archivo binario ArduSmartHome_Client.elf listo para ser grabado el el microcontrolador Ardino UNO.

```
Create eeprom image (ihex format)
avr-objcopy -j .eeprom --no-change-warnings --change-section-lma .eeprom=0 -O ihex ArduSmartHome_Client.elf
Finished building: ArduSmartHome_Client.eep

Invoking: Print Size
avr-size --format=avr --mcu=atmega328p ArduSmartHome_Client.elf
AVR Memory Usage
----------------
Device: atmega328p

Program:   22564 bytes (68.9% Full)
(.text + .data + .bootloader)

Data:       1582 bytes (77.2% Full)
(.data + .bss + .noinit)


Finished building: sizedummy

23:47:50 Build Finished (took 431ms)
```

Ilustración 27: Compilación proyecto ArduSmartHome_Client.

3.2.2 Configuración sensor inalámbrico, Cliente MQTT.

La configuración de los Sensores inalámbricos Cliente MQTT[2] se realiza en tiempo de compilación mediante el fichero de configuración *config.h* incluido en el proyecto *ArduSmartHome_Client*.

Siempre que se agrege un nuevo Sensor inalambrico Cliente MQTT[2] al sistema es obligatorio configurar su identificador unico que le identifique en la red MQTT[2] mediante la definición ID. [*Ilustración 28*]

Dada la modularidad en el desarrollo, tambien es posible la activacion y desactivación de los distintos sensores de los que puede disponer el Sensor inalambrico Cliente MQTT[2] mediante las respectivas definiciones a tal efecto. [*Ilustración 28*]

```
/*-----------------------------------------------------
 *
 * Client information, ID, etc.
 *
 *-----------------------------------------------------*/
const char* ID = "salon";

/*-----------------------------------------------------
 *
 * Definitions for enable sensors/actuators.
 *
 *-----------------------------------------------------*/
#define LDR
#define DHT
#define BUTTON
#define RELAY
#define MIC
```

Ilustración 28: Configuracion Sensor inalámbrico. ID y sensores.

El fichero de configuración *config.h* tambien permite la modificación de los parametros de red y la configuración del protocolo MQTT[2].

3.2.3 Librerias Arduino.

Es este apartado se describen las librerías de la plataforma Arduino[1] utilizadas en la implementación del Sensor inalámbrico Cliente MQTT[2].

- **SoftwareSerial[10].**

  La librería SoftwareSerial[10] ha sido desarrollada para permitir la comunicación en serie en otros pines digitales del Arduino[1], mediante el uso de software es posible replicar la funcionalidad del puerto serie. Es posible tener varios puertos serie de software con velocidades de hasta 115.200bps. Un parámetro permite la señalización invertida para dispositivos que requieren ese protocolo.

  La versión de SoftwareSerial[10] incluida en el entorno de desarrollo 1.0 y posteriores se basa en la biblioteca NewSoftSerial de Mikal Hart.

- **WiFiEsp[11].**

  La librería WiFiEsp[11] permite que una placa Arduino[1] se conecte a una red WiFi haciendo uso de un modulo ESP8266[4].

  Características de la librería WiFiEsp[11]:

  - APIs compatibles con la biblioteca estándar de Arduino[1] WiFi.
  - Utiliza comandos AT del firmware ESP estándar.
  - Soporta puertos serie hardware y software.
  - Nivel de seguimiento de errores configurable.
  - Soporta SDK ESP versión 1.1.1 o superior y versión AT 0.25 o superior.

- **TaskScheduler[12].**

  La librería TaskScheduler[12] es una implementación ligera de multitarea cooperativa en microcontroladores Arduino[1]. la multitarea cooperativa quiere decir que las tareas son responsables de ser "buenos vecinos", es decir, ejecutar sus métodos de devolución de llamada rápidamente y de una manera no bloqueante, devolviendo el control al planificador lo antes posible.

  Esta implementación ligera de multitarea cooperativa soporta:

  - Ejecución periódica de la tarea, estableciendo un período de ejecución dinámico en milisegundos o microsegundos (si se habilita explícitamente) .
  - Número de interacciones limitado o infinito de tareas.
  - Ejecución de tareas en una secuencia predefinida.
  - Cambio dinámico de los parámetros de ejecución de tareas.
  - Ahorro de energía al entrar en el modo de reposo inactivo cuando las tareas no están programadas para ejecutarse.
  - Soporte para la invocación de tareas por evento mediante Status Request object.
  - Soporte para ID de tareas y puntos de control para el manejo de errores WatchDog.

- Soporte para Local Task Storage Pointer que permite usar el mismo código de devolución de llamada para múltiples tareas.

- Soporte para priorización de tareas en capas.

En la *ilustración 29* se muestra el diagrama de flujo del ciclo de vida de las tareas creadas con la librería TaskScheduler[12].

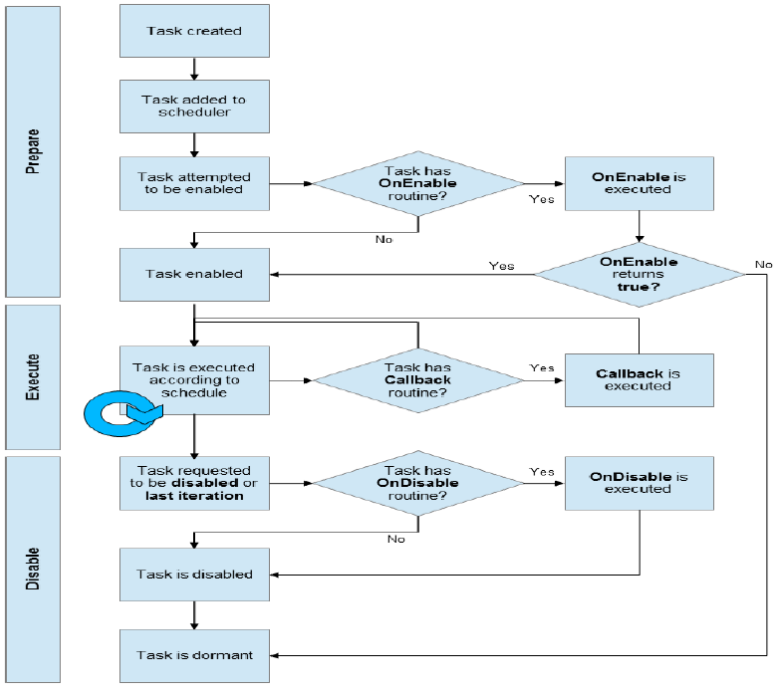


Ilustración 29: Ciclo de vida de las tareas. Librería TaskScheduler

El planificador ejecuta los métodos de devolución de llamada de las tareas en el orden en que se agregaron las tareas a la cadena, desde la primera hasta la última. El planificador se detiene después de procesar la cadena de tareas con el fin de permitir que otras sentencias en el código principal del método loop() se ejecuten. Esto se conoce como un "scheduling pass". Normalmente, no es necesario tener ninguna otra instrucción en el método loop() que no sea el método execute() del planificador.

Las tareas realizan ciertas funciones, que podrían requerir la ejecución periódica o única, la actualización de variables específicas o la espera de eventos específicos. Las tareas también podrían estar controlando hardware específico, o ser activadas por interrupciones de hardware.

- **PUBSubClient[13].**

Esta librería proporciona un cliente MQTT[2] ligero para realizar el envío de mensajes sencillos de publicación/suscripción con un servidor MQTT[2].

Limitaciones de la librería:

- Sólo puede publicar mensajes QoS 0. Puede suscribirse en QoS 0 o QoS 1.
- El tamaño máximo del mensaje, incluyendo el encabezado, es 128 bytes por defecto. Esto se puede configurar a través de MQTT_MAX_PACKET_SIZE en PUBSubClient.h.
- El intervalo keepalive se establece en 15 segundos de forma predeterminada. Esto se puede configurar a través de MQTT_KEEPALIVE en PUBSubClient.h.
- El cliente utiliza MQTT[2] 3.1.1 de forma predeterminada. Se puede cambiar para utilizar MQTT[2] 3.1 cambiando el valor de MQTT_VERSION en PUBSubClient.h.

- **DHTlib[14].**

  La librería DHTlib[14] permite la comunicación con los sensores de temperatura y humedad DHT11, DHT21 y DHT22 La comunicación con los sensores se realiza mediante el protocolo de comunicación 1-Wire, esto significa que solo se requiere un único cable para la comunicación con el sensor DHT.

- **QueueArray[15].**

  La librería QueueArray[15] implementa una cola genérica dinámica haciendo uso internamente de un array como estructura de datos. Esto permite agregar la estructura de datos abstractos FIFO (First In - First Out) a un programa para su uso.

3.2.4 Tareas.

Una tarea se define como el conjunto de funciones que controlan y gestionan un dispositivo o servicio dentro del programa, requiriendo de una ejecución programada. Así pues, disponemos en nuestro desarrollo de una tarea por cada sensor que se ha implementado en el sistema, mas dos tareas encargadas de gestionar los servicios de conectividad inalámbrica Wifi y cliente MQTT[2].

Este concepto de tarea combina los siguientes aspectos:

- Código de programa que realiza actividades específicas como métodos de devolución de llamada.
- Intervalo de ejecución.
- Número de interacciones de ejecución.

- Evento de inicio de ejecución, como funciones de configuracion de tarea.

- Puntero a una área Local Task Storage.

Las tareas se dividen entre productoras (sensores) y consumidoras (cliente MQTT[2]), la comunicación entre ellas se realiza mediante la implementación de una cola [*Ilustración 30*].

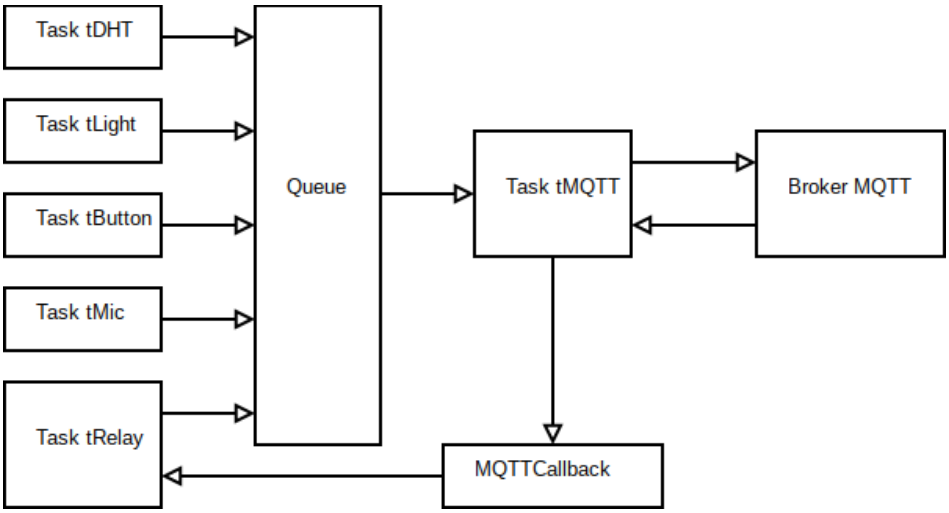


Ilustración 30: Diagrama de bloques comunicación entre tareas.

- **Task tWifi.**

  La tarea tWifi es la encargada de la conectividad inalámbrica con la red Wifi del sistema.

  Esta tarea realiza la función de WatchDog ejecutándose cada minuto.

  Además de realizar la reconexión a la red en caso de perdida de conectividad, actúa sobre la tarea que proporciona el servicio de Cliente MQTT[2], iniciando y deteniendo su ejecución según se disponga de conexión a la red inalámbrica.

- **Task tMQTT.**

  La tarea tMQTT realiza las funciones de cliente MQTT[2] y de tarea consumidora a través de la cola creada para tal efecto consumiendo los datos que producen las tareas productoras (sensores).

  El cliente MQTT[2] esta compuesto por dos funciones principales, tMQTTClient encargada de mantener la comunicación MQTT[2] y de la publicación de tópicos en el sistema y la función MQTTCallback que es llamada por la función tMQTTClient cada vez que se recibe un tópico al que se ha suscrito, realizando la acción determinada en el.

- **Task tDHT.**

  La tarea tDHT es una tarea de tipo productora. Realiza la lectura del sensor DHT11 cada cinco segundos y envía a la cola los datos de temperatura y humedad recibidos utilizando los tópicos "temperature/data" y "humidity/data" respectivamente.

  La tarea realiza un muestreo de los datos recibidos del sensor DHT11. Se recogen cinco muestras y se calcula su media. Esto proporciona una mayor calidad en la medición, eliminando el ruido que genera el propio sensor como son las muestras aleatorias.

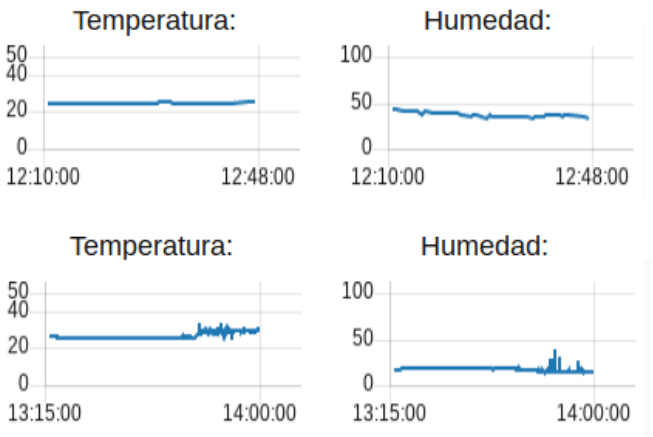
  

  Ilustración 31: Muestreo sensor DHT11

  En la *ilustración 31* se muestra una comparativa en las lecturas realizadas por el sensor DHT11 con muestreo (gráficas superiores) y sin muestreo (gráficas inferiores). Hacia el final de las gráficas se puede observar el ruido generado en la medición una vez eliminado el muestreo.

- **Task tButton.**

  La tarea tButton es una tarea de tipo productora. Realiza continuamente lecturas del estado del interruptor de tipo botón, enviando a la cola su estado si a ocurrido un cambio utilizando el tópico "button/status".

- **Task tRelay.**

  El control del actuador relé se realiza mediante la variable global de tipo booleano RELAY_STATUS definida en el fichero de configuración *config.h*.

  La tarea tRelay es la encargada de cambiar el estado del actuador relé de forma continua según lo definido en la variable RELAY_STATUS. Este método de trabajo permite de una forma sencilla que otras funciones del programa puedan gobernar el estado del actuador relé. La función MQTTCallback establece el estado en la variable RELAY_STATUS según los datos recibidos en el tópico "relay/set".

  Además la tarea tRelay es una tarea productora, enviando a la cola de forma continua la información del estado del actuador relé, utilizando el tópico "relay/status".

- **Task tLight.**

  La tarea tLight es una tarea productora, envía a la cola de forma continua la información recibida del sensor de iluminación LDR utilizando el tópico "light/data". Esta información es la cantidad de iluminación que dispone en un momento dado la instancia en la que se encuentra ubicado el sensor.

  Los test de medida realizados durante el desarrollo han concluido que no es necesario aplicar muestreo al sensor de iluminación LDR, ya que este sensor realiza una medición estable, no detectando muestras re ruido significativas durante la medición.

- **Task tMic.**

  La tarea tMic es una tarea productora, envía a la cola de forma continua la información recibida del sensor de Micrófono Sensible utilizando el tópico "mic/data". Esta información es la cantidad de ruido ambiental existente en un momento dado dentro de la instancia en la que se encuentra ubicado el sensor.

  Los test de medida realizados durante el desarrollo han concluido que no es necesario aplicar muestreo al sensor de Micrófono Sensible, ya que este sensor realiza una medición estable, no detectando muestras re ruido significativas durante la medición.

3.2.5 Diseño de multitarea colaborativa. Prioridad entre tareas.

Para fines de ejecución las tareas se enlazan en cadenas de ejecución, que son procesadas por el planificador en el orden en que se agregaron enlazándolas entre sí.

```
Scheduler runner, hprunner;

runner.setHighPriorityScheduler(&hprunner);

Task tMQTT(TASK_IMMEDIATE, TASK_FOREVER, &tMQTTClient, &hprunner, false, &tMQTTSetup);

Task tWifi(TASK_MINUTE, TASK_FOREVER, &tWifiCallback, &runner, false, &tWifiSetup);

Task tDHT(TASK_SECOND * 5, TASK_FOREVER, &tDHTCallback, &runner, false, &tDHTSetup);

Task tLight(TASK_SECOND * 2, TASK_FOREVER, &tLightCallback, &runner, false, &tLightSetup);

Task tButton(TASK_IMMEDIATE, TASK_FOREVER, &tButtonCallback, &runner, false, &tButtonSetup);

Task tRelay(TASK_IMMEDIATE, TASK_FOREVER, &tRelayCallback, &runner, false, &tRelaySetup);

Task tMic(TASK_SECOND * 2, TASK_FOREVER, &tMicCallback, &runner, false, &tMicSetup);
```

Ilustración 32: Configuración de prioridad entre tareas.

La priorización de tareas utilizando la API de TaskScheduler[12] se implementa creando varios planificadores y organizarlos en capas de priorización. Las tareas se asignan a los planificadores según la prioridad que se quiera asignar. Las tareas asignadas a las capas "superiores" se evalúan con más frecuencia

para su invocación y se les da prioridad en la ejecución en caso de incidencia de programación.

En el desarrollo del proyecto se han implementado dos planificadores, organizarlos en dos capas de priorización. La tarea tMQTT se la ha dotado de una mayor prioridad asignándola al planificador *hprunner*, dejando el resto de tareas que componen el programa asignadas al planificador *runner*. [*Ilustración 32*].

La priorización de tareas se logra ejecutando toda la cadena de tareas del planificador de mayor prioridad (*hprunner*) para cada paso de la cadena de menor prioridad (*runner*). Así se consigue que las tareas que se han definido con una mayor prioridad se evalúe su ejecución con mayor frecuencia.

Secuencia de evaluación del planificador:

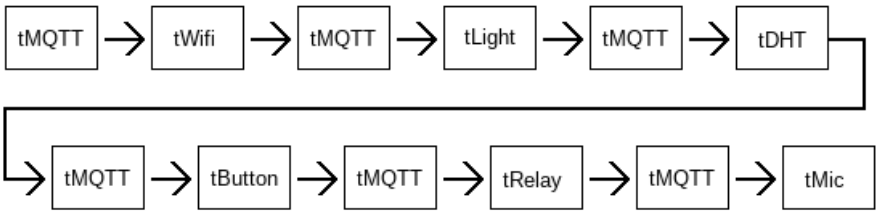


Ilustración 33: Secuencia de evaluación del planificador.

Cada capa de prioridad agrega sobrecarga de planificación a la ejecución general de la cadena de tareas. La idea principal es minimizar esta sobrecarga de planificación, evitando dentro de los requerimientos del programa, el uso del menor numero de capas posible.

La sobrecarga de planificación es posible calcularla de forma teórica. En este caso disponemos de dos capas prioritarias de 7 tareas. En las que disponemos de 6 tareas con prioridad base y 1 tarea con prioridad más alta:

$$O = (B + B * P1) * T = (6 + 6 * 1) * 18 = \textbf{216 microsegundos.}$$

Dónde:

- O - Sobrecarga de planificación.
- B - Número de tareas en la capa base.
- P1 - Número de tareas en la capa de prioridad 1
- T - Planificación de sobrecarga de una sola evaluación de ejecución de tareas. Actualmente este dato con Arduino[1] Uno funcionando a 16Mhz es de entre 15ms y 18ms

Los cálculos realizados son simplemente teóricos para realizar una demostración en el mejor de los casos, donde todas las tareas involucradas tienen un tiempo de ejecución inmediato. No se hay tenido en cuenta las tareas que tienen programado un tiempo de delay en cada ciclo de ejecución.

Más información sobre la multitarea colaborativa y la priorización en tareas en la documentación de la API de TaskScheduler[12].

4. Valoración económica.

A continuación se realiza una valoración del coste de los materiales empleados, así como una valoración estimada del coste de las horas de desarrollo.

Coste del hardware que compone el Broker MQTT[2]:

Descripción	Cantidad	Precio Unitario	Precio Total
Arduino[1] UNO R3	1	6,03 €	6,03 €
Dragino[5] Yun Shield v2.4	1	33,29 €	33,29 €
Adaptador de corriente	1	4,76 €	4,76 €
Cable RJ45 40cm.	1	0,52 €	0,52 €
			Total: 44,60 €

Coste del hardware que compone el Sensor Inalámbrico Cliente MQTT[2] (2 unidades):

Descripción	Cantidad	Precio Unitario	Precio Total
Arduino[1] UNO R3	2	6,03 €	12,06 €
Arduino[1] ESP8266[4] WiFi Shield v1.0	2	5,80 €	11,60 €
Arduino[1] Sensor Shield v4.0.	2	4.02 €	8,04 €
Kit de sensores Arduino[1]	2	11,35 €	22,70 €
Cablecillos H/H 40 uni.	2	2,56 €	5,12 €
Portapila 9v	2	1,09 €	2,18 €
Bateria 9v	2	1,95 €	3,90 €
			Total: 65,60 €

Coste estimado del desarrollo del Proyecto (En horas):

Descripción	Cantidad	Precio Unitario	Precio Total
Planificación	5	30 €	150 €
Diseño/Montaje Hardware	8	30 €	240 €
Diseño/Desarrollo Software	20	30 €	600 €
Diseño/Desarrollo Dashboard[7]	10	30 €	300 €
			Total: 1.290 €

5. Conclusiones.

En este apartado se comentan las conclusiones obtenidas después de la realización del proyecto, así como un análisis objetivo de algunas mejoras que podrian implementarse en una segunda versión del proyecto. Además, en el ultimo apartado se describe la valoración personal respecto a la realizacón del proyecto.

5.1. Conclusiones.

Los objetivos del proyecto han estado fijados desde el inicio del desarrollo en los cuatro puntos que se especifican en el principio de esta memoria, pudiendo concluir que se han alcanzado cada uno de ellos con el sistema global que se a diseñado y con cada uno de los prototipos que lo forman.

- Se han integrado varios sensores con una placa de desarrollo Arduino[1] UNO R3, como son de temperatura, humedad, iluminación y sonido ambiental, junto con actuadores como son un relé y un pulsador. Ademas se ha dotarlo de conectividad inalámbrica haciendo uso de un modulo ESP8266[4].

- Se ha instalado un servidor Broker MQTT[2] Mosquitto[3] en una placa de desarrollo Dragino[5] YUN Shield.

- Haciendo uso de la librería PUBSubClient[13] de Arduino[1], se ha implementado un cliente MQTT[2] en los sensores inalámbricos, integrándolo con los sensores definidos en el primer punto, quedando establecida una red de comunicación Machine-to-Machine (M2M) junto al servidor Broker MQTT[2] Mosquitto[3].

- Haciendo uso del motor de flujos Node-RED[6], se ha logrado implementar una plataforma interactiva de acceso Web, permitiendo la interpretación de los diferentes datos obtenidos de los sensores.

Dado que el objetivo del proyecto no era un desarrollo de un producto comercial completo de hardware, si no el desarrollo didactico de un prototipo basado en la plataforma de desarrollo de prototipado Arduino[1], no se ha llevado a cabo un desarrollo final como producto comercial, llevando a cabo el diseño electronico de integración del conjunto dentro de un PCB o el diseño de cajas para albergar en conjunto de los componentes. En una segunda fase de implementación sería un objetivo principal, ya que dotaría al sistema de una posible aceptación comercial.

El aprendizaje y conocimientos adquiridos sobre la plataforma Arduino[1] han sido los deseados. Uniendo las horas de investigación junto a las de desarrollo se ha conseguido una base para dar una continuidad a la creación de proyectos dentro de esta plataforma o mejorando aspectos de este proyecto además de la posibilidad de abarcar otros de nueva índole.

5.2. Propuesta de mejoras.

A medida que avanzaba el proyecto se han encontrado diferentes mejoras que en algunos casos se han podido ir aplicando durante su desarrollo, y en otros casos, bien por cuestiones de materiales o temporales, han quedado para futuras revisiones.

Algunas de estas mejoras que se podrán aplicar al proyecto:

- Extender el desarrollo del proyecto a una mayor cantidad de sensores, como detectores de llama, detectores de gas y añadir mas actuadores de tipo relé a los Sensores inalámbricos Clientes MQTT[2]..
- Desarrollar en el software empotrado del Sensor inalámbrico Cliente MQTT[2] un sistema de suspensión de ahorro de energía. Siempre que esto permita la interacción en tiempo real con el resto de sistema.
- Implementación de un sistema de persistencia de datos. Esto permite el almacenamiento de los diferentes datos obtenidos de los sensores en una base de datos, evitando su perdida en cada reinicio del sistema.
- Dotar a la Web Dashboard[7] de un mayor numero de controles y de una mejora gráfica como puede ser, incluir el plano de la vivienda con la ubicación de cada Sensor inalámbrico Cliente MQTT[2].
- Desarrollo del PCB que integre todos los componentes electrónicos que forman el Sensor inalámbrico Cliente MQTT[2].
- Crear una solución mas comercial, mediante el diseño y desarrollo de una serie de cajas para albergar el servidor MQTT[2] y todos los componentes que componen el Sensor inalámbrico Cliente MQTT[2].

Estas son solo algunas mejoras que se podrían aplicar, pero evidentemente no las únicas, y puesto que nos encontramos con un proyecto de software libre abierto y en constante desarrollo, a medida que el proyecto siga avanzando se encontraran nuevas ideas y mejoras que se puedan implementar.

5.3. Valoración personal.

En primer lugar hay que indicar que la valoración personal sobre este proyecto es muy buena. La realización de este tipo de proyectos como finalización de unos estudios de ingeniería, tanto en informática como en electrónica, es muy acertado ya que permite poner en practica los conocimientos adquiridos de una forma totalmente practica.

Este proyecto me ha permitido conocer la metodología para el desarrollo de proyectos de Hardware y Software Libre con el uso de diferentes tecnologías para alcanzar los objetivos marcados, analizando las ventajas y desventajas del abanico interminable de herramientas disponibles. También he aprendido a investigar las diferentes fuentes de información disponibles en la web, discriminando dentro del aluvión de información, estudiando los problemas encontrados por otros usuarios y así alcanzando las soluciones mas eficientes.

Por otro lado me ha permitido conocer proyectos muy interesantes en materia domótica domestica, y los beneficios que podemos obtener de la automatización aplicada al hogar. Siendo una tecnología con mucho futuro, se encuentra actualmente copada por soluciones de fabricantes cerradas y con precios elevados. Realizando proyectos sobre plataformas abiertas como es Arduino[1] permiten abrir una pequeña ventana a que este tipo de tecnología sean mas accesibles.

Por todo lo anterior estoy muy contento por el trabajo realizado y el resultado final de este Trabajo Final de Grado.

6. Glosario

Arduino: Es una compañía de hardware libre y una comunidad tecnológica que diseña y manufactura placas electrónicas de desarrollo de hardware y software, compuesta respectivamente por circuitos impresos que integran un microcontrolador y un entorno de desarrollo (IDE).

AVR: Familia de microcontroladores RISC del fabricante Atmel.

Shield: Interfaces disponibles para conexión directa Arduino.

MQTT: Message Queue Telemetry Transport.

Domotica: Sistemas capaces de automatizar una vivienda o edificación.

WiFi: Mecanismo de conexión de dispositivos electrónicos de forma inalámbrica.

API: Interfaz de programación de aplicaciones.

M2M: Conectividad Machine-to-Machine.

Dashboard:Es una interfaz que dispone de los instrumentos donde el usuario puede administrar un software determinado.

Broker MQTT: Dentro de una red MQTT es el encargado de gestionar la red y de transmitir los mensajes.

UART: Transmisor-Receptor Asíncrono Universal, transmision y recepción de datos serie.

TTL: Familia electronica transistor-transistor logic.

DHT11: Sensor de humedad y temperatura.

LDR: Fotoresistencia.

IDE: Entorno de desarrollo integrado.

SDK:Kit de desarrollo de software.

AT: Conjunto de comandos Hayes.

QoS: Calidad de servicio.

FIFO: First In, First Out. Estructura de datos para implementar colas.

PC: Ordenador personal.

JVM: Maquina virtual de Java.

REST: Estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web

Websocket: Tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP.

AMQP: Protocolo de estándar abierto en la capa de aplicaciones de un sistema de comunicación. Las características son la orientación a mensajes, encolamiento, enrutamiento punto-a-punto y publicación-subscripción, exactitud y seguridad.

WatchDog: Mecanismo de seguridad que provoca un reset del sistema en caso de que éste se haya bloqueado.

1-Wire: Protocolo de comunicaciones en serie basado en un bus, un maestro y varios esclavos de una sola línea de datos en la que se alimentan.

7. Bibliografía

[1]. **Arduino**.(22/02/2017) https://www.arduino.cc/

[2]. **MQTT**. (22/04/2017) http://MQTT.org/

[3]. **Mosquitto**.(20/04/2017) https://Mosquitto.org/

[4]. **ESP8266.** (27/03/2017) http://www.ESP8266.com/

[5]. **Dragino Yun Shield**.(22/02/2017) http://www.Dragino.com/products/yunshield.html

[6]. **Node-RED**.(07/05/2017) https://nodered.org/

[7]. **Node-RED Dashboard**. (07/05/2017) https://github.com/Node-RED/Node-RED-Dashboard

[8]. **Eclipse Neon**. (20/04/2017) http://www.Eclipse.org/neon/

[9]. **AVR Eclipse Plugin**. (20/04/2017) http://avr-elipse.sourceforge.net/wiki/index.php/The_AVR_Eclipse_Plugin

[10]. **SoftwareSerial**. (22/04/2017) https://www.arduino.cc/en/Reference/SoftwareSerial

[11]. **WiFiEsp**.(27/03/2017) https://github.com/bportaluri/WiFiEsp

[12]. **TaskScheduler**. (27/03/2017) https://github.com/arkhipenko/TaskScheduler

[13]. **PUBSubClient**.(22/04/2017) http://PUBSubClient.knolleary.net/

[14]. **DHTlib**. (27/03/2017)http://playground.arduino.cc/Main/DHTlib

[15]. **QueueArray**. (27/03/2017) https://playground.arduino.cc/Code/QueueArray

8. Anexos

8.1. Anexo 1. Especificaciones técnicas

8.1.1 Especificaciones técnicas hardware Arduino.

Especificaciones técnicas de la placa Arduino[1] UNO R3:

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Características del Dragino[5] Yun Shield v2.4:

- Sistema operativo Linux de código abierto (OpenWrt) integrado.
- Bajo consumo de energía.
- Compatible con Arduino[1] IDE 1.5.4 o superior.
- Administrado por GUI Web o SSH vía LAN o WiFi.
- Software actualizable vía red.
- Servidor web incorporado.
- Soporte de conexión a Internet a través de puerto LAN o Wi-Fi.
- Soporte de almacenamiento USB.
- El diseño Failsafe proporciona un sistema robusto.
- Compatible con Arduino[1] Leonardo, Uno, Duemilanove, Diecimila, Mega, Due, Teensy.
- Soporte de alimentación mediante PoE.

Especificaciones técnicas del Dragino[5] Yun Shield v2.4:

Processor	400MHz, 24K MIPS
Flash	16 MBytes
RAM	64 MBytes
Power Input	7v ~ 23v vía Arduino[1] VIN pin
Ethernet	1x 10M/100M RJ45 connector
WiFi	150M WiFi 802.11 b/g/n
External Antenna	I-Pex connector
USB	2x USB 2.0 host connector
Button	1x Reset button
I/O	Compatible with 3.3v or 5v I/O Arduino[1].
Storage	1x micro SD card socket

8.1.2 Especificaciones técnicas Sensores.

Características y conexiones. Sensor de Temperatura y Humedad DHT11.

El Sensor de Temperatura y Humedad cuenta con un sensor DHT11, el cual, envía una señal digital a su salida. Tiene un tamaño ultra compacto, bajo consumo de energía, con una señal de transmisión de hasta 20 metros.

- Características:
  - Voltaje de Suministro: 3.3 ~ 5.5V DC
  - Salida: Señal Digital.
  - Rango de Medición: Humedad 20-90% RH, Temperatura 0 ~ 50 °C.
  - Precisión: Humedad + -5% RH, Temperatura + -2 °C.
  - Resolución: Humedad 1% RH, Temperatura 1 °C.
  - Estabilidad a Largo Plazo: <± 1% RH / Año.
- Conexiones:
  - Pin +5v del Arduino[1] -> Pin "VCC" del módulo.
  - Pin GND del Arduino[1] -> Pin “GND” del módulo.
  - Pin D5 del Arduino[1] -> Pin "DATA" del módulo.

Características y conexiones. Sensor LDR.

El módulo Sensor LDR contiene una fotoresistencia, la cual es una resistencia variable dependiente de la cantidad de luz en su entorno. Los valores de su resistencia, sensibilidad, coeficiente de temperatura y su curva de voltaje-corriente dependen directamente de la cantidad de luz que recibe el sensor.

- Conexiones:
  - Pin +5v del Arduino[1] -> Pin +5v del módulo.
  - Pin GND del Arduino[1] -> Pin “-” del módulo.
  - Pin A0 del Arduino[1] -> Pin "S" del módulo.

Características y conexiones. Sensor de Micrófono Sensible.

El modulo Sensor de Micrófono Sensible nos permite capturar el nivel de sonido ambiental próximo al sensor. El umbral de sensibilidad se puede ajustar mediante el potenciómetro integrado en el modulo.

Para la detección de sonido el módulo tiene dos salidas;

- AO. Tipo de salida analógica. Permite la lectura de la señal de tensión de salida en tiempo real del micrófono.
- DO. Tipo de salida digital. Cuando la intensidad del sonido alcanza un cierto umbral, el nivel de señal de salida pasa de baja a alta.
  - Conexiones:
    - Pin +5v del Arduino[1] -> Pin +5v del módulo.
    - Pin GND del Arduino[1] -> Pin “-” del módulo.
    - Pin A1 del Arduino[1] -> Pin A0 del módulo.
    - Pin D7 del Arduino[1] -> Pin D0 del módulo.

Características y conexiones. Interruptor tipo Botón.

El módulo Interruptor tipo Botón tiene integrado un interruptor de tipo botón común, que permite la interacción humana con el sistema.

- Conexiones:
  - Pin +5v del Arduino[1] -> Pin +5v del módulo.
  - Pin GND del Arduino[1] -> Pin “-” del módulo.
  - Pin D4 del Arduino[1] -> Pin "S" del módulo.

Características y conexiones. Actuador Relé.

El modulo Relé permite controlar circuitos de hasta 240V AC 10A.

- Conexiones:
  - Pin +5v del Arduino[1] -> Pin +5v del módulo.
  - Pin GND del Arduino[1] -> Pin “-” del módulo.
  - Pin D6 del Arduino[1] -> Pin "S" del módulo.

8.2. Anexo 2. Instalación Broker MQTT.

La instalación de los paquetes necesarios para servidor Mosquitto[3] se realiza desde los repositorios del Dragino[5] Yun Shield utilizando el comando opkg. [*Ilustración 34*].

```
root@ArduSmartHome:~# opkg install libmosquitto
Installing libmosquitto (0.15-1) to root...
Downloading http://www.dragino.com/downloads/downloads/motherboards/ms14/Firmware/Yun/Packages--v2.x/libmosq
uitto_0.15-1_ar71xx.ipk.
Configuring libmosquitto.
root@ArduSmartHome:~# opkg install mosquitto
Installing mosquitto (0.15-1) to root...
Downloading http://www.dragino.com/downloads/downloads/motherboards/ms14/Firmware/Yun/Packages--v2.x/mosquit
to_0.15-1_ar71xx.ipk.
Configuring mosquitto.
root@ArduSmartHome:~# opkg install mosquitto-client
Installing mosquitto-client (0.15-1) to root...
Downloading http://www.dragino.com/downloads/downloads/motherboards/ms14/Firmware/Yun/Packages--v2.x/mosquit
to-client_0.15-1_ar71xx.ipk.
Configuring mosquitto-client.
```

Ilustración 34: Instalación de paquetes Mosquitto

Desde el entorno web integrado también permite la instalación de los paquetes de software, pero no permite la configuración del servidor Mosquitto[3]. [*Ilustración 35*].

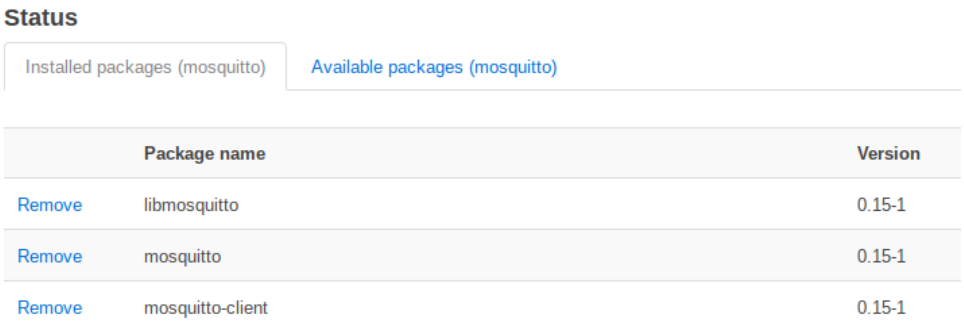


Ilustración 35: Instalación de paquetes Mosquitto (Entorno Web)

Es necesario configurar el servidor Mosquitto[3] como servicio con autoarranque, que permita iniciar el servidor en cada inicio del sistema [*Ilustración 36*].

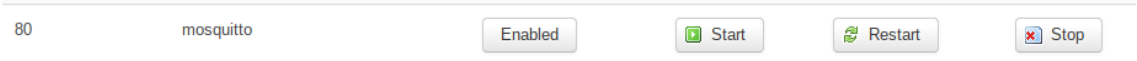


Ilustración 36: Configuración de Mosquitto como servicio.

Una vez iniciado el servidor Mosquito los sensores inalámbricos ya pueden establecer comunicación con el servidor mediante el protocolo MQTT[2] [*Ilustración 37*] y comenzar el envio de información mediante la publicación y subscripción de topicos [*Ilustración 38*].

```
1495639593: New connection from 192.168.240.109.
1495639593: New client connected from 192.168.240.109 as dormitorio.
1495639593: New connection from 192.168.240.141.
1495639593: New client connected from 192.168.240.141 as salon.
```

Ilustración 37: Conexion clientes MQTT

```
1495639984: Received PUBLISH from salon (d0, q0, r0, m0, 'salon', ... (5 bytes))
1495639984: Received SUBSCRIBE from salon
1495639984:     salon/relay/set (QoS 1)
1495639984: salon 1 salon/relay/set
1495639984: Sending SUBACK to salon
1495639984: Received PUBLISH from salon (d0, q0, r0, m0, 'salon/light/data', ... (6 bytes))
1495639985: Received PUBLISH from salon (d0, q0, r0, m0, 'salon/mic/data', ... (6 bytes))
1495639985: Received PUBLISH from salon (d0, q0, r0, m0, 'salon/light/data', ... (6 bytes))
1495639985: Received PUBLISH from salon (d0, q0, r0, m0, 'salon/light/data', ... (6 bytes))
1495639986: Received PUBLISH from salon (d0, q0, r0, m0, 'salon/mic/data', ... (6 bytes))
1495639987: Received PUBLISH from salon (d0, q0, r0, m0, 'salon/light/data', ... (6 bytes))
1495639987: Received PUBLISH from salon (d0, q0, r0, m0, 'salon/mic/data', ... (6 bytes))
1495639989: Received PUBLISH from salon (d0, q0, r0, m0, 'salon/light/data', ... (6 bytes))
1495639989: Received PUBLISH from salon (d0, q0, r0, m0, 'salon/mic/data', ... (6 bytes))
```

Ilustración 38: Publicación y subscripción de tópicos MQTT

Securización del servidor Mosquitto[3].

El servidor Mosquitto[3] por defecto permite conexiones anónimas, esto es, que cualquier cliente pueda conectarse y capturar los datos que se envían dentro de la red MQTT[2]. Para evitar esta situación, es necesario modificar los parámetros *allow_anonymous* y *password_file* en el fichero de configuración **/etc/Mosquitto/Mosquitto.conf** [*Ilustración 39*].

```
# Boolean value that determines whether clients that connect
# without providing a username are allowed to connect. If set to
# false then a password file should be created (see the
# password_file option) to control authenticated client access.
# Defaults to true.
allow_anonymous false

# Control access to the broker using a password file. The file is a
# text file # of lines in the format:
# username:password
# The password (and colon) may be omitted if desired, although this
# offers very little in the way of security.
password_file /etc/mosquitto/passwd
```

Ilustración 39: Securizar la configuración Mosquitto

Los datos de autenticación de usuarios se configuran en el fichero que ha sido definido en el parámetro *password_file* con el formato "*user:password*", una linea por usuario al que se permita la conexión con el servidor Mosquitto[3] [*Ilustración 40*].

```
root@ArduSmartHome:~# cat /etc/mosquitto/passwd
ardusmarthome:ardusmarthome
```

Ilustración 40: Fichero de usuarios Mosquitto

8.3. Anexo 3. Instalación y configuración Node-RED.

8.3.1 Instalación Node-RED.

Node-RED[6] trabaja sobre NodeJS, es necesario instalar el paquete node desde los repositorios del fabricante con el comando opkg:

```
root@ArduSmartHome:~# opkg install node
Installing node (v0.10.33-1) to root...
Downloading http://www.dragino.com/downloads/downloads/motherboards/ms14/Firmware/Yun/Packages--v2.x/node_v0
.10.33-1_ar71xx.ipk.
Configuring node.
```

Ilustración 41: Instalación del paquete node

El motor Node-RED[6] se instala en modo Standalone, esto quiere decir que se ejecuta como un proceso NodeJS independiente. Para proceder con la instalación el paquete del programa está disponible en el repositorio github del proyecto Node-RED[6]:

https://github.com/Node-RED/Node-RED/releases/download/0.14.6/Node-RED-0.14.6.zip

** Se utiliza la versión 0.14.6 de Node-RED[6], por ser la ultima versión que ofrece compatibilidad con la versión de NodeJS v0.10.33.*

Al encontrarnos con un proyecto en un fuerte y continuo desarrollo, el proceso de instalación lo podemos encontrar en la web del proyecto Node-RED[6]:

http://nodered.org/docs/getting-started/installation

Al igual que para Node-RED[6], el procedimiento de instalación del modulo Node-RED-Dashboard[7] se encuentra en el repositorio github del proyecto:

https://github.com/Node-RED/Node-RED-Dashboard

```
Welcome to Node-RED
===================

24 May 22:41:53 - [info] Node-RED version: v0.14.6
24 May 22:41:53 - [info] Node.js  version: v0.10.33
24 May 22:41:53 - [info] Linux 3.3.8 mips BE
24 May 22:41:53 - [info] Loading palette nodes
24 May 22:42:29 - [info] Dashboard version 2.3.9 started at /ui
24 May 22:43:17 - [warn] ---------------------------------------------------
24 May 22:43:17 - [warn] [rpi-gpio] Info : Ignoring Raspberry Pi specific node
24 May 22:43:17 - [warn] ---------------------------------------------------
24 May 22:43:17 - [info] Settings file  : /root/.node-red/settings.js
24 May 22:43:18 - [info] User directory : /root/.node-red
24 May 22:43:18 - [info] Flows file     : /root/.node-red/flows_ArduSmartHome.json
24 May 22:43:20 - [info] Server now running at http://127.0.0.1:1880/
24 May 22:43:20 - [info] Starting flows
24 May 22:43:25 - [info] Started flows
24 May 22:43:27 - [info] [mqtt-broker:12790186.a49416] Connected to broker: ArduSmartHome@mqtt://127.0.0.1:1
883
```

Ilustración 42: Ejecución de Node-RED

Para configurar que Node-RED[6] se ejecute durante el inicio del Servidor Broker MQTT[2], es necesario la creación de un script de inicio que permita su lanzamiento como servicio nodejs en el inicio del sistema [*Ilustración 43*].

```
#!/bin/sh /etc/rc.common

start() {
        # /usr/bin/node-red > /var/log/node-red.log
        forever start -o /var/log/node-red.log -p /root /opt/node-red/red.js --userDir /root/.node-red
}

stop() {
        forever stop "/opt/node-red/red.js"
}

status() {
        forever list
}
```

Ilustración 43: Script de inicio de Node-RED

8.3.2 Configuración Node-RED.

Una vez que disponemos de Node-RED[6] instalado y en ejecución, podemos acceder a su url de configuración mediante la dirección IP del Servidor Broker MQTT[2]:

http://192.168.0.20:1880

La configuración de los flujos de Node-RED[6] se encuentra en el fichero de texto plano *flows_ArduSmartHome.json*.

Este fichero de configuración se puede importar desde la interfaz de Node-RED[6] como se muestra en la *Ilustración 44*.

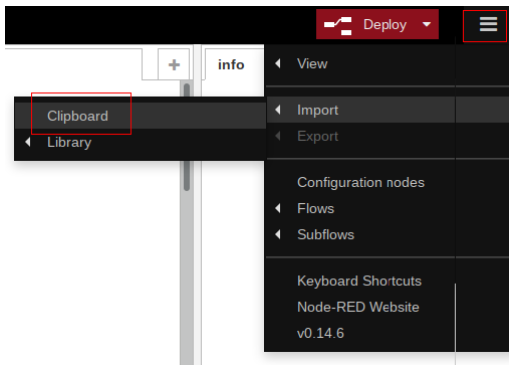


Ilustración 44: Importar configuración Node-RED

Basta con copiar y pegar el contenido del fichero en el cuadro de dialogo y pulsar el botón import. [*Ilustración 45*].

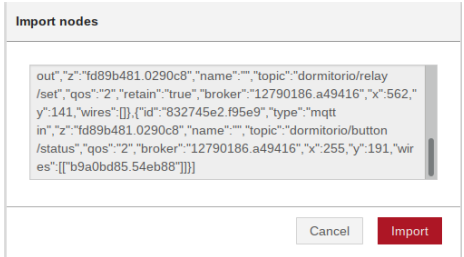


Ilustración 45: Cuadro de dialogo import nodes

Una vez realizada y completada la importación de nodos, ya se disponen los flujos totalmente configurados en la interfaz de Node-RED[6]. [*Ilustración 46*].

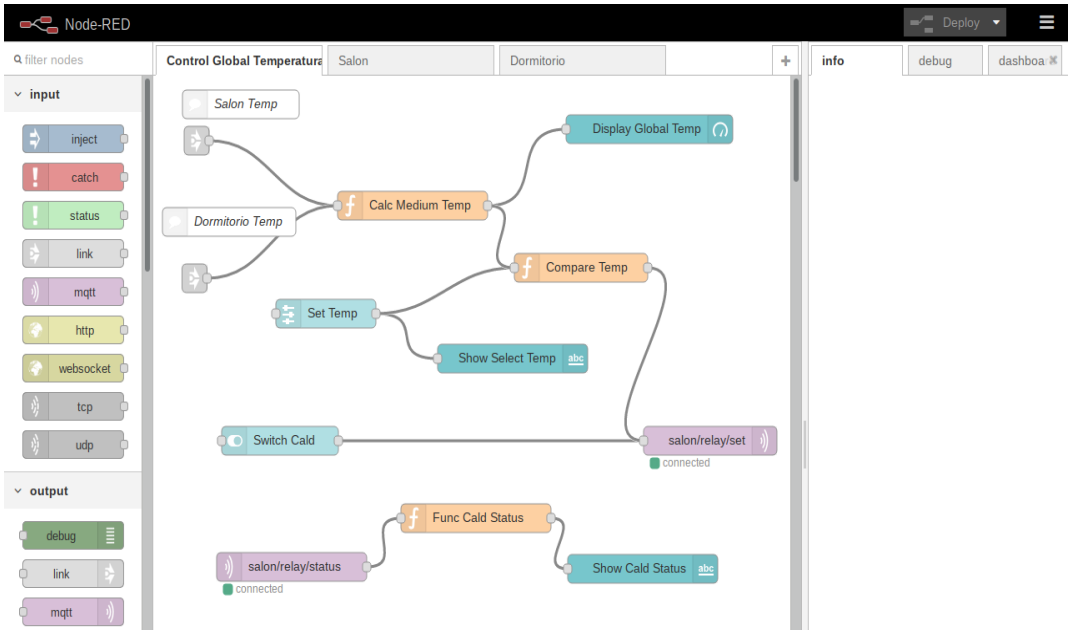


Ilustración 46: Interfaz Node-RED

8.4. Anexo 4. Implementación de funciones Node-RED.

- **Función *Calc Medium Temp***. Recibe como parámetros los datos de temperatura de las zonas de la vivienda y realiza el calculo de la temperatura media. [*Ilustración 47*].

```
context.SalonTemp === context.SalonTemp || 0.00;
context.DormitorioTemp === context.DormitorioTemp || 0.00;
context.Value === context.Value || 0.00;

if (msg.topic === 'salon/temperature/data') {
  context.SalonTemp = parseFloat(msg.payload);
} else if (msg.topic === 'dormitorio/temperature/data') {
  context.DormitorioTemp = parseFloat(msg.payload);
}

context.Value = ((context.SalonTemp + context.DormitorioTemp) / 2);

return {topic: 'GetTemp', payload: context.Value};
```

Ilustración 47: Función Calc Medium Temp

- **Función *Compare Temp***. Compara la temperatura media recibida con la temperatura seleccionada por el usuario. Si la temperatura seleccionada es mayor, su salida es "1" activando el actuador Relé. [*Ilustración 48*].

```
context.SetTemp === context.SetTemp || 0.00;
context.GetTemp === context.GetTemp || 0.00;
context.Value === context.Value || 0;
context.OldValue === context.OldValue || 0;

if (msg.topic === 'SetTemp') {
  context.SetTemp = msg.payload;
} else if (msg.topic === 'GetTemp') {
  context.GetTemp = msg.payload;
}

if ( context.SetTemp > context.GetTemp ) {
    context.Value = 1;
} else {
    context.Value = 0;
}

if ( context.Value != context.OldValue ) {
    context.OldValue = context.Value;
    return { topic: 'SetRelay', payload: context.Value };
}
```

Ilustración 48: Función Compare Temp

- **Función *Func Cald Status***. Muestra el estado de la caldera mediante texto en la interfaz de usuario Dashboard[7]. [*Ilustración 49*].

```
var msg1 = { payload:"Encendida" };
var msg2 = { payload:"Apagada" };

if (msg.payload === "1.00") {
   return [ msg1 ];
} else if (msg.payload === "0.00") {
   return [ msg2 ];
}
```

Ilustración 49: Función Func Cald Status

8.5. Anexo 5. Manual de Usuario.

Mediante nuestro navegador Web favorito, accedemos a la pagina Web de control de ArduSmartHome escribiendo en la barra de direcciones la siguiente URL:

http://192.168.0.20:1880/ui

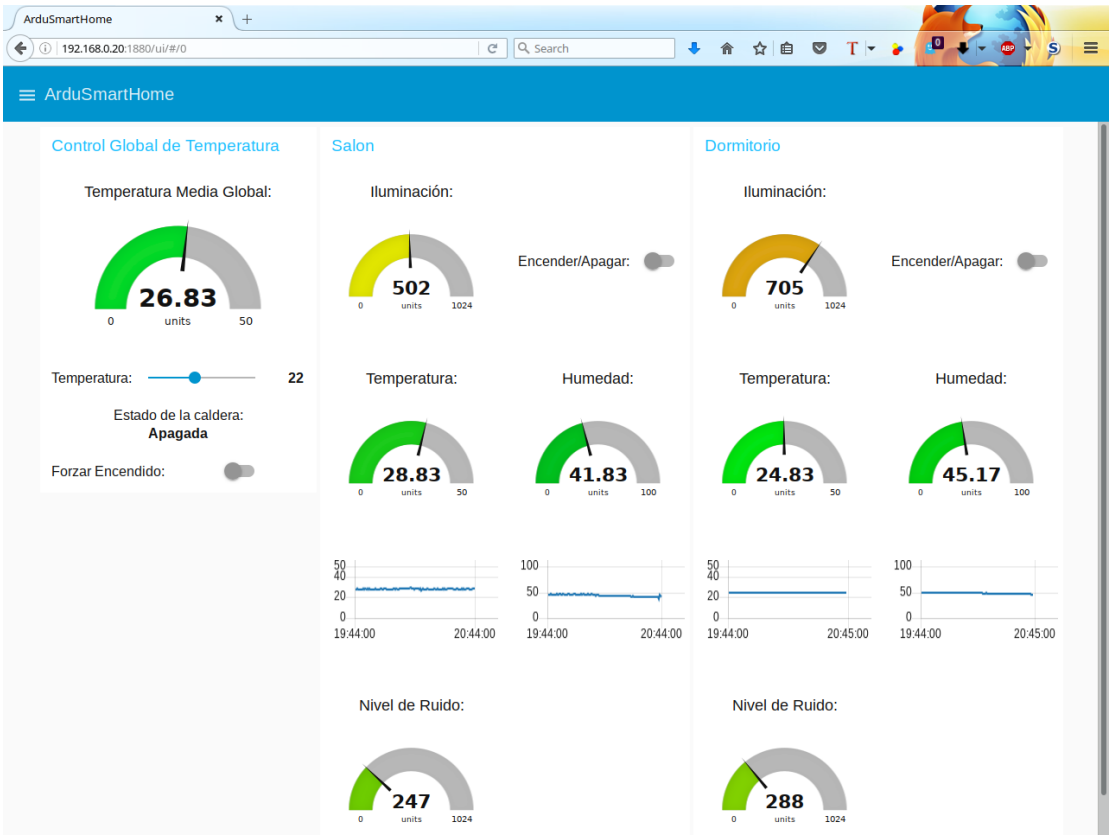


Ilustración 50: Web Dashboard

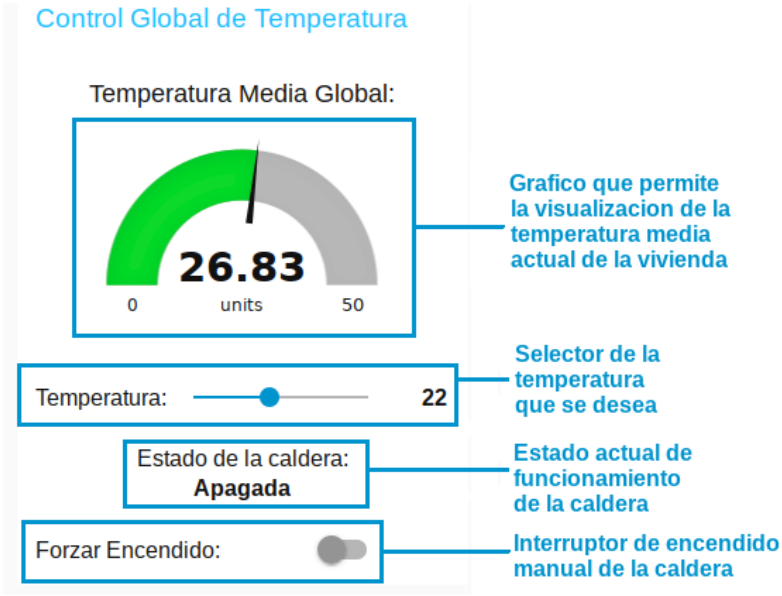


Ilustración 51: Disposición de controles del Control global de Temperatura

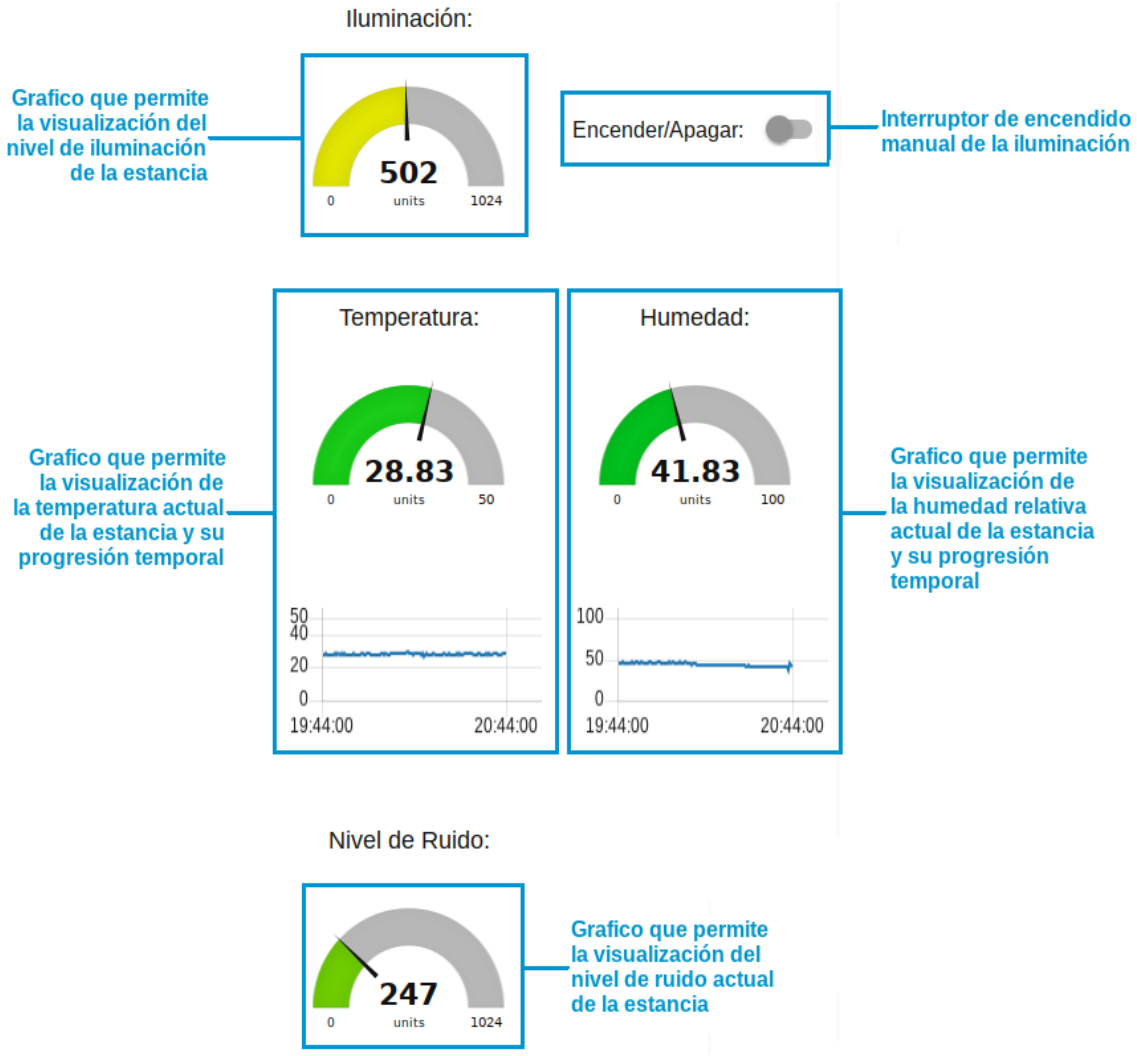


Ilustración 52: Disposición de controles de Zona

8.6. Anexo 6. Código Fuente.

```
/*
 * config.h
 *
 *          75.663 TFG - Arduino - 2016-17 - Estudios de Informatica
Multimedia y Telecomunicacion
 *
 *      Author: Miguel Angel Sanchez Munoz
 */

#ifndef CONFIG_H_
#define CONFIG_H_

#include <utility/EspDrv.h>

/*-------------------------------------------------
 *
 * Definitions for enable debug and serial port baudrate.
 *
 *-------------------------------------------------*/
//#define DEBUG
#define DEBUG_BAUD 115200

/*-------------------------------------------------
 *
 * Client information, ID, etc.
 *
 *-------------------------------------------------*/
const char* ID = "salon";

/*-------------------------------------------------
 *
 * Definitions for enable sensors/actuators.
 *
 *-------------------------------------------------*/
#define LDR
#define DHT
#define BUTTON
#define RELAY
#define MIC

/*-------------------------------------------------
 *
 * Data Queue type definition.
 *
 *-------------------------------------------------*/
typedef struct SensorData
{
      char* topic;
      double data;
} SensorData;

/*-------------------------------------------------
 *
 * Pinout definitions for hardware connection interface.
 *
 *-------------------------------------------------*/
#define RX 2 // SoftwareSerial[10] RX
#define TX 3 // SoftwareSerial[10] TX
```

```
#define BUTTON_PIN 4
#define DHT11_PIN 5
#define RELAY_PIN 6
#define MIC_PIN_D 7
#define LIGTH_PIN A0
#define MIC_PIN_A A1
#define LED_PIN 12

/*-----------------------------------------------------
 *
 * Control variables of the actuator devices.
 *
 *-----------------------------------------------------*/
bool RELAY_STATUS = false;
bool BUTTON_STATUS = true;
bool LED_STATUS = false;

/*-----------------------------------------------------
 *
 * Configuration for network
 *
 *-----------------------------------------------------*/
// Update these with values suitable for your network.
char ssid[] = "ArduSmartHome";            // your network SSID (name)
char pass[] = "ArduSmartHome";        // your network password
int status = WL_IDLE_STATUS;     // the Wifi radio's status
const char*  MQTT_SERVER = "192.168.240.1";
const int MQTT_PORT = 1883;
const char* MQTT_USER = "ardusmarthome";
const char* MQTT_PASS = "ardusmarthome";

/*-----------------------------------------------------
 *
 * MQTT[2] Topics
 *
 *-----------------------------------------------------*/
const char* TopicLightData = "light/data";
const char* TopicTempData = "temperature/data";
const char* TopicHumiData = "humidity/data";
const char* TopicRelayStatus = "relay/status";
const char* TopicRelaySet = "relay/set";
const char* TopicButtonStatus = "button/status";
const char* TopicMicData = "mic/data";

#endif /* CONFIG_H_ */


/*
 * main.cpp
 *
 *          75.663 TFG - Arduino - 2016-17 - Estudios de Informatica
Multimedia y Telecomunicacion
 *
 *      Author: Miguel Angel Sanchez Munoz
 */
/* Application includes. */
#include "config.h"
#include <Arduino.h>
#include <Taskplanificador.h>
#include <WiFiEspClient.h>
#include <WiFiEsp.h>
```

```
#include <PUBSubClient.h>
#include <QueueArray.h>
#include <dht.h>
#include <SoftwareSerial.h>

SoftwareSerial Serial1(RX, TX);
WiFiEspClient espClient;
PUBSubClient client(espClient);
QueueArray <SensorData> queue;
planificador runner, hprunner;

char TopicIDLightData[30];
char TopicIDTempData[30];
char TopicIDHumiData[30];
char TopicIDRelayStatus[30];
char TopicIDRelaySet[30];
char TopicIDButtonSet[30];
char TopicIDButtonStatus[30];
char TopicIDMicData[30];


/* Task functions and callback methods prototypes. */

      bool tMQTTSetup();
      void tMQTTClient();
      void MQTTCallback(char* topic, byte* payload, unsigned int
length);
      Task tMQTT(TASK_IMMEDIATE, TASK_FOREVER, &tMQTTClient,
&hprunner, false, &tMQTTSetup);

      bool tWifiSetup();
      void tWifiCallback();
      Task tWifi(TASK_MINUTE, TASK_FOREVER, &tWifiCallback, &runner,
false, &tWifiSetup);

#ifdef DHT
      bool tDHTSetup();
      void tDHTCallback();
      Task tDHT(TASK_SECOND * 5, TASK_FOREVER, &tDHTCallback, &runner,
false, &tDHTSetup);
#endif
#ifdef LDR
      bool tLightSetup();
      void tLightCallback();
      Task tLight(TASK_SECOND * 2, TASK_FOREVER, &tLightCallback,
&runner, false, &tLightSetup);
#endif
#ifdef BUTTON
      bool tButtonSetup();
      void tButtonCallback();
      Task tButton(TASK_IMMEDIATE, TASK_FOREVER, &tButtonCallback,
&runner, false, &tButtonSetup);
#endif
#ifdef RELAY
      bool tRelaySetup();
      void tRelayCallback();
      Task tRelay(TASK_IMMEDIATE, TASK_FOREVER, &tRelayCallback,
&runner, false, &tRelaySetup);
#endif
#ifdef MIC
      bool tMicSetup();
```

```
	void tMicCallback();
	Task tMic(TASK_SECOND * 2, TASK_FOREVER, &tMicCallback, &runner,
false, &tMicSetup);
#endif


#ifdef DHT
/*-----------------------------------------------------
 *@brief          Task Temperature/Humidity sersor fuctions. Captures
 *                      ambient Temperature/Humidity level.
 *
 * @param[in]     None
 *
 * @return        None
 *-----------------------------------------------------*/
dht DHT11;
SensorData temperature;
SensorData humidity;
int DHTLoop = 1;
int samples = 6;
double tempAverage = 0;
double humiAverage = 0;

bool tDHTSetup() {

	temperature.topic = TopicIDTempData;
	humidity.topic = TopicIDHumiData;
	return true;
}

void tDHTCallback() {
#ifdef DEBUG
	Serial.println("tDHTCallback");
#endif

	DHT11.read11(DHT11_PIN);	// READ DATA

	tempAverage = tempAverage + DHT11.temperature;
	humiAverage = humiAverage + DHT11.humidity;

	if (DHTLoop == samples) {

		temperature.data = tempAverage / samples;
		tempAverage = 0;
		humidity.data = humiAverage / samples;
		humiAverage = 0;

		if (!queue.isFull()) {
			queue.enqueue(temperature);
		}

		if (!queue.isFull()) {
			queue.enqueue(humidity);
		}
		DHTLoop = 0;
	}

	DHTLoop ++;
}
/*-----------------------------------------------------*/
#endif
```

```
#ifdef LDR
/*-----------------------------------------------------
 *@brief          Task Light sersor fuctions. Captures ambient light
level.
 *
 * @param[in]     None
 *
 * @return        None
 *-----------------------------------------------------*/
SensorData lightData;

bool tLightSetup() {

      lightData.topic = TopicIDLightData;

      return true;
}

void tLightCallback() {
#ifdef DEBUG
      Serial.println("tLightCallback");
#endif

      lightData.data = analogRead(LIGTH_PIN);

      if (!queue.isFull()) {
            queue.enqueue(lightData);
      }
}
/*-----------------------------------------------------*/
#endif

#ifdef BUTTON
/*-----------------------------------------------------
 *@brief          Task Button sersor fuctions. Human interacction.
 *
 * @param[in]     None
 *
 * @return        None
 *-----------------------------------------------------*/
SensorData buttonData;
bool BUTTON_OLD_STATUS = true;

bool tButtonSetup() {

      pinMode (BUTTON_PIN, INPUT);
      buttonData.topic = TopicIDButtonStatus;

      return true;
}

void tButtonCallback() {
#ifdef DEBUG
      Serial.println("tButtonCallback");
#endif

      BUTTON_STATUS = (bool)digitalRead(BUTTON_PIN);

      if(!BUTTON_STATUS){
            buttonData.data = !BUTTON_STATUS;
```

```
            if (!queue.isFull()) {
                  queue.enqueue(buttonData);
            }
      }
}
/*-----------------------------------------------------*/
#endif

#ifdef RELAY
/*-----------------------------------------------------
 *@brief          Task Relay actuator fuctions.Allows to operate with
 *                      large voltages, lamps or thermostats
 *
 * @param[in]     None
 *
 * @return        None
 *-----------------------------------------------------*/
SensorData relayData;
bool RELAY_OLD_STATUS = false;

bool tRelaySetup() {

      pinMode (RELAY_PIN, OUTPUT);
      relayData.topic = TopicIDRelayStatus;

      return true;
}

void tRelayCallback() {
#ifdef DEBUG
      Serial.println("tRelayCallback");
#endif

      digitalWrite(RELAY_PIN, RELAY_STATUS);

      if (RELAY_STATUS != RELAY_OLD_STATUS){
            RELAY_OLD_STATUS = RELAY_STATUS;
            relayData.data = RELAY_STATUS;
            if (!queue.isFull()) {
                  queue.enqueue(relayData);
            }
      }
}
/*-----------------------------------------------------*/
#endif

#ifdef MIC
/*-----------------------------------------------------
 *@brief          Task Micro sersor fuctions. Captures ambient sound
level.
 *
 * @param[in]     None
 *
 * @return        None
 *-----------------------------------------------------*/
SensorData micData;

bool tMicSetup() {

      pinMode (MIC_PIN_D, INPUT);
      micData.topic = TopicIDMicData;
```

```
	return true;
}

void tMicCallback() {
#ifdef DEBUG
	Serial.println("tMicCallback");
#endif

	micData.data = analogRead(MIC_PIN_A);

	if (!queue.isFull()) {
		queue.enqueue(micData);
	}
}
/*-----------------------------------------------------*/
#endif


/*-----------------------------------------------------
 *@brief		Task control network connexion fuctions. It makes
the connection
 *			with the Wi-Fi network and controls
periodically
 *
 * @param[in]	None
 *
 * @return		None
 *-----------------------------------------------------*/
bool tWifiSetup() {

	Serial1.begin(9600);
	WiFi.init(&Serial1);

	return true;
}

void tWifiCallback() {
#ifdef DEBUG
	Serial.println("tWifiCallback");
#endif

	if (WiFi.status() != WL_CONNECTED) {

		tMQTT.disable();

	status = WiFi.begin(ssid, pass);

		if(status == WL_CONNECTED){
#ifdef DEBUG
			Serial.print("Wifi connected: ");
			Serial.println(WiFi.localIP());
#endif
			tMQTT.enable();
		}
    }
}
/*-----------------------------------------------------*/

/*-----------------------------------------------------
 *@brief		Task Debug fuctions. Print debugging information
```

```
 *                      through the serial port.
 *
 * @param[in]     None
 *
 * @return        None
 *-----------------------------------------------------*/
SensorData data;
char msgPayload[8] = "";

void MQTTCallback(char* topic, byte* payload, unsigned int length) {
#ifdef DEBUG
      Serial.println("MQTTCallback");
#endif

  if (strcmp(topic, TopicIDRelaySet) == 0){
      if ( (char)payload[0] == '1') {
            RELAY_STATUS = true;
      } else {
            RELAY_STATUS = false;
      }
  }

}

bool tMQTTSetup() {

      pinMode (LED_PIN, OUTPUT);

      sprintf (TopicIDLightData, "%s/%s", ID, TopicLightData);
      sprintf (TopicIDTempData, "%s/%s", ID, TopicTempData);
      sprintf (TopicIDHumiData, "%s/%s", ID, TopicHumiData);
      sprintf (TopicIDRelaySet, "%s/%s", ID, TopicRelaySet);
      sprintf (TopicIDRelayStatus, "%s/%s", ID, TopicRelayStatus);
      sprintf (TopicIDButtonStatus, "%s/%s", ID, TopicButtonStatus);
      sprintf (TopicIDMicData, "%s/%s", ID, TopicMicData);

      client.setServer(MQTT_SERVER, MQTT_PORT);
      client.setCallback(MQTTCallback);

      return true;
}

void tMQTTClient() {
#ifdef DEBUG
      Serial.println("tMQTTClient");
#endif

      if (!client.connected()) {

            if (client.connect(ID, MQTT_USER, MQTT_PASS)) {
#ifdef DEBUG
                  Serial.println("MQTT[2] Client connect");
#endif
                  client.publish(ID, "hello");

#ifdef RELAY
                  client.subscribe(TopicIDRelaySet, 1);
#endif

            }
      } else {
```

```
            client.loop();

            if (!queue.isEmpty()) {
                  data = queue.dequeue();
                  dtostrf(data.data, 2, 2, msgPayload);
#ifdef DEBUG
                  Serial.print(data.topic);
                  Serial.println(msgPayload);
#else
                  client.publish(data.topic, msgPayload);
#endif
               }

      }

}
/*-----------------------------------------------------*/


/*-----------------------------------------------------
 *@brief          Main Setup application.
 *
 * @param[in]     None
 *
 * @return        None
 *-----------------------------------------------------*/
void setup() {
#ifdef DEBUG
      Serial.begin(DEBUG_BAUD);
#endif

      runner.init();

      runner.setHighPrioritypLanificador(&hprunner);

      runner.addTask(tMQTT);

      runner.addTask(tWifi);
      tWifi.enable();

#ifdef LDR
      runner.addTask(tLight);
      tLight.enable();
#endif
#ifdef DHT
      runner.addTask(tDHT);
      tDHT.enable();
#endif
#ifdef BUTTON
      runner.addTask(tButton);
      tButton.enable();
#endif
#ifdef RELAY
      runner.addTask(tRelay);
      tRelay.enable();
#endif
#ifdef MIC
      runner.addTask(tMic);
      tMic.enable();
#endif
```

```
}
/*-------------------------------------------------*/

/*-------------------------------------------------
 *@brief          Main Loop application.
 *
 * @param[in]     None
 *
 * @return        Function never end.
 *-------------------------------------------------*/
void loop() {
  runner.execute();
}
/*-------------------------------------------------*/
```