UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Internet de las cosas. Sistema electrónico de control basado en Arduino.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Javier Martínez Patiño

Tutor: José Vicente Busquets Mataix

Septiembre 2015

Agradecimientos

En primer lugar, quiero agradecer a mis padres y mi hermano su apoyo y su esfuerzo tanto durante la realización del proyecto como todos estos años, ya que sin ellos nada de esto habría sido posible.

En segundo lugar, agradezco el interés de mi familia para ayudarme en todo lo necesario para llevar a cabo el proyecto, en especial a mis abuelos por estar siempre dispuestos.

También quiero agradecer a mi tutor el tiempo dedicado y la ayuda que me ha ofrecido, estando siempre disponible para resolver las dudas o consultas que me han ido surgiendo durante el tiempo que ha durado el desarrollo de este proyecto.

Para no olvidarme de nadie, me gustaría agradecer a todo aquel que en algún momento se ha interesado por este proyecto o me ha apoyado de alguna forma ya que sin ellos tampoco habría sido posible terminarlo.

Y en último lugar, quiero expresar mi agradecimiento a la comunidad de desarrolladores de software libre, ya que gracias a ellos es posible realizar proyectos de este tipo con componentes complejos. Sin su ayuda, en el mejor de los casos se necesitaría muchísimo más tiempo para finalizarlos mientras que en el peor sería imposible llevarlos a cabo.

Resumen

El principal objetivo de este proyecto es implementar un sistema de Internet de las Cosas apoyándose en plataformas de hardware libre como son Arduino y NodeMCU, utilizando algunos de los componentes hardware disponibles para estas plataformas.

Se analizarán los protocolos que se están utilizando hoy en día para realizar la comunicación entre estos sistemas, desplegando una estructura que permita ponerlo en funcionamiento. También se profundizará en el control de algunos aparatos domésticos para integrarlos en el proyecto utilizando módulos relé para Arduino o incluso, atacando directamente al equipo si dispone de algún control remoto que funcione por infrarrojos como suele ocurrir con los aparatos de aire acondicionado.

Con todo esto será posible desplegar un sistema de Internet de las Cosas con orientación a la domótica distribuyendo hardware conectado a sensores que tomen información del medio y que, utilizando los protocolos adecuados y sus capacidades de red funcionando de forma autónoma, puedan transmitir esa información para que los demás elementos puedan actuar dependiendo de esos valores, activando los aparatos que se requiera en cada caso.

Además, se implementarán algunos servicios que permitirán el control manual de los aparatos deseados y la supervisión del sistema mediante el registro y posterior análisis de los eventos que se generen dentro de éste.

Para llevarlo a cabo, se presentarán una serie de soluciones software y configuraciones hardware, además de la arquitectura de comunicaciones necesaria para que todas las partes puedan comunicarse entre sí y el software y firmware necesario para cada uno de los componentes del sistema.

Palabras clave: Internet de las Cosas, Arduino, Microcontrolador, ESP8266, NodeMCU, MQTT, IR.

Tabla de contenidos

1. Introducción

1.1 Motivación

Internet de las Cosas (Internet of Things o IoT) es el término que se utiliza para referirse a la interacción entre objetos del mundo cotidiano que puedan ser identificables de manera unívoca como también a sus representaciones virtuales mediante un tipo de estructura de red clásica, sin importar la extensión de ésta (desde redes LAN hasta redes WAN o incluso Internet).

Estos objetos son capaces de intercambiar información entre ellos y dependiendo de esta información pueden actuar de manera autónoma con la mínima supervisión del ser humano, o incluso sin ella. Hoy en día esta tecnología está en alza y el número de dispositivos que pueden incluirse en este tipo de sistemas crece exponencialmente, ya que está concebido para trabajar sobre las actuales redes de comunicación y le aporta una evolución continua al mismo ritmo al que avanza Internet. Con la introducción de la tecnología IPv6 con la que se pretende modernizar la actual IPv4, el número de dispositivos que se pueden direccionar a través de Internet crece a 2^{128}, con lo que IoT se afianza como una solución de futuro.

Para llevar a cabo esta comunicación, que es el motor principal de esta tecnología, todos los elementos deben utilizar el mismo protocolo. Hoy en día existen numerosos protocolos de comunicación para implementar sistemas IoT debido a su potencial y a la rápida introducción que ha tenido en los últimos años en el mercado, por lo que algunas de las empresas tecnológicas más grandes del mundo están desarrollando sus propios sistemas e implementaciones.

Como se puede intuir, los campos de aplicación para el Internet de las Cosas son prácticamente ilimitados: recopilación de datos en tiempo real para control de tráfico, monitorización de pacientes, análisis meteorológico y ambiental y prevención de catástrofes naturales o incendios, etc. Además abre infinidad de campos de aplicación nuevos que hasta ahora eran impensables. Con el IoT se hacen mucho más accesibles sistemas que hasta ahora eran excesivamente caros y complejos de instalar. La domótica es un claro ejemplo de como ha sido posible reducir los costes valiéndose del IoT para implementar esta tecnología, dando lugar además a sistemas abiertos que se distancia de las soluciones propietarias que se encontraban en el mercado hasta la fecha.

1.2 Objetivos

El principal objetivo de este proyecto es utilizar plataformas de hardware libre como Arduino o NodeMCU para implementar un sistema de Internet de las Cosas con orientación domótica que pueda recoger datos mediante sensores, enviarlos utilizando la red y posteriormente recibir y procesar esos datos para llevar a cabo las acciones para las que se ha diseñado.

Para que todo ello funcione correctamente, es necesario implementar soluciones tanto para el hardware (Arduino o NodeMCU) como para el servidor, que se encarga de centralizar los datos obtenidos por cada elemento y distribuirlos para que todos puedan tener la información útil para su funcionamiento actualizada.

En el lado del servidor se van a implementar dos servicios básicos y se utilizará otro que permita comunicación entre dispositivos con capacidad para IoT, utilizando el protocolo que mejores soluciones ofrezca para este proyecto.

El primer servicio se encarga de gestionar todos los eventos críticos en el sistema para poder llevar un registro de éstos. Este servicio está a su vez conectado con una base de datos para guardar los registros de eventos y poder consultarlos posteriormente para analizar posibles accesos no autorizados o anomalías de funcionamiento en el sistema.

El segundo de ellos gestiona peticiones de clientes para permitir la interacción del usuario con la infraestructura del sistema. De este modo ofrecemos un front-end al usuario para poder enviar órdenes a un sistema, a priori, autónomo.

El servidor de comunicación es el núcleo de cualquier sistema IoT. Se emplea para organizar de forma eficiente los datos y realizar las conexiones con el protocolo adecuado entre los diferentes dispositivos que son capaces de obtener información y los que deben recibir esa información para poder realizar su tarea, de modo que cada elemento suscriptor de información únicamente recibe los datos (se suscribe) de la información valiosa para él y las comunicaciones necesarias se reducen de forma considerable.

En el lado de los clientes, se han distribuido dos placas Arduino programadas para controlar un sistema de aire acondicionado y un sistema calefactor respectivamente. Estas placas deberán contar con algún hardware adicional que pueda aportarles conectividad de red, ya que el potencial de la tecnología de Internet de las Cosas se basa en la capacidad de comunicación de los elementos para intercambiar información entre ellos. Estos clientes se conectan al servidor de comunicaciones para obtener los datos correspondientes al estado del sistema y poder controlar, dependiendo de estos datos, los equipos conectados a ellos. Los datos a su vez son

recogidos por una placa NodeMCU funcionando de forma autónoma, que está conectada a un sensor de temperatura y humedad.

Con toda esta infraestructura, el sistema permitirá gobernar de forma autónoma o de forma manual (ordenes enviadas directamente por el usuario) un relé conectado a un sistema calefactor y un aparato de aire acondicionado. Hay que tener en cuenta que el sistema de aire a controlar debe contar con una unidad de recepción de infrarrojos para poder comunicarnos con él, de cualquier otro modo el control sería muy complejo e inabordable en un proyecto de estas características. Para facilitar el control del aire, contamos con una pequeña pantalla que nos indica las instrucciones a seguir para que el funcionamiento sea el correcto.

1.3 Planificación

Para la realización del proyecto serán necesarias las siguientes tareas:

- Analizar los requisitos del proyecto para ofrecer una solución teniendo en cuenta las limitaciones y las funcionalidades que debe incluir.

- Estudiar las soluciones disponibles para sistemas de Internet de las Cosas y elegir la que mejor se adapte al proyecto en términos de eficiencia en las comunicaciones y compatibilidad con los componentes hardware de que se dispone.

- Configurar un servidor con capacidades para IoT que permita la transmisión y recepción de datos por parte de los diferentes elementos que integran el sistema.

- Implementar una serie de servicios que permitan operar con el sistema desde el exterior y registrar cualquier evento de importancia para este, ofreciendo un listado con esa información que se podrá consultar desde el cliente.

- Desplegar una infraestructura de comunicaciones que permita la interacción entre las diferentes partes para trabajar de forma coordinada.

- Desplegar una serie de clientes, tanto para adquirir datos como para controlar sistemas, de forma que podamos aplicar el sistema IoT a un sistema de control con orientación domótica.

- Analizar los protocolos de infrarrojos de los diferentes sistemas de aire acondicionado disponibles en el mercado para ofrecer una solución al control de este tipo de aparatos desde Arduino lo más amplia posible e integrarla posteriormente en nuestro sistema de control domótico.

2. Análisis y diseño

En este apartado se analizarán los requisitos que debe cumplir el proyecto y se analizarán y justificarán las estrategias a seguir para que al final del desarrollo el sistema sea capaz de cumplir con esos requisitos.

2.1 Requisitos del proyecto

Al ser un sistema de Internet de las Cosas, el proyecto debe cumplir dos requisitos principales. El primero de ellos es el desarrollo y puesta en marcha de un sistema con múltiples elementos que sean capaces de interactuar entre ellos mediante el intercambio de información utilizando una red local. Este es el requisito principal de cualquier proyecto IoT. Cumpliendo este requisito, la base de este proyecto queda cubierta. Sin embargo, el intercambio de información entre dispositivos sin un objetivo específico no aporta ningún valor. Por este motivo, se ha enfocado el uso de un sistema IoT al control de aparatos domésticos. El segundo requisito principal, por tanto, es el control telemático (tanto manual como automático) de diversos aparatos domésticos utilizando la infraestructura de Internet de las Cosas para el intercambio de información entre los diferentes elementos que conforman el sistema. Los dispositivos a controlar serán los siguientes: equipo de aire acondicionado que disponga de un receptor de infrarrojos para la comunicación con el control remoto (requisito indispensable para poder controlar el aparato), calefactor conectado a relé.

Cumpliendo estos dos requisitos se dispondrá de un sistema IoT con aplicaciones domóticas completamente funcional.

Adicionalmente, debe ser posible tener registrados todos los eventos que han ocurrido en el sistema, por lo que se hace necesario desarrollar un servicio para este fin.

2.2 Análisis y justificación del diseño

Para implementar un sistema de estas características, se plantean una serie de problemas que deben ser analizados cuidadosamente para obtener un resultado final satisfactorio. Se deben fijar los elementos que son necesarios y las tecnologías que ofrecen una mejor solución a la hora de

cumplir los requisitos fijados anteriormente. También es importante analizar los diferentes protocolos de comunicación IoT que existen actualmente en el mercado para valorar el uso de éstas o implementar un protocolo ad-hoc para este proyecto.

Al ser un proyecto en el que se debe trabajar tanto con software como con hardware, es imprescindible ser cuidadoso a la hora de elegir los componentes que se van a utilizar, ya que algunos ofrecen mejor rendimiento o mejor compatibilidad que otros con las soluciones IoT desarrolladas por las comunidades de usuarios y programadores que se van a emplear. Es muy común encontrar problemas a la hora de utilizar un protocolo con cierto módulo porque la librería que permite su uso no está adaptada para ese hardware. Debemos ser conscientes de esto en todo momento, ya que es algo muy común en este tipo de plataformas abiertas que tienen soporte de una comunidad heterogénea de desarrolladores.

Teniendo en cuenta todo esto, el siguiente paso es comenzar a aclarar algunos aspectos del diseño e implementación. Aquí se analizarán algunas de las soluciones disponibles y se tomará la decisión sobre cuál de ellas es la más adecuada para el caso que nos ocupa.

Soluciones de diseño hardware:

En primer lugar, debemos dar una solución al problema de conectividad de red de las placas Arduino. Si bien es cierto que esta plataforma dispone de modelos con conectividad de red nativa, el modelo del que se dispone para este proyecto no cuenta con esta característica. Por ello, la primera elección de diseño está enfocada a este aspecto. La popularidad de la plataforma Arduino ha motivado la aparición de infinidad de hardware que le aporta a estas placas algunas características muy interesantes. Hoy en día existen periféricos para Arduino con conectividad de red tanto cableada como inalámbrica.

Aquí debemos tomar la primera decisión: proporcionar conectividad de red utilizando cableado Ethernet o utilizar una solución inalámbrica. Para sistemas IoT, donde lo que buscamos es la integración de un gran número de dispositivos en una red para transmitir información entre ellos, la opción más sensata es utilizar conectividad de red inalámbrica. Esto nos permite una flexibilidad a la hora de colocar los dispositivos muy superior a la que nos ofrece la solución cableada, en la que es necesario pasar cable y colocar una toma al lado de cada dispositivo, encareciendo el precio final del proyecto y dificultando la movilidad de los elementos en caso de que así se requiera.

Una vez tomada la decisión, hay que elegir el hardware a utilizar. Existen varias soluciones actualmente para proporcionar conectividad WiFi a las placas Arduino. La más conocida es la Arduino WiFi Shield, la solución desarrollada y soportada por la comunidad Arduino. Sin embargo, su precio hace necesario buscar una opción más económica. Por suerte ha aparecido

en el mercado un chip WiFi con un precio muy reducido que cumple perfectamente su función, lo que lo convierte en una solución firme para este proyecto. Este chip ofrece la posibilidad de funcionar en modo autónomo y gestionar hardware muy simple, lo que también será útil para ubicar sensores donde necesitemos. El chip en cuestión es el ESP8266 y se hablará de él posteriormente.

Para el control del aparato de aire acondicionado necesitamos un receptor y un led emisor de infrarrojos. Con esto se implementará en Arduino un programa que permitirá enviar las órdenes y ajustar la temperatura de este. También necesitaremos un módulo relé para el control del calefactor, que será gestionado por otra placa. Todos estos elementos son muy económicos.

Por último, los datos de temperatura y humedad que serán los que utilice el sistema para el control del aire en modo automático serán proporcionados por un sensor de temperatura y humedad DHT11, un sensor ampliamente utilizado en proyectos de este tipo y con un gran soporte de librerías para su manejo.

Soluciones de diseño software:

En este apartado debemos estudiar los elementos software necesarios para llevar a cabo el proyecto. Para no entrar en demasiados detalles, de los que se hablará posteriormente, se van a citar los componentes software que integran el sistema y las tecnologías que hay detrás de estos.

Cuando hablamos del software en un proyecto de Internet de las Cosas el primer paso es analizar qué solución de comunicación entre elementos se va a utilizar. Existen infinidad de opciones disponibles para este tipo de sistemas, algunas de código abierto y otras propietarias. También existe la posibilidad de crear un protocolo propio basándose en tecnologías de red como son TCP y UDP.

De entre todos los protocolos de comunicación disponibles que se han investigado, se ha elegido para la realización del proyecto el protocolo MQTT. Su elección ha sido debida a que es un protocolo con gran soporte de la comunidad de usuarios, contando con múltiples implementaciones de éste tanto para servidores como para clientes de todo tipo.

Una vez los elementos del sistema son capaces de enviar y recibir información, debemos ser capaces de crear un punto de entrada al sistema desde el exterior que gestione las peticiones de los usuarios y las transfiera al sistema IoT de tal forma que éste pueda tratarlas. Esto es conocido como front-end. Además de este servicio, también se implementará un servidor de registros para almacenar todos los eventos del sistema almacenado información tal como el dispositivo o usuario que lo ha generado y el valor de ese evento, contando para ello con una

conexión a una base de datos relacional. Los dos servicios serán implementados en el lenguaje Python debido a la cantidad de librerías y utilidades de que dispone para la conexión con bases de datos y los frameworks que ofrece para desplegar servicios de un modo muy sencillo. Para la base de datos se utilizará un servidor MySQL conectado al servidor de registros donde se almacenarán todas las entradas que éste reciba. La comunicación con estos dos servicios se realizará utilizando una arquitectura REST, lo que permite la implementación de ambos de una forma muy cómoda y facilita la gestión de las peticiones.

Con las tecnologías, dispositivos, componentes y herramientas mencionados anteriormente el sistema IoT será capaz de gestionar diversos aparatos domésticos con o sin la supervisión de un usuario de forma telemática.

El usuario será capaz de elegir el modo de funcionamiento del aire acondicionado, pudiendo seleccionar entre automático y manual. En automático, el Arduino conectado a él será capaz de analizar la temperatura ambiente de la habitación y ajustar el funcionamiento del aire. En modo manual, será el usuario el que pueda fijar que temperatura de funcionamiento quiere o incluso apagarlo. También podrá conectar o desconectar el calefactor conectado al relé interactuando con el sistema mediante el front-end.

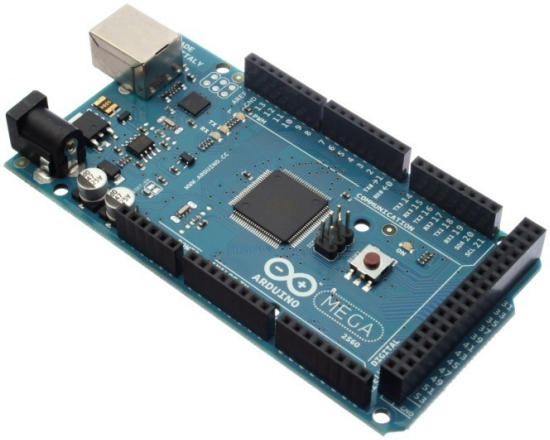
3. Material utilizado

A continuación se describirán los componentes hardware que se van a utilizar en este proyecto. La placa Arduino era un componentes base de este proyecto. El resto de componentes se han elegido teniendo en cuenta los aspectos analizados en el apartado anterior.

3.1 Arduino Mega 2560

Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos destinados a múltiples disciplinas y niveles de complejidad. Esto le aporta una versatilidad que la sitúa como una de las mejores soluciones hardware para proyectos que no requieran sistemas de cómputo complejos.

La plataforma cuenta con diferentes tipos de placas para ajustarse a las necesidades del usuario. En este proyecto se ha utilizado el Arduino Mega 2560, una placa basada en el microcontrolador Atmega 2560 del fabricante Atmel incluido en su familia de microprocesadores con arquitectura RISC de ocho bits y que se caracteriza por su bajo consumo. La principal diferencia con las otras placas es la cantidad de pines que ofrece y cuenta con más cantidad de memoria, lo que nos permite manejar más dispositivos utilizando librerías específicas en un mismo firmware o almacenar en la memoria interna mayor cantidad de valores para aplicaciones complejas.



Arduino Mega 2560 revisión 3

Las características más destacables de esta placa son las siguientes:

Característica	Descripción
Micrcontrolador	Atmega 2560
Voltaje de trabajo	5V
Voltaje de entrada	6-20V
Alimentación de la placa	Mediante cable USB o entrada dedicada
Pines digitales	54
Pines digitales con soporte PWM	12
Pines de entrada analógica	16
Corriente máxima de salida para pines de E/S	40mA
Corriente máxima para pines de alimentación	200mA
Memoria flash	256 KB (8KB están reservados al bootloader)
SRAM	8KB
EEPROM	4KB
Velocidad del reloj	16MHz
Tensiones de salida de alimentación	5 y 3.3V

3.2 Módulos WiFi

Para poder desplegar los sensores y comunicar los componentes del sistema sin la necesidad de cablear, se ha recurrido a diferentes módulos con conectividad WiFi. Estos módulos trabajan tanto en modo esclavo con otro hardware como de forma independiente.

El chip WiFi ESP8266 es un SoC con la pila de protocolos TCP / IP integrada para proporcionar comunicación WiFi. El ESP8266 es capaz de guardar código en su memoria interna para ejecutarlo por sí mismo, o puede funcionar como esclavo recibiendo las instrucciones por la interfaz serie. En el mercado lo podemos encontrar integrado en diferentes módulos, que le proporcionan pines para poder conectarse con un dispositivo maestro y/o con diferentes esclavos.

Este chip suele tener preinstalado un firmware para poder controlarlo mediante comandos AT. Con este firmware es capaz de realizar diferentes funciones de red, como crear un servidor web. Sin embargo, si se requieren protocolos más avanzados (MQTT en este proyecto), se hace necesario sustituir este firmware por otro con soporte a esos protocolos.

Hay que tener en cuenta que este chip trabaja a 3.3V, por lo que debemos ser cuidados al conectarlo con cualquier sistema electrónico que no trabaje con estos niveles de tensión y asegurarnos de realizar las conversiones eléctricas necesarias con los dispositivos pertinentes o con algún montaje destinado a tal fin (en este proyecto se ha utilizado un divisor resistivo para proteger el pin de recepción del módulo de la salida de Arduino).

Para este proyecto se han utilizado dos modelos diferentes de este chip integrados en dos módulos. Cada uno de ellos ofrece unas características que veremos a continuación y que lo hacen más adecuados para situarlos en un lugar o en otro en este proyecto.

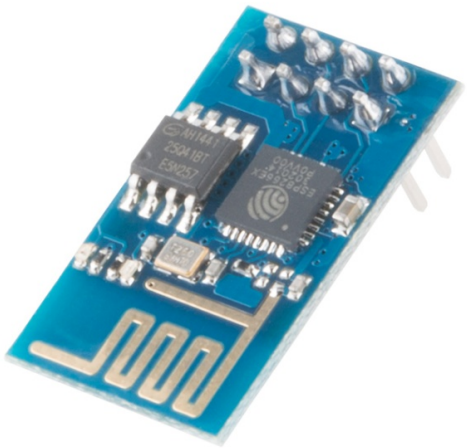
3.2.1 ESP8266 versión 01

Es el más económico de los que hay disponibles en el mercado, pero ofrece unas características suficientes para aprovechar las capacidades de comunicación del ESP8266. Dispone de dos pines GPIO y los pines Tx y Rx utilizados por la interfaz serial para comunicarse con el chip.

Es ideal para funcionar como esclavo de otro hardware más potente, como puede ser Arduino. También es capaz de funcionar de forma autónoma, pero está bastante limitado por el reducido número de pines GPIO de que dispone.

Por todo ello, este módulo funcionará como esclavo de las placas Arduino por su pequeño tamaño, proporcionando conectividad de red y ofreciendo una conexión al servidor de datos IoT.

Existe un aspecto a tener en cuenta con este módulo y es que requiere una fuente de alimentación externa a Arduino. Esto es debido al consumo de corriente que presenta en su arranque, el cual puede llegar a picos de 300 o 400mA. Como ya hemos visto, la placa Arduino tiene una corriente máxima de salida de 200mA, lo que puede acarrear problemas en la inicialización y el arranque del módulo.

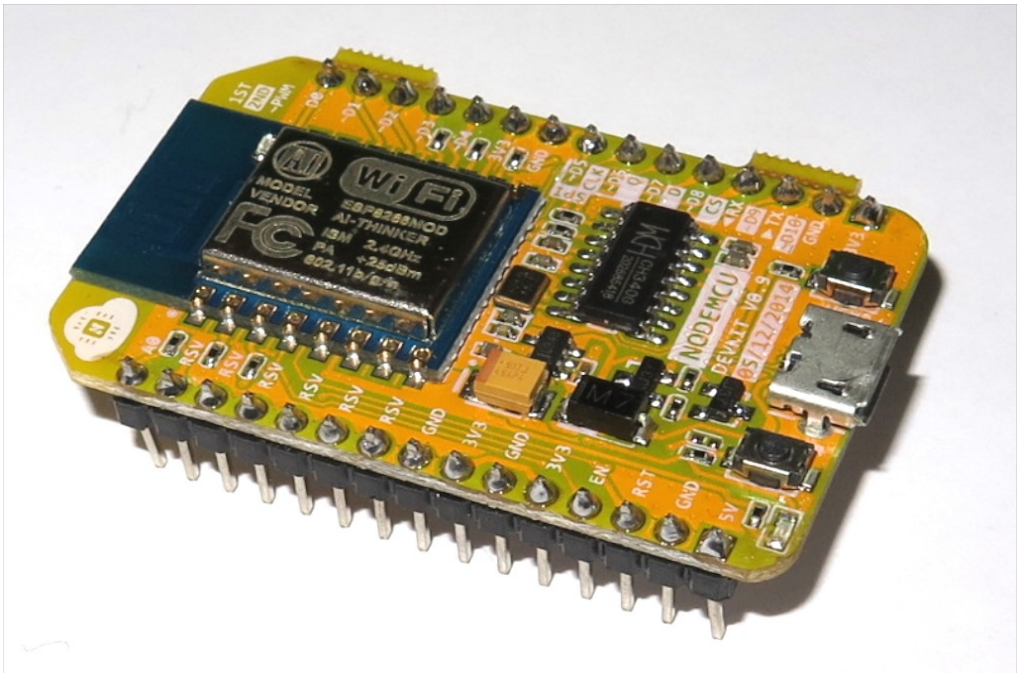


Módulo ESP8266 versión 01

3.2.2 NodeMCU dev board

Es una placa diseñada para integrar el módulo WiFi ESP8266-12, uno de los más extendidos dada la cantidad de pines de los que dispone y que le aportan gran capacidad para manejar sensores y actuadores por sí mismo. Además de este chip WiFi, tiene otros componentes y pines que aportan una sencillez de programación que no aportan los módulos ESP, ya que esta placa la lleva integrada una interfaz USB a serial (disponible a través de un puerto microUSB, por donde también se puede alimentar). También lleva incorporadas entradas analógicas conectadas a un convertidor A/D, lo que le permite ser más versátil que cualquier módulo ESP de forma autónoma.

Es una solución que se adapta muy bien a infinidad de proyectos y que ofrece muchas posibilidades a los desarrolladores por un precio asequible (se puede encontrar por poco más de 8€), que guarda más parecido con hardware como Arduino que con los otros módulos ESP que se pueden encontrar en el mercado.



NodeMCU dev board integrando el módulo ESP8266 versión ESP-12

3.3 Emisor y receptor IR

Para poder realizar el control del aire acondicionado, debemos tener en cuenta dos aspectos básicos. El primero de ellos es asegurarnos de que el sistema que vamos a controlar se puede manejar mediante un emisor de infrarrojos y se dispone del control remoto original. Si esto no se cumple, va a ser imposible controlarlo utilizando las herramientas y soluciones ofrecidas en este proyecto. Una vez estamos seguros de contar con un sistema de aire compatible, el segundo requisito es conectar al Arduino algún hardware que permita el envío y recepción de órdenes utilizando este protocolo. Esto será posible utilizando un led emisor y un receptor de infrarrojos.

3.3.1 Receptor de infrarrojos

Un receptor de infrarrojos, o receptor IR, es un hardware que recibe información de un control remoto por infrarrojos a otro dispositivo mediante la decodificación de señales. En general, el

receptor emite un código para identificar de forma exclusiva la señal de infrarrojos que recibe. Este código se utiliza entonces para convertir las señales desde el mando a distancia en un formato que puede ser entendido por el dispositivo conectado a él.

De esta forma, seremos capaces de identificar el código que emite el mando original del aparato y posteriormente, utilizando los mecanismos y técnicas que creamos más convenientes, emitirlos desde nuestro Arduino para manejar el equipo de aire.

En este caso el receptor utilizado es un VS1838B, un dispositivo muy económico disponible en tiendas de electrónica o en Internet. Con este receptor y el software adecuado, seremos capaces de ver y analizar las órdenes que emite el control remoto original del aire acondicionado.

Es un hardware muy simple, únicamente dispone de tres pines. Uno de ellos corresponde a la señal con los datos obtenidos y los otros dos corresponden a la alimentación. El pin de la señal se puede conectar a cualquier pin digital de Arduino y con el software correspondiente se decodificará la señal.

Receptor IR modelo VS1838B

3.3.2 Emisor de infrarrojos

Para poder enviar órdenes hacia el aparato de aire acondicionado, lo único que necesitamos es un dispositivo capaz de emitir luz infrarroja en la frecuencia adecuada. Utilizando este elemento con Arduino y un software que nos permita emitir esa luz de la forma que nosotros queramos, podremos enviar órdenes para controlar el sistema de aire y que éste las pueda interpretar.

Por tanto, la solución más sencilla para emitir luz es utilizar un LED. Un LED es básicamente un dispositivo con un material semiconductor que es capaz de emitir radiación electromagnética. Si bien es cierto que los LED que estamos acostumbrados a ver emiten radiación electromagnética del espectro de la luz visible, también existen algunos capaces de emitir radiación con frecuencias inferiores a este espectro (conocida como luz infrarroja).

Con esto seremos capaces de emitir inrofmación de forma que el aparato de aire sea capaz de recibirlas e interpretarlas del mismo modo que lo haría con las órdenes emitidas por el control remoto original del propio aparato.

LED emisor de infrarrojos

3.4 Pantalla LCD Nokia 5110

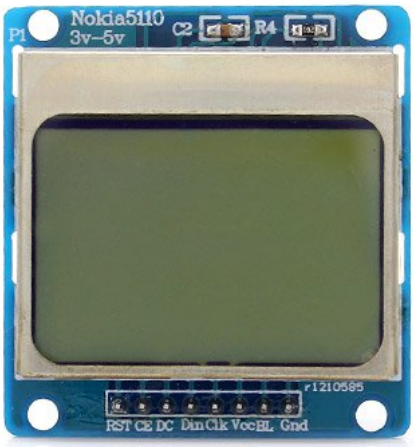
Este proyecto tiene como base la aplicación de un sistema IoT al control de ciertos aparatos domésticos. El control se realiza de forma telemática o de forma automática por parte del sistema, sin embargo, en ciertos momentos el sistema necesita la interacción del usuario para el arranque y su correcto funcionamiento posterior. Es por esto que necesitamos algún elemento hardware que nos aporte la capacidad de comunicar instrucciones al usuario para facilitar las tareas que éste deba realizar.

Para poder mostrar mensajes desde el sistema la mejor solución es usar una pantalla. De este modo, tendremos la posibilidad de poder mostrar mensajes una vez los elementos estén ubicados en su lugar sin la necesidad de conectar un PC u otro elemento capaz de leer mensajes por la interfaz serie de la placa Arduino.

En este caso, se va a utilizar un módulo con una pantalla LCD bastante simple igual a la que se utilizaba en los dispositivos móviles Nokia 5110. Este módulo dispone de los siguientes pines para su control desde un dispositivo maestro:

- RST: Permite el reseteo de la pantalla.
- CE: Señal de habilitación del módulo.
- DC: Pin de modo de recepción datos/comandos.
- Din: Pin de recepción de datos.
- Clk: Entrada de reloj.
- Vcc: Pin de alimentación del módulo.
- BL: Pin de control de la retroiluminación de la pantalla.
- GND: Pin de conexión a masa.

El módulo está diseñado para trabajar con voltajes entre 2.7 y 3.3V, por lo tanto puede ser alimentado directamente desde nuestra placa. Sin embargo, a la hora de la conexión de los pines de control, se debe en cuenta que hay que proteger las entradas del módulo de las salidas digitales de la placa microcontroladora para evitar dañarla debido a que la corriente de salida de Arduino es superior a la corriente de entrada que tolera el módulo LCD.



Módulo pantalla Nokia 5110

3.5 Conversor USB a TTL

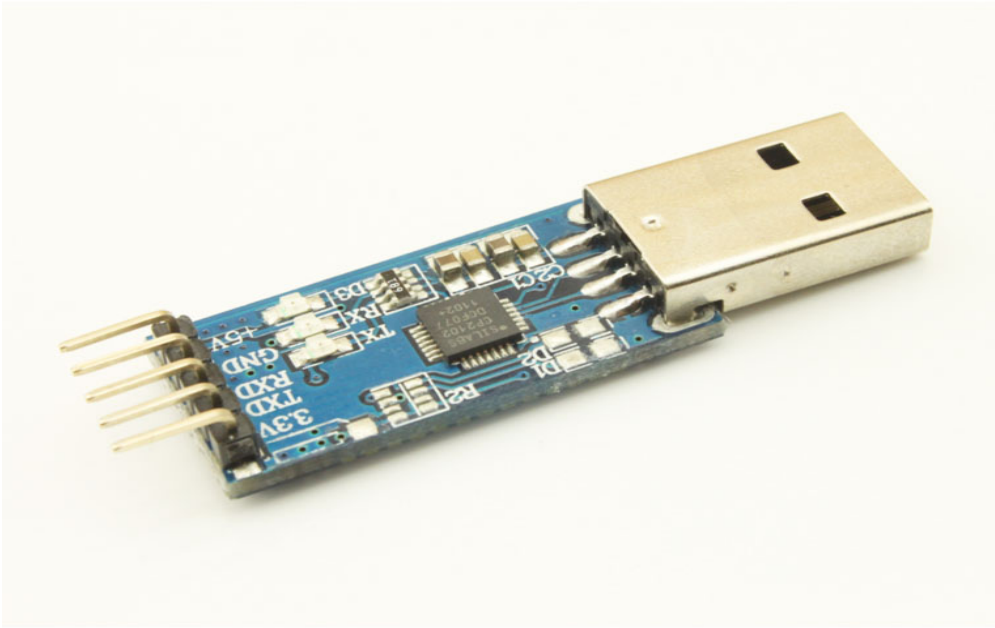
Una vez hemos hablado del hardware que se empleará en este proyecto, únicamente nos queda hablar de una herramienta imprescindible sin la que no sería posible comunicarse con ciertos componentes para sustituir su firmware o enviarles el código que deberán ejecutar. Esta herramienta es el convertidor USB a TTL.

Básicamente, podríamos decir que un convertidor USB a TTL es un dispositivo que se conecta a un puerto USB (normalmente de un PC) y a otro hardware con una interfaz RS-232 y permite la comunicación entre ambos transformando los paquetes del protocolo USB a paquetes RS-232 y a la inversa, de forma que es posible la comunicación entre dos dispositivos que, en principio, es imposible comunicar.

Además de esto, también incluye niveles de alimentación de 3.3 y 5V acompañados de un nivel de masa, que nos permiten usar este módulo como fuente de alimentación para los módulos WiFi.

El conversor dispone de dos interfaces, la interfaz USB de la que también recibe alimentación y la interfaz RS-232. La interfaz RS-232 dispone, a su vez, de 5 pines que se detallan a continuación:

- 5V: Pin con tensión de 5V para alimentación de circuitos
- 3.3V: Pin con tensión de 3.3V para alimentación de circuitos.
- Rx: Pin de recepción de datos del protocolo RS-232.
- Tx: Pin de transmisión de datos del protocolo RS-232.
- GND: Pin con tensión de 0V.



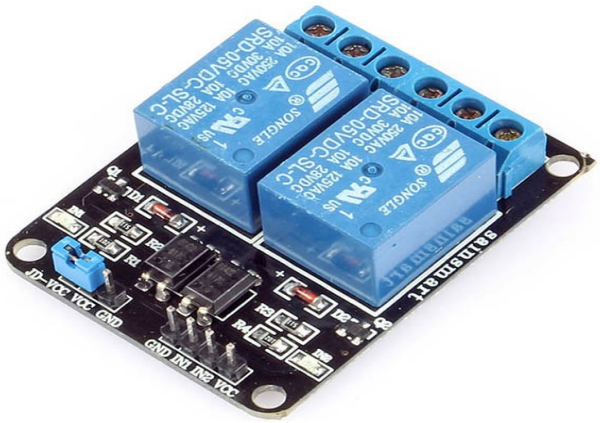
Convertidor USB a TTL

3.6 Módulo relé HL-52S

Para poder controlar diferentes sistemas eléctricos utilizando componentes electrónicos, necesitamos algún dispositivo que permita abrir y cerrar circuitos eléctricos con voltajes altos y intensidades de corriente mayores que las que soporta cualquier circuito electrónico. Para ello disponemos de los relés.

El relé es un dispositivo electromagnético. Funciona como un interruptor controlado por un circuito eléctrico en el que, por medio de una bobina y un electroimán, se acciona un juego de uno o varios contactos que permiten abrir o cerrar otros circuitos eléctricos independientes. La gran ventaja de los relés electromagnéticos es la completa separación eléctrica entre la corriente de accionamiento, la que circula por la bobina del electroimán, y los circuitos controlados por los contactos, lo que hace que se puedan manejar altos voltajes o elevadas potencias con pequeñas tensiones de control.

En este proyecto, se ha utilizado un módulo que integra dos relés y está especialmente diseñado para su manejo desde placas microcontroladas.



Módulo con dos relés modelo HL-52S.

4. Herramientas utilizadas

Para llevar a cabo este proyecto, es imprescindible contar con diferentes herramientas que nos van a permitir programar tanto el hardware como los servicios software necesarios y poner todo ello en funcionamiento.

4.1 Firmware adicional para NodeMCU y ESP8266

Para poder comunicarnos con el servidor MQTT desde los clientes, necesitamos cambiar el firmware AT que vienen por defecto en los módulos ESP8266, ya que este no nos proporciona soporte para este tipo de protocolos de comunicación avanzados. Existen varios firmware que nos aportan estas características, entre otras muchas, que el firmware original no ofrece.

Después de un largo proceso de investigación en el que se han analizado las soluciones más extendidas por la red para dotar a este chip de la capacidad de comunicarse utilizando el protocolo MQTT, se han obtenido unas cuantas soluciones que nos lo permiten. No obstante, hay que tener en cuenta que son soluciones desarrolladas por usuarios, por lo que no han cumplido ningún estándar de calidad ni se han sometido a las pruebas necesarias para considerarlas como soluciones totalmente operativas. Sin embargo, las utilizaremos dado que son completamente funcionales y que este proyecto tiene un objetivo académico, siendo un aspecto interesante el hecho de trabajar e investigar con soluciones software creadas por otros desarrolladores.

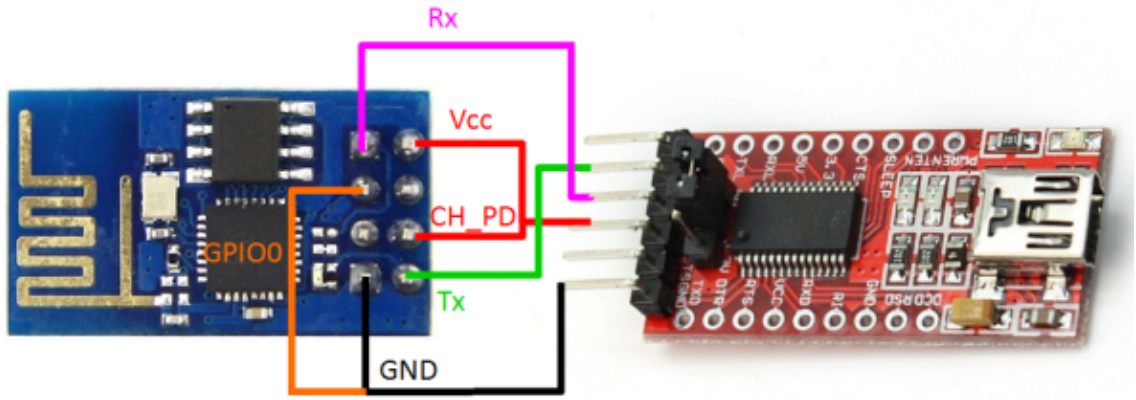
En este proyecto se han utilizado dos firmwares distintos:

- ESP_bridge: firmware puente que esta desarrollado para trabajar con la librería Espduino diseñada para la plataforma Arduino. Recibe las órdenes desde las funciones de esta librería mediante comandos SLIP y realiza las conexiones necesarias. El firmware unido a la librería permiten al módulo enviar y recibir información con el protocolo MQTT y con la arquitectura REST.

- NodeMCU firmware: firmware diseñado para la plataforma NodeMCU. Este firmware incluye un sistema de ficheros y permite la ejecución de código al arranque por parte del chip ESP8266, lo que le aporta un modo de funcionamiento autónomo muy interesante. Este firmware incluye una API completa que incluye desde funcionalidades

básicas de red hasta comunicación mediante protocolos avanzados como MQTT. Adicionalmente permite el control de hardware simple utilizando los pines GPIO de los módulos, pudiendo controlar un gran número de dispositivos por sí mismo.

Para poder introducir este nuevo firmware en los módulos WiFi (término conocido como flashear), es imprescindible contar con un adaptador que nos permita comunicar un dispositivo con interfaz USB con otro con interfaz RS-232. Esta función la realiza el conversor USB a TTL, aunque también sería posible con un conversor FTDI. Para que el módulo ESP reconozca que lo que estamos haciendo es introducir un nuevo firmware en su memoria interna, debemos conectar los pines de la siguiente forma:



Cableado necesario para sobrescribir el firmware

Para poner el ESP en modo flash, debemos conectar el pin GPIO 0 a GND. De esta forma grabará en las direcciones de memoria reservadas al firmware la información que reciba por la interfaz serie.

Una vez conectado el conversor al PC y al ESP, es el momento de enviar el código desde el primero hacia el segundo. Existen diferentes utilidades desarrolladas por la comunidad, pero para este caso vamos a utilizar un código en Python que viene incluido en la librería Espduino y se lanza desde la terminal. Con este programa es posible enviar un fichero binario a una dirección de memoria determinada del módulo, con lo que se podrá modificar cualquier dirección de memoria. Permite “flashear” más de un binario en la misma orden escribiendo la dirección de memoria donde guardar el fichero binario y a continuación indicando la ruta de ese fichero en el PC. El formato de la orden es el siguiente:

[nombre de la utilidad en python] -p [puerto al que está conectado el conversor USB a serie] write_flash [dirección de memoria donde almacenar el fichero 1] [ruta del fichero 1] ...

Un ejemplo concreto extraído del uso de la herramienta en el proyecto sería la siguiente:

./esptool.py -p /dev/ttyUSB0 write_flash 0x00000 ../release/0x00000.bin 0x40000 ../release/0x40000.bin

Cuando el proceso haya finalizado, es imprescindible desconectar el pin GPIO 0 de GND, de lo contrario el módulo arrancará de nuevo en modo flash.

Con el nuevo firmware disponible en los módulos ESP, pudiendo así comunicarse todos ellos utilizando el protocolo y el servidor MQTT, ya podemos comenzar a implementar los códigos de los clientes Arduino y de la placa NodeMCU para darles la funcionalidad deseada.

4.2 Herramientas de programación del hardware.

En este apartado se hablará de las herramientas que nos permiten editar, cargar y analizar el código necesario para que funcionen los componentes hardware del sistema, tanto las placas Arduino como los módulos ESP8266 en modo esclavo y autónomo.

4.2.1 Programación sobre la plataforma Arduino.

La plataforma Arduino está basada en el lenguaje C y soporta todas las funciones del estándar C y C++. Dispone de una amplia diversidad de lenguajes de programación de alto nivel, además de los mencionados como C#, Java, Python, Perl, PHP, Visual Basic, Matlab, entre otros.

Los programas compilados con Arduino se enlazan contra la librería AVR Libc. AVR Libc es un proyecto de software libre que tiene como objetivo proporcionar una librería C para microcontroladores Atmel AVR que pueda ser utilizada con el compilador GCC. Consta de tres partes fundamentales:

- avr-binutils: Es una colección de programas empleados para la manipulación de objetos binarios creados para micocontroladores con arquitectura Atmel AVR.

- avr-gcc: Compilador GNU C para microcontroladores con arquitectura Atmel AVR.

- avr-libc: Contiene librerías estáticas y los archivos de cabecera necesarios para el proceso de compilación.

Características específicas de la API de AVR Libc

La API de la librería AVR Libc dispone de algunas funciones de programación específicas para microcontroladores. A continuación se exponen algunos ejemplos:

Las primeras corresponden a las señales de interrupción:

- *cli()*: Desactiva las interrupciones globales (Clear Interrupts).
- *sei()*: Activa las interrupciones globales (Set Interrupts).

Estas instrucciones afectan al temporizador y a la comunicación en serie. La llamada a la función *delayMicroseconds()* desactivaría las interrupciones.

Otro punto interesante son las funciones de temporización. Existen diversas funciones que nos permiten generar retardos en el sistema. Usualmente estos retardos se usan para tareas de sincronización o espera. Para ello tenemos disponible la función *delay()* que introduce un retardo en segundos igual al argumento pasado, pero para obtener más precisión, la función *delayMicroseconds()* genera el menor retardo posible en Arduino y ronda los 2 microsegundos. Sin embargo, es posible reducir este retardo utilizando la instrucción en ensamblador *nop()*, que consume un ciclo de reloj sin realizar ningún trabajo efectivo. Teniendo en cuenta que el reloj del Arduino opera a 16 Mhz, la duración (período) de un ciclo de reloj es de aproximadamente 62.5 nanosegundos.

Por último, existen definiciones como *cbi* y *sbi* que son mecanismos estándar de AVR cuya utilidad es la de facilitar la manipulación de bits dentro de variables, principalmente de tipo PORT.

Librerías

Como en todo lenguaje de programación, en Arduino también es posible la inclusión de librerías propias que proporcionen una funcionalidad extra y así favorecer el desarrollo modular de aplicaciones. Existen una serie de librerías estándar ofrecidas por la plataforma software Arduino. En el caso de necesitar soporte para hardware adicional, lo más común es que sea necesario incluir en el código la librería que aporte la funcionalidad en concreto para dicho hardware.

Estructura de programación en Arduino

Al estar inspirado en el lenguaje y entorno de programación Wiring, que a su vez está inspirado en la plataforma de programación Processing, sigue el patrón de diseño de éstos:

```
void setup(){
        //Código de la función setup
}

void loop(){
        //Código de la función loop
}
```

La función *setup()* se usa para tareas de inicialización de los sistemas conectados a la placa, puertos de comunicación, variables, etc. Esto es debido a que esta función es ejecutada una única vez por la placa después del arranque o reseteo.

Una vez se ha finalizado la ejecución de esta función, la placa pasará a ejecutar la función *loop()*.

Esta segunda función se ejecuta de manera cíclica y es la que contiene el código principal de las aplicaciones. Su ejecución en forma de bucle infinito nos permite responder de forma periódica a los eventos que puedan ocurrir dentro o fuera del sistema.

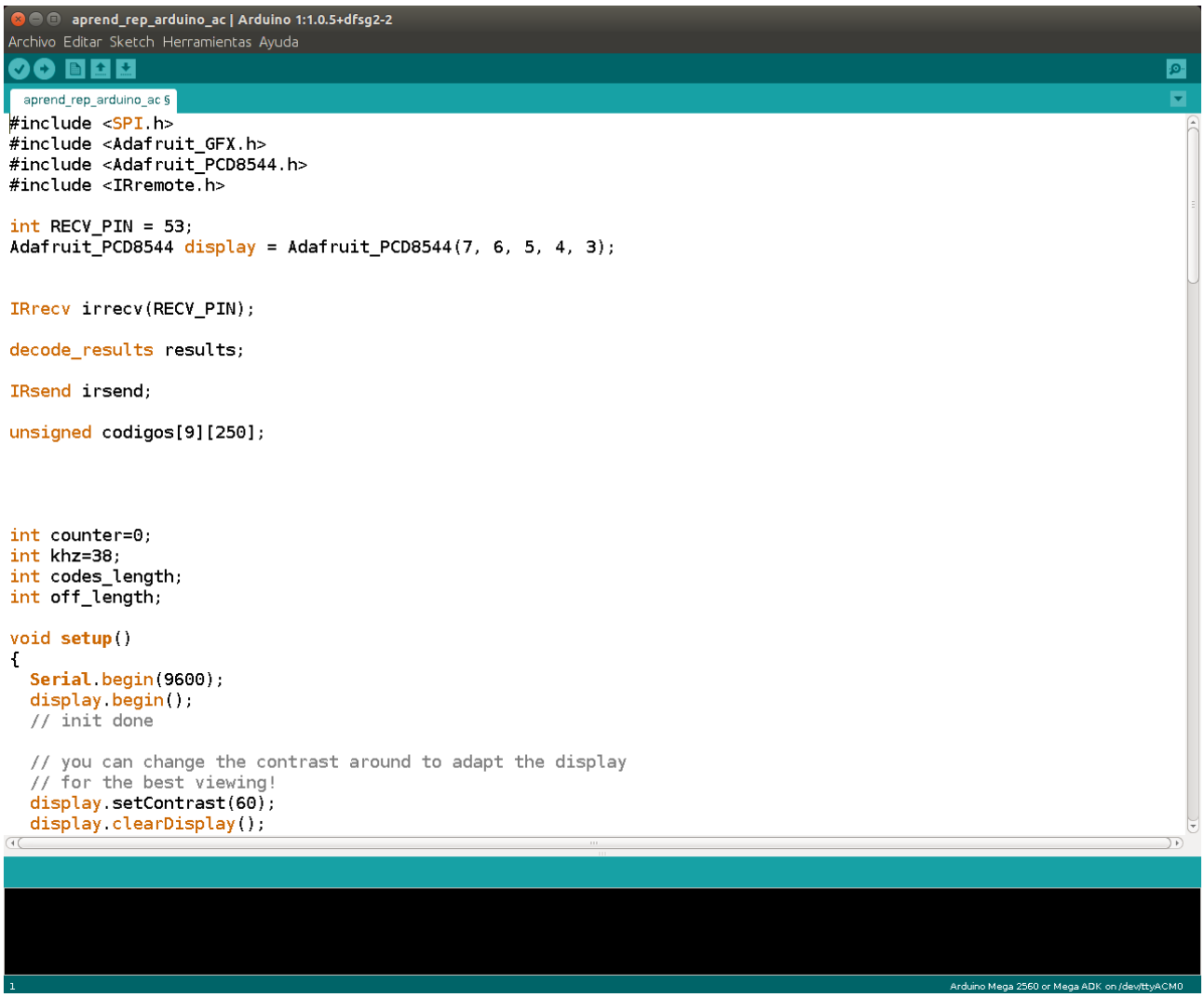
Además de estas funciones, el programador puede crear otras diferentes que podrán ser llamadas desde cualquiera de estas dos para permitir una lectura más clara y la reutilización del código.

4.2.2 Entorno de desarrollo original

El entorno de desarrollo original de Arduino es una herramienta de código abierto basado en el entorno de desarrollo de Processing, que permite escribir el código de programas conocidos como sketch, cargarlos en la placa Arduino y comunicarse con ellos mediante un terminal serie emulado conectado al puerto serie de la placa.

Este entorno está compuesto por diferentes herramientas: un editor de texto para escribir el código, una consola que muestra información del programa que está en ejecución en la placa y permite enviarle información, una barra de herramientas con botones para las funciones comunes y una barra de menú superior que incluye todas las funciones disponibles en el entorno de desarrollo.

Este entorno integra de una forma adecuada todas las herramientas necesarias para que un programador pueda escribir, cargar, ejecutar y comunicarse con el programa que se ejecuta sobre cualquier placa Arduino en muy poco tiempo.



Entorno de desarrollo Arduino IDE.

4.2.3 Entorno de desarrollo alternativo. Microsoft Visual Studio.

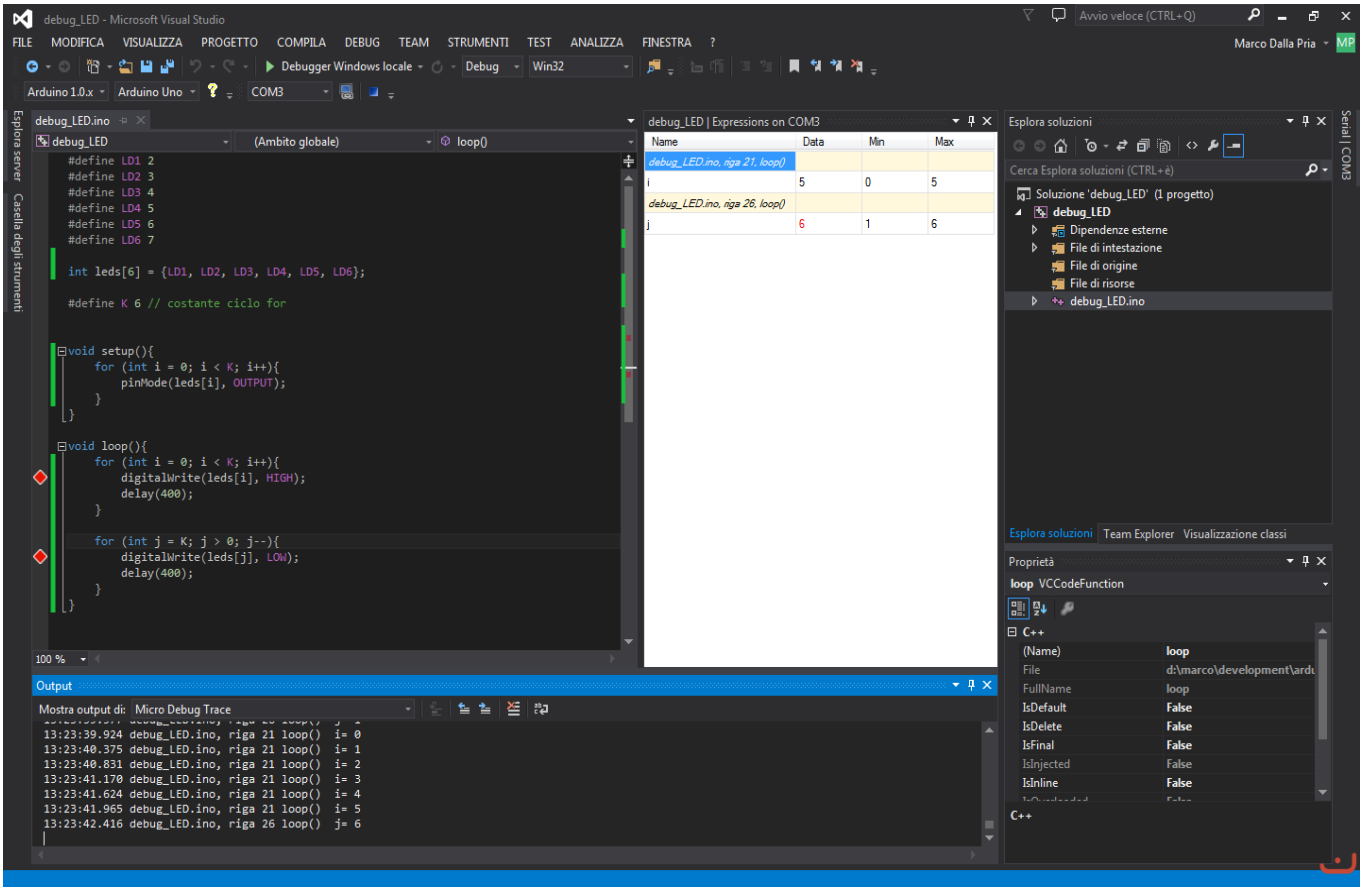
El entorno de Arduino nos aporta gran simplicidad, lo que para usuarios con poca experiencia es ideal. Sin embargo, esta simplicidad se puede convertir en una limitación cuando el usuario adquiere cierta destreza y los programas a desarrollar se vuelven más complejos, necesitando un proceso de depuración avanzado.

En el caso de máquinas con sistema operativo Windows, existe la posibilidad de utilizar Visual Studio como entorno de desarrollo. Para ello se requiere la instalación de un plugin conocido como Visual Micro.

Este plugin configura automáticamente el entorno Visual Studio con el objetivo de desarrollar programas para la plataforma Arduino. Añade soporte para todas las versiones de Arduino y también otras placas compatibles, como Intel Galileo, Teensy, Attiny, avrIO, LaunchPad, ChipKIT/Pic32, STM32 y Texas Instruments StellarPad.

Otra característica muy útil que nos ofrece es el sistema de autocompletado IntelliSense, que supone una de las principales ventajas comparado al entorno de desarrollo original de Arduino. Este sistema ofrece ayudas para una edición de código más fácil, lo que se traduce en menor tiempo para la codificación.

Además nos ofrece un eficiente monitor serie para la comunicación con la aplicación, lo que nos permite una depuración mucho más avanzada que el entorno original incluyendo puntos de parada en la ejecución y estado de variables.



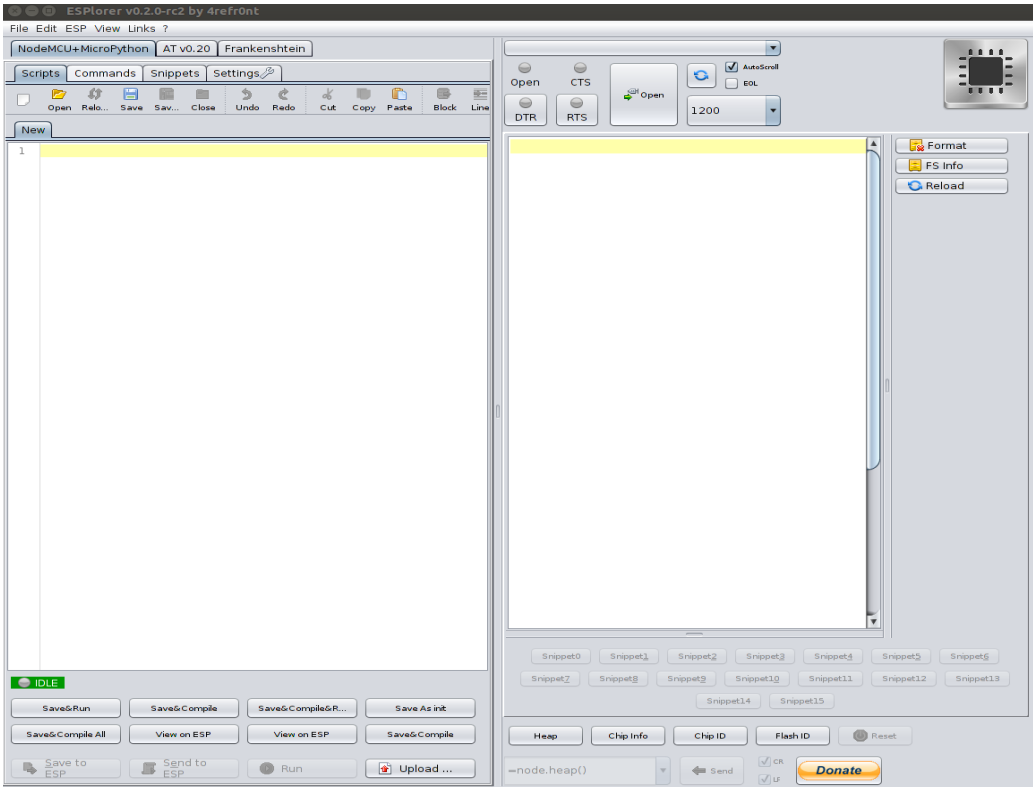
Entorno de desarrollo Visual Studio con el plugin Visual Micro.

4.2.4 ESPlorer. Herramienta de desarrollo y depuración para el chip ESP8266.

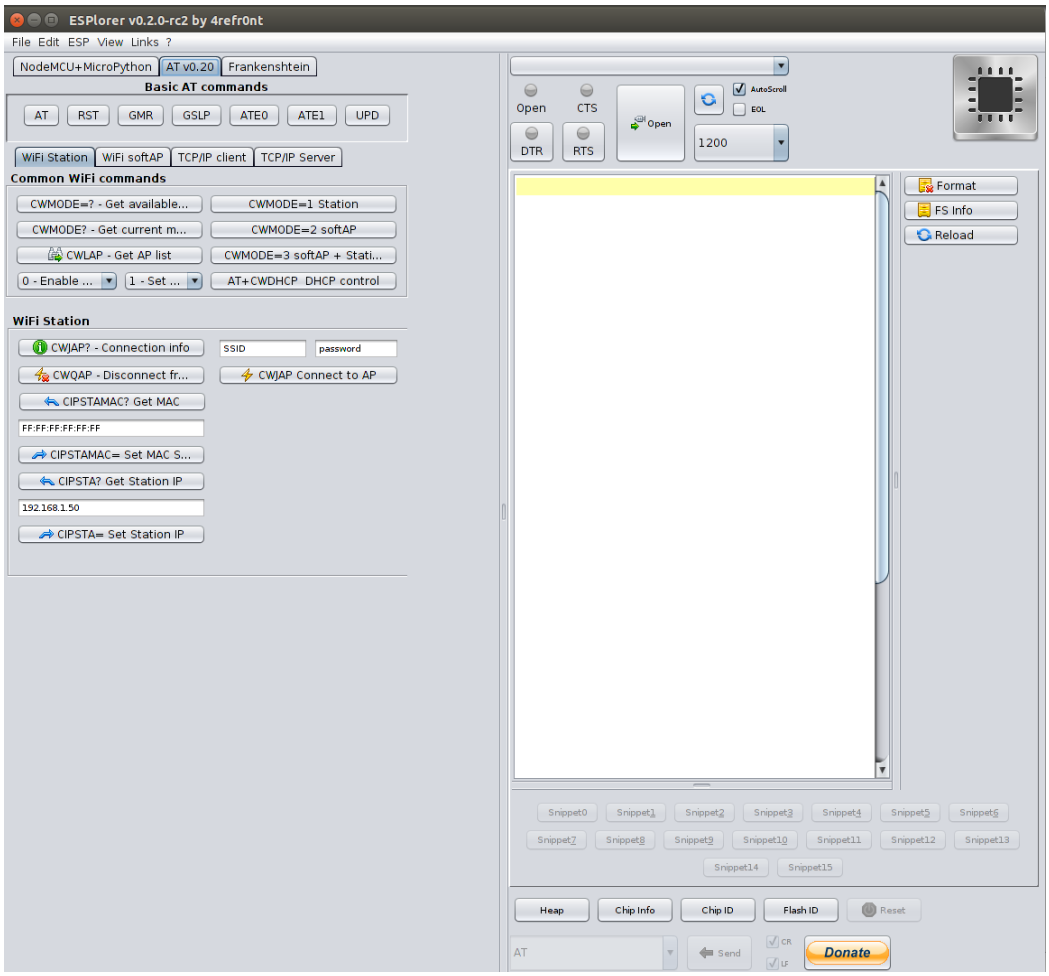
Esta herramienta, desarrollada en JAVA, permite la comunicación entre el PC y el chip. Además de esto incluye un editor de texto y un sistema de reconocimiento del firmware del módulo con el que se está comunicando. Con este sistema, es capaz de ofrecer diferentes utilidades dependiendo del firmware detectado.

Esta herramienta resulta imprescindible para trabajar con el software NodeMCU, puesto que es necesario codificar los scripts en LUA y probarlos. Una vez el script se ha probado y su funcionamiento es el correcto, se debe subir el fichero que lo contiene a la memoria del módulo con el nombre “init.lua”, ya que de esta forma el firmware es capaz de buscar en el sistema de ficheros el archivo con este nombre y ejecutarlo de forma automática. Es muy importante realizar bien la depuración del script, dado que si el código incluido en el fichero “init.lua” no funciona correctamente, el módulo entrará en un bucle de reinicios y será necesario volver a flashear el firmware para poder sustituir el código.

Sin embargo, no solo es interesante utilizar esta herramienta con el firmware NodeMCU, también resulta útil con el firmware AT que viene incluido por defecto en la placa debido al control avanzado de las comunicaciones entre el módulo y la aplicación.



Vista de opciones para el firmware NodeMCU



Vista de las opciones para el firmware AT.

4.3 Herramientas de programación del servidor.

Para que el sistema IoT realice su funcionamiento de forma correcta, necesita de la implementación y uso de diversos servicios en una máquina que centralicen, registren y distribuyan la información o que la reciban desde el exterior para poder procesarla de forma interna posteriormente.

Esto requiere ciertas herramientas y frameworks que nos permitan implementar, probar y desplegar estos servicios.

4.3.1 Servicio MQTT, comunicación IoT

Como ya hemos dicho, en este proyecto vamos a utilizar el protocolo de comunicación MQTT para realizar la recolección y distribución de los datos del sistema entre los diferentes elementos que lo integran. Este servicio será, por tanto, el encargado de recibir los datos que proporcionan los emisores de información (sensores, clientes desde el front-end) y comunicarlos a los clientes que anteriormente los hayan requerido (realizando suscripción a esta información) todo ello utilizando este protocolo.

Para poder realizar toda esta comunicación, obviamente, necesitamos un servicio capaz de gestionar todos estos datos y conexiones. Existen infinidad de herramientas disponibles que ofrecen servicios MQTT para la gestión de datos en este tipo de sistemas, pero en este caso nos hemos centrado en una solución de software libre con gran aceptación debido a su simplicidad y su eficiencia. Esta herramienta llamada Mosquitto ofrece un servicio ligero de mensajería implementando los protocolos MQTT 3.1 y 3.1.1 y siguiendo un esquema publicación/suscripción como ya hemos comentado anteriormente.

Esta herramienta está disponible en múltiples plataformas, lo que nos aporta flexibilidad a la hora de elegir nuestro entorno de trabajo. En este caso, se ha utilizado bajo la plataforma Ubuntu/Linux y su uso ha sido bastante sencillo.

Para arrancar el servicio, únicamente debemos instalar en el sistema el broker. Una vez hecho esto, se arranca el servicio y automáticamente éste abre el puerto 1883 para escuchar las peticiones de suscripción o el envío de datos. Es posible configurar el servicio para que arranque al iniciar la máquina, así como configurar el puerto de escucha. En sistemas Linux, este arranque con el sistema se establece por defecto.

También es posible arrancar varios servicios en la misma máquina de forma que se puede utilizar un mismo servidor para gestionar varios sistemas IoT con servicios separados. Desde terminal existe la posibilidad de pasar ciertos parámetros que configuran la ejecución del servicio. Estos parámetros varían desde el puerto de escucha hasta la posibilidad de seleccionar los certificados para arrancar el servicio, utilizando comunicaciones seguras mediante SSL.

Utilizando esta herramienta, por tanto, conseguimos un núcleo para las comunicaciones de nuestro sistema IoT.

4.3.2 Bottle para Python

Una vez se ha conseguido que los distintos elementos se puedan comunicar, es hora de pasar a implementar los servicios que le aportan algo más de funcionalidad al sistema. Para ello se han creado dos servicios en Python, cuya funcionalidad ya se ha comentado.

Sin embargo, estos servicios deben soportarse sobre una herramienta llamada Bottle para poder ejecutarse como un servicio web y gestionar las peticiones REST que reciban, ya que de otro modo no sería posible utilizar esta solución.

Bottle es un rápido, sencillo y ligero WSGI micro framework para Python. Cabe aclarar que WSGI es una interfaz simple y universal entre los servidores web y las aplicaciones web o frameworks. Está distribuido como un módulo unitario y no tiene dependencias que no sean la biblioteca estándar de Python. Esto nos permite diseñar aplicaciones en Python para ofrecer servicios utilizando tecnologías y estructuras de comunicación propias de la web de forma muy intuitiva y transparente.

En el siguiente ejemplo se muestra la simplicidad con la que es posible crear un servicio web utilizando esta herramienta.

```
from bottle import route, run, template

@route('/hello/<name>')
def index(name):
   return template('<b>Hello {{name}}</b>!', name=name)

run(host='localhost', port=8080)
```

El framework de Bottle permite definir una ruta en formato URI y asociarla a una función, de modo que cuando algún cliente se conecte a esa ruta, el servicio ejecutara la función asociada a ella. También nos permite configurar la interfaz y el puerto a los que se asociará el servicio al iniciar su ejecución.

De este modo, la programación y puesta en marcha de los servicios necesarios para el funcionamiento de este proyecto es mucho más sencilla y el tiempo necesario para el desarrollo es mucho menor.

4.3.3 Gestor de bases de datos MySQL

La última herramienta utilizada se hace necesaria para la implementación del servidor de registro de eventos. El servicio se conecta con la base de datos para guardar todos los eventos que recibe. De esta forma es capaz de mantener una relación de los eventos y guardarlos en el almacenamiento persistente de la máquina. Para ello se ha utilizado el sistema de gestión de bases de datos MySQL.

MySQL es una herramienta de gestión de bases de datos relacional, multiusuario, multihilo y de código abierto, basado en lenguaje de consulta estructurado (SQL). Tiene una gran flexibilidad, ya que se ejecuta en prácticamente todas la plataformas. Su uso está sujeto a dos licencias diferentes, la primera de ellas es una Licencia Publica General de GNU (GPL) para el uso de desarrolladores y la segunda es una licencia comercial de Oracle para su uso en empresas.

Su instalación es bastante sencilla, pero también está incluida como componente clave en LAMP (pila de software empresarial de código abierto destinado a desarrollo web) que nos permite instalar un paquete de software para servidores (incluyendo servidor Apache, servicios SSL para seguridad, etc.) de forma muy sencilla.

Para su configuración se utiliza su cliente en línea de comandos, aunque también existen entornos visuales que facilitan la creación de las bases de datos y sus tablas. Para ofrecer un funcionamiento eficiente se debe crear un usuario y otorgarle los permisos necesarios dependiendo de las necesidades de la aplicación. En este proyecto se ha creado un usuario Syslog con permisos de lectura y escritura, permitiendo registrar y consultar eventos, pero no eliminarlos. De este modo nos aseguramos que desde el exterior del sistema no sea posible eliminar entradas de registro que podrían aportar información significativa en caso de accesos no autorizados.

Una vez configurado el usuario, creamos la base de datos y la tabla con un script en Python. Esto se hace una única vez, pudiendo a partir de este momento registrar los eventos en la tabla recién creada desde el servicio de registro de eventos y consultar el listado de todos los eventos ocurridos.

5. Tecnologías empleadas

Con el fin de llevar a cabo este proyecto, es necesario apoyarse en ciertas tecnologías existentes en el mercado. Aunque la elección de estas tecnologías y las herramientas a utilizar ya ha sido tomada, es interesante analizarlas y compararlas con las posibilidades de que disponemos hoy en día.

5.1 Protocolos para IoT disponibles

Como ya se ha comentado en este proyecto, actualmente el campo del Internet de las Cosas está experimentando una evolución vertiginosa. Por este motivo existen infinidad de implementaciones de protocolos para esta tecnología, lo que exige un análisis profundo para determinar qué solución es más conveniente en cada caso. Existen soluciones tanto abiertas (desarrolladas por comunidades de usuarios utilizando licencias de código libre) como privadas (desarrolladas por empresas para el uso en sus sistemas). Es posible utilizar algunas de estas soluciones en proyectos de este tipo, sin embargo se debe analizar la compatibilidad con los elementos a conectar y la carga que suponen las comunicaciones en la red. De este modo será posible elegir el protocolo que mejor se adapte a cada situación en particular.

En el presente proyecto, se deben analizar aspectos de los protocolos como la compatibilidad con el hardware disponible (tanto con las placas Arduino como con los módulos WiFi) y la posibilidad de analizar el código para la resolución de problemas.

5.1.1 MQTT

Por este motivo, se ha decidido utilizar MQTT, un protocolo de conectividad máquina-a-máquina (M2M)/Internet de las Cosas. Utiliza una arquitectura de comunicación publicación/suscripción orientada a mensajes, pero con una baja simplicidad en el protocolo lo que aporta ligereza en los dispositivos y un correcto funcionamiento en redes con ancho de banda pequeño, alta latencia o problemas de fiabilidad y entrega.

Dicho protocolo funciona sobre redes TCP/IP, lo que permite su uso sobre la práctica totalidad de las redes de comunicaciones que operan hoy en día (incluyendo Internet). MQTT tiene

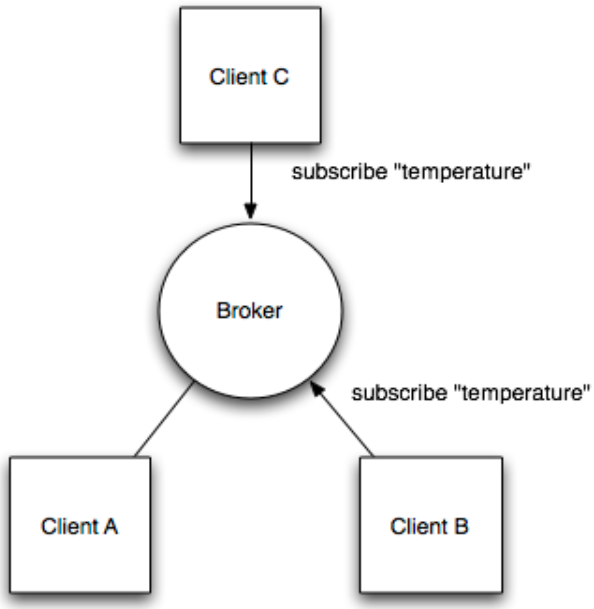
reservado por la IANA el puerto 1883 sobre TCP y el puerto 8883 también está reservado para este protocolo sobre SSL.

Adicionalmente a la seguridad que proporcionan las conexiones SSL, el protocolo MQTT permite desde su versión 3.1 el uso de nombre de usuario y contraseña. Es posible además incrementar la seguridad utilizando sobre lo anterior aplicaciones de encriptación en los extremos de la comunicación. Sin embargo, no es recomendable introducir más capas de seguridad de las necesarias por las sobrecargas que ello introduce en las comunicaciones, perdiendo así la esencia de ligereza del protocolo.

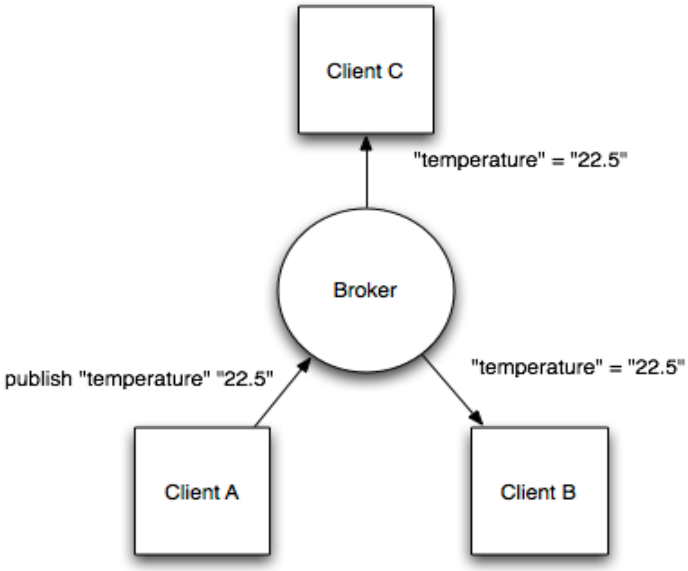
Nos permite distinguir entre tres niveles de calidad de servicio (QoS): “Fire and forget”, “delivered at least once” y “delivered exactly once”. Esto permite ajustar la exigencia de red para las comunicaciones y el comportamiento del broker a la hora de recibir y enviar el mensaje a los suscriptores.

Aunque fue inventado por desarrolladores de empresas privadas (IBM y Arcom), se encuentra ahora mismo bajo una licencia de software libre y desde 2013 está bajo revisión en OASIS para convertirse en un estándar de comunicación para sistemas IoT.

En la primera fase el cliente B y el cliente C envían una petición de suscripción al *topic* “temperature”:



A partir de este momento, los clientes quedan suscritos a “temperature” y el broker hará llegar cualquier dato que se publique en este *topic* a los clientes suscritos, como podemos ver en la siguiente imagen:



Con estos diagramas es posible observar la simplicidad que ofrece el protocolo a la hora de enviar una información a múltiples dispositivos que previamente hayan solicitado la información, con lo que se reduce al mínimo el uso de la red sin introducir retardos en el envío de la información.

5.1.2 CoAP

CoAP es el protocolo de aplicación restrigida del CoRE (Entornos de recursos restringidos) del grupo IETF.

Al igual que HTTP, CoAP es un protocolo de transferencia de documentos. Sin embargo, CoAP ha sido desarrollado para dispositivos con recursos limitados. El tamaño de paquetes del protocolo CoAP es mucho menor que el de HTTP, de forma que permite ahorrar memoria y exige menor capacidad de cómputo.

CoAP funciona sobre UDP, pero no sobre TCP. Sigue una arquitectura cliente/servidor en la que los elementos del sistema se comunican mediante datagramas UDP sin establecimiento de flujo de datos. Los clientes se comunican con el servidor enviando peticiones GET, POST, PUT y DELETE al estilo de REST y el servidor responde a estas peticiones. De este modo, existe una gran facilidad para que CoAP pueda operar con HTTP y RESTful web utilizando unos simples proxies.

CoAP no ofrece seguridad en el protocolo mediante SSL/TSL ya que utiliza UDP para las comunicaciones, sin embargo es posible utilizar DTLS (Datagram Transport Layer Security) de forma equivalente siempre que los clientes lo soporten.

Incluye en el mensaje un campo de calidad de servicio (QoS), que permite marcarlo como “confirmable” o “nonconfirmable”. En el primer caso, el emisor del mensaje requiere un paquete de confirmación de recepción (ack packet) del lado del receptor. En el segundo caso esto no es requerido, siendo equivalente al nivel “fire and forget” del protocolo MQTT.

5.1.3 Comparación entre ambos

Tanto MQTT como CoAP pueden ser utilizados como protocolos para aplicaciones y sistemas IoT. Sin embargo mantienen algunas diferencias fundamentales.

MQTT sigue una estructura muchos a muchos en su arquitectura permitiendo la emisión de información a muchos clientes simultáneamente utilizando un broker central. Esto le aporta además cierto apoyo a la persistencia de los mensajes, lo que lo convierte en la mejor solución para sistemas con información en tiempo real.

CoAP es, a su vez, un protocolo de uno a uno para la transferencia de información de estado entre el cliente y el servidor. A pesar de que cuenta sistemas para la observación de los recursos, COAP es el más adecuado para un modelo de transferencia de estado.

MQTT suele comportarse de forma correcta en redes con dispositivos NAT ya que establece conexiones TCP. Sin embargo, CoAP puede presentar problemas a la hora de realizar comunicaciones por el uso de UDP en el protocolo, lo que requiere el uso de túneles o redirección de puertos para su correcto funcionamiento.

Los motivos expuestos aquí han sido los que han llevado a la elección del protocolo MQTT como el más adecuado para la realización del proyecto.

5.2 Tecnologías adicionales

Con el fin de aportar valor al proyecto y ampliar las capacidades que nos puede ofrecer nuestro sistema IoT implementando los servicios secundarios de nuestra infraestructura, es necesario apoyarse en tecnologías y protocolos de red.

Estas tecnologías nos permitirán añadir la funcionalidad del servidor de registro de eventos y ofrecer un front-end para la comunicación del usuario de forma remota con el sistema. También nos permitirán interactuar con algunos equipos desde los elementos internos, como es el caso del equipo de aire acondicionado que utiliza la tecnología infrarroja para su control.

5.2.1 REST

Para implementar los servicios que nos proporcionarán funcionalidades extra al sistema necesitamos una tecnología que nos permita ofrecer una interfaz web a la que el usuario pueda conectarse para enviar órdenes al sistema. Para ello hemos elegido REST.

REST (REpresentational State Transfer) es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP. Nos permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP, por lo que es increíblemente más simple y convencional que otras alternativas que se han usado en los últimos diez años como SOAP y XML-RPC. REST se definió en el 2000 por Roy Fielding, coautor principal también de la especificación HTTP. En ciertos aspectos se podría considerar REST como un framework para construir aplicaciones web respetando HTTP.

Por lo tanto, es el tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a Internet.

Existen tres niveles de calidad a la hora de aplicar REST en el desarrollo de una aplicación web y más concretamente una API que se recogen en un modelo llamado Richardson Maturity Model en honor al desarrollador que lo estableció Leonard Richardson, padre de la arquitectura orientada a recursos. Estos niveles son:

1. Uso correcto de URIs
2. Uso correcto de HTTP.
3. Implementar Hypermedia.

Además de estas tres reglas, nunca se debe guardar estado en el servidor, toda la información que se requiere para mostrar la información que se solicita debe estar en la consulta por parte del cliente, ya sea en la misma ruta del recurso o bien añadiendo parámetros en el formato que el servidor sea capaz de interpretar (XML, JSON, HTML, texto plano, etc).

Al no guardar estado, REST nos aporta muchas ventajas, ya que podemos escalar mejor sin tener que preocuparnos de aspectos como el almacenamiento de variables de sesión e incluso, podemos utilizar distintas tecnologías para servir determinadas partes o recursos de una misma API.

5.2.2 Python

Python no es una tecnología en sí misma, es un lenguaje de programación. Sin embargo, al ser un lenguaje interpretado que aporta infinidad de librerías y herramientas que dan soporte a la mayoría de tecnologías de red que se usan en la actualidad, es interesante incluirlo en este apartado. De este modo, podemos podemos implementar y apoyarnos en diferentes tipos de tecnologías desde un mismo entorno de una forma muy sencilla, pudiendo aprovechar en cada caso la que mejor se adapte a nuestras necesidades o incluso pudiendo utilizar más de una desde el mismo código para poder soportar replicación de recursos.

En este proyecto, se ha utiliza Python para implementar los diferentes servicios adicionales que se han integrado en el sistema de Internet de las Cosas que nos ocupa: servidor de registros de eventos y front-end al sistema para la interacción del usuario con éste. De esta forma, ayudándonos de la flexibilidad de Python podemos crear servicios que utilicen diferentes tecnologías y que tengas finalidades distintas con la capacidad de comunicarse entre ellos.

5.2.3 Protocolos IR para equipos de aire acondicionado

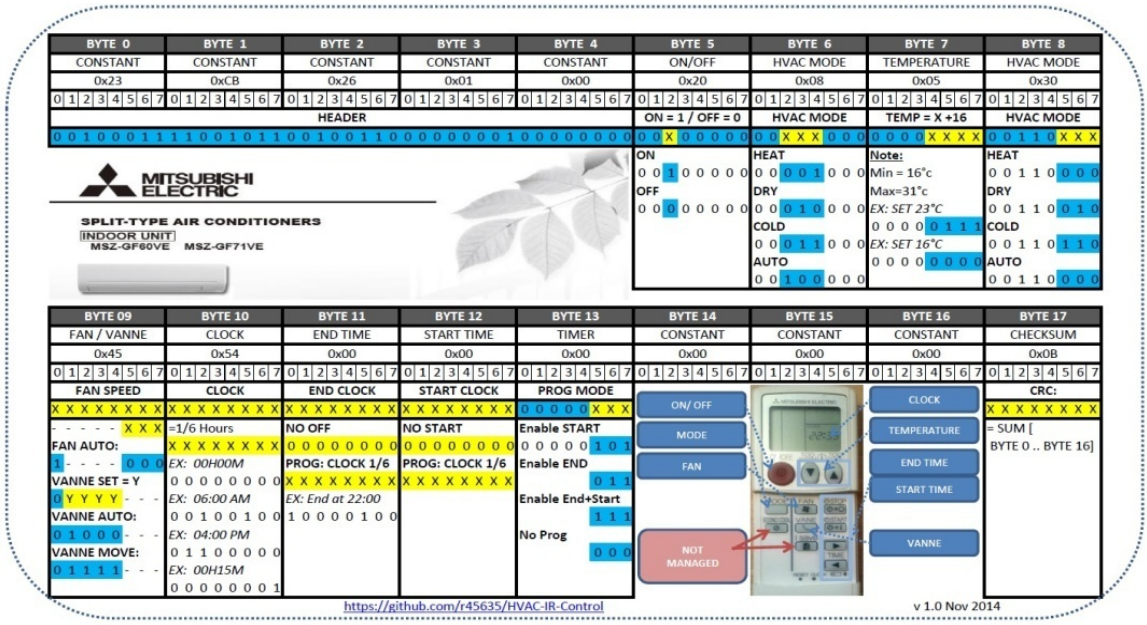
Como parte importante de cualquier sistema domótico, es indispensable poder controlar elementos del hogar de forma automática o con la menor interacción posible por parte del ser humano. En este caso se ha optado por controlar un sistema de aire acondicionado. Para el control del aparato se ha usado un led emisor de infrarrojos, lo que nos permite reducir al máximo el coste ya que no tendríamos que modificar el sistema de aire instalado en la mayoría de hogares y en los que no lo esté, nos permite instalar cualquier modelo de los que se comercializa en la actualidad. En el caso de que no hubiera ningún sistema de aire acondicionado se podría valorar la idea de adquirir uno con características específicas para IoT, sin embargo, no nos aporta ningún beneficio si lo comparamos con el coste que supone.

Existen dos tipos de soluciones para poder abordar el control mediante IR de cualquier sistema que funcione utilizando este tipo de comunicación y asegurarnos de no afectar al sistema de forma que, si en algún momento se desea trasladar el equipo a otro lugar o retirar el sistema domótico, el aparato seguiría funcionando del mismo modo que cuando se adquirió. Esto también resulta interesante para garantizar un correcto funcionamiento del propio aparato, ya que al no ser manipulado la garantía del fabricante no se pierde en ningún momento.

La primera de ellas es el envío de la orden deseada conociendo el protocolo de comunicación para esa marca y ese aparato en concreto. Esta solución es la que nos da un mayor control del aparato, pero es muy compleja ya que requiere la decodificación del protocolo si éste no es libre y la posterior codificación de cada orden.

La segunda de ellas, que es la que se empleará en este proyecto, consiste en la replicación de los códigos que emite el control remoto original del sistema a controlar. Esta solución aporta facilidad y garantía de funcionamiento que no se consigue con el método anterior. Esto es debido a que el receptor del equipo a controlar recibe el mismo código que emite su control remoto y si se realiza bien la copia del código original, el riesgo de fallo en el protocolo al enviar la orden es nulo.

Para poder controlar de forma correcta el sistema de aire, necesitamos almacenar los códigos completos que emite el control remoto original del aparato, sin poder guardar una codificación de éste para ahorrar espacio. Esto es debido a que los aparatos a/c reciben siempre toda la información del equipo: estado (encendido/apagado), posición de los aireadores, velocidad del ventilador, temperatura, etc. Como consecuencia de esto, los códigos que envía el mando de un aparato de estas características son complejos y muy dependientes del fabricante. Para ello las librerías IR de Arduino nos ofrecen una función que permite enviar un código IR en formato RAW (crudo). Con esta alternativa el envío y recepción se hacen más complejos, pero seremos capaces de reproducir cualquier tipo de codificación que requiera el sistema a controlar sin importar la complejidad o tamaño de los códigos.



Protocolo IR para equipo de aire acondicionado Mitsubishi

6. Implementación

En el siguiente apartado se realizará la descripción de los pasos seguidos a la hora de implementar el código que permitirá a los distintos elementos del sistema funcionar de la manera que se ha especificado. Para ello, este apartado se divide en tres subapartados en los que se observarán los detalles de implementación de los servicios de servidor, los clientes Arduino y NodeMCU y por último, el arranque de todos estos servicios y clientes de modo que la puesta en marcha del sistema sea satisfactoria.

6.1 Implementación del servidor

En este apartado se tratará en profundidad la implementación de los servicios que nos van a permitir introducir valor añadido al sistema IoT, por lo que aun no estando incluidos dentro de las comunicaciones IoT, no dejan de ser piezas fundamentales de la estructura del proyecto.

Se observarán los servicios de servidor de registro de eventos y el front-end de acceso a la infraestructura. Recordemos que estos servicios han sido implementados en Python, por lo que se lanzan como cualquier otro programa codificado en este lenguaje.

6.1.1 Servidor de registro de eventos

Vamos a entrar en detalle en el desarrollo e implementación del servidor de registro de eventos. Aquí veremos el código en Python que permite recibir los eventos mediante una API implementada con REST y almacenar esos registros de eventos en una base de datos MySQL a la que se accede desde este mismo código.

Para ello, debemos etiquetar cada una de las funciones con la ruta que queramos asignar mediante la herramienta Bottle (de la que ya hemos hablado), que nos permite asignar funciones en Python a rutas en formato URI propias de REST. De este modo será sencillo ejecutar la función correcta cuando se reciba una petición para el servicio en la ruta correcta.

Las librerías necesarias para la ejecución del servicio son las siguientes:

- MySQLdb
- JSON
- HTML

Además de esto, no debemos olvidar la importación del paquete Bottle para poder ejecutar la aplicación como servicio, en el que se han implementado dos funciones, *put_event()* y *get_events()*, de las que hablaremos con detalle a continuación.

La primera de ellas, *put_event()*, se utiliza para registrar los nuevos eventos que se reciben en la base de datos para poder consultarlos cuando sea necesario. Se encuentra disponible en la ruta */newEvent* del servidor utilizando una petición del tipo POST, lo que es simple de deducir si observamos la cabecera de la función:

```
@route('/newEvent',method='POST')

def put_event():
```

Como ya hemos visto, esta es la forma en la que el framework Bottle marca las funciones y las asigna a rutas en formato URI para que puedan ser accesibles de forma muy simple mediante peticiones HTTP.

Cuando el servicio recibe una petición a esta dirección y con este tipo, se ejecuta la función asociada que tiene como primer objetivo comprobar que los parámetros de la petición son los correctos. Estos parámetros correctos deben incluir como mínimo el identificador del dispositivo o usuario que genera el evento y la prioridad del evento en el sistema. De este modo nos aseguramos de que cualquier evento tiene asignado una entidad que lo genera y un nivel de importancia para el sistema. Una vez se ha comprobado los parámetros, utilizando para ello la librería de codificación y decodificación de datos en formato JSON, se genera con estos una sentencia SQL que posteriormente se ejecutará contra el servidor MySQL elegido.

Para poder introducir datos en la base de datos, se ha generado un código sencillo en Python que genera las tablas con el usuario adecuado para inicializar todos los componentes. Esto solo debe

hacerse la primera vez que se ejecuta el servicio cuando las tablas no están creadas, no es necesario entrar en detalle ya que no es el objetivo de este proyecto.

Hecho esto se lanza la sentencia contra el servidor para almacenar el evento en la tabla correspondiente, añadiendo un índice interno de la tabla a cada evento y el instante de tiempo en el que se ha registrado el evento en la base de datos. Si la sentencia finaliza con un resultado correcto, se se guardan los cambios en la base de datos del servidor. En el caso de que ocurra algún error, se muestra un mensaje con el error que se haya producido al conectar con la base de datos. Después de esto, se cierra la conexión con el servidor.

El código completo de esta función es el siguiente:

```
@route('/newEvent',method='POST')
def put_event():
	data=request.body.readline()
	print data
	if not data:
		abort(400,'No data recived')
	entity = json.loads(data)

	if not entity.has_key('_dev') or not entity.has_key('_prio'):
		abort(400,'No device or priority specified')

	bd = MySQLdb.connect("localhost","serverlog","serverlogvirtualbox","serverlog")
	cursor = bd.cursor()
	sql="INSERT INTO LOGS (DISPOSITIVO, PRIORIDAD, VALOR) VALUES ('"+entity["_dev"]+"','"+str(entity["_prio"])+",'"+entity["_value"]+"')"

	try:
		cursor.execute(sql)
		bd.commit()
	except:
		bd.rollback()
```

```
        bd.close()
```

La segunda función, *get_events()*, se usa para listar todos los eventos registrados en el servidor hasta ese momento, de forma que es posible encontrar cualquier comportamiento anómalo. Se encuentra disponible en la ruta */listEvents* del servidor utilizando una petición del tipo GET, lo que es simple de deducir si observamos la cabecera de la función:

```
@route('/listEvents', method = 'GET')
def get_events():
```

Esta función no recibe ningún parámetro, ya que debe obtener todo el contenido de la tabla para generar la respuesta. El programa realiza una búsqueda de todas las entradas de la tabla de la base de datos con una sentencia generada para este fin, que siempre es la misma. Una vez hecho esto, y utilizando la librería HTML para Python, se genera en el programa una respuesta en HTML que permite formatear los resultados obtenidos del servidor de base de datos en tablas para facilitar la visualización de los datos. De este modo cada fila es un evento registrado en el servidor, y cada columna es un atributo del evento, como son: instante en que ocurrió, dispositivo o usuario que lo generó, prioridad en el sistema y valor del evento.

Cabe destacar que para implementar estas funciones, es indispensable el uso de librerías que permitan la comunicación entre Python y el servidor de bases de datos, en este caso MySQL. Para realizar la comunicación en este caso se ha utilizado la librería MySQLdb. Esta librería permite abrir conexiones a la base de datos, ejecutar sentencias SQL contra ellas y generar, en caso de ser necesario, listas para almacenar las respuestas a las consultas.

El código completo de esta función es el siguiente:

```
@route('/listEvents', method = 'GET')
def get_events():
        bd = MySQLdb.connect("localhost","serverlog","serverlogvirtualbox","serverlog")
        cursor = bd.cursor()
        sql= "SELECT INSTANTE, DISPOSITIVO, PRIORIDAD, VALOR FROM LOGS
ORDER BY INSTANTE"
        resp="<!DOCTYPE html><html><h1>List of events</h1>"
        try:
```

```
            cursor.execute(sql)
            resultados = cursor.fetchall()
        except:
            bd.rollback()
            abort(404, "No events")
        resp+=HTML.table(resultados,header_row=['Instante','Dispositivo','Prioridad','Valor'])
        resp+="</html>"
        return resp
```

Con todo esto, ya tenemos un servicio capaz de almacenar todos los registros ocurridos en nuestro sistema, información que se podrá consultar accediendo a éste desde el siguiente servicio.

6.1.2 Front-end de acceso al sistema

A continuación nos centraremos en el desarrollo y la implementación del front-end del sistema. Este servicio nos permitirá acceder al sistema para gestionar los diferentes equipos que se pueden controlar, así como también observar el listado de los eventos que han sido registrados hasta ese momento.

Para poder llevar a cabo estas acciones, el servicio ofrece una API de tipo REST del mismo modo que el servicio anterior. De este modo, el usuario se conecta a una URI (que internamente estará asociada a una función del código del servicio) y permitirá al usuario realizar la acción deseada. Para aportar cierto nivel de seguridad, el front-end recibe como parámetro en el cuerpo de la petición REST un nombre de usuario y una contraseña. Si el nombre de usuario y la contraseña no coinciden con los esperados, no se permite realizar ninguna acción dentro del sistema y se registrará un evento de intento de acceso no autorizado en el servidor de registro de eventos. Este control de acceso es muy simple y está implementado dada su importancia en este tipo de sistemas, pero queda fuera del objetivo principal del proyecto y por tanto no se ha profundizado en él.

Para ello, como en el caso anterior, debemos asignar a cada función una ruta con la herramienta Bottle.

En este servicio se han implementado tres funciones, de las que se hablará en profundidad a continuación.

La primera de ellas, *ac_event()*, se usa para establecer el modo de control del equipo de aire acondicionado, pudiendo elegir entre manual (fijando a continuación una temperatura por el usuario) o automático (en la que la propia placa Arduino fija la temperatura dependiendo de la temperatura actual recibida desde el sensor y unos umbrales prefijados en su firmware). Esta función es accesible desde la ruta */ac_mode* y acepta peticiones del tipo POST. Con esto, es posible para un usuario controlar totalmente el aparato un aire acondicionado de forma remota o sin necesidad de su interacción.

Como en el servicio anterior, la función comienza obteniendo los parámetros del cuerpo de la petición para poder determinar la acción a aplicar. Si falta alguno de los parámetros necesarios, se devuelve un mensaje de error y se aborta la ejecución de la petición. Para esta función, los parámetros necesarios son: modo de funcionamiento, usuario y contraseña. La comprobación se realiza, como en el servicio anterior, utilizando la librería JSON.

Una vez se ha analizado el cuerpo de la petición y se comprueba que existen los parámetros, se deben comprobar uno a uno para asegurar que su contenido es interpretable por el sistema (en el caso del modo de funcionamiento) o correctos (en el caso del usuario y la contraseña).

En este punto, el programa ha recibido los datos correctos y está en disposición de enviar las órdenes pertinentes. Lo primero que se hace es enviar el evento recibido al servidor de registro de eventos. Para ello se utiliza la librería JSON, que codifica los parámetros recibidos en el cuerpo de la nueva petición y se definen las cabeceras de la petición (en este caso la cabecera es: 'Content-type':'application/json'). Hecho esto, se genera una petición al servidor de registros utilizando su URL, la cabecera y el cuerpo del mensaje generados anteriormente y se lanza la petición utilizando la librería urllib2, una librería que permite la conexión a URL disponible para Python.

Por último, es necesario hacer llegar la información del cliente a la parte del sistema encargada de dar las órdenes a los equipos físicos controlados. Para ello, se ha utilizado la librería Paho MQTT Client. Esta librería ofrece comunicación con servicios MQTT a los programas desarrollados en Python. De esta forma, el front-end es capaz de publicar la nueva información en los *topics* correspondientes (en este caso el *topic /acmode*) de forma que cuando el broker MQTT se la haga llegar al cliente Arduino, este podrá tratarla y aplicar las órdenes correctas al sistema de aire controlado. De esta forma no se interfiere ni se incrementa la complejidad del cliente Arduino, ya que únicamente se debe suscribir a un *topic* más en la inicialización y que posteriormente se gestionará cuando se reciba alguna información de este.

La segunda de estas funciones es *rele_mode()*. Esta función se usa para encender o apagar el aparato conectado al segundo cliente Arduino, es decir, el que está conectado al relé. Es accesible desde la URI */rele* y acepta peticiones del tipo POST. Su implementación es exactamente igual a la de la función anterior, únicamente debemos tener en cuenta que en este caso el parámetro de modo de funcionamiento debe ser activado o desactivado. Se mantiene por tanto la comprobación de los parámetros en la recepción de la petición del usuario y la contraseña.

Al ser prácticamente igual que la función anterior, la estructura que sigue es la misma. Se comprueban los parámetros del cuerpo de la petición, los cuales viene codificados en JSON. Después se codifica el cuerpo del mensaje y las cabeceras para enviarlas el servidor de registro de eventos y guardar el nuevo evento de cambio de estado del relé. Una vez hecho esto, se envía el evento y se hace la conexión al broker MQTT para publicar el nuevo estado en el tópic correspondiente. De este modo el broker hará llegar el mensaje al cliente Arduino para que pueda actuar sobre el relé como corresponda.

Con estas dos funciones, el usuario será capaz de tener cierto control sobre el sistema, pudiendo aplicar cambios sobre él de forma manual. Sin embargo, es interesante que el usuario tenga la posibilidad de analizar datos del sistema del mismo modo que controlarlo.

Por este motivo se ha implementado la tercera función, *get_events()*, que describiremos a continuación. Con esta función, hacemos accesible las consultas al servidor de registro de eventos desde el exterior del sistema, de un modo muy parecido a como lo haría un proxy. Esto es debido a que el acceso a este servidor de registro únicamente está permitido desde elementos internos del sistema. De este modo, el front-end es quien realiza en última instancia la consulta al servidor y envía la respuesta que ha recibido al usuario que originó la petición. Para ello, la función utiliza la librería urllib2 fijando como URL destino de la petición la ruta de la función *listEvents* del servidor de registro de eventos. Cuando obtiene la respuesta de éste, obtiene los datos y los envía al usuario. De esta forma se enmascara el proceso interno de reenvío de la petición, siendo éste incapaz de percibir que no es posible acceder a este servicio desde el exterior. Para mantener una coherencia entre las URI a las funciones de los dos servicios que tienen el mismo objetivo, esta función es accesible desde la ruta */listEvents* con peticiones de tipo GET.

6.2 Implementación de los clientes

En este apartado se tratará en profundidad la implementación del firmware de los clientes Arduino conectados a los equipos domésticos, tanto el que controla el aparato de aire como el que controla el calefactor. También se tratará el desarrollo e implementación del código que

permite a la placa NodeMCU obtener datos del sensor de temperatura y publicarlos utilizando la interfaz de red WiFi y el protocolo MQTT.

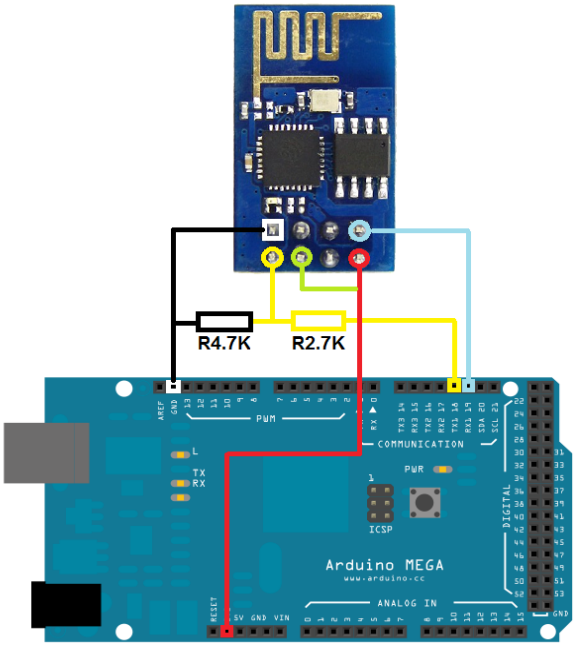
En los clientes Arduino, podremos observar la estructura que debe seguir el firmware para poder conectarse al broker MQTT utilizando el módulo WiFi.

6.2.1 Cliente de control de aire acondicionado

En primer lugar trataremos la implementación del código para el cliente Arduino encargado de controlar el equipo de aire acondicionado. Este firmware realiza diferentes tareas muy diferenciadas que unidas nos aportan la funcionalidad completa que debe tener este cliente. Por este motivo, la implementación de las diferentes tareas se ha realizado de forma separada, uniéndolas a posteriori para obtener el código final.

La primera tarea que debe realizar el cliente Arduino es la comunicación por red. Esta tarea es fundamental en sistemas de Internet de las Cosas, ya que sin esa comunicación con otros elementos esta tecnología no existiría. Por tanto, el primer paso es codificar un firmware muy simple que nos permita comprobar la funcionalidad de red del módulo WiFi y la compatibilidad entre el módulo y la placa, así como la interoperabilidad del software de los componentes.

En este punto se hace imprescindible hablar sobre las librerías a utilizar para que los diferentes componentes puedan comunicarse utilizando el mismo protocolo, en este caso MQTT. Para ello se ha utilizado la librería Espduino disponible en GitHub. Esta librería está diseñada para funcionar de forma conjunta con el firmware ESP_bridge, del que ya se ha hablado anteriormente en el apartado correspondiente. Con este firmware operando en el módulo, la librería Espduino nos ofrece funciones específicas para la conexión a redes WiFi y a servicios MQTT. También dispone de funciones que se invocan al recibir cualquier información de los *topics* a los que estamos suscritos. Todo ello simplifica mucho la programación de las placas Arduino a la hora de integrar esta plataforma en sistemas IoT.



Conexionado del ESP-01 con Arduino

La simplicidad que nos aporta esta librería viene dada por el número de funciones que nos ofrece y que podemos utilizar de una forma muy simple, creando un objeto ESP al que se le debe indicar la interfaz serie al que está conectado el módulo. Una vez hecho esto, se crea un objeto MQTT y se le pasa como parámetro el objeto ESP anterior. Con esto ya tenemos disponibles los objetos sobre los que se aplicarán todas las operaciones de suscripción, publicación y conexión que necesitemos para las comunicaciones de red. La librería incluye ejemplos para la conexión a servicios MQTT, que se hacen muy útiles para comprender como funciona.

Existen cuatro funciones principales que debemos tener en cuenta, ya que posteriormente deberán ser asignadas al objeto MQTT:

- *mqttConnected()*: función que se ejecuta cuando el el sistema establece la conexión con el servidor MQTT. Suele utilizarse para realizar las suscripciones a los *topics* correspondientes. Este cliente en concreto se suscribe a los siguientes *topics*: */temp*, */hum* y */acmode*. En los dos primeros se publica la información relativa a la temperatura y la humedad desde el cliente NodeMCU conectado al sensor DHT11, mientras que desde el último el front-end publica el modo de funcionamiento recibido por el usuario desde el exterior.

- *mqttDisconnected()*: función que se ejecuta cuando el sistema se desconecta del servidor MQTT. No funciona correctamente, pero debe ser declarada para el correcto funcionamiento del sistema.

- *mqttData()*: se ejecuta cuando se recibe información desde algún tópic al que se ha suscrito el cliente. Esta función es la encargada de tratar la información recibida y actuar consecuentemente.

- *mqttPublished()*: se ejecuta cuando el cliente publica alguna información en algún tópic. Su funcionamiento no acaba de ser correcto, pero debe ser declarada.

También existe otra función, esta vez correspondiente al objeto ESP, que debe ser tenida en cuenta ya que es la encargada de establecer la conexión con el servidor MQTT. Esta función, a la que hemos llamado *WiFiCb()*, es la encargada de establecer los parámetros de comunicación con el servidor. Estos parámetros son los siguientes: IP, puerto y seguridad. La seguridad aún no es totalmente funcional en la librería, lo que nos lleva a utilizar la conexión sin seguridad.

Una vez implementadas las funciones, debemos asignarlas al evento del objeto correspondiente para que el sistema sea capaz de invocarlas en el momento que se requiera de forma automática. Para ello se utilizan las siguientes instrucciones:

```
/*setup mqtt events */
 mqtt.connectedCb.attach(&mqttConnected);
 mqtt.disconnectedCb.attach(&mqttDisconnected);
 mqtt.publishedCb.attach(&mqttPublished);
 mqtt.dataCb.attach(&mqttData);

 /*setup WiFi*/
 esp.WiFiCb.attach(&WiFiCb);
```

Después de asignadas, inicializamos los objetos con las funciones correspondientes:

```
esp.enable();
delay(500);
esp.reset();
```

```
 delay(500);

 while(!esp.ready());

.

.

.

mqtt.begin("dispositivo", "usuarioMQTT", "contraseña", 60, 1)
```

Únicamente debemos inicializar y conectar el módulo a la red WiFi utilizando el objeto ESP de la siguiente forma:

```
esp.WiFiConnect("nombre_de_la_red","contraseña");
```

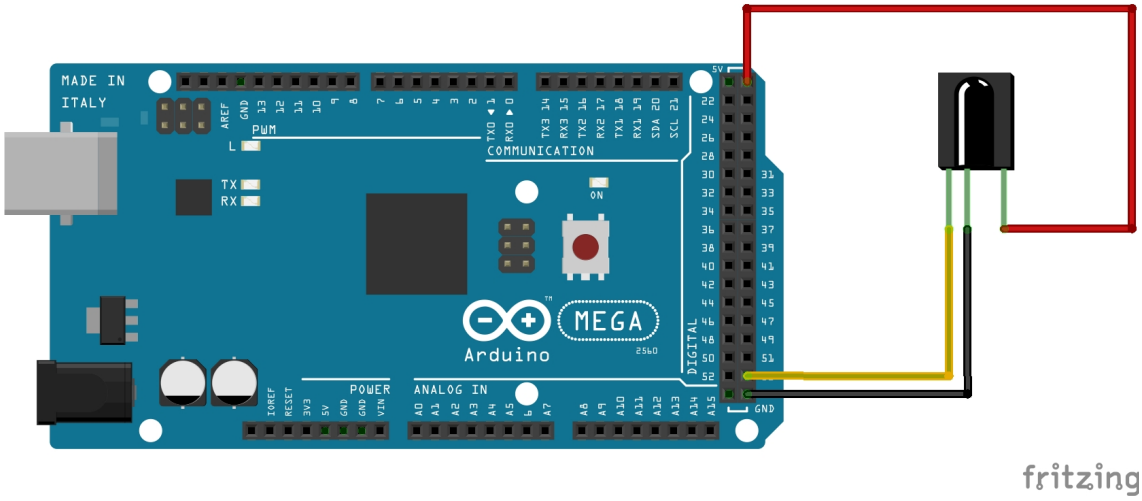
y conectar el módulo al servidor MQTT utilizando la función siguiente:

```
mqtt.connect("192.168.1.132", 11883, false);
```

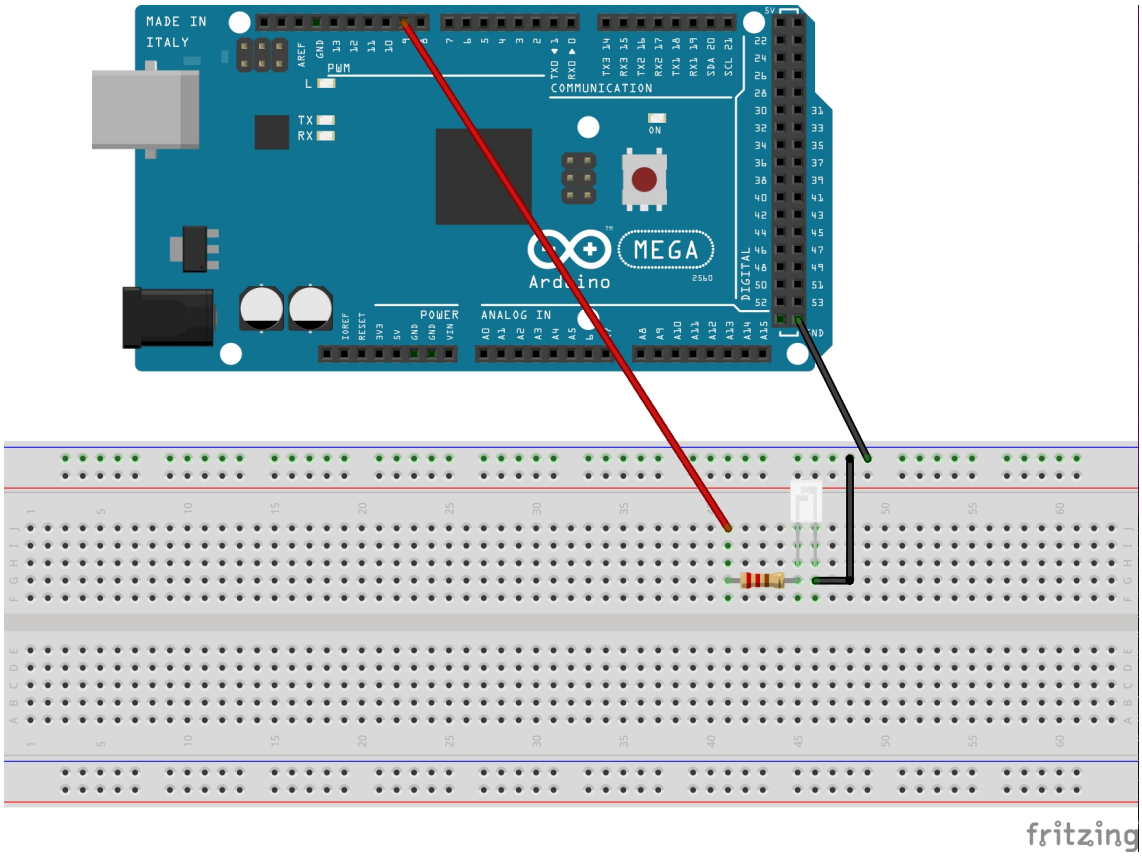
En la que debemos fijar tres parámetros: IP, puerto y seguridad requerida. En esta versión, sin embargo, las comunicaciones utilizando seguridad no son totalmente funcionales, por lo qu no es posible incluirlo en el presente proyecto.

Con todo esto, disponemos de una placa Arduino con capacidad para comunicarse utilizando un protocolo IoT, lo que nos aporta posibilidades prácticamente infinitas. Sin embargo, este sistema no realiza ninguna acción por sí mismo más allá del hecho de permitir enviar y recibir información. Por este motivo llegamos a la siguiente tarea que debe realizar este cliente, el aprendizaje y repetición de códigos IR para el control de un equipo de aire acondicionado doméstico.

Para ello, el primer paso es realizar el montaje en dos placas diferentes del emisor y el receptor de infrarrojos. De este modo nos será más fácil comprobar que el sistema es capaz de recibir el código correcto desde el control remoto original del equipo y posteriormente es capaz de replicarlo de forma exacta para que el sistema lo pueda interpretar.



Conexionado del receptor de IR con Arduino



Conexionado de LED emisor de IR con Arduino

Para poder operar con códigos IR necesitamos la librería IRremote. Esta librería permite codificar y decodificar información con protocolo IR. Una vez realizado el montaje, debemos implementar dos códigos simples. El primero de ellos se ejecutará en la placa conectada al receptor y mostrará por pantalla la representación del código enviado por el control remoto original del aparato. Utilizando este código como constante en programa de la placa conectada al emisor, veremos si el código emitido por el control remoto y por el emisor Arduino son iguales. En caso afirmativo, seremos capaces de controlar el sistema de aire desde nuestra placa.

Una vez se comprueba la funcionalidad de la forma anterior, debemos montar el receptor y el emisor en la misma placa. De esta forma, el siguiente programa a implementar deberá tener dos modos de funcionamiento, el modo aprendizaje (donde se memorizan los códigos) y el modo repetición (donde se emiten uno a uno para comprobar si el funcionamiento es correcto). Para ello se crea una matriz en la que se almacenarán los códigos elegidos, esto es, los códigos de temperatura desde 20º C hasta 27º C y el código de apagado. En el modo aprendizaje se van almacenando los códigos en la posición correspondiente, guardando además la longitud de los códigos de temperaturas y del código de apagado. Esto se hace porque en algunos sistemas la longitud del código de apagado es menor, debido a que el código no necesita contener tanta información. De este modo, el sistema es mucho más flexible e independiente y podría funcionar con un número mayor de fabricantes de estos sistemas.

Es importante destacar que la operación de memorización de los códigos requiere cierta atención por parte del usuario, ya que si bien no llega a ser compleja si que es necesario tener claros los siguientes aspectos. El mando del aire acondicionado debe ser preparado a la temperatura inicial de memorización antes de comenzar a emitir los códigos. El receptor es muy sensible, por lo que la preparación del mando a la temperatura de 20º C debe hacerse en otra estancia o tapando el emisor del control remoto con la mano. Una vez hecho esto, se debe apagar el mando. Para comenzar a memorizar los códigos, únicamente será necesario encender el control remoto, lo que ya enviará la temperatura de 20º C, que es la última que se fijó. A partir de ese momento se irá subiendo la temperatura de grado en grado para que el sistema vaya almacenándolas. La última temperatura a registrar será la de 27º C y el último código a guardar será el de apagado.

Como esto puede resultar complejo si nunca se ha tratado antes o no se comprende su funcionamiento, se ha decidido añadir al sistema una pantalla para poder mostrar la información de configuración sin necesidad de que la placa Arduino tenga que estar conectada a un PC.

Esto nos lleva a la tercera tarea que debe realizar cliente, la emisión de mensajes mediante un módulo de pantalla LCD. El módulo ya se ha descrito en el apartado de hardware, siendo su uso relativamente sencillo gracias a la libería Adafruit_PCD8544 dependiente de la librería Adafruit_GFX. Estas librerías, disponibles en GitHub, permiten mostrar por nuestra pantalla tanto texto como iconos, imágenes, etc. De esta forma, el uso de la pantalla para mostrar información es tan simple como el uso de la función *print()* para el terminal serie de Arduino.

Utilizando el ejemplo que incluye esta librería, podemos implementar el código necesario para emitir por esta pantalla los mensajes necesarios para facilitar al usuario la tarea de memorización de los códigos.

Integrando los códigos de todas las tareas en uno solo, dispondremos de un cliente Arduino capaz de comunicarse son un sistema de Internet de las Cosas y controlar un sistema de aire acondicionado utilizando un patrón de aprendizaje y repetición de las órdenes originales del control remoto de éste.

Se muestran a continuación el diagrama de flujo del programa que se ejecuta sobre este cliente para observar su comportamiento global y una fotografía del montaje final del cliente.

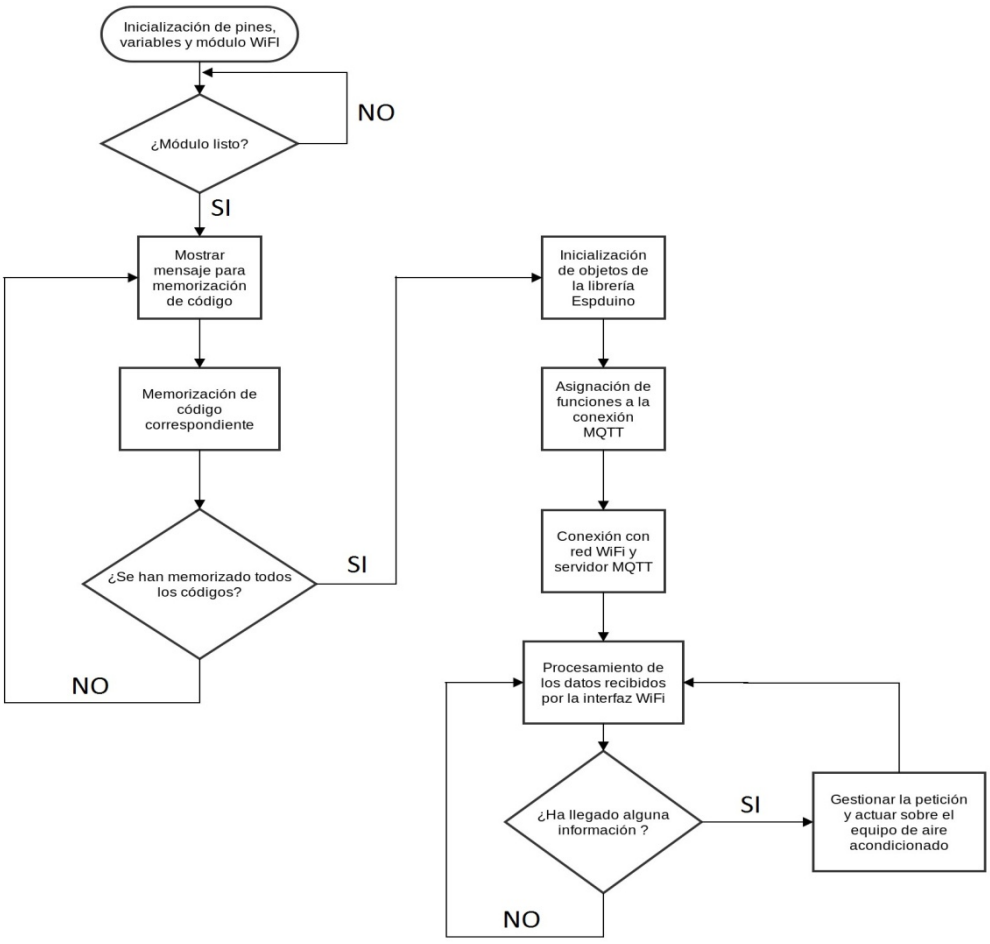
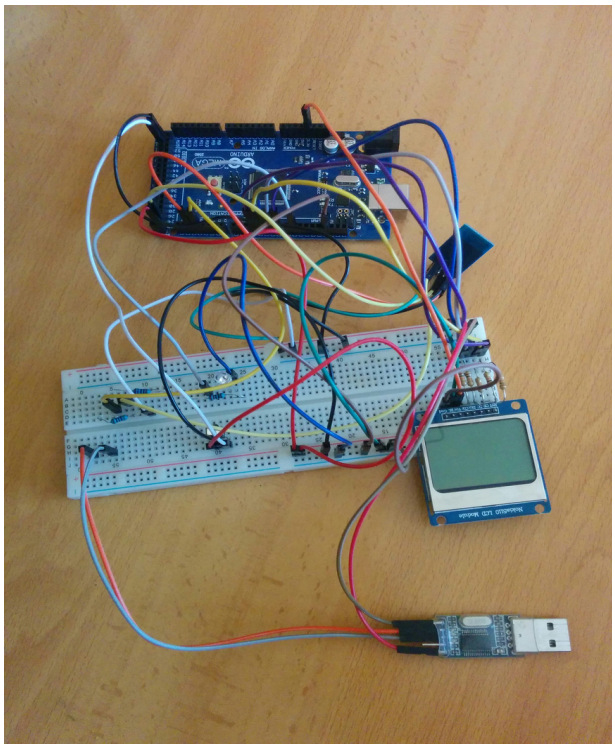


Diagrama de flujo del código del cliente



Montaje final del cliente para control de aire acondicionado.

6.2.2 Cliente de control de calefactor

En segundo lugar se profundizará en la implementación del firmware que gestionará el cliente Arduino conectado al calefactor.

Para éste código, se ha utilizado la librería Espduino de la misma forma que en el cliente de control del aire acondicionado. Su estructura también es la misma, exceptuando el aprendizaje y repetición que sí encontramos en el cliente del aire acondicionado y la conexión con el módulo de pantalla LCD.

De esta forma, el código de este cliente tiene la misma base que el cliente que se ha explicado arriba con las funciones de manejo del módulo de red y del objeto MQTT para realizar las conexiones oportunas y poder tratar los datos de igual forma que se hacia en el anterior.

La diferencia principal es la suscripción que se realiza en la función *mqttConnected()*, siendo en este caso al tópic */rele*, el cual recibe la información publicada por el front-end cuando éste recibe alguna orden del usuario. También cambia el código de tratamiento de información en la función *mqttData()*, de forma que permita conectar y desconectar el aparato doméstico conectado al relé cuando se reciba por el servicio MQTT la orden de encender o apagar. De esta forma es posible implementar un segundo cliente reutilizando casi la totalidad del código del cliente anterior.

También se debe tener en cuenta el manejo del relé, que es el componente que nos permite activar y desactivar el aparato a controlar. Este componente es muy fácil de controlar desde el punto de vista de Arduino, ya que únicamente necesitamos conectar un pin digital al puerto de control del relé y alimentar el módulo con los pines de tensión de la placa (5V y GND). Una vez hecho esto, lo único que tenemos que hacer es fijar niveles de tensión al pin de control del relé, con lo que éste abrirá o cerrará el circuito de control del aparato en cuestión. Este pin de control se fija en Arduino como un pin de salida se fija a valores alto o bajo con las funciones básicas disponibles en Arduino para ello (*pinMode()* y *digitalWrite()*).

A continuación se puede observar un diagrama de flujo del firmware para este cliente y una fotografía del montaje del sistema completo.

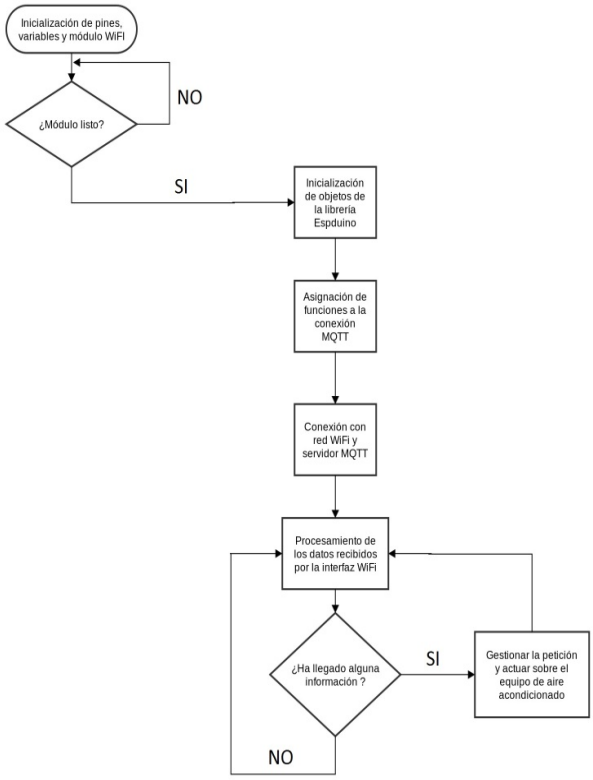
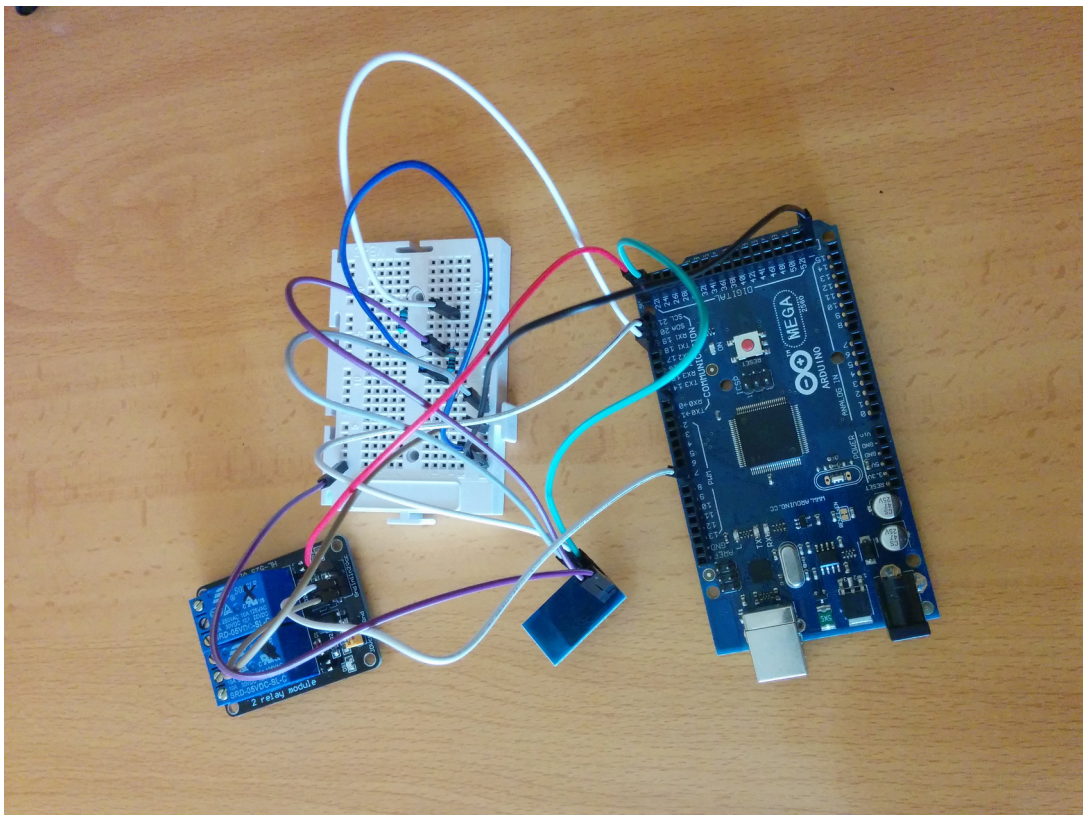


Diagrama de flujo del código del cliente



Montaje del cliente para control de un calefactor.

6.2.3 NodeMCU con sensor de temperatura y humedad

Por último debemos analizar el código que permite a la placa NodeMCU conectarse a la red WiFi, establecer conexión con el servidor MQTT y recoger información del sensor de temperatura y humedad DHT11 para poder publicarla en los *topics* correspondientes y que, de esta manera, pueda llegar a la placa Arduino que se haya suscrito en ellos previamente.

Ya hemos hablado del firmware necesario para que este hardware pueda trabajar de forma autónoma y también de la herramienta que hemos utilizado para desarrollar y trabajar con esta plataforma, por lo que en este punto nos centraremos en la implementación del código que nos permitirá que esta placa pueda cumplir con su función.

El firmware del que hemos dotado a la placa es capaz de gestionar por sí mismo todo el hardware incluido, desde el microcontrolador hasta la interfaz serie o los puertos GPIO. De esta forma la placa puede funcionar sin la necesidad de un dispositivo maestro que esté enviándole órdenes continuamente. Por este motivo se ha elegido como solución para ubicar el sensor de

temperatura sin necesidad de conectarlo a otra placa Arduino que supondría problemas de coste, espacio e instalación.

Este firmware permite la comunicación vía serie con otros dispositivos, lo que nos da la opción de comprobar su funcionamiento además de enviarle comandos linea a linea o programas enteros. El lenguaje de programación, como ya se ha comentado antes en éste proyecto, es LUA. Es un lenguaje de script similar a Python que nos permite usar la API disponible desde el firmware para crear programas que controlen el dispositivo. Las funciones que nos aporta este firmware son muy variadas, y van desde el control de la interfaz WiFi hasta conexión con servidores MQTT o control de sensores y actuadores simples mediante los puertos GPIO.

El firmware incluye un sistema de ficheros que nos permite almacenar programas en la memoria flash del chip y poder ejecutarlos en el arranque. Esto es posible renombrando el fichero deseado a “init.lua”. De esta forma, cuando el chip arranca busca en la memoria el código identificado por ese nombre y comienza a ejecutarlo.

Se ha probado con diversos programas, de los cuales algunos permiten su ejecución al inicio y otros llevan al chip a un arranque fallido. Parece que es un problema común del firmware debido a su reciente aparición y su aún escaso análisis por parte de la comunidad de desarroladores.

En la práctica, se debe conectar la placa al PC para iniciar la ejecución del programa. Una vez arranca y se lanza la ejecución, es posible desconectar el módulo del PC e instalarlo en el lugar que se desee, siempre que se mantenga alimentado y no se reinicie por ninguna causa.

Veremos parte por parte que funciones se emplean y para que funciona cada una de ellas. Las primeras lineas de código corresponden al establecimiento de la conexión de la interfaz WiFi con el punto de acceso.

```
WiFi.setmode(WiFi.STATION)

WiFi.sta.config("nombre_de_la_red", "contraseña")
```

Seguimos con la inicialización de las variables necesarias para la conexión con el broker MQTT.

```
BROKER = "ip_servidor_MQTT”
```

```
BRPORT = “puerto_broker_MQTT”

BRUSER = "usuario_en_caso_de_requerir_autenticación_en_el_broker”

BRPWD  = "contraseña_del_usuario”

CLIENTID = "identificación_de_dispositivo_en_el_broker”
```

A continuación fijamos los parámetros anteriores y establecemos la conexión con el servidor MQTT. Una vez conectado llamamos a la función *run_main_prog()*, que lanza la lectura de temperatura y humedad y publica los valores en el tópic correspondiente cada dos segundos

```
m = mqtt.Client( CLIENTID, 30, BRUSER, BRPWD)


m:connect( BROKER , BRPORT, 0, function(conn)

    run_main_prog()

end)
```

Estas tareas están separadas en funciones diferentes y cada una llama a la siguiente hasta que los datos son publicados. La primera tarea a la que se llama desde *run_main_prog()* es *read()*, que obtiene del sensor DHT11 la temperatura y la humedad. Para ello, el firmware incluye librerías de control de sensores y actuadores, entre los que se incluye una que nos permite obtener los datos de este sensor. El código de la función es el siguiente:

```
function read()
  pin = 4
  status,temp,humi,temp_decimial,humi_decimial = dht.read11(pin)
  if( status == dht.OK ) then
    -- Float firmware using this example
    temperature=temp
    humidity= humi
  end
  publish_temp()
```

```
end



function run_main_prog()

   tmr.alarm(2, 5000, 1, read )

end
```

Desde la función anterior se llama a *publish_temp()*, que se encarga de publicar el valor de la temperatura leída del sensor en el *topic /temp*. Cuando publica el resultado llama a la función *publish_hum()*, que publica en este caso el valor de la humedad leído en el *topic /hum*. El código de estas funciones se puede ver a continuación.

```
function publish_temp()

  m:publish("/temp",temperature,0,0, function(conn)

  end)

  publish_hum()

end




function publish_hum()

  m:publish("/hum",humidity,0,0, function(conn)

  end)

end
```

Con este programa, la placa NodeMCU es capaz de publicar los valores de humedad y temperatura en el servidor MQTT de forma autónoma.

Para poder observar con más facilidad el flujo de ejecución del código, se ha preparado el siguiente diagrama, además de añadir también una fotografía del montaje final.

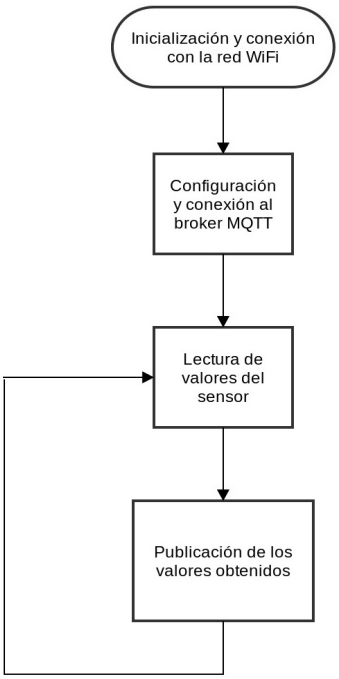
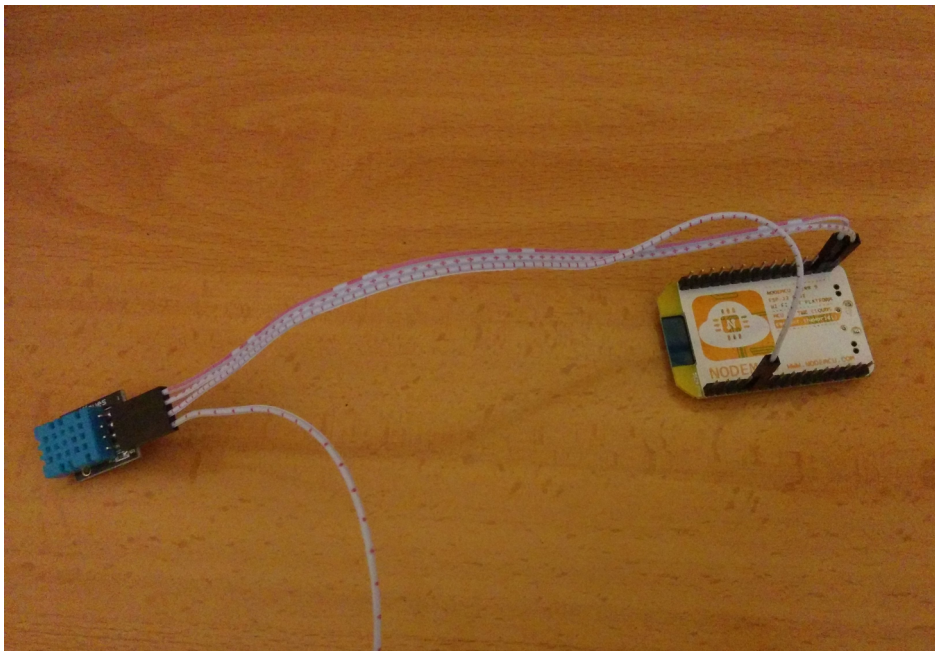


Diagrama de flujo del código del cliente NodeMCU.



Montaje del cliente NodeMCU con sensor DHT11

6.3 Funcionamiento global del sistema

Para que cada componente funcione del modo esperado, necesitamos seguir unos pasos a la hora de iniciar los componentes del sistema. Esto nos permitirá asegurar que los elementos independientes son inicializados anteriormente a los dependientes.

Así pues, el primer elemento que se debe inicializar es el broker del servicio MQTT. De esta forma, activamos el servicio básico de comunicaciones del sistema. Una vez éste está activo, podemos arrancar los servicios adicionales de servidor o los clientes hardware. Los clientes hardware no tienen un orden necesario de inicialización, ya que cada uno únicamente se comunica con el broker MQTT, que sí ha sido inicializado. Los servicios, sin embargo, si deben seguir un orden a causa del uso que hace el front-end del servicio de registro de eventos. Por lo tanto, primero se debe arrancar el servidor de registro, y posteriormente se activará el servicio de front-end.

Para que los diferentes elementos del sistema se puedan comunicar correctamente es necesario indicar correctamente la dirección de los servicios principales a los servicios que los usan. Por ello, hay que configurar correctamente los cliente Arduino y NodeMCU, así como el front-end, para indicar la dirección IP del broker MQTT y permitir la comunicación entre ellos. Del mismo

modo, hay que indicar en el servicio de registro de eventos la dirección IP del servidor MySQL y a su vez, indicar al front-end también la dirección del servicio de registro.

En este punto, el sistema está configurado y listo para funcionar del modo esperado.

7. Pruebas

Una vez conectados todos los componentes, implementados los servicios y programas que se ejecutarán en los clientes y arrancados todos los componentes del sistema, es el momento de realizar unas pequeñas pruebas para comprobar su correcto funcionamiento global.

Es necesario destacar que la conectividad de los componentes y servicios que forman el sistema, así como el normal funcionamiento de todas sus tareas, ya han sido comprobados durante la etapa de desarrollo. Esto es necesario dada la integración de todos los elementos para que el sistema funcione de forma adecuada, lo que resultaría difícil de analizar si probamos todas las partes al mismo tiempo.

Las pruebas que se han llevado a cabo para los elementos de forma individual son muy dependientes del tipo de programa o servicio implementado.

Para los servicios del sistema se ha comprobado la conexión entre ellos y con el broker MQTT, la recogida de datos de las peticiones y las acciones llevadas a cabo por estos.

Por otro lado, se ha comprobado en todos los clientes que es posible conectarse a la red WiFi. Una vez realizada esta comprobación, se han implementado códigos simples para probar la conexión de clientes al broker MQTT y realizar suscripciones a *topics*. Estas pruebas satisfactorias con un número elevado de suscripciones por cliente (más de 50) indican que el sistema podría escalar con facilidad en el futuro, añadiendo más sensores o incluso clientes que controlen más sistemas domésticos. Para el cliente de control del calefactor se ha comprobado que se conecta y desconecta el calefactor de forma correcta. Por otro lado, se han comprobado todas las tareas que realiza el cliente de control del aparato de aire acondicionado comprobando: el funcionamiento del módulo de pantalla LCD, el control correcto del aparato de aire enviando las órdenes almacenadas y observando si el equipo de aire responde y la adecuada recepción y tratamiento de la información recibida por los *topics* para actuar posteriormente acorde a ella.

Después de haber realizado estas pruebas en la fase de implementación, podemos pasar a probar el sistema de forma global sin que, presumiblemente, existan grandes contratiempos de funcionamiento.

Para ofrecer un entorno de servidor donde alojar los servicios del sistema, se ha creado una máquina virtual con el software de virtualización Virtualbox. Esta máquina ejecutará un sistema operativo Ubuntu Server en su versión 14.04.2. Sobre este sistema operativo se ha instalado la herramienta LAMP, una pila de software empresarial para desarrollo web que incluye la mayor

parte del software necesario a la hora de ejecutar los servicios desarrollados en este proyecto (de la que ya se ha hablado). El servidor de registro de eventos, por motivos de seguridad, se conecta a la interfaz de red *localhost* para evitar que cualquier elemento externo a esta máquina pueda falsear la base de datos de los registros. De este modo, únicamente el front-end puede enviar peticiones de registro de nuevos eventos y consultarlos.

De este modo, se configuran e inicializan todos los elementos de acuerdo a lo establecido en el apartado anterior y se comienza a probar el sistema de forma completa.

El primer punto es comprobar que los clientes y servicios se han inicializado de forma correcta y se conectan y suscriben a los *topics* del servidor MQTT. Una vez cumplido esto, los componentes ya son capaces enviar y recibir información, según corresponda a la función que deben realizar.

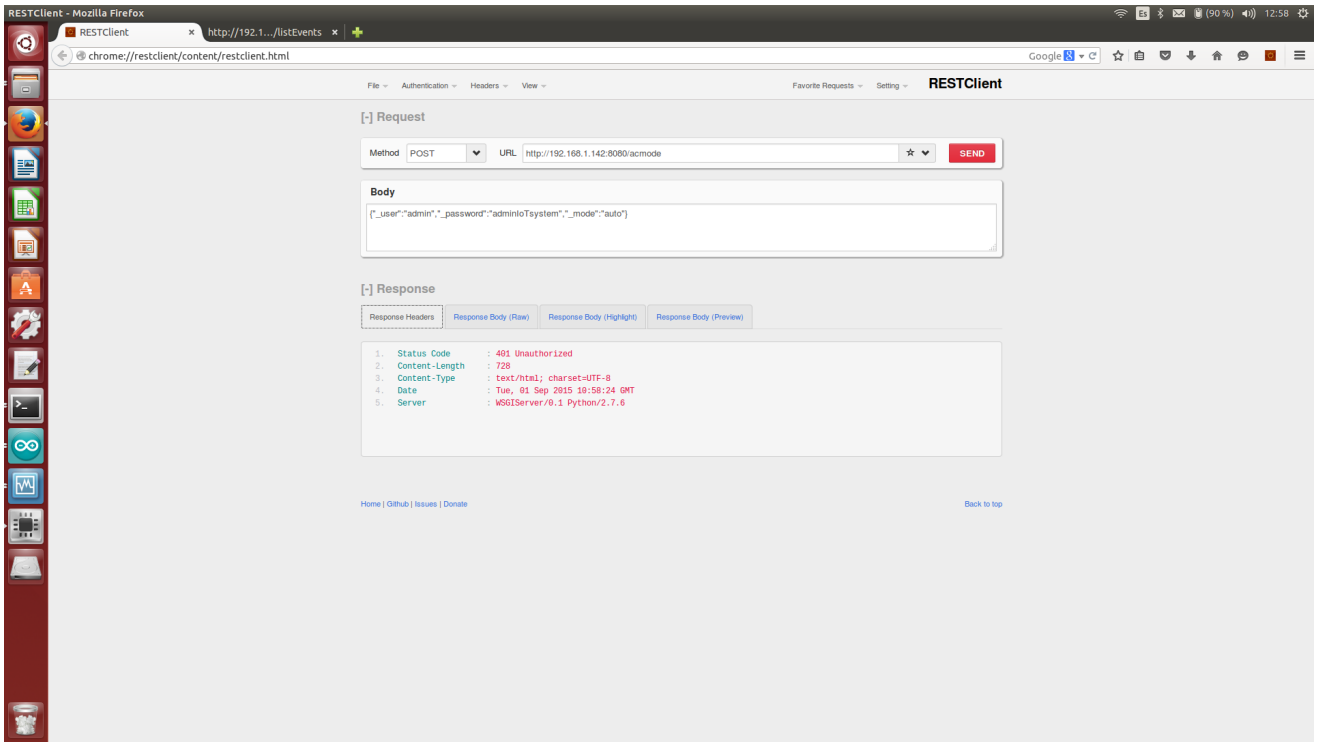
El paso siguiente es utilizar un cliente REST desde navegador en un PC conectado a la misma re que nuestro sistema para acceder al front-end y enviar órdenes tanto al cliente de aire acondicionado como al del calefactor. Primero probamos a introducir datos erróneos en las peticiones y observamos que el servidor nos responde con un mensaje de parámetros incorrectos. Una vez realizada esta comprobación, comenzamos a enviar órdenes correctas. Enviamos encendido del calefactor y comprobamos que se conecta. Fijamos modo manual y temperatura de 22º C en el aire acondicionado y comprobamos que también se conecta. Enviamos orden de apagado del calefactor y observamos como también se desconecta. A continuación enviamos la orden de apagado manual del aire acondicionado y observamos como permanece apagado.

Probadas todas estas funcionalidades, enviamos la orden de modo de funcionamiento automático al aire acondicionado y conectamos otro cliente. Desde éste, volvemos a enviar la orden de activado del calefactor.

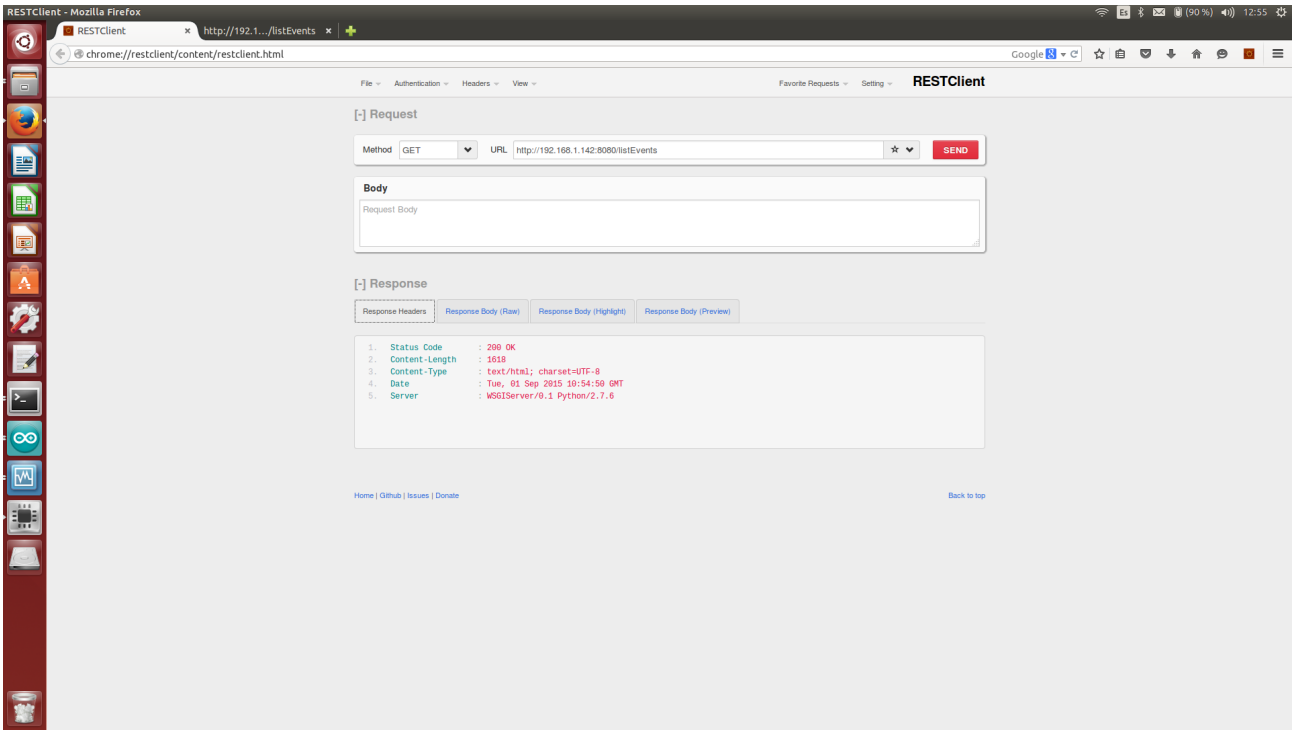
Una vez realizadas estas pruebas el sistema queda con el aire acondicionado funcionando en modo automático y el calefactor conectado. Desde el primer cliente enviamos la petición de listar todos los eventos registrados y analizamos la respuesta. En la respuesta se observa que todas las órdenes que hemos enviado al sistema han quedado registradas.

Finalizadas todas estas pruebas, podemos repetirlas accediendo al front-end desde un cliente externo a la red donde se encuentra el sistema. Para ello es necesaria la configuración del router de acceso a Internet de forma que permita el reenvío de puertos al servicio para que éste sea accesible desde Internet.

Con esto quedaría debidamente probado el funcionamiento del sistema IoT con orientación domótica desarrollado en el presente proyecto.



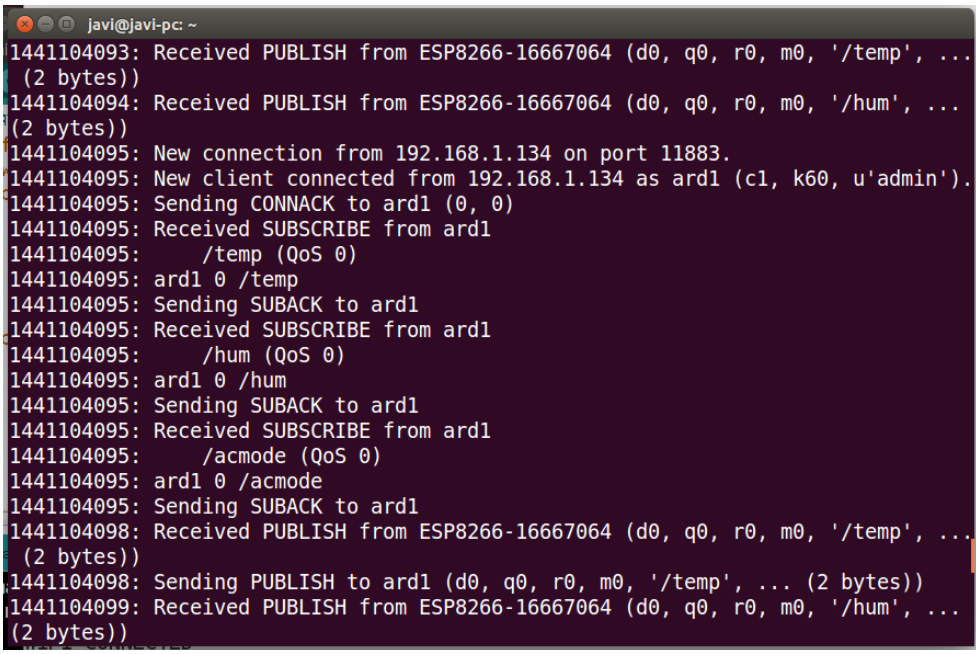
Captura de acceso al front-end con usuario o contraseña incorrectos.



Captura del resultado de la consulta al servicio de registro de eventos

```
javi@javi-pc: ~
1441102273: Sending CONNACK to paho/66AA8A391649D04133 (0, 0)
1441102349: Socket error on client paho/66AA8A391649D04133, disconnecting.
1441102356: New connection from 192.168.1.142 on port 11883.
1441102356: New client connected from 192.168.1.142 as paho/152EE7E91DDFC2EE69 (
c1, k60).
1441102356: Sending CONNACK to paho/152EE7E91DDFC2EE69 (0, 0)
1441102445: Client paho/152EE7E91DDFC2EE69 has exceeded timeout, disconnecting.
1441102445: Socket error on client paho/152EE7E91DDFC2EE69, disconnecting.
1441102963: New connection from 192.168.1.140 on port 11883.
1441102963: New client connected from 192.168.1.140 as ESP8266-16667064 (c1, k30
, u'user').
1441102963: Sending CONNACK to ESP8266-16667064 (0, 0)
1441102968: Received PUBLISH from ESP8266-16667064 (d0, q0, r0, m0, '/temp', ...
 (2 bytes))
1441102969: Received PUBLISH from ESP8266-16667064 (d0, q0, r0, m0, '/hum', ...
(2 bytes))
1441102973: Received PUBLISH from ESP8266-16667064 (d0, q0, r0, m0, '/temp', ...
 (2 bytes))
1441102974: Received PUBLISH from ESP8266-16667064 (d0, q0, r0, m0, '/hum', ...
(2 bytes))
1441102978: Received PUBLISH from ESP8266-16667064 (d0, q0, r0, m0, '/temp', ...
 (2 bytes))
1441102979: Received PUBLISH from ESP8266-16667064 (d0, q0, r0, m0, '/hum', ...
(2 bytes))
```

Conexión y publicación de datos del cliente NodeMCU



Conexión, suscripción y publicación de datos para cliente Arduino.

8. Conclusiones y trabajo futuro

En la actualidad es posible implementar sistemas de Internet de las Cosas con componentes electrónicos asequibles y con plataformas de desarrollo sencillas. Arduino es una de las plataformas más extendidas para el desarrollo e implementación de este tipo de sistemas, ya que cuenta con un amplio soporte por parte de su comunidad de usuarios y desarrolladores para este tipo de aplicaciones.

Sin embargo, existe una gran variedad de componentes tanto hardware como software que hacen necesario estudiar primero los requisitos de la aplicación para elegir la mejor solución que permita integración entre las diferentes partes del sistema y esté adaptada al tipo de dispositivo sobre el que va a trabajar.

Con el presente proyecto se ha podido comprobar la viabilidad del desarrollo de sistemas de control domótico apoyados en tecnologías de Internet de las Cosas con Arduino, utilizando para ello métodos de comunicación que implementan protocolos IoT y permiten la integración de dispositivos con recursos limitados. También se ha demostrado que es posible gobernar diferentes aparatos domésticos utilizando pequeños componentes para Arduino.

Del mismo modo, se ha demostrado que existen soluciones incluso más económicas que también resultan sencillas de utilizar y que, siendo menos potentes que plataformas como Arduino, nos aportan muchas posibilidades en dispositivos muy pequeños, pudiendo crear de forma sencilla una red de estos dispositivos conectados a sensores y actuadores simples que permitan sistemas IoT mucho más amplios y escalables.

Debido al rápido crecimiento que está experimentando esta tecnología, y la integración que se puede hacer de ella en sistemas críticos tanto por privacidad como por riesgo para las personas, se deberían estudiar en el futuro ciertos aspectos que debidos a la implementación en Arduino conlleva un gran nivel de complejidad implementar. También existen aspectos menos críticos, pero que sin embargo aportarían valor añadido al sistema y que permitirían una mejor interacción con el sistema de control de los equipos domésticos controlados por el sistema.

Algunos de los aspectos sobre los que se podría continuar el proyecto en el futuro son:

- Añadir seguridad en todas las comunicaciones internas (entre clientes y broker MQTT).

- Mejorar sistema de comprobación de usuario y contraseña en los accesos al front-end del sistema con base de datos de usuarios permitidos.

- Registrar todas las conexiones al front-end para analizar posibles ataques y accesos no autorizados.

- Crear una interfaz web o una aplicación móvil para facilitar el envío de órdenes y las consultas que se realizan al sistema desde el front-end.

- Integrar más aparatos domésticos para tener un control más amplio de los sistemas del hogar.

9. Bibliografía

Arduino:

http://playground.arduino.cc/Main/ArduinoPinCurrentLimitations [28-06-15]

https://github.com/esp8266/Arduino [08-07-15]

ESP8266:

http://www.roboremo.com/flashing-nodemcu-firmware-to-esp8266.html [15-07-15]

http://www.xess.com/blog/esp8266-reflash/ [08-07-15]

http://www.nodemcu.com/docs/index/ [23-07-15]

http://www.esp8266.com/ [02-08-15]

https://github.com/tuanpmt/espduino [28-07-15]

Servicios adicionales:

http://bottlepy.org/docs/dev/index.html [14-07-15]

http://mysql-python.sourceforge.net/MySQLdb.html [18-7-15]

https://eclipse.org/paho/clients/python/docs/ [23-07-15]

Tecnologías empleadas:

http://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php [12-8-15]

http://www.analysir.com/blog/2015/01/06/reverse-engineering-mitsubishi-ac-infrared-protocol/ [30-07-15]