UNIVERSIDAD DE VALLADOLID

E.T.S.I. TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO DE INGENIERÍA DE TECNOLOGÍAS ESPECIFICAS DE TELECOMUNICACIÓN, MENCIÓN EN SISTEMAS ELECTRÓNICOS

Sistema Electrónico de Control para Toldo Conectado a Internet

Autor:
David García Gutiérrez

Tutor:
Jesús M. Hernández Mangas

Valladolid, febrero de 2017

TÍTULO:	**Sistema electrónico de control para toldo conectado a Internet**
AUTOR:	**David García Gutiérrez**
TUTOR:	**Jesús M. Hernández Mangas**
DEPARTAMENTO:	**Electricidad y Electrónica**

Tribunal	
PRESIDENTE:	**José Vicente Antón**
VOCAL:	**Jesús Arias**
SECRETARIO:	**Jesús M. Hernández Mangas**
SUPLENTE:	**Pedro López**
SUPLENTE:	**Luis Marqués**
FECHA:	
CALIFICACIÓN:	

Resumen del proyecto

En la actualidad, la tecnología está evolucionando hacia el Internet de las Cosas. Debido a esto, se ha decidido realizar un toldo automático que se puede controlar desde un smartphone o una tablet. De esta manera, se puede extender o recoger el toldo desde cualquier lugar cuando se desee.

Mediante el desarrollo de este proyecto, se pretende diseñar una placa de circuito impreso con conexión a Internet a través de la tecnología Wi-Fi. La funcionalidad de ésta, será mover el toldo según la posición de unos interruptores. Además, tomará medidas de unos sensores y obedecerá órdenes del usuario desde una aplicación para el móvil. Este diseño incluye la parte hardware con su correspondiente software.
Además, es necesario el desarrollo de la aplicación para el smartphone o tablet, en la cuál se puede leer información sobre el estado del toldo y cambiar su posición.
Por último, es necesario establecer un protocolo de comunicación entre el toldo y la aplicación. En este caso, se ha decido utilizar MQTT.

Palabras clave

Internet de las cosas, Wi-Fi, MQTT, módulo Wi-Fi, Mosquitto, dispositivo, Arduino, Android Studio, hardware, software.

Abstract

Today, the technology is envolving towards the Internet of Things. Due to this, it has been decided to make an automatic awning that can be controlled from a smartphone or tablet. In this way, it can be extended or retracted from any place when the user wants.

Through the developmetns of this project, it is intended to design a printed circuit board with Internet connection through the Wi-Fi technology. The functionality of this, it will be to move the awning according to the position of some switches. In addition, it will take measurementents of the sensors and will obey user´s orders from a mobile application. This design includes the hardware part with its corresponding software.

In addition, it is necessary to develop an application for the smartphone or tablet, in which the user can be read information about the state of the awning and changes its position.
Finally, it is necessary to establish a communication protocol between awning and the application. In this case, it has been decided to use MQTT.

Key words

Internet of things, Wi-Fi, MQTT, Wi-Fi module, Mosquitto, device, Arduino, Android Studio, hardware, software.

Agradecimientos

A mis padres por ofrecerme la posibilidad de estudiar esta carrera y ser un punto de apoyo diario.
A mis compañeros de clase por hacer que estos años se hayan hecho más llevaderos.
A mis amigos de toda la vida por su apoyo durante estos años.
A mi tutor por darme la posibilidad de llevar a cabo este proyecto.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Objetivos

El objetivo de este proyecto es el diseño e implementación de un dispositivo conectado al Internet de las Cosas (*Internet of Things, IoT*) que permita controlar la posición de un toldo.

La posición se podrá cambiar a través de una aplicación para el móvil o mediante dos interruptores. Para ello, este sistema electrónico deberá tener conexión a Internet. Además, dispondrá de un sensor de temperatura exterior, un sensor de humedad exterior, una conexión para anemómetro y un sensor de intensidad luminosa. Con la información de estos sensores, se implementará un modo automático en el cuál se subirá o bajará el toldo según la proximidad de las mediciones de los sensores a unos umbrales que se podrán cambiar a través de la aplicación.

También se podrá activar un modo vacaciones para que el toldo suba o baje entorno a una hora concreta a la que se añadirán algunos minutos aleatoriamente para que no cambie de posición exactamente a la hora definida. En la aplicación se podrán definir dos horas, una para que el toldo baje y otra para que suba.
Para controlar el giro del motor del toldo se utilizarán dos Triac, uno para cada sentido de giro.

1.2. Decisiones del diseño

Como se ha comentado en los objetivos, el sistema electrónico a diseñar debe tener conexión a Internet. Para ello, se ha escogido la tecnología Wi-Fi ya que en casi todos los hogares hay una conexión Wi-Fi. En el apartado 1.4 se hace un estudio de las diferentes opciones y sus características. Para que el dispositivo se pueda a conectar al Wi-Fi del hogar debe tener un módulo Wi-Fi. Buscando las diferentes opciones que hay en el mercado, se ha seleccionado el módulo ESP8266.

Una vez que el dispositivo está conectado a Internet, necesita de un protocolo para comunicarse con la aplicación del móvil. Estudiando los diferentes protocolos de comunicación se ha selecciona MQTT, el cuál se explicará en el apartado 1.5.

Tomadas estas decisiones, hay que elegir el microcontrolador que debe llevar el dispositivo. Este debe tener un convertidor analógico-digital para el anemómetro, dos pines para permitir la comunicación I2C con los sensores, un RTC (*Real Time Clock, Reloj de Tiempo Real*) para saber la hora y dos UART(*Universal Asynchronous Receiver-Transmitter, Receptor-Transmisor Asíncrono Universal*), una para comunicarse con el módulo Wi-Fi y otra para poder programar el microcontrolador. Con todas estas necesidades, se eligió el LPC2103. Sin embargo, al estudiar el ESP8266 nos encon-

tramos con que el módulo lleva un microcontrolador interno que se puede programar. Además, tiene una UART, un convertidor analógico-digital y pines preparados para la comunicación I2C. Debido a estas características, se ha decidido utilizar el módulo Wi-Fi como microcontrolador, de esta manera se ahorra coste de fabricación y espacio en la PCB. En concreto, se ha escogido el ESP8266-12F puesto que tiene más pines disponibles para poder utilizarlos. En la sección de hardware ya se explicará el módulo con más detalle.

El resumen de las decisiones tomadas, es el siguiente:

- **Tecnología de comunicación y dispositivo para conectarse a Internet:** Wi-Fi y módulo Wi-Fi ESP8266-12F.
- **Protocolo de comunicación:** MQTT.
- **Microcontrolador:** Módulo Wi-Fi ESP8266-12F, puesto que tiene un microcontrolador interno que se puede programar.

1.3. Internet de las Cosas

Debido a que se va a diseñar un dispositivo del Internet de las Cosas, vamos a explicar brevemente que significa y en que consiste este concepto.
El IoT puede verse como un paradigma en el que los sistemas electrónicos cotidianos (coche, frigorífico,etc.) tienen la capacidad de conectarse a la red, enviar datos a través de ella y/o recibirlos, interpretarlos y actuar consecuentemente para nuestro beneficio. Estos datos se pueden obtener a través de los smartphones o tablets y desde otros dispositivos IoT. Debido a esta conexión entre dispositivos, también se les conoce como conexión M2M (*Machine to Machine, Máquina a Máquina*). En la Figura 1.1 se muestra una imagen de lo que representa el Internet de las Cosas, todo tipo de dispositivos conectados a la red.

Figura 1.1: Ejemplo de Internet de las Cosas

Un ejemplo de Internet de las Cosas es una nevera que está conectada a la red y avisa al usuario si la temperatura es baja o alta. Toda esta información la enviaría el frigorífico a un servidor y desde nuestro smartphone o tablet se podrá consultar.
Otro ejemplo, puede ser un ciudad inteligente (*Smart City*) en la que se instalan múltiples sensores y cámaras para recoger información sobre lo que está sucediendo en la ciudad. Está información es subida a un servidor en la red y desde los smartphones o desde el coche se puede consultar el estado de la ciudad. Por ejemplo, si se desea aparcar en una determinada calle te avisa de los aparcamientos que están libres o si una calle está cortada por algún accidente te avisa para que selecciones otra ruta para ir al destino.
Para poder conectar todos estos objetos a Internet, se necesita asignarles una dirección IP para que puedan ser identificados en la red. Debido a esto se está haciendo la transición del protocolo IPv4 a IPv6 ya que se pasa de tener 32 bits a 128 bits para direcciones IP. Este cambio provoca que se tengan 2^{96} direcciones más que con IPv4.
Además, para que los objetos se puedan conectar a la red deben tener alguna tecnología de comunicación inalámbrica incorporada. Algunas de estas tecnologías se explican en el siguiente apartado.
También será necesario el empleo de un protocolo de comunicación entre el dispositivo y el servidor para enviar y/o recibir datos. Estos protocolos se explicarán brevemente en el apartado 1.5, excepto MQTT que se estudiará más detalladamente por ser el empleado en este proyecto.
Por último, destacar que la principal preocupación del IoT que es la seguridad puesto que los datos personales (aficiones, datos médicos, etc) pueden estar en la red sin protección y podrían ser consultados por empresas que han instalado los dispositivos o por otras que compren esa información para fabricar productos según las aficiones de la gente. Esto provoca que la información privada de cada persona pueda estar en manos ajenas.

1.4. Tecnologías de comunicación inalámbrica

Como ya se ha comentado en el apartado anterior, en el IoT todos los dispositivos van a estar conectados entre sí y para ello, se necesita un estándar de comunicación inalámbrica. Actualmente existen muchos estándares y todavía no se ha impuesto ninguno sobre los otros.
A la hora de elegir un estándar se tienen en cuenta varios factores:

- **Consumo de energía:** ¿El dispositivo será alimentado por batería o será enchufado a la pared?. Los dispositivos con baterías deben conservar la mayor cantidad de energía posible para limitar la necesidad de cambiar o recargar la batería.
- **Velocidad de transmisión:** En este caso hay que tener en cuenta la cantidad y el tipo de datos que se van a intercambiar ya que no es lo mismo una aplicación de streaming que una de domótica. La aplicación de streaming requerirá mayor velocidad.
- **Alcance:** Se debe considerar la distancia que el dispositivo necesitará para enviar los datos. Esto dependerá de la aplicación.
- **Frecuencia:** La frecuencia con la que se comunica un dispositivo afectará su capacidad para penetrar en los edificios, así como la distancia total que puede recorrer la señal. En general, las frecuencias más bajas pueden penetrar mejor los edificios que las más altas.
- **Topología de la red:** Se debe tener en cuenta si la comunicación será directa entre dos dispositivos o si habrá un intermediario.

A continuación, vamos a explicar brevemente algunos de estos estándares y sus características:

1.4.1. Wi-Fi/IEEE 802.11

El Wi-Fi es una de las opciones más utilizadas por los desarrolladores ya que se puede encontrar un punto de acceso en cualquier hogar y muchos sitios públicos. Sus características son las siguientes:
Las velocidades de transmisión de datos que ofrece van desde 1 Mbps a cientos Mbps, en rangos de hasta 300 metros, dependiendo del estándar IEEE utilizado ya que tiene varias versiones.
Trabaja en las bandas de frecuencia de 2,4 GHz y 5 GHz.
Su mayor inconveniente es el elevado consumo de energía. Actualmente los proveedores de Wi-Fi están trabajando en reducir el consumo con nuevas características y estándares.
Su estándar más reciente es el IEEE 802.11ah, el cuál trabaja en el espectro de 915 MHz y puede proporcionar hasta 300 kbps en un rango de hasta 1 km. Además, añade nuevas características para prolongar la duración de la batería.

1.4.2. Bluetooth

Es uno de los principales estándares de comunicación de corto alcance. Conecta dispositivos inalámbricos con smartphones o tablets, lo que hace que sea como una pasarela a Internet, ya que estos dispositivos no tienen conexión directa a Internet. Algunos ejemplos de dispositivos que utilizan Bluetooth son los monitores de frecuencia cardíaca que suben sus datos a servidores o cerraduras controladas por el móvil que envían información de su estado a compañias de seguridad. Las características de Bluetooth son:
Trabaja en la frecuencia de 2,4 GHz.
Su rango de alcance va desde 50 metros a 150 metros.
La velocidad de transmisión que alcanza es de 1 Mbps.
Como en el caso del Wi-Fi, su mayor inconveniente es el elevado consumo de energía. Para solucionarlo, se ha desarrollado una versión del estándar de baja energía al que se conoce como Bluetooth

Smart (*BLE, Bluetooth Low Energy*). Bluetooth Smart es un protocolo útil para aplicaciones IoT ya que ofrece características similares a Bluetooth con reducción del consumo de energía. Sin embargo, tiene el inconveniente de que no está diseñado para la transferencia de una gran cantidad de datos.
La principal ventaja que tiene sobre otras tecnologías competidoras es su fácil integración en los smartphones y otros dispositivos móviles.

1.4.3. 6LoWPAN

6LoWPAN (*IPv6 oer Low power Wireless Personal Area Networks, IPv6 sobre Redes Inalámbricas de Área Personal de Baja Potencia*), es un protocolo pensado para dispositivos con un consumo de energía bajo. Esta diseñado para proporcionar conectividad a dispositivos muy pequeños como es el caso de los sensores. Su campo de aplicación es la domótica. Sus características son:
Está recogido en el estándar RFC6282. Opera en las bandas de 2,4 GHz, 868 MHz y 916 MHz.
Respecto a la velocidad de transmisión y su alcance se desconocen los datos ya que es una tecnología nueva y está en desarrollo.

1.4.4. Thread

Es un nuevo protocolo IPv6 que está diseñado específicamente para el campo de la domótica. Se basa en varios estándares: IEEE 802.15.4, IPv6 y 6LoWPAN. Sus principales ventajas son que tiene una alta escalabilidad, la seguridad, el bajo uso energético y su sencilla instalación (a través de su smartphone y tablet). Sus características son:
Trabaja en la banda de frecuencias de 2,4 GHz.
Respecto a la velocidad de transmisión y su alcance se desconocen los datos.
Su mayor inconveniente es que es una tecnología nueva y aún está en desarrollo.

1.4.5. Zigbee

Es un sistema de comunicación inalámbrica centrado en la comunicación entre dispositivos con una baja tasa de datos con el objetivo de tener el menor consumo energético. Es una tecnología basada en el estándar IEEE 802.15.4. Su campo de aplicación es la domótica. Las características de esta tecnología son:
Opera en el rango de 2,4 GHz pero también permite trabajar en las bandas de 868 MHz y 916 MHz.
La máxima velocidad de transmisión que puede alcanzar es de 250 kbps.
El rango de funcionamiento es de 10 a 100 metros.
Una de las principales ventajas de Zigbee es la facilidad y el bajo coste que supone fabricar un dispositivo con esta tecnología. Por ejemplo, sólo requiere un 10 % del hardware que es necesario para producir un dispositivo con Bluetooth.

1.4.6. NFC

La tecnología NFC (*Near Field Communication, Comunicación de Campo Cercano*) permite interacciones bidireccionales simples y seguras entre dispositivos electrónicos. Sus principales usos son el pago y la identificación:

- **Pago con el teléfono móvil:** Nos permite pagar acercando el teléfono móvil con chip NFC a un dispositivo de lectura. Este método de pago lleva camino de ser el método del futuro.
- **Identificación:** Se puede emplear para acceder a lugares donde es precisa identificación. Esto se puede realizar acercando el móvil con chip NFC a un dispositivo de lectura. Un ejemplo son los bonos de autobús.

Sus características principales son:
Está recogida en el estándar ISO/IEC 18000-3.
Funciona en la banda de los 13,56 MHz, para utilizar esta banda no hace falta licencia.
El rango de alcance es de 10 cm.
Las velocidades de transmisión de datos van desde 100 kbps a 420 kbps.

1.4.7. SigFox

Es una alternativa de amplio alcance, en términos de alcance está entre Wi-Fi y la comunicación móvil.
SigFox responde a las necesidades de muchas aplicaciones M2M que funcionan con una batería pequeña y no requieren grandes transferencias de datos. Se utiliza en dispositivos en los que el alcance de Wi-Fi se queda corto y la comunicación móvil es muy cara y consume demasiada energía.
SigFox utiliza la tecnología Ultra Narrow BAND (*UNB, Ultra Banda Estrecha*), la cuál está diseñada para funcionar con velocidades de 10 bps a 1000 bps. Sus principales características son:
Trabaja en el rango de frecuencias de 900 MHz.
Su alcance es de 30 a 50 km en ambientes rurales y de 3 a 10 km en ambientes urbanos.
Su velocidad de transferencia es de 10 a 1000 bps.

1.4.8. LoRa

LoRa es una red de comunicaciones basada en el protocolo LoraWAN (*Long Range Wide-area network, Red Extendida de Largo Alcance*). Al igual que SigFox, es una red de baja tasa de transferencia, no tiene un gran consumo de energía y tiene un alcance mayor que el Wi-Fi. Sus características son:
Opera la banda de frecuencias de 2,4 GHz.
Su alcance es de 15 km en entornos rurales y de 2 a 5 km en entornos urbanos.
La velocidad de transferencia es de 0,3 a 50 kbps.
Actualmente, la red LoRa está en fase de despliegue por lo que aún no hay muchos dispositivos compatibles con esta tecnología, ya que no se podrán utilizar hasta que esté desplegada completamente.

1.4.9. Elección

Como se ha podido comprobar, existen muchas tecnologías para conectar nuestro dispositivo a Internet. Es este caso, hemos elegido Wi-Fi ya que en prácticamente todos los hogares hay una conexión Wi-Fi, lo que hace que sea mucho más fácil instalar el toldo en cualquier casa.

1.5. Protocolo de comunicación entre la aplicación y el toldo

Todos los dispositivos IoT al estar conectados a la red deben cumplir con alguno de los protocolos de la Familia de Protocolos de Internet de la IETF(*Internet Engineering Task Force, Grupo de Trabajo en Ingeniería de Internet*). Estos protocolos son FTP, HTTP, POP, SMTP y TelNet. Su mayor inconveniente es que están diseñados para dispositivos ricos en recursos con mucha energía, memoria y opciones de conexión. Por lo que, estos protocolos se han considerado demasiado pesados para la mayoría de aplicaciones IoT, ya que estas suelen ser de baja potencia y con recursos limitados. Debido a esto se han tenido que desarrollar nuevos protocolos que abordan estas necesidades. A continuación, explicaremos brevemente estos protocolos y con algo más de detalle MQTT puesto que es el elegido para este proyecto.

1.5.1. XMPP

XMPP(*Extensible Messaging and Presence Protocol, Protocolo de Mensajería y Presencia Extendible*), es un protocolo diseñado específicamente para la mensajería. Los mensajes que envía son en formato XML. Permite a los usuarios enviar mensajes en tiempo real y gestionar la presencia del usuario (en línea, fuera de línea, ocupado).La versión para IoT admite enviar y recibir mensajes de máquinas. XMPP funciona sobre TCP. Algunas de las aplicaciones que lo utilizan son Facebook y WhatsApp.

1.5.2. AMQP

AMQP(*Advanced Message Queue Protocol,Protocolo Avanzado de Mensajes de Cola*), consiste en un estándar abierto para el intercambio de mensajes entre aplicaciones (M2M). Es un protocolo orientado a mensajes que proporciona características como el enrutamiento y la gestión de colas. Funciona sobre TCP.

1.5.3. REST

REST (*Representational State Transfer,Transferencia de Estado Representacional*), se emplea para el intercambio de datos entre aplicaciones y para integrar aplicaciones que pertenecen a varios dominios. Utiliza HTTP como el protocolo base, por lo que funciona sobre TCP. REST se basa en la estructura cliente/servidor y es ampliamente utilizado no solo en las aplicaciones IoT, sino también en las arquitecturas de servicios más modernas en la actualidad.

1.5.4. Stomp

Stomp (*Simple or Streaming Text Oriented Messaging Protocol, Protocolo de Mensajería Orientada a Texto Simple o en Streaming*), es un protocolo orientado a mensajería de texto. Los clientes se conectan a un intermediario para intercambiar mensajes. Esto posibilita que pueda ser utilizado desde cualquier lenguaje de programación, ya que el intermediario se encarga de enviar los mensajes a otros clientes.
Su funcionamiento es similar a HTTP, por lo que funciona sobre TCP.

1.5.5. CoAP

CoAP (*Constrained Application Protocol,Protocolo de Aplicación Restringido*), es un protocolo basado en la transferencia de mensajes entre aplicaciones M2M. Se ejecuta a través de UDP que es menos fiable que TCP, pero utiliza mensajería repetitiva para la confiabilidad en lugar de conexiones consistentes. Por ejemplo, si un sensor envía un mensaje y se pierde, el siguiente llegará a los pocos segundos y su contenido será parecido al anterior. Los mensajes UDP son más pequeños que los de TCP, esto provoca que los periodos de transmisión sean más cortos.
Se trata de un protocolo cliente-servidor donde el cliente realiza una solicitud y el servidor devuelve una respuesta.

1.5.6. MQTT

Para este proyecto se ha elegido el protocolo MQTT porque es más maduro que por ejemplo, CoAP. Además, de que para el módulo Wi-Fi ESP8266 ya existen librerías en Arduino que permiten el fácil manejo del protocolo, evitando de esta manera el tener que utilizar los comandos AT.
MQTT (Message Queue Telemetry Transport) es un protocolo utilizado para la comunicación máquina a máquina (M2M) en el Internet de las Cosas. Es un protocolo simple y ligero ya que requiere pocos recursos para la transmisión de mensajes cortos de telemetría y de control, desde/hacia una red

de sensores/actuadores. Además,su implementación está basada en un mecanismo de publicación-suscripción, lo que hace necesaria la existencia de un tercer elemento, que recibe el nombre de "broker". El broker actúa como canal de comunicación entre los actores que publican información y los actores que reciben dicha información. Aunque MQTT no es una invención de IBM (International Business Machines), ha sido esta empresa la que ha contribuido en su impulso definitivo, sacando en el año 2010 la versión 3.1 open source. Desde el año 2014, existe un estándar a cargo del organismo OASIS (Advancing Open Standards for the Information Society). La versión más actual del protocolo es la 3.1.1.
El protocolo MQTT se encuentra en las capas superiores del modelo OSI, y normalmente se apoya en TCP/IP. Esto provoca que los participantes de una aplicación MQTT deben tener una pila TCP/IP. A continuación, en la Figura 1.2, se muestra el modelo de referencia OSI y en que capas está MQTT.

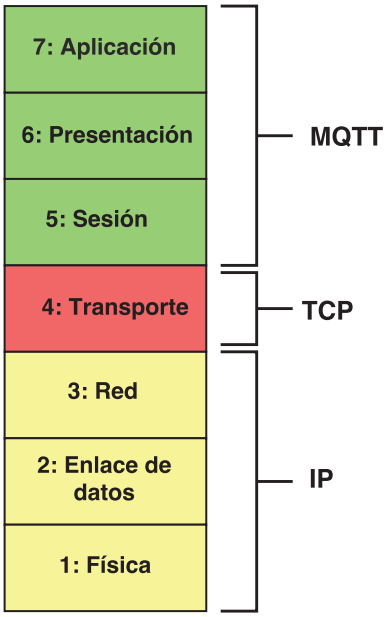


Figura 1.2: Ejemplo protocolo MQTT

Características

El protocolo MQTT presenta una serie de características que lo hacen idóneo para el mundo del Internet de las Cosas:

- Como se ha comentado anteriormente, se basa en la publicación-suscripción, esto provoca que no se necesita conocer el origen o el destino de los mensajes, reduciendo la complejidad de la red. Los clientes sólo necesitan conocer la dirección IP del broker.
- Debido a que es un protocolo basado en la publicación-suscripción, se emplean cabeceras muy reducidas, la comunicación sólo se produce bajo demanda, etc. lo que hace que sea un protocolo ligero. Por tanto, se ajusta a redes y dispositivos con pocos recursos y baja velocidad de transmisión, como una red de sensores.
- La existencia de un elemento intermediario, broker, junto con la simplicidad de su implementación, da lugar a una gran escalabilidad de la red ya que solamente con aumentar recursos en el broker se pueden añadir un mayor número de clientes.
- Implementa hasta tres niveles de QoS (calidad de servicio), lo que permite garantizar la fiabilidad y la integridad de la comunicación.

Funcionamiento

La estrategia de comunicación del protocolo MQTT se basa en la publicación de mensajes de tipos específicos, identificados con tópicos o topics, y la suscripción a estos mensajes. Los mensajes que se publican en la red se clasifican por diferentes tipos y los clientes MQTT reciben solamente los mensajes de los tópicos a los que están subscritos. Del mismo modo, los clientes también pueden publicar mensajes de los tópicos que manejan y de esta manera comunicarse con otros clientes que gestionan ese tipo de tópicos.
A continuación, en la Figura 1.3, se muestra un ejemplo en el que un sensor de temperatura publica la temperatura medida con el topic "temperatura", a este topic se suscriben los clientes MQTT, en este caso un ordenador y un móvil. A los clientes que están suscritos al topic temperatura les llegan los mensajes con la temperatura medida que publica el sensor de temperatura.

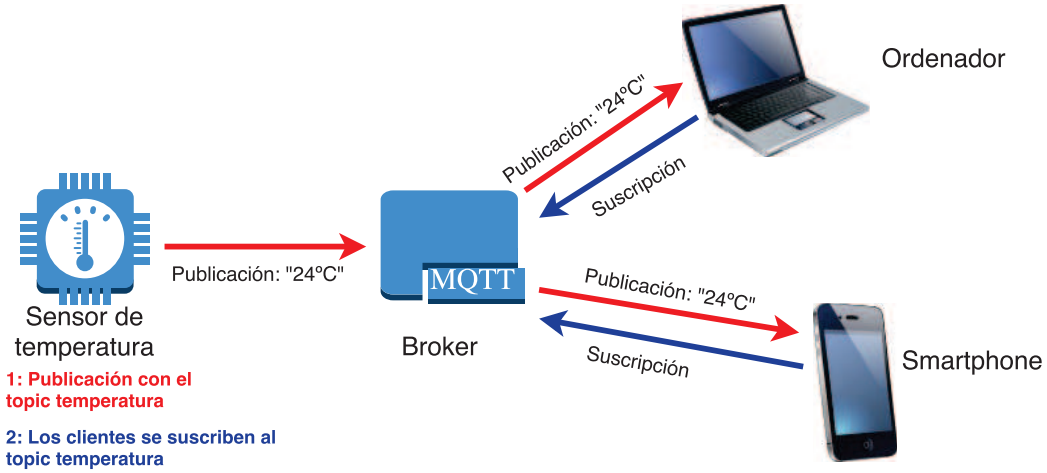


Figura 1.3: Ejemplo protocolo MQTT

Gracias al broker que hace de intermediario entre publicador y suscriptor, se produce un desacoplo entre en el transmisor y receptor. Este desacoplo se produce en varios niveles:

- El nodo que publica un mensaje no necesita conocer nada sobre el nodo destino. Con el protocolo MQTT, el nodo origen sólo necesita conocer la dirección IP del broker y el puerto utilizado.
- No es necesario que el transmisor y el receptor estén conectados a la vez. El broker puede almacenar mensajes para clientes que no están actualmente conectados.
- Transmisor y receptor no necesitan estar sincronizados ni esperando el uno al otro.

Actores principales del protocolo MQTT

Los actores que participan en la red son los clientes MQTT y los servidores o brokers.
Los **clientes** MQTT pueden ser los que recolectan información del medio como es el caso de los sensores y sistemas embebidos o aplicaciones que ejecutan alguna librería MQTT y que de alguna forma interactúan con los datos. Los clientes pueden hacer la función de publicadores y suscriptores de mensajes y además, tienen la posibilidad de controlar y configurar los sensores mediante comandos, si es que son nodos de sensores.

El **broker** es un servicio (software) que establece la comunicación, a nivel de aplicación, entre los diferentes clientes. Como se ha comentado anteriormente, realiza la función de intermediario entre los publicadores y los suscriptores. Además, es el responsable de recibir los mensajes, filtrarlos y encaminarlos a los clientes suscritos según el topic.
Existen varios sotwares disponibles para actuar como broker, estos son RabbitMQ, HiveMQ o Mosquitto. Para la aplicación del toldo vamos a emplear Mosquitto, el cuál se explicará más adelante.

¿Qué son los topics en el protocolo MQTT?

Como se ha explicado en las secciones anteriores, los clientes publican y/o se suscriben topics. Un topic se refiere a un tema o asunto concreto,es un identificador que define el contenido del mensaje. Todo cliente que quiera publicar un mensaje MQTT, es responsable de asignarlo un topic concreto y los clientes subscritos a ese topic, recibirán esos mensajes. Los suscriptores pueden recibir sólo mensajes de un topic concreto o de varios topics si la suscripción se hace a través de algún identificador común a varios topics. Los topics se organizan según una estructura jerárquica en árbol, que se conoce como topic tree, utilizando el carácter / para formar un topic de varios niveles. Además, en los topics de varios niveles se utiliza el topic string que es una cadena de caracteres que identifica el topic final del mensaje. Existen dos tipos de cadenas:

- **Multinivel:** Emplea el carácter # para admitir cualquier nivel mostrado en la cadena.
- **Simple:** Utiliza el carácter + para admitir sólo hasta el nivel de la cadena.

Ejemplo de topics

Imagina una aplicación de Internet de las Cosas para gestionar un huerto de forma inteligente mediante el protocolo MQTT. El huerto está divido en varias partes y cada parte tiene diferentes plantas, árboles y características de riego. El topic raíz es "**huerto**".

Algunos subniveles podrían ser:

- huerto/frutales/perales/temperatura
- huerto/semilleros/humedad

Si se utilizan los caracteres comodín:

- **huerto/frutales/#**
  En este caso, los suscriptores a esta cadena recibirán los mensajes identificados con los topics huerto/frutales y todos los de niveles inferiores, como por ejemplo huerto/frutales/perales.
- **huerto/+**
  Los suscriptores de esta cadena recibirán los mensajes de ese nivel, por ejemplo, mensajes del tipo huerto/semilleros, pero no recibirán los mensajes de niveles inferiores como es huerto/semilleros/humedad.

A continuación, en la Figura 1.4 se muestra un ejemplo:
El Publicador 1 publica mensajes con el topic huerto/frutales/temperatura y el Publicador 2 con el topic huerto/frutales/humedad. El Suscriptor 1 sólo recibe los mensajes del Publicador 2 ya que está suscrito al topic huerto/frutales/humedad, mientras que al Suscriptor 2 le llegan los mensajes de los

dos publicadores ya que al estar suscrito al topic huerto/frutales/#, recibe los mensajes de ese topic y de los niveles inferiores, que en este caso son huerto/frutales/temperatura y huerto/frutales/humedad.

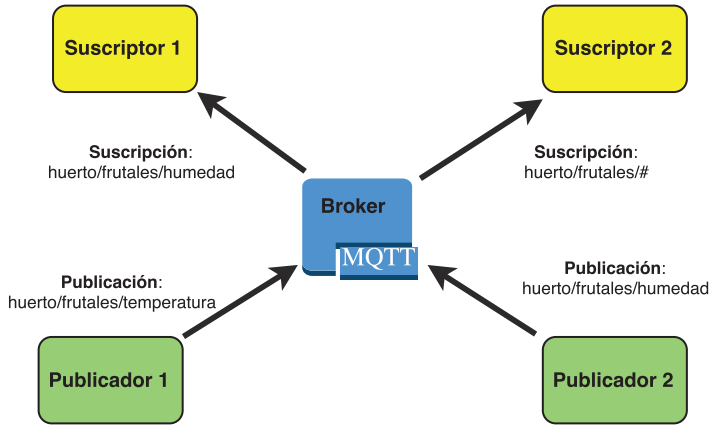


Figura 1.4: Ejemplo topics protocolo MQTT

Garantía de entrega de mensajes

Como se ha comentado anteriormente, MQTT ofrece tres tipos de calidades de servicio para la entrega de mensajes.

Si la calidad de servicio es 0, el mensaje se entrega una vez o no se entrega. El receptor no envía un acuse de recibo aunque haya recibido el mensaje. Este no es almacenado en el receptor y puede perderse si el cliente se desconecta o si falla el servidor.
QoS=0 es la modalidad de transferencia más rápida, pero la menos segura.

Mientras que si la calidad de servicio es 1,el mensaje siempre se entrega como mínimo una vez. Si el emisor no recibe el acuse de recibo del receptor, el mensaje se envía de nuevo o hasta que se reciba un acuse de recibo. Una vez recibido, el mensaje se borra del emisor.

Por último, si la calidad de servicio es 2, el mensaje se entrega siempre una vez. El mensaje se almacena en el emisor y receptor hasta que se procesa.
QoS=2 es la modalidad de transferencia más segura, pero la más lenta. Esto se debe a que se deben de realizar como mínimo dos pares de transmisiones entre emisor y receptor antes de que el mensaje pueda suprimirse del emisor.

1.6. Planificación del proyecto

Al principio del proyecto se elaboró un plan de trabajo para realizar el proyecto, ya que según la normativa debe llevar alrededor de 300 horas. Para ello, se dividió en varias fases y se desarrolló un diagrama de Gantt para estimar el tiempo de cada fase. Además, se ha ido rellenando otro diagrama con las horas reales. En la Figura 1.5 se muestra el inicial, mientras que en la Figura 1.6 se muestra el final. Comparando estos diagramas se puede ver que se ha acabado 2 meses más tarde debido a imprevistos como el pedido de las PCB que tardo el doble de tiempo. Aunque sean 2 meses de retraso ya que han estado las vacaciones de Navidad de por medio, solamente han sido 70 horas más de las estimadas, ya que algunas fases han requerido más tiempo de lo estimado, como por ejemplo, el desarrollo de la aplicación para el móvil que ha durado 20 horas más, esto se debe a que ha sido la

primera aplicación en Android que he desarrollado. También se han tenido problemas con el sensor de luz debido a su tamaño, lo que ha provocado un pérdida de tiempo.
El objetivo de esta planificación es aprender a estimar el tiempo y planificar las fases que tiene cada proyecto.
Las fases en las que se ha divido el proyecto son las siguientes:

- Fase 1: Especificación y planificación
- Fase 2: Búsqueda y selección de los componentes
- Fase 3: Captura esquemática
  - Posicionado de los componentes
  - Creación de los nuevos componentes
  - Rutado de conexiones
  - Documentación del esquemático
  - Análisis del consumo energético
  - Análisis térmico
  - Generación del B.O.M
  - Control de cambios y documentación
- Fase 4: Diseño placa de circuito impreso (PCB)
  - Creación de nuevos componentes
  - Definición del borde de la placa y conectores
  - Posicionado de componentes
  - Rutado de conexiones
  - Definición de planos de disipación térmica
  - Planos de masa y otros
  - Documentación PCB
  - Generación de fichero Gerber
  - Control de cambios y documentación
- Fase 5: Fabricación PCB
  - Envío al fabricante
  - Montaje de los componentes
  - Documentación y control de cambios
- Fase 6: Simulación y desarrollo del software
  - Control de elementos hardware: Drivers
  - Programa principal. Especificaciones
  - Creación aplicación móvil
  - Verificación del software
  - Documentación y control de cambios
- Fase 7: Verificación del hardware

    - Verificación inicial
    - Verificación del software
    - Resolución de problemas
    - Análisis de prestaciones
    - Montaje final
    - Documentación y control de cambios
- Fase 8: Documentación del proyecto
    - Informe y presentación final

Id	Modo de tarea	Nombre de tarea	Duración	Comienzo
1		**Sistema electrónico de control para toldo conectado a Internet**	**312,15 horas**	**lun 26/09/16**
2		Inicio	0 horas	lun 26/09/16
3		**Fase 1: Especificación y planificación**	**5 horas**	**lun 26/09/16**
4		**Fase 2: Búsqueda y selección de los componentes**	**10 horas**	lun 26/09/16
5		**Fase 3: Captura esquemática**	**39,5 horas**	**lun 03/10/16**
6		Posicionado de los componentes	4 horas	lun 03/10/16
7		Creación de los nuevos componentes	12 horas	lun 03/10/16
8		Rutado de conexiones	2,5 horas	mié 05/10/16
9		Documentación del esquemático	10 horas	jue 06/10/16
10		Análisis del consumo energético	4 horas	vie 07/10/16
11		Análisis térmico	4 horas	lun 10/10/16
12		Generación del B.O.M	1 hora	mar 11/10/16
13		Control de cambios y documentación	2 horas	mar 11/10/16
14		**Fase 4: Diseño placa de circuito impreso (PCB)**	**27,65 horas**	**mar 11/10/16**
15		Creación de nuevos componentes	14 horas	mar 11/10/16
16		Definición de borde de la placa y conectores	0,5 horas	vie 14/10/16
17		Posicionado de componentes	4 horas	lun 17/10/16
18		Rutado de conexiones	5 horas	lun 17/10/16
19		Definición de planos de disipación térmica	0,5 horas	mar 18/10/16
20		Planos de masa y otros	0,5 horas	mar 18/10/16
21		Documentación PCB	1 hora	mar 18/10/16
22		Generación de ficheros Gerber	0,15 horas	mar 18/10/16
23		Control de cambios y documentación	2 horas	mar 18/10/16
24		**Fase 5: Fabricación PCB**	**106 horas**	**mié 19/10/16**
25		Envío al fabricante PCB	15 días	mié 19/10/16
26		Montaje de los componentes	14 horas	vie 11/11/16
27		Documentación y control de cambios	2 horas	mar 15/11/16
28		**Fase 6: Simulación y desarrollo software**	**77 horas**	**mié 19/10/16**
29		Control de elementos hardware: Drivers	20 horas	mié 19/10/16
30		Programa principal. Especificaciones	15 horas	lun 24/10/16
31		Creación web/app móvil	40 horas	jue 27/10/16
32		Verificación del software	40 horas	lun 24/10/16
33		Documentación y control de cambios	2 horas	mar 08/11/16
34		**Fase 7: Verificación del harware**	**81 horas**	**mar 15/11/16**
35		Verificación inicial	2 horas	mar 15/11/16
36		Verificación del software	10 horas	mar 15/11/16
37		Resolución de problemas	60 horas	jue 17/11/16
38		Análisis de prestaciones	5 horas	jue 01/12/16
39		Montaje final	2 horas	vie 02/12/16
40		Documentación y control de cambios	2 horas	vie 02/12/16
41		**Fase 8: Documentación del proyecto**	**30 horas**	**mié 07/12/16**
42		Informe final	20 horas	mié 07/12/16
43		Presentación final	10 horas	mar 13/12/16
44		Fin	0 horas	jue 15/12/16

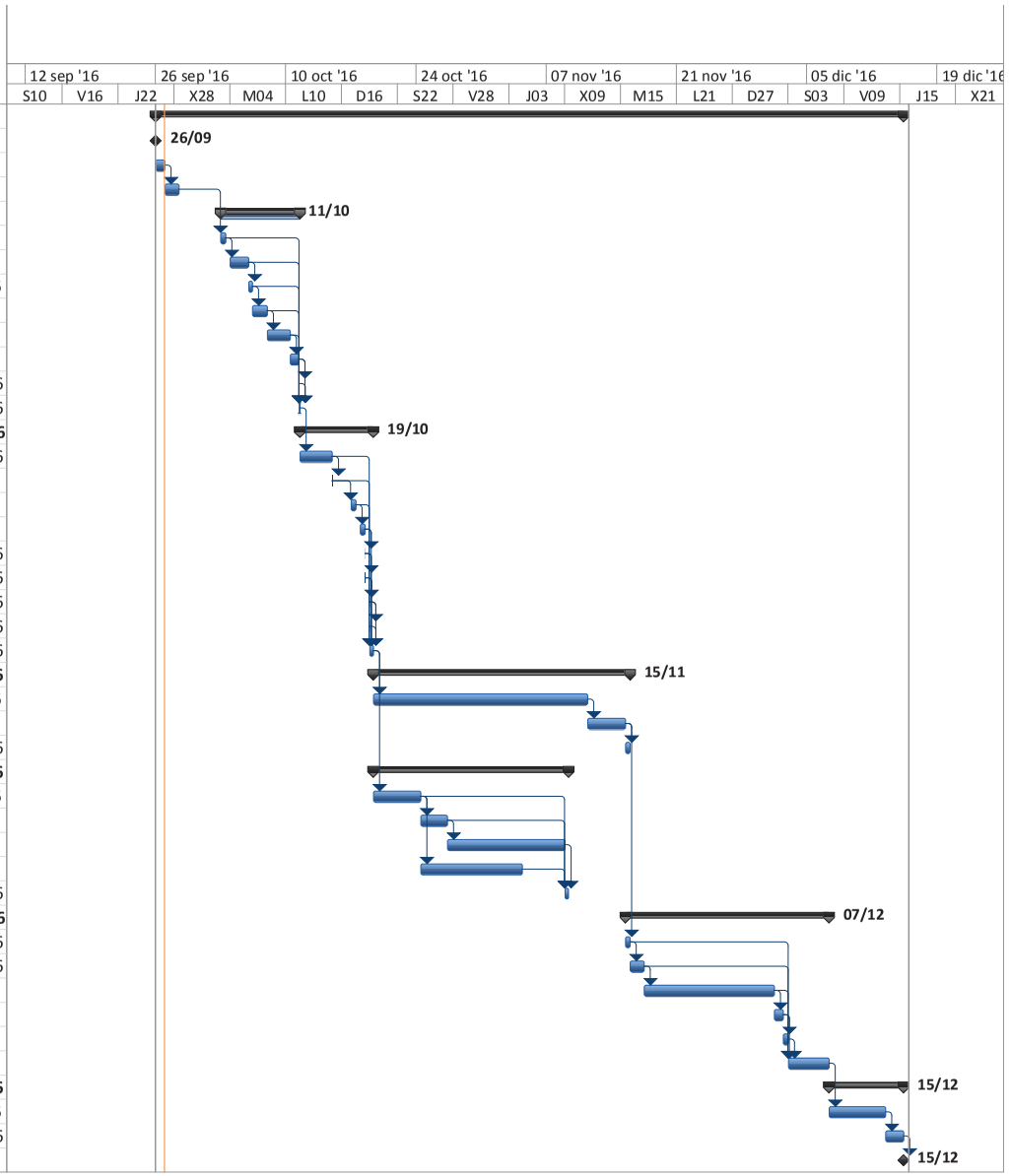


Figura 1.5: Diagrama de Gantt inicial

Id	Modo de tarea	Nombre de tarea	Duración	Comienzo
1		**Sistema electrónico de control para toldo conectado a Internet**	**381,65 horas**	**lun 26/09/16**
2		Inicio	0 horas	lun 26/09/16
3		**Fase 1: Especificación y planificación**	**5 horas**	**lun 26/09/16**
4		**Fase 2: Búsqueda y selección de los componentes**	**18 horas**	lun 26/09/16
5		**Fase 3: Captura esquemática**	**89 horas**	**jue 06/10/16**
6		Posicionado de los componentes	7 horas	jue 06/10/16
7		Creación de los nuevos componentes	12 horas	vie 07/10/16
8		Rutado de conexiones	3 horas	mar 11/10/16
9		Documentación del esquemático	10 horas	mar 11/10/16
10		Análisis del consumo energético	20 horas	vie 14/10/16
11		Análisis térmico	4 horas	mié 26/10/16
12		Generación del B.O.M	1 hora	jue 27/10/16
13		Control de cambios y documentación	2 horas	jue 27/10/16
14		**Fase 4: Diseño placa de circuito impreso (PCB)**	**33,65 horas**	**mar 18/10/16**
15		Creación de nuevos componentes	19 horas	jue 27/10/16
16		Definición de borde de la placa y conectores	0,5 horas	vie 04/11/16
17		Posicionado de componentes	5 horas	vie 04/11/16
18		Rutado de conexiones	5 horas	vie 04/11/16
19		Definición de planos de disipación térmica	0,5 horas	lun 07/11/16
20		Planos de masa y otros	0,5 horas	lun 07/11/16
21		Documentación PCB	1 hora	lun 07/11/16
22		Generación de ficheros Gerber	0,15 horas	mar 08/11/16
23		Control de cambios y documentación	2 horas	mar 08/11/16
24		**Fase 5: Fabricación PCB**	**126 horas**	**mar 08/11/16**
25		Envío al fabricante PCB	19 días	mar 08/11/16
26		Montaje de los componentes	10 horas	mié 07/12/16
27		Documentación y control de cambios	2 horas	lun 12/12/16
28		**Fase 6: Simulación y desarrollo software**	**123 horas**	**vie 02/12/16**
29		Control de elementos hardware: Drivers	12 horas	vie 02/12/16
30		Programa principal. Especificaciones	10 horas	vie 09/12/16
31		Creación web/app móvil	60 horas	lun 09/01/17
32		Verificación del software	40 horas	lun 12/12/16
33		Documentación y control de cambios	1 hora	lun 23/01/17
34		**Fase 7: Verificación del harware**	**89 horas**	**lun 12/12/16**
35		Verificación inicial	3 horas	lun 12/12/16
36		Verificación del software	25 horas	lun 23/01/17
37		Resolución de problemas	50 horas	lun 19/12/16
38		Análisis de prestaciones	5 horas	vie 30/12/16
39		Montaje final	5 horas	mié 25/01/17
40		Documentación y control de cambios	1 hora	vie 27/01/17
41		**Fase 8: Documentación del proyecto**	**60 horas**	**vie 27/01/17**
42		Informe final	40 horas	vie 27/01/17
43		Presentación final	20 horas	mar 07/02/17
44		Fin	0 horas	vie 10/02/17

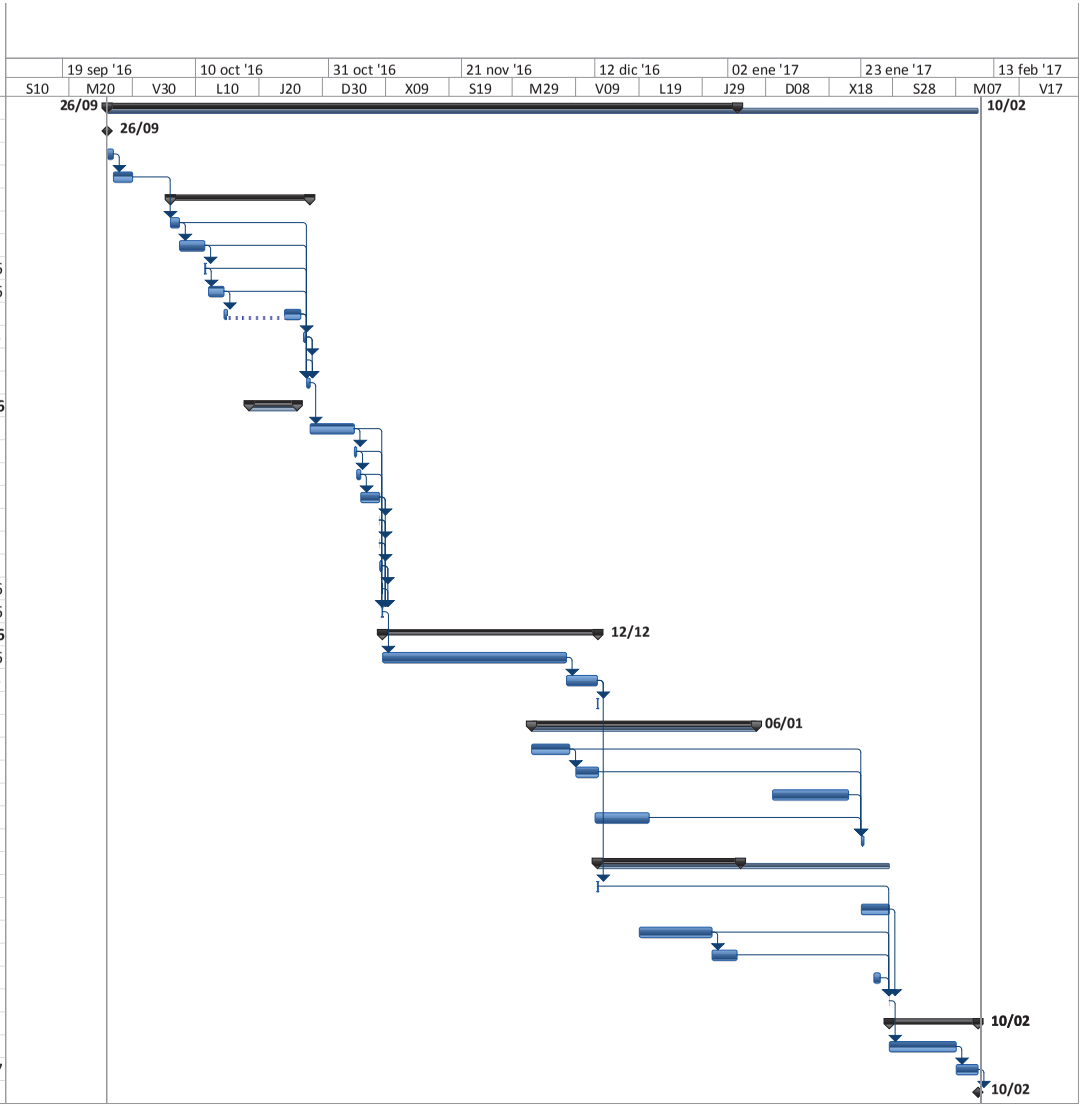


Figura 1.6: Diagrama de Gantt real

Capítulo 2

Hardware

Durante este apartado se va explicar cada una de las partes del hardware que forman nuestro dispositivo. Entre ellas, está el módulo Wi-Fi, la fuente de alimentación, los Triac, el programador, los sensores, etc. Además, se comentará sobre el proceso de fabricación desde el diseño de la PCB hasta el montaje final.
Al final se han diseñado tres PCBs pequeñas dentro de una grande:
La placa principal va dentro de una caja y es la encargada de controlar la posición del toldo. Esta lleva incorporada el módulo Wi-Fi, la fuente de alimentación y los Triac.
Otra, es exterior ya que tiene incorporados los sensores. Irá colocada en el toldo. Para conectarla a la principal se necesitan seis cables:

- Alimentación de 3,3 voltios de los sensores.
- Alimentación de 5 voltios del anemómetro.
- GND.
- Señal de datos de I2C.
- Señal de reloj de I2C.
- Datos para el convertidor analógico-digital, ya que el anemómetro es analógico.

Esta placa está diseñada con los sensores en la misma cara, pero en futuros diseños se pondrá en una cara el sensor de luz y en otra el de humedad y temperatura. Esto se debe a que es preferible que al sensor de humedad y temperatura no le de el sol para que no mida más temperatura de la del aire. Para proteger los sensores de las condiciones climatológicas se ha pensado en colocar una goma líquida transparente de la empresa Plasti-Dip[1]. Este producto es un spray que se aplica sobre la placa y cuando se seca, queda como una goma por encima de los sensores. Esta goma se puede quitar fácilmente cuando se desee. En la Figura 2.1, se muestra una foto del spray.

[1]Se puede adquirir:http://www.plasti-dip.es/plastidip-goma-protectora-spray-incoloro-mate-p-7985.html

Figura 2.1: Spray Plasti-Dip

La última placa, esta formada por el programador. Este en el diseño original estaba en la placa principal. Sin embargo, se decidió hacer aparte para ahorrar espacio en la placa principal y poder hacerla más pequeña para meterla en la caja.
La caja utilizada se muestra en la Figura 2.2, en un compartimiento va situada la placa principal y en el otro los interruptores con su mecanismo. Al principio, se quería meter todo en una caja de un solo compartimiento pero debido a la altura de los componentes y del mecanismo de los interruptores fue imposible.

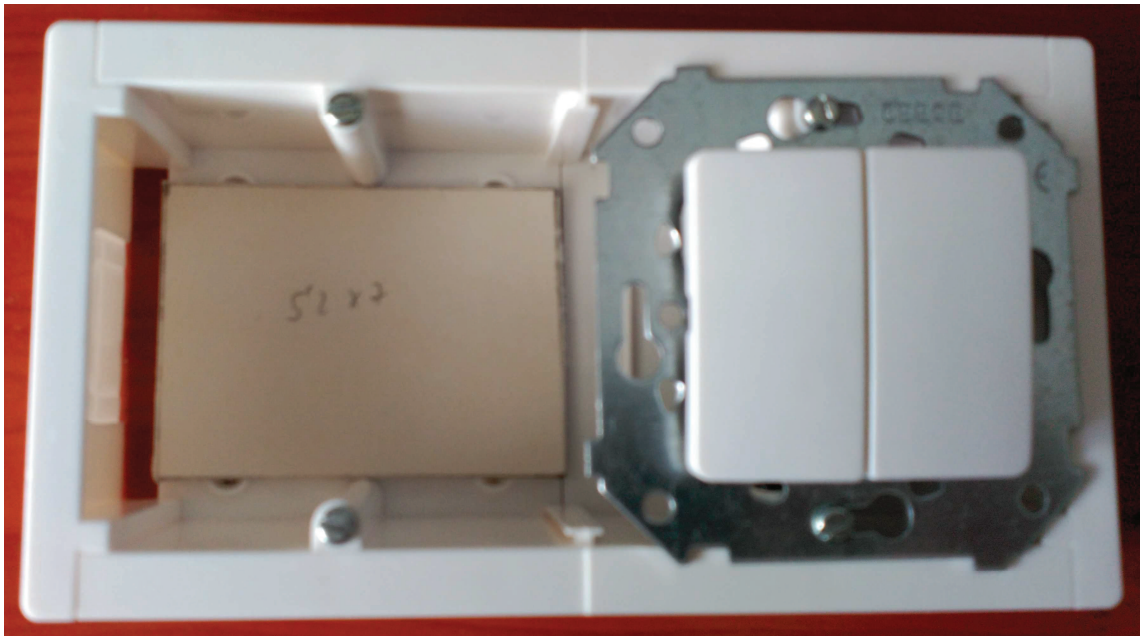


Figura 2.2: Caja utilizada

Por otro lado, el esquemático completo se puede ver en el Apéndice A, en él se incluye un análisis térmico y eléctrico del dispositivo, mientras que en el Apéndice B se muestra el diseño de la PCB. Estos esquemáticos se han realizado con Proteus Isis y Proteus Ares respectivamente.
Para realizar el esquemático se han tenido que crear algunos componentes como es el caso del módulo Wi-Fi, la fuente de alimentación, los sensores y los conectores. Para hacer un esquemático completo en cada componente se indica el nombre y la huella que tiene. Además, se especifica el código de Farnell y el precio para poder generar el listado de materiales (Apéndice C).
Todo esto se completa con un análisis térmico y eléctrico del dispositivo.
Para calcular el consumo eléctrico hay que sumar el consumo de todas las fuentes, en este caso, de las fuentes de 3,3 y 5 voltios. A la fuente de 5 voltios sólo está conectado el anemómetro que consume

3 mA. El FT232RL también está conectado a la fuente de 5 voltios pero es a la del PC por lo que no consume en nuestro dispositivo. Mientras que a la fuente de 3,3 voltios están conectados los leds, los sensores de luz y humedad-temperatura y el módulo Wi-Fi. Sumando el consumo de todos estos componentes, el consumo total es 209,45 mA.
Respecto al análisis térmico, apenas se ha podido calcular ya que se desconoce la resistencia térmica de muchos componentes. Sólo se conoce la resistencia térmica del regulador, del FT232RL y de los triac. El componente que más se calienta es el regulador que lo hace en 8,8 ^{o}C por encima de la temperatura ambiente.
Estos análisis se pueden ver con más detalle en el esquemático completo.

2.1. Módulo ESP8266-12F

Como ya se ha comentado anteriormente, el módulo Wi-Fi tiene un microcontrolador interno que se puede programar. Debido a esto se escogió el modelo ESP8266-12F porque tiene más pines disponibles en la parte exterior ya que todos los módulos de esta familia tienen el mismo número de pines y se diferencian en que unos modelos sacan más pines al exterior que otros, además de las mejoras que incluyan respecto a funcionalidad. En la Figura 2.3, se puede apreciar el conexionado interno del módulo ESP8266 que es el mismo para todos los modelos de la familia. En la Figura 2.4, se muestra número de pines disponibles por fuera que tiene el modelo elegido para este proyecto.

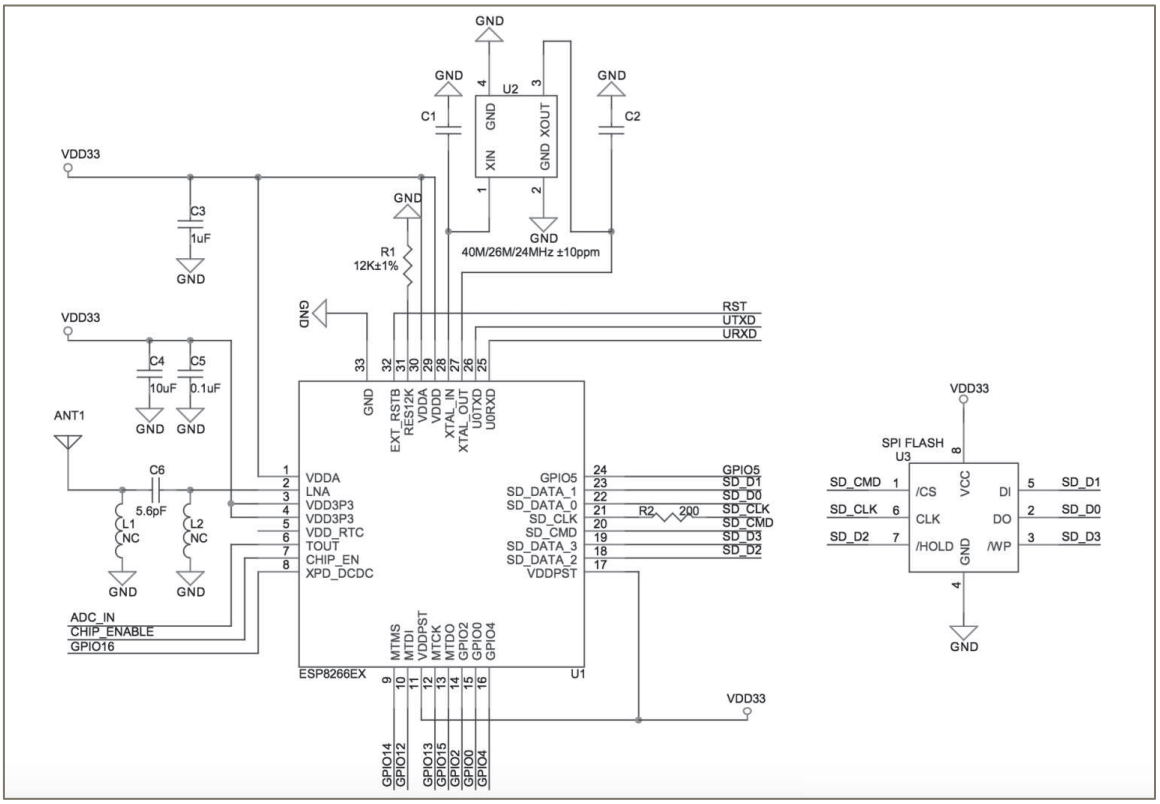


Figura 2.3: Esquema interno del módulo ESP8266-12F

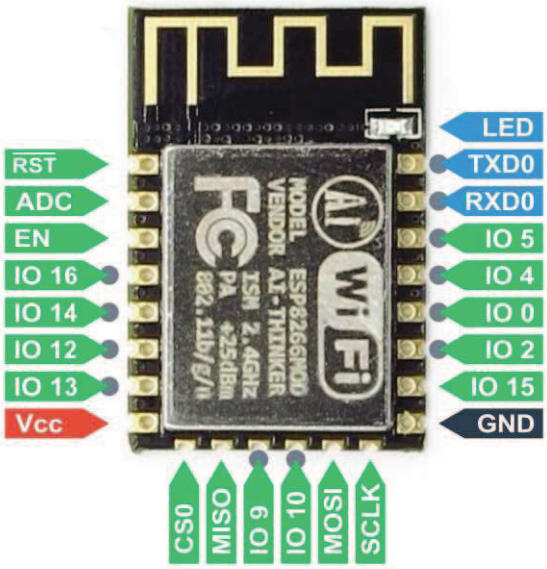


Figura 2.4: Módulo ESP8266-12F

A continuación, vamos a comentar las características del módulo:

- Tensión de alimentación 3,3 voltios.
- Tiene integrado un microcontrolador de bajo consumo de 32 bits.
- También, tiene un convertidor analógico-digital de 10 bits.
- Cuenta con 4 pines para la comunicación SPI: MOSI, MISO, CLK Y $\overline{CS}$.
- Además, posibilita la comunicación I2C mediante los pines GPIO14 (SCL) y GPIO2 (SDA). Estos pines también se pueden utilizar como de propósito general.
- Posee 16 pines de propósito general. Sin embargo, en el modelo 12F sólo hay 9 disponibles.
- Incorpora una UART.
- Cumple con el estándar Wi-Fi (IEEE 802.11), por lo que trabaja en la banda de frecuencias de 2,4 GHz.
- Lleva incorporada una antena Wi-Fi.
- Funciona a 40 MHz, 80 MHz y 160 MHz.
- Rango de temperatura de funcionamiento: -40 ºC a 125 ºC.
- Como se puede ver en la Figura 2.2, el chip del módulo lleva una memoria SPI integrada.

2.2. Fuente de alimentación

El dispositivo va conectado a un enchufe que proporciona 220 voltios de corriente alterna y mediante una fuente de alimentación y reguladores se consiguen 3,3 voltios. Esto se debe a que el motor necesita estar alimentado a 220 voltios de corriente alterna y que el módulo Wi-Fi y los sensores necesitan 3,3 voltios, para conseguir esta reducción de tensión hace falta una fuente de alimentación para convertir y reducir la tensión.
Al principio del proyecto se decidió diseñar una fuente de alimentación lineal con un fusible, un

transformador de 220 VAC a 6 voltios y un rectificador de diodos. Esta fuente de alimentación se muestra en la Figura 2.5.

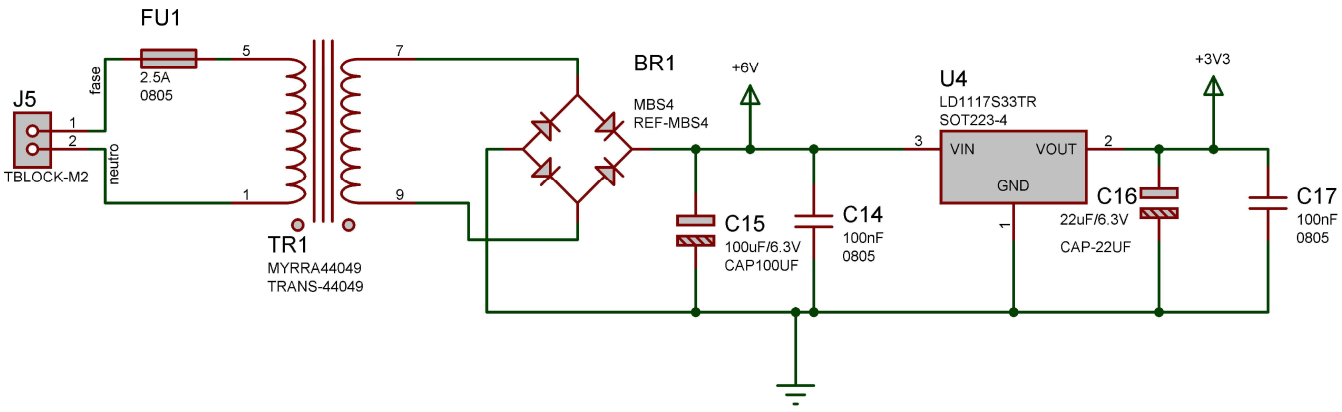


Figura 2.5: Diseño en Proteus de la fuente de alimentación lineal

Sin embargo, debido al requisito de que entrase en la caja, se optó por una fuente de alimentación ya fabricada y que proporciona 5 voltios a la salida. De esta manera, se ahorra espacio en la PCB ya que la fuente de alimentación tiene dimensiones parecidas al transformador, y se ahorra altura puesto que el transformador entra justo en la caja. A la hora de elegir la fuente de alimentación se propusieron dos modelos:

- Vigortronix VTX-214-005-405 que tiene dos salidas que proporcionan 5 voltios cada una. Incluye fusible y está aislada. Se puede adquirir en Farnell y su código es 2627168.
- Convertidor AC-DC que tiene una salida que proporciona 5 voltios. Es un módulo para Arduino. No incluye fusible y no está aislada. Pero es más barata y ocupa menos espacio que la anterior. Se adquiere en Itead y su nombre completo es AD-DC Converter Voltage Regulator Switching Power Supply Module 5V 500mA V3 for Arduino.

En un principio, se ha elegido la primera. Pero, se han pedido las dos ya que la primera tenía un periodo de entrega demasiado largo. Para evitar problemas, se ha realizado una doble huella para el diseño de la PCB. Como las placas han llegado antes que la primera fuente de alimentación, se ha optado por poner la segunda.Las principales características de esta fuente son:

- Tensión de entrada: 90-264 VAC.
- Tensión de salida: 5 voltios.
- Corriente de salida: 500 mA.
- Frecuencia 50/60 Hz.

Una vez que la fuente proporciona 5 voltios rectificados, hay que reducir la tensión a 3,3 voltios para poder alimentar el módulo Wi-Fi y los sensores. Para ello, se ha empleado el regulador LD1117S33TR cuyas características son:

- Tensión de entrada: 4,75-15 voltios.
- Tensión de salida: 3,3 voltios.
- Tensión de dropout: 1,1 voltios.

- Corriente de salida: 800 mA.
- Rango de temperatura de trabajo: 0 a 125 ºC.

Como se ha podido comprobar las limitaciones a la hora de escoger la fuente de alimentación han sido mecánicas ya que hay una limitación de altura y de espacio en la PCB.

2.3. Triac y optotriac

Para que se pueda activar el motor de 220 VAC desde el módulo Wi-Fi hace falta utilizar un réle o un triac. Para nuestro dispositivo, hemos preferido utilizar el triac porque ocupa menos espacio. Además, necesitamos dos triac: uno para cada sentido de giro del motor. A su vez, hace falta un optotriac para controlar cada uno de los triac.
Antes de explicar como funciona un Triac, vamos a estudiar brevemente que es un SCR(*Silicon Controlled Rectifier, Rectificador de Silicio Controlado*) y como funciona, puesto que el Triac tiene un comportamiento parecido.

2.3.1. SCR

Un SCR es un componente electrónico de conmutación y tiene tres terminales: ánodo (A), cátodo (K) y puerta (G). Es un dispositivo rectificador unidireccional, es decir, deja circular la corriente eléctrica en un único sentido: desde el ánodo hasta el cátodo como un diodo. Es controlado a través de G. Su símbolo eléctrico se muestra en la Figura 2.6.

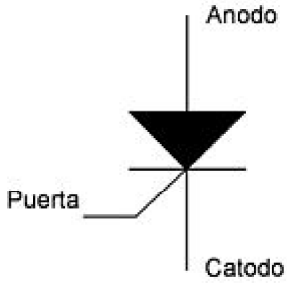


Figura 2.6: Símbolo SCR

El SCR se emplea como dispositivo de conmutación en corriente continua.

Funcionamiento

El funcionamiento de un SCR se puede entender como un circuito implementado por dos transistores bipolares de unión (BJTs), esto se muestra en la Figura 2.7.

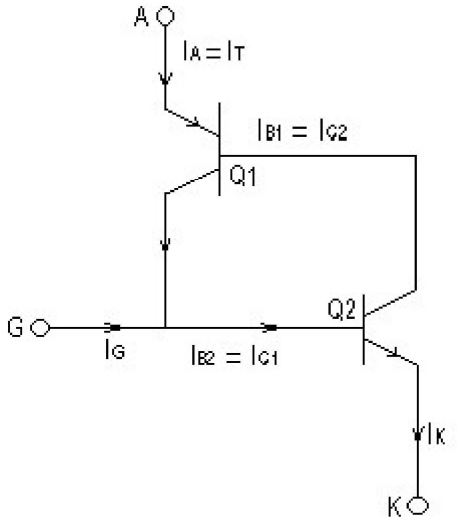


Figura 2.7: Circuito equivalente SCR

Por lo que se debe tener en cuenta las siguientes características de los BJTs:

- Cuando se le aplica una tensión entre base y emisor mayor que el voltaje umbral (se entiende que esta tensión es en valor absoluto, ya que si el transistor es PNP, el voltaje umbral sería negativo), el transistor conmuta de OFF a ON, es decir, comienza a conducir.
- Amplifica la corriente de base en un factor dado por el parámetro β. Por lo que la corriente de colector es la corriente de base por dicho parámetro

Utilizando el circuito equivalente, vamos a explicar el funcionamiento con un ejemplo.
Suponemos que inicialmente solo se aplica un voltaje $V_{AK}>0$ voltios:

- Hasta que no se aplique una señal en la puerta no hay conducción en el SCR, por lo que $I_A=0$ amperios, el SCR está abierto.
- Cuando se aplica un voltaje en la puerta, V_{GK}, suficientemente positivo ($V_{BE}>V_T$), la corriente de puerta I_G, el transistor Q2 comienza funcionar ya que $I_G=I_{B2}=I_{C1}$. Como hemos comentado antes en las propiedades de los BJTs, si hay una tensión de base, la de colector es la corriente de base por un factor β, por lo que se genera una corriente I_{C2} y una corriente I_{B1} (porque son iguales). Debido a que hay una corriente de base en Q1, este transistor comienza a conducir. De esta manera, comienza la conducción desde el ánodo hasta el cátodo.
- Mientras haya un voltaje V_{AK} suficientemente positivo, seguirá existiendo corriente I_A, aunque no haya pulso en la puerta, G, ya que la corriente de colector de Q1 alimenta la base de Q2. Esto provoca que Q2 siga conduciendo y a su vez, esto hace que Q1 siga conduciendo. La única forma de que deje de funcionar es disminuyendo la corriente I_{AK} por debajo de una corriente de mantenimiento que se especifica en el datasheet del componente o desconectando el SCR de la tensión.

2.3.2. Triac

El triac se puede ver como dos SCR conectados en anti-paralelo pero con una sola puerta (Figura 2.8). Debido a esto su funcionamiento está basado en el SCR.

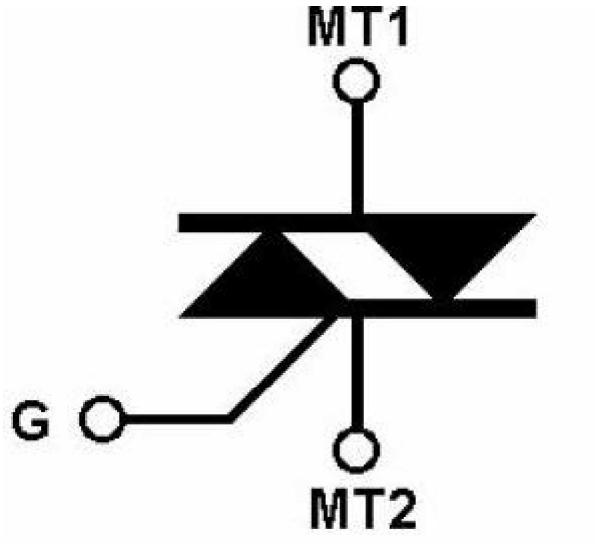


Figura 2.8: Símbolo triac

Al igual que el SCR, el triac se utiliza como dispositivo de conmutación, pero en corriente alterna debido a que la corriente puede ir en los dos sentidos de MT1 a MT2 y de MT2 a MT1.

Funcionamiento

Su funcionamiento se basa en el del SCR pero en corriente alterna. El triac también se dispara por la puerta, G.
Para explicar el funcionamiento, vamos a suponer que en el terminal MT1 hay una corriente alterna y en MT2 hay 0 voltios:

- Si hay una tensión de disparo en la puerta (I_G>0) y en MT1 está la parte positiva de la señal de alterna, el triac conducirá la corriente de MT1 a MT2.
- Mientras que si hay una tensión de disparo en la puerta (I_G>0) y en MT1 está la parte negativa de la señal de alterna, el triac conducirá la corriente de MT2 a MT1.

Al igual que el SCR una vez que comienza a conducir, la única manera de desactivarlo es desconectandolo o reduciendo la corriente entre terminales por debajo de la corriente de mantenimiento especificada en el datasheet.

El triac escogido para nuestro dispositivo es el BTA08, cuyas características son las siguientes:

- Proporciona una tensión máxima de 600 voltios.
- Corriente de disparo de la puerta: 10 mA.
- Máxima tensión de puerta para no estropear el componenete: 1,5 voltios.
- Máxima corriente de puerta que soporta para no estropease: 4 amperios. Si se aplica una corriente de puerta mayor, el componente se romperá.
- Rango de temperatura de trabajo: -40 a 150 ºC.
- No especifica la corriente de mantenimiento.

2.3.3. Optotriac

Como se puede observar en la Figura 2.9, el optotriac es un dispositivo electrónico que está formado por una fuente emisora de luz (un led) y un fotoreceptor que en este caso es un triac.

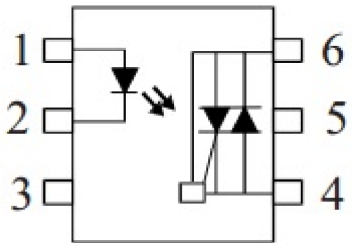


Figura 2.9: Esquema interno optotriac

Cuando le llega una señal eléctrica a los extremos del led, este emite una señal luminosa, que recibe el receptor. Este al recibir la señal luminosa, genera en sus bornes una tensión eléctrica que será la señal de salida.
Existen dos tipos de optotriac: el normal y el de paso por cero. Se diferencian en que el optotriac de paso por cero, activa el triac del receptor sólo en los cruces por cero de la tensión alterna. Por lo que independientemente de cuando se active esta señal, el triac no sacará la tensión de salida hasta que la señal de corriente alterna pase por cero. Con este mecanismo se buscar evitar transitorios y ruidos eléctricos.

Una de sus aplicaciones es aislar dos circuitos, uno que trabaja a poca tensión y otro a mucha tensión. Debido a esto, se ha empleado un optotriac para cada triac en el proyecto, para separar el circuito del módulo Wi-Fi (baja tensión) de los triac (alta tensión).

El optotriac escogido para nuestro dispositivo es el MOC3063XSM y sus características son:

- Tiene mecanismo de cruce por cero.
- Voltaje de funcionamiento del led: 1,2 voltios
- Corriente de funcionamiento del led: 50 mili-amperios
- Tensión de salida del triac: 600 voltios.
- Máxima corriente de funcionamiento del triac: 1amperio.
- Rango de temperatura de trabajo: -55 a 150 ºC.

2.4. Motor

Respecto al motor, el factor más importante a la hora de elegir uno, es la potencia. Buscando los motores AC en Farnell, hemos encontrado uno de 400 vátios para el cuál se ha diseñado nuestro dispositivo. Se escogido este porque el resto de motores no supera los 100 vátios y como mínimo un motor para un toldo tiene que tener 140 vátios.
Para desarrollar el proyecto no se ha comprado el motor ya que su precio era de 206 euros más 18 euros de costes de envío. Para simular el motor se han utilizado dos bombillas para indicar cada uno de los sentidos de giro del motor: una para indicar la subida del toldo y otro para la bajada del toldo. El conexionado de las bombillas se puede ver en la Figura 2.10. En la placa tenemos un **bloque**

terminal de bornas con dos salidas: una para la señal de subida y otra para la señal de bajada, y otro **bloque terminal de bornas** de dos salidas: una para la fase y otro para el neutro de la alimentación. Como los **bloques terminales de bornas** son algo pequeños se ha utilizado una regleta con tres espacios para empalmes. En esta se ha empalmado el cable de la señal de subida que sale de la placa y un cable de una bombilla. Análogamente, se ha hecho lo mismo para la señal de bajada y la otra bombilla. En el otro espacio disponible se ha conectado el cable de fase de la señal eléctrica y los dos cables de alimentación de las bombillas. Respecto al neutro, sólo se ha conectado la PCB al neutro de la señal eléctrica, ya que el motor está conectado al neutro a través de la PCB.

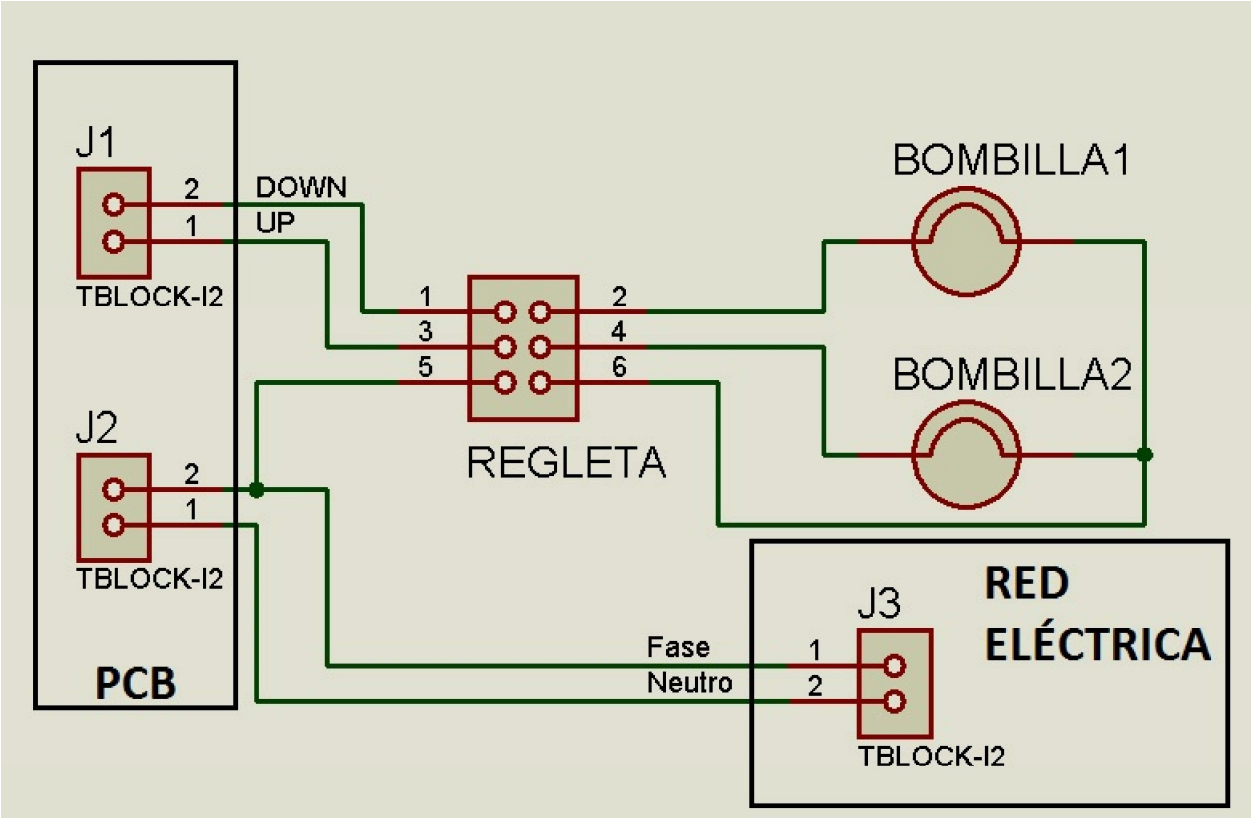


Figura 2.10: Conexionado de las bombillas y la PCB

2.5. Sensores

2.5.1. Sensor de temperatura y humedad

Uno de los objetivos del proyecto, es que nuestro dispositivo contará con un sensor de temperatura y otro de humedad. Estudiando los sensores que hay disponibles en el mercado, se ha decido utilizar el sensor SHT31 (Figura 2.11), que es un sensor de humedad y temperatura. De esta manera, se puede hacer la PCB de los sensores más pequeña y es más barato que comprar por separado un sensor para cada cosa. Las características del sensor elegido son las siguientes:

- Tensión de alimentación: 2,4 a 5,5 voltios.
- Salida de tipo digital.
- Interfaz I2C para la comunicación con el módulo Wi-Fi.
- Rango de medidas: -40 a 125 ºC y 0 a 100 %RH.
- Resolución: 0,05 ºC y 0,05 %RH.

- Rango de temperatura de operación: -40 a 125 ºC.

Figura 2.11: Sensor SHT31

En la Figura 2.12, se puede ver el diagrama de bloques interno del sensor. En esta imagen, podemos ver como los sensores dan mediciones analógicas que son pasadas al convertidor analógico-digital. Una vez convertidas son enviadas al módulo Wi-Fi a través del interfaz I2C. También, se puede apreciar que tiene un sistema de reset para casos de bloqueo.

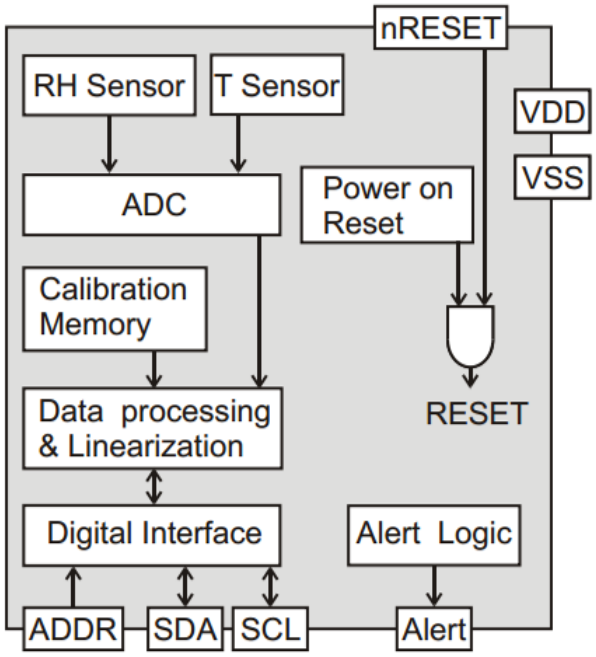


Figura 2.12: Diagrama de bloques interno sensor SHT31

2.5.2. Sensor de luz

El sensor de luz elegido para nuestro diseño es el VEML6030 (Figura 2.13). Sus principales características son:

- Tensión de alimentación: 2,6 a 3,3 voltios.
- Salida de tipo digital.
- Interfaz I2C para la comunicación con el módulo Wi-Fi.
- Resolución: 0,0036 lux.
- Rango de medidas: 0 a 120.000 lux.

- Rango de temperatura de operación: -25 a 85 ºC.

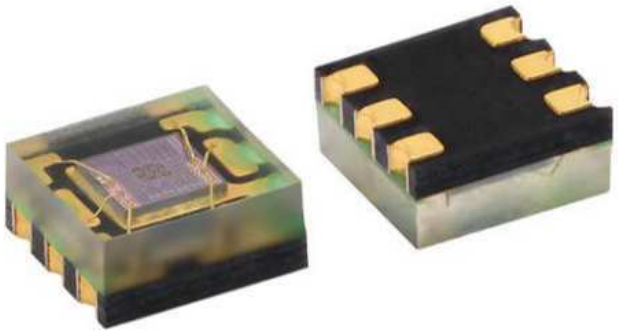


Figura 2.13: Sensor VEML6030

Su inconveniente es el tamaño ya que es demasiado pequeño y en la facultad no disponemos del material necesario para soldarlo. Después de intentar dos sensores de este tipo y comprobar que era imposible, se decidió pedir uno más grande para terminar nuestro dispositivo. Este nuevo sensor es el BH1750FVI (Figura 2.14), viene soldado en una placa y con cinco pines disponibles para su soldadura: VCC, GND, SDA, SCL y ADD. Se ha elegido este sensor porque su soldadura es fácil ya que sólo hay que soldar cinco cables. Las características de este sensor son:

- Tensión de alimentación: 2,6 a 3,3 voltios.
- Salida de tipo digital.
- Interfaz I2C para la comunicación con el módulo Wi-Fi.
- Resolución: desde 0,5 lux a 4 lux, según el modo de medida.
- Rango de medidas: 0 a 65.535 lux.
- Rango de temperatura de operación: -40 a 100 ºC.

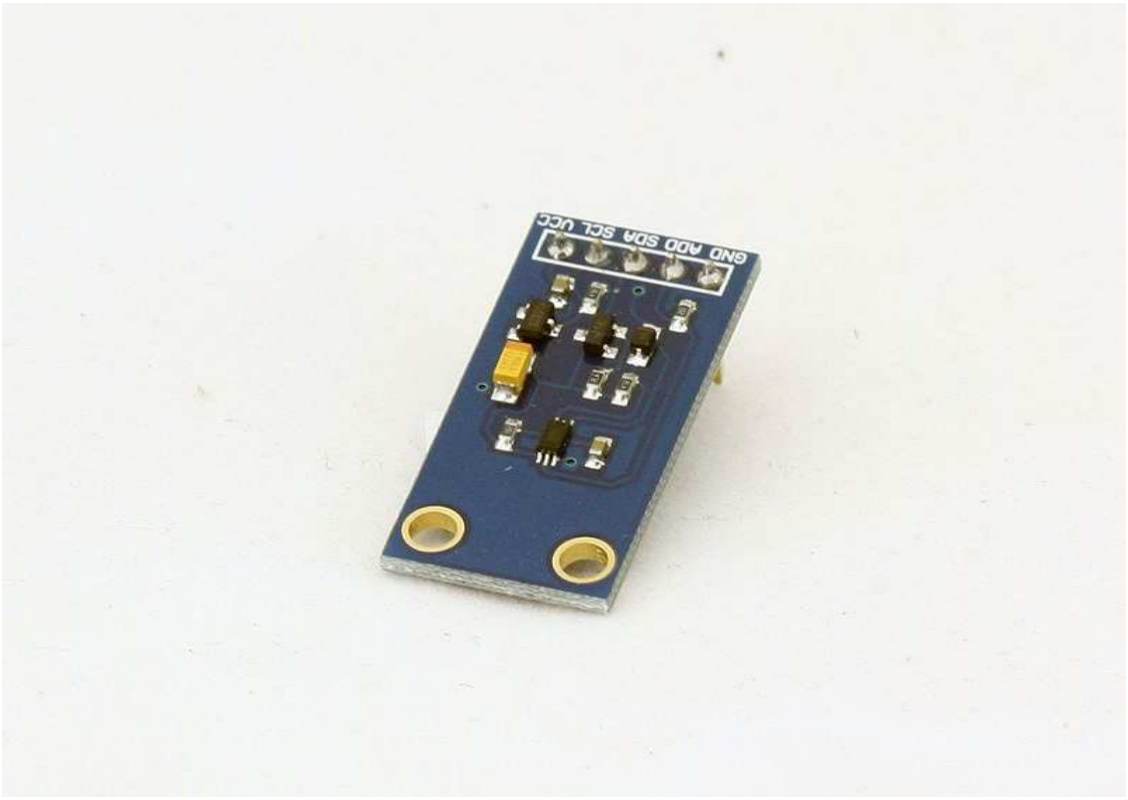


Figura 2.14: Sensor BH1750FVI

2.5.3. Anemómetro

En las especificaciones del proyecto no se pedía un anemómetro Sin embargo, en la fase de planificación pensamos que era necesario uno ya que un día de mucho viento se puede arrancar el toldo. Buscando las opciones, sólo se ha encontrado uno[2] que se pueda conectar a nuestra PCB ya que el resto ya vienen preparados con más sensores y van conectados directamente al motor. Sus características son las siguientes:

- Tensión de alimentación: 7 a 24 voltios.
- Cuenta con 3 hilos: VCC, GND y datos.
- Salida de tipo analógica.
- Rango de medidas: 0 a 180 km/h.
- Resolución: 0,1 m/s.
- Rango de salida: 0 a 2 voltios.

Como se puede ver, el diseño los hemos hecho para un anemómetro que este alimentado con 5 voltios y este necesita 7. Esto se debe a que hemos supuesto otro anemómetro con estas características pero con una tensión de alimentación menor. Para la realización de este proyecto, no se ha comprado el anemómetro debido a que vale 85 €. Para simularlo, se ha utilizado un potenciómetro. Si en un futuro se quiere comercializar el dispositivo que hemos desarrollado, habría que buscar un anemómetro que se alimente con 5 voltios o cambiar la fuente de alimentación por la que diseñamos pero con un transformador que proporcione unos 8 voltios en el secundario ya que el puente de diodos resta entre

[2]Se puede encontrar aquí: http://www.coldfire-electronica.com/esp/item/100/42/anemometro-sensor-de-velocidad-del-viento-wanalog-voltage-output

0,7 y 1 voltio. Si se elige la opción de poner otro anemómetro, se tendría que contactar con empresas de toldos para que nos proporcionen los datos de los anemómetros que emplean. Mientras que si elegimos la segunda opción, se puede hacer un poco más grande la PCB ya que al realizar el montaje hemos comprobado que la placa entra holgadamente en la caja y de altura aunque vaya justo, se puede meter ya que sólo hay que colocar correctamente el transformador para que no coincida con las pestañas de la tapadera.

2.5.4. Protocolo I2C

Debido a que los sensores son digitales y se comunican con el módulo Wi-Fi a través de una interfaz I2C. Vamos a explicar brevemente en que consiste.
Fue desarrollado por Philips en los años 80. Es un protocolo de bus multipunto bidireccional que emplea dos hilos referidos a masa:

- SDA: Señal de datos.
- SCL: Señal de reloj.

Se trata de una comunicación serie síncrona y half-duplex (no puede transmitir y recibir a la vez). Además, es una comunicación maesto-esclavo en la que puede haber varios maestros.
Los dispositivos se conectan al bus con driver de tipo colector/drenador abierto (conexionado tipo AND: el 0 vence al 1). Para que aparezca un 1 lógico se necesitan resistencias de pull-up externas. En la Figura 2.15 se muestra el conexionado de varios dispositivos al bus. Mientras que en la Figura 2.16 se puede ver lo que es un dispositivo con driver de colector/drenador abierto.

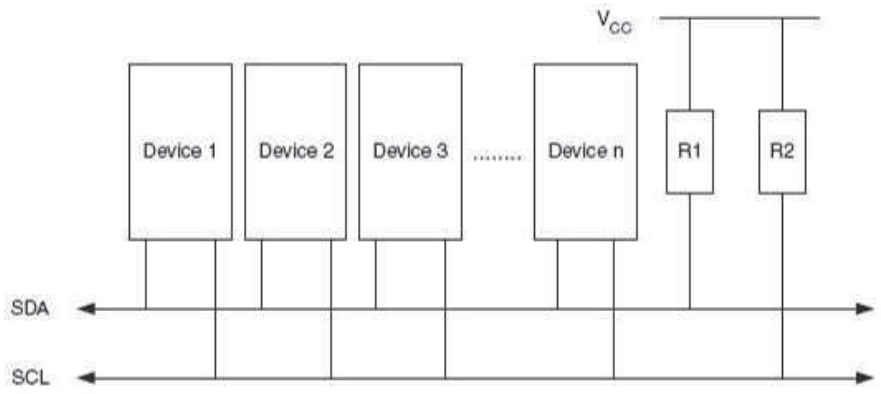


Figura 2.15: Conexión I2C

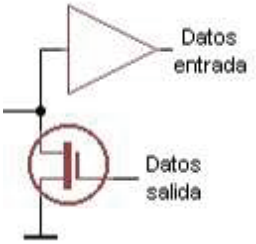


Figura 2.16: Dispositivo de drenador abierto

El protocolo tiene tres modos de funcionamiento:

- Modo estándar: alcanza velocidades de 100 kbps y la máxima capacidad del bus es de 400 pF.
- Modo rápido: alcanza velocidades de 400 kbps y la máxima capacidad del bus es de 400 pF.
- Modo alta velocidad: en este modo se pueden distinguir dos casos en uno alcanza velocidades de 1700 kbps y la capacidad del bus es de 400 pF, mientras que en el otro alcanza velocidades de 3400 kbps y la máxima capacidad del bus es 100 pF.

Comunicación I2C

Solamente los maestros pueden iniciar la comunicación. La condición inicial, de bus libre, es cuando las señales SDA y SCL están en alto. En este estado cualquier maestro puede establecer la condición de **start**. Esta condición se presenta cuando un dispositivo maestro pone en estado bajo la línea de datos (SDA) y deja en alto la línea de reloj (SCL). Esta condición se puede ver en la Figura 2.17.

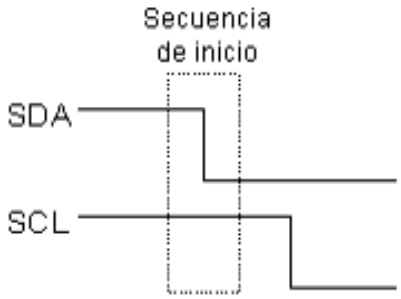


Figura 2.17: Condición de start

Después de la condición de start, se envía un byte con la dirección de 7 bits del dispositivo con el que se desea comunicar y un bit que indica escritura o lectura.
Si el esclavo está disponible contesta con un bit de ACK que es un 0 lógico. En la Figura 2.18, se muestra un ejemplo de esta trama.

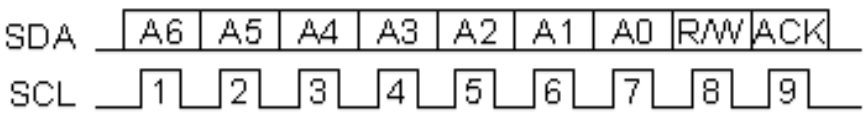


Figura 2.18: Inicio de trama

Si el bit de lectura/escritura fue puesto a 0 (escritura), el dispositivo maestro envía datos al esclavo. Por cada byte que envía recibe un ACK de el esclavo.
Si el bit de lectura/escritura fue puesto a 1 (lectura), el esclavo envía los datos y el maestro envía un ACK por cada byte que recibe.
Una vez enviados los datos tanto en un sentido como en el otro, el maestro genera una señal de **stop**. Para ello, cambia la línea de datos de bajo a alto mientras el reloj está en alto, esto se muestra en la Figura 2.19.

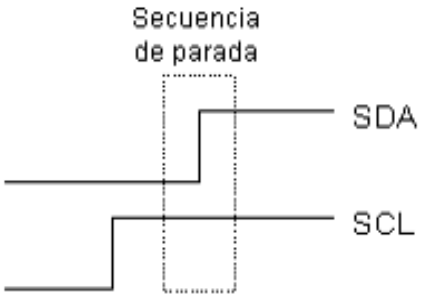


Figura 2.19: Condición de stop

2.6. Programador

Para programar el módulo Wi-Fi hace falta conectarlo al ordenador a través de un puerto USB. Debido a esto, hace falta que nuestra PCB tenga un conector mini USB (Figura 2.20) y un convertidor de niveles USB a TTL (Figura 2.21).

Figura 2.20: Conector mini USB Molex 67503

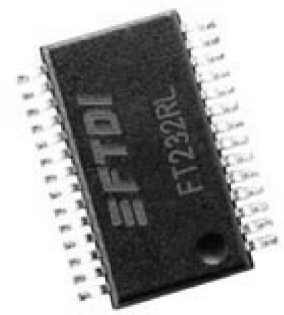


Figura 2.21: FT232RL

El convertidor de niveles que hemos escogido es el FT232R. Su función es convertir la señal USB a TTL para que el módulo Wi-Fi pueda interpretarla.
Se conecta con el ordenador a través de las líneas de datos D+ y D- del conector mini USB. Para comunicarse con el módulo lo hace a través de la UART de éste (TXD y RXD). Además, utiliza los pines _DTR (*Data Terminal Ready Control Output, Terminal de Salida de Datos para Control*) y _RTS(*Request to Send Control Output, Petición para enviar*) para conectarlos al bootloader y reset del módulo Wi-Fi.
Debido a que el puerto USB de el ordenador proporciona 5 voltios, el FT232R está alimentado a 5 voltios y las señales a su salida son de 5 voltios en alto. Por lo que no se puede conectar directamente al módulo Wi-Fi, ya que sólo admite 3,3 voltios en sus patillas. Por lo que tenemos que poner un divisor de tensión que reduzca los 5 voltios a 3,3 voltios, en las líneas de transmisión de datos, reset y bootloader. En la línea de recepción de datos no hace falta ya que es una salida del ESP8266.

2.7. Diseño de la Placa de Circuito Impreso (PCB)

Una vez se tiene el esquemático terminado, el siguiente paso es asignar las huellas a los diferentes componentes para luego situarlos sobre la placa y de esta manera conectarlos mediante pistas. Para la realización de la PCB se utiliza el programa Proteus Ares.
Lo primero que se ha hecho, ha sido asignar las huellas a los componentes que ya se encuentran dentro de la librería de Proteus, como es el caso de las resistencias y de los condensadores cerámicos. Para estos componentes se ha seleccionado la métrica 0805, para este tamaño en Proteus ya existen huellas por defecto.
Sin embargo, para otros componentes no hay huellas en la librería de Proteus y se han tenido que diseñar, como es el caso de el módulo Wi-Fi, el conector mini USB, los optoacopladores, los triac, los diodos led, los condensadores electrolíticos, la fuente de alimentación, los Terminal Block y los sensores.
A continuación, vamos a explicar brevemente los pasos a seguir para hacer una nueva huella:

- El primer paso es comprobar que el datasheet del componente permite la extracción de páginas. Esto se puede consultar mediante Acrobat Reader en el menú Archivo->Propiedades. En extracción de página debe indicar permitido.
  Si no permite extraer páginas se debe desproteger el archivo PDF. Para ello, se puede utilizar algún programa online o APDFPR (Advanced PDF Password Recovery, 5.0.3 Enterprise Edition).

- Una vez comprobado esto, hay que tomar una instantánea de la imagen del datasheet que indica las dimensiones del componente.

- Después, hay que imprimir dicha imagen utilizando una impresora de tipo Postscript Encapsulado.
  Este tipo de impresora debe estar instalada en nuestro ordenador. Si no lo está, se puede añadir la impresora HP LaserJet 5/5M Postscript. Al instalarla hay que indicar que se quiere imprimir un fichero.

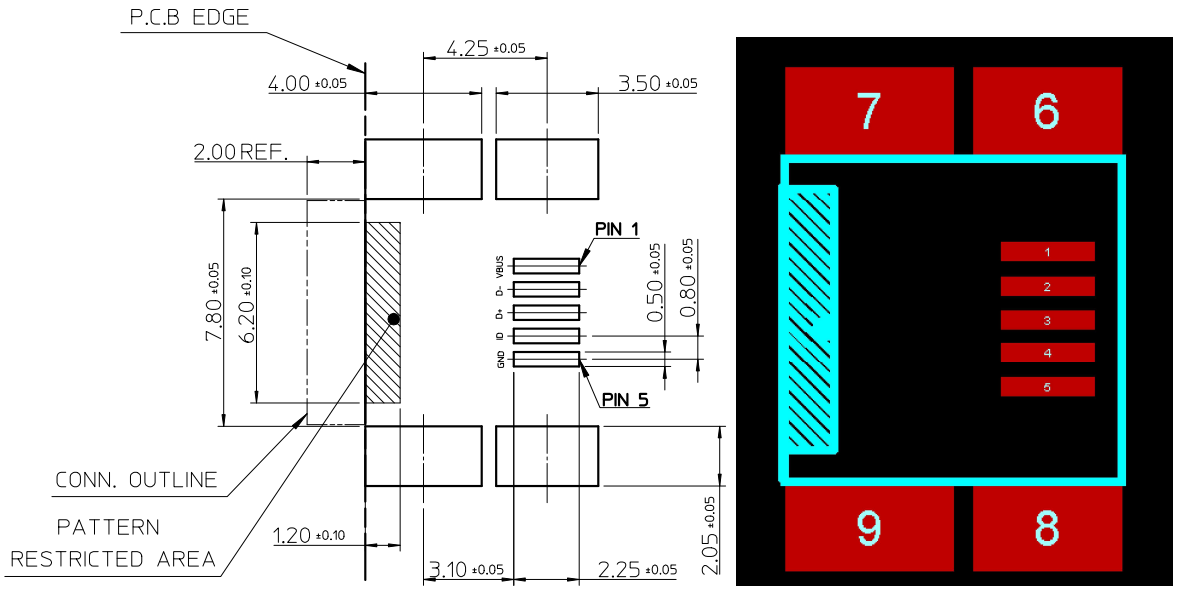
- Una vez impresa la imagen en formato PS, hay que abrirla con el conjunto de programas Ghostscript/GhostView y en el menú File, se debe pinchar en PS to EPS e indicar que calcule automáticamente el "Bounding Box".

- El siguiente paso, es convertir el fichero .EPS a .DXF, para ello hay que utilizar el programa Print2CAD 2012 OCR (Autocad 2004-2006).

- El siguiente paso es importar el fichero .DXF en Proteus Ares. En el menú "File->Import DXF"seleccionamos el fichero indicado y marcamos la opción de todas las capas y elegimos un factor de escala.
  En Proteus aparecerá la imagen del componente que hemos seleccionado. Para comprobar que el factor de escala escogido es el correcto, hay que medir una de las cotas y comprobar si coincide con las del dibujo. Si no coinciden, hay que multiplicar el facto de escala escogido por lo que mide ese lado en el diseño y el resultado se divide por lo que hemos medido en proteus. Por ejemplo: si se coge un factor de escala 10 y una cota del componente mide 20 mm y al realizar la medición nos da 30 mm, habría que calcular el nuevo factor de escala de la siguiente manera: 10*20/30=6,666666
  Con este nuevo factor de escala se vuelve a importar la imagen y se repite la comprobación. Este proceso hay que repetirlo hasta que coincidan las medidas.

- Por último, una vez que se tiene el factor de escala apropiado hay que dibujar el componente en la capa Top Silk y añadir los pads en la capa Top Copper. Para dibujarlo, la mejor opción es dibujar encima de la imagen que se ha importado del datasheet. Una vez terminado, seleccionamos el componente y con el botón derecho pinchamos en "Make 2D symbol package". De esta manera hemos creado la huella de un componente.

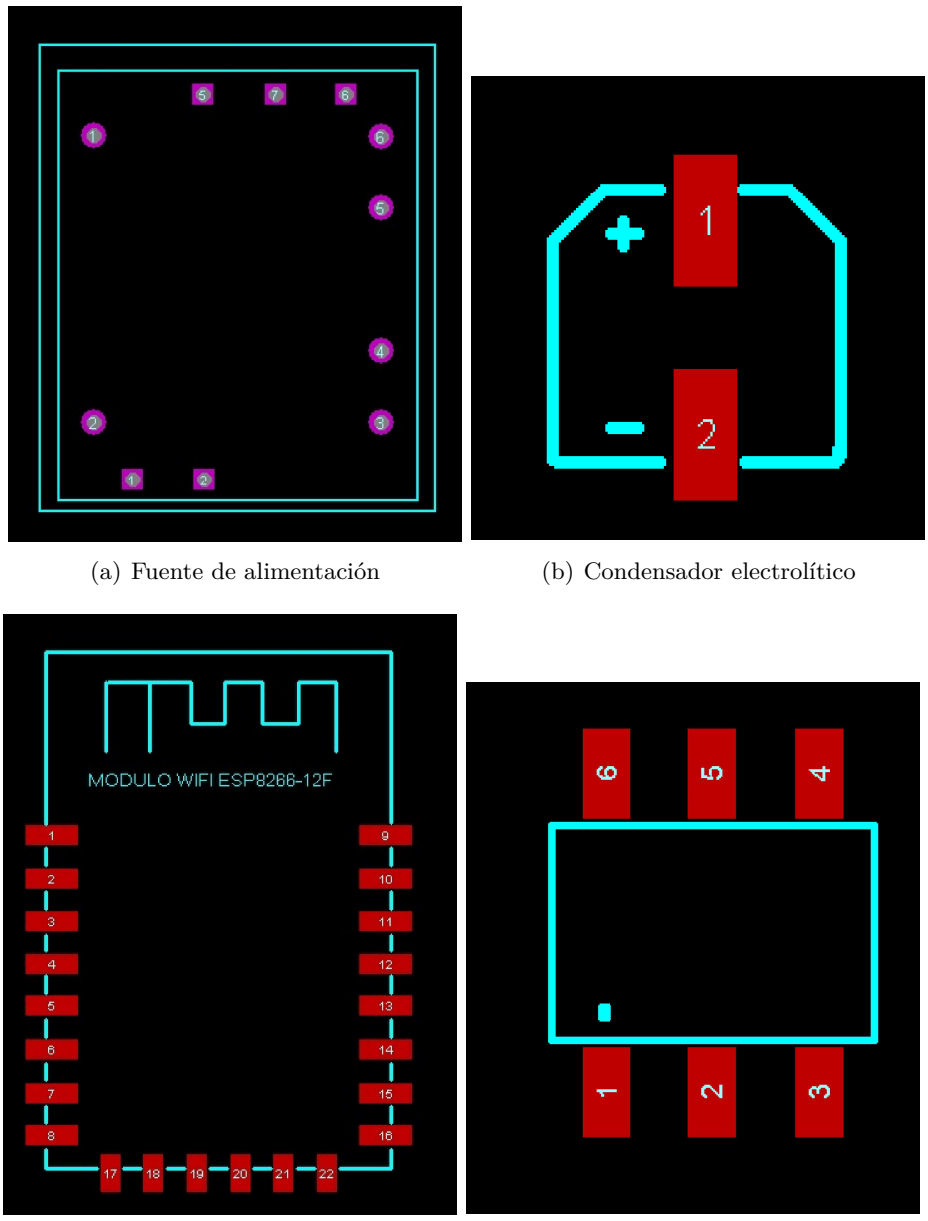
Ahora, vamos a ver algunas de las huellas diseñadas. Por ejemplo en la Figura 2.22, se puede ver en la imagen de la derecha el dibujo del conector mini USB en el datasheet y en la imagen de la derecha la huella que se ha diseñado.



(a) Dibujo en el datasheet (b) Huella diseñada

Figura 2.22: Conector mini USB Molex67503-1230

Por otro lado, en la Figura 2.23, se pueden ver algunas de las huellas realizadas. En la subfigura (a) se puede apreciar la doble huella diseñada para la fuente de alimentación ya que en la fase de diseño aún se dudaba de la fuente de alimentación que se iba a emplear. En la subfigura (b) se muestra la huella del condensador electrolítico. Mientras que en la subfigura (c) se puede observar la huella del módulo ESP8266-12F. Por último, en la subfigura (d) se puede apreciar la huella del optoacoplador. Comparando estas huellas, se puede ver que los pads de la fuente de alimentación son diferentes a las del resto. Esto se debe a que es un componente de inserción y los otros componentes son de montaje superficial.



(a) Fuente de alimentación (b) Condensador electrolítico

(c) ESP8266-12F (d) Optoacoplador

Figura 2.23: Huellas diseñadas

Una vez que se han creado las huellas de los componentes que no las tenían, estas deben asignarse a cada uno de ellos y es muy importante que se realice de forma correcta, fijándose detalladamente en los patillajes, ya que un error en la asignación, provoca que el dispositivo no funcione y se tenga que hacer algún apaño en la placa como puede ser cortar alguna pista y realizar conexiones con cables. Debido a esto, es recomendable tener presente el datasheet de los componentes en el momento de asignación de la huella.
Con el esquemático finalizado y todas las huellas asignadas, es el momento de pasar a Proteus Ares. El primer paso es definir el borde de la placa, en este caso es de 10 x 10 cm. Seguidamente, dentro de la placa definimos 3 regiones ya que vamos a sacar tres placas (principal, sensores y programador) más pequeñas de una grande. Con las tres regiones definidas, colocamos todos los componentes y trazamos las rutas entre ellos. Este último paso, se lleva a cabo teniendo en cuenta algunas consideraciones para el correcto diseño de una PCB:

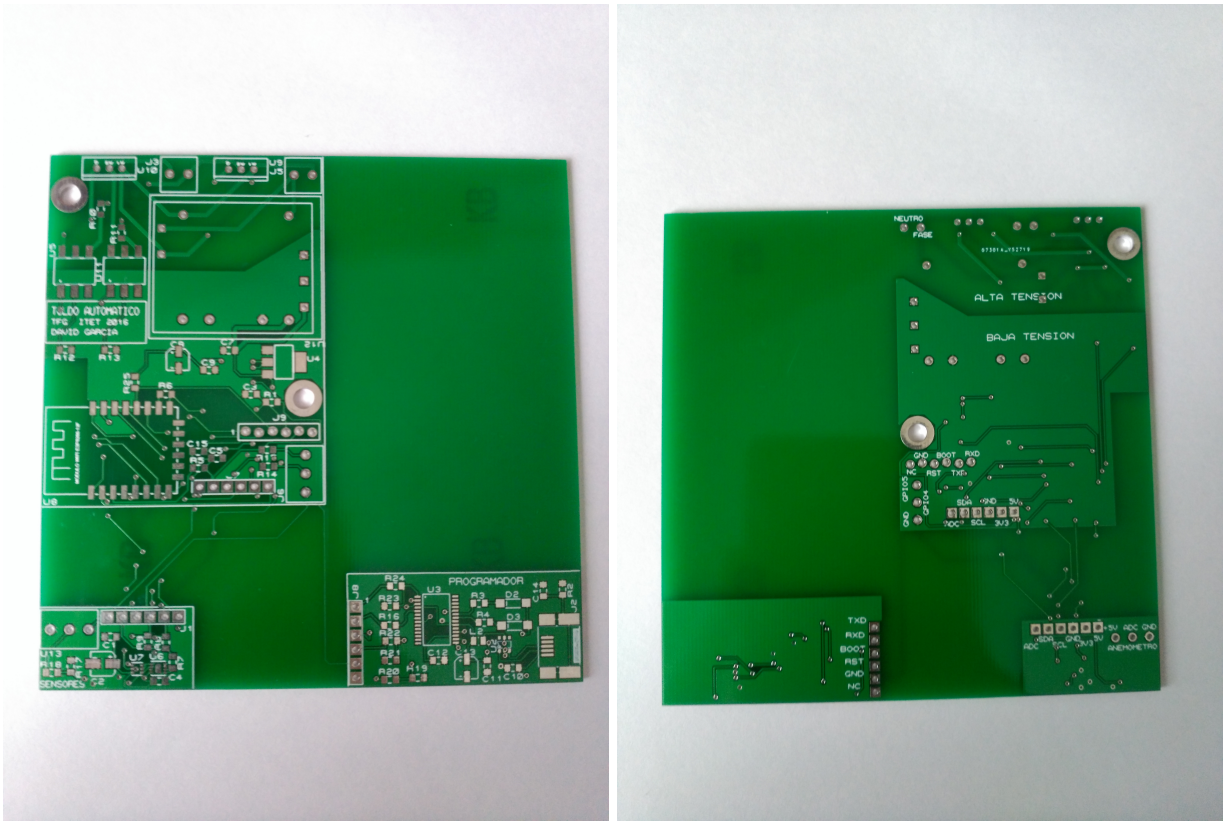
- Los conectores deben estar lo más cerca posible del borde. De esta manera, nos aseguramos de que son fácilmente accesibles.

- Las pistas deben ser lo más cortas posibles, separar unas de otras lo máximo posible. De esta manera, se evitan acoplamientos capacitivos entre pistas.
- Los condensadores de desacoplo deben estar lo más próximo al componente que deben proteger.
- Colocar uno o varios planos de masa para asegurar el mejor camino de retorno para la corriente.

Con todas las rutas establecidas, es momento de colocar la serigrafía necesaria. En este caso, se ha añadido el nombre de cada placa, el nombre del autor y de la carrera.
Por último, se generan los archivos Gerber. Estos archivos son los que se envían al fabricante de PCB, puesto que contienen la información necesaria para la fabricación de las placas. Para fabricar las PCB los fichero se han enviado a la empresa **ITEAD Studio**, cuyas instalaciones se encuentran en China. Como se ha comentado en otros apartados anteriores, el Layout completo se puede encontrar en el Apéndice B de este documento.

2.8. Proceso de fabricación

Una vez recibidas las placas como la que se muestra en la Figura 2.24, cortamos una por donde están definidas las regiones, así obtenemos las tres placas sin tener que pedir tres modelos de PCB diferentes y de esta manera abaratamos el coste de fabricación. Para ello, es necesario contar con una cizalla que permita cortar la PCB.

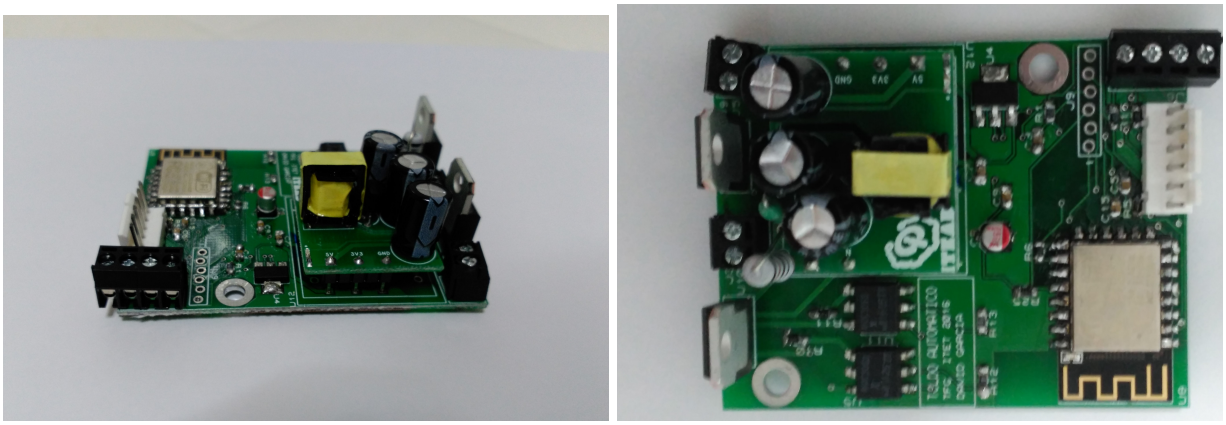


(a) Cara superior (b) Cara inferior

Figura 2.24: PCB original

Con las tres placas cortadas, el siguiente paso es montar los componentes. Para esta tarea, en el laboratorio contamos con soldadores, microscopios y pantallas ya que las piezas son tan pequeñas que se hace difícil soldarlas a simple vista.
A la hora de soldar cada componente hay que asegurarse de que está colocado correctamente ya que

una equivocación en la soldadura de algún pin puede provocar que se estropee el componente. Debido a esto es recomendable tener a disposición el esquemático y el Layout. En las Figuras 2.25, 2.26 y 2.27 se muestran las tres placas resultantes de este proceso.



(a) Perspectiva lateral

(b) Perspectiva superior

Figura 2.25: Placa principal

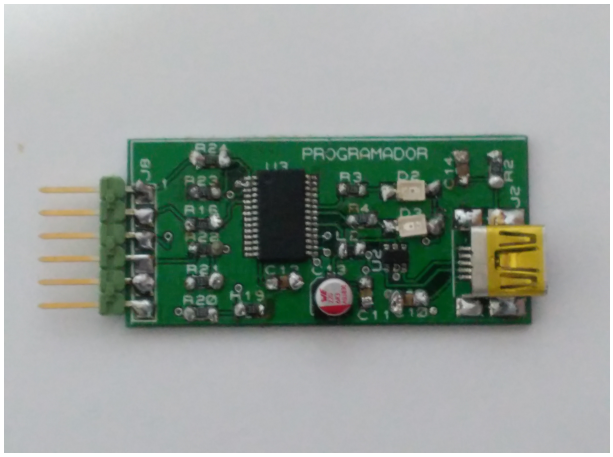


Figura 2.26: Placa del programador

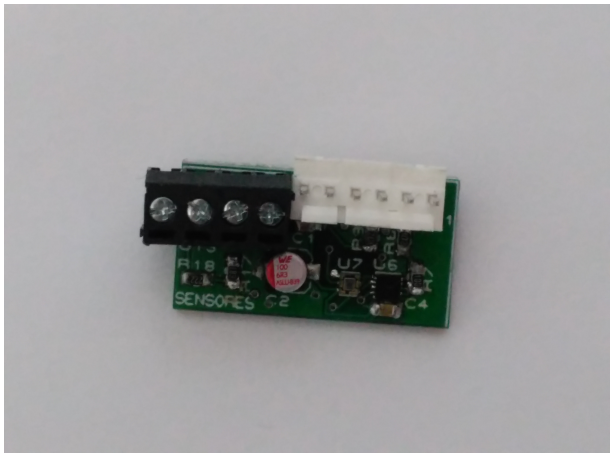


Figura 2.27: Placa de los sensores

2.8.1. Errores de diseño

En este apartado vamos a comentar sobre algunos errores que se han cometido en el diseño de la PCB. De esta manera, se evitarán en diseños futuros.
Durante el montaje de los componentes, se han encontrado los siguientes fallos que se pueden mejorar en otros diseños:

- La huella SSOP28 del FT232RL es algo justa. Los pads deberían ser un poco más largos. Esta huella ya viene definida en Proteus.
- Los pads de los componentes de inserción deberían ser más grandes, ya que se quedan un poco justos para que suelde el estaño.
- La huella de los optoacopladores es un poco pequeña. Debería ser algo más ancha para que el componente encaje bien.
- Los bloques terminales de bornas utilizados son de mala calidad ya que dan la sensación de que se van a arrancar si se conecta algo pesado. Además, son algo pequeños para que entre bien el cable. De hecho, para el conexionado de las bombillas se ha tenido que utilizar una regleta porque no entran dos cables empalmados en el bloque terminal de bornas.

Una vez montado todo, se comprueba si funciona correctamente. Lo primero es conectar el programador al ordenador y ver si lo detecta. En este caso, el ordenador lo reconoce.
El siguiente paso es grabar un programa de ejemplo para verificar que funciona el programador y el ESP8266. Durante esta prueba se encontró el problema de que el programador no podía programar el módulo Wi-Fi. Tras varías comprobaciones con el multimetro para verificar que todo estaba bien soldado y con el osciloscopio para visualizar la señal en cada punto del circuito, nos dimos cuenta de que la señal que salía del programador tenía un nivel de 5 voltios cuando debería tener 3,3 voltios. Esto se debe a que el divisor de tensión que hay entre el programador y el ESP8266, estaba al revés,es decir, reducía el nivel de la señal del módulo al convertir de niveles. En la Figura 2.28, se muestra como estaba el divisor de tensión, mientras que en la Figura 2.29 se puede ver como debería estar el divisor. En estas Figuras solo se aprecia para la línea de TXD del convertidor y la línea RXD del módulo Wi-Fi, pero también sucede en las lineas _DTR y _RTS del FT232RL, y en las líneas de reset y bootloader del ESP8266-12F.

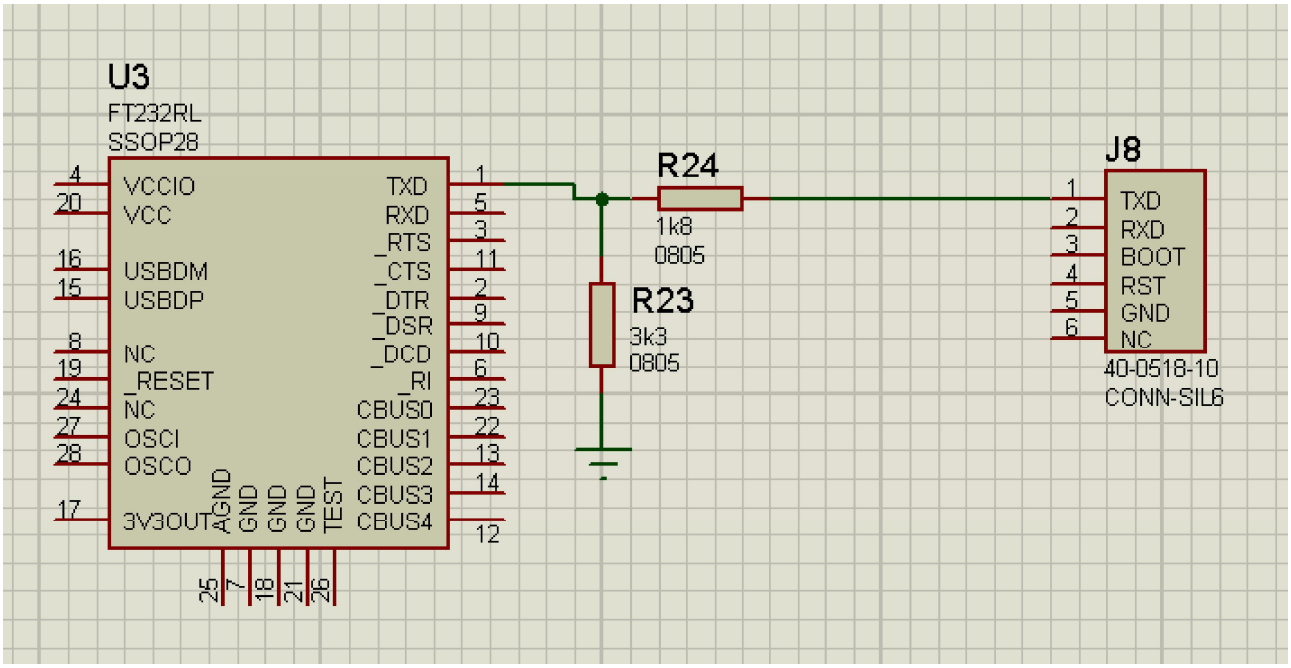


Figura 2.28: Divisor de tensión erróneo

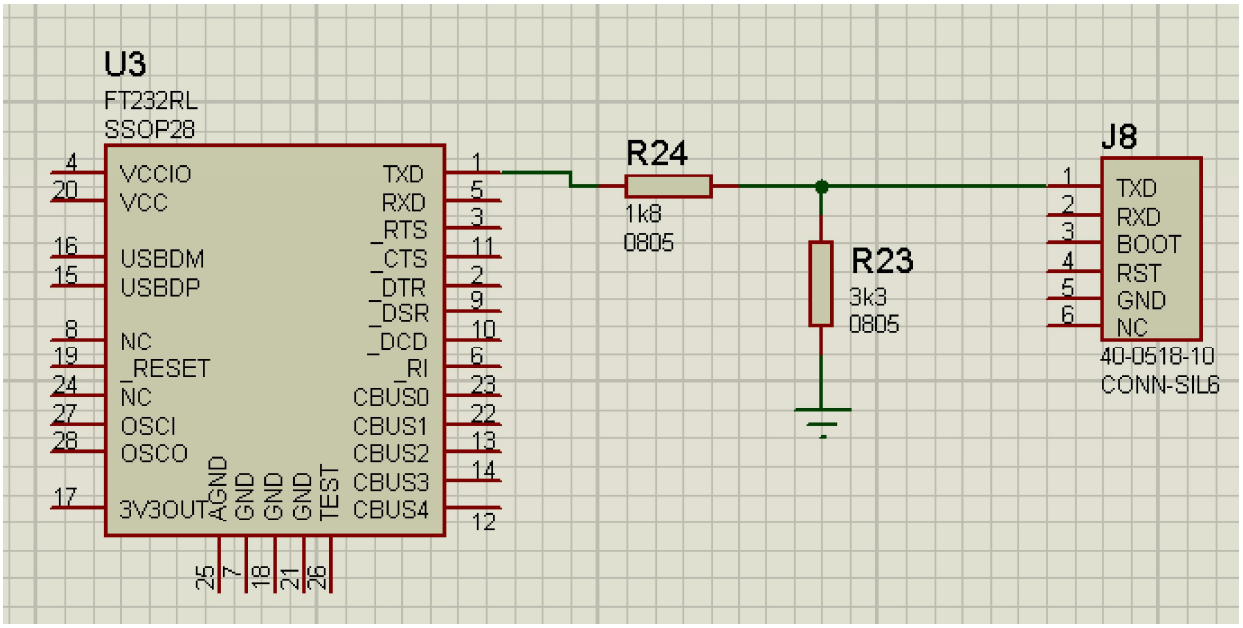


Figura 2.29: Divisor de tensión correcto

La solución ha este problema ha sido cortar algunas pistas del programador con un cutter y puentear otras con cables. El resultado de este apaño se muestra en la Figura 2.30.

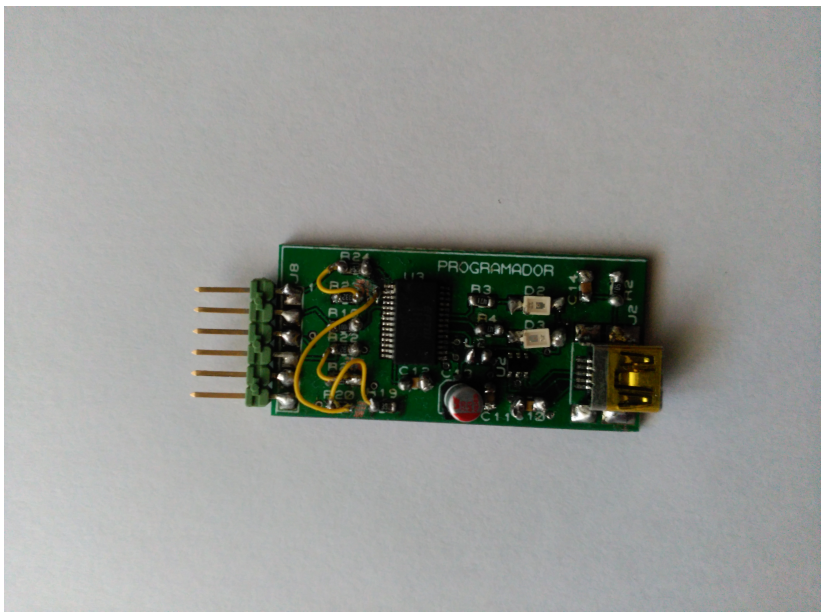
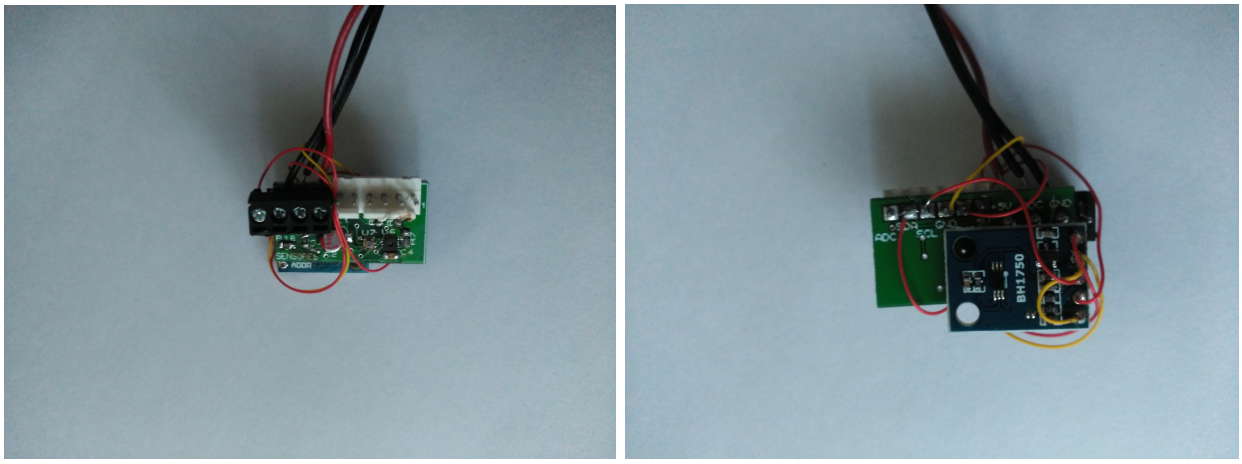


Figura 2.30: Programador arreglado

El otro problema que se ha encontrado, lo hemos comentado en el apartado del sensor de luz y es que es demasiado pequeño. Debido a esto hemos pedido uno que viene en una placa y solo se tienen que soldar unos cables. Además, hemos pegado esta placa en la parte posterior de la PCB de los sensores para que el sensor de luz este en una cara y los otros en otra, el motivo de esto ya lo hemos comentado anteriormente. En la Figura 2.31, se muestra el resultado de esta operación.



(a) Cara superior (b) Cara inferior

Figura 2.31: Placa de los sensores

Una vez que todo el hardware funciona correctamente, se puede meter en la caja y realizar el montaje final. En la Figura 2.32, se muestra la caja con los interruptores y la placa principal. Mientras que en la Figura 2.33, se puede ver el montaje final con todo metido en la caja, las bombillas y la placa de los sensores.

Figura 2.32: Interior de la caja

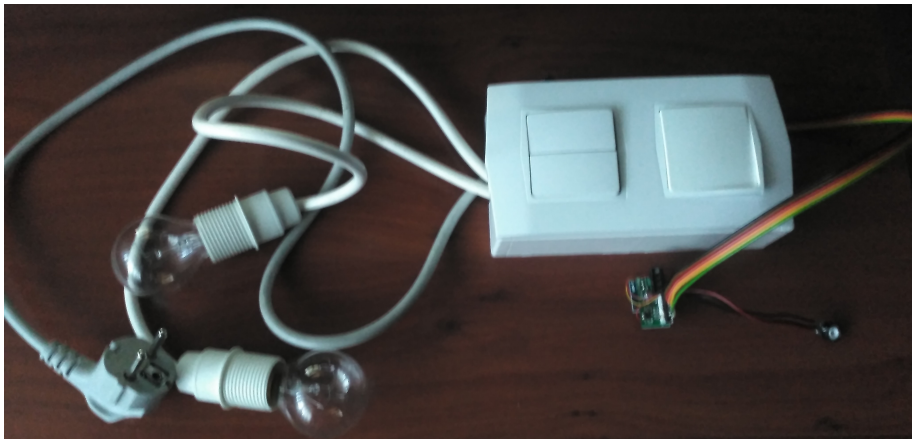


Figura 2.33: Montaje final

Capítulo 3

Software

En este capítulo se va a explicar como funciona el toldo. Para ello, se va a describir las funciones que se han programado. Además, se va hablar sobre el broker: ¿qué es y como se configura? Por último, se explicará el software de la aplicación para el móvil.

3.1. Parte del toldo

3.1.1. Instalación y configuración del entorno de desarrollo

Como se ha comentado anteriormente, se ha optado por utilizar el módulo Wi-Fi como microcontrolador ya que tiene uno integrado. Además, debido a que en Arduino ya hay librerías que implementan el protocolo MQTT, se ha decidido utilizar el IDE(*Integrated Development Environment, Entorno de Desarrollo Integrado*) de Arduino para programar el módulo Wi-Fi.
Por tanto, el primer paso es descargar el IDE de Arduino. Esto se puede en la propia página de Arduino: http://www.arduino.cc, en la pestaña de software.
El siguiente paso es descargar el plugin ESP8266 para Arduino que incluye librerías y las placas de toda la familia ESP. Los pasos que hay que seguir para instalar el plugin son los siguientes:
Lo primero es comprobar que la versión del IDE de Arduino que hemos instalado sea superior o igual a la 1.6.4.
Después, en el IDE de Arduino, pinchar en Archivo->preferencias y aparece el siguiente menú (Figura 3.1).
En el cuadro que está marcado en rojo hay que pegar el siguiente enlace:

http://arduino.esp8266.com/package_esp8266com_index.json

Una vez copiado este enlace, pinchar en OK.

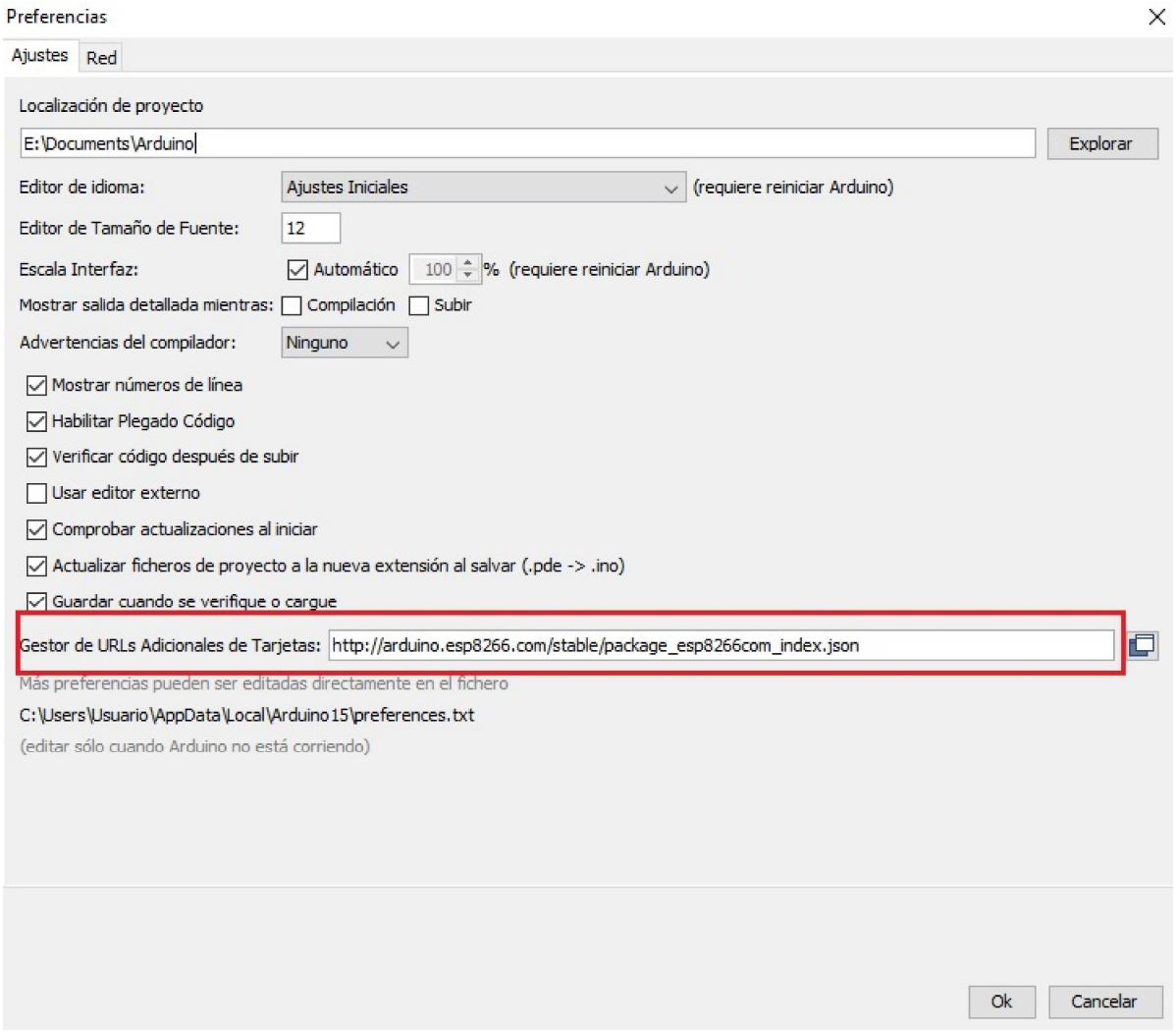


Figura 3.1: Preferencias IDE Arduino

Por último, pinchar en Herramientas->Placa->Gestor de tarjetas y aparecerá la siguiente ventana (Figura 3.2). En ella buscamos esp8266 y abajo a la derecha aparece la opción de instalar. Dando al botón de instalar, comienza a instalarse y cuando acaba ya está el pluggin instalado.

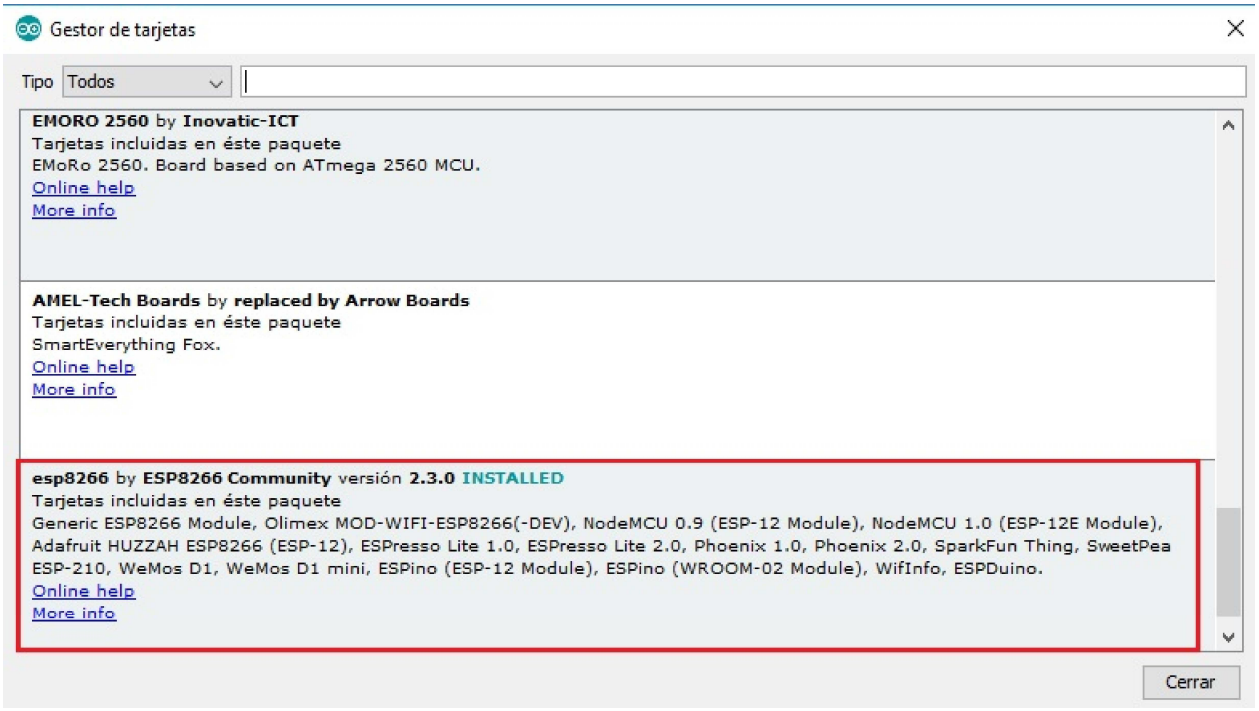


Figura 3.2: Gestor tarjetas IDE Arduino

La última librería que hace falta instalar es la de MQTT de Adafruit. Para ello, primero hay que descargarla del siguiente enlace:

https://github.com/knolleary/pubsubclient

Una vez descarga, en el IDE de Arduino, pinchar en Programa->Incluir librería->Gestionar librerías y aparecerá la siguiente ventana (Figura 3.3). En el cuadro de arriba a la derecha buscamos Adafruit MQTT y cuando aparezca, pinchamos en instalar.

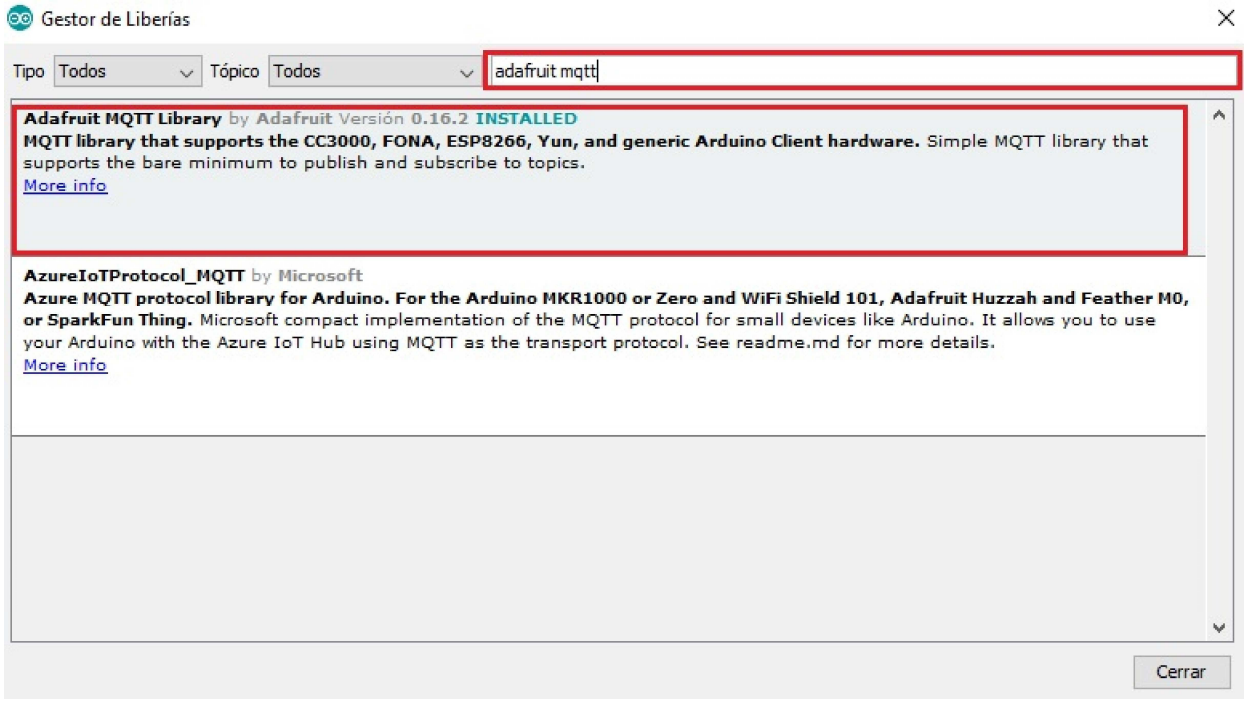


Figura 3.3: Instalación librería MQTT de Adafruit

También, se utilizan las librerías Wire y UDP, pero estas no hace falta instalarlas ya que vienen instaladas por defecto en el IDE de Arduino.

3.1.2. Funcionamiento del toldo

En este apartado vamos a explicar el funcionamiento de nuestro dispositivo. Para ello, se va ir describiendo el software que se ha programado. Todo este software se puede ver completo en el Apéndice D.
Lo primero que hace el programa nada más arrancar, es conectarse al Wi-Fi que esté configurado y al servidor MQTT que está definido en el programa. Para realizar esto, se ha utilizado la librería MQTT de Adafruit. Después de conectarse al broker, se suscribe a todos los tópicos que se han indicado, en este caso son los siguientes:

- david/toldo/Horas: indica las horas de subida o bajada para el modo vacaciones.
- david/toldo/TemperaturaUmbrales: indica los dos umbrales de temperatura para el modo automático.
- david/toldo/HumedadUmbrales: indica los umbrales de la humedad para el modo automático.
- david/toldo/LuzSubir1: indica el umbral de luz para la subida del toldo en el modo automático.
- david/toldo/LuzBajar1: indica el umbral de luz para la bajada del toldo en el modo automático.
- david/toldo/Cambiar: este tópico permite cambiar la posición del toldo desde la aplicación.
- david/toldo/Clase: indica la clase del toldo.
- david/toldo/modoVacaciones: se utiliza para activar y desactivar el modo vacaciones.

Estas tres acciones se realizan en la función **"setup"**. Además, en esta, se inicializa la comunicación I2C que se utilizará más adelante para obtener las mediciones de los sensores.
Después se ejecuta la función **"loop"** que se está repitiendo todo el tiempo.

Dentro de la función **"loop"**, lo primero que se comprueba es si se ha producido un error en las suscripciones iniciales. Si no ha ocurrido, se sigue ejecutando el programa.

Después, se toma el tiempo cuando empieza el bucle para que al final de él, se pueda calcular lo que tarda en repetirse. Una vez hecho esto, se comprueba si la conexión al broker sigue activa, para ello se llama a la función **"MQTT_connect"**. Si la conexión sigue activa el programa continua ejecutandose, pero si no está activa, se reintenta conectar cada 5 segundos hasta que lo consigue. Si no lo consigue en 5 intentos, hay que resetear el módulo Wi-Fi.

Después de comprobar la conexión a MQTT, se ejecuta la función **"publicarValoresPorDefecto"** que como su propio nombre indica, publica los valores por defecto de los umbrales de temperatura, de los de humedad, de los de luz, de la clase del toldo y de la posición del toldo. Los tópicos que publica son los siguientes:

- david/toldo/Inicial1: en este tópico publica el modo vacaciones, la clase del toldo, el estado inicial del toldo y si se ha producido algún error en las suscripciones.
- david/toldo/TemperaturaUmbrales: publica los dos umbrales de temperatura para el modo automático.
- david/toldo/HumedadUmbrales: publica los umbrales de la humedad para el modo automático.
- david/toldo/LuzSubir1: indica el umbral de luz para la subida del toldo en el modo automático.

- david/toldo/LuzBajar1: indica el umbral de luz para la bajada del toldo en el modo automático.

Esta función solamente se ejecuta una vez al arrancar y ya no se vuelve a llamar, a pesar de que se repite el bucle.

Seguidamente, se comprueba si alguno de los tópicos a los que se ha suscrito tiene un nuevo mensaje. En caso afirmativo, se guarda el valor del mensaje en la variable correspondiente.

El siguiente paso, es obtener la hora de un servidor, en concreto, *clepsidra.tel.uva.es*, ya que en la universidad no se puede consultar la hora de otro servidor. Esto es por los cortafuegos que tiene la red. Para utilizar el toldo en cualquier hogar, se puede utilizar el servidor *hora.rederis.es*.
Para obtener la hora del servidor, se utiliza el protocolo NTP(*Network Time Protocol, Protocolo de Tiempo de Redes*). Este protocolo sirve para sincronizar dispositivos con una única referencia horaria. Su estructura es cliente-servidor. Además, trabaja por el puerto 123, utilizando el protocolo UDP. También, emplea el horario universal coordinado (UTC), que básicamente es un estándar de alte precisión de tiempo atómico.
En nuestro programa hay tres funciones que implementan este protocolo: **"sendNTPpacket"**, **"NTP"** y **"obtenerFechaHora"**. En la primera de ellas, se configura y envía un paquete solicitando el tiempo al servidor. En la segunda función, se comprueba si ha recibido un paquete del servidor. Una vez recibido el paquete, se trabaja para conseguir la fecha y la hora en segundos. Por último, en la función **"obtenerFechaHora"** se convierten los segundos a la fecha y hora actuales.
En el programa se pide la hora al servidor nada más arrancar y una vez al día, en concreto, a las 11 de la noche. Con los segundos que devuelve la función **"NTP"** y millis() se consigue un factor de correción que nos permite llevar una cuenta interna de la hora. La función que lleva esta cuenta es **"conseguirSegundos"**. De esta manera, cuando se quiere consultar la hora no hace falta pedirla al servidor. Se pide una vez al día al servidor por motivos de sincronización.

Lo siguiente que se comprueba es la clase del toldo, ya que dependiendo de esta, es más o menos resistente al viento. LA clase se puede modificar en la aplicación y se envía al toldo a través del tópico david/toldo/Clase. En la función **"determinarClaseToldo"** se asigna un viento máximo que no puede superarse porque si lo hace, el toldo puede ser arrancado. La velocidad del viento que soporta según la clase es la siguiente:

- Clase 1: 28 km/h
- Clase 2: 38 km/h
- Clase 3: 49 km/h

Una vez que se conoce la clase del toldo, se hace una medición inicial del viento en la función **"obtenerViento"**. Si es superior al umbral que se haya definido, el toldo se recoge automáticamente y no deja que se pueda bajar. Cada 5 minutos se vuelve a recibir el viento medido, el cuál es un promedio de las últimas 100 muestras tomadas. Cada muestra se guarda en un vector de tamaño 100. Una vez que se han tomado 100 muestras, cada vez que se coge otra, se borra la más antigua del vector. De esta manera, se tiene la velocidad del viento de los últimos 5 minutos. Las muestras se toman cada 3 segundos. La medición del viento, se envía a la aplicación del móvil a través del tópico david/toldo/viento.

Si el viento es inferior al umbral establecido, pasamos a evaluar la posición de los interruptores.
Si los dos interruptores están en la posición 00, el toldo está en modo manual, es decir, sólo funciona con la manivela. Además, si está en este modo, desde la aplicación del móvil se puede subir y bajar

hasta la altura deseada.
Si los interruptores están en la posición 10, el toldo sube. Mientras que si están en la posición 01, el toldo baja. Para saber el tiempo que el toldo está subiendo o bajando, en el programa se ha tenido que guardar el valor antiguo de los interruptores y de cada una de estas dos opciones sacar otras dos, haciendo un total de seis opciones. Por otra parte, se ha estimado que el toldo tarda 30 segundos en pasar de un estado a otro. Este tiempo es una estimación, si se vende el dispositivo habría que calcular este tiempo en función de la velocidad del motor. El estado del toldo se envía a la aplicación mediante el tópico david/toldo/estadoToldo.
Por último, si los interruptores están en la posición 11, es el modo automático. En este modo, se puede subir y bajar el toldo desde la aplicación, al igual que sucede en el modo manual. Pero si esta opción está desactivada, en este modo hay otras dos opciones: modo vacaciones y modo automático que sube o baja el toldo en función de las mediciones de los sensores y unos umbrales que se pueden modificar en la aplicación.
Si está activado el modo vacaciones, lo primero que se hace es comprobar si el mes es Mayo, Junio, Julio, Agosto o Septiembre, puesto que en los otros meses el tiempo no se suele bajar el toldo debido a las condiciones climatológicas. Después se obtiene la hora y se compara con las dos horas que se pueden agregar en la aplicación y que son enviadas a través del tópico david/toldo/Horas. Si coincide con alguna de ellas, se genera un número aleatorio del 1 al 3 y según ese número se aplica una máscara a la función millis():

- 1: millis()& 0xFF
- 2: millis()&0x3FF
- 3: millis()&0x4FF

El resultado obtenido se suma a los segundos que devuelve la función **"conseguirSegundos"** y se obtiene la hora a la que subirá o bajará el toldo. Con estas máscaras, el toldo cambiará de posición entre en punto e y media de la hora definida en el teléfono, es decir, si en el móvil se ha puesto las 10, el toldo subirá o bajará entre las 10 y las 10:30.
Si el toldo está en una posición cercana a estar subido, se sube. Si no lo está, se baja. Lo mismo sucede para el caso de bajar.
Se definen dos horas para que en una sube o baje y en la otra haga lo contrario.
Por otro lado, si el modo vacaciones está desactivado, el toldo funciona en modo automático, es decir, sube o baja en función de unos umbrales definidos y las mediciones de los sensores.
Cada vez que se activa este modo se hace una primera medición de los sensores de humedad-temperatura y luz. Luego, se va cogiendo una muestra cada 6 segundos hasta llenar un vector de tamaño 100 y se hace el promedio de estas muestras. Cada 10 minutos se obtiene una nueva medición de los sensores. Una vez que se tienen 100 muestras, al tomar una nueva se borra la más antigua. Además, estas mediciones son enviadas al móvil mediante los tópicos david/toldo/Temperatura, david/toldo/Humedad y david/toldo/Luz. Todo este proceso se realiza en las funciones **"obtenerTemperaturaHumedad"** y **"obtenerLuz"**.
Cuando se tienen las condiciones climatológicas, se llama a la función **"modoAutomatico"**. En esta a su vez, se llama a la función **"calculaProbabilidad"**, que como su propio nombre indica, calcula la probabilidad de subida o bajada del toldo según el valor de la medición y los umbrales. En la Figura 3.4, se muestran las rectas que se han utilizado. La recta roja representa la subida del toldo y la recta azul representa la bajada, aunque los umbrales sean los de la temperatura, para la humedad y la luz es lo mismo. Aunque, en el caso de la humedad, el umbral de bajada es más pequeño que el de subida. Por lo que, la curva de subida tiene pendiente positiva y la de bajada pendiente negativa. Estas curvas se pueden ver en la Figura 3.5.

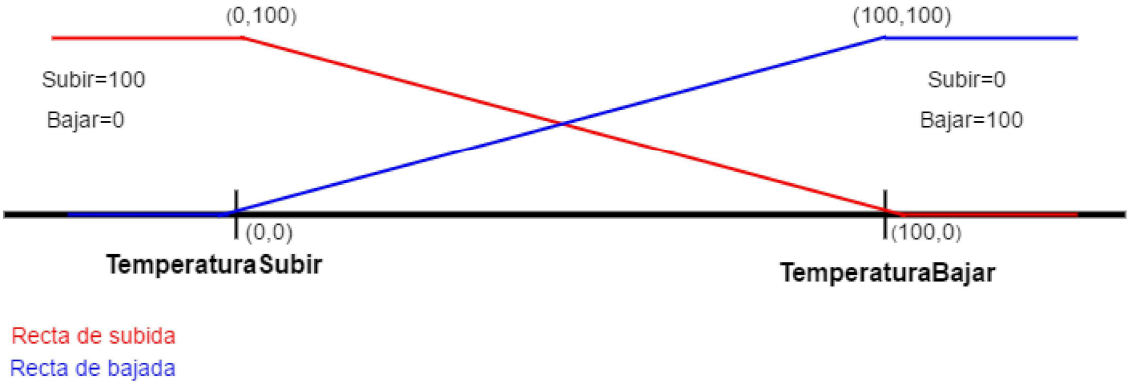


Figura 3.4: Rectas utilizadas para calcular la probabilidad de subida o bajada del toldo

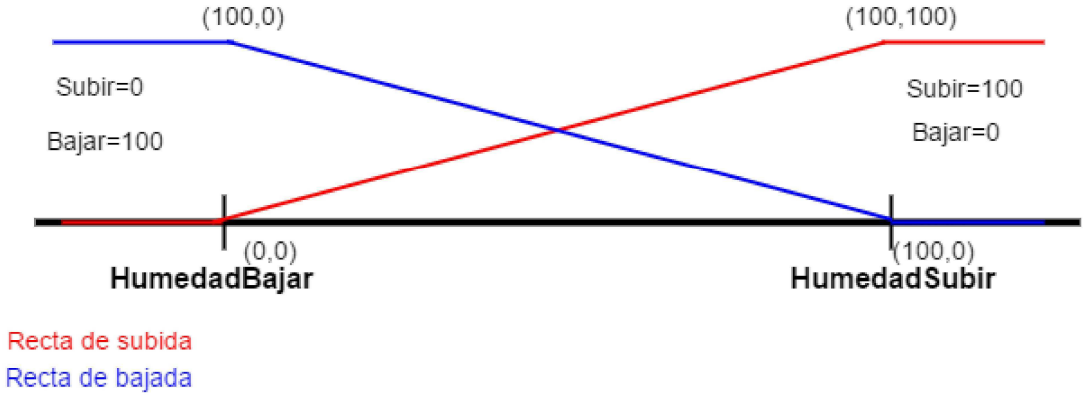


Figura 3.5: Rectas según la humedad para calcular la probabilidad de subida o bajada del toldo

A la función **"calculaProbabilidad"** se le pasa los umbrales de subida y bajada, la medición del sensor y la pendiente de la curva. Si la medición del sensor es igual o menor que el umbral de subida, la variable subir será igual a 100 y la variable bajar será 0. Mientras que, si la medición es superior o igual al umbral de bajar y la variable subir será 0, la variable bajar será igual a 100. Pero, si la medición está entre los dos umbrales hay que calcular el valor de estas dos variables. Para ello, se utilizan las ecuaciones de las rectas.
La ecuación de la recta de subida es:

$$y=\frac{100}{UmbralInferior-UmbralSuperior}*x+(100\text{-}UmbralInferior*\frac{100}{UmbralInferior-UmbralSuperior})$$

La ecuación de la recta de bajada es:

$$y=\frac{-100}{UmbralInferior-UmbralSuperior}*x+(0\text{-}UmbralInferior*\frac{-100}{UmbralInferior-UmbralSuperior})$$

Esta función, se llama seis veces. En tres de ellas, se evalúa la ecuación de subir y se obtiene una variable subir que es la suma de los tres resultados. Lo mismo sucede con las otras tres, pero evaluando la ecuación de bajar y obteniendo la suma de los resultados en la variable bajar.
Por último, en la función **"modoAutomatico"** se comparan las dos variables y si subir es mayor o igual que bajar, se sube el toldo. Mientras que, si subir es menor que bajar, se baja el toldo. Además, se llama a la función**"determinarNoche"** que indica si es de noche o no según el mes y la hora. Si es de noche aunque se cumplan las condiciones para bajar el toldo, el toldo permanece subido.
Este método para determinar la probabilidad de subida o bajada del toldo proviene de la lógica difusa.

Lógica difusa

La lógica difusa es una técnica de la inteligencia computacional que permite trabajar con información que tiene un alto grado de imprecisión, en esto se diferencia de la lógica convencional, puesto que esta trabaja con información bien definida y precisa. Se trata de una lógica multi-valuada, ya que permite valores intermedios para poder definir evalucaciones entre si/no, verdadero/falso, negro/blanco, caliente/frío, etc.
El concepto de Lógica Difusa fue concebido por Lofti A. Zadeh, quién inconforme con los conjuntos clásicos (*crisp sets*) que sólo permiten dos opciones, la pertenencia o no de un elemento a dicho conjunto. Presentó la lógica difusa como una forma de procesar información permitiendo pertenencias parciales a unos conjuntos, que en contraposición a los clásicos los denominó Conjuntos Difusos (*fuzzy sets*). Un elemento forma parte de un conjunto con un determinado grado de pertenencia.
Zadeh afirma que la lógica difusa trata de copiar la forma en la que los humanos toman decisiones. Además, menciona que la gente no requiere información numérica precisa del medio que le rodea para desarrollar tareas de control altamente adaptables, por ejemplo caminar por una acera sin chocarse con otras personas y los postes.

El concepto clave para entender como trabaja la lógica difusa es el conjunto difuso.
Un conjunto difuso puede definirse de forma general como un conjunto con límites difusos. Sea X, un rango de valores, también conocido como Universo del discurso y sus elementos se denotan como x. En la lógica convencional se define un conjunto C sobre X mediante la función característica de C como f_c.
Esta función es 1, si el elemento x pertenece al conjunto C y 0 si el elemento x no pertenece al conjunto C.
Si se generaliza esta función para que los valores asignados a los elementos del conjunto caigan en un rango particular y de esta manera, indicar el grado de pertenencia de los elementos a es conjunto, tendremos una función de pertenencia de un conjunto difuso. La función de pertenencia μ_C por la que se define un conjunto difuso C, sería:

$$\mu_C\text{=X->[0,1]}$$

Donde μ_C(x)=1 si x está totalmente en C, μ_C(x)=0 si no está en C y 0<μ_C(x)<1 si x está parcialmente en C. Este valor entre 0 y 1 representa el grado de pertenencia de un elemento x a un conjunto C.
A continuación, se va explicar un ejemplo de la diferencia entre las dos lógicas. El ejemplo, trata sobre la altura de una población.
En la Figura 3.7, se muestran las gráficas de un conjunto clásico (la de arriba) y de un conjunto difuso (la de abajo) para determinar si una persona es alta o no.
En la gráfica de la lógica clásica, se separa en 1,8 metros las personas que son altas de las que no. Cualquier persona que mida 1,8 metros o superior es alta. Mientras que si mide menos, es una persona baja.
Por otro lado, en la gráfica del conjunto difuso, la transición de persona baja a persona alta es más suave. Cada persona tendrá un grado de pertenencia al conjunto de los altos. Este grado de pertenencia se determina con la medida de dicha persona y la ecuación que define la curva de la gráfica.

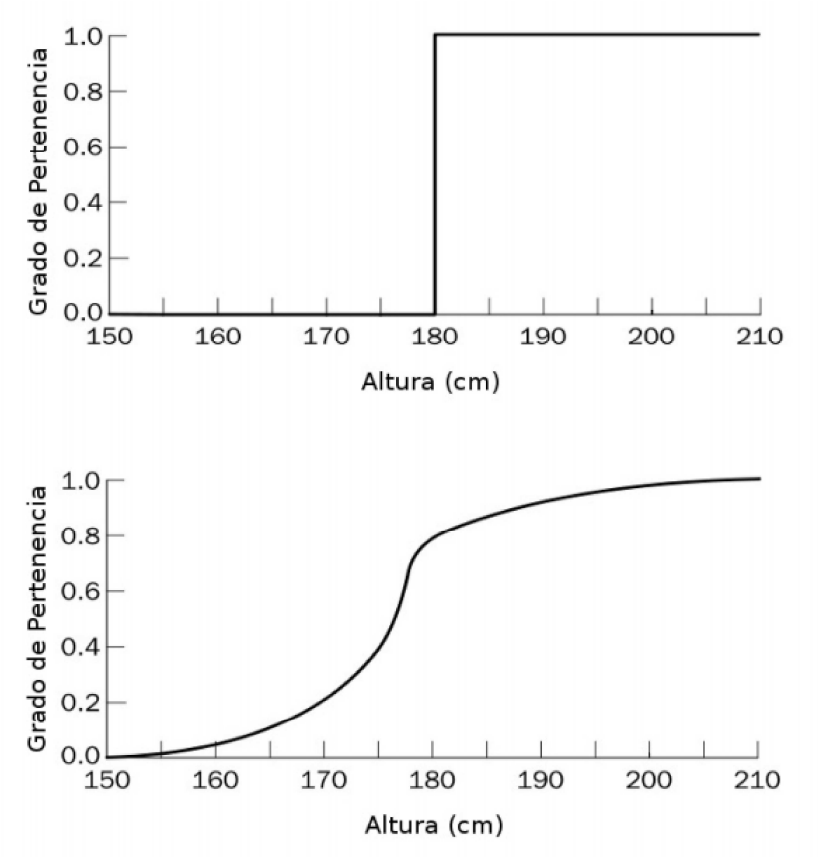


Figura 3.6: Gráficas para lógica clásica y lógica difusa

Según estas gráficas:

- Una persona de 1,95 metros, tiene un grado de pertenencia de 1 al conjunto de personas altas para ambas lógicas.
- Una persona de 1,87 metros, tiene un grado de pertenencia al conjunto de personas altas de 1 en la lógica clásica y 0,95 en la lógica difusa.
- Mientras que una persona de 1,79 metros, tiene un grado de pertenencia al conjunto de personas altas de 0 en la lógica clásica y 0,71 en la lógica difusa.

Por último, en nuestra aplicación las rectas empleadas sirven para conocer el grado de pertenencia que tiene el toldo según la medición del sensor para los conjuntos subir y bajar.

3.1.3. Resumen de funcionamiento

Para resumir el funcionamiento del toldo, se ha realizado un diagrama de estados. Este diagrama se puede ver en la Figura 3.6.

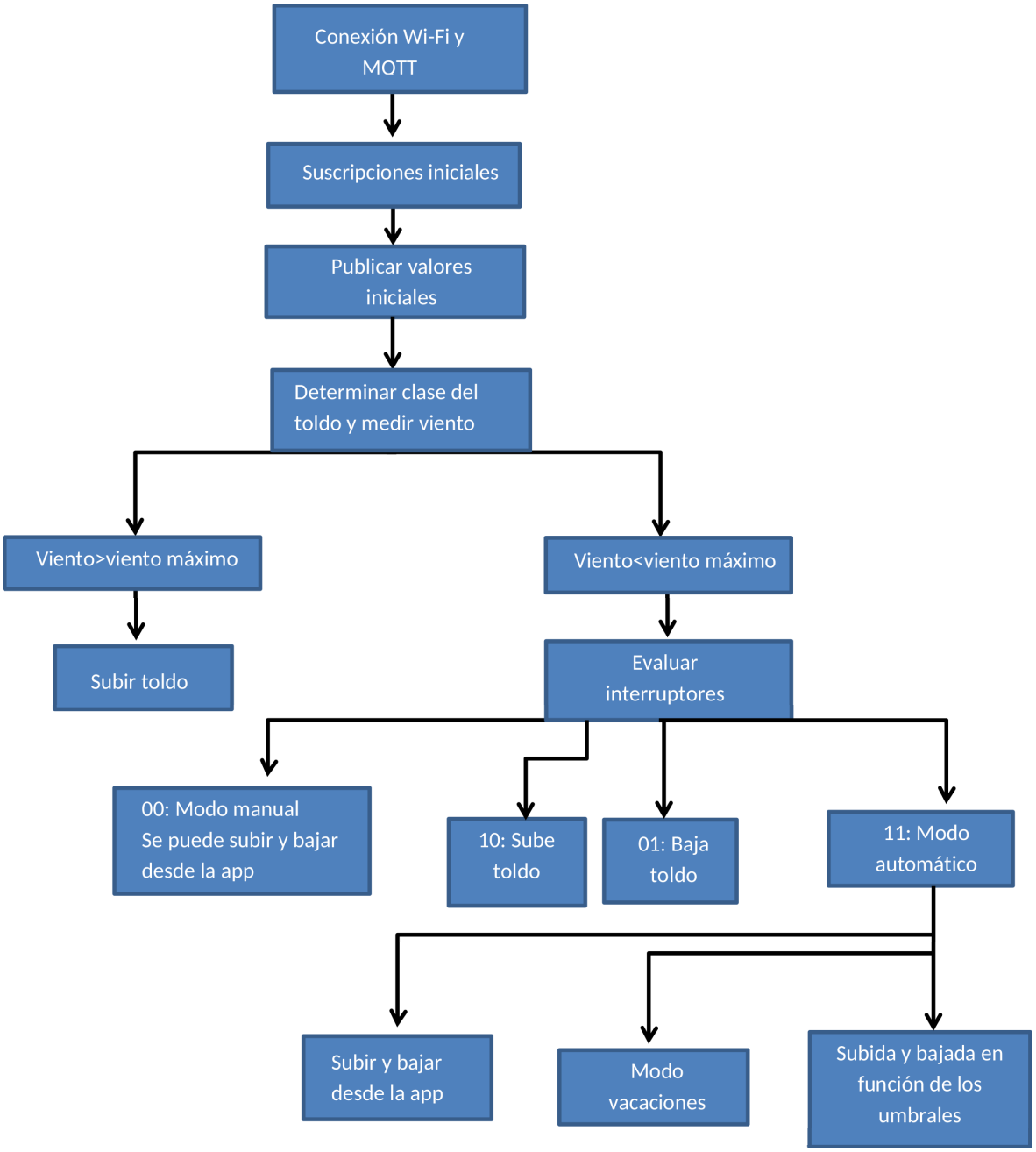


Figura 3.7: Diagrama de estados

3.1.4. Modificación de librería MQTT

Para desarrollar nuestro software se ha utilizado la librería MQTT de Adafruit. Sin embargo, esta no incluye retención de los mensajes que publican los tópicos. La retención se tiene que configurar en el cliente. Para nuestra dispositivo, es interesante que se retengan estos valores para que cuando se abra la aplicación del móvil aparezcan los últimos parámetros introducidos. La solución ha sido modificar está librería para que los mensajes publicados incluyan la retención en el broker.
En el fichero Adafruit_MQTT.cpp hay que hacer las siguientes modificaciones:

- Sustituir línea 296 por:
  bool Adafruit_MQTT::publish(const char *topic, const char *data, uint8_t qos, uint8_t retain)
  {
- Sustituir línea 297 por:
  return publish(topic, (uint8_t*)(data), strlen(data), qos, retain);

- Sustituir línea 300 por:
  bool Adafruit_MQTT::publish(const char *topic, uint8_t *data, uint16_t bLen, uint8_t qos, uint8_t retain)
- Sustituir línea 302 por:
  uint16_t len = publishPacket(buffer, topic, data, bLen, qos, retain);
- Sustituir línea 637 por:
  uint8_t *data, uint16_t bLen, uint8_t qos, uint8_t retain) {
- Sustituir línea 650 por:
  p[0] = MQTT_CTRL_PUBLISH << 4 | qos << 1 | (retain ? 1 : 0);
- Sustituir línea 773 por:
  const char *feed, uint8_t q, uint8_t r) {
- Añadir debajo de la línea 776:
  retain = r;
- Sustituir la línea 782 (antigua línea 781) por:
  return mqtt->publish(topic, payload, qos, retain);
- Sustituir línea 788 por:
  return mqtt->publish(topic, payload, qos, retain);
- Sustituir línea 794 por:
  return mqtt->publish(topic, payload, qos, retain);
- Sustituir línea 798 por:
  return mqtt->publish(topic, payload, qos, retain);
- Sustituir línea 804 por:
  return mqtt->publish(topic, payload, bLen, qos, retain);

Se muestra al completo este fichero en el Apéndice E (sin modificar) y en el Apéndice G (modificado). Mientras que en el fichero Adafruit_MQTT.h, hay que realizar los siguientes cambios:

- Sustituir línea 181 por:
  bool publish(const char *topic, const char *payload, uint8_t qos = 0, uint8_t retrain = 1);
- Sustituir línea 182 por:
  bool publish(const char *topic, uint8_t *payload, uint16_t bLen, uint8_t qos = 0, uint8_t retrain = 1);
- Sustituir línea 247 por:
  uint16_t publishPacket(uint8_t *packet, const char *topic, uint8_t *payload, uint16_t bLen, uint8_t qos, uint8_t retain);
- Sustituir línea 257 por:
  Adafruit_MQTT_Publish(Adafruit_MQTT *mqttserver, const char *feed, uint8_t qos = 0, uint8_t retain = 1);
- Añadir debajo de la línea 270:
  uint8_t retain;

Este fichero se puede ver al completo en el Apéndice F (sin modificar) y en el Apéndice H (modificado).

3.2. Mosquitto

Como se ha comentado anteriormente, el broker utilizado para este proyecto es Mosquitto. Mosquitto es un broker open source que implementa las versiones 3.1 y 3.1.1 del protocolo MQTT. Debido a que es open source, se puede descargar y ejecutar en cualquier dispositivo, en diferentes sistemas operativos como Windows, Mac OS X, Linux, etc. Para descargarlo, se puede hacer en su página oficial. [1]
El dispositivo que tiene instalado Mosquitto debe de estar encendido las 24 horas del día ya que los clientes pueden publicar y suscribirse en cualquier momento. Una solución es instalar Mosquitto en el router ya que esta siempre encendido o como en el caso de este proyecto en el cual Mosquitto está instalado en mi ordenador que está encendido todo el día.
La configuración predeterminada de Mosquitto está preparada para no usar la autenticación de nombre de usuario y contraseña, y acepta todas las conexiones en el puerto 1883. El puerto 1883 es un puerto estándar para comunicaciones sin encriptar. Si se desea mayor seguridad en las comunicaciones hay que configurar Mosquitto para que emplee el puerto 8883 ya que utilizauna encriptación **SSL** (Secure Socket Layer, capa de conexión segura).

Para tener instalado Mosquitto en el ordenador, hay que instalar **dtdns** para asignar un nombre al ordenador, puesto que cada vez que se conecta a la red tiene un IP diferente, lo que provoca que el nombre del broker (es la dirección IP) cambie. Instalando **dtdns** en el ordenador y creando una cuenta, se asigna un nombre al ordenador y tanto en el módulo Wi-Fi y la aplicación se pone ese nombre de broker. De esta manera, cada vez que cambie la IP no hace falta modificar el código de la aplicación y el módulo Wi-Fi. Se puede descargar **dtdns** de su página oficial[2], también en la página hay que crear una cuenta de usuario. El nombre que le he asignado a mi ordenador es david2016.dtdns.net.

Otro aspecto que se debe tener en cuenta es que los routers comerciales están configurados para no dejar pasar algunos paquetes, los de MQTT están incluidos entre ellos. Por lo que, se tiene que desmilitarizar el router de la red a la que está conectada nuestro broker. Así, puede recibir todos los paquetes MQTT. En la facultad el router del laboratorio de proyectos ya está desmilitarizado, por lo que no ha hecho falta desmilitarizar el router para este proyecto.

3.2.1. Funcionamiento de Mosquitto sin comunicaciones seguras

Una vez instalado Mosquitto en el ordenador, se abre un terminal y se pone la dirección de la carpeta en la cuál está instalado Mosquitto. A continuación, se ejecuta el siguiente comando para añadir un nuevo usuario:

mosquitto_passwd.exe -c .\mosquitto.pwd Nombre_usuario

Mosquitto pide una contraseña para ese usuario, una vez introducida, vuelve a pedir la contraseña para su confirmación. Este proceso se muestra en la Figura 3.8.
Si ya hay un usuario creado y se quiere acceder a él, hay que escribir el mismo comando sin el -c, ya que esto significa crear usuario.

[1] https://www.mosquitto.org/
[2] https://www.dtdns.com/

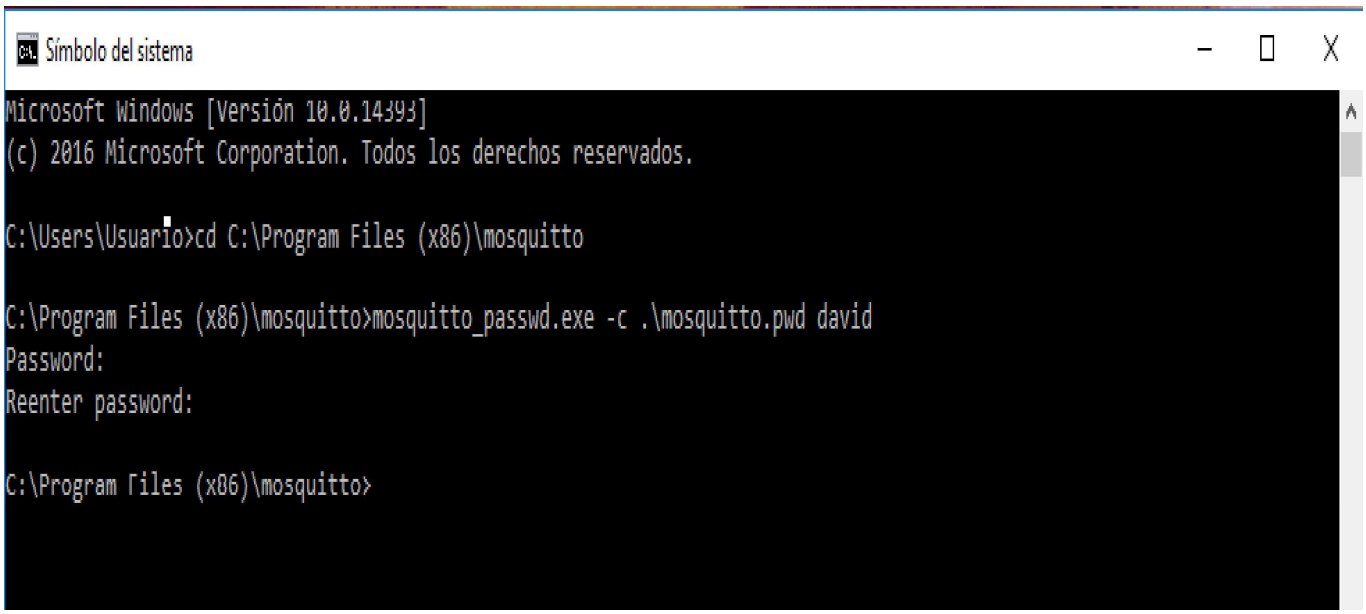


Figura 3.8: Nuevo usuario Mosquitto

A continuación, si se ejecuta el comando:

mosquitto -v -c .\mosquitto.conf

se muestran los puertos que están disponibles para la comunicación, en este caso sólo está disponible el puerto 1883 ya que no se ha configurado Mosquitto para comunicaciones seguras. Los puertos disponibles se muestran en la Figura 3.9.

Figura 3.9: Puertos disponibles para la comunicación

Una vez que se ha creado el usuario y se han visto los puertos disponibles, se abre otro terminal para que haga de cliente suscriptor. Para ello, se direcciona a la carpeta donde está instalado Mosquitto y se ejecuta el siguiente comando:

mosquitto_sub.exe -h host_broker -u Nombre_usuario -P contraseña -t tópico

Si en el comando, después del tópico se añade -v se recibe el nombre del tópico y el mensaje. En la Figura 3.10 se muestra un ejemplo de un cliente que se suscribe al tópico temperatura. Este cliente se queda esperando a que otro cliente publique mensajes con el tópico temperatura.

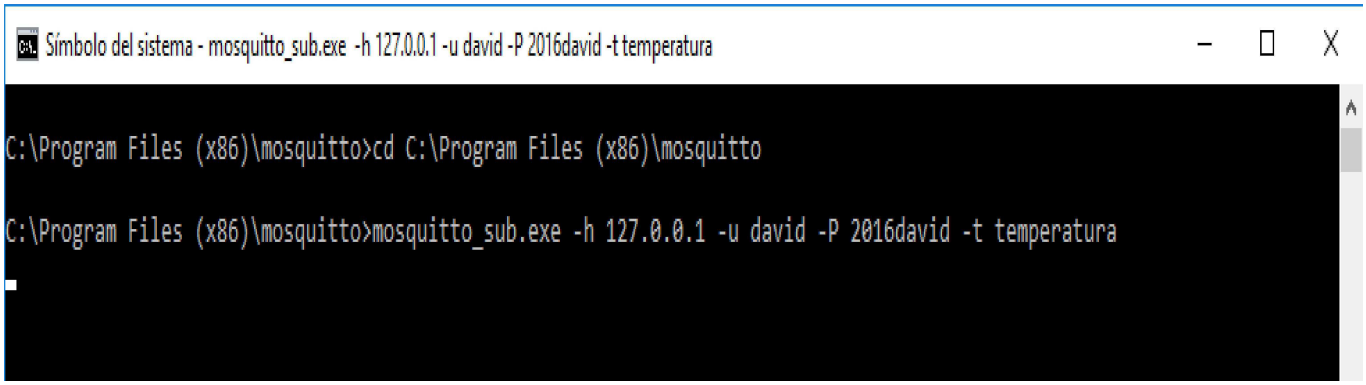


Figura 3.10: Cliente se suscribe a tópico temperatura

Cuando se ejecuta el comando anterior en el cliente, se notifica al broker de que hay un nuevo suscriptor, el puerto que emplea para las comunicaciones (1883 en este caso), el tópico al que se suscrito, la dirección que ha empleado del host, etc. Esto se puede ver en la Figura 3.11.

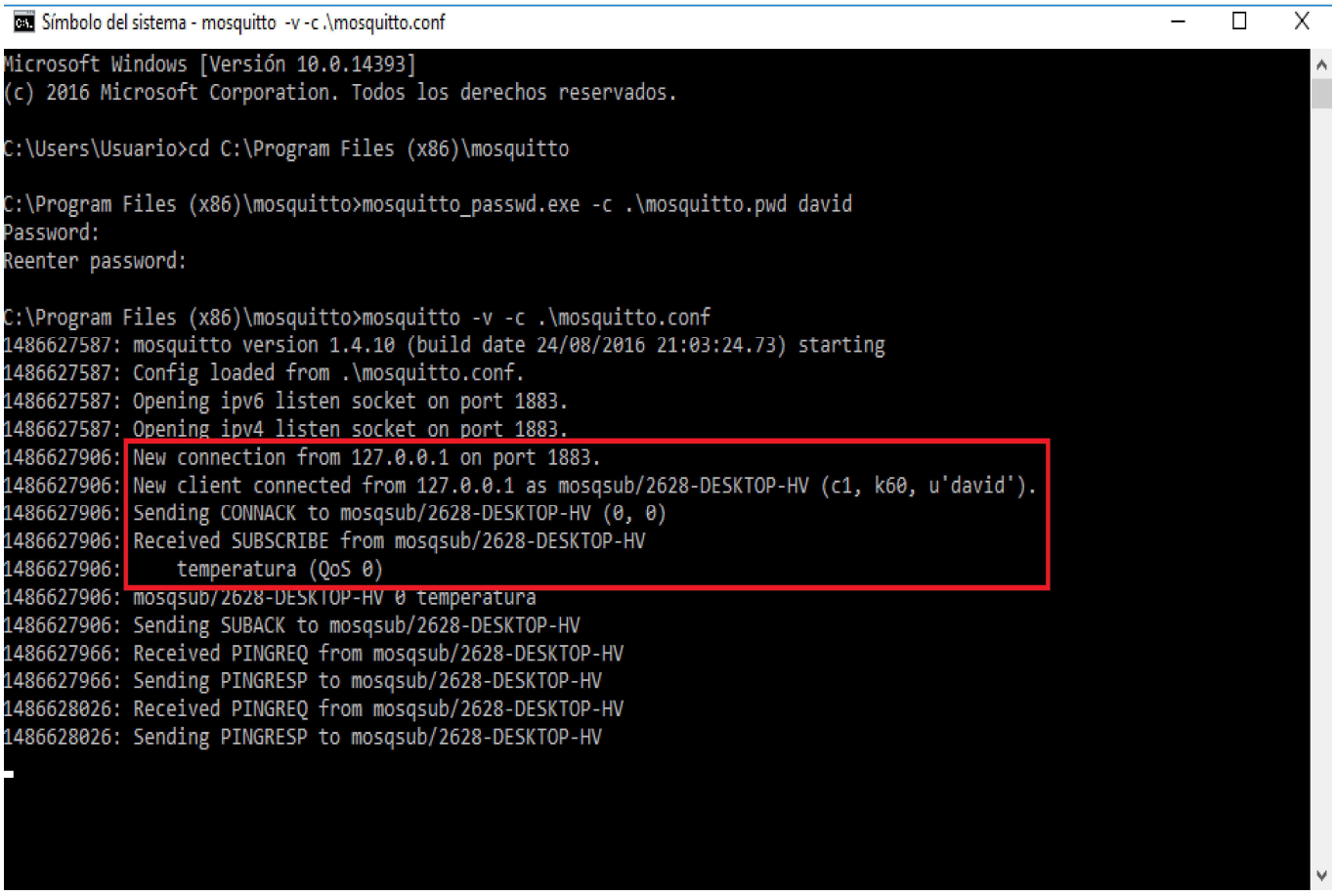


Figura 3.11: Broker cuando se suscribe un cliente

Por último, se abre otro terminal que va a publicar sobre un tópico. Para añadir este cliente, hay que situarse en la carpeta que está instalado Mosquitto y ejecutar el siguiente comando:

mosquitto_pub.exe -h host_broker -u Nombre_usuario -P contraseña -t tópico -m "mensaje"

En la Figura 3.12 se muestra como se configura el cliente que publica, el mensaje publicado pasa por el broker al cual se le notifica que hay un nuevo cliente, el puerto que utiliza, el tópico que publica, el mensaje,etc. esto se puede ver en la Figura 3.13. Después le llega el mensaje al suscriptor como se ve en la Figura 3.14.

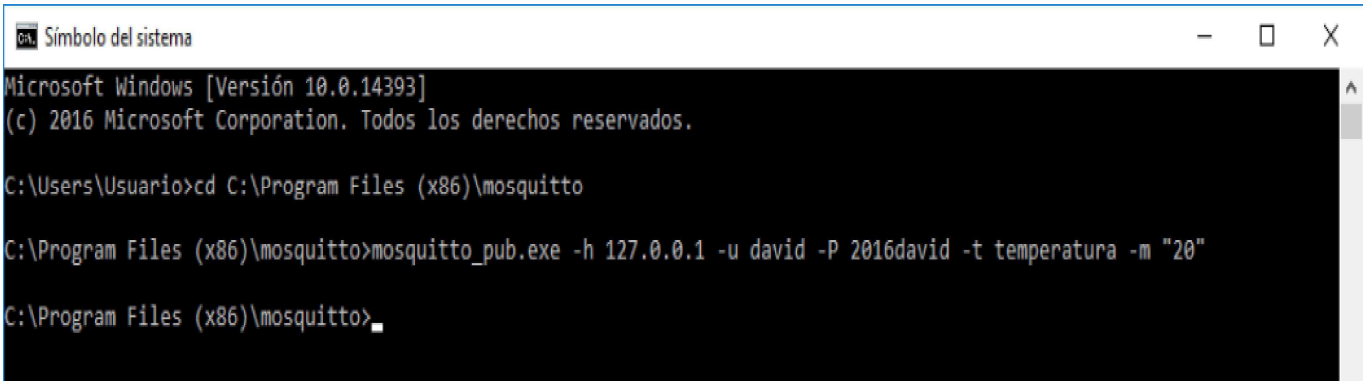


Figura 3.12: Cliente que publica un mensaje con el topic temperatura

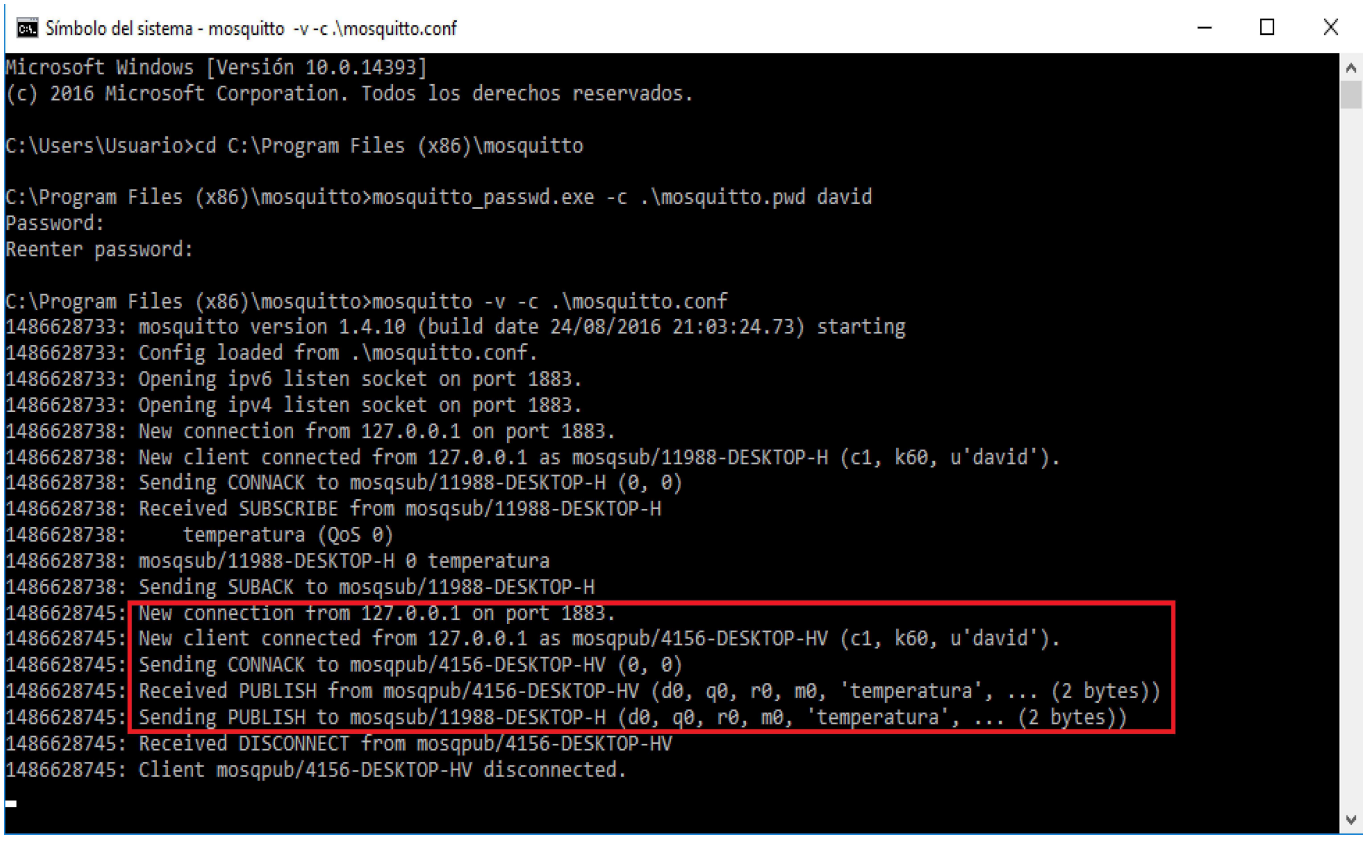


Figura 3.13: Broker cuando un cliente publica un mensaje

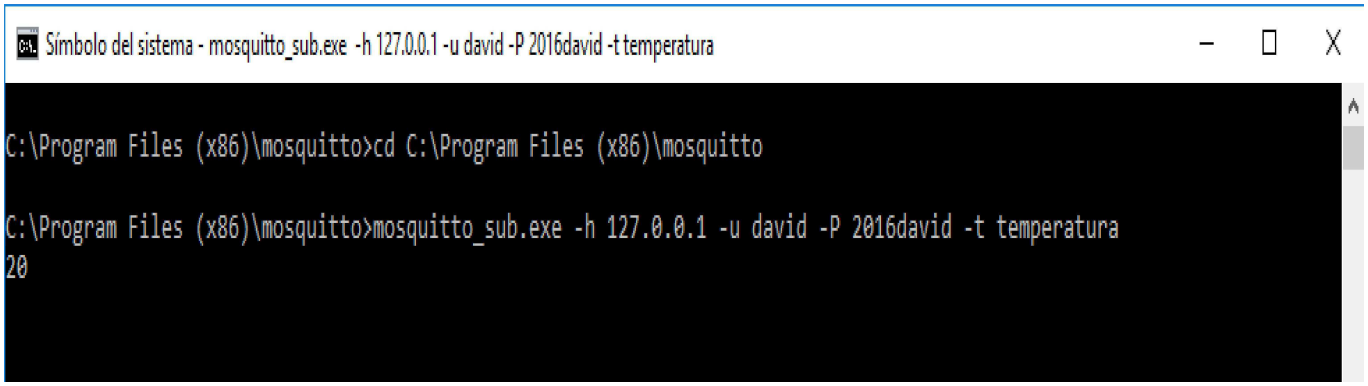


Figura 3.14: Cliente suscriptor cuando otro cliente publica un mensaje

Este proceso que se ha explicado, es un ejemplo ya que en la aplicación del toldo automático los clientes van a ser el módulo Wi-Fi y la aplicación del móvil. Como la finalidad del proyecto es la venta del sistema del toldo, el usuario y la contraseña deberían ser diferentes para cada persona pero para no añadir más complejidad al proyecto se va a utilizar los mismos para todos los clientes.

3.2.2. Configurar Mosquitto para comunicaciones seguras en Windows

Para configurar Mosquitto para comunicaciones seguras, lo primero que se debe hacer es borrar el archivo mosquitto.conf y crear uno nuevo con el siguiente contenido:
listener 1883
listener 8883
cafile E:\Documents\SSL\ca.crt
certfile E:\Documents\SSL\server.crt
keyfile E:\Documents\SSL\server.key

Una vez creado el archivo mosquitto.conf, hay que generar los ficheros ca.crt, server.crt y server.key. Esto se hace debido a que el servidor que aloja Mosquitto requiere un certificado X.509 que deber ser firmado por una CA (Autoridad de Certificación). Para evitar la compra de un certificado y que sea firmado por una CA autentica, se va a generar un certificado raíz .ªuto-firmado"por una CA.
Para realizar este proceso hay que descargar e instalar el software OpenSSL[2]. A continuación, se abre un terminal y se sitúa en la carpeta bin que se encuentra en el directorio donde esta instalado OpenSSL. En primer lugar, se genera el certificado CA con el siguiente comando:

openssl req -new -x509 -days 3650 -keyout ca.key -out ca.crt

con este certificado obtenemos los ficheros ca.key y ca.crt (el que se necesita para mosquitto). Después de ejecutar el comando, pide una frase de paso para encriptar la contraseña y toda la información sobre el certificado (país, estado, etc.). También solicita una serie de datos sobre el sitio desde donde se crea el certificado, estos datos son el país, la localidad, el nombre de la empresa, etc., estos datos son opcionales ya que se pueden saltar pulsando el enter. Todo este proceso se puede ver en la Figura 3.15.

[2]https://www.openssl.org/

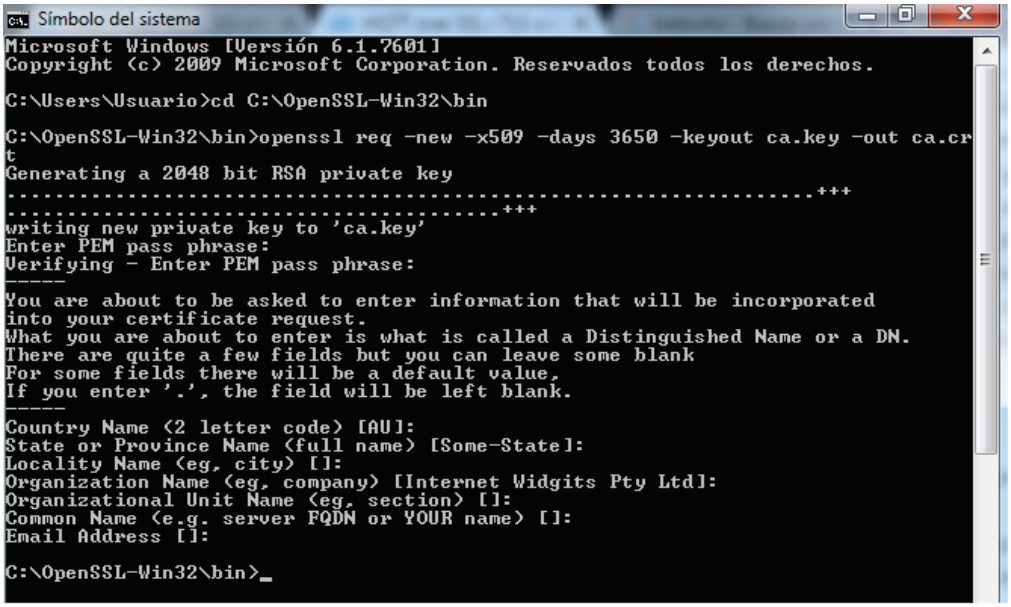


Figura 3.15: Generación certificado CA

A continuación, hay que generar el fichero server.key, el cual contiene una contraseña para el servidor. Esto se hace ejecutando el siguiente comando:

openssl genrsa -des3 -out server.key 1024

Para que la contraseña esté encriptada se necesita la frase de paso que se ha introducido anteriormente. En la Figura 3.16, se muestra la ejecución de este comando.

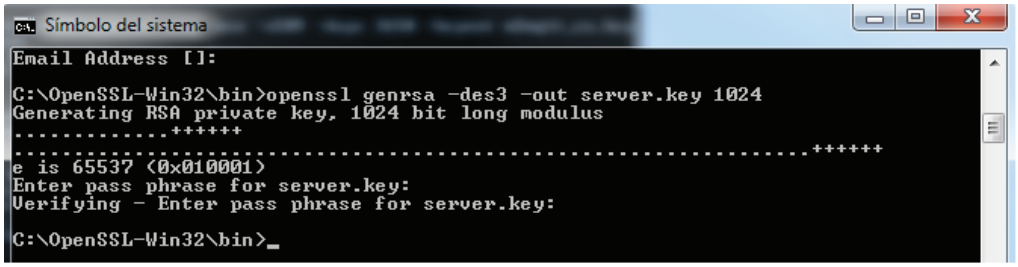


Figura 3.16: Generación fichero server.key

Una vez hecho esto, se pasa a generar la solicitud de certificado del servidor para que sea firmado por la CA. Para ello se ejecuta el siguiente comando:

openssl req -out server.csr -key server.key -new

De esta manera se genera la solicitud en el archivo server.csr que contiene toda la información del servidor. En este caso también se pide la información sobre el país, nombre de la empresa,etc, esto hay que rellenarlo con los mismos datos que se han puesto anteriormente, en nuestro caso esta información se saltado con el enter. En la Figura 3.17, se muestra este proceso en el cual se puede ver como e solicita la frase de paso para encriptar la información.

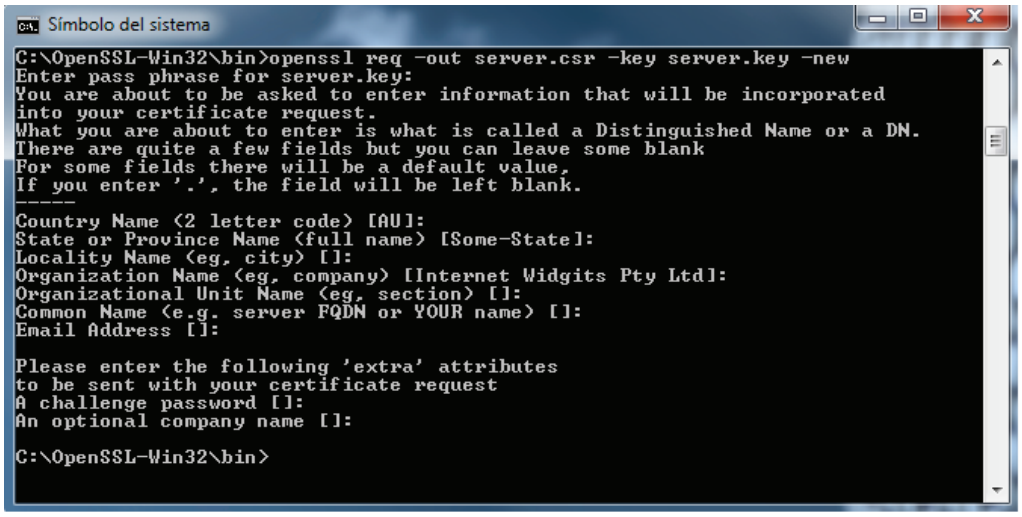


Figura 3.17: Generación de la solicitud de certificado

El último paso es firmar la solicitud y obtener el certificado final del broker. Para ello, hay que ejecutar el siguiente comando:

openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days 3650

Con esto se consigue el fichero que faltaba por generar: server.crt. En la Figura 3.18 se muestra este proceso.

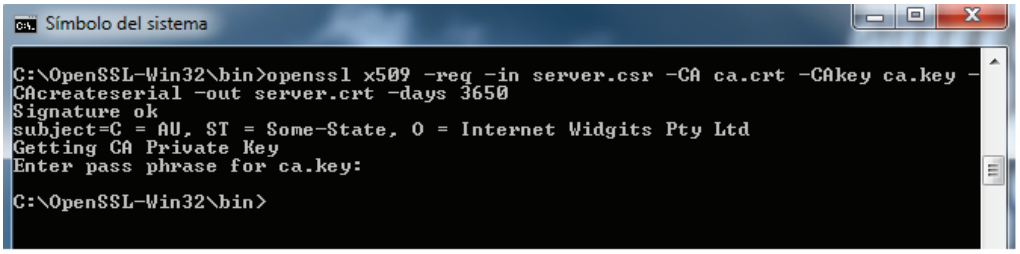


Figura 3.18: Generación fichero server.crt

Por último, se copian los ficheros ca.crt, server.crt y server.key y se pegan en la carpeta donde esta instalado Mosquitto. De esta manera, se tiene configurado Mosquitto para comunicaciones seguras.

3.2.3. Funcionamiento de Mosquitto con comunicaciones seguras

En el apartado anterior, se ha explicado como configurar Mosquitto para comunicaciones seguras. Ahora se va a mostrar un ejemplo de como funciona Mosquitto con esta configuración. Para ello, se va a crear un usuario, un cliente se va a suscribir a un topic y otro cliente va a publicar un mensaje con ese topic.
El primer paso es crear un usuario, por lo que hay que abrir un terminal y colocarlo en la carpeta donde esta instalado Mosquitto. Después se ejecuta el siguiente comando:
mosquitto_passwd.exe -c .\mosquitto.pwd Nombre_usuario

Una vez se ejecuta el comando, hay que introducir una contraseña para ese usuario. Este proceso es el mismo que el de la Figura 3.8 que se vio en el ejemplo del funcionamiento de Mosquitto sin configurar.A continuación, se comprueban que puertos están disponibles ejecutando el comando:

mosquitto -v -c .\mosquitto.conf

En este caso, están disponibles los puertos 1883 y 8883, además pide la frase de paso que se ha utilizado en la generación del certificado.En el ejemplo de Mosquitto sin configurar sólo estaba disponible el puerto 1883 y no pedía frase de paso. En la figura 3.19 se muestran los puertos disponibles.

Figura 3.19: Puertos disponibles para la comunicación segura

El siguiente paso a realizar, es abrir un terminal para que haga la función de cliente suscriptor. Se direcciona el terminal a la carpeta que está instalado Mosquitto y se ejecuta el comando:

mosquitto_sub.exe -h host_broker -u Nombre_usuario -P contraseña -p 8883 -t tópico –cafile E:\Documents\SSL\ca.crt

En la Figura 3.20 se muestra como el cliente se suscribe al topic temperatura. Mientras que en la Figura 3.21 se muestra el broker, al cual se le notifica que tiene un nuevo cliente y por que puerto se ha conectado.

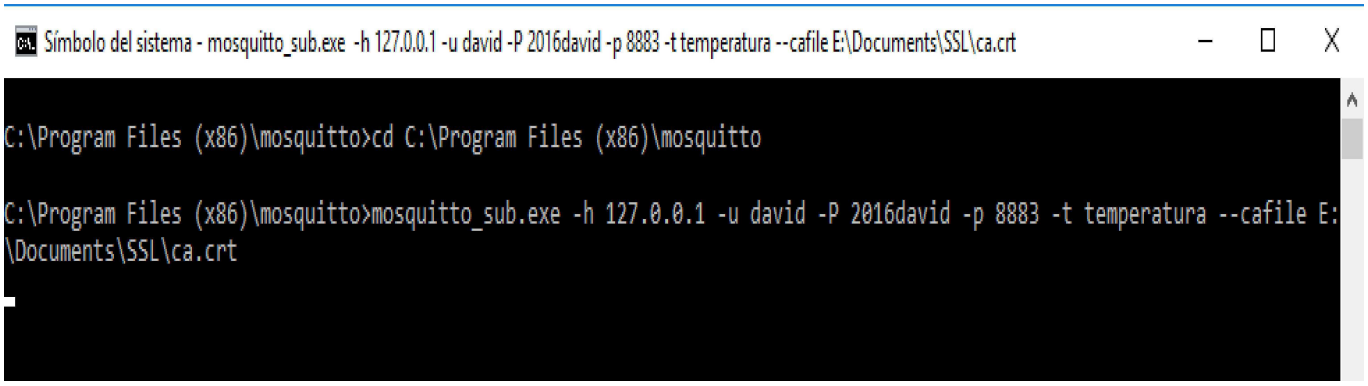


Figura 3.20: Cliente se suscribe a tópico temperatura por el puerto 8883

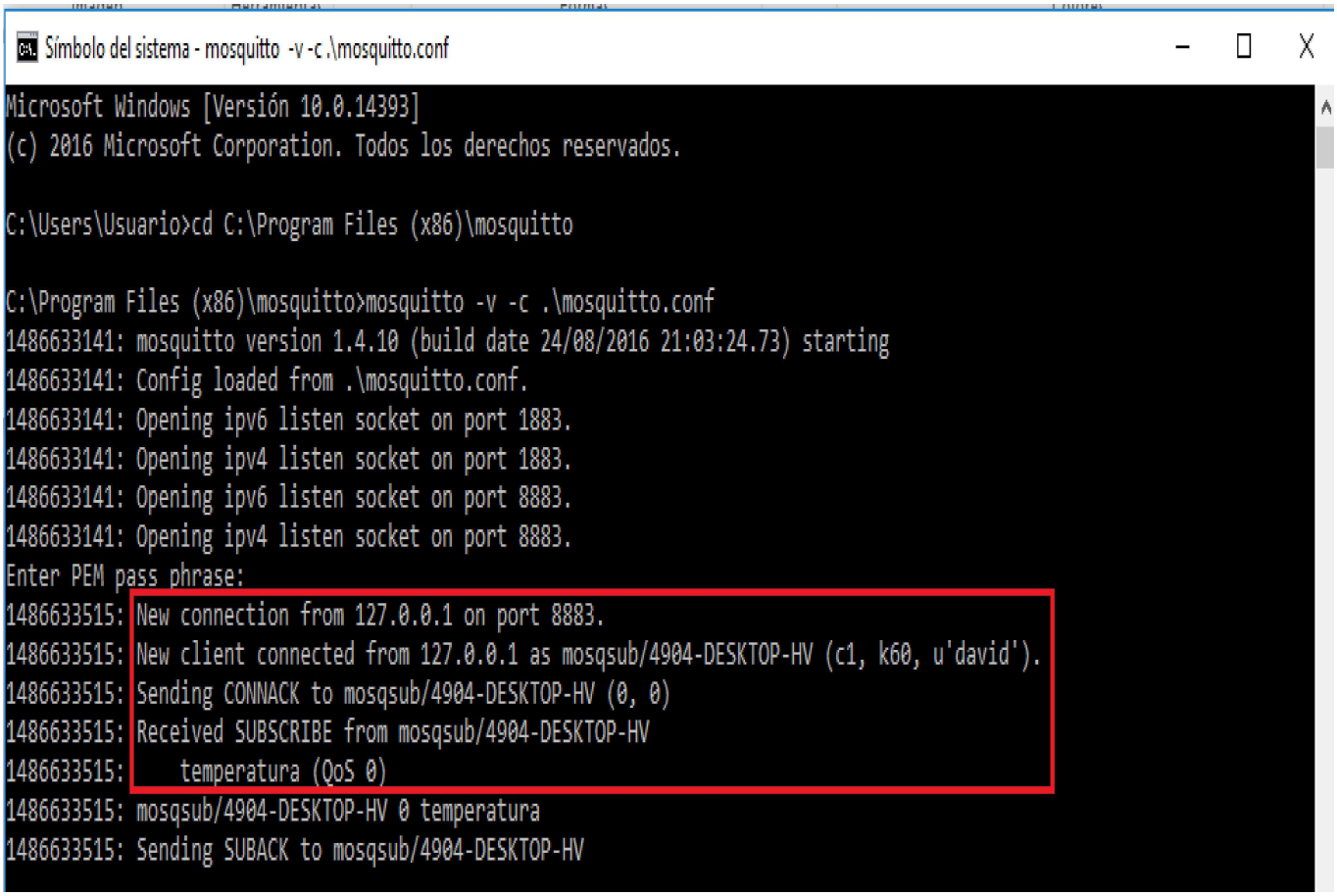


Figura 3.21: Broker cuando se suscribe cliente por el puerto 8883

Por último, se abre otro terminal para el cliente publicador y se escribe el siguiente comando:

mosquitto_pub.exe -h host_broker -u Nombre_usuario -P contraseña -p 8883 -t tópico -m "mensaje-cafile E:\Documents\SSL\ca.crt

En la Figura 3.22, se muestra como el cliente publica el mensaje. En la Figura 3.23, se muestra la conexión del publicador al broker, donde se ve el puerto por el que se conecta y el mensaje que quiere publicar. Mientras que en la Figura 3.24, se puede ver como el suscriptor recibe el mensaje.

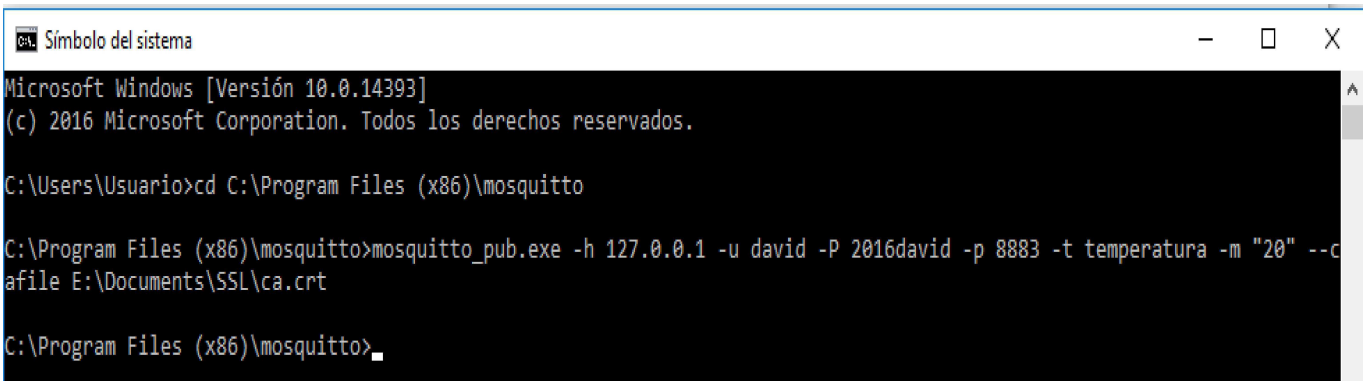


Figura 3.22: Broker cuando se suscribe cliente por el puerto 8883

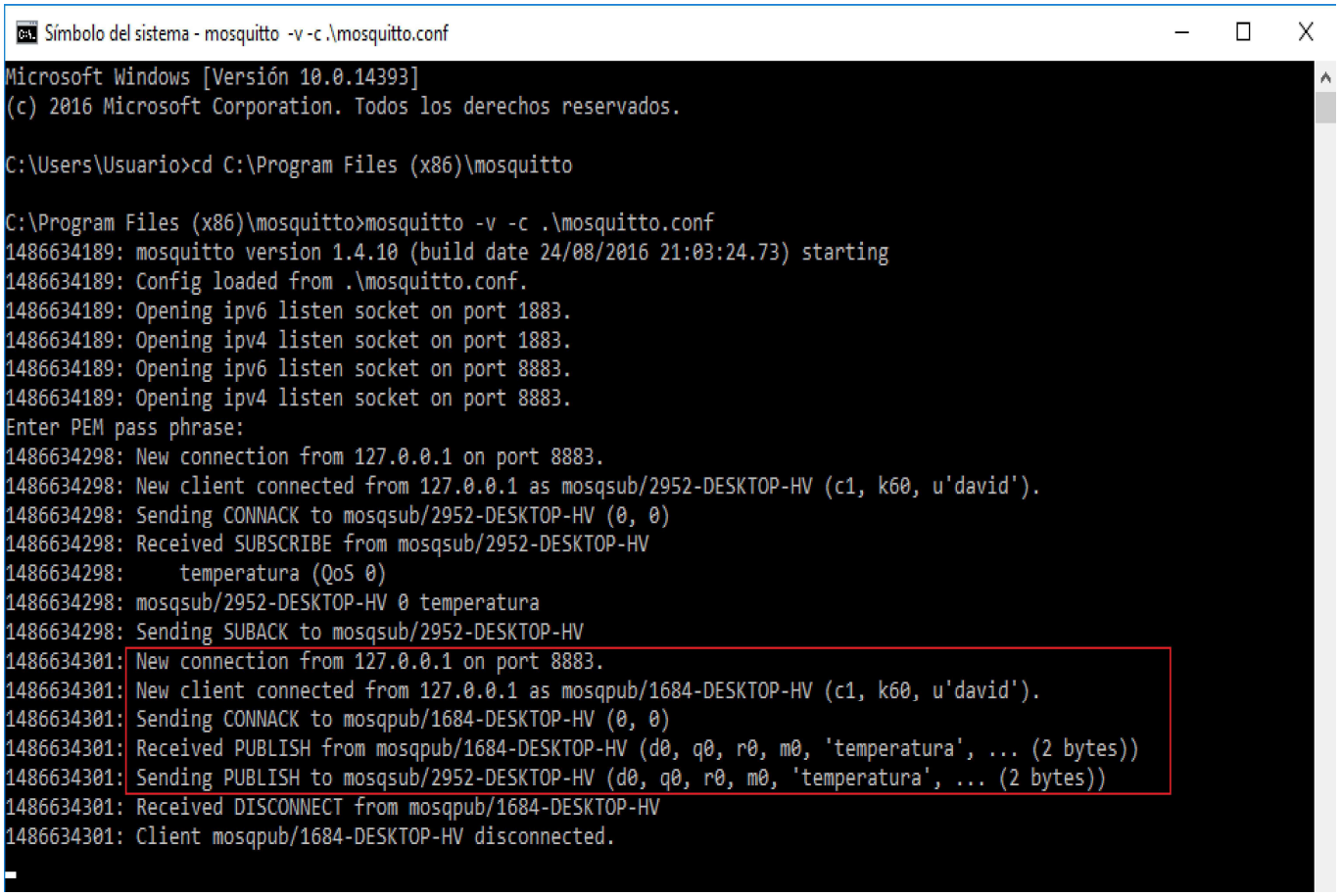


Figura 3.23: Broker cuando se suscribe cliente por el puerto 8883

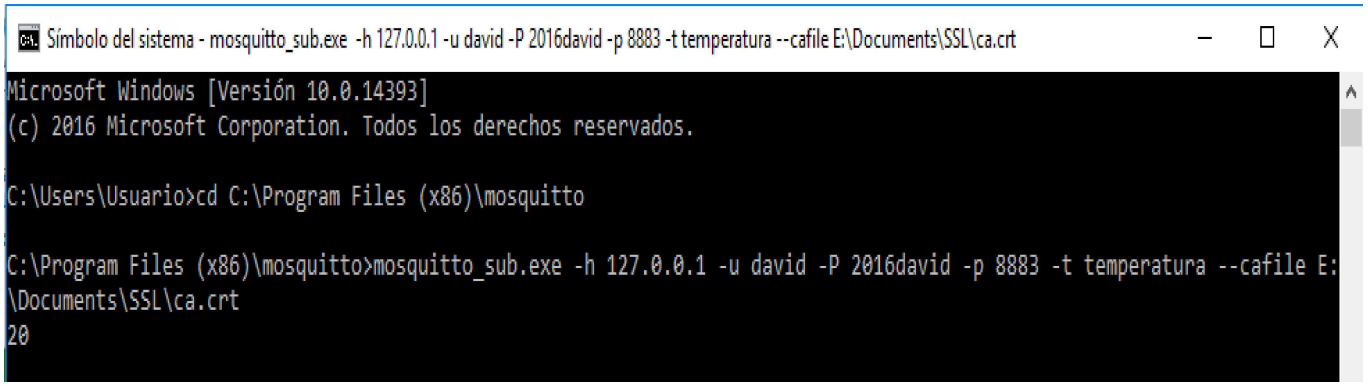


Figura 3.24: Broker cuando se suscribe cliente por el puerto 8883

3.3. Aplicación

3.3.1. Android Studio

Para la programación de la aplicación se ha utilizado el entorno de desarrollo Android Studio.
Este IDE (Entorno de Desarrollo Integrado) fue anunciado el 16 de mayo de 2013 en la conferencia de Google I/O. Pero hasta diciembre de 2014 no fue publicada una versión definitiva.
Cuando se lanzó Android Studio, sustituyó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android.
Está basado en el software IntelliJ IDEA de la compañía JetBrains frente al plugin ADT que emplea Eclipse. Además, está disponible para los sistemas operativos Windows, MAC OS X y Linux.

Características

Android Studio incluye mejoras respecto a Eclipse. Debido a esto, es el entorno de desarrollo empleado en la actualidad. Las principales características que añade respecto a Eclipse son las siguientes:

- Incorpora la herramienta **Gradle**, que se encarga de gestionar y automatizar la construcción de proyectos, como puede ser las tareas de complicación, testing y empaquetado.
- Integra **Instant Run**, que permite aplicar cambios mientras la aplicación se ejecuta sin la necesidad de compilar un nuevo **apk**.
- Permite la importación de proyectos realizados en Eclipse.
- Herramientas **Lint** para detectar problemas de rendimiento, uso, compatibilidad de versión, etc.
- Emplea **ProGuard** para optimizar y reducir el código del proyecto al exportar al **apk**.
- Posibilita la integración de plantillas de código y GitHub.
- Incorpora un soporte para programar aplicaciones para Android Wear (sistema operativo para dispositivos corporales como los smartcwatch).
- Tiene compatibilidad con C++ y NDK.
- Soporte integrado para **Google Cloud Platform**. De esta manera facilita el acceso a los servicios que proporciona Google en la nube.
- Añade más tipos de dispositivos y resoluciones para su vista previa en el emulador.
- Incorpora un editor de diseño que permite visualizar y modificar el layout.

Instalación

En primer lugar, hay que descargar Android Studio de la página oficial [3]. Durante su instalación, hay que marcar la opción de Android SDK, ya que hay que instalarlo.

Seguidamente, hay que instalar las librerías de Java JDK(*Java Development Kit, Kit de Desarrollo de Java*) en su versión 7 o superior. Estas librerías son imprescindibles para ejecutar el emulador de Android y algunas herramientas de depuración. Para poder utilizar todas las herramientas de desarrollo que ofrece Android Studio, hay que instalar el JDK completo.
Algunas páginas web en las que se explica la instalación de Android Studio, solamente indican que hay que instalar Java JRE (*Java Runtime Edition*) en vez del JDK. El JRE se utiliza para elementos de

[3]Se puede descargar aquí: https://developer.android.com/studio/index.html

Internet. Sin embargo, no incluye todas las herramientas disponibles en Android Studio como sucede con el JDK. Además, el JRE está incluido en el JDK.
Por tanto, lo recomendable es instalar las librerías de Java JDK. Su descarga se puede realizar en la página de Oracle.[4].

Estructura de un proyecto

En primer lugar, se debe crear el proyecto. Al crear un proyecto, lo primero que pide es el nombre del proyecto, el dominio de la compañía y la localización del proyecto en el ordenador. Todo esto se puede apreciar en la Figura 3.25.

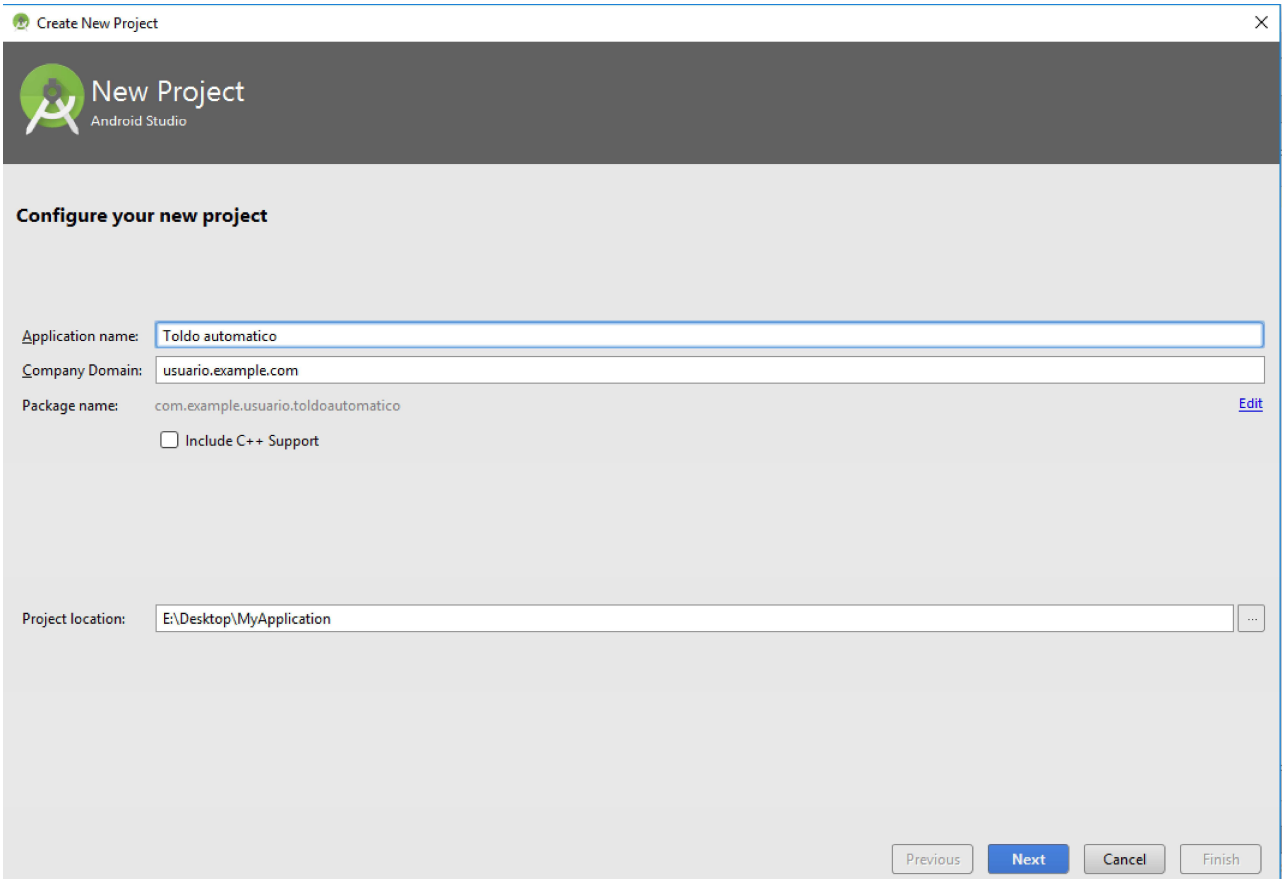


Figura 3.25: New Project

Rellenados estos datos, pinchamos en Next y nos aparece una ventana como la que se muestra en la Figura 3.26. En esta ventana, hay que seleccionar el tipo de dispositivo para el cuál va a ser nuestra aplicación (teléfonos y tablets, relojes, televisiones, gafas). Además, se debe seleccionar la API (*Application Programming Interface*) más baja que soportará la aplicación. En este caso se ha seleccionado la API 10 que corresponde a Android 2.3.3 (GingerBread).

[4]Se puede descargar aquí:http://www.oracle.com/technetwork/es/java/javase/downloads/index.html

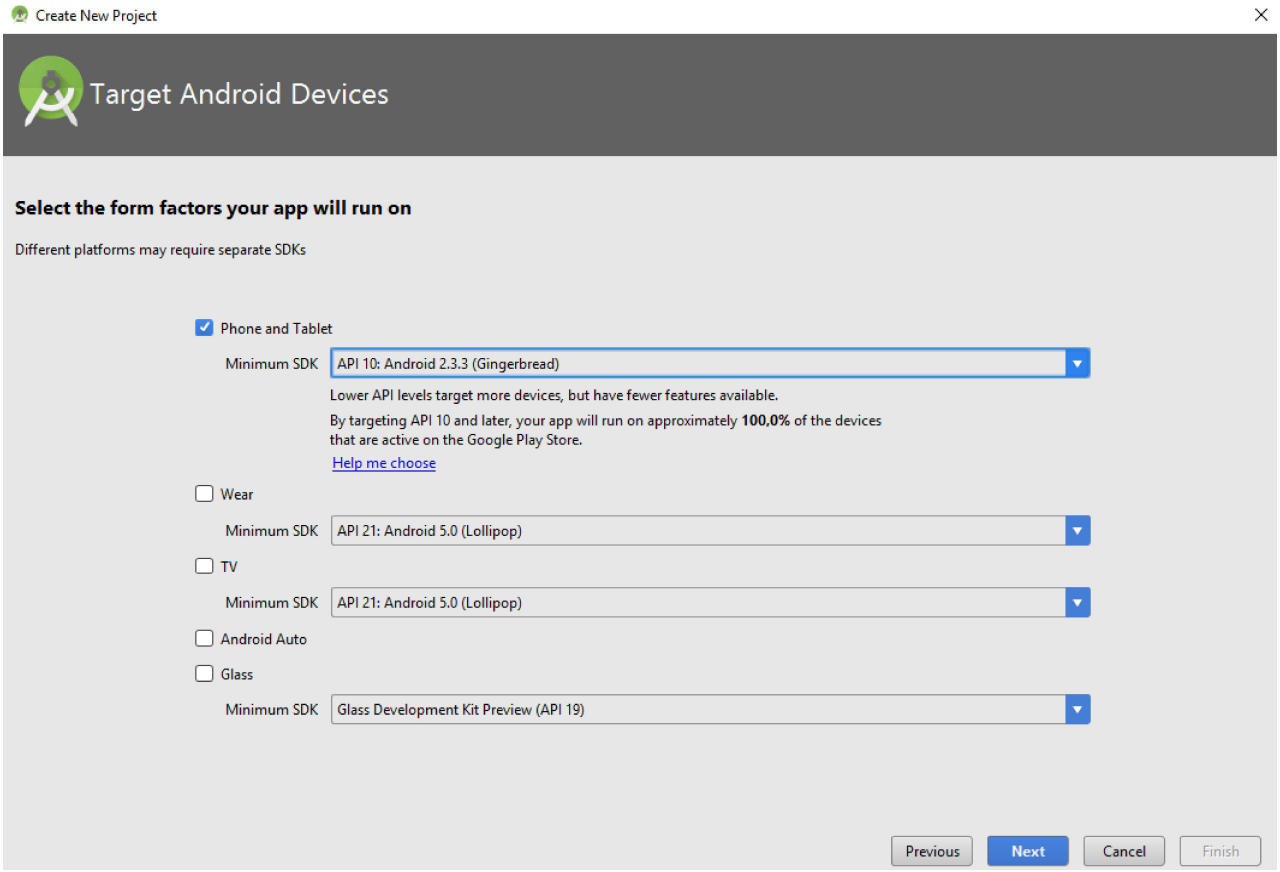


Figura 3.26: Target Android Devices

Una vez seleccionada la mínima API que soporta la aplicación, se debe seleccionar una plantilla por defecto para el interfaz de la aplicación. Las plantillas por defecto de las que dispone Android Studio se muestran en la Figura 3.27. En nuestro caso se ha escogido la tercera, que es una plantilla vacía con una barra en la parte superior.

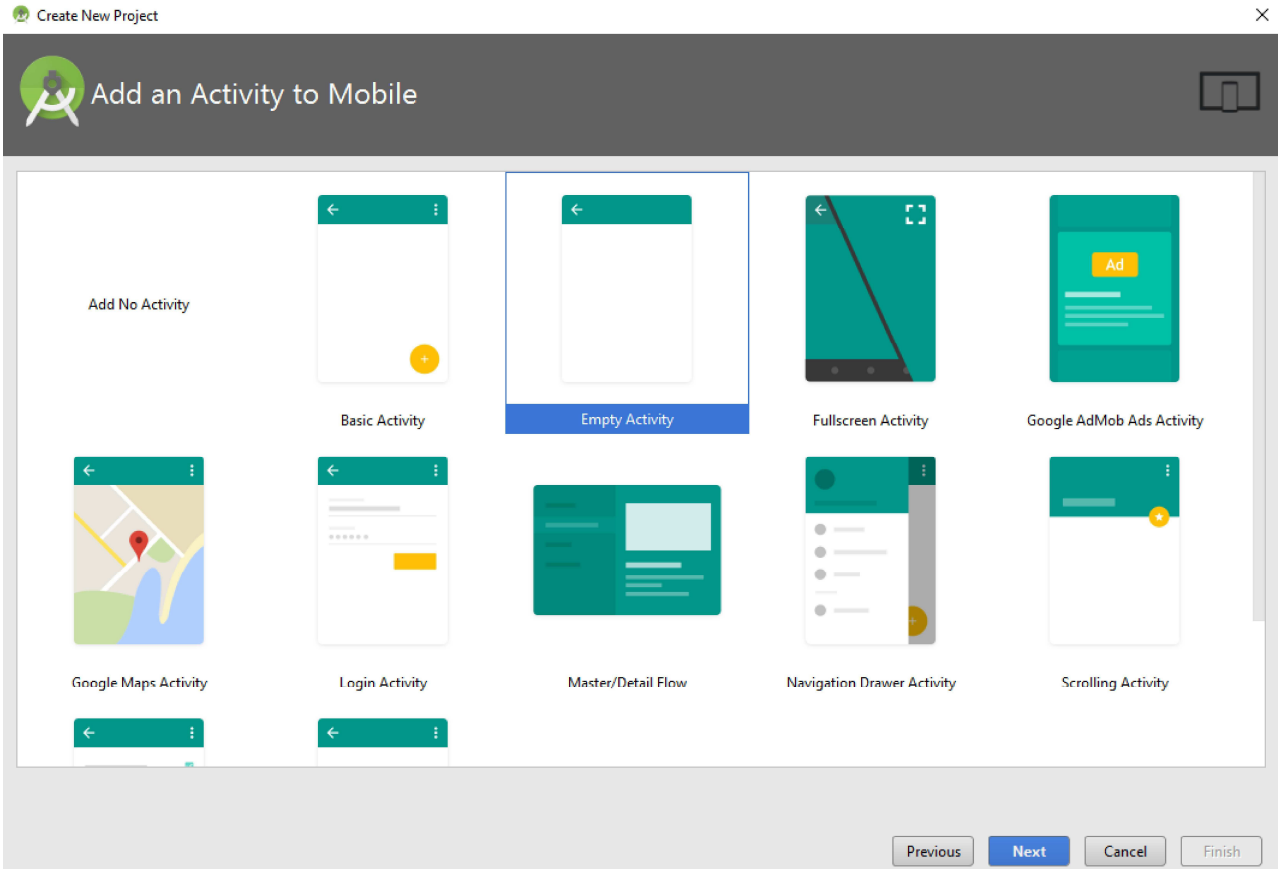


Figura 3.27: Add an Activity to mobile

Por último, nos aparee una ventana que muestra el Layout seleccionado y nos permite cambiar el nombre de la Activity principal y del Layout. En este caso, hemos dejado los nombres por defecto. Esta ventana se muestra en la Figura 3.28.

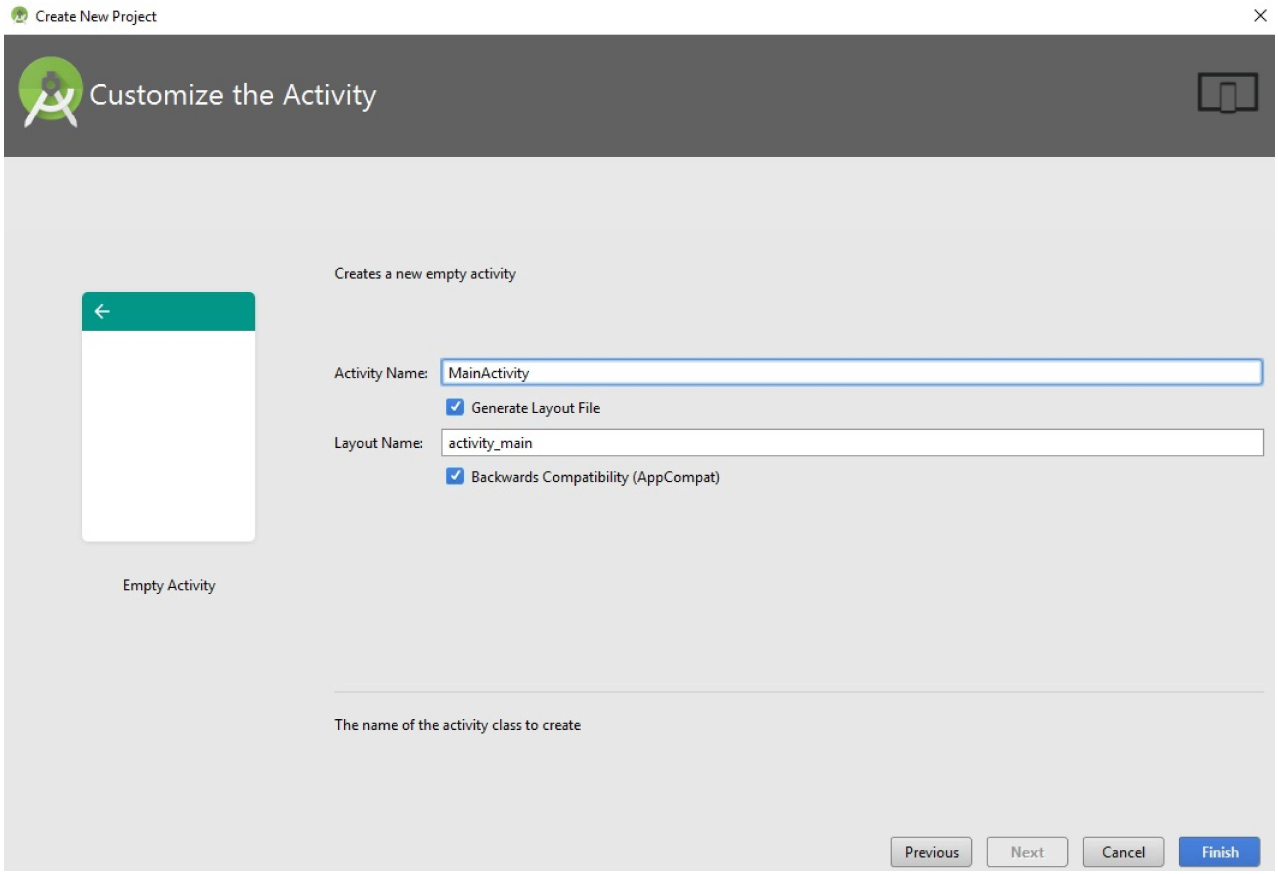


Figura 3.28: Customize the Activity

Por último, pinchamos en Finish y se crea el proyecto con la configuración seleccionada.
Una vez creado el proyecto, vamos a explicar brevemente su estructura y sus partes principales.
Cada proyecto en Android Studio contiene uno o más módulos con archivos de código fuente y archivos de recursos. Entre los tipos de módulos se incluyen los siguientes:

- Módulos de apps para Android
- Módulos de bibliotecas
- Módulos de Google App Engine

En la Figura 3.29, se muestran los módulos que tiene nuestro proyecto. El más interesante es el módulo **app** ya que contiene la carpeta **scr** y está a su vez la carpeta **main** donde se encuentra el código de nuestra aplicación, ficheros de descripción y recursos utilizados.

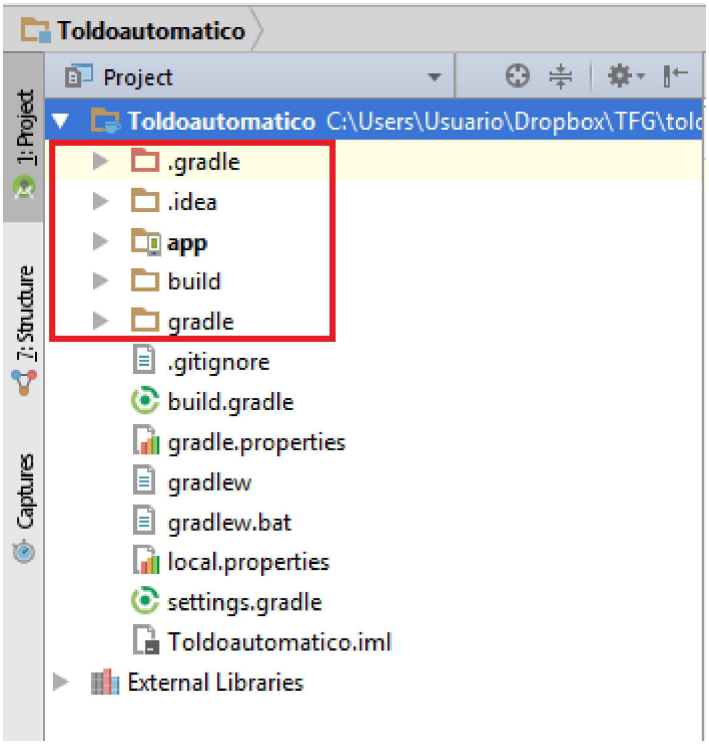


Figura 3.29: Módulos proyecto

La carpeta **main** contiene otras carpetas y archivos importantes de nuestro proyecto. En la Figura 3.30 se muestra su contenido.

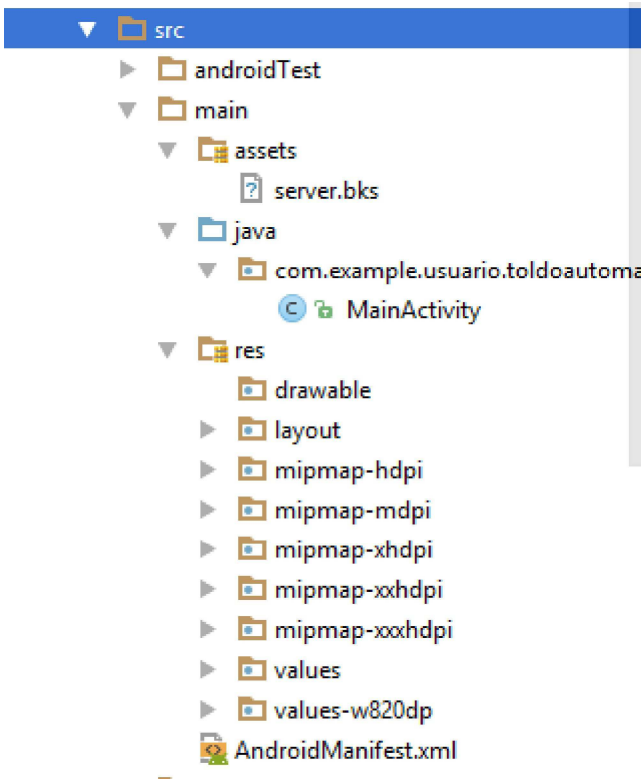


Figura 3.30: Contenido carpeta **main**

A continuación, vamos a describir brevemente las carpetas y archivos que contiene:

- **Carpeta assets**: esta carpeta no se crea por defecto. La hemos creado ya que el archivo ser-

ver.bks es necesario para la comunicación por el puerto 8883.

- **Carpeta java**: contiene el código fuente de nuestra aplicación. En este caso, solamente tiene una clase (MainActivity) ya que no se han creado más.
- **Carpeta res**: contiene todos los ficheros de recursos necesarios para el proyecto: imágenes, layouts, cadenas de texto, etc. Esta carpeta se divide en subcarpetas que se clasifican en función de los recursos.
- **AndroidManifest.xml**: este fichero describe la aplicación. En el se declaran los permisos que necesita la aplicación para su correcto funcionamiento Además, se indica la versión mínima de Android para poder ejecutarla, la versión de la aplicación, el nombre de la aplicación, el icono que empleará, etc.

Otras carpetas y archivos que son importantes en nuestro proyecto y se encuentran en el módulo **app**, son los siguientes (ver Figura 3.31):

- **Carpeta build**: contiene varios ficheros de código que se generan automáticamente cada vez que se compila el programa.
- **Carpeta libs**: contiene las librerías externas de java que se desean añadir, puesto que son necesarias para el desarrollo de nuestra aplicación. En este caso, hemos utilizado la librería Mqtt Paho.
- **build.gradle**:contiene la información necesaria para la compilación del proyecto, como es el caso de las referencias a librerías externas utilizadas, la versión del SDK de Android empleada para compilar, la mínima versión de Android que soporta nuestra aplicación, etc.

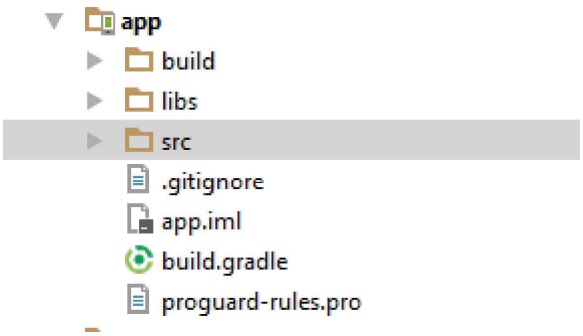


Figura 3.31: Otros ficheros módulo **app**

Por último, explicaremos brevemente como es la interfaz de usuario en Android Studio. En la Figura 3.32, se muestra la ventana principal de Android Studio y una serie de elementos marcados con números que describimos a continuación:

1. **Barra de herramientas**: permite realizar una gran variedad de acciones, como la ejecución y la depuración de la aplicación.
2. **Barra de navegación**: ayuda a explorar el proyecto y abrir archivos para editar. Proporciona una vista más compacta que la estructura visible en la ventana Project.
3. **Ventana de editor**: es el área en la que se puede crear y modificar código.
4. **Ventanas de herramientas**: permiten acceder a tareas específicas, como la administración de proyectos, la búsqueda y los controles de versión, entre otras.

5. **Barra de estado**: muestra el estado del proyecto y el IDE, además de advertencias o mensajes.

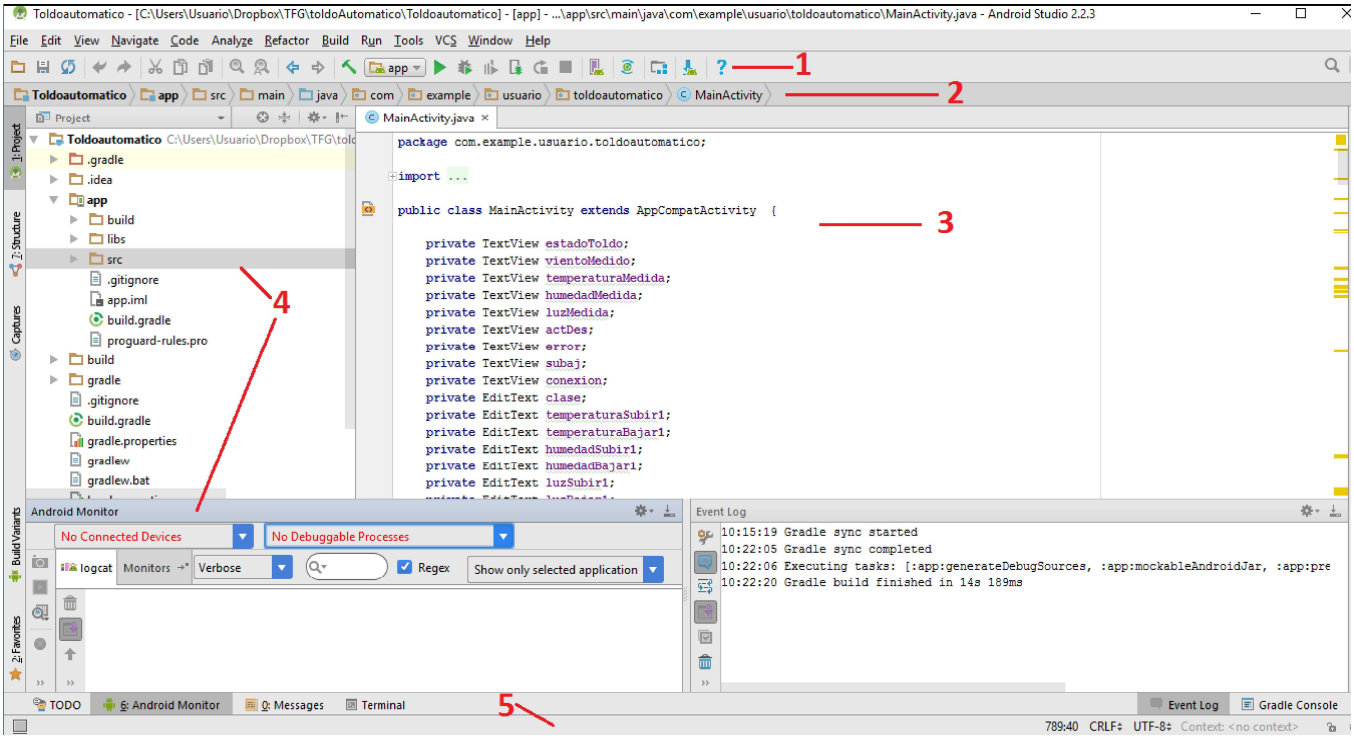


Figura 3.32: Interfaz de usuario

3.3.2. Aplicación Toldo automático

Programación

Para el desarrollo de la aplicación, se ha utilizado la librería de java Mqtt Paho[5]. Una vez descargada la librería, hay que pegarla los ficheros .jar dentro de la carpeta de la aplicación, en concreto, en la carpeta app/libs.
Seguidamente, dentro de Android Studio, se debe añadir:
maven {
url "https://repo.eclipse.org/content/repositories/paho-releases/"
}
en el archivo build.gradle, en la función buildsript dentro de repositories, esto se puede ver en la Figura 3.33. Además, este archivo se muestra al completo en el Apéndice I.

[5]Se puede descargar aquí: https://github.com/eclipse/paho.mqtt.java/tree/master/org.eclipse.paho.client.mqttv3

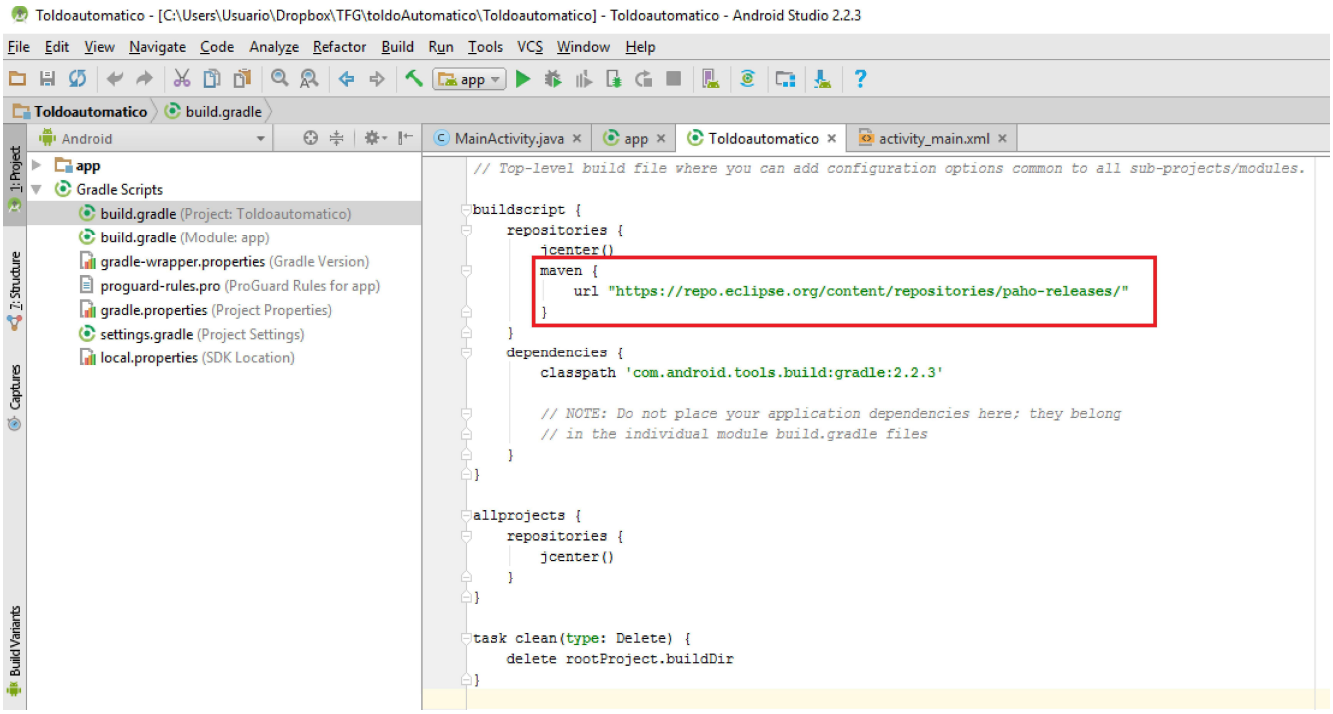


Figura 3.33: Archivo build.gradle

Después, se debe declarar el servicio Paho Android en el archivo AndroidManifest.xml. Dentro de la etiqueta <application>hay que pegar:
*<service android:name="org.eclipse.paho.android.service.MqttService"> </*service>
Además, el servicio Paho Android necesita unos permisos para trabajar. En el archivo AndroidManifest.xml, hay que añadir antes de la etiqueta <application>:
<uses-permission android:name="android.permission.WAKE_LOCK"/ >
<uses-permission android:name="android.permission.INTERNET"/ >
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/ >
<uses-permission android:name="android.permission.READ_PHONE_STATE"/ >
En la Figura 3.34, se muestra donde se añaden estas líneas en el archivo AndroidManifest.xml. Además, en el apéndice J se incluye el archivo AndroidManifest.xml.

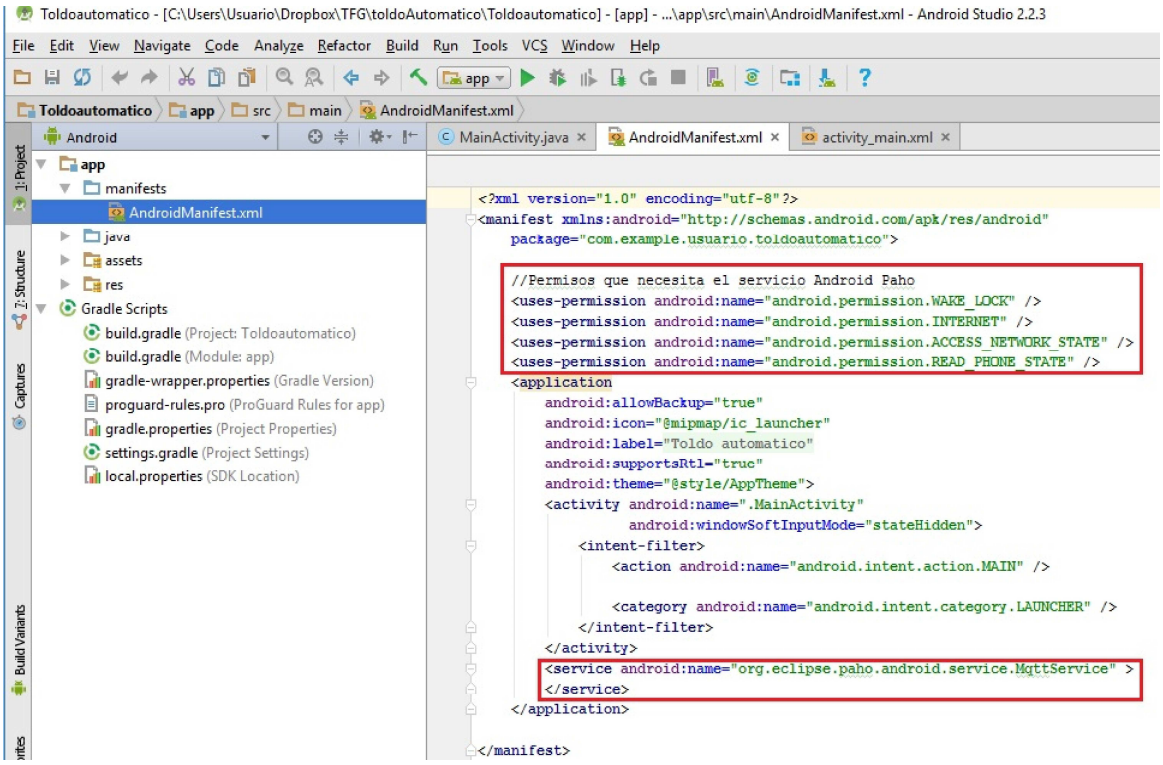


Figura 3.34: Archivo AndroidManifest.xml

Una vez se han configurado estos archivos, se puede programar la aplicación utilizando los métodos de la librería Mqtt Paho.
Vamos a explicar brevemente cómo se ha programado. El código completo se puede ver en el Apéndice K.
Destacar que todo el código se ha escrito en la clase **MainActivity**. Dentro de esta clase, el método principal es **onCreate** que se ejecuta siempre que se inicia la aplicación.
En este método, lo primero que se hace es conectarse al broker por el puerto 8883, para ello se utilizan las clase y métodos adecuados de la librería Mqtt Paho. Para establecer la conexión en modo seguro hay que descargar la librería BouncyCastle [6]. El .jar descargado hay que pegarlo en la carpeta donde se tenga instalado java en el ordenador. Una vez hecho esto, se abre un terminal y se direcciona a esa carpeta y se ejecuta el siguiente comando:

keytool -import -alias david2016.dtdns.net -file server.crt -keypass 2016david -keystore server.bks -storetype BKS -storepass 2016david -providerClass org.bouncycastle.jce.provider.BouncyCastleProvider -providerpath bcprov-ext-jdk14-1.53.jar

El resultado de eso es un archivo llamado **server.bks**. Después en el directorio de la aplicación en la carpeta scr/main, hay que crear una carpeta que se llama **assets** y dentro de ella pegar el archivo **server.bks**. De esta manera, se tiene la aplicación configurada para poder establecer comunicación con el broker por el puerto 8883.

El siguiente paso, es comprobar si se ha conectado al broker. En caso afirmativo, se llama al método **subscribeTopic** tantas veces como tópicos hay para suscribirse. En este método se suscribe a un tópico con calidad de servicio 1. Los métodos a loas que se suscribe son:

- david/toldo/Inicial1

[6]Se descarga aquí: http://repo2.maven.org/maven2/org/bouncycastle/bcprov-ext-jdk14/1.53/bcprov-ext-jdk14-1.53.jar

- david/toldo/estadoToldo
- david/toldo/viento
- david/toldo/temperatura
- david/toldo/humedad
- david/toldo/luz
- david/toldo/Clase
- david/toldo/TemperaturaUmbrales
- david/toldo/HumedadUmbrales
- david/toldo/LuzSubir1
- david/toldo/LuzBajar1
- david/toldo/modoVacaciones
- david/toldo/Horas
- david/toldo/errorSuscripcion
- david/toldo/Cambiar

Seguidamente, se llaman a los métodos de publicar como son **publishClass**, **publishTemperaturaSubir**,**publishTemperaturaBajar**, **publishHumedadSubir**,**publishHumedadBajar**, **publishLuzSubir**,**publishLuzBajar**, **publishModoVacaciones**,**publishHora1**, **publishHora2** y **publishCambiarEstadoToldo**. Los tópicos que publican son:

- david/toldo/Clase
- david/toldo/TemperaturaUmbrales
- david/toldo/HumedadUmbrales
- david/toldo/LuzSubir1
- david/toldo/LuzBajar1
- david/toldo/modoVacaciones
- david/toldo/Horas
- david/toldo/Cambiar

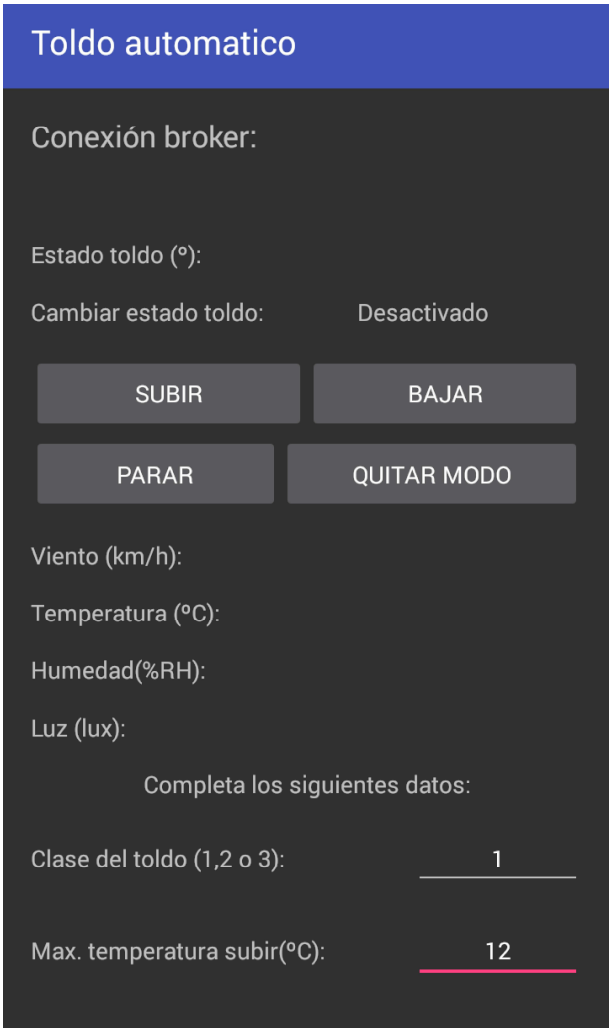
En estos métodos se recoge el valor del editText y se publica en el servidor. Para ello, emplea el método **publish** de la librería de Mqtt Paho y se configura para que los mensajes se retengan en el broker.

Por último, se ha implementado el método **onSaveInstanceState** que guarda el valor de todas las variables, y el método **onRestoreInstanceState** que restaura el valor de las variables. Estos métodos se ejecutan cuando se destruye la actividad, esto ocurre al girar la pantalla del móvil o dejando la aplicación en segundo plano y volviéndola abrir. Utilizando estos métodos no se pierden los valores de las variables y se pueden volver a mostrar correctamente. Ya que sin estos métodos, habría que reiniciar la aplicación y en ella aparecerían los valores por defecto y el toldo tendría los últimos valores publicados.

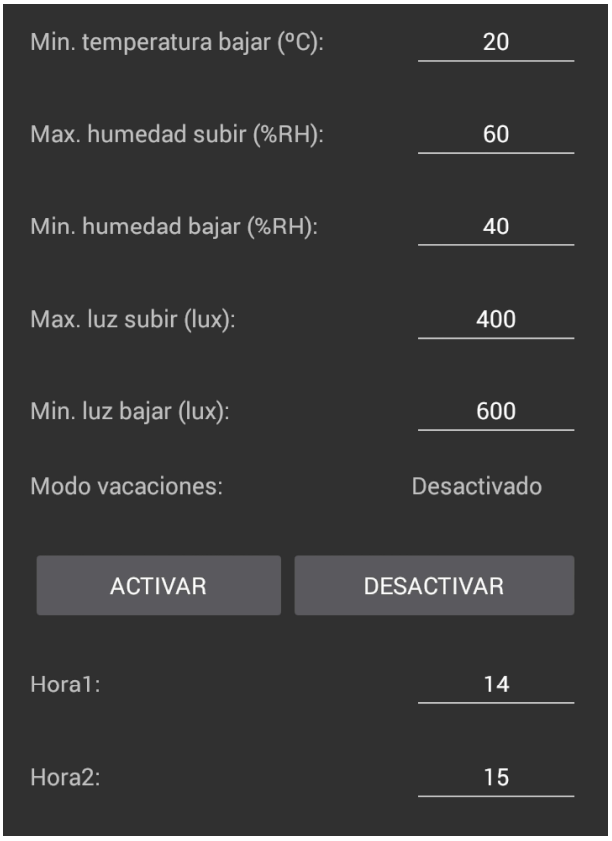
Como se ha comentado anteriormente, esto es una explicación breve del código. Este se muestra al completo en el Apéndice K.

Manual de usuario

El interfaz de la aplicación se compone de varios TextViews, editText, botones y un ScrollView para subir y bajar en la pantalla. Esta interfaz se puede ver en la Figura 3.35 y su código en el Apéndice L.



(a) Parte superior de la aplicación

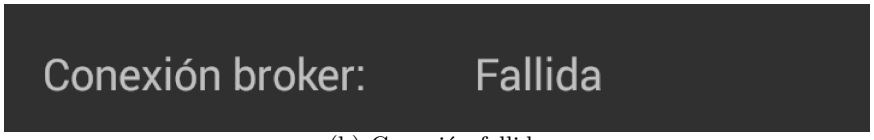


(b) Parte inferior de la aplicación

Figura 3.35: Interfaz de la aplicación

Lo primero que se muestra es si se ha conectado al broker o no. En la Figura 3.36, se muestran los mensajes que aparecen cuando se conecta y cuando no puede.

(a) Conexión establecida

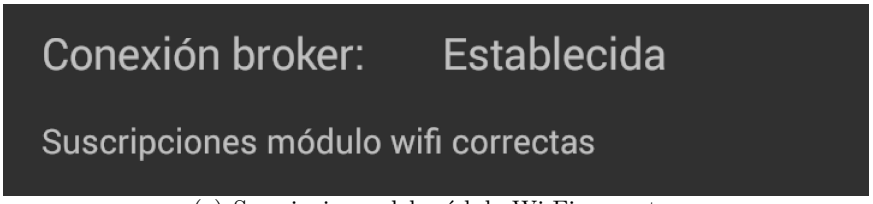


(b) Conexión fallida

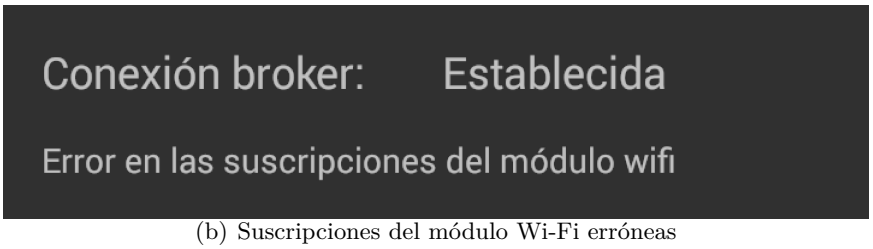
Figura 3.36: Conexión broker

Una vez establecida la conexión, lo siguiente que se muestra es un mensaje avisando si el módulo

Wi-Fi se ha suscrito correctamente a los tópicos o no. Esto se puede ver en la Figura 3.37.



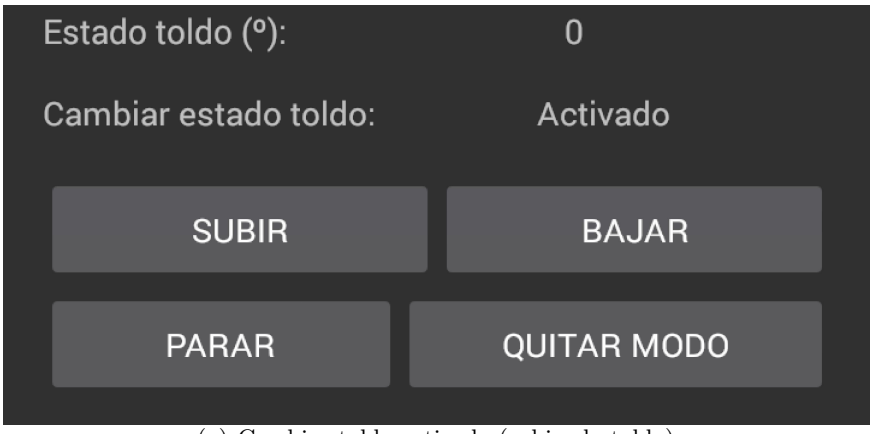
(a) Suscripciones del módulo Wi-Fi correctas



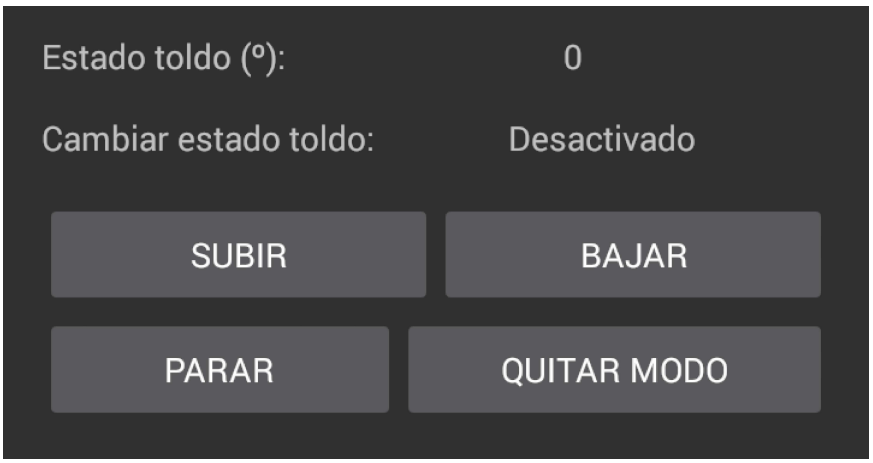
(b) Suscripciones del módulo Wi-Fi erróneas

Figura 3.37: Suscripciones del módulo Wi-Fi

Después, se muestra el estado del toldo: si está en 0 es que el toldo está subido completamente y si está en 90, el toldo está bajado completamente. Debajo, hay cuatro botones para subir y bajar el toldo, parar la acción de subir o bajar y salir del modo de cambiar la posición del toldo desde la aplicación y funcione según los interruptores. Si se ha dado al botón subir, bajar o parar en Çambiar estado toldo:"pone activado mientras que si está dado quitar modo aparece desactivado. En la Figura 3.38, se muestra el estado del toldo con un valor y con el modo activado y desactivado.



(a) Cambiar toldo activado (subiendo toldo)



(b) Cambiar toldo desactivado

Figura 3.38: Estado del toldo

A continuación, se muestran las mediciones que publican los sensores de la parte del toldo. Esto se puede ver en la Figura 3.39.

Figura 3.39: Datos de los sensores

Lo siguiente son los datos que se pueden modificar para enviar al módulo Wi-Fi, como es el caso de los umbrales de temperatura, humedad o luz y de la clase. Si se introduce un valor incorrecto, aparece un mensaje de error y no publica el valor. Por ejemplo, el umbral de temperatura de subir debe ser más pequeño que el de bajar, lo mismo sucede con los umbrales de luz, mientras que con los umbrales de humedad el de subir es mayor que el de bajar y respecto a la clase sólo puede ser 1,2 o 3. En la Figura 3.40, se muestra un ejemplo de un mensaje de error.

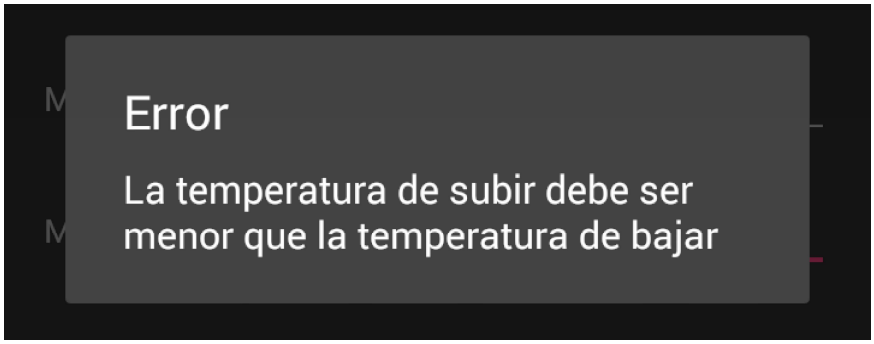


Figura 3.40: Mensaje de error

Por último, está la configuración del modo vacaciones y sus opciones. Hay dos botones uno para activar el modo y otra para desactivar. Encima de los mensajes aparece un mensaje con la opción marcada (activado o desactivado). Mientras que debajo de los botones, hay dos editText para modificar las horas de subida o bajada de toldo en modo automático. Si se intenta publicar una hora superior a 24, aparece un mensaje de error y no se publica. En la Figura 3.41, se muestra una imagen con el modo vacaciones activado.

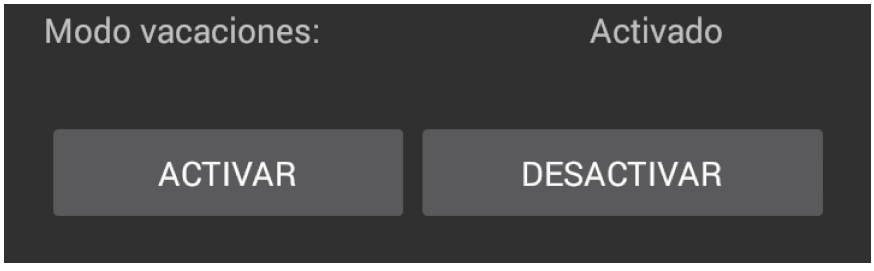


Figura 3.41: Modo vacaciones activado

Capítulo 4

Presupuesto

En este capítulo, se pretende realizar un presupuesto de lo que cuesta diseñar nuestro dispositivo. Además, se quiere conseguir un precio razonable para la venta al cliente. Para que su precio no sea demasiado elevado, hay que vender la mayor cantidad posible. De esta manera, el precio de desarrollo del ingeniero se reparte entre todos los dispositivos.

En la Tabla 4.1, se muestra el precio que vale el material para un dispositivo. A la hora de venderlo esta cantidad es fija en todos los dispositivos. Este presupuesto está hecho para sólo un dispositivo, pero al comprar el material, hay una cantidad mínima requerida. Sin embargo, esto no es de gran importancia ya que si se vende el producto va a ser cuando se tengan que fabricar varios dispositivos porque sino nadie lo va a comprar por el precio elevado. Debido a esto, se queda el precio unidad del material para cada dispositivo.
En este presupuesto no se ha tenido en cuenta el toldo ni el motor ya que esto depende del gusto del cliente y de las dimensiones que tenga el toldo. El precio del toldo se puede consultar en Leroy Merlin, [1] puesto que tiene varios modelos y con una gran variedad de precios. El precio de los toldos manuales oscila desde los 69 €que cuesta el más barato hasta los 749,95 €que cuesta el más caro.

Por otro lado, al coste del material hay que sumarle el coste de desarrollo del ingeniero que consiste en el número de horas que tarda el ingeniero en diseñar el producto por el precio por hora. Este coste se muestra en la Tabla 4.2. Este se va reduciendo según el número de unidades que se vendan.

También, se debe añadir el coste de publicar la aplicación en Google Play. Este pago, se realiza sólo una vez para dar de alta en Google Play al desarrollador de la aplicación y tiene un coste de 23,45 €. Después, de las ganancias de la aplicación, Google se queda un 30 % de la facturación total.

Por último, hay que tener en cuenta el precio de tener un servidor. Buscando en Internet se ha encontrado el **servicio de alojamiento dedicado KS-1** de la empresa Kimsufi [2] y que tiene un coste mensual de 6,04 €. Este servicio proporciona 500 gb de capacidad y velocidades de 100 Mbps. En principio, con estas características es suficiente para alojar Mosquitto en él. El coste de este servidor se podría compensar con las ganancias de la aplicación. Si no fuese así, habría que cobrar una cantidad simbólica a los usuarios para mantener el servidor, por ejemplo, una tasa anual de 10 €.

En la Tabla 4.3, se muestra el precio total del producto según el número de unidades que se diseñen. Como se puede observar, a partir de 500 dispositivos fabricados, el precio se asemeja al precio del material, el cuál no se puede rebajar porque sino serían pérdidas para el vendedor.

[1] Toldos Leroy Merlin: http://www.leroymerlin.es/productos/jardin/toldos/toldos.html
[2] Los servidores de esta empresa se pueden consultar aquí: https://www.kimsufi.com/es/servidores.xml

Componente	**Coste unidad (€)**	**Unidades**	**Coste total (€)**
Resistencia 10k	0,0119	5	0,0595
Resistencia 1M	0,0120	1	0,0120
Resistencia 470R	0,0120	4	0,0480
Resistencia 1k	0,0119	4	0,0476
Resistencia 10R	0,0123	1	0,0123
Resistencia 4k7	0,0120	2	0,0240
Resistencia 18k	0,0195	1	0,0195
Resistencia 2k2	0,0118	1	0,0118
Resistencia 3k3	0,0119	3	0,0357
Resistencia 1k8	0,0196	3	0,0588
Condensador 100 nF	0,0134	9	0,1206
Condensador 10 nF	0,2020	2	0,4040
Condensador 10 uF	0,1770	1	0,1770
Condensador 22 uF	0,3210	2	0,6420
FT232RL	4,2500	1	4,2500
Protección electoestática USBLC6	0,1090	1	0,1090
Regulador 3,3 V	0,3090	1	0,3090
Optotriac MOC3063XSM	1,0400	2	2,0800
Sensor SHT31	5,3400	1	5,3400
Sensor VEML6030	1,6700	1	1,6700
Triac BTA08	1,3700	2	2,7400
Fuente de alimentación	2,3500	1	2,3500
Led rojo	0,2520	1	0,2520
Led verde	0,2180	1	0,2180
Conector MOLEX 22-23-2061	0,0830	2	0,1660
Conector mini USB	0,5010	1	0,5010
Block Terminal 2 salidas	0,2290	2	0,4580
Block Terminal 3 salidas	0,2640	2	0,5280
Conector 40-0518-10	3,4000	1	3,4000
Módulo Wi-Fi esp8266-12F	2,3200	1	2,3200
Ferrita MMZ2012R301A	0,0900	1	0,0900
Anemómetro	85,0000	1	85,0000
PCB	1,8720	1	1,8720
Caja	0,9500	1	0,9500
Tapadera caja	3,9500	1	3,9500
Interruptores	6,9500	1	6,9500
Embellecedor	3,9500	1	3,9500
Coste total material (€)	131,1258		

Tabla 4.1: Coste según el número de dispositivos fabricados

Horas	**Precio por hora (\euro)**	**Total (€)**
380	15	5700

Tabla 4.2: Presupuesto ingeniero

Unidades	**Coste material (€)**	**Coste aplicación (€)**	**Coste ingeniero (€)**	**Coste total (€)**
1	131,1258	23,45	5700	5854,5758
10	131,1258	2,345	570	703,4708
100	131,1258	0,02345	57	188,14925
500	131,1258	0,0000469	11,4	142,5258469
1000	131,1258	4,69E-08	5,7	136,8258
2000	131,1258	2,345E-11	2,85	133,9758
5000	131,1258	4,69E-15	1,14	132,2658
10000	131,1258	4,69E-19	0,57	131,6958

Tabla 4.3: Coste según el número de dispositivos fabricados

Capítulo 5

Conclusiones

5.1. Aspectos a mejorar

Una de las posibles mejoras del proyecto es introducir la opción de elegir el broker, al que el usuario quiere conectarse, desde la aplicación. La complejidad de esta mejora, se encuentra en cambiar el broker del módulo Wi-Fi ya que si cambia el de la aplicación también tiene que cambiar en el toldo. La solución sería enviar el nombre del nuevo broker desde la app al módulo Wi-Fi y desconectar la aplicación y el módulo Wi-Fi del antiguo broker y conectarlos al nuevo.
Con esta mejora se quiere evitar que cuando se desee cambiar de broker por algún motivo, no haya que buscar en el código de ambas partes y tener que cambiar cosas.

Otra mejora es que haya un usuario y contraseña en el broker por cada toldo fabricado. Actualmente sólo hay un usuario. Con esto se pretende que los mensajes sean privados y no se mezclen con los de otros clientes, ya que si un cliente envía la orden de bajar toldo, bajarán todos los toldos que utilizan el mismo usuario en el broker. Si hay un usuario por cada cliente, sólo bajará su toldo.

Como se ha comentado en el apartado de los sensores, una mejora de diseño es colocar el sensor de luz en una cara de la PCB y el sensor de temperatura-humedad en la otra. De esta manera, se evita que el sensor de temperatura obtenga una mayor temperatura de la real debido a calentamiento por el impacto del sol.

5.2. Conclusiones finales

Ya hemos visto, que el Internet de las Cosas está en continuo crecimiento. De hecho, actualmente, existen más dispositivos conectados a la red que personas en el mundo, y este número va a seguir aumentando. Por lo que tener la posibilidad de desarrollar un dispositivo IoT, es muy interesante, puesto que desarrollar estos dispositivos es el futuro de los ingenieros.

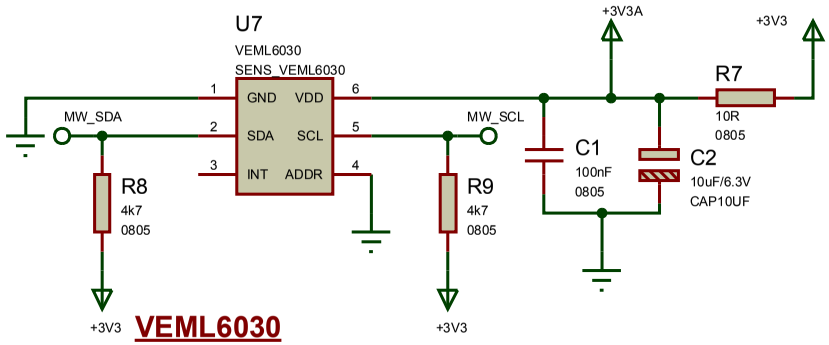
Respecto a las tecnologías inalámbricas existen muchas para todo tipo de aplicaciones. Sin embargo, en un futuro se acabarán imponiendo unas a otras. De esta manera todo estará estandarizado.
Análogamente, sucede lo mismo con los protocolos de comunicación.

Desde mi punto de vista, este proyecto es de gran utilidad, ya que permite afianzar conocimientos de electrónica mediante el diseño del esquemático y de la PCB. Además, posibilita el estudio de protocolos como MQTT y tecnologías inalámbricas como Wi-Fi, ya que estos temas apenas son estudiados en nuestra mención. Por último, también permite la introducción al mundo del desarrollo de aplicaciones Android, que en la actualidad es un factor muy importante, puesto que todo está evolucionando

al Internet de las Cosas.

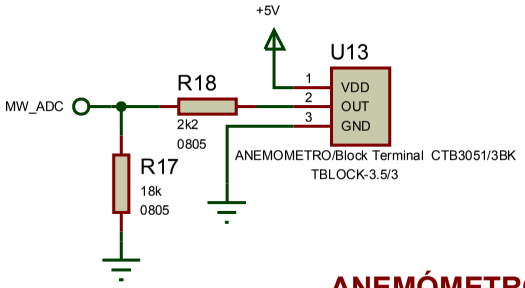
Apéndice A

Esquemático



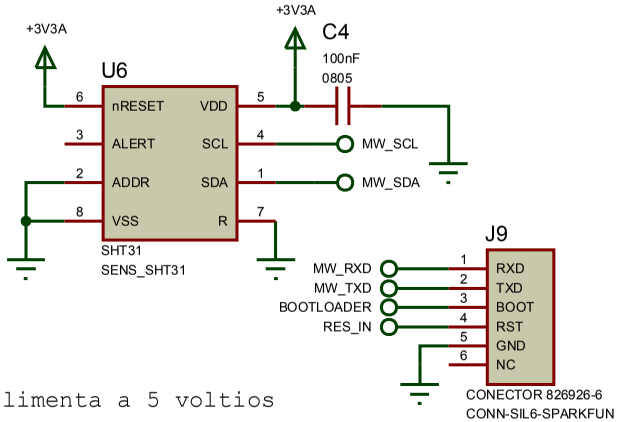
VEML6030

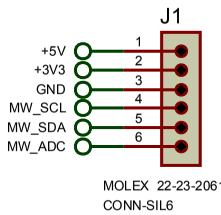
Sensor digital de luz.
Con ADDR conectada a tierra su dirección de esclavo de 7 bits es 0010000.
Si se quiere escribir la dirección es 00100000=0x20, mientras que si se quiere leer la dirección es 00100001 =0x21.



SHT31

Sensor digital de humedad y temperatura.
Su dirección de esclavo es 1000100.
Para lectura su dirección es 10001000=0x45
y para escritura su dirección es 10001001=0x44.

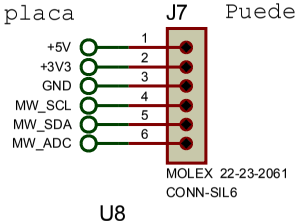


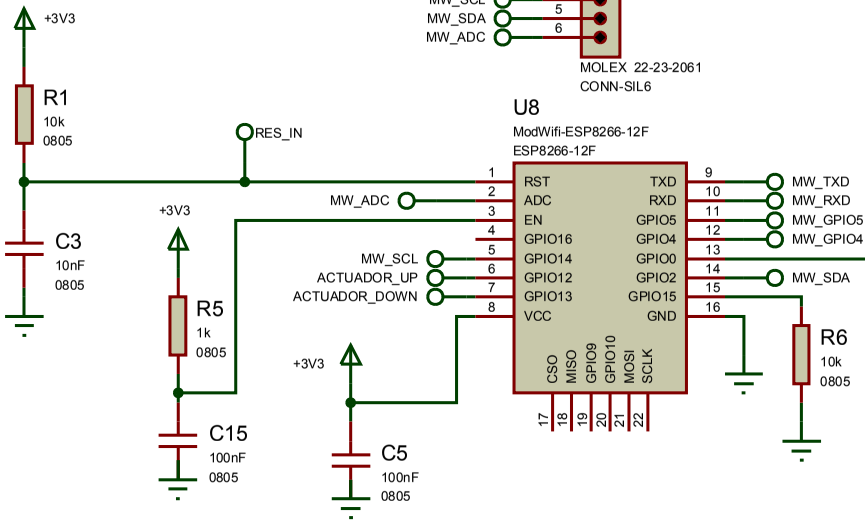


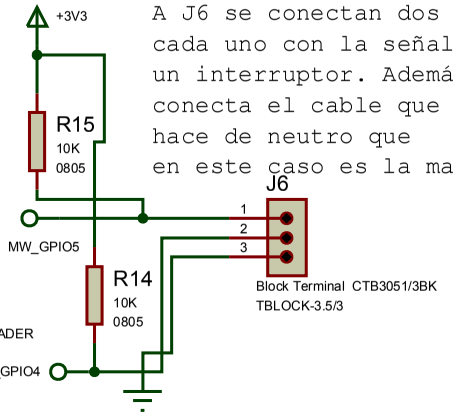
J1 y J7 se utilizan para conectar el bus I2C, las tensiones de 3,3V y 5V, la masa y la señal del anemómetro entre las dos placas.
En el esquema he puesto SEN en la placa del sensor y MW en la del módulo.

ANEMÓMETRO

Anemómetro que se alimenta a 5 voltios y proporciona una salida analógica entre 0,4 voltios y 2 voltios.
Puede medir vientos de hasta 180 km/h.



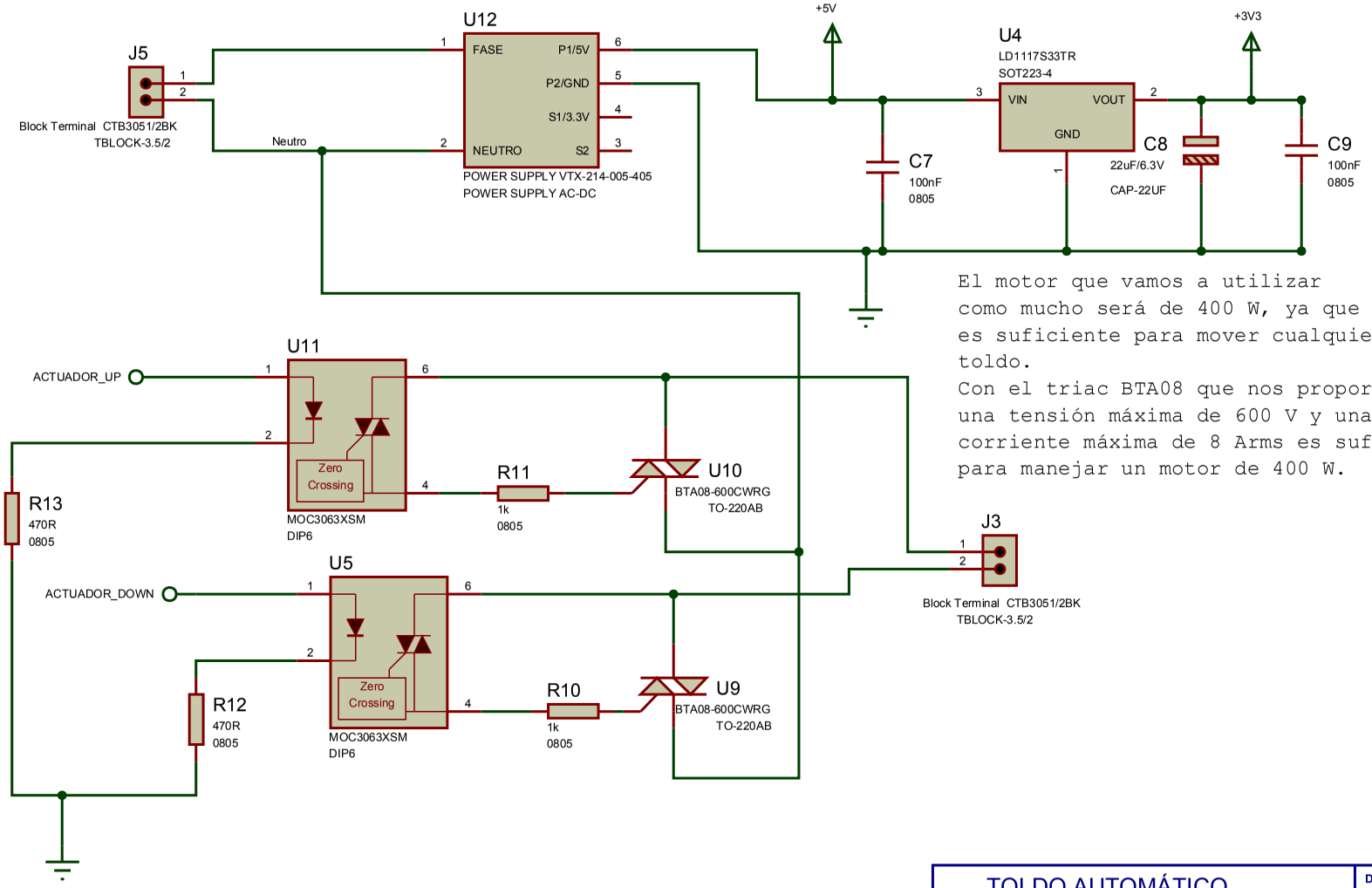




A J6 se conectan dos cables cada uno con la señal de un interruptor. Además se conecta el cable que hace de neutro que en este caso es la masa.

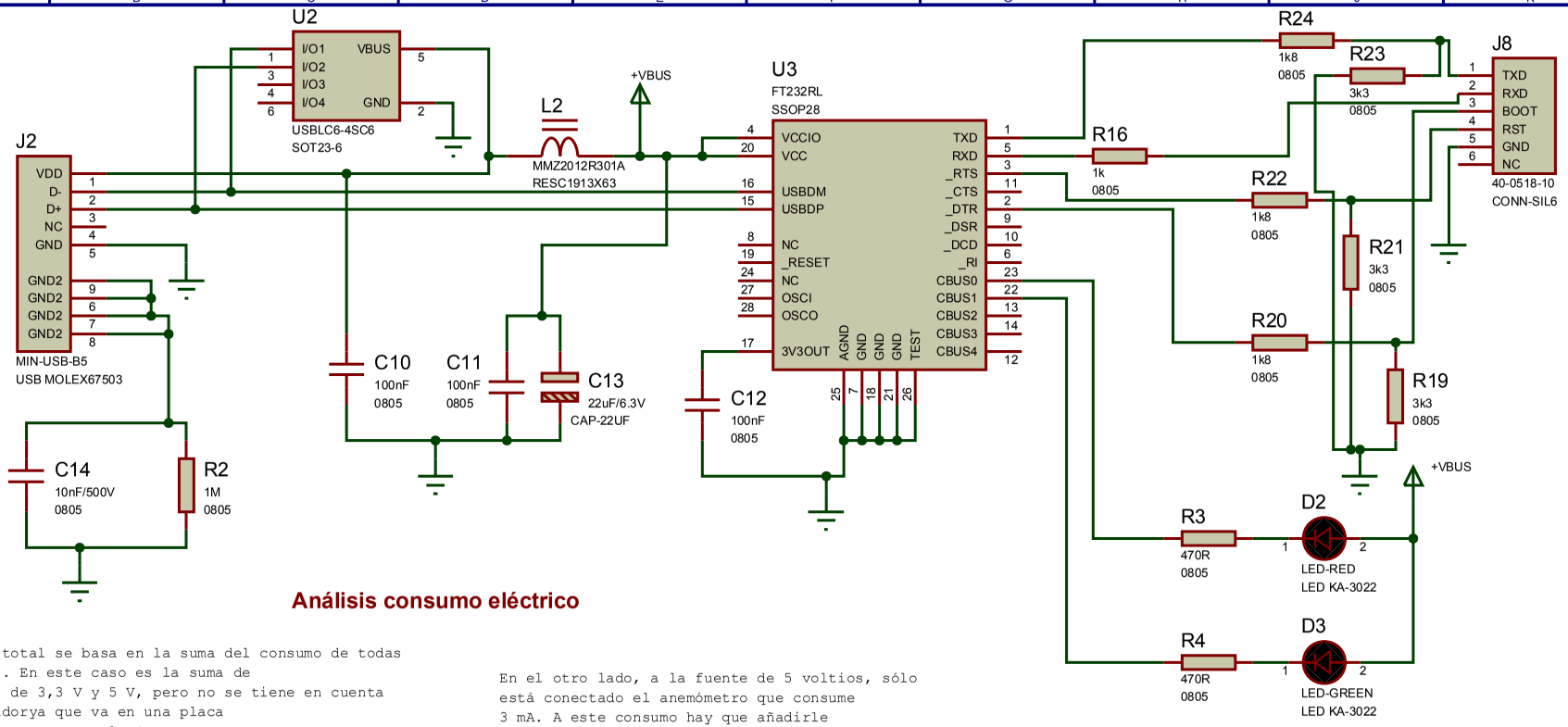
TOLDO AUTOMÁTICO	DATE: 11/02/2017
FILE NAME: EsquematicoModWifi.DSN	
DESIGN TITLE: Esquemático toldo	PAGE: 1 of 3
BY: David García	

Para la fuente de alimentación tenemos dos opciones:
- VIGORTRONIX VTX-214-005-405
-AC-DC Converter Voltage de ITEAD
Para el listado de materiales ponemos la primera ya que es la más cara y es de Farnell.
El componente tiene una doble huella para que en el momento del montaje seleccionemos la que mejor se adapte.



El motor que vamos a utilizar como mucho será de 400 W, ya que es suficiente para mover cualquier toldo.
Con el triac BTA08 que nos proporciona una tensión máxima de 600 V y una corriente máxima de 8 Arms es suficiente para manejar un motor de 400 W.

TOLDO AUTOMÁTICO	DATE: 11/02/2017
FILE NAME: EsquematicoModWifi.DSN	
DESIGN TITLE: Esquemático toldo	PAGE: 2 of 3
BY: David García	



Análisis consumo eléctrico

El consumo total se basa en la suma del consumo de todas las fuentes. En este caso es la suma de las fuentes de 3,3 V y 5 V, pero no se tiene en cuenta el programadorya que va en una placa aparte que se conecta al PC.

Fuente 3,3V

A esta fuente está conectado el led rojo, el led verde, los sensores de luz y humedad-temperatura y el módulo wifi.
Tanto el led rojo como el led verde consumen 3 mA cada uno.
Mientras que que el sensor de luz consume 0,045 mA y el sensor de humedad-temperatura consume 0,8 mA.
Por último, el módulo wifi consume 200 mA, de los cuales 120 mA consume el propio módulo y el resto lo consume la memoria SPI que lleva incorporada el chip.
El consumo total de esta fuente es 206,845 mA.

Fuente 5V

El programador es independiente al consumo total, pero lo analizamos para comprobar que el PC proporciona la suficiente corriente para su correcto funcionamiento.
Del programador, sólo está conectado el FT232RL que consume 15 mA. Como el PC proporciona 500 mA, es más que suficiente para que funcione.

En el otro lado, a la fuente de 5 voltios, sólo está conectado el anemómetro que consume 3 mA. A este consumo hay que añadirle el consumo de la fuente de 3,3V.
Por lo que el consumo de esta fuente es 209,45 mA.

Análisis térmico

Para la realización del análisis térmico, una vez conocido el consumo de cada dispositivo, solo hace falta buscar información acerca de la resistencia térmica del elemento a analizar.

El regulador LD1117S33TR sube 8,8 °C por encima de la temperatura ambiente.

El FT232RL sube 4,2 °C por encima de la temperatura ambiente.

El triac sube 1,6 °C por encima de la temperatura ambiente

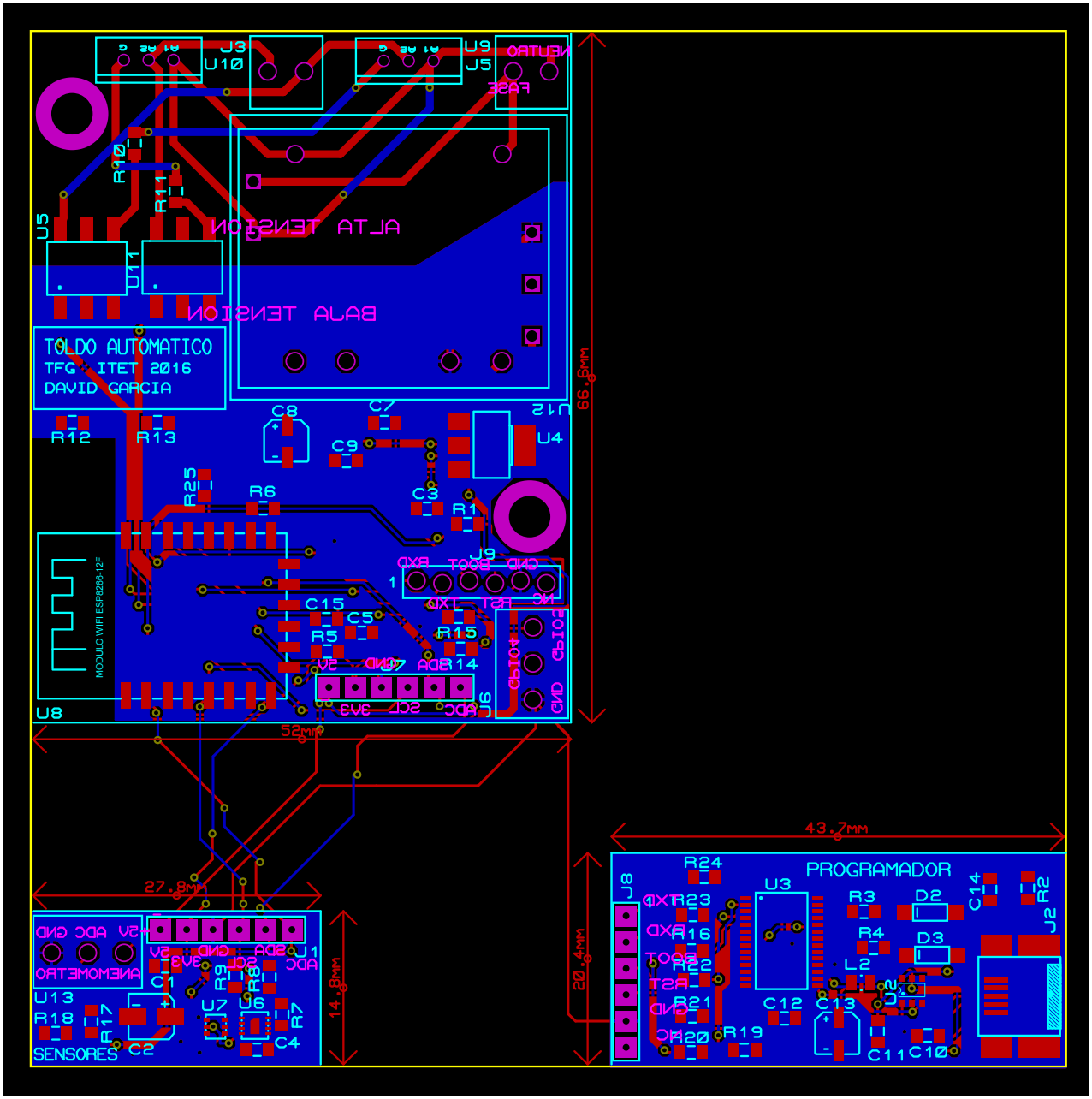
Para realizar el análisis térmico del módulo wifi, el transformador, el anemómetro y los sensores de luz y humedad-temperatura no conocemos el valor de la resistencia térmica, por lo que no podemos calcular cuanto se calientan.

J8 es un connsil-6 hembra que se emplea para conectar la placa del programador, cuyo esquemático es el de esta hoja, con la placa en la que está el módulo wifi, el cual contiene el micro que vamos a programar.
Para conectar ambas placas se usa J8 y J9 que es un connsil macho.

TOLDO AUTOMÁTICO	DATE: 11/02/2017
FILE NAME: EsquematicoModWifi.DSN	PAGE: 3 of 3
DESIGN TITLE: Esquemático toldo	
BY: David García	

Apéndice B

Layout



Apéndice C

Listado de materiales

Bill Of Materials For Esquemático Toldo

Design Title : Esquemático toldo
Author : David García
Revision :
Design Created : miércoles, 5 de octubre de 2016
Design Last Modified : sábado, 11 de febrero de 2017
Total Parts In Design : 62

25 Resistors

Quantity:	References	Value	Price	Rf	Package
5	R1, R6, R14, R15, R25	10k	0.0119	9237755	0805
1	R2	1M	0.012	9237992	0805
4	R3, R4, R12, R13	470R	0.012	9237445	0805
4	R5, R10, R11, R16	1k	0.0119	9237496	0805
1	R7	10R	0.0123	9237240	0805
2	R8, R9	4k7	0.012	9237704	0805
1	R17	18k	0.0195	9237780	0805
1	R18	2k2	0.0118	9237534	0805
3	R19, R21, R23	3k3	0.0119	9237682	0805
3	R20, R22, R24	1k8	0.0196	9237526	0805

14 Capacitors

Quantity:	References	Value	Price	Rf	Package
9	C1, C4, C5, C7, C9-C12, C15	100nF	0.0134	1759265	0805
1	C2	10uF/6.3V	0.177	2466348	CAP10UF
1	C3	10nF	0.202	1284130	0805
2	C8, C13	22uF/6.3V	0.321	2346354	CAP-22UF
1	C14	10nF/500V	0.202	1284130	0805

12 Integrated Circuits

Quantity:	References	Value	Price	Rf	Package
1	U2	USBLC6-4SC6	0.109	2473978	SOT23-6
1	U3	FT232RL	4.25	1146032	SSOP28
1	U4	LD1117S33TR	0.309	1202826	SOT223-4
2	U5, U11	MOC3063XSM	1,04	1683337	DIP6
1	U6	SHT31	5.34	2474218	SENS_SHT31
1	U7	VEML6030	1.67	2627811	SENS_VEML6030
1	U8	ModWifi-ESP8266-12F	2,32	ITEAD	ESP8266-12F
2	U9, U10	BTA08-600CWRG	1.37	1057270	TO-220AB
1	U12	POWER SUPPLY VTX-214	5.98	2627168	POWER SUPPLY AC-DC
1	U13	ANEMOMETRO/Block Ter	0.264	3882627	TBLOCK-3.5/3

2 Diodes

Quantity:	References	Value	Price	Rf	Package
1	D2	LED-RED	0.252	1142617	LED KA-3022
1	D3	LED-GREEN	0.218	1142615	LED KA-3022

9 Miscellaneous

Quantity:	References	Value	Price	Rf	Package
2	J1, J7	MOLEX 22-23-2061	0.083	1462922	CONN-SIL6
1	J2	MIN-USB-B5	0.501	2313554	USB MOLEX67503
2	J3, J5	Block Terminal CTB3	0.229	3882615	TBLOCK-3.5/2
1	J6	Block Terminal CTB3	0.264	3882627	TBLOCK-3.5/3
1	J8	40-0518-10	3.4	1674779	CONN-SIL6
1	J9	CONECTOR 826926-6	0.249	1248144	CONN-SIL6-SPARKFUN
1	L2	MMZ2012R301A	0.09	1669724	RESC1913X63

Apéndice D

Software toldo (Arduino)

ToldoAutomatico.ino

```
#include <ESP8266WiFi.h>

//Librerias de Adafruit para el protocolo MQTT
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"

#include <Wire.h>   //Libreria para las funciones de I2C
#include <WiFiUdp.h>

//Para poder usar el pin ADC
extern "C" {
#include "user_interface.h"
}

//Definimos macros para la conexion wifi
#define ssid  "Wifi_Lab_Proyectos"
#define pass  "R3B00TY0URM1ND"
//Definimos datos del broker
#define servidor "david2016.dtdns.net"
#define puertoServidor 8883
//Definimos macros para el usuario del broker
#define username "david"
#define key "2016david"

//Definimos las direcciones de los sensores
#define addr_sensortemphum 0x44
#define addr_sensorluz     0x23
//Asignamos un nombre a los pines que vamos a utilizar del ESP8266
#define actuador_up 12
#define actuador_down 13
#define gpio4 4
#define gpio5 5
#define SDApin 2
#define SCLpin 14

#define LEAP_YEAR(Y)  (((1970+Y)>0) && !((1970+Y)%4) &&
          (((1970+Y)%100)  ||  !((1970+Y)%400))) //Anyo bisiesto

#define tiempoMaximo 30000  //30 segundos
//Variables globales
int estadoToldo=0,estadoToldoantiguo=0; //Indica la posicion del toldo
int tiempoMoviendo=0; //Indica el tiempo que el toldo se esta moviendo
int arranque=0;  //Variable para obtener varios datos solo al arranque del programa
```

```
44 long int factorCorreccion=0,epoch2,millisCorregidos;//Variables para obtener la hora
45 int pulsador1=0, pulsador2=0,
46   pulsador1antiguo=0, pulsador2antiguo=0;//Posicion de los interruptores
47 unsigned long t1inicial=0, t1final=0,
48       t2inicial=0, t2final=0, t1automatico=0,
49       t2automatico=0; //Para calcular el tiempo que esta el toldo en una posicion
50 int muestra1=0,muestra2=0,muestra3=0; //Numero de muestras en la medicion de los
       sensores
51 int m1=0,m2=0;//Para indicar el sentido de giro del motor
52 int aux1=0,aux2=0,aux3=0,aux4=2;//Auxiliares para tener en cuenta el estado del toldo
53 float humMuestra[100], tempMuestra[100],
54   vienMuestra[100],luzMuestra[100]; //Para guardar la medicion de los sensores
55 float temperaturapromedio=0, humedadpromedio=0,
56     vientopromedio=0, luzpromedio=0; //Para guardar el promedio calculado
57 float temperatura=0, humedad=0,
58     viento=0,luz=0;//Variables para ir almacenando la suma de todas las muestras
59 int vientoantiguo[100],tempantiguo[100],
60   humantiguo[100],luzantiguo[100];//Guarda la medida para dicha muestra
61 float vientokm=0;//Para pasar el viento a unidades de km
62 int inicio=0,inicio5=0;//Variables auxiliares para hacer medidas iniciales
63 int clase=1; //Iniciamos la clase del toldo a 1
64 int vientomaximo; //Maximo viento que aguanta el toldo
65 int x=0, y=0,z=0;
66 int temperaturasubir1=12,temperaturabajar1=20, temperaturaumbrales;//Umbrales de la
       temperatura
67 int humedadsubir1=60 ,humedadbajar1=40, humedadumbrales;//Umbrales de la humedad
68 int luzsubir1=400, luzbajar1=600;//Umbrales de la luz
69 int modovacaciones=0;//Indica si esta activado o no el modo vacaciones
70 int datosAleatorios[6];//Guarda la hora y fecha en el modo vacaciones
71 int hora1=14, hora2=15,horas;//Horas de activacion para el modo vacaciones
72 int numeroAleatorio=0;//Para aplicar diferente mascara en el modo vacaciones
73 long int aleatorio;//Guarda los ms obtenidos al aplicar la mascara
74 int error=0;//Para indicar si hay error en las suscripciones del esp8266
75 int inicio1=0, inicio2=0, inicio3=0;//Variables auxiliares para el modo vacaciones
76 unsigned long t0; //Tiempo inicial en el que empieza el bucle principal
77 int datosFechaHora[6];//Para obtener la fecha y hora
78 int z1=0, z2=0,z3=0;//Variables auxiliares para la toma de muestras de los sensores
79 int modoautomatico=0,modoautomaticoantiguo=2;//Para indicar el modo
80 int tfinal;//Tiempo al final del bucle
81 int topInicial1;//Variable para enviar el topico inicial1
82 int arranque1=0,arranque2=0;//Variables auxiliares para tomar valores inicales
83 int cambiartoldo=2,subaj=2;//Variables para que se pueda subir/bajar desde la app
84 int noche=0;//Indica si es de noche
85 int subir=0,bajar=0,subir1=0,subir2=0,subir3=0,
86     bajar1=0,bajar2=0,bajar3=0;//Para determinar subir o bajar en modo automatico
87 int nvueltas=0;//Variable para resincronizar la hora una vez al dia
88
89 //Configuracion MQTT
90 //Creamos un cliente wifi que es el esp8266 para conectar con mosquitto
91 WiFiClientSecure client;
92 //Configuramos el cliente mqtt pasando el cliente wifi,
93 //el servidor MQTT y los datos de acceso del usuario
94 Adafruit_MQTT_Client mqtt(&client, servidor, puertoServidor,username,key);
95
96
97 //Topicos sobre los que va a publicar
98 Adafruit_MQTT_Publish pHoras                =Adafruit_MQTT_Publish(&mqtt, username "/
       toldo/Horas",1,1);
99 Adafruit_MQTT_Publish pTemperaturaUmbrales  =Adafruit_MQTT_Publish(&mqtt, username "/
       toldo/TemperaturaUmbrales",1,1);
```

```
100 Adafruit_MQTT_Publish pHumedadUmbrales        =Adafruit_MQTT_Publish(&mqtt, username "/
    toldo/HumedadUmbrales",1,1);
101 Adafruit_MQTT_Publish pTemperatura            =Adafruit_MQTT_Publish(&mqtt, username "/
    toldo/temperatura",1,1);
102 Adafruit_MQTT_Publish pHumedad                =Adafruit_MQTT_Publish(&mqtt, username "/
    toldo/humedad",1,1);
103 Adafruit_MQTT_Publish pLuz                    =Adafruit_MQTT_Publish(&mqtt, username "/
    toldo/luz",1,1);
104 Adafruit_MQTT_Publish pViento                 =Adafruit_MQTT_Publish(&mqtt, username "/
    toldo/viento",1,1);
105 Adafruit_MQTT_Publish pLuzSubir1              =Adafruit_MQTT_Publish(&mqtt, username "/
    toldo/LuzSubir1",1,1);
106 Adafruit_MQTT_Publish pLuzBajar1              =Adafruit_MQTT_Publish(&mqtt, username "/
    toldo/LuzBajar1",1,1);
107 Adafruit_MQTT_Publish pestadoToldo            =Adafruit_MQTT_Publish(&mqtt, username "/
    toldo/estadoToldo",1,1);
108 Adafruit_MQTT_Publish pTopInicial1            =Adafruit_MQTT_Publish(&mqtt, username "/
    toldo/Inicial1",1,1);
109 Adafruit_MQTT_Publish pClase                  =Adafruit_MQTT_Publish(&mqtt, username "/
    toldo/Clase",1,1);
110 Adafruit_MQTT_Publish pError                  =Adafruit_MQTT_Publish(&mqtt, username "/
    toldo/errorSuscripcion",1,1);
111
112 //Topicos a los que se va a suscribir
113 Adafruit_MQTT_Subscribe sHoras                  =Adafruit_MQTT_Subscribe(&mqtt,
    username "/toldo/Horas");
114 Adafruit_MQTT_Subscribe sTemperaturaUmbrales    =Adafruit_MQTT_Subscribe(&mqtt,
    username "/toldo/TemperaturaUmbrales");
115 Adafruit_MQTT_Subscribe sHumedadUmbrales       =Adafruit_MQTT_Subscribe(&mqtt,
    username "/toldo/HumedadUmbrales");
116 Adafruit_MQTT_Subscribe sLuzSubir1            =Adafruit_MQTT_Subscribe(&mqtt,
    username "/toldo/LuzSubir1");
117 Adafruit_MQTT_Subscribe sLuzBajar1            =Adafruit_MQTT_Subscribe(&mqtt,
    username "/toldo/LuzBajar1");
118 Adafruit_MQTT_Subscribe sCambiarToldo         =Adafruit_MQTT_Subscribe(&mqtt,
    username "/toldo/Cambiar");
119 Adafruit_MQTT_Subscribe sClase                =Adafruit_MQTT_Subscribe(&mqtt,
    username "/toldo/Clase");
120 Adafruit_MQTT_Subscribe smodoVacaciones       =Adafruit_MQTT_Subscribe(&mqtt,
    username "/toldo/modoVacaciones");
121
122 Adafruit_MQTT_Subscribe *subscription1;
123
124
125 //Configuracion NTP
126 unsigned int localPort=2390; //Puerto local para escuchar los paquetes UDP
127
128 IPAddress timeServerIP; //Variable para obtener la IP del servidor
129 const char* ntpServerName= "clepsidra.tel.uva.es";
130
131 const int NTP_PACKET_SIZE=48; //En el mensaje NTP el tiempo esta en los primeros 48
    bytes
132
133 byte packetBuffer [NTP_PACKET_SIZE]; //Buffer para guardar los paquetes entrantes y
    salientes
134 static  const int diasPorMes[]={31,28,31,30,31,30,31,31,30,31,30,31}; // Dias que tiene
     cada mes
135
136 //Creamos una variable de tipo UDP para enviar y recibir paquetes a traves de UDP
137 WiFiUDP udp;
```

```
138 void MQTT_connect();
139
140 void setup() {
141   Serial.begin(115200);
142   Serial.println(F("Toldo␣automatico"));
143   //Definimos entradas y salidas
144   pinMode(actuador_up, OUTPUT);
145   pinMode(actuador_down, OUTPUT);
146   pinMode(gpio4, INPUT);
147   pinMode(gpio5, INPUT);
148
149   //Conectamos el modulo esp8266 al wifi
150   Serial.println();
151   Serial.print("Conectando␣a␣");
152   Serial.println(ssid);
153
154   WiFi.begin(ssid,pass);
155
156   //Comprobamos que se ha conectado
157   while(WiFi.status()!=WL_CONNECTED){
158     delay(500);
159     Serial.print(".");
160   }
161   //Mostramos que se ha conectado y obtenemos la IP local
162   Serial.println();
163   Serial.println("WiFi␣conectado");
164   Serial.println("Direccion␣IP:␣");
165   Serial.println(WiFi.localIP());
166
167   //Conexion UDP para obtener fecha y hora del servidor NTP
168   Serial.println("Iniciando␣UDP");
169   udp.begin(localPort);
170   Serial.println("Puerto␣local:␣");
171   Serial.println(udp.localPort());
172
173   suscripcionesIniciales();
174   randomSeed(millis());
175   //Inicializamos I2C
176   Wire.begin(SDApin, SCLpin);
177   Wire.setClock(100000);
178
179 }
180
181 //-------------Funciones I2C--------------------//
182 int I2C_EnviaTrama(char addr, char *datos, char n_datos){
183   int i;
184   Wire.beginTransmission(addr); //Envia Start y la direccion de escritura del esclavo
185   for(i=0;i<n_datos;i++){
186     Wire.write(datos[i]);
187   }
188
189   return Wire.endTransmission(); //0: OK
190                                  //2: NACK on ADDR
191                                  //3: NACK on data
192                                  //4: Bus busy
193 }
194 void I2C_LeeTrama(char addr, char *datos, char n_datos){
195   int i=0;
196   Wire.beginTransmission(addr);
197   Wire.endTransmission();
```

```
  Wire.requestFrom(addr,n_datos); //Envia Start y la direccion de lectura del esclavo
  while(Wire.available()>0){
    datos[i]=Wire.read();
    i++;
  }

}
char crc8(char *datos, int len){
  char polinomio=0x31;
  char crc=0xFF;
  int i,j;

  for (j=0; j<len;j++)
  {
    crc ^= *datos++;
    for (i=0; i<8; i++ )
    {
      crc = ( crc & 0x80 )
            ? (crc << 1) ^ polinomio
            : (crc << 1);
     }
  }

  return crc;
}
void comprobarACK(int ack){
  if(ack==0){
   Serial.println("Transmision correcta");
  }else if(ack==2){
   Serial.println("Direccion erronea");
  }else if(ack==3){
   Serial.println("Error, NACK en los datos");
  }else if(ack==4){
   Serial.println("Bus lleno");
  }
}

//------------Funciones MQTT-----------//
void MQTT_connect(){
  int ret, reintentos=5;

  //Comprobamos si ya esta conectado
  if(mqtt.connected()){
    return;  //Si esta conectado salimos de la funcion
  }

  Serial.print("Conectando MQTT...");
  while((ret=mqtt.connect())!=0){ //Connect devuelve 0 si esta conectado
    Serial.println(mqtt.connectErrorString(ret));
    Serial.println("Reintentando conectar MQTT en 5 segundos...");
    mqtt.disconnect();
    delay(5000);
    reintentos--;
    if(reintentos==0){
      //Hay que resetear el esp8266
      while(1);
    }
  }
  Serial.println ("MQTT conectado");
 }
```

```
258 void publicarValoresPorDefecto(){
259   //Funcion para publicar los valores por defecto de los datos configurables en la app
260   //En el topico Inicial1 metemos el valor de clase,estado del toldo, modoVacaciones y
          el error
261   topInicial1=(modovacaciones|(clase<<1)|(estadoToldo<<2)|(error<<3));
262   //Publicacion del topico inicial
263   Serial.print(F("\nEnviando topico inicial 1 "));
264   Serial.print(topInicial1);
265   Serial.print("...");
266   if (! pTopInicial1.publish(topInicial1)) {
267     Serial.println(F("Error"));
268   } else {
269     Serial.println(F("OK!"));
270   }
271   //Publicacion del topico clase
272   Serial.print(F("\nEnviando topico clase "));
273   Serial.print(clase);
274   Serial.print("...");
275   if (! pClase.publish(clase)) {
276     Serial.println(F("Error"));
277   } else {
278     Serial.println(F("OK!"));
279   }
280   //Publicacion del topico TemperaturaUmbrales
281   temperaturaumbrales=temperaturabajar1|(temperaturasubir1<<8);
282   Serial.print(F("\nEnviando temperatura umbrales "));
283   Serial.print(temperaturaumbrales);
284   Serial.print("...");
285   if (! pTemperaturaUmbrales.publish(temperaturaumbrales)) {
286     Serial.println(F("Error"));
287   } else {
288     Serial.println(F("OK!"));
289   }
290   //Publicacion del topico HumedadUmbrales
291   humedadumbrales=humedadbajar1|(humedadsubir1<<8);
292   Serial.print(F("\nEnviando humedad umbrales "));
293   Serial.print(humedadumbrales);
294   Serial.print("...");
295   if (! pHumedadUmbrales.publish(humedadumbrales)) {
296     Serial.println(F("Error"));
297   } else {
298     Serial.println(F("OK!"));
299   }
300   //Publicacion del topico LuzSubir1
301   Serial.print(F("\nEnviando luz subir 1 "));
302   Serial.print(luzsubir1);
303   Serial.print("...");
304   if (! pLuzSubir1.publish(luzsubir1)) {
305     Serial.println(F("Error"));
306   } else {
307     Serial.println(F("OK!"));
308   }
309   //Publicacion del topico LuzBajar1
310   Serial.print(F("\nEnviando luz bajar 1 "));
311   Serial.print(luzbajar1);
312   Serial.print("...");
313   if (! pLuzBajar1.publish(luzbajar1)) {
314     Serial.println(F("Error"));
315   } else {
316     Serial.println(F("OK!"));
```

```
317     }
318   }
319   void suscripcionesIniciales(){//Suscripciones al arranque del programa
320     if(mqtt.subscribe(&smodoVacaciones)==false){
321       error=1;
322       Serial.println("Error suscripcion modo vacaciones");
323     }
324     else if(mqtt.subscribe(&sClase)==false){
325       error=2;
326       Serial.println("Error suscripcion clase");
327     }
328     else if(mqtt.subscribe(&sTemperaturaUmbrales)==false){
329       error=3;
330       Serial.println("Error suscripcion temperatura umbrales");
331     }
332     else if(mqtt.subscribe(&sHumedadUmbrales)==false){
333       error=4;
334       Serial.println("Error suscripcion humedad umbrales");
335     }
336     else if(mqtt.subscribe(&sLuzSubir1)==false){
337       error=5;
338       Serial.println("Error suscripcion luz subir 1");
339     }
340     else if(mqtt.subscribe(&sLuzBajar1)==false){
341       error=6;
342       Serial.println("Error suscripcion luz bajar 1");
343     }
344     else if(mqtt.subscribe(&sHoras)==false){
345       error=7;
346       Serial.println("Error suscripcion horas");
347     }
348     else if(mqtt.subscribe(&sCambiarToldo)==false){
349       error=8;
350       Serial.println("Error cambiar toldo");
351     }
352     else{
353       error=0;
354       Serial.println("Suscripciones correctas");
355     }
356   }
357   void suscripciones(){
358     subscription1 = mqtt.readSubscription();
359     if(!subscription1) return;
360
361     if(subscription1==&smodoVacaciones){//Modo vacaciones
362       Serial.print(F("Got: "));
363       Serial.println((char *)smodoVacaciones.lastread);
364       modovacaciones=atoi((char *)smodoVacaciones.lastread);
365       Serial.print("Modo vacaciones: ");
366       Serial.println(modovacaciones);
367     }
368     if(subscription1==&sClase){//Clase del toldo
369       Serial.print(F("Got: "));
370       Serial.println((char *)sClase.lastread);
371       clase=atoi((char *)sClase.lastread);
372       Serial.print("Clase: ");
373       Serial.println(clase);
374     }
375     if(subscription1==&sTemperaturaUmbrales) { //Temperatura umbrales
376       Serial.print(F("Got temperatura umbrales: "));
```

```
377       Serial.println((char *)sTemperaturaUmbrales.lastread);
378       temperaturaumbrales=atoi((char *)sTemperaturaUmbrales.lastread);
379       temperaturabajar1=temperaturaumbrales&0x00FF;
380       temperaturasubir1=(temperaturaumbrales&0xFF00)>>8;
381       Serial.print("Temperatura bajar: ");
382       Serial.println(temperaturabajar1);
383       Serial.print("Temperatura subir: ");
384       Serial.println(temperaturasubir1);
385     }
386     if(subscription1==&sHumedadUmbrales){  //Humedad umbrales
387       Serial.print(F("Got humedad umbrales: "));
388       Serial.println((char *)sHumedadUmbrales.lastread);
389       humedadumbrales=atoi((char *)sHumedadUmbrales.lastread);
390       humedadbajar1=humedadumbrales&0x00FF;
391       humedadsubir1=(humedadumbrales&0xFF00)>>8;
392       Serial.print("Humedad bajar: ");
393       Serial.println(humedadbajar1);
394       Serial.print("Humedad subir: ");
395       Serial.println(humedadsubir1);
396     }
397     if(subscription1==&sLuzSubir1){  //Luz 1 subir
398       Serial.print(F("Got luz subir: "));
399       Serial.println((char *)sLuzSubir1.lastread);
400       luzsubir1=atoi((char *)sLuzSubir1.lastread);
401       Serial.print("Luz subir: ");
402       Serial.println(luzsubir1);
403     }
404     if(subscription1==&sLuzBajar1){ //Luz 1 bajar
405       Serial.print(F("Got luz bajar: "));
406       Serial.println((char *)sLuzBajar1.lastread);
407       luzbajar1=atoi((char *)sLuzBajar1.lastread);
408       Serial.print("Luz bajar: ");
409       Serial.println(luzbajar1);
410     }
411     if(subscription1==&sHoras){//Horas para el modo vacaciones
412       Serial.print(F("Got horas: "));
413       Serial.println((char *)sHoras.lastread);
414       horas=atoi((char *)sHoras.lastread);
415       hora1=horas&0x00FF;
416       hora2=(horas&0xFF00)>>8;
417       Serial.print("Hora1: ");
418       Serial.println(hora1);
419       Serial.print("Hora2: ");
420       Serial.println(hora2);
421     }
422     if(subscription1==&sCambiarToldo){ //Cambiar posicion del toldo
423       Serial.print(F("Got cambiar toldo: "));
424       Serial.println((char *)sCambiarToldo.lastread);
425       cambiartoldo=atoi((char *)sCambiarToldo.lastread);
426       subaj=(cambiartoldo&0x0006)>>1;
427       cambiartoldo=cambiartoldo&0x0001;
428     }
429   }
430
431   //Funcion para enviar el estado del toldo ya que se utiliza bastante en el bucle
        principal
432   void enviarEstadoToldo(){
433
434     if(estadoToldo<0){
435       estadoToldo=0;
```

```
436     }
437     else if(estadoToldo>tiempoMaximo){
438       estadoToldo=tiempoMaximo;
439     }
440
441     Serial.print(F("\nEnviando␣el␣estado␣del␣toldo␣"));
442     Serial.print(estadoToldo);
443     Serial.print("...");
444     if (! pestadoToldo.publish(estadoToldo)) {
445       Serial.println(F("Error"));
446     } else {
447       Serial.println(F("OK!"));
448     }
449   }
450
451   //----------Funciones NTP------------//
452   //Enviar una solicitud NTP al servidor en la direccion dada
453   unsigned long sendNTPpacket(IPAddress& address){
454     Serial.println("Enviando␣un␣paquete␣NTP...");
455     //Ponemos todos los bytes del buffer a 0
456     memset(packetBuffer, 0, NTP_PACKET_SIZE);
457     //Inicializamos con los valores necesarios para formar la soliitud NTP
458     packetBuffer[0]=0b11100011; //LI, Version, Modo
459     packetBuffer[1]=0; //Tipo de reloj
460     packetBuffer[2]=6; //Intervalo de muestreo
461     packetBuffer[3]=0xEC; //Precision del reloj
462     //8 byes a 0 para retardo y dispersion
463     packetBuffer[12]  = 49;
464     packetBuffer[13]  = 0x4E;
465     packetBuffer[14]  = 49;
466     packetBuffer[15]  = 52;
467
468     //Ahora que todos los campos tienen valores,
469     //podemos enviar un paquete solicitando el tiempo
470     udp.beginPacket(address,123);
471     udp.write(packetBuffer,NTP_PACKET_SIZE);
472     udp.endPacket();
473   }
474   int NTP(){
475     unsigned long int highword, lowword,secsSince1900,seventyYears, epoch;
476     //Obtenemos la direccion IP de un servidor aleatorio de la pila
477     WiFi.hostByName(ntpServerName,timeServerIP);
478
479     sendNTPpacket(timeServerIP);//Enviamos un paquete NTP al servidor
480     delay(1000); //Espera a ver si hay una respuesta disponible
481
482     int cb=udp.parsePacket();   //Obtenemos la longitud del paquete
483     if(!cb){
484       Serial.println("Paquete␣no␣disponible");
485     }else{
486       Serial.print("Paquete␣recibido,␣longitud=␣");
487       Serial.println(cb);
488
489       //Una vez que hemos recibido el paquete, leemos los datos de el
490       udp.read(packetBuffer, NTP_PACKET_SIZE); //Lee el paquete que esta en el buffer
491
492       //La marca de tiempo empieza en el byte 40 del paquete recibido y su tamanyo es de
              4 bytes,
493       //o de dos palabras. En primer lugar, extraemos las dos palabras:
494
```

```
495     highword= word(packetBuffer[40], packetBuffer[41]);
496     lowword= word(packetBuffer[42], packetBuffer[43]);
497
498     //Combinamos los 4 bytes (2 palabras) en un entero largo
499     //este es el tiempo NTP (segundos desde el 1 de enero de 1900)
500     secsSince1900=(highword<<16)|lowword;
501     Serial.print("Segundos desde 1900: ");
502     Serial.println(secsSince1900);
503
504     //Ahora convertimos el tiempo NTO en el tiempo diario
505     Serial.print("Unix time: ");
506     //Unix time empieza el 1 de enero de 1970. En segundos es 2208988800
507     seventyYears= 2208988800UL;
508     //Restamos setenta anyos:
509     epoch =secsSince1900-seventyYears;
510     epoch+=3600; //sumamos 3600 segundos ya que Espanya esta en la zona horaria GTM+1
511     Serial.println(epoch);
512     return epoch;
513   }
514 }
515  void obtenerFechaHora(int *datos, unsigned long epoch){
516   int hora,segundos,minutos,anyo,mes,dia, longitudmes;
517   unsigned long dias;
518   //Imprimimos la hora, los minutos y los segundos
519   //Serial.print("El tiempo UTC es: "); //UTC es el tiempo del meridiano de Greenwich (
        GMT)
520   //Conseguimos la hora
521   hora=(epoch%86400L)/3600; //Conseguimos la hora (86400 son los segundos que hay por
        dia)
522   minutos=(epoch%3600)/60;//Conseguimos los minutos (3600 segundos por minuto)
523   segundos=epoch%60;//Conseguimos los segundos
524   //Serial.print(hora);
525   //Serial.print(':');
526   if(((epoch%3600)/60)<10){
527     //En los primeros 10 minutos de cada hora ponemos un 0
528     //Serial.print('0');
529     }
530   //Serial.print(minutos);
531   //Serial.print(':');
532   if((epoch%60)<10){
533     //En los primeros 10 segundos de cada minutos ponemos un 0
534     //Serial.print('0');
535   }
536   //Serial.println(segundos);
537
538   dia= (((epoch/86400) + 4) %7)+1;
539   anyo = 0;
540   dias= 0;
541   while((unsigned)(dias += (LEAP_YEAR(anyo) ? 366 : 365)) <= (epoch/86400))
542   {
543     anyo++;
544   }
545   dias -= LEAP_YEAR(anyo) ? 366 : 365;
546   epoch=(epoch/86400) -dias;
547   dias = 0;
548   mes = 0;
549   longitudmes = 0;
550   for (mes=0; mes<12; mes++) {
551     if (mes==1) { // february
552       if (LEAP_YEAR(anyo)) {
```

```
553           longitudmes=29;
554         } else {
555           longitudmes=28;
556         }
557       } else {
558         longitudmes= diasPorMes[mes];
559       }
560 
561       if (epoch >=longitudmes) {
562         epoch -= longitudmes;
563       } else {
564           break;
565       }
566     }
567     mes += 1;
568     dia = epoch + 1;
569     anyo += 1970;
570    /*Serial.print("La fecha es: ");
571     Serial.print(dia);
572     Serial.print("/");
573     Serial.print(mes);
574     Serial.print("/");
575     Serial.println(anyo);*/
576     datos[0]=segundos;
577     datos[1]=minutos;
578     datos[2]=hora;
579     datos[3]=dia;
580     datos[4]=mes;
581     datos[5]=anyo;
582 }
583 //Funcion para llevar la cuenta interna de la hora
584 long int conseguirSegundos(){
585   return ((millis()/1000+factorCorreccion)); //Devuelve segundos
586 }
587 //Funciones sensores
588 void obtenerTemperaturaHumedad(){
589   int temperaturamedida,humedadmedida,ack;
590   char adquisicionSHT31[]={0x22,0x20};       //0x22: 2 muestras por segundo
591                                              //0x20: Repeatability: medium
592   char datosLeidosSHT31[6];
593   int medido=0;
594   //SENSOR DE TEMPERATURA Y HUMEDAD SHT31
595   //Configuramos el sensor para que trabaje en modo periodico de adquisicion de datos
596   ack=I2C_EnviaTrama(addr_sensortemphum,adquisicionSHT31,2);
597   //comprobarACK(ack);
598   delay(300);
599   I2C_LeeTrama(addr_sensortemphum,datosLeidosSHT31,6);
600 
601   //Comprobamos si los crc recibidos son correctos
602   if(datosLeidosSHT31[2]!=crc8(datosLeidosSHT31,2)){
603     Serial.println("CRC␣temperatura␣erroneo");
604   }
605   if (datosLeidosSHT31[5]!=crc8(datosLeidosSHT31+3,2)){
606     Serial.println("CRC␣humedad␣erroneo");
607   }
608 
609   //Calculamos la temperatura y la humedad
610   temperaturamedida=(datosLeidosSHT31[0]*256)+datosLeidosSHT31[1]; //Obtenemos la
      temperatura medida concatenando los dos primeros datos recibidos
```

```
611   humedadmedida=(datosLeidosSHT31[3]*256)+datosLeidosSHT31[4];//Obtenemos la
          tempreatura medida concatenando los datos de humedad recibidos
612
613   temperaturamedida=-45+((175* temperaturamedida)/65535); //Obtenemos la temperatura en
           grados centigrados
614   humedadmedida=(100* humedadmedida)/65535; //Obtenemos la humedad en %humedad relativa
615
616
617   //Hacemos una primera medicion de los sensores para tener una cierta idea del tiempo
          que hace
618   if(inicio==0){
619     //Reseteamos las variables donde guardamos las sumas de las mediciones
620     temperatura=0;
621     humedad=0;
622     muestra2=0;
623     y=0;
624     //Calculamos los nuevos datos
625     temperaturapromedio=temperaturamedida;
626     humedadpromedio=humedadmedida;
627     //Asignamos un valor a estas variables para que envie los datos
628     inicio=1;
629     medido=1;
630   }
631
632   //Tomamos una muestra de la temperatura y la humedad cada 6 segundos
633   //Una vez tenemos 100 muestras hacemos el promedio
634   if(((millis()/1000)%6)==0){ //Pasamos millis() a segundos y comprobamos que es
          multiplo de 6
635                               //como por cada segundo hace 3 veces el bucle principal
636                               //y solo queremos una muestra,
637     z2++;                    //en una variable contamos 3 y cuando llegue que coja una
            muestra
638   }
639   if(z2==3){      //Tenemos una muestra
640     z2=0;        //Reiniciamos la variable para contar las veces que entra en cada
          segundo
641     tempMuestra[muestra2]=temperaturamedida;  //guardamos la muestra en una matriz
642     humMuestra[muestra2]=humedadmedida;      //hacemos lo mismos que con la temperatura
643     //Cuando y=1 es que ya ha tomado 10 muestras una vez
644     //y queremos ir borrando los datos del vector por las medidas nuevas
645     if(y==1){
646       temperatura-=tempantiguo[muestra2];     //Restamos una medida
647       humedad-=humantiguo[muestra2];
648     }
649     temperatura+=tempMuestra[muestra2];       //vamos sumando la suma de las muestras
650     tempantiguo[muestra2]=tempMuestra[muestra2];
651
652     humedad+=humMuestra[muestra2];
653     humantiguo[muestra2]=humMuestra[muestra2];
654     muestra2++;
655
656     if(muestra2==100){       //LLegamos a 100 muestras a los 10 minutos y hacemos un
          promedio de ellas
657      //Guardamos en un vector el promedio de las 100 muestras
658      temperaturapromedio=temperatura/100;
659      humedadpromedio=humedad/100;
660
661      muestra2=0;
662      y=1;
663      medido=1;
```

```
664     }
665   }
666
667   if(medido==1){
668     //Una vez tenemos calculada la temperatura la enviamos por MQTT
669     Serial.print(F("\nEnviando temperatura "));
670     Serial.print(temperaturapromedio);
671     Serial.print("...");
672     if (! pTemperatura.publish(temperaturapromedio)) {
673       Serial.println(F("Error"));
674     } else {
675       Serial.println(F("OK!"));
676     }
677     //Ahora enviamos la humedad
678     Serial.print(F("\nEnviando humedad "));
679     Serial.print(humedadpromedio);
680     Serial.print("...");
681     if (! pHumedad.publish(humedadpromedio)) {
682       Serial.println(F("Error"));
683     } else {
684       Serial.println(F("OK!"));
685     }
686   }
687 }
688 void obtenerLuz(){
689   char ComandoSensorLuz[]={0x13}; //4lux resolucion
690                                   //16ms tarda en medir
691   char datosLeidosLuz[2];
692   int medido=0,luzmedida,ack;
693   ack=I2C_EnviaTrama(addr_sensorluz,ComandoSensorLuz,1);
694   //comprobarACK(ack);
695
696
697   I2C_LeeTrama(addr_sensorluz,datosLeidosLuz,2);
698   luzmedida=(datosLeidosLuz[1])|(datosLeidosLuz[0]<<8);//Concatenamos los datos leidos
          para
699                                                        //obtener la cantidad de luz
                                                             medida
700   luzmedida=luzmedida/1.2;
701
702   //Realizamos el promedio de la luz, para ello tomamos 100 muestras en 10 minutos
703   //Ademas hacemos una media de los 10 ultimos promedios
704   if(inicio5==0){
705     //Reseteamos las variables donde guardamos las sumas de las mediciones
706     muestra3=0;
707     luz=0;
708     z=0;
709     //Calculamos los nuevos datos
710     luzpromedio=luzmedida;
711     Serial.print("Luz: ");
712     Serial.println(luzpromedio);
713     inicio5=1;
714     medido=1;
715    }
716
717   //Tomamos una muestra de la luz cada 6 segundos
718   //Una vez tenemos 100 muestras hacemos el promedio
719   if(((millis()/1000)%6)==0){ //Pasamos millis() a segundos y comprobamos que es
          multiplo de 6
720                               //como por cada segundo hace el bucle principal 3 veces
```

```
721                                  //y solo queremos una muestra,
722       z3++;                     //en una variable contamos 3 y cuando llegue que coja una
              muestra
723     }
724     if(z3==3){                          //Tenemos una muestra
725       z3=0;                             //Reiniciamos la variable para contar las veces que
              entra en cada segundo
726       luzMuestra[muestra3]=luzmedida; //guardamos la muestra en una matriz
727
728       if(z==1){
729         luz-=luzantiguo[muestra3];     //Restamos una medida
730       }
731       luz+=luzMuestra[muestra3];       //vamos sumando la suma de las muestras
732       luzantiguo[muestra3]=luzMuestra[muestra3];
733       muestra3++;
734
735       if(muestra3==100){//LLegamos a 100 muestras a los 10 minutos y hacemos un promedio
              de ellas
736       luzpromedio=luz/100;
737       muestra3=0;
738       z=1;
739       medido=1;
740      }
741     }
742
743     if(medido==1){
744       //Una vez tenemos calculada la temperatura la enviamos por MQTT
745       Serial.print(F("\nEnviando␣luz␣"));
746       Serial.print(luzpromedio);
747       Serial.print("...");
748       if (! pLuz.publish(luzpromedio)) {
749         Serial.println(F("Error"));
750       } else {
751         Serial.println(F("OK!"));
752       }
753     }
754 }
755
756 void obtenerViento(){
757     int medido=0;
758     //Hacemos una primera medicion del tiempo para saber la velocidad del viento
759     if(arranque==0){
760       vientopromedio=system_adc_read();
761       arranque=1;
762       medido=1;
763       }
764     if(modoautomatico==1 && z1>3){
765       z1=0;
766     }
767     //Despues vamos tomando muestras cada 3 segundos y hacemos un promedio de 100
          muestras (5 minutos)
768     if(((millis()/1000)%3)==0){//Como hace el bucle principal 3 veces por segundo,
          empleamos una variable para que solo
769                               //cuando valga 3 tome la muestra en modo automatico,
770                               //mientras que en modo manual lo hace 50 veces
771       z1++;
772     }
773
774
775     if((z1==3 && modoautomatico==1) || (z1==50 && modoautomatico==0)){
```

```
776       z1=0;
777       vienMuestra[muestra1]=system_adc_read();  //Tomamos una muestra
778       //Si x vale 1, ya se han tomado las 100 primeras muestras
779       //y queremos ir borrando las mas antiguas
780       if(x==1){
781         viento-=vientoantiguo[muestra1];
782       }
783       viento+=vienMuestra[muestra1];                    //Vamos sumando las muestras
784       vientoantiguo[muestra1]=vienMuestra[muestra1];   //Guardamos el valor de cada
              muestra
785                                                         //en concreto para luego borrarla
786       muestra1++;
787       if(muestra1==100){ //Cuando tenemos 100 muestras calculamos el promedio del
788                          //viento durante los ultimos 5 minutos
789        vientopromedio=viento/100;
790        muestra1=0;
791        x=1;
792        medido=1;
793       }
794     }
795
796   if(medido==1){
797       vientokm=vientopromedio/5.68;
798       //Ahora enviamos el viento
799       Serial.print(F("\nEnviando viento "));
800       Serial.print(vientokm);
801       Serial.print("...");
802       if (! pViento.publish(vientokm)) {
803         Serial.println(F("Error"));
804       } else {
805         Serial.println(F("OK!"));
806       }
807   }
808 }
809 //-------Funciones de funcionamiento------///
810 void determinarNoche(){
811
812   obtenerFechaHora(datosFechaHora,conseguirSegundos()); //Obtenemos la hora actual
813   //Simulacion de la luz segun el mes
814   //Segun el mes y la hora sera de noche antes o mas tarde
815   if(datosFechaHora[4]==1 || datosFechaHora[4]==12 || datosFechaHora[4]==11){ //Enero,
          Noviembre y Diciembre
816     if(datosFechaHora[2]<8 || datosFechaHora[2]>=18 ){
817       noche=1;
818     }
819     else {
820       noche=0;
821     }
822   }
823   else if(datosFechaHora[4]==2){ //Febrero
824     if(datosFechaHora[2]<8 || datosFechaHora[2]>=19 ){
825       noche=1;
826     }
827     else {
828       noche=0;
829     }
830   }
831   else if(datosFechaHora[4]==3 || datosFechaHora[4]==9 || datosFechaHora[4]==10){//
          Marzo,Septiembre y Octubre
832    if(datosFechaHora[2]<8 || datosFechaHora[2]>=20){
```

```
833         noche=1;
834       }
835       else {
836         noche=0;
837       }
838     }
839     else if(datosFechaHora[4]==4 || datosFechaHora[4]==5 || datosFechaHora[4]==8){//Abril
          , Mayo y Agosto
840       if(datosFechaHora[2]<7 || datosFechaHora[2]>=21){
841         noche=1;
842       }
843       else {
844         noche=0;
845       }
846     }
847     else if(datosFechaHora[4]==6 || datosFechaHora[4]==7){//Junio y Julio
848       if(datosFechaHora[2]<7 || datosFechaHora[2]>=22 ){
849         noche=1;
850       }
851       else{
852         noche=0;
853       }
854     }
855 }
856 void determinarClaseToldo(){
857   //Hay 3 tipos de clase para los toldos
858   //Segun la clase tienen mas o menos resistencia al viento
859   if(clase==1){
860     vientomaximo=159;  //28 km/h
861   }else if(clase==2){
862     vientomaximo=216; //38 km/h
863   }else if (clase==3){
864     vientomaximo=278;  //49 km/h
865   }
866 }
867 //Funcion generica para calcular la probabilidad de subir o bajar segun los umbrales
868 int calculaProbabilidad(int umbralInferior, int umbralSuperior, float valorX, int
      pendiente){
869   float a,b;
870   if(pendiente==0){ //Pendiente negativa
871     a=100/(umbralInferior-umbralSuperior);
872     b=100-a*umbralInferior;
873     if(valorX<=umbralInferior){
874       return 100;
875     }
876     else if(valorX>=umbralSuperior){
877       return 0;
878     }
879     else{
880       return (a*valorX+b);
881     }
882   }
883   else{  //Pendiente positiva
884     a=-100/(umbralInferior-umbralSuperior);
885     b=0-a*umbralInferior;
886     if(valorX<=umbralInferior){
887       return 0;
888     }
889     else if(valorX>=umbralSuperior){
890       return 100;
```

```
891       }
892       else{
893         return (a*valorX+b);
894       }
895     }
896   }
897   //Ahora establecemos las condiciones para subir o bajar el toldo automaticamente
898   void modoAutomatico(){
899
900     //Curva de subida de la temperatura
901     subir1=calculaProbabilidad( temperaturasubir1,temperaturabajar1,temperaturapromedio
          ,0);
902     //Curva de bajada de la temperatura
903     bajar1=calculaProbabilidad( temperaturasubir1,temperaturabajar1,temperaturapromedio
          ,1);
904
905     //Curva de bajada de la humedad
906     bajar2=calculaProbabilidad(humedadbajar1,humedadsubir1,humedadpromedio,0);
907     //Curva de subida de la humedad
908     subir2=calculaProbabilidad(humedadbajar1,humedadsubir1,humedadpromedio,1);
909
910     //Curva de subida de la luz
911     subir3=calculaProbabilidad(luzsubir1,luzbajar1,luzpromedio,0);
912     //Curva de bajada de la luz
913     bajar3=calculaProbabilidad(luzsubir1,luzbajar1,luzpromedio,1);
914
915     subir=subir1+subir2+subir3;//Suma de las 3 probabilidades de subida
916     bajar=bajar1+bajar2+bajar3;//Suma de las 3 probabilidades de bajada
917
918     determinarNoche();
919     if(noche==1){//Si es de noche, el toldo debe estar subido
920       m1=1;
921       m2=0;
922     }
923     else{
924       if(subir>=bajar){//Si es de dia, se evaluan las condiciones
925         m1=1;
926         m2=0;
927       }
928       else{
929         m1=0;
930         m2=1;
931       }
932     }
933   }
934   void cambiarToldo(){
935     //Podemos cambiar la posicion del toldo desde la app si el toldo esta subido o bajado
936     if(subaj==2){//El toldo se mantiene en la posicion anterior
937       estadoToldo=estadoToldo;
938     }
939     else{
940      if(subaj==1){//Toldo subido-->lo bajamos
941         m1=0;
942         m2=1;
943       }
944      else if(subaj==0){//Toldo bajado-->lo subimos
945         m1=1;
946         m2=0;
947       }
948
```

```
949       //Calculamos el estado del toldo
950       if(m1==1)
951         estadoToldo=estadoToldo-tfinal;
952       else
953         estadoToldo=estadoToldo+tfinal;
954     }
955     if(estadoToldo<0){
956      estadoToldo=0;
957     }
958     else if(estadoToldo>tiempoMaximo){
959       estadoToldo=tiempoMaximo;
960     }
961     if(estadoToldoantiguo!=estadoToldo && estadoToldo%5==0){
962       enviarEstadoToldo();
963     }
964 }
965 void loop() {
966 
967   while(error!=0){
968     suscripcionesIniciales();
969     if(arranque1==0 || error==0){
970       Serial.print(F("\nEnviando␣resultado␣de␣las␣suscripciones␣"));
971       Serial.print(error);
972       Serial.print("...");
973       arranque1=1;
974       if (! pError.publish(error)) {
975       Serial.println(F("Error"));
976        } else {
977       Serial.println(F("OK!"));
978        }
979     }
980   }
981 
982   t0=millis(); //Obtenemos los milisegundos desde que arranco el programa
983   //Comprobamos si la conexion MQTT esta activa y si no lo esta
984   //que se conecte
985   MQTT_connect();
986   if(arranque==0){
987     publicarValoresPorDefecto();
988   }
989   //Leemos los pines de los interruptores
990   pulsador1=digitalRead(gpio4);
991   pulsador2=digitalRead(gpio5);
992   //Nos suscribimos a todos los topicos
993   suscripciones();
994   //Obtenemos la fecha y la hora
995   if(arranque==1){
996     obtenerFechaHora(datosFechaHora,conseguirSegundos());
997     if(datosFechaHora[2]==22){
998       nvueltas=0;
999     }
1000  }
1001  if((datosFechaHora[2]==23 && datosFechaHora[1]==1 && nvueltas==0) || arranque==0){
1002    epoch2=NTP();
1003    obtenerFechaHora(datosFechaHora,epoch2);
1004    factorCorreccion=epoch2-(millis()/1000);
1005    nvueltas=1;
1006  }
1007  //Miramos que tipo de clase es el toldo para determinar
1008  //la velocidad del viento maxima que soporta el toldo
```

```
1009    determinarClaseToldo();
1010
1011    //Obtenemos un promedio del viento
1012    obtenerViento();
1013    if(vientopromedio>vientomaximo){//Recogemos el toldo y no dejamos que se baje
1014      m1=1;
1015      m2=0;
1016    }else{
1017      //Evaluamos la posicion de los interruptores
1018      //Modo Manual
1019      if(pulsador1==0 && pulsador2==0 && pulsador1antiguo==0 && pulsador2antiguo==0){
1020         if(cambiartoldo==1){//Desde la app podemos subir o bajar el toldo
1021          cambiarToldo();
1022         }
1023         else{//Si no cambiamos en la app, dejamos todo apagado
1024           m1=0;
1025           m2=0;
1026           if(modoautomatico==1){
1027             enviarEstadoToldo();
1028           }
1029         }
1030         aux1=0;
1031         aux2=0;
1032         aux3=0;
1033         modoautomatico=0;
1034         inicio=0;
1035         inicio5=0;
1036      }
1037      //Subir toldo
1038      else if(pulsador1==1 &&  pulsador1antiguo==0 && pulsador2==0){
1039         t1inicial=millis();
1040         Serial.print("Tiempo␣inicial␣1:␣");
1041         Serial.println(t1inicial);
1042        m1=1;
1043        m2=0;
1044         Serial.println("MODO␣2");
1045         aux1=1;
1046         aux2=0;
1047         aux3=0;
1048         clase=3;
1049         modoautomatico=0;
1050         inicio=0;
1051      }
1052      else if(pulsador1==0 && pulsador1antiguo==1 && pulsador2==0){
1053        m1=0;
1054        m2=0;
1055        if(aux1==1){
1056        t1final=millis();
1057        Serial.print("Tiempo␣final␣1:␣");
1058        Serial.println(t1final);
1059        Serial.println("MODO␣3");
1060        aux1=0;
1061        modoautomatico=0;
1062        tiempoMoviendo=(t1final-t1inicial);
1063        estadoToldo=estadoToldo-tiempoMoviendo;
1064        aux2=0;
1065        aux3=0;
1066        inicio=0;
1067        enviarEstadoToldo();
1068      }
```

```
1069  }
1070      else if(pulsador1==0 && pulsador1antiguo==1 && pulsador2==1){
1071        enviarEstadoToldo();
1072        m1=0;
1073        m2=1;
1074        t1automatico=millis();
1075        aux1=0;
1076        aux2=0;
1077        aux3=1;
1078        aux4=0;
1079        modoautomatico=0;
1080        inicio=0;
1081        Serial.print("Tiempo automatico 1: ");
1082        Serial.println(t1automatico);
1083        Serial.println("MODO 4");
1084
1085      }
1086      //Bajar toldo
1087      else if(pulsador2==1 && pulsador2antiguo==0 && pulsador1==0){
1088         m1=0;
1089         m2=1;
1090         t2inicial=millis();
1091         Serial.print("Tiempo inicial 2: ");
1092         Serial.println(t2inicial);
1093         Serial.println("MODO 5");
1094         aux1=0;
1095         aux2=1;
1096         aux3=0;
1097         modoautomatico=0;
1098         inicio=0;
1099      }
1100      else if(pulsador2==0 && pulsador2antiguo==1 && pulsador1==0){
1101        m1=0;
1102        m2=0;
1103        modoautomatico=0;
1104        if(aux2==1){
1105          t2final=millis();
1106          Serial.print("Tiempo final 2: ");
1107          Serial.println(t2final);
1108          Serial.println("MODO 6");
1109          aux2=0;
1110          tiempoMoviendo=(t2final-t2inicial);
1111          estadoToldo=estadoToldo+tiempoMoviendo;
1112        }
1113        aux1=0;
1114        aux3=0;
1115        inicio=0;
1116        enviarEstadoToldo();
1117      }
1118      else if(pulsador2==0 && pulsador2antiguo==1 && pulsador1==1){
1119       enviarEstadoToldo();
1120        m1=1;
1121        m2=0;
1122        aux1=0;
1123        aux2=0;
1124        aux3=1;
1125        aux4=1;
1126        inicio=0;
1127        modoautomatico=0;
1128        t1automatico=millis();
```

```
1129        Serial.print("Tiempo automatico 1: ");
1130        Serial.println(t1automatico);
1131        Serial.println("MODO 7");
1132
1133      }
1134      //Modo automatico
1135      else if(pulsador1==1 && pulsador2==1 && (pulsador1antiguo==0 || pulsador2antiguo
              ==0)){
1136        m1=0;
1137        m2=0;
1138        modoautomatico=0;
1139        if(aux3==1)
1140        {
1141          t2automatico=millis();
1142          Serial.print("Tiempo automatico 2: ");
1143          Serial.println(t2automatico);
1144          tiempoMoviendo=(t2automatico-t1automatico);
1145          if(aux4==0)
1146              estadoToldo=estadoToldo+tiempoMoviendo;
1147           else if(aux4==1)
1148            estadoToldo=estadoToldo-tiempoMoviendo;
1149           Serial.println("MODO 8");
1150           aux3=0;
1151        }
1152        aux1=0;
1153        aux2=0;
1154        enviarEstadoToldo();
1155      }
1156      else if(pulsador1==1 && pulsador2==1){
1157        if(pulsador1antiguo==0 || pulsador2antiguo==0){
1158          m1=0;
1159          m2=0;
1160        }
1161        aux1=0;
1162        aux2=0;
1163        aux3=0;
1164        if(cambiartoldo==1){
1165          cambiarToldo();
1166        }
1167        else{
1168          if(modovacaciones==1){//Modo vacaciones activado
1169             modoautomatico=0;
1170             if(arranque2==0){
1171              arranque2=1;
1172              horas=hora1|(hora2<<8);
1173              Serial.print(F("\nEnviando horas "));
1174              Serial.print(horas);
1175              Serial.print("...");
1176              if (! pHoras.publish(horas)) {
1177                Serial.println(F("Error"));
1178              } else {
1179                Serial.println(F("OK!"));
1180              }
1181             }
1182             //Comprobamos si es una epoca del anyo en la que hace buen tiempo
1183             //ya que solo se baja el toldo si hace buena temperatura
1184            if(datosFechaHora[4]==5 || datosFechaHora[4]==6|| datosFechaHora[4]==7
1185                  || datosFechaHora[4]==8 || datosFechaHora[4]==9){
1186               obtenerFechaHora(datosFechaHora,conseguirSegundos()); //Obtenemos la hora
                      actual
```

```
1187                if(datosFechaHora[2]==hora1 && inicio1==0){
1188                   //Si la hora coincide con las horas de los topicos, calculamos la hora
                           aleatoria
1189                   numeroAleatorio=random(1,4); //Segun el numero aleatorio aplicamos una
                           mascara u otra
1190                   //Nos quedamos con la parte baja de millis() para sumarlos al
1191                   //tiempo actual y de esta manera obtener una hora aleatoria
1192                   //Para ello, aplicamos una mascara u otra para dar mas aleotoridad
1193                   if(numeroAleatorio==1){
1194                     Serial.println(numeroAleatorio);
1195                     aleatorio=millis()&0xFF;
1196                   }
1197                   else if(numeroAleatorio==2){
1198                     Serial.println(numeroAleatorio);
1199                     aleatorio=millis()&0x3FF;
1200                   }
1201                   else if(numeroAleatorio==3){
1202                     Serial.println(numeroAleatorio);
1203                     aleatorio=millis()&0x4FF;
1204                   }
1205                   //Conseguimos la hora aleatoria
1206                   obtenerFechaHora(datosAleatorios, (conseguirSegundos()+aleatorio));
1207                   inicio1=1;
1208                   inicio2=0;
1209                   inicio3=0;
1210                }
1211                else if(datosFechaHora[2]==hora2 && inicio2==0){
1212                   //Nos quedamos con la parte baja de millis() para sumarlos al tiempo
                           actual
1213                   //y de esta manera obtener una hora aleatoria
1214                   numeroAleatorio=random(1,4);
1215                   if(numeroAleatorio==1){
1216                     Serial.println(numeroAleatorio);
1217                     aleatorio=millis()&0xFF;
1218                   }
1219                   else if(numeroAleatorio==2){
1220                     Serial.println(numeroAleatorio);
1221                     aleatorio=millis()&0x3FF;
1222                   }
1223                   else if(numeroAleatorio==3){
1224                     Serial.println(numeroAleatorio);
1225                     aleatorio=millis()&0x4FF;
1226                   }
1227                   obtenerFechaHora(datosAleatorios, (conseguirSegundos()+aleatorio));
1228                   inicio1=0;
1229                   inicio2=1;
1230                   inicio3=0;
1231                }
1232               //Comprobamos si la hora actual coincide con la hora aleatoria para actuar
                      sobre el toldo
1233               if(datosFechaHora[2]==datosAleatorios[2] && datosFechaHora[1]==
                      datosAleatorios[1] && inicio3==0){
1234                 inicio3=1;
1235                 if(estadoToldo<15){ //Si el toldo esta casi subido lo bajamos
1236                   m1=0;
1237                   m2=1;
1238                   estadoToldo=tiempoMaximo;
1239                 }
1240                 else{//Si el toldo esta casi bajado lo subimos
1241                   m1=1;
```

```
1242                    m2=0;
1243                    estadoToldo=0;
1244                  }
1245                  //Publicamos el estado del toldo
1246                  enviarEstadoToldo();
1247                }
1248              }
1249            }
1250            else if(modovacaciones==0){
1251              modoautomatico=1;
1252              obtenerTemperaturaHumedad();
1253              obtenerLuz();
1254              modoAutomatico();
1255              if(m1==1)
1256                estadoToldo=estadoToldo-tfinal;
1257              else
1258                estadoToldo=estadoToldo+tfinal;
1259
1260              if(estadoToldo<0){
1261                 estadoToldo=0;
1262              }
1263              else if(estadoToldo>tiempoMaximo){
1264                 estadoToldo=tiempoMaximo;
1265              }
1266              if(estadoToldoantiguo!=estadoToldo && estadoToldo%100==0){
1267                enviarEstadoToldo();
1268            }
1269           }
1270          }
1271        }
1272      }
1273
1274      if(m1==0 && m2==0){
1275          digitalWrite(actuador_down,LOW);
1276          digitalWrite(actuador_up,LOW);
1277      }else if(m1==1 && m2==0){
1278          digitalWrite(actuador_down,LOW);
1279          digitalWrite(actuador_up,HIGH);
1280      }else if(m1==0 && m2==1){
1281          digitalWrite(actuador_down,HIGH);
1282          digitalWrite(actuador_up,LOW);
1283      }
1284
1285      pulsador1antiguo=pulsador1;
1286      pulsador2antiguo=pulsador2;
1287      estadoToldoantiguo=estadoToldo;
1288      modoautomaticoantiguo=modoautomatico;
1289      tfinal=millis()-t0;
1290
1291      while((millis()-t0)<20);//El programa tiene que tardar 20 ms en ejecutarse
1292  }//Cierra loop
```

Apéndice E

Adafruit_MQTT.cpp (sin modificar)

Adafruit_MQTT_Antiguo.cpp

```
// The MIT License (MIT)
//
// Copyright (c) 2015 Adafruit Industries
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in all
// copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
// SOFTWARE.
#include "Adafruit_MQTT.h"

#if defined(ARDUINO_SAMD_ZERO) || defined(ARDUINO_SAMD_MKR1000)
static char *dtostrf (double val, signed char width, unsigned char prec, char *sout) {
  char fmt[20];
  sprintf(fmt, "%%%d.%df", width, prec);
  sprintf(sout, fmt, val);
  return sout;
}
#endif

#if defined(ESP8266)
int strncasecmp(const char * str1, const char * str2, int len) {
    int d = 0;
    while(len--) {
        int c1 = tolower(*str1++);
        int c2 = tolower(*str2++);
        if(((d = c1 - c2) != 0) || (c2 == '\0')) {
    return d;
        }
    }
    return 0;
```

```
}
#endif


void printBuffer(uint8_t *buffer, uint16_t len) {
  DEBUG_PRINTER.print('\t');
  for (uint16_t i=0; i<len; i++) {
    if (isprint(buffer[i]))
      DEBUG_PRINTER.write(buffer[i]);
    else
      DEBUG_PRINTER.print("␣");
    DEBUG_PRINTER.print(F("␣[0x"));
    if (buffer[i] < 0x10)
      DEBUG_PRINTER.print("0");
    DEBUG_PRINTER.print(buffer[i],HEX);
    DEBUG_PRINTER.print("],␣");
    if (i %8 == 7) {
      DEBUG_PRINTER.print("\n\t");
    }
  }
  DEBUG_PRINTER.println();
}

/* Not used now, but might be useful in the future
static uint8_t *stringprint(uint8_t *p, char *s) {
  uint16_t len = strlen(s);
  p[0] = len >> 8; p++;
  p[0] = len & 0xFF; p++;
  memmove(p, s, len);
  return p+len;
}
*/

static uint8_t *stringprint(uint8_t *p, const char *s, uint16_t maxlen=0) {
  // If maxlen is specified (has a non-zero value) then use it as the maximum
  // length of the source string to write to the buffer.  Otherwise write
  // the entire source string.
  uint16_t len = strlen(s);
  if (maxlen > 0 && len > maxlen) {
    len = maxlen;
  }
  /*
  for (uint8_t i=0; i<len; i++) {
    Serial.write(pgm_read_byte(s+i));
  }
  */
  p[0] = len >> 8; p++;
  p[0] = len & 0xFF; p++;
  strncpy((char *)p, s, len);
  return p+len;
}


// Adafruit_MQTT Definition ////////////////////////////////////////////////

Adafruit_MQTT::Adafruit_MQTT(const char *server,
                             uint16_t port,
                             const char *cid,
                             const char *user,
                             const char *pass) {
```

```
104   servername = server;
105   portnum = port;
106   clientid = cid;
107   username = user;
108   password = pass;
109
110   // reset subscriptions
111   for (uint8_t i=0; i<MAXSUBSCRIPTIONS; i++) {
112     subscriptions[i] = 0;
113   }
114
115   will_topic = 0;
116   will_payload = 0;
117   will_qos = 0;
118   will_retain = 0;
119
120   packet_id_counter = 0;
121
122 }
123
124
125 Adafruit_MQTT::Adafruit_MQTT(const char *server,
126                              uint16_t port,
127                              const char *user,
128                              const char *pass) {
129   servername = server;
130   portnum = port;
131   clientid = "";
132   username = user;
133   password = pass;
134
135   // reset subscriptions
136   for (uint8_t i=0; i<MAXSUBSCRIPTIONS; i++) {
137     subscriptions[i] = 0;
138   }
139
140   will_topic = 0;
141   will_payload = 0;
142   will_qos = 0;
143   will_retain = 0;
144
145   packet_id_counter = 0;
146
147 }
148
149 int8_t Adafruit_MQTT::connect() {
150   // Connect to the server.
151   if (!connectServer())
152     return -1;
153
154   // Construct and send connect packet.
155   uint8_t len = connectPacket(buffer);
156   if (!sendPacket(buffer, len))
157     return -1;
158
159   // Read connect response packet and verify it
160   len = readFullPacket(buffer, MAXBUFFERSIZE, CONNECT_TIMEOUT_MS);
161   if (len != 4)
162     return -1;
163   if ((buffer[0] != (MQTT_CTRL_CONNECTACK << 4)) || (buffer[1] != 2))
```

```
164       return -1;
165     if (buffer[3] != 0)
166       return buffer[3];
167
168     // Setup subscriptions once connected.
169     for (uint8_t i=0; i<MAXSUBSCRIPTIONS; i++) {
170       // Ignore subscriptions that aren't defined.
171       if (subscriptions[i] == 0) continue;
172
173       boolean success = false;
174       for (uint8_t retry=0; (retry<3) && !success; retry++) { // retry until we get a
              suback
175         // Construct and send subscription packet.
176         uint8_t len = subscribePacket(buffer, subscriptions[i]->topic, subscriptions[i]->
              qos);
177         if (!sendPacket(buffer, len))
178     return -1;
179
180         if(MQTT_PROTOCOL_LEVEL < 3) // older versions didn't suback
181     break;
182
183         // Check for SUBACK if using MQTT 3.1.1 or higher
184         // TODO: The Server is permitted to start sending PUBLISH packets matching the
185         // Subscription before the Server sends the SUBACK Packet. (will really need to
              use callbacks - ada)
186
187         //Serial.println("\t**looking for suback");
188         if (processPacketsUntil(buffer, MQTT_CTRL_SUBACK, SUBACK_TIMEOUT_MS)) {
189     success = true;
190     break;
191         }
192         //Serial.println("\t**failed, retrying!");
193       }
194       if (! success) return -2; // failed to sub for some reason
195     }
196
197     return 0;
198   }
199
200   int8_t Adafruit_MQTT::connect(const char *user, const char *pass)
201   {
202     username = user;
203     password = pass;
204     return connect();
205   }
206
207   uint16_t Adafruit_MQTT::processPacketsUntil(uint8_t *buffer, uint8_t waitforpackettype,
        uint16_t timeout) {
208     uint16_t len;
209     while (len = readFullPacket(buffer, MAXBUFFERSIZE, timeout)) {
210
211       //DEBUG_PRINT("Packet read size: "); DEBUG_PRINTLN(len);
212       // TODO: add subscription reading & call back processing here
213
214       if ((buffer[0] >> 4) == waitforpackettype) {
215         //DEBUG_PRINTLN(F("Found right packet"));
216         return len;
217       } else {
218         ERROR_PRINTLN(F("Dropped␣a␣packet"));
219       }
```

```
220   }
221   return 0;
222 }
223
224 uint16_t Adafruit_MQTT::readFullPacket(uint8_t *buffer, uint16_t maxsize, uint16_t
      timeout) {
225   // will read a packet and Do The Right Thing with length
226   uint8_t *pbuff = buffer;
227
228   uint8_t rlen;
229
230   // read the packet type:
231   rlen = readPacket(pbuff, 1, timeout);
232   if (rlen != 1) return 0;
233
234   DEBUG_PRINT(F("Packet Type:\t")); DEBUG_PRINTBUFFER(pbuff, rlen);
235   pbuff++;
236
237   uint32_t value = 0;
238   uint32_t multiplier = 1;
239   uint8_t encodedByte;
240
241   do {
242     rlen = readPacket(pbuff, 1, timeout);
243     if (rlen != 1) return 0;
244     encodedByte = pbuff[0]; // save the last read val
245     pbuff++; // get ready for reading the next byte
246     uint32_t intermediate = encodedByte & 0x7F;
247     intermediate *= multiplier;
248     value += intermediate;
249     multiplier *= 128;
250     if (multiplier > (128UL*128UL*128UL)) {
251       DEBUG_PRINT(F("Malformed packet len\n"));
252       return 0;
253     }
254   } while (encodedByte & 0x80);
255
256   DEBUG_PRINT(F("Packet Length:\t")); DEBUG_PRINTLN(value);
257
258   if (value > (maxsize - (pbuff-buffer) - 1)) {
259       DEBUG_PRINTLN(F("Packet too big for buffer"));
260       rlen = readPacket(pbuff, (maxsize - (pbuff-buffer) - 1), timeout);
261   } else {
262     rlen = readPacket(pbuff, value, timeout);
263   }
264   //DEBUG_PRINT(F("Remaining packet:\t")); DEBUG_PRINTBUFFER(pbuff, rlen);
265
266   return ((pbuff - buffer)+rlen);
267 }
268
269 const __FlashStringHelper* Adafruit_MQTT::connectErrorString(int8_t code) {
270    switch (code) {
271       case 1: return F("The Server does not support the level of the MQTT protocol
          requested");
272       case 2: return F("The Client identifier is correct UTF-8 but not allowed by the
          Server");
273       case 3: return F("The MQTT service is unavailable");
274       case 4: return F("The data in the user name or password is malformed");
275       case 5: return F("Not authorized to connect");
276       case 6: return F("Exceeded reconnect rate limit. Please try again later.");
```

```
277       case 7: return F("You have been banned from connecting. Please contact the MQTT
              server administrator for more details.");
278       case -1: return F("Connection failed");
279       case -2: return F("Failed to subscribe");
280       default: return F("Unknown error");
281     }
282 }
283
284 bool Adafruit_MQTT::disconnect() {
285
286   // Construct and send disconnect packet.
287   uint8_t len = disconnectPacket(buffer);
288   if (! sendPacket(buffer, len))
289     DEBUG_PRINTLN(F("Unable to send disconnect packet"));
290
291   return disconnectServer();
292
293 }
294
295
296 bool Adafruit_MQTT::publish(const char *topic, const char *data, uint8_t qos) {
297     return publish(topic, (uint8_t*)(data), strlen(data), qos);
298 }
299
300 bool Adafruit_MQTT::publish(const char *topic, uint8_t *data, uint16_t bLen, uint8_t
      qos) {
301   // Construct and send publish packet.
302   uint16_t len = publishPacket(buffer, topic, data, bLen, qos);
303   if (!sendPacket(buffer, len))
304     return false;
305
306   // If QOS level is high enough verify the response packet.
307   if (qos > 0) {
308     len = readFullPacket(buffer, MAXBUFFERSIZE, PUBLISH_TIMEOUT_MS);
309     DEBUG_PRINT(F("Publish QOS1+ reply:\t"));
310     DEBUG_PRINTBUFFER(buffer, len);
311     if (len != 4)
312       return false;
313     if ((buffer[0] >> 4) != MQTT_CTRL_PUBACK)
314       return false;
315     uint16_t packnum = buffer[2];
316     packnum <<= 8;
317     packnum |= buffer[3];
318
319     // we increment the packet_id_counter right after publishing so inc here too to
          match
320     packnum++;
321     if (packnum != packet_id_counter)
322       return false;
323   }
324
325   return true;
326 }
327
328 bool Adafruit_MQTT::will(const char *topic, const char *payload, uint8_t qos, uint8_t
      retain) {
329
330   if (connected()) {
331     DEBUG_PRINT(F("Will defined after connect"));
332     return false;
```

```
333   }
334
335   will_topic = topic;
336   will_payload = payload;
337   will_qos = qos;
338   will_retain = retain;
339
340   return true;
341
342 }
343
344 bool Adafruit_MQTT::subscribe(Adafruit_MQTT_Subscribe *sub) {
345   uint8_t i;
346   // see if we are already subscribed
347   for (i=0; i<MAXSUBSCRIPTIONS; i++) {
348     if (subscriptions[i] == sub) {
349       DEBUG_PRINTLN(F("Already subscribed"));
350       return true;
351     }
352   }
353   if (i==MAXSUBSCRIPTIONS) { // add to subscriptionlist
354     for (i=0; i<MAXSUBSCRIPTIONS; i++) {
355       if (subscriptions[i] == 0) {
356         DEBUG_PRINT(F("Added sub ")); DEBUG_PRINTLN(i);
357         subscriptions[i] = sub;
358         return true;
359       }
360     }
361   }
362
363   DEBUG_PRINTLN(F("no more subscription space :("));
364   return false;
365 }
366
367 bool Adafruit_MQTT::unsubscribe(Adafruit_MQTT_Subscribe *sub) {
368   uint8_t i;
369
370   // see if we are already subscribed
371   for (i=0; i<MAXSUBSCRIPTIONS; i++) {
372
373     if (subscriptions[i] == sub) {
374
375       DEBUG_PRINTLN(F("Found matching subscription and attempting to unsubscribe."));
376
377       // Construct and send unsubscribe packet.
378       uint8_t len = unsubscribePacket(buffer, subscriptions[i]->topic);
379
380       // sending unsubscribe failed
381       if (! sendPacket(buffer, len))
382         return false;
383
384       // if QoS for this subscription is 1 or 2, we need
385       // to wait for the unsuback to confirm unsubscription
386       if(subscriptions[i]->qos > 0 && MQTT_PROTOCOL_LEVEL > 3) {
387
388         // wait for UNSUBACK
389         len = readFullPacket(buffer, MAXBUFFERSIZE, CONNECT_TIMEOUT_MS);
390         DEBUG_PRINT(F("UNSUBACK:\t"));
391         DEBUG_PRINTBUFFER(buffer, len);
392
```

```
393         if ((len != 5) || (buffer[0] != (MQTT_CTRL_UNSUBACK << 4))) {
394           return false;  // failure to unsubscribe
395         }
396       }
397
398       subscriptions[i] = 0;
399       return true;
400     }
401
402   }
403
404   // subscription not found, so we are unsubscribed
405   return true;
406
407 }
408
409 void Adafruit_MQTT::processPackets(int16_t timeout) {
410   uint16_t len;
411
412   uint32_t elapsed = 0, endtime, starttime = millis();
413
414   while (elapsed < (uint32_t)timeout) {
415     Adafruit_MQTT_Subscribe *sub = readSubscription(timeout - elapsed);
416     if (sub) {
417       //Serial.println("**** sub packet received");
418       if (sub->callback_uint32t != NULL) {
419   // huh lets do the callback in integer mode
420   uint32_t data = 0;
421   data = atoi((char *)sub->lastread);
422   //Serial.print("*** calling int callback with : "); Serial.println(data);
423   sub->callback_uint32t(data);
424       }
425       else if (sub->callback_double != NULL) {
426   // huh lets do the callback in doublefloat mode
427   double data = 0;
428   data = atof((char *)sub->lastread);
429   //Serial.print("*** calling double callback with : "); Serial.println(data);
430   sub->callback_double(data);
431       }
432       else if (sub->callback_buffer != NULL) {
433   // huh lets do the callback in buffer mode
434   //Serial.print("*** calling buffer callback with : "); Serial.println((char *)sub->
      lastread);
435   sub->callback_buffer((char *)sub->lastread, sub->datalen);
436       }
437       else if (sub->callback_io != NULL) {
438         // huh lets do the callback in io mode
439         //Serial.print("*** calling io instance callback with : "); Serial.println((
              char *)sub->lastread);
440         ((sub->io_feed)->*(sub->callback_io))((char *)sub->lastread, sub->datalen);
441       }
442     }
443
444     // keep track over elapsed time
445     endtime = millis();
446     if (endtime < starttime) {
447       starttime = endtime; // looped around!")
448     }
449     elapsed += (endtime - starttime);
450   }
```

```
451 }
452
453 Adafruit_MQTT_Subscribe *Adafruit_MQTT::readSubscription(int16_t timeout) {
454   uint16_t i, topiclen, datalen;
455
456   // Check if data is available to read.
457   uint16_t len = readFullPacket(buffer, MAXBUFFERSIZE, timeout); // return one full
        packet
458   if (!len)
459     return NULL;  // No data available, just quit.
460   DEBUG_PRINT("Packet len: "); DEBUG_PRINTLN(len);
461   DEBUG_PRINTBUFFER(buffer, len);
462
463   // Parse out length of packet.
464   topiclen = buffer[3];
465   DEBUG_PRINT(F("Looking for subscription len ")); DEBUG_PRINTLN(topiclen);
466
467   // Find subscription associated with this packet.
468   for (i=0; i<MAXSUBSCRIPTIONS; i++) {
469     if (subscriptions[i]) {
470       // Skip this subscription if its name length isn't the same as the
471       // received topic name.
472       if (strlen(subscriptions[i]->topic) != topiclen)
473         continue;
474       // Stop if the subscription topic matches the received topic. Be careful
475       // to make comparison case insensitive.
476       if (strncasecmp((char*)buffer+4, subscriptions[i]->topic, topiclen) == 0) {
477         DEBUG_PRINT(F("Found sub #")); DEBUG_PRINTLN(i);
478         break;
479       }
480     }
481   }
482   if (i==MAXSUBSCRIPTIONS) return NULL; // matching sub not found ???
483
484   uint8_t packet_id_len = 0;
485   uint16_t packetid;
486   // Check if it is QoS 1, TODO: we dont support QoS 2
487   if ((buffer[0] & 0x6) == 0x2) {
488     packet_id_len = 2;
489     packetid = buffer[topiclen+4];
490     packetid <<= 8;
491     packetid |= buffer[topiclen+5];
492   }
493
494   // zero out the old data
495   memset(subscriptions[i]->lastread, 0, SUBSCRIPTIONDATALEN);
496
497   datalen = len - topiclen - packet_id_len - 4;
498   if (datalen > SUBSCRIPTIONDATALEN) {
499     datalen = SUBSCRIPTIONDATALEN-1; // cut it off
500   }
501   // extract out just the data, into the subscription object itself
502   memmove(subscriptions[i]->lastread, buffer+4+topiclen+packet_id_len, datalen);
503   subscriptions[i]->datalen = datalen;
504   DEBUG_PRINT(F("Data len: ")); DEBUG_PRINTLN(datalen);
505   DEBUG_PRINT(F("Data: ")); DEBUG_PRINTLN((char *)subscriptions[i]->lastread);
506
507   if ((MQTT_PROTOCOL_LEVEL > 3) &&(buffer[0] & 0x6) == 0x2) {
508     uint8_t ackpacket[4];
509
```

```
510     // Construct and send puback packet.
511     uint8_t len = pubackPacket(ackpacket, packetid);
512     if (!sendPacket(ackpacket, len))
513       DEBUG_PRINT(F("Failed"));
514   }
515
516   // return the valid matching subscription
517   return subscriptions[i];
518 }
519
520 void Adafruit_MQTT::flushIncoming(uint16_t timeout) {
521   // flush input!
522   DEBUG_PRINTLN(F("Flushing input buffer"));
523   while (readPacket(buffer, MAXBUFFERSIZE, timeout));
524 }
525
526 bool Adafruit_MQTT::ping(uint8_t num) {
527   //flushIncoming(100);
528
529   while (num--) {
530     // Construct and send ping packet.
531     uint8_t len = pingPacket(buffer);
532     if (!sendPacket(buffer, len))
533       continue;
534
535     // Process ping reply.
536     len = processPacketsUntil(buffer, MQTT_CTRL_PINGRESP, PING_TIMEOUT_MS);
537     if (buffer[0] == (MQTT_CTRL_PINGRESP << 4))
538       return true;
539   }
540
541   return false;
542 }
543
544 // Packet Generation Functions //////////////////////////////////////////////////
545
546 // The current MQTT spec is 3.1.1 and available here:
547 //   http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718028
548 // However this connect packet and code follows the MQTT 3.1 spec here (some
549 // small differences in the protocol):
550 //   http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html#connect
551 uint8_t Adafruit_MQTT::connectPacket(uint8_t *packet) {
552   uint8_t *p = packet;
553   uint16_t len;
554
555   // fixed header, connection messsage no flags
556   p[0] = (MQTT_CTRL_CONNECT << 4) | 0x0;
557   p+=2;
558   // fill in packet[1] last
559
560 #if MQTT_PROTOCOL_LEVEL == 3
561     p = stringprint(p, "MQIsdp");
562 #elif MQTT_PROTOCOL_LEVEL == 4
563     p = stringprint(p, "MQTT");
564 #else
565     #error "MQTT level not supported"
566 #endif
567
568   p[0] = MQTT_PROTOCOL_LEVEL;
569   p++;
```

```
570
571   // always clean the session
572   p[0] = MQTT_CONN_CLEANSESSION;
573
574   // set the will flags if needed
575   if (will_topic && pgm_read_byte(will_topic) != 0) {
576
577     p[0] |= MQTT_CONN_WILLFLAG;
578
579     if(will_qos == 1)
580       p[0] |= MQTT_CONN_WILLQOS_1;
581     else if(will_qos == 2)
582       p[0] |= MQTT_CONN_WILLQOS_2;
583
584     if(will_retain == 1)
585       p[0] |= MQTT_CONN_WILLRETAIN;
586
587   }
588
589   if (pgm_read_byte(username) != 0)
590     p[0] |= MQTT_CONN_USERNAMEFLAG;
591   if (pgm_read_byte(password) != 0)
592     p[0] |= MQTT_CONN_PASSWORDFLAG;
593   p++;
594
595   p[0] = MQTT_CONN_KEEPALIVE >> 8;
596   p++;
597   p[0] = MQTT_CONN_KEEPALIVE & 0xFF;
598   p++;
599
600   if(MQTT_PROTOCOL_LEVEL == 3) {
601     p = stringprint(p, clientid, 23);  // Limit client ID to first 23 characters.
602   } else {
603     if (pgm_read_byte(clientid) != 0) {
604       p = stringprint(p, clientid);
605     } else {
606       p[0] = 0x0;
607       p++;
608       p[0] = 0x0;
609       p++;
610       DEBUG_PRINTLN(F("SERVER GENERATING CLIENT ID"));
611     }
612   }
613
614   if (will_topic && pgm_read_byte(will_topic) != 0) {
615     p = stringprint(p, will_topic);
616     p = stringprint(p, will_payload);
617   }
618
619   if (pgm_read_byte(username) != 0) {
620     p = stringprint(p, username);
621   }
622   if (pgm_read_byte(password) != 0) {
623     p = stringprint(p, password);
624   }
625
626   len = p - packet;
627
628   packet[1] = len-2;  // don't include the 2 bytes of fixed header data
629   DEBUG_PRINTLN(F("MQTT connect packet:"));
```

```
630   DEBUG_PRINTBUFFER(buffer, len);
631   return len;
632 }
633
634
635 // as per http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#
      _Toc398718040
636 uint16_t Adafruit_MQTT::publishPacket(uint8_t *packet, const char *topic,
637                                       uint8_t *data, uint16_t bLen, uint8_t qos) {
638   uint8_t *p = packet;
639   uint16_t len=0;
640
641   // calc length of non-header data
642   len += 2;                // two bytes to set the topic size
643   len += strlen(topic); // topic length
644   if(qos > 0) {
645     len += 2; // qos packet id
646   }
647   len += bLen; // payload length
648
649   // Now you can start generating the packet!
650   p[0] = MQTT_CTRL_PUBLISH << 4 | qos << 1;
651   p++;
652
653   // fill in packet[1] last
654   do {
655     uint8_t encodedByte = len % 128;
656     len /= 128;
657     // if there are more data to encode, set the top bit of this byte
658     if ( len > 0 ) {
659       encodedByte |= 0x80;
660     }
661     p[0] = encodedByte;
662     p++;
663   } while ( len > 0 );
664
665   // topic comes before packet identifier
666   p = stringprint(p, topic);
667
668   // add packet identifier. used for checking PUBACK in QOS > 0
669   if(qos > 0) {
670     p[0] = (packet_id_counter >> 8) & 0xFF;
671     p[1] = packet_id_counter & 0xFF;
672     p+=2;
673
674     // increment the packet id
675     packet_id_counter++;
676   }
677
678   memmove(p, data, bLen);
679   p+= bLen;
680   len = p - packet;
681   DEBUG_PRINTLN(F("MQTT␣publish␣packet:"));
682   DEBUG_PRINTBUFFER(buffer, len);
683   return len;
684 }
685
686 uint8_t Adafruit_MQTT::subscribePacket(uint8_t *packet, const char *topic,
687                                        uint8_t qos) {
688   uint8_t *p = packet;
```

```
689   uint16_t len;
690
691   p[0] = MQTT_CTRL_SUBSCRIBE << 4 | MQTT_QOS_1 << 1;
692   // fill in packet[1] last
693   p+=2;
694
695   // packet identifier. used for checking SUBACK
696   p[0] = (packet_id_counter >> 8) & 0xFF;
697   p[1] = packet_id_counter & 0xFF;
698   p+=2;
699
700   // increment the packet id
701   packet_id_counter++;
702
703   p = stringprint(p, topic);
704
705   p[0] = qos;
706   p++;
707
708   len = p - packet;
709   packet[1] = len-2; // don't include the 2 bytes of fixed header data
710   DEBUG_PRINTLN(F("MQTT subscription packet:"));
711   DEBUG_PRINTBUFFER(buffer, len);
712   return len;
713 }
714
715
716
717 uint8_t Adafruit_MQTT::unsubscribePacket(uint8_t *packet, const char *topic) {
718
719   uint8_t *p = packet;
720   uint16_t len;
721
722   p[0] = MQTT_CTRL_UNSUBSCRIBE << 4 | 0x1;
723   // fill in packet[1] last
724   p+=2;
725
726   // packet identifier. used for checking UNSUBACK
727   p[0] = (packet_id_counter >> 8) & 0xFF;
728   p[1] = packet_id_counter & 0xFF;
729   p+=2;
730
731   // increment the packet id
732   packet_id_counter++;
733
734   p = stringprint(p, topic);
735
736   len = p - packet;
737   packet[1] = len-2; // don't include the 2 bytes of fixed header data
738   DEBUG_PRINTLN(F("MQTT unsubscription packet:"));
739   DEBUG_PRINTBUFFER(buffer, len);
740   return len;
741
742 }
743
744 uint8_t Adafruit_MQTT::pingPacket(uint8_t *packet) {
745   packet[0] = MQTT_CTRL_PINGREQ << 4;
746   packet[1] = 0;
747   DEBUG_PRINTLN(F("MQTT ping packet:"));
748   DEBUG_PRINTBUFFER(buffer, 2);
```

```
749   return 2;
750 }
751
752 uint8_t Adafruit_MQTT::pubackPacket(uint8_t *packet, uint16_t packetid) {
753   packet[0] = MQTT_CTRL_PUBACK << 4;
754   packet[1] = 2;
755   packet[2] = packetid >> 8;
756   packet[3] = packetid;
757   DEBUG_PRINTLN(F("MQTT puback packet:"));
758   DEBUG_PRINTBUFFER(buffer, 4);
759   return 4;
760 }
761
762 uint8_t Adafruit_MQTT::disconnectPacket(uint8_t *packet) {
763   packet[0] = MQTT_CTRL_DISCONNECT << 4;
764   packet[1] = 0;
765   DEBUG_PRINTLN(F("MQTT disconnect packet:"));
766   DEBUG_PRINTBUFFER(buffer, 2);
767   return 2;
768 }
769
770 // Adafruit_MQTT_Publish Definition ////////////////////////////////////////
771
772 Adafruit_MQTT_Publish::Adafruit_MQTT_Publish(Adafruit_MQTT *mqttserver,
773                                              const char *feed, uint8_t q) {
774   mqtt = mqttserver;
775   topic = feed;
776   qos = q;
777 }
778 bool Adafruit_MQTT_Publish::publish(int32_t i) {
779   char payload[12];
780   ltoa(i, payload, 10);
781   return mqtt->publish(topic, payload, qos);
782 }
783
784 bool Adafruit_MQTT_Publish::publish(uint32_t i) {
785   char payload[11];
786   ultoa(i, payload, 10);
787   return mqtt->publish(topic, payload, qos);
788 }
789
790 bool Adafruit_MQTT_Publish::publish(double f, uint8_t precision) {
791   char payload[41];  // Need to technically hold float max, 39 digits and minus sign.
792   dtostrf(f, 0, precision, payload);
793   return mqtt->publish(topic, payload, qos);
794 }
795
796 bool Adafruit_MQTT_Publish::publish(const char *payload) {
797   return mqtt->publish(topic, payload, qos);
798 }
799
800 //publish buffer of arbitrary length
801 bool Adafruit_MQTT_Publish::publish(uint8_t *payload, uint16_t bLen) {
802
803   return mqtt->publish(topic, payload, bLen, qos);
804 }
805
806
807 // Adafruit_MQTT_Subscribe Definition ////////////////////////////////////////
808
```

```
809 Adafruit_MQTT_Subscribe::Adafruit_MQTT_Subscribe(Adafruit_MQTT *mqttserver,
810                                                  const char *feed, uint8_t q) {
811   mqtt = mqttserver;
812   topic = feed;
813   qos = q;
814   datalen = 0;
815   callback_uint32t = 0;
816   callback_buffer = 0;
817   callback_double = 0;
818   callback_io = 0;
819   io_feed = 0;
820 }
821
822 void Adafruit_MQTT_Subscribe::setCallback(SubscribeCallbackUInt32Type cb) {
823   callback_uint32t = cb;
824 }
825
826 void Adafruit_MQTT_Subscribe::setCallback(SubscribeCallbackDoubleType cb) {
827   callback_double = cb;
828 }
829
830 void Adafruit_MQTT_Subscribe::setCallback(SubscribeCallbackBufferType cb) {
831   callback_buffer = cb;
832 }
833
834 void Adafruit_MQTT_Subscribe::setCallback(AdafruitIO_Feed *f, SubscribeCallbackIOType
      cb) {
835   callback_io = cb;
836   io_feed = f;
837 }
838
839 void Adafruit_MQTT_Subscribe::removeCallback(void) {
840   callback_uint32t = 0;
841   callback_buffer = 0;
842   callback_double = 0;
843   callback_io = 0;
844   io_feed = 0;
845 }
```

Apéndice F

Adafruit_MQTT.h (sin modificar)

Adafruit_MQTT_Antiguo.h

```
// The MIT License (MIT)
//
// Copyright (c) 2015 Adafruit Industries
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in all
// copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
// SOFTWARE.
#ifndef _ADAFRUIT_MQTT_H_
#define _ADAFRUIT_MQTT_H_

#include "Arduino.h"

#if defined(ARDUINO_SAMD_ZERO) || defined(ARDUINO_STM32_FEATHER)
#define strncpy_P(dest, src, len) strncpy((dest), (src), (len))
#define strncasecmp_P(f1, f2, len) strncasecmp((f1), (f2), (len))
#endif

#define ADAFRUIT_MQTT_VERSION_MAJOR 0
#define ADAFRUIT_MQTT_VERSION_MINOR 16
#define ADAFRUIT_MQTT_VERSION_PATCH 2

// Uncomment/comment to turn on/off debug output messages.
//#define MQTT_DEBUG
// Uncomment/comment to turn on/off error output messages.
#define MQTT_ERROR

// Set where debug messages will be printed.
#define DEBUG_PRINTER Serial
// If using something like Zero or Due, change the above to SerialUSB
```

```
44
45  // Define actual debug output functions when necessary.
46  #ifdef MQTT_DEBUG
47    #define DEBUG_PRINT(...) { DEBUG_PRINTER.print(__VA_ARGS__); }
48    #define DEBUG_PRINTLN(...) { DEBUG_PRINTER.println(__VA_ARGS__); }
49    #define DEBUG_PRINTBUFFER(buffer, len) { printBuffer(buffer, len); }
50  #else
51    #define DEBUG_PRINT(...) {}
52    #define DEBUG_PRINTLN(...) {}
53    #define DEBUG_PRINTBUFFER(buffer, len) {}
54  #endif
55
56  #ifdef MQTT_ERROR
57    #define ERROR_PRINT(...) { DEBUG_PRINTER.print(__VA_ARGS__); }
58    #define ERROR_PRINTLN(...) { DEBUG_PRINTER.println(__VA_ARGS__); }
59    #define ERROR_PRINTBUFFER(buffer, len) { printBuffer(buffer, len); }
60  #else
61    #define ERROR_PRINT(...) {}
62    #define ERROR_PRINTLN(...) {}
63    #define ERROR_PRINTBUFFER(buffer, len) {}
64  #endif
65
66  // Use 3 (MQTT 3.0) or 4 (MQTT 3.1.1)
67  #define MQTT_PROTOCOL_LEVEL 4
68
69  #define MQTT_CTRL_CONNECT      0x1
70  #define MQTT_CTRL_CONNECTACK   0x2
71  #define MQTT_CTRL_PUBLISH      0x3
72  #define MQTT_CTRL_PUBACK       0x4
73  #define MQTT_CTRL_PUBREC       0x5
74  #define MQTT_CTRL_PUBREL       0x6
75  #define MQTT_CTRL_PUBCOMP      0x7
76  #define MQTT_CTRL_SUBSCRIBE    0x8
77  #define MQTT_CTRL_SUBACK       0x9
78  #define MQTT_CTRL_UNSUBSCRIBE  0xA
79  #define MQTT_CTRL_UNSUBACK     0xB
80  #define MQTT_CTRL_PINGREQ      0xC
81  #define MQTT_CTRL_PINGRESP     0xD
82  #define MQTT_CTRL_DISCONNECT   0xE
83
84  #define MQTT_QOS_1 0x1
85  #define MQTT_QOS_0 0x0
86
87  #define CONNECT_TIMEOUT_MS 6000
88  #define PUBLISH_TIMEOUT_MS 500
89  #define PING_TIMEOUT_MS    500
90  #define SUBACK_TIMEOUT_MS  500
91
92  // Adjust as necessary, in seconds.  Default to 5 minutes.
93  #define MQTT_CONN_KEEPALIVE 300
94
95  // Largest full packet we're able to send.
96  // Need to be able to store at least ~90 chars for a connect packet with full
97  // 23 char client ID.
98  #define MAXBUFFERSIZE (150)
99
100 #define MQTT_CONN_USERNAMEFLAG    0x80
101 #define MQTT_CONN_PASSWORDFLAG    0x40
102 #define MQTT_CONN_WILLRETAIN      0x20
103 #define MQTT_CONN_WILLQOS_1       0x08
```

```
104 #define MQTT_CONN_WILLQOS_2        0x18
105 #define MQTT_CONN_WILLFLAG         0x04
106 #define MQTT_CONN_CLEANSESSION     0x02
107
108 // how many subscriptions we want to be able to track
109 #define MAXSUBSCRIPTIONS 5
110
111 // how much data we save in a subscription object
112 // eg max-subscription-payload-size
113 #if defined  (__AVR_ATmega32U4__) || defined(__AVR_ATmega328P__)
114   #define SUBSCRIPTIONDATALEN 20
115 #else
116   #define SUBSCRIPTIONDATALEN 100
117 #endif
118
119 class AdafruitIO_Feed;  // forward decl
120
121 //Function pointer that returns an int
122 typedef void (*SubscribeCallbackUInt32Type)(uint32_t);
123 // returns a double
124 typedef void (*SubscribeCallbackDoubleType)(double);
125 // returns a chunk of raw data
126 typedef void (*SubscribeCallbackBufferType)(char *str, uint16_t len);
127 // returns an io data wrapper instance
128 typedef void (AdafruitIO_Feed::*SubscribeCallbackIOType)(char *str, uint16_t len);
129
130 extern void printBuffer(uint8_t *buffer, uint16_t len);
131
132 class Adafruit_MQTT_Subscribe;  // forward decl
133
134 class Adafruit_MQTT {
135  public:
136   Adafruit_MQTT(const char *server,
137                 uint16_t port,
138                 const char *cid,
139                 const char *user,
140                 const char *pass);
141
142   Adafruit_MQTT(const char *server,
143                 uint16_t port,
144                 const char *user = "",
145                 const char *pass = "");
146   virtual ~Adafruit_MQTT() {}
147
148   // Connect to the MQTT server.  Returns 0 on success, otherwise an error code
149   // that indicates something went wrong:
150   //   -1 = Error connecting to server
151   //    1 = Wrong protocol
152   //    2 = ID rejected
153   //    3 = Server unavailable
154   //    4 = Bad username or password
155   //    5 = Not authenticated
156   //    6 = Failed to subscribe
157   // Use connectErrorString() to get a printable string version of the
158   // error.
159   int8_t connect();
160   int8_t connect(const char *user, const char *pass);
161
162   // Return a printable string version of the error code returned by
163   // connect(). This returns a __FlashStringHelper*, which points to a
```

```
164   // string stored in flash, but can be directly passed to e.g.
165   // Serial.println without any further processing.
166   const __FlashStringHelper* connectErrorString(int8_t code);
167
168   // Sends MQTT disconnect packet and calls disconnectServer()
169   bool disconnect();
170
171   // Return true if connected to the MQTT server, otherwise false.
172   virtual bool connected() = 0;  // Subclasses need to fill this in!
173
174   // Set MQTT last will topic, payload, QOS, and retain. This needs
175   // to be called before connect() because it is sent as part of the
176   // connect control packet.
177   bool will(const char *topic, const char *payload, uint8_t qos = 0, uint8_t retain =
          0);
178
179   // Publish a message to a topic using the specified QoS level.  Returns true
180   // if the message was published, false otherwise.
181   bool publish(const char *topic, const char *payload, uint8_t qos = 0);
182   bool publish(const char *topic, uint8_t *payload, uint16_t bLen, uint8_t qos = 0);
183
184   // Add a subscription to receive messages for a topic.  Returns true if the
185   // subscription could be added or was already present, false otherwise.
186   // Must be called before connect(), subscribing after the connection
187   // is made is not currently supported.
188   bool subscribe(Adafruit_MQTT_Subscribe *sub);
189
190   // Unsubscribe from a previously subscribed MQTT topic.
191   bool unsubscribe(Adafruit_MQTT_Subscribe *sub);
192
193   // Check if any subscriptions have new messages.  Will return a reference to
194   // an Adafruit_MQTT_Subscribe object which has a new message.  Should be called
195   // in the sketch's loop function to ensure new messages are recevied.  Note
196   // that subscribe should be called first for each topic that receives messages!
197   Adafruit_MQTT_Subscribe *readSubscription(int16_t timeout=0);
198
199   void processPackets(int16_t timeout);
200
201   // Ping the server to ensure the connection is still alive.
202   bool ping(uint8_t n = 1);
203
204 protected:
205   // Interface that subclasses need to implement:
206
207   // Connect to the server and return true if successful, false otherwise.
208   virtual bool connectServer() = 0;
209
210   // Disconnect from the MQTT server.  Returns true if disconnected, false otherwise.
211   virtual bool disconnectServer() = 0;  // Subclasses need to fill this in!
212
213   // Send data to the server specified by the buffer and length of data.
214   virtual bool sendPacket(uint8_t *buffer, uint16_t len) = 0;
215
216   // Read MQTT packet from the server.  Will read up to maxlen bytes and store
217   // the data in the provided buffer.  Waits up to the specified timeout (in
218   // milliseconds) for data to be available.
219   virtual uint16_t readPacket(uint8_t *buffer, uint16_t maxlen, int16_t timeout) = 0;
220
221   // Read a full packet, keeping note of the correct length
222   uint16_t readFullPacket(uint8_t *buffer, uint16_t maxsize, uint16_t timeout);
```

```
223   // Properly process packets until you get to one you want
224   uint16_t processPacketsUntil(uint8_t *buffer, uint8_t waitforpackettype, uint16_t
          timeout);
225
226   // Shared state that subclasses can use:
227   const char *servername;
228   int16_t portnum;
229   const char *clientid;
230   const char *username;
231   const char *password;
232   const char *will_topic;
233   const char *will_payload;
234   uint8_t will_qos;
235   uint8_t will_retain;
236   uint8_t buffer[MAXBUFFERSIZE];  // one buffer, used for all incoming/outgoing
237   uint16_t packet_id_counter;
238
239  private:
240   Adafruit_MQTT_Subscribe *subscriptions[MAXSUBSCRIPTIONS];
241
242   void     flushIncoming(uint16_t timeout);
243
244   // Functions to generate MQTT packets.
245   uint8_t connectPacket(uint8_t *packet);
246   uint8_t disconnectPacket(uint8_t *packet);
247   uint16_t publishPacket(uint8_t *packet, const char *topic, uint8_t *payload, uint16_t
          bLen, uint8_t qos);
248   uint8_t subscribePacket(uint8_t *packet, const char *topic, uint8_t qos);
249   uint8_t unsubscribePacket(uint8_t *packet, const char *topic);
250   uint8_t pingPacket(uint8_t *packet);
251   uint8_t pubackPacket(uint8_t *packet, uint16_t packetid);
252 };
253
254
255 class Adafruit_MQTT_Publish {
256  public:
257   Adafruit_MQTT_Publish(Adafruit_MQTT *mqttserver, const char *feed, uint8_t qos = 0);
258
259   bool publish(const char *s);
260   bool publish(double f, uint8_t precision=2);  // Precision controls the minimum
          number of digits after decimal.
261                                                 // This might be ignored and a higher
                                                        precision value sent.
262   bool publish(int32_t i);
263   bool publish(uint32_t i);
264   bool publish(uint8_t *b, uint16_t bLen);
265
266
267 private:
268   Adafruit_MQTT *mqtt;
269   const char *topic;
270   uint8_t qos;
271 };
272
273 class Adafruit_MQTT_Subscribe {
274  public:
275   Adafruit_MQTT_Subscribe(Adafruit_MQTT *mqttserver, const char *feedname, uint8_t q=0)
          ;
276
277   void setCallback(SubscribeCallbackUInt32Type callb);
```

```
278   void setCallback(SubscribeCallbackDoubleType callb);
279   void setCallback(SubscribeCallbackBufferType callb);
280   void setCallback(AdafruitIO_Feed *io, SubscribeCallbackIOType callb);
281   void removeCallback(void);
282
283   const char *topic;
284   uint8_t qos;
285
286   uint8_t lastread[SUBSCRIPTIONDATALEN];
287   // Number valid bytes in lastread. Limited to SUBSCRIPTIONDATALEN-1 to
288   // ensure nul terminating lastread.
289   uint16_t datalen;
290
291   SubscribeCallbackUInt32Type callback_uint32t;
292   SubscribeCallbackDoubleType callback_double;
293   SubscribeCallbackBufferType callback_buffer;
294   SubscribeCallbackIOType     callback_io;
295
296   AdafruitIO_Feed *io_feed;
297
298  private:
299   Adafruit_MQTT *mqtt;
300 };
301
302
303 #endif
```

Apéndice G

Adafruit_MQTT.cpp (modificado)

Adafruit_MQTT.cpp

```
// The MIT License (MIT)
//
// Copyright (c) 2015 Adafruit Industries
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in all
// copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
// SOFTWARE.
#include "Adafruit_MQTT.h"

#if defined(ARDUINO_SAMD_ZERO) || defined(ARDUINO_SAMD_MKR1000)
static char *dtostrf (double val, signed char width, unsigned char prec, char *sout) {
  char fmt[20];
  sprintf(fmt, "%%%d.%df", width, prec);
  sprintf(sout, fmt, val);
  return sout;
}
#endif

#if defined(ESP8266)
int strncasecmp(const char * str1, const char * str2, int len) {
    int d = 0;
    while(len--) {
        int c1 = tolower(*str1++);
        int c2 = tolower(*str2++);
        if(((d = c1 - c2) != 0) || (c2 == '\0')) {
    return d;
        }
    }
    return 0;
```

```
}
#endif


void printBuffer(uint8_t *buffer, uint16_t len) {
  DEBUG_PRINTER.print('\t');
  for (uint16_t i=0; i<len; i++) {
    if (isprint(buffer[i]))
      DEBUG_PRINTER.write(buffer[i]);
    else
      DEBUG_PRINTER.print(" ");
    DEBUG_PRINTER.print(F(" [0x"));
    if (buffer[i] < 0x10)
      DEBUG_PRINTER.print("0");
    DEBUG_PRINTER.print(buffer[i],HEX);
    DEBUG_PRINTER.print("], ");
    if (i % 8 == 7) {
      DEBUG_PRINTER.print("\n\t");
    }
  }
  DEBUG_PRINTER.println();
}

/* Not used now, but might be useful in the future
static uint8_t *stringprint(uint8_t *p, char *s) {
  uint16_t len = strlen(s);
  p[0] = len >> 8; p++;
  p[0] = len & 0xFF; p++;
  memmove(p, s, len);
  return p+len;
}
*/

static uint8_t *stringprint(uint8_t *p, const char *s, uint16_t maxlen=0) {
  // If maxlen is specified (has a non-zero value) then use it as the maximum
  // length of the source string to write to the buffer.  Otherwise write
  // the entire source string.
  uint16_t len = strlen(s);
  if (maxlen > 0 && len > maxlen) {
    len = maxlen;
  }
  /*
  for (uint8_t i=0; i<len; i++) {
    Serial.write(pgm_read_byte(s+i));
  }
  */
  p[0] = len >> 8; p++;
  p[0] = len & 0xFF; p++;
  strncpy((char *)p, s, len);
  return p+len;
}


// Adafruit_MQTT Definition ////////////////////////////////////////////////

Adafruit_MQTT::Adafruit_MQTT(const char *server,
                             uint16_t port,
                             const char *cid,
                             const char *user,
                             const char *pass) {
```

```
104   servername = server;
105   portnum = port;
106   clientid = cid;
107   username = user;
108   password = pass;
109
110   // reset subscriptions
111   for (uint8_t i=0; i<MAXSUBSCRIPTIONS; i++) {
112     subscriptions[i] = 0;
113   }
114
115   will_topic = 0;
116   will_payload = 0;
117   will_qos = 0;
118   will_retain = 0;
119
120   packet_id_counter = 0;
121
122 }
123
124
125 Adafruit_MQTT::Adafruit_MQTT(const char *server,
126                              uint16_t port,
127                              const char *user,
128                              const char *pass) {
129   servername = server;
130   portnum = port;
131   clientid = "";
132   username = user;
133   password = pass;
134
135   // reset subscriptions
136   for (uint8_t i=0; i<MAXSUBSCRIPTIONS; i++) {
137     subscriptions[i] = 0;
138   }
139
140   will_topic = 0;
141   will_payload = 0;
142   will_qos = 0;
143   will_retain = 0;
144
145   packet_id_counter = 0;
146
147 }
148
149 int8_t Adafruit_MQTT::connect() {
150   // Connect to the server.
151   if (!connectServer())
152     return -1;
153
154   // Construct and send connect packet.
155   uint8_t len = connectPacket(buffer);
156   if (!sendPacket(buffer, len))
157     return -1;
158
159   // Read connect response packet and verify it
160   len = readFullPacket(buffer, MAXBUFFERSIZE, CONNECT_TIMEOUT_MS);
161   if (len != 4)
162     return -1;
163   if ((buffer[0] != (MQTT_CTRL_CONNECTACK << 4)) || (buffer[1] != 2))
```

```
164     return -1;
165   if (buffer[3] != 0)
166     return buffer[3];
167
168   // Setup subscriptions once connected.
169   for (uint8_t i=0; i<MAXSUBSCRIPTIONS; i++) {
170     // Ignore subscriptions that aren't defined.
171     if (subscriptions[i] == 0) continue;
172
173     boolean success = false;
174     for (uint8_t retry=0; (retry<3) && !success; retry++) { // retry until we get a
          suback
175       // Construct and send subscription packet.
176       uint8_t len = subscribePacket(buffer, subscriptions[i]->topic, subscriptions[i]->
            qos);
177       if (!sendPacket(buffer, len))
178   return -1;
179
180       if(MQTT_PROTOCOL_LEVEL < 3) // older versions didn't suback
181   break;
182
183       // Check for SUBACK if using MQTT 3.1.1 or higher
184       // TODO: The Server is permitted to start sending PUBLISH packets matching the
185       // Subscription before the Server sends the SUBACK Packet. (will really need to
            use callbacks - ada)
186
187       //Serial.println("\t**looking for suback");
188       if (processPacketsUntil(buffer, MQTT_CTRL_SUBACK, SUBACK_TIMEOUT_MS)) {
189   success = true;
190   break;
191       }
192       //Serial.println("\t**failed, retrying!");
193     }
194     if (! success) return -2; // failed to sub for some reason
195   }
196
197   return 0;
198 }
199
200 int8_t Adafruit_MQTT::connect(const char *user, const char *pass)
201 {
202   username = user;
203   password = pass;
204   return connect();
205 }
206
207 uint16_t Adafruit_MQTT::processPacketsUntil(uint8_t *buffer, uint8_t waitforpackettype,
       uint16_t timeout) {
208   uint16_t len;
209   while (len = readFullPacket(buffer, MAXBUFFERSIZE, timeout)) {
210
211     //DEBUG_PRINT("Packet read size: "); DEBUG_PRINTLN(len);
212     // TODO: add subscription reading & call back processing here
213
214     if ((buffer[0] >> 4) == waitforpackettype) {
215       //DEBUG_PRINTLN(F("Found right packet"));
216       return len;
217     } else {
218       ERROR_PRINTLN(F("Dropped␣a␣packet"));
219     }
```

```
220   }
221   return 0;
222 }
223
224 uint16_t Adafruit_MQTT::readFullPacket(uint8_t *buffer, uint16_t maxsize, uint16_t
      timeout) {
225   // will read a packet and Do The Right Thing with length
226   uint8_t *pbuff = buffer;
227
228   uint8_t rlen;
229
230   // read the packet type:
231   rlen = readPacket(pbuff, 1, timeout);
232   if (rlen != 1) return 0;
233
234   DEBUG_PRINT(F("Packet Type:\t")); DEBUG_PRINTBUFFER(pbuff, rlen);
235   pbuff++;
236
237   uint32_t value = 0;
238   uint32_t multiplier = 1;
239   uint8_t encodedByte;
240
241   do {
242     rlen = readPacket(pbuff, 1, timeout);
243     if (rlen != 1) return 0;
244     encodedByte = pbuff[0]; // save the last read val
245     pbuff++; // get ready for reading the next byte
246     uint32_t intermediate = encodedByte & 0x7F;
247     intermediate *= multiplier;
248     value += intermediate;
249     multiplier *= 128;
250     if (multiplier > (128UL*128UL*128UL)) {
251       DEBUG_PRINT(F("Malformed packet len\n"));
252       return 0;
253     }
254   } while (encodedByte & 0x80);
255
256   DEBUG_PRINT(F("Packet Length:\t")); DEBUG_PRINTLN(value);
257
258   if (value > (maxsize - (pbuff-buffer) - 1)) {
259       DEBUG_PRINTLN(F("Packet too big for buffer"));
260       rlen = readPacket(pbuff, (maxsize - (pbuff-buffer) - 1), timeout);
261   } else {
262     rlen = readPacket(pbuff, value, timeout);
263   }
264   //DEBUG_PRINT(F("Remaining packet:\t")); DEBUG_PRINTBUFFER(pbuff, rlen);
265
266   return ((pbuff - buffer)+rlen);
267 }
268
269 const __FlashStringHelper* Adafruit_MQTT::connectErrorString(int8_t code) {
270    switch (code) {
271       case 1: return F("The Server does not support the level of the MQTT protocol
          requested");
272       case 2: return F("The Client identifier is correct UTF-8 but not allowed by the
          Server");
273       case 3: return F("The MQTT service is unavailable");
274       case 4: return F("The data in the user name or password is malformed");
275       case 5: return F("Not authorized to connect");
276       case 6: return F("Exceeded reconnect rate limit. Please try again later.");
```

```
277       case 7: return F("You have been banned from connecting. Please contact the MQTT
              server administrator for more details.");
278       case -1: return F("Connection failed");
279       case -2: return F("Failed to subscribe");
280       default: return F("Unknown error");
281     }
282 }
283
284 bool Adafruit_MQTT::disconnect() {
285
286   // Construct and send disconnect packet.
287   uint8_t len = disconnectPacket(buffer);
288   if (! sendPacket(buffer, len))
289     DEBUG_PRINTLN(F("Unable to send disconnect packet"));
290
291   return disconnectServer();
292
293 }
294
295
296 bool Adafruit_MQTT::publish(const char *topic, const char *data, uint8_t qos,uint8_t
      retain) {
297     return publish(topic, (uint8_t*)(data), strlen(data), qos);
298 }
299
300 bool Adafruit_MQTT::publish(const char *topic, uint8_t *data, uint16_t bLen, uint8_t
      qos,uint8_t retain) {
301   // Construct and send publish packet.
302   uint16_t len = publishPacket(buffer, topic, data, bLen, qos,retain);
303   if (!sendPacket(buffer, len))
304     return false;
305
306   // If QOS level is high enough verify the response packet.
307   if (qos > 0) {
308     len = readFullPacket(buffer, MAXBUFFERSIZE, PUBLISH_TIMEOUT_MS);
309     DEBUG_PRINT(F("Publish QOS1+ reply:\t"));
310     DEBUG_PRINTBUFFER(buffer, len);
311     if (len != 4)
312       return false;
313     if ((buffer[0] >> 4) != MQTT_CTRL_PUBACK)
314       return false;
315     uint16_t packnum = buffer[2];
316     packnum <<= 8;
317     packnum |= buffer[3];
318
319     // we increment the packet_id_counter right after publishing so inc here too to
          match
320     packnum++;
321     if (packnum != packet_id_counter)
322       return false;
323   }
324
325   return true;
326 }
327
328 bool Adafruit_MQTT::will(const char *topic, const char *payload, uint8_t qos, uint8_t
      retain) {
329
330   if (connected()) {
331     DEBUG_PRINT(F("Will defined after connect"));
```

```
332     return false;
333   }
334
335   will_topic = topic;
336   will_payload = payload;
337   will_qos = qos;
338   will_retain = retain;
339
340   return true;
341
342 }
343
344 bool Adafruit_MQTT::subscribe(Adafruit_MQTT_Subscribe *sub) {
345   uint8_t i;
346   // see if we are already subscribed
347   for (i=0; i<MAXSUBSCRIPTIONS; i++) {
348     if (subscriptions[i] == sub) {
349       DEBUG_PRINTLN(F("Already subscribed"));
350       return true;
351     }
352   }
353   if (i==MAXSUBSCRIPTIONS) { // add to subscriptionlist
354     for (i=0; i<MAXSUBSCRIPTIONS; i++) {
355       if (subscriptions[i] == 0) {
356         DEBUG_PRINT(F("Added sub ")); DEBUG_PRINTLN(i);
357         subscriptions[i] = sub;
358         return true;
359       }
360     }
361   }
362
363   DEBUG_PRINTLN(F("no more subscription space :("));
364   return false;
365 }
366
367 bool Adafruit_MQTT::unsubscribe(Adafruit_MQTT_Subscribe *sub) {
368   uint8_t i;
369
370   // see if we are already subscribed
371   for (i=0; i<MAXSUBSCRIPTIONS; i++) {
372
373     if (subscriptions[i] == sub) {
374
375       DEBUG_PRINTLN(F("Found matching subscription and attempting to unsubscribe."));
376
377       // Construct and send unsubscribe packet.
378       uint8_t len = unsubscribePacket(buffer, subscriptions[i]->topic);
379
380       // sending unsubscribe failed
381       if (! sendPacket(buffer, len))
382         return false;
383
384       // if QoS for this subscription is 1 or 2, we need
385       // to wait for the unsuback to confirm unsubscription
386       if(subscriptions[i]->qos > 0 && MQTT_PROTOCOL_LEVEL > 3) {
387
388         // wait for UNSUBACK
389         len = readFullPacket(buffer, MAXBUFFERSIZE, CONNECT_TIMEOUT_MS);
390         DEBUG_PRINT(F("UNSUBACK:\t"));
391         DEBUG_PRINTBUFFER(buffer, len);
```

```
392
393         if ((len != 5) || (buffer[0] != (MQTT_CTRL_UNSUBACK << 4))) {
394           return false;  // failure to unsubscribe
395         }
396       }
397
398       subscriptions[i] = 0;
399       return true;
400     }
401
402   }
403
404   // subscription not found, so we are unsubscribed
405   return true;
406
407 }
408
409 void Adafruit_MQTT::processPackets(int16_t timeout) {
410   uint16_t len;
411
412   uint32_t elapsed = 0, endtime, starttime = millis();
413
414   while (elapsed < (uint32_t)timeout) {
415     Adafruit_MQTT_Subscribe *sub = readSubscription(timeout - elapsed);
416     if (sub) {
417       //Serial.println("**** sub packet received");
418       if (sub->callback_uint32t != NULL) {
419   // huh lets do the callback in integer mode
420   uint32_t data = 0;
421   data = atoi((char *)sub->lastread);
422   //Serial.print("*** calling int callback with : "); Serial.println(data);
423   sub->callback_uint32t(data);
424       }
425       else if (sub->callback_double != NULL) {
426   // huh lets do the callback in doublefloat mode
427   double data = 0;
428   data = atof((char *)sub->lastread);
429   //Serial.print("*** calling double callback with : "); Serial.println(data);
430   sub->callback_double(data);
431       }
432       else if (sub->callback_buffer != NULL) {
433   // huh lets do the callback in buffer mode
434   //Serial.print("*** calling buffer callback with : "); Serial.println((char *)sub->
      lastread);
435   sub->callback_buffer((char *)sub->lastread, sub->datalen);
436       }
437       else if (sub->callback_io != NULL) {
438         // huh lets do the callback in io mode
439         //Serial.print("*** calling io instance callback with : "); Serial.println((
              char *)sub->lastread);
440         ((sub->io_feed)->*(sub->callback_io))((char *)sub->lastread, sub->datalen);
441       }
442     }
443
444     // keep track over elapsed time
445     endtime = millis();
446     if (endtime < starttime) {
447       starttime = endtime; // looped around!")
448     }
449     elapsed += (endtime - starttime);
```

```
450   }
451 }
452
453 Adafruit_MQTT_Subscribe *Adafruit_MQTT::readSubscription(int16_t timeout) {
454   uint16_t i, topiclen, datalen;
455
456   // Check if data is available to read.
457   uint16_t len = readFullPacket(buffer, MAXBUFFERSIZE, timeout); // return one full
        packet
458   if (!len)
459     return NULL;  // No data available, just quit.
460   DEBUG_PRINT("Packet len:"); DEBUG_PRINTLN(len);
461   DEBUG_PRINTBUFFER(buffer, len);
462
463   // Parse out length of packet.
464   topiclen = buffer[3];
465   DEBUG_PRINT(F("Looking for subscription len:")); DEBUG_PRINTLN(topiclen);
466
467   // Find subscription associated with this packet.
468   for (i=0; i<MAXSUBSCRIPTIONS; i++) {
469     if (subscriptions[i]) {
470       // Skip this subscription if its name length isn't the same as the
471       // received topic name.
472       if (strlen(subscriptions[i]->topic) != topiclen)
473         continue;
474       // Stop if the subscription topic matches the received topic. Be careful
475       // to make comparison case insensitive.
476       if (strncasecmp((char*)buffer+4, subscriptions[i]->topic, topiclen) == 0) {
477         DEBUG_PRINT(F("Found sub #")); DEBUG_PRINTLN(i);
478         break;
479       }
480     }
481   }
482   if (i==MAXSUBSCRIPTIONS) return NULL; // matching sub not found ???
483
484   uint8_t packet_id_len = 0;
485   uint16_t packetid;
486   // Check if it is QoS 1, TODO: we dont support QoS 2
487   if ((buffer[0] & 0x6) == 0x2) {
488     packet_id_len = 2;
489     packetid = buffer[topiclen+4];
490     packetid <<= 8;
491     packetid |= buffer[topiclen+5];
492   }
493
494   // zero out the old data
495   memset(subscriptions[i]->lastread, 0, SUBSCRIPTIONDATALEN);
496
497   datalen = len - topiclen - packet_id_len - 4;
498   if (datalen > SUBSCRIPTIONDATALEN) {
499     datalen = SUBSCRIPTIONDATALEN-1; // cut it off
500   }
501   // extract out just the data, into the subscription object itself
502   memmove(subscriptions[i]->lastread, buffer+4+topiclen+packet_id_len, datalen);
503   subscriptions[i]->datalen = datalen;
504   DEBUG_PRINT(F("Data len:")); DEBUG_PRINTLN(datalen);
505   DEBUG_PRINT(F("Data:")); DEBUG_PRINTLN((char *)subscriptions[i]->lastread);
506
507   if ((MQTT_PROTOCOL_LEVEL > 3) &&(buffer[0] & 0x6) == 0x2) {
508     uint8_t ackpacket[4];
```

```
509
510     // Construct and send puback packet.
511     uint8_t len = pubackPacket(ackpacket, packetid);
512     if (!sendPacket(ackpacket, len))
513       DEBUG_PRINT(F("Failed"));
514   }
515
516   // return the valid matching subscription
517   return subscriptions[i];
518 }
519
520 void Adafruit_MQTT::flushIncoming(uint16_t timeout) {
521   // flush input!
522   DEBUG_PRINTLN(F("Flushing input buffer"));
523   while (readPacket(buffer, MAXBUFFERSIZE, timeout));
524 }
525
526 bool Adafruit_MQTT::ping(uint8_t num) {
527   //flushIncoming(100);
528
529   while (num--) {
530     // Construct and send ping packet.
531     uint8_t len = pingPacket(buffer);
532     if (!sendPacket(buffer, len))
533       continue;
534
535     // Process ping reply.
536     len = processPacketsUntil(buffer, MQTT_CTRL_PINGRESP, PING_TIMEOUT_MS);
537     if (buffer[0] == (MQTT_CTRL_PINGRESP << 4))
538       return true;
539   }
540
541   return false;
542 }
543
544 // Packet Generation Functions //////////////////////////////////////////////////
545
546 // The current MQTT spec is 3.1.1 and available here:
547 //   http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718028
548 // However this connect packet and code follows the MQTT 3.1 spec here (some
549 // small differences in the protocol):
550 //   http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html#connect
551 uint8_t Adafruit_MQTT::connectPacket(uint8_t *packet) {
552   uint8_t *p = packet;
553   uint16_t len;
554
555   // fixed header, connection messsage no flags
556   p[0] = (MQTT_CTRL_CONNECT << 4) | 0x0;
557   p+=2;
558   // fill in packet[1] last
559
560 #if MQTT_PROTOCOL_LEVEL == 3
561     p = stringprint(p, "MQIsdp");
562 #elif MQTT_PROTOCOL_LEVEL == 4
563     p = stringprint(p, "MQTT");
564 #else
565     #error "MQTT level not supported"
566 #endif
567
568   p[0] = MQTT_PROTOCOL_LEVEL;
```

```
569   p++;
570
571   // always clean the session
572   p[0] = MQTT_CONN_CLEANSESSION;
573
574   // set the will flags if needed
575   if (will_topic && pgm_read_byte(will_topic) != 0) {
576
577     p[0] |= MQTT_CONN_WILLFLAG;
578
579     if(will_qos == 1)
580       p[0] |= MQTT_CONN_WILLQOS_1;
581     else if(will_qos == 2)
582       p[0] |= MQTT_CONN_WILLQOS_2;
583
584     if(will_retain == 1)
585       p[0] |= MQTT_CONN_WILLRETAIN;
586
587   }
588
589   if (pgm_read_byte(username) != 0)
590     p[0] |= MQTT_CONN_USERNAMEFLAG;
591   if (pgm_read_byte(password) != 0)
592     p[0] |= MQTT_CONN_PASSWORDFLAG;
593   p++;
594
595   p[0] = MQTT_CONN_KEEPALIVE >> 8;
596   p++;
597   p[0] = MQTT_CONN_KEEPALIVE & 0xFF;
598   p++;
599
600   if(MQTT_PROTOCOL_LEVEL == 3) {
601     p = stringprint(p, clientid, 23);  // Limit client ID to first 23 characters.
602   } else {
603     if (pgm_read_byte(clientid) != 0) {
604       p = stringprint(p, clientid);
605     } else {
606       p[0] = 0x0;
607       p++;
608       p[0] = 0x0;
609       p++;
610       DEBUG_PRINTLN(F("SERVER GENERATING CLIENT ID"));
611     }
612   }
613
614   if (will_topic && pgm_read_byte(will_topic) != 0) {
615     p = stringprint(p, will_topic);
616     p = stringprint(p, will_payload);
617   }
618
619   if (pgm_read_byte(username) != 0) {
620     p = stringprint(p, username);
621   }
622   if (pgm_read_byte(password) != 0) {
623     p = stringprint(p, password);
624   }
625
626   len = p - packet;
627
628   packet[1] = len-2;  // don't include the 2 bytes of fixed header data
```

```
629   DEBUG_PRINTLN(F("MQTT connect packet:"));
630   DEBUG_PRINTBUFFER(buffer, len);
631   return len;
632 }
633
634
635 // as per http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#
      _Toc398718040
636 uint16_t Adafruit_MQTT::publishPacket(uint8_t *packet, const char *topic,
637                                       uint8_t *data, uint16_t bLen, uint8_t qos,uint8_t
                                          retain) {
638   uint8_t *p = packet;
639   uint16_t len=0;
640
641   // calc length of non-header data
642   len += 2;                // two bytes to set the topic size
643   len += strlen(topic); // topic length
644   if(qos > 0) {
645     len += 2; // qos packet id
646   }
647   len += bLen; // payload length
648
649   // Now you can start generating the packet!
650   p[0] = MQTT_CTRL_PUBLISH << 4 | qos << 1| retain<<0;
651   p++;
652
653   // fill in packet[1] last
654   do {
655     uint8_t encodedByte = len % 128;
656     len /= 128;
657     // if there are more data to encode, set the top bit of this byte
658     if ( len > 0 ) {
659       encodedByte |= 0x80;
660     }
661     p[0] = encodedByte;
662     p++;
663   } while ( len > 0 );
664
665   // topic comes before packet identifier
666   p = stringprint(p, topic);
667
668   // add packet identifier. used for checking PUBACK in QOS > 0
669   if(qos > 0) {
670     p[0] = (packet_id_counter >> 8) & 0xFF;
671     p[1] = packet_id_counter & 0xFF;
672     p+=2;
673
674     // increment the packet id
675     packet_id_counter++;
676   }
677
678   memmove(p, data, bLen);
679   p+= bLen;
680   len = p - packet;
681   DEBUG_PRINTLN(F("MQTT publish packet:"));
682   DEBUG_PRINTBUFFER(buffer, len);
683   return len;
684 }
685
686 uint8_t Adafruit_MQTT::subscribePacket(uint8_t *packet, const char *topic,
```

```
                                        uint8_t qos) {
  uint8_t *p = packet;
  uint16_t len;

  p[0] = MQTT_CTRL_SUBSCRIBE << 4 | MQTT_QOS_1 << 1;
  // fill in packet[1] last
  p+=2;

  // packet identifier. used for checking SUBACK
  p[0] = (packet_id_counter >> 8) & 0xFF;
  p[1] = packet_id_counter & 0xFF;
  p+=2;

  // increment the packet id
  packet_id_counter++;

  p = stringprint(p, topic);

  p[0] = qos;
  p++;

  len = p - packet;
  packet[1] = len-2; // don't include the 2 bytes of fixed header data
  DEBUG_PRINTLN(F("MQTT subscription packet:"));
  DEBUG_PRINTBUFFER(buffer, len);
  return len;
}



uint8_t Adafruit_MQTT::unsubscribePacket(uint8_t *packet, const char *topic) {

  uint8_t *p = packet;
  uint16_t len;

  p[0] = MQTT_CTRL_UNSUBSCRIBE << 4 | 0x1;
  // fill in packet[1] last
  p+=2;

  // packet identifier. used for checking UNSUBACK
  p[0] = (packet_id_counter >> 8) & 0xFF;
  p[1] = packet_id_counter & 0xFF;
  p+=2;

  // increment the packet id
  packet_id_counter++;

  p = stringprint(p, topic);

  len = p - packet;
  packet[1] = len-2; // don't include the 2 bytes of fixed header data
  DEBUG_PRINTLN(F("MQTT unsubscription packet:"));
  DEBUG_PRINTBUFFER(buffer, len);
  return len;

}

uint8_t Adafruit_MQTT::pingPacket(uint8_t *packet) {
  packet[0] = MQTT_CTRL_PINGREQ << 4;
  packet[1] = 0;
```

```
747   DEBUG_PRINTLN(F("MQTT ping packet:"));
748   DEBUG_PRINTBUFFER(buffer, 2);
749   return 2;
750 }
751
752 uint8_t Adafruit_MQTT::pubackPacket(uint8_t *packet, uint16_t packetid) {
753   packet[0] = MQTT_CTRL_PUBACK << 4;
754   packet[1] = 2;
755   packet[2] = packetid >> 8;
756   packet[3] = packetid;
757   DEBUG_PRINTLN(F("MQTT puback packet:"));
758   DEBUG_PRINTBUFFER(buffer, 4);
759   return 4;
760 }
761
762 uint8_t Adafruit_MQTT::disconnectPacket(uint8_t *packet) {
763   packet[0] = MQTT_CTRL_DISCONNECT << 4;
764   packet[1] = 0;
765   DEBUG_PRINTLN(F("MQTT disconnect packet:"));
766   DEBUG_PRINTBUFFER(buffer, 2);
767   return 2;
768 }
769
770 // Adafruit_MQTT_Publish Definition ////////////////////////////////////////////
771
772 Adafruit_MQTT_Publish::Adafruit_MQTT_Publish(Adafruit_MQTT *mqttserver,
773                                              const char *feed, uint8_t q,uint8_t r) {
774   mqtt = mqttserver;
775   topic = feed;
776   qos = q;
777   retain=r;
778 }
779 bool Adafruit_MQTT_Publish::publish(int32_t i) {
780   char payload[12];
781   ltoa(i, payload, 10);
782   return mqtt->publish(topic, payload, qos,retain);
783 }
784
785 bool Adafruit_MQTT_Publish::publish(uint32_t i) {
786   char payload[11];
787   ultoa(i, payload, 10);
788   return mqtt->publish(topic, payload, qos,retain);
789 }
790
791 bool Adafruit_MQTT_Publish::publish(double f, uint8_t precision) {
792   char payload[41];  // Need to technically hold float max, 39 digits and minus sign.
793   dtostrf(f, 0, precision, payload);
794   return mqtt->publish(topic, payload, qos,retain);
795 }
796
797 bool Adafruit_MQTT_Publish::publish(const char *payload) {
798   return mqtt->publish(topic, payload, qos,retain);
799 }
800
801 //publish buffer of arbitrary length
802 bool Adafruit_MQTT_Publish::publish(uint8_t *payload, uint16_t bLen) {
803
804   return mqtt->publish(topic, payload, bLen, qos,retain);
805 }
806
```

```
807
808 // Adafruit_MQTT_Subscribe Definition ////////////////////////////////////////
809
810 Adafruit_MQTT_Subscribe::Adafruit_MQTT_Subscribe(Adafruit_MQTT *mqttserver,
811                                                  const char *feed, uint8_t q) {
812   mqtt = mqttserver;
813   topic = feed;
814   qos = q;
815   datalen = 0;
816   callback_uint32t = 0;
817   callback_buffer = 0;
818   callback_double = 0;
819   callback_io = 0;
820   io_feed = 0;
821 }
822
823 void Adafruit_MQTT_Subscribe::setCallback(SubscribeCallbackUInt32Type cb) {
824   callback_uint32t = cb;
825 }
826
827 void Adafruit_MQTT_Subscribe::setCallback(SubscribeCallbackDoubleType cb) {
828   callback_double = cb;
829 }
830
831 void Adafruit_MQTT_Subscribe::setCallback(SubscribeCallbackBufferType cb) {
832   callback_buffer = cb;
833 }
834
835 void Adafruit_MQTT_Subscribe::setCallback(AdafruitIO_Feed *f, SubscribeCallbackIOType
      cb) {
836   callback_io = cb;
837   io_feed = f;
838 }
839
840 void Adafruit_MQTT_Subscribe::removeCallback(void) {
841   callback_uint32t = 0;
842   callback_buffer = 0;
843   callback_double = 0;
844   callback_io = 0;
845   io_feed = 0;
846 }
```

Apéndice H

Adafruit_MQTT.h (modificado)

Adafruit_MQTT.h

```
// The MIT License (MIT)
//
// Copyright (c) 2015 Adafruit Industries
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in all
// copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
// SOFTWARE.
#ifndef _ADAFRUIT_MQTT_H_
#define _ADAFRUIT_MQTT_H_

#include "Arduino.h"

#if defined(ARDUINO_SAMD_ZERO) || defined(ARDUINO_STM32_FEATHER)
#define strncpy_P(dest, src, len) strncpy((dest), (src), (len))
#define strncasecmp_P(f1, f2, len) strncasecmp((f1), (f2), (len))
#endif

#define ADAFRUIT_MQTT_VERSION_MAJOR 0
#define ADAFRUIT_MQTT_VERSION_MINOR 16
#define ADAFRUIT_MQTT_VERSION_PATCH 2

// Uncomment/comment to turn on/off debug output messages.
//#define MQTT_DEBUG
// Uncomment/comment to turn on/off error output messages.
#define MQTT_ERROR

// Set where debug messages will be printed.
#define DEBUG_PRINTER Serial
// If using something like Zero or Due, change the above to SerialUSB
```

```
44
45  // Define actual debug output functions when necessary.
46  #ifdef MQTT_DEBUG
47    #define DEBUG_PRINT(...) { DEBUG_PRINTER.print(__VA_ARGS__); }
48    #define DEBUG_PRINTLN(...) { DEBUG_PRINTER.println(__VA_ARGS__); }
49    #define DEBUG_PRINTBUFFER(buffer, len) { printBuffer(buffer, len); }
50  #else
51    #define DEBUG_PRINT(...) {}
52    #define DEBUG_PRINTLN(...) {}
53    #define DEBUG_PRINTBUFFER(buffer, len) {}
54  #endif
55
56  #ifdef MQTT_ERROR
57    #define ERROR_PRINT(...) { DEBUG_PRINTER.print(__VA_ARGS__); }
58    #define ERROR_PRINTLN(...) { DEBUG_PRINTER.println(__VA_ARGS__); }
59    #define ERROR_PRINTBUFFER(buffer, len) { printBuffer(buffer, len); }
60  #else
61    #define ERROR_PRINT(...) {}
62    #define ERROR_PRINTLN(...) {}
63    #define ERROR_PRINTBUFFER(buffer, len) {}
64  #endif
65
66  // Use 3 (MQTT 3.0) or 4 (MQTT 3.1.1)
67  #define MQTT_PROTOCOL_LEVEL 4
68
69  #define MQTT_CTRL_CONNECT      0x1
70  #define MQTT_CTRL_CONNECTACK   0x2
71  #define MQTT_CTRL_PUBLISH      0x3
72  #define MQTT_CTRL_PUBACK       0x4
73  #define MQTT_CTRL_PUBREC       0x5
74  #define MQTT_CTRL_PUBREL       0x6
75  #define MQTT_CTRL_PUBCOMP      0x7
76  #define MQTT_CTRL_SUBSCRIBE    0x8
77  #define MQTT_CTRL_SUBACK       0x9
78  #define MQTT_CTRL_UNSUBSCRIBE  0xA
79  #define MQTT_CTRL_UNSUBACK     0xB
80  #define MQTT_CTRL_PINGREQ      0xC
81  #define MQTT_CTRL_PINGRESP     0xD
82  #define MQTT_CTRL_DISCONNECT   0xE
83
84  #define MQTT_QOS_1 0x1
85  #define MQTT_QOS_0 0x0
86
87  #define CONNECT_TIMEOUT_MS 6000
88  #define PUBLISH_TIMEOUT_MS 500
89  #define PING_TIMEOUT_MS    500
90  #define SUBACK_TIMEOUT_MS  500
91
92  // Adjust as necessary, in seconds.  Default to 5 minutes.
93  #define MQTT_CONN_KEEPALIVE 300
94
95  // Largest full packet we're able to send.
96  // Need to be able to store at least ~90 chars for a connect packet with full
97  // 23 char client ID.
98  #define MAXBUFFERSIZE (150)
99
100 #define MQTT_CONN_USERNAMEFLAG    0x80
101 #define MQTT_CONN_PASSWORDFLAG    0x40
102 #define MQTT_CONN_WILLRETAIN      0x20
103 #define MQTT_CONN_WILLQOS_1       0x08
```

```
104 #define MQTT_CONN_WILLQOS_2        0x18
105 #define MQTT_CONN_WILLFLAG         0x04
106 #define MQTT_CONN_CLEANSESSION     0x02
107
108 // how many subscriptions we want to be able to track
109 #define MAXSUBSCRIPTIONS 15
110
111 // how much data we save in a subscription object
112 // eg max-subscription-payload-size
113 #if defined  (__AVR_ATmega32U4__) || defined(__AVR_ATmega328P__)
114   #define SUBSCRIPTIONDATALEN 20
115 #else
116   #define SUBSCRIPTIONDATALEN 100
117 #endif
118
119 class AdafruitIO_Feed;  // forward decl
120
121 //Function pointer that returns an int
122 typedef void (*SubscribeCallbackUInt32Type)(uint32_t);
123 // returns a double
124 typedef void (*SubscribeCallbackDoubleType)(double);
125 // returns a chunk of raw data
126 typedef void (*SubscribeCallbackBufferType)(char *str, uint16_t len);
127 // returns an io data wrapper instance
128 typedef void (AdafruitIO_Feed::*SubscribeCallbackIOType)(char *str, uint16_t len);
129
130 extern void printBuffer(uint8_t *buffer, uint16_t len);
131
132 class Adafruit_MQTT_Subscribe;  // forward decl
133
134 class Adafruit_MQTT {
135  public:
136   Adafruit_MQTT(const char *server,
137                 uint16_t port,
138                 const char *cid,
139                 const char *user,
140                 const char *pass);
141
142   Adafruit_MQTT(const char *server,
143                 uint16_t port,
144                 const char *user = "",
145                 const char *pass = "");
146   virtual ~Adafruit_MQTT() {}
147
148   // Connect to the MQTT server.  Returns 0 on success, otherwise an error code
149   // that indicates something went wrong:
150   //   -1 = Error connecting to server
151   //    1 = Wrong protocol
152   //    2 = ID rejected
153   //    3 = Server unavailable
154   //    4 = Bad username or password
155   //    5 = Not authenticated
156   //    6 = Failed to subscribe
157   // Use connectErrorString() to get a printable string version of the
158   // error.
159   int8_t connect();
160   int8_t connect(const char *user, const char *pass);
161
162   // Return a printable string version of the error code returned by
163   // connect(). This returns a __FlashStringHelper*, which points to a
```

```
164   // string stored in flash, but can be directly passed to e.g.
165   // Serial.println without any further processing.
166   const __FlashStringHelper* connectErrorString(int8_t code);
167
168   // Sends MQTT disconnect packet and calls disconnectServer()
169   bool disconnect();
170
171   // Return true if connected to the MQTT server, otherwise false.
172   virtual bool connected() = 0;  // Subclasses need to fill this in!
173
174   // Set MQTT last will topic, payload, QOS, and retain. This needs
175   // to be called before connect() because it is sent as part of the
176   // connect control packet.
177   bool will(const char *topic, const char *payload, uint8_t qos = 0, uint8_t retain =
          0);
178
179   // Publish a message to a topic using the specified QoS level.  Returns true
180   // if the message was published, false otherwise.
181   bool publish(const char *topic, const char *payload, uint8_t qos = 0,uint8_t retain
          =1);
182   bool publish(const char *topic, uint8_t *payload, uint16_t bLen, uint8_t qos = 0,
          uint8_t retain=1);
183
184   // Add a subscription to receive messages for a topic.  Returns true if the
185   // subscription could be added or was already present, false otherwise.
186   // Must be called before connect(), subscribing after the connection
187   // is made is not currently supported.
188   bool subscribe(Adafruit_MQTT_Subscribe *sub);
189
190   // Unsubscribe from a previously subscribed MQTT topic.
191   bool unsubscribe(Adafruit_MQTT_Subscribe *sub);
192
193   // Check if any subscriptions have new messages.  Will return a reference to
194   // an Adafruit_MQTT_Subscribe object which has a new message.  Should be called
195   // in the sketch's loop function to ensure new messages are recevied.  Note
196   // that subscribe should be called first for each topic that receives messages!
197   Adafruit_MQTT_Subscribe *readSubscription(int16_t timeout=0);
198
199   void processPackets(int16_t timeout);
200
201   // Ping the server to ensure the connection is still alive.
202   bool ping(uint8_t n = 1);
203
204 protected:
205   // Interface that subclasses need to implement:
206
207   // Connect to the server and return true if successful, false otherwise.
208   virtual bool connectServer() = 0;
209
210   // Disconnect from the MQTT server.  Returns true if disconnected, false otherwise.
211   virtual bool disconnectServer() = 0;  // Subclasses need to fill this in!
212
213   // Send data to the server specified by the buffer and length of data.
214   virtual bool sendPacket(uint8_t *buffer, uint16_t len) = 0;
215
216   // Read MQTT packet from the server.  Will read up to maxlen bytes and store
217   // the data in the provided buffer.  Waits up to the specified timeout (in
218   // milliseconds) for data to be available.
219   virtual uint16_t readPacket(uint8_t *buffer, uint16_t maxlen, int16_t timeout) = 0;
220
```

```
221   // Read a full packet, keeping note of the correct length
222   uint16_t readFullPacket(uint8_t *buffer, uint16_t maxsize, uint16_t timeout);
223   // Properly process packets until you get to one you want
224   uint16_t processPacketsUntil(uint8_t *buffer, uint8_t waitforpackettype, uint16_t
          timeout);
225
226   // Shared state that subclasses can use:
227   const char *servername;
228   int16_t portnum;
229   const char *clientid;
230   const char *username;
231   const char *password;
232   const char *will_topic;
233   const char *will_payload;
234   uint8_t will_qos;
235   uint8_t will_retain;
236   uint8_t buffer[MAXBUFFERSIZE];  // one buffer, used for all incoming/outgoing
237   uint16_t packet_id_counter;
238
239  private:
240   Adafruit_MQTT_Subscribe *subscriptions[MAXSUBSCRIPTIONS];
241
242   void     flushIncoming(uint16_t timeout);
243
244   // Functions to generate MQTT packets.
245   uint8_t connectPacket(uint8_t *packet);
246   uint8_t disconnectPacket(uint8_t *packet);
247   uint16_t publishPacket(uint8_t *packet, const char *topic, uint8_t *payload, uint16_t
          bLen, uint8_t qos, uint8_t retain);
248   uint8_t subscribePacket(uint8_t *packet, const char *topic, uint8_t qos);
249   uint8_t unsubscribePacket(uint8_t *packet, const char *topic);
250   uint8_t pingPacket(uint8_t *packet);
251   uint8_t pubackPacket(uint8_t *packet, uint16_t packetid);
252 };
253
254
255 class Adafruit_MQTT_Publish {
256  public:
257   Adafruit_MQTT_Publish(Adafruit_MQTT *mqttserver, const char *feed, uint8_t qos = 0,
          uint8_t retain=1);
258
259   bool publish(const char *s);
260   bool publish(double f, uint8_t precision=2);  // Precision controls the minimum
          number of digits after decimal.
261                                                 // This might be ignored and a higher
                                                        precision value sent.
262   bool publish(int32_t i);
263   bool publish(uint32_t i);
264   bool publish(uint8_t *b, uint16_t bLen);
265
266
267 private:
268   Adafruit_MQTT *mqtt;
269   const char *topic;
270   uint8_t qos;
271   uint8_t retain;
272 };
273
274 class Adafruit_MQTT_Subscribe {
275  public:
```

```
276   Adafruit_MQTT_Subscribe(Adafruit_MQTT *mqttserver, const char *feedname, uint8_t q=0)
          ;
277
278   void setCallback(SubscribeCallbackUInt32Type callb);
279   void setCallback(SubscribeCallbackDoubleType callb);
280   void setCallback(SubscribeCallbackBufferType callb);
281   void setCallback(AdafruitIO_Feed *io, SubscribeCallbackIOType callb);
282   void removeCallback(void);
283
284   const char *topic;
285   uint8_t qos;
286
287   uint8_t lastread[SUBSCRIPTIONDATALEN];
288   // Number valid bytes in lastread. Limited to SUBSCRIPTIONDATALEN-1 to
289   // ensure nul terminating lastread.
290   uint16_t datalen;
291
292   SubscribeCallbackUInt32Type callback_uint32t;
293   SubscribeCallbackDoubleType callback_double;
294   SubscribeCallbackBufferType callback_buffer;
295   SubscribeCallbackIOType     callback_io;
296
297   AdafruitIO_Feed *io_feed;
298
299  private:
300   Adafruit_MQTT *mqtt;
301 };
302
303
304 #endif
```

Apéndice I

build.gradle

build.gradle

```
// Top-level build file where you can add configuration options common to all sub-
    projects/modules.

buildscript {
    repositories {
        jcenter()
        maven {
            url "https://repo.eclipse.org/content/repositories/paho-releases/"
        }
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.2.3'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        jcenter()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Apéndice J

AndroidManifest.xml

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.usuario.toldoautomatico">

    //Permisos que necesita el servicio Android Paho
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity"
                  android:windowSoftInputMode="stateHidden">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name="org.eclipse.paho.android.service.MqttService" >
        </service>
    </application>

</manifest>
```

Apéndice K

MainActivity.java

MainActivity.java

```
1  package com.example.usuario.toldoautomatico;
2
3  import android.support.v7.app.AlertDialog;
4  import android.support.v7.app.AppCompatActivity;
5  import android.os.Bundle;
6  import android.util.Log;
7  import android.view.KeyEvent;
8  import android.view.View;
9  import android.widget.Button;
10 import android.widget.EditText;
11 import android.widget.TextView;
12 import android.view.inputmethod.EditorInfo;
13 import android.widget.Toast;
14
15 import org.eclipse.paho.android.service.MqttAndroidClient;
16 import org.eclipse.paho.client.mqttv3.IMqttActionListener;
17 import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
18 import org.eclipse.paho.client.mqttv3.IMqttToken;
19 import org.eclipse.paho.client.mqttv3.MqttClient;
20 import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
21 import org.eclipse.paho.client.mqttv3.MqttException;
22 import org.eclipse.paho.client.mqttv3.MqttMessage;
23 import org.eclipse.paho.client.mqttv3.MqttTopic;
24 import org.eclipse.paho.client.mqttv3.MqttCallback;
25 import org.w3c.dom.Text;
26
27 import java.io.IOException;
28 import java.io.InputStream;
29 import java.io.UnsupportedEncodingException;
30
31 import static java.lang.Integer.parseInt;
32
33 public class MainActivity extends AppCompatActivity  {
34
35     private TextView estadoToldo;
36     private TextView vientoMedido;
37     private TextView temperaturaMedida;
38     private TextView humedadMedida;
39     private TextView luzMedida;
40     private TextView actDes;
41     private TextView error;
42     private TextView subaj;
43     private TextView conexion;
```

```
44      private EditText clase;
45      private EditText temperaturaSubir1;
46      private EditText temperaturaBajar1;
47      private EditText humedadSubir1;
48      private EditText humedadBajar1;
49      private EditText luzSubir1;
50      private EditText luzBajar1;
51      private EditText hora1;
52      private EditText hora2;
53      private Button activar;
54      private Button desactivar;
55      private Button subir;
56      private Button bajar;
57      private Button parar;
58      private Button quitarModo;
59
60      String topic1="david/toldo/estadoToldo";
61      String topic2="david/toldo/viento";
62      String topic3="david/toldo/temperatura";
63      String topic4="david/toldo/humedad";
64      String topic5="david/toldo/luz";
65      String topic6="david/toldo/Clase";
66      String topic7="david/toldo/TemperaturaUmbrales";
67      String topic8="david/toldo/HumedadUmbrales";
68      String topic9="david/toldo/LuzSubir1";
69      String topic10="david/toldo/LuzBajar1";
70      String topic11="david/toldo/modoVacaciones";
71      String topic12="david/toldo/Horas";
72      String topic13="david/toldo/Inicial1";
73      String topic14="david/toldo/errorSuscripcion";
74      String topic15="david/toldo/Cambiar";
75
76
77      int claseIntro=1,luzSubirIntro=8000,luzBajarIntro=12000,h1Intro=14,h2Intro=15,
            modoVacaciones=0,
78          inicial1,inicial2,estadoToldoIntro, errorRecibido, subirBajar=0,tempSubirIntro
                =12,
79          tempBajarIntro=20,humSubirIntro=60,humBajarIntro=40, horas, temperaturaUmbrales
                ,
80          humedadUmbrales, tempSubirDesp, humSubirDesp, h2Desp;
81      String messageClase="1";
82      String tempSubir="12";
83      String tempBajar="20";
84      String humSubir="60";
85      String humBajar="40";
86      String luzSubir="400";
87      String luzBajar="600";
88      String h1="14";
89      String h2="15";
90      String estToldo,viento,temperatura,humedad,luz,tempUmb,humUmb,hs;
91      private static String USERNAME = "david";
92      private static String PASSWORD = "2016david";
93      private static final String TAG = "MainActivity";
94      private MqttAndroidClient client;
95
96      @Override
97      protected void onCreate(Bundle savedInstanceState) {
98          super.onCreate(savedInstanceState);
99          setContentView(R.layout.activity_main);
100
```

```
        estadoToldo=(TextView) findViewById(R.id.textView2);
        vientoMedido=(TextView) findViewById(R.id.textView4);
        temperaturaMedida=(TextView) findViewById(R.id.textView6);
        humedadMedida=(TextView) findViewById(R.id.textView8);
        luzMedida=(TextView) findViewById(R.id.textView10);
        actDes=(TextView) findViewById(R.id.textView20);
        error=(TextView) findViewById(R.id.textView24);
        clase= (EditText) findViewById(R.id.editText);
        temperaturaSubir1=(EditText) findViewById(R.id.editText2);
        temperaturaBajar1=(EditText) findViewById(R.id.editText3);
        humedadSubir1=(EditText) findViewById(R.id.editText4);
        humedadBajar1=(EditText) findViewById(R.id.editText5);
        luzSubir1=(EditText) findViewById(R.id.editText6);
        luzBajar1=(EditText) findViewById(R.id.editText7);
        hora1=(EditText) findViewById(R.id.editText8);
        hora2=(EditText) findViewById(R.id.editText9);
        activar=(Button) findViewById(R.id.button);
        desactivar=(Button) findViewById(R.id.button2);
        subir=(Button) findViewById(R.id.button3);
        bajar=(Button) findViewById(R.id.button4);
        parar=(Button) findViewById(R.id.button5);
        quitarModo=(Button) findViewById(R.id.button6);
        subaj=(TextView) findViewById(R.id.textView28);
        conexion=(TextView) findViewById(R.id.textView25);

        clase.setText(messageClase);
        temperaturaSubir1.setText(tempSubir);
        temperaturaBajar1.setText(tempBajar);
        humedadSubir1.setText(humSubir);
        humedadBajar1.setText(humBajar);
        luzSubir1.setText(luzSubir);
        luzBajar1.setText(luzBajar);
        actDes.setText("Desactivado");
        subaj.setText("Desactivado");
        hora1.setText(h1);
        hora2.setText(h2);

        //Generamos un id de cliente aleatorio
        String clientId = MqttClient.generateClientId();
        //Creamos una instancia de MqttAndroidClient
        client=new MqttAndroidClient(this.getApplicationContext(),"ssl://david2016.
            dtdns.net:8883", clientId);
        //Primero configuramos el nombre y la contraseyna del broker
        //Para ello creamos una instancia de MqttConnectOptions
        MqttConnectOptions options = new MqttConnectOptions();
        options.setUserName(USERNAME);
        options.setPassword(PASSWORD.toCharArray());
        //Ahora intentamos conectar con el broker MQTT y retornamos el token
        //el token nos sirve para saber si se ha conectado con el broker correctamente

        try {
            InputStream input=this.getApplicationContext().getAssets().open("server.bks
                ");
            options.setSocketFactory(client.getSSLSocketFactory(input,"2016david"));
            IMqttToken token = client.connect(options);
            token.setActionCallback(new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken iMqttToken) {
                    Log.d(TAG, "Conexion establecida");
                    conexion.setText("Establecida");
```

```
159                     Toast.makeText(getApplicationContext(), "Conexion establecida",
                            Toast.LENGTH_LONG).show();
160
161                     subscribeTopic(topic13);
162                     subscribeTopic(topic1);
163                     subscribeTopic(topic2);
164                     subscribeTopic(topic3);
165                     subscribeTopic(topic4);
166                     subscribeTopic(topic5);
167                     subscribeTopic(topic6);
168                     subscribeTopic(topic7);
169                     subscribeTopic(topic8);
170                     subscribeTopic(topic9);
171                     subscribeTopic(topic10);
172                     subscribeTopic(topic11);
173                     subscribeTopic(topic12);
174                     subscribeTopic(topic14);
175                     subscribeTopic(topic15);
176
177                     publishClass();
178                     publishTemperaturaSubir();
179                     publishTemperaturaBajar();
180                     publishHumedadSubir();
181                     publishHumedadBajar();
182                     publishLuzSubir();
183                     publishLuzBajar();
184                     publishModoVacaciones();
185                     publishHora1();
186                     publishHora2();
187                     publishCambiarEstadoToldo();
188
189                 }
190
191                 @Override
192                 public void onFailure(IMqttToken iMqttToken, Throwable exception) {
193                     Log.d(TAG, "Conexion fallida");
194                     conexion.setText("Fallida");
195                     Toast.makeText(getApplicationContext(), "Conexion fallida", Toast.
                            LENGTH_LONG).show();
196                 }
197             });
198         } catch (MqttException e) {
199             e.printStackTrace();
200         }catch (IOException e) {
201             e.printStackTrace();
202         }
203         client.setCallback(new MqttCallback() {
204             @Override
205             public void connectionLost(Throwable throwable) {
206
207             }
208
209             @Override
210             public void messageArrived(String topic, MqttMessage message) throws
                    Exception {
211                 if(topic.compareTo(topic1)==0){
212                     estadoToldoIntro=Integer.parseInt(new String(message.getPayload()))
                            ;
213                     estadoToldoIntro=estadoToldoIntro*3/1000;
214                     estToldo=Integer.toString(estadoToldoIntro);
```

```
215                     estadoToldo.setText(estToldo);
216                 }
217                 else if(topic.compareTo(topic2)==0) {
218                     viento=new String(message.getPayload());
219                     vientoMedido.setText(viento);
220 
221                 }
222                 else if(topic.compareTo(topic3)==0){
223                     temperatura=new String(message.getPayload());
224                     temperaturaMedida.setText(temperatura);
225                 }
226                 else if(topic.compareTo(topic4)==0){
227                     humedad=new String(message.getPayload());
228                     humedadMedida.setText(humedad);
229                 }
230                 else if(topic.compareTo(topic5)==0){
231                     luz=new String(message.getPayload());
232                     luzMedida.setText(luz);
233                 }
234                 else if(topic.compareTo(topic6)==0){
235                     clase.setText(new String(message.getPayload()));
236                     claseIntro=Integer.parseInt(new String(message.getPayload()));
237                 }
238                 else if(topic.compareTo(topic7)==0){
239                     temperaturaUmbrales=Integer.parseInt(new String(message.getPayload
                            ()));
240                     tempUmb=Integer.toString(temperaturaUmbrales);
241 
242                     tempBajarIntro=temperaturaUmbrales&0x00FF;
243                     tempBajar=Integer.toString(tempBajarIntro);
244                     temperaturaBajar1.setText(tempBajar);
245 
246                     tempSubirIntro=(temperaturaUmbrales&0xFF00)>>8;
247                     tempSubir=Integer.toString(tempSubirIntro);
248                     temperaturaSubir1.setText(tempSubir);
249 
250                 }
251                 else if(topic.compareTo(topic8)==0){
252                     humedadUmbrales=Integer.parseInt(new String(message.getPayload()));
253                     humUmb=Integer.toString(humedadUmbrales);
254 
255                     humBajarIntro=humedadUmbrales&0x00FF;
256                     humBajar=Integer.toString(humBajarIntro);
257                     humedadBajar1.setText(humBajar);
258 
259                     humSubirIntro=(humedadUmbrales&0xFF00)>>8;
260                     humSubir=Integer.toString(humSubirIntro);
261                     humedadSubir1.setText(humSubir);
262                 }
263                 else if(topic.compareTo(topic9)==0){
264                     luzSubir1.setText(new String(message.getPayload()));
265                     luzSubirIntro=Integer.parseInt(new String(message.getPayload()));
266                 }
267                 else if(topic.compareTo(topic10)==0){
268                     luzBajar1.setText(new String(message.getPayload()));
269                     luzBajarIntro=Integer.parseInt(new String(message.getPayload()));
270                 }
271                 else if(topic.compareTo(topic11)==0){
272                     modoVacaciones = Integer.parseInt(new String(message.getPayload()))
                            ;
```

```
273                     if(modoVacaciones==0){
274                         actDes.setText("Desactivado");
275                     }
276                     else if(modoVacaciones==1){
277                         actDes.setText("Activado");
278                     }
279 
280                 }
281                 else if(topic.compareTo(topic12)==0){
282                     horas=Integer.parseInt(new String(message.getPayload()));
283                     hs=Integer.toString(horas);
284 
285                     h1Intro=horas&0x00FF;
286                     h1=Integer.toString(h1Intro);
287                     hora1.setText(h1);
288 
289                     h2Intro=(horas&0xFF00)>>8;
290                     h2=Integer.toString(h2Intro);
291                     hora2.setText(h2);
292                 }
293                 else if(topic.compareTo(topic13)==0){
294                     inicial1 = Integer.parseInt(new String(message.getPayload()));
295                     modoVacaciones=inicial1&0x0001;
296                     if(modoVacaciones==0){
297                         actDes.setText("Desactivado");
298                     }
299                     else if(modoVacaciones==1){
300                         actDes.setText("Activado");
301                     }
302                     claseIntro=((inicial1&0x0002)>>1);
303                     messageClase=Integer.toString(claseIntro);
304                     clase.setText(messageClase);
305 
306                     estadoToldoIntro=((inicial1&0x0004)>>2);
307                     estToldo=Integer.toString(estadoToldoIntro);
308                     estadoToldo.setText(estToldo);
309 
310                     errorRecibido=((inicial1&0x0008)>>3);
311                     if(errorRecibido==0){
312                         error.setText("Suscripciones␣modulo␣wifi␣correctas");
313                     }
314                     else {
315                         error.setText("Error␣en␣las␣suscripciones␣del␣modulo␣wifi");
316                     }
317 
318                 }
319                 else if(topic.compareTo(topic14)==0){
320                     errorRecibido = Integer.parseInt(new String(message.getPayload()));
321                     if(errorRecibido==0){
322                         error.setText("Suscripciones␣modulo␣wifi␣correctas");
323                     }
324                     else{
325                         error.setText("Error␣en␣las␣suscripciones␣del␣modulo␣wifi");
326                     }
327                 }
328                 else if(topic.compareTo(topic15)==0){
329                     subirBajar = Integer.parseInt(new String(message.getPayload()));
330                     subirBajar=subirBajar&0x0001;
331                     if(subirBajar==0){
332                         subaj.setText("Desactivado");
```

```
333                     }
334                     else{
335                         subaj.setText("Activado");
336                     }
337                 }
338             }
339 
340             @Override
341             public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
342 
343             }
344         });
345     }
346     //Metodo para guardar el valor de las variables ya que se borra al girar el movil o
347     //dejar la aplicacion en segundo plano
348     protected void onSaveInstanceState(Bundle outState){
349         super.onSaveInstanceState(outState);
350         outState.putInt("ERROR",errorRecibido);
351         outState.putInt("CLASE",claseIntro);
352         outState.putInt("LUZSUBIR",luzSubirIntro);
353         outState.putInt("LUZBAJAR",luzBajarIntro);
354         outState.putInt("HORA1",h1Intro);
355         outState.putInt("HORA2",h2Intro);
356         outState.putInt("MODOVACACIONES",modoVacaciones);
357         outState.putInt("INICIAL1",inicial1);
358         outState.putInt("INICIAL2",inicial2);
359         outState.putInt("STOLDO",estadoToldoIntro);
360         outState.putInt("SUBAJ",subirBajar);
361         outState.putInt("TEMPSUBIR",tempSubirIntro);
362         outState.putInt("TEMPBAJAR",tempBajarIntro);
363         outState.putInt("HUMSUBIR",humSubirIntro);
364         outState.putInt("HUMBAJAR",humBajarIntro);
365         outState.putInt("HUMBRALES",humedadUmbrales);
366         outState.putInt("TUMBRALES",temperaturaUmbrales);
367         outState.putInt("HDESP",h2Desp);
368         outState.putInt("TDESP",tempSubirDesp);
369         outState.putInt("HDESP",humSubirDesp);
370 
371         outState.putString("TU",tempUmb);
372         outState.putString("HU",humUmb);
373         outState.putString("HS",hs);
374         outState.putString("MCLASE",messageClase);
375         outState.putString("TEMPS",tempSubir);
376         outState.putString("TEMPB",tempBajar);
377         outState.putString("HUMS",humSubir);
378         outState.putString("HUMB",humBajar);
379         outState.putString("LUZS",luzSubir);
380         outState.putString("LUZB",luzBajar);
381         outState.putString("H1",h1);
382         outState.putString("H2",h2);
383         outState.putString("ESTOLDO",estToldo);
384         outState.putString("VIENTO",viento);
385         outState.putString("TEMPERATURA",temperatura);
386         outState.putString("HUMEDAD",humedad);
387         outState.putString("LUZ",luz);
388     }
389     //Metodo para restaurar valores
390     protected void onRestoreInstanceState(Bundle savedInstanceState){
391         super.onRestoreInstanceState(savedInstanceState);
392         //Restauramos strings
```

```
393             messageClase=savedInstanceState.getString("MCLASE");
394             clase.setText(messageClase);
395             tempUmb=savedInstanceState.getString("TU");
396             humUmb=savedInstanceState.getString("HU");
397             hs=savedInstanceState.getString("HS");
398             tempSubir=savedInstanceState.getString("TEMPS");
399             temperaturaSubir1.setText(tempSubir);
400             tempBajar=savedInstanceState.getString("TEMPB");
401             temperaturaBajar1.setText(tempBajar);
402             humSubir=savedInstanceState.getString("HUMS");
403             humedadSubir1.setText(humSubir);
404             humBajar=savedInstanceState.getString("HUMB");
405             humedadBajar1.setText(humBajar);
406             luzSubir=savedInstanceState.getString("LUZS");
407             luzSubir1.setText(luzSubir);
408             luzBajar=savedInstanceState.getString("LUZB");
409             luzBajar1.setText(luzBajar);
410             h1=savedInstanceState.getString("H1");
411             hora1.setText(h1);
412             h2=savedInstanceState.getString("H2");
413             hora2.setText(h2);
414             estToldo=savedInstanceState.getString("ESTOLDO");
415             estadoToldo.setText(estToldo);
416             temperatura=savedInstanceState.getString("TEMPERATURA");
417             temperaturaMedida.setText(temperatura);
418             viento=savedInstanceState.getString("VIENTO");
419             vientoMedido.setText(viento);
420             humedad=savedInstanceState.getString("HUMEDAD");
421             humedadMedida.setText(humedad);
422             luz=savedInstanceState.getString("LUZ");
423             luzMedida.setText(luz);
424
425             //Restauramos ints
426             errorRecibido=savedInstanceState.getInt("ERROR");
427             claseIntro=savedInstanceState.getInt("CLASE");
428             luzSubirIntro=savedInstanceState.getInt("LUZSUBIR");
429             luzBajarIntro=savedInstanceState.getInt("LUZBAJAR");
430             h1Intro=savedInstanceState.getInt("HORA1");
431             h2Intro=savedInstanceState.getInt("HORA2");
432             modoVacaciones=savedInstanceState.getInt("MODOVACACIONES");
433             inicial1=savedInstanceState.getInt("INICIAL1");
434             inicial2=savedInstanceState.getInt("INICIAL2");
435             estadoToldoIntro=savedInstanceState.getInt("STOLDO");
436             subirBajar=savedInstanceState.getInt("SUBAJ");
437             tempSubirIntro=savedInstanceState.getInt("TEMPSUBIR");
438             tempBajarIntro=savedInstanceState.getInt("TEMPBAJAR");
439             humSubirIntro=savedInstanceState.getInt("HUMSUBIR");
440             humBajarIntro=savedInstanceState.getInt("HUMBAJAR");
441             humedadUmbrales=savedInstanceState.getInt("HUMBRALES");
442             temperaturaUmbrales=savedInstanceState.getInt("TUMBRALES");
443             h2Desp=savedInstanceState.getInt("HDESP");
444             tempSubirDesp=savedInstanceState.getInt("TDESP");
445             humSubirDesp=savedInstanceState.getInt("HDESP");
446         }
447         public void publishTopic(String payload, String topic) {
448             byte[] encodedPayload = new byte[0];
449             try {
450                 encodedPayload = payload.getBytes("UTF-8");
451                 MqttMessage message = new MqttMessage(encodedPayload);
452                 message.setRetained(true);
```

```
453             client.publish(topic, message);
454         } catch (UnsupportedEncodingException | MqttException e) {
455             e.printStackTrace();
456         }
457     }
458
459
460
461     private void subscribeTopic(String topic) {
462         int qos = 1;
463         try {
464             client.subscribe(topic,qos);
465
466         } catch (MqttException e) {
467             e.printStackTrace();
468         }
469     }
470
471     public void publishClass(){
472         clase.setOnKeyListener(new View.OnKeyListener() {
473             @Override
474             public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
475                 if((keyEvent.getAction()==KeyEvent.ACTION_DOWN) &&
476                         (keyCode==KeyEvent.KEYCODE_ENTER)){
477                     messageClase = clase.getText().toString();
478                     claseIntro= parseInt(messageClase);
479                     if(claseIntro==1 || claseIntro==2 || claseIntro==3){
480                         publishTopic(messageClase,topic6);
481                     }
482                     else{
483                         AlertDialog alertDialog;
484                         alertDialog=new AlertDialog.Builder(MainActivity.this).create()
                                ;
485                         alertDialog.setTitle("Error");
486                         alertDialog.setMessage("La␣clase␣solo␣puede␣ser␣1,2␣o␣3");
487                         alertDialog.show();
488                     }
489
490                     return true;
491                 }
492                 return false;
493             }
494         });
495     }
496
497     public void publishTemperaturaSubir(){
498         temperaturaSubir1.setOnKeyListener(new View.OnKeyListener() {
499             @Override
500             public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
501                 if((keyEvent.getAction()==KeyEvent.ACTION_DOWN) &&
502                         (keyCode==KeyEvent.KEYCODE_ENTER)){
503                     tempSubir = temperaturaSubir1.getText().toString();
504                     tempSubirIntro= Integer.parseInt(tempSubir);
505                     tempSubirDesp=tempSubirIntro<<8;
506                     tempSubir=Integer.toString(tempSubirDesp);
507
508                     tempBajar = temperaturaBajar1.getText().toString();
509                     tempBajarIntro= Integer.parseInt(tempBajar);
510
511                     temperaturaUmbrales=tempSubirDesp|tempBajarIntro;
```

```
512                     tempUmb=Integer.toString(temperaturaUmbrales);
513                     if(tempSubirIntro<tempBajarIntro) {
514                         publishTopic(tempUmb, topic7);
515                     }
516                     else{
517                         AlertDialog alertDialog;
518                         alertDialog=new AlertDialog.Builder(MainActivity.this).create()
                                ;
519                         alertDialog.setTitle("Error");
520                         alertDialog.setMessage("La temperatura de subir debe ser menor
                            que la temperatura de bajar");
521                         alertDialog.show();
522                     }
523                     return true;
524                 }
525                 return false;
526             }
527         });
528     }
529
530     public void publishTemperaturaBajar(){
531         temperaturaBajar1.setOnKeyListener(new View.OnKeyListener() {
532             @Override
533             public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
534                 if((keyEvent.getAction()==KeyEvent.ACTION_DOWN) &&
535                         (keyCode==KeyEvent.KEYCODE_ENTER)){
536
537                     tempSubir = temperaturaSubir1.getText().toString();
538                     tempSubirIntro= Integer.parseInt(tempSubir);
539                     tempSubirDesp=tempSubirIntro<<8;
540                     tempSubir=Integer.toString(tempSubirDesp);
541
542                     tempBajar = temperaturaBajar1.getText().toString();
543                     tempBajarIntro= Integer.parseInt(tempBajar);
544
545                     temperaturaUmbrales=tempSubirDesp|tempBajarIntro;
546                     tempUmb=Integer.toString(temperaturaUmbrales);
547                     if(tempBajarIntro>tempSubirIntro){
548                         publishTopic(tempUmb,topic7);
549                     }
550                     else{
551                         AlertDialog alertDialog;
552                         alertDialog=new AlertDialog.Builder(MainActivity.this).create()
                                ;
553                         alertDialog.setTitle("Error");
554                         alertDialog.setMessage("La temperatura de bajar debe ser mayor
                            que la temperatura de subir");
555                         alertDialog.show();
556                     }
557
558                     return true;
559                 }
560                 return false;
561             }
562         });
563     }
564     public void publishHumedadSubir(){
565         humedadSubir1.setOnKeyListener(new View.OnKeyListener() {
566             @Override
567             public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
```

```
568                 if((keyEvent.getAction()==KeyEvent.ACTION_DOWN) &&
569                         (keyCode==KeyEvent.KEYCODE_ENTER)){
570                     humSubir = humedadSubir1.getText().toString();
571                     humSubirIntro=Integer.parseInt(humSubir);
572                     humSubirDesp=humSubirIntro<<8;
573                     humSubir=Integer.toString(humSubirDesp);
574 
575                     humBajar = humedadBajar1.getText().toString();
576                     humBajarIntro=Integer.parseInt(humBajar);
577 
578                     humedadUmbrales=humSubirDesp|humBajarIntro;
579                     humUmb=Integer.toString(humedadUmbrales);
580 
581                     if(humSubirIntro>humBajarIntro) {
582                         publishTopic(humUmb, topic8);
583                     }
584                     else{
585                         AlertDialog alertDialog;
586                         alertDialog=new AlertDialog.Builder(MainActivity.this).create()
                                ;
587                         alertDialog.setTitle("Error");
588                         alertDialog.setMessage("La humedad de subir debe ser mayor que
                                la humedad de bajar");
589                         alertDialog.show();
590                     }
591 
592                     return true;
593                 }
594                 return false;
595             }
596         });
597     }
598 
599     public void publishHumedadBajar(){
600         humedadBajar1.setOnKeyListener(new View.OnKeyListener() {
601             @Override
602             public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
603                 if((keyEvent.getAction()==KeyEvent.ACTION_DOWN) &&
604                         (keyCode==KeyEvent.KEYCODE_ENTER)){
605                     humSubir = humedadSubir1.getText().toString();
606                     humSubirIntro=Integer.parseInt(humSubir);
607                     humSubirDesp=humSubirIntro<<8;
608                     humSubir=Integer.toString(humSubirDesp);
609 
610                     humBajar = humedadBajar1.getText().toString();
611                     humBajarIntro=Integer.parseInt(humBajar);
612 
613                     humedadUmbrales=humSubirDesp|humBajarIntro;
614                     humUmb=Integer.toString(humedadUmbrales);
615                     if(humBajarIntro<humSubirIntro){
616                         publishTopic(humUmb,topic8);
617                     }
618                     else{
619                         AlertDialog alertDialog;
620                         alertDialog=new AlertDialog.Builder(MainActivity.this).create()
                                ;
621                         alertDialog.setTitle("Error");
622                         alertDialog.setMessage("La humedad de bajar debe ser menor que
                                la humedad de subir");
623                         alertDialog.show();
```

```
624                     }
625
626                     return true;
627                 }
628                 return false;
629             }
630         });
631     }
632     public void publishLuzSubir(){
633         luzSubir1.setOnKeyListener(new View.OnKeyListener() {
634             @Override
635             public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
636                 if((keyEvent.getAction()==KeyEvent.ACTION_DOWN) &&
637                         (keyCode==KeyEvent.KEYCODE_ENTER)){
638                     luzSubir = luzSubir1.getText().toString();
639                     luzSubirIntro=Integer.parseInt(luzSubir);
640                     if(luzSubirIntro<luzBajarIntro) {
641                         publishTopic(luzSubir, topic9);
642                     }
643                     else{
644                         AlertDialog alertDialog;
645                         alertDialog=new AlertDialog.Builder(MainActivity.this).create()
                                ;
646                         alertDialog.setTitle("Error");
647                         alertDialog.setMessage("La luz de subir debe ser menor que la
                            luz de bajar");
648                         alertDialog.show();
649                     }
650                     return true;
651                 }
652                 return false;
653             }
654         });
655     }
656
657     public void publishLuzBajar(){
658         luzBajar1.setOnKeyListener(new View.OnKeyListener() {
659             @Override
660             public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
661                 if((keyEvent.getAction()==KeyEvent.ACTION_DOWN) &&
662                         (keyCode==KeyEvent.KEYCODE_ENTER)){
663                     luzBajar = luzBajar1.getText().toString();
664                     luzBajarIntro=Integer.parseInt(luzBajar);
665                     if(luzBajarIntro>luzSubirIntro){
666                         publishTopic(luzBajar,topic10);
667                     }
668                     else{
669                         AlertDialog alertDialog;
670                         alertDialog=new AlertDialog.Builder(MainActivity.this).create()
                                ;
671                         alertDialog.setTitle("Error");
672                         alertDialog.setMessage("La luz de bajar debe ser mayor que la
                            luz de subir");
673                         alertDialog.show();
674                     }
675
676                     return true;
677                 }
678                 return false;
679             }
```

```
680         });
681     }
682 
683     public void publishModoVacaciones(){
684         activar.setOnClickListener(new View.OnClickListener() {
685             @Override
686             public void onClick(View view) {
687                 publishTopic("1",topic11);
688                 actDes.setText("Activado");
689             }
690         });
691         desactivar.setOnClickListener(new View.OnClickListener() {
692             @Override
693             public void onClick(View view) {
694                 publishTopic("0",topic11);
695                 actDes.setText("Desactivado");
696             }
697         });
698     }
699     public void publishHora1(){
700         hora1.setOnKeyListener(new View.OnKeyListener() {
701             @Override
702             public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
703                 if((keyEvent.getAction()==KeyEvent.ACTION_DOWN) &&
704                         (keyCode==KeyEvent.KEYCODE_ENTER)){
705                     h1 = hora1.getText().toString();
706                     h1Intro=Integer.parseInt(h1);
707 
708                     h2 = hora2.getText().toString();
709                     h2Intro=Integer.parseInt(h2);
710                     h2Desp=h2Intro<<8;
711                     h2=Integer.toString(h2Desp);
712 
713 
714                     horas=h2Desp|h1Intro;
715                     hs=Integer.toString(horas);
716                     if(h1Intro>=0 && h1Intro<=24){
717                         publishTopic(hs,topic12);
718                     }
719                     else{
720                         AlertDialog alertDialog;
721                         alertDialog=new AlertDialog.Builder(MainActivity.this).create()
                                ;
722                         alertDialog.setTitle("Error");
723                         alertDialog.setMessage("Hora␣incorrecta");
724                         alertDialog.show();
725                     }
726                     return true;
727                 }
728                 return false;
729             }
730         });
731     }
732     public void publishHora2(){
733         hora2.setOnKeyListener(new View.OnKeyListener() {
734             @Override
735             public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
736                 if((keyEvent.getAction()==KeyEvent.ACTION_DOWN) &&
737                         (keyCode==KeyEvent.KEYCODE_ENTER)){
738 
```

```
739                     h1 = hora1.getText().toString();
740                     h1Intro=Integer.parseInt(h1);
741
742                     h2 = hora2.getText().toString();
743                     h2Intro=Integer.parseInt(h2);
744                     h2Desp=h2Intro<<8;
745                     h2=Integer.toString(h2Desp);
746
747
748                     horas=h2Desp|h1Intro;
749                     hs=Integer.toString(horas);
750                     if(h2Intro>=0 && h2Intro<=24){
751                         publishTopic(hs,topic12);
752                     }
753                     else{
754                         AlertDialog alertDialog;
755                         alertDialog=new AlertDialog.Builder(MainActivity.this).create()
                                ;
756                         alertDialog.setTitle("Error");
757                         alertDialog.setMessage("Hora␣incorrecta");
758                         alertDialog.show();
759                     }
760                     return true;
761                 }
762                 return false;
763             }
764         });
765     }
766     public void publishCambiarEstadoToldo(){
767         subir.setOnClickListener(new View.OnClickListener() {
768             @Override
769             public void onClick(View view) {
770                 publishTopic("1",topic15);
771                 subaj.setText("Activado");
772             }
773         });
774         bajar.setOnClickListener(new View.OnClickListener() {
775             @Override
776             public void onClick(View view) {
777                 publishTopic("3",topic15);
778                 subaj.setText("Activado");
779             }
780         });
781         parar.setOnClickListener(new View.OnClickListener() {
782             @Override
783             public void onClick(View view) {
784                 publishTopic("5",topic15);
785                 subaj.setText("Activado");
786             }
787         });
788        quitarModo.setOnClickListener(new View.OnClickListener() {
789             @Override
790             public void onClick(View view) {
791                 publishTopic("0",topic15);
792                 subaj.setText("Desactivado");
793             }
794         });
795     }
796 }
```

Apéndice L

activity_main.xml

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.usuario.toldoautomatico.MainActivity"
    android:background="?android:attr/colorBackground"
    android:weightSum="1"
    android:orientation="vertical">

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true">

        <LinearLayout
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true">

            <LinearLayout
                android:orientation="horizontal"
                android:layout_width="match_parent"
                android:layout_height="match_parent">

                <TextView
                    android:text="Conexion␣broker:"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:id="@+id/textView23"
                    android:layout_weight="0.52"
                    android:textSize="18sp" />

```

```
44                  <TextView
45                      android:layout_width="wrap_content"
46                      android:layout_height="wrap_content"
47                      android:id="@+id/textView25"
48                      android:layout_weight="1.10"
49                      android:textSize="18sp" />
50              </LinearLayout>
51
52              <LinearLayout
53                  android:orientation="horizontal"
54                  android:layout_width="match_parent"
55                  android:layout_height="match_parent"
56                  android:paddingTop="15dp">
57
58                  <TextView
59                      android:layout_width="wrap_content"
60                      android:layout_height="wrap_content"
61                      android:id="@+id/textView24"
62                      android:layout_weight="1"
63                      android:textSize="14sp" />
64              </LinearLayout>
65
66              <LinearLayout
67                  android:orientation="horizontal"
68                  android:layout_width="wrap_content"
69                  android:layout_height="wrap_content"
70                  android:paddingTop="15dp">
71
72                  <TextView
73                      android:text="Estado␣toldo␣():"
74                      android:layout_width="217dp"
75                      android:layout_height="wrap_content"
76                      android:id="@+id/textView"
77                      android:textSize="14sp" />
78
79                  <TextView
80                      android:layout_width="match_parent"
81                      android:layout_height="match_parent"
82                      android:id="@+id/textView2"
83                      android:textSize="14sp"
84                      android:gravity="center" />
85              </LinearLayout>
86
87              <LinearLayout
88                  android:orientation="horizontal"
89                  android:layout_width="match_parent"
90                  android:layout_height="match_parent"
91                  android:paddingTop="15dp">
92
93                  <TextView
94                      android:text="Cambiar␣estado␣toldo:"
95                      android:layout_width="wrap_content"
96                      android:layout_height="wrap_content"
97                      android:id="@+id/textView27"
98                      android:layout_weight="1" />
99
100                 <TextView
101                     android:layout_width="wrap_content"
102                     android:layout_height="wrap_content"
103                     android:id="@+id/textView28"
```

```
104                 android:layout_weight="1" />
105
106         </LinearLayout>
107
108         <LinearLayout
109             android:orientation="horizontal"
110             android:layout_width="match_parent"
111             android:layout_height="match_parent"
112             android:paddingTop="15dp">
113
114             <Button
115                 android:text="Subir"
116                 android:layout_width="wrap_content"
117                 android:layout_height="wrap_content"
118                 android:id="@+id/button3"
119                 android:layout_weight="1" />
120
121             <Button
122                 android:text="Bajar"
123                 android:layout_width="wrap_content"
124                 android:layout_height="wrap_content"
125                 android:id="@+id/button4"
126                 android:layout_weight="1" />
127         </LinearLayout>
128
129         <LinearLayout
130             android:orientation="horizontal"
131             android:layout_width="match_parent"
132             android:layout_height="match_parent">
133
134             <Button
135                 android:text="Parar"
136                 android:layout_width="wrap_content"
137                 android:layout_height="wrap_content"
138                 android:id="@+id/button5"
139                 android:layout_weight="1" />
140
141             <Button
142                 android:text="Quitar␣modo"
143                 android:layout_width="wrap_content"
144                 android:layout_height="wrap_content"
145                 android:id="@+id/button6"
146                 android:layout_weight="1" />
147         </LinearLayout>
148
149         <LinearLayout
150             android:orientation="horizontal"
151             android:layout_width="wrap_content"
152             android:layout_height="wrap_content"
153             android:paddingTop="15dp">
154
155             <TextView
156                 android:text="Viento␣(km/h):"
157                 android:layout_width="216dp"
158                 android:layout_height="wrap_content"
159                 android:id="@+id/textView3"
160                 android:textSize="14sp" />
161
162             <TextView
163                 android:layout_width="match_parent"
```

```
164                 android:layout_height="match_parent"
165                 android:id="@+id/textView4"
166                 android:textSize="14sp"
167                 android:gravity="center" />
168         </LinearLayout>
169 
170         <LinearLayout
171             android:orientation="horizontal"
172             android:layout_width="wrap_content"
173             android:layout_height="wrap_content"
174             android:paddingTop="15dp">
175 
176             <TextView
177                 android:text="Temperatura␣(C):"
178                 android:layout_width="217dp"
179                 android:layout_height="wrap_content"
180                 android:id="@+id/textView5"
181                 android:textSize="14sp" />
182 
183             <TextView
184                 android:layout_width="match_parent"
185                 android:layout_height="match_parent"
186                 android:id="@+id/textView6"
187                 android:textSize="14sp"
188                 android:gravity="center" />
189         </LinearLayout>
190 
191         <LinearLayout
192             android:orientation="horizontal"
193             android:layout_width="wrap_content"
194             android:layout_height="wrap_content"
195             android:paddingTop="15dp">
196 
197             <TextView
198                 android:text="Humedad(%RH):"
199                 android:layout_width="219dp"
200                 android:layout_height="wrap_content"
201                 android:id="@+id/textView7"
202                 android:textSize="14sp" />
203 
204             <TextView
205                 android:layout_width="match_parent"
206                 android:layout_height="match_parent"
207                 android:id="@+id/textView8"
208                 android:textSize="14sp"
209                 android:gravity="center" />
210         </LinearLayout>
211 
212         <LinearLayout
213             android:orientation="horizontal"
214             android:layout_width="wrap_content"
215             android:layout_height="wrap_content"
216             android:paddingTop="15dp">
217 
218             <TextView
219                 android:text="Luz␣(lux):"
220                 android:layout_width="214dp"
221                 android:layout_height="wrap_content"
222                 android:id="@+id/textView9"
223                 android:textSize="14sp" />
```

```
224
225                 <TextView
226                     android:layout_width="match_parent"
227                     android:layout_height="match_parent"
228                     android:id="@+id/textView10"
229                     android:textSize="14sp"
230                     android:gravity="center" />
231             </LinearLayout>
232
233             <LinearLayout
234                 android:orientation="horizontal"
235                 android:layout_width="match_parent"
236                 android:layout_height="match_parent"
237                 android:paddingTop="15dp">
238
239                 <TextView
240                     android:text="Completa␣los␣siguientes␣datos:"
241                     android:layout_width="wrap_content"
242                     android:layout_height="wrap_content"
243                     android:id="@+id/textView11"
244                     android:layout_weight="1.16"
245                     android:textSize="14sp"
246                     android:gravity="center" />
247             </LinearLayout>
248
249             <LinearLayout
250                 android:orientation="horizontal"
251                 android:layout_width="wrap_content"
252                 android:layout_height="match_parent"
253                 android:paddingTop="15dp">
254
255                 <TextView
256                     android:text="Clase␣del␣toldo␣(1,2␣o␣3):␣"
257                     android:layout_width="227dp"
258                     android:layout_height="30dp"
259                     android:id="@+id/textView12"
260                     android:textSize="14sp" />
261
262                 <EditText
263                     android:layout_width="101dp"
264                     android:layout_height="wrap_content"
265                     android:inputType="number"
266                     android:ems="10"
267                     android:id="@+id/editText"
268                     android:textSize="14sp"
269                     android:gravity="center" />
270
271             </LinearLayout>
272
273             <LinearLayout
274                 android:orientation="horizontal"
275                 android:layout_width="wrap_content"
276                 android:layout_height="wrap_content"
277                 android:paddingTop="15dp">
278
279                 <TextView
280                     android:text="Max.␣temperatura␣subir(C):"
281                     android:layout_width="227dp"
282                     android:layout_height="30dp"
283                     android:id="@+id/textView13"
```

```
284                     android:layout_weight="1" />
285
286                 <EditText
287                     android:layout_width="101dp"
288                     android:layout_height="wrap_content"
289                     android:inputType="number"
290                     android:ems="10"
291                     android:id="@+id/editText2"
292                     android:layout_weight="1"
293                     android:gravity="center"
294                     android:textSize="14sp" />
295             </LinearLayout>
296
297             <LinearLayout
298                 android:orientation="horizontal"
299                 android:layout_width="wrap_content"
300                 android:layout_height="wrap_content"
301                 android:paddingTop="15dp">
302
303                 <TextView
304                     android:text="Min.␣temperatura␣bajar␣(C):"
305                     android:layout_width="227dp"
306                     android:layout_height="30dp"
307                     android:id="@+id/textView14"
308                     android:textSize="14sp" />
309
310                 <EditText
311                     android:layout_width="101dp"
312                     android:layout_height="wrap_content"
313                     android:inputType="number"
314                     android:ems="10"
315                     android:id="@+id/editText3"
316                     android:layout_weight="1"
317                     android:gravity="center"
318                     android:textSize="14sp" />
319             </LinearLayout>
320
321             <LinearLayout
322                 android:orientation="horizontal"
323                 android:layout_width="match_parent"
324                 android:layout_height="match_parent"
325                 android:paddingTop="15dp">
326
327                 <TextView
328                     android:text="Max.␣humedad␣subir␣(%RH):"
329                     android:layout_width="227dp"
330                     android:layout_height="30dp"
331                     android:id="@+id/textView15"
332                     android:textSize="14sp" />
333
334                 <EditText
335                     android:layout_width="101dp"
336                     android:layout_height="wrap_content"
337                     android:inputType="number"
338                     android:ems="10"
339                     android:id="@+id/editText4"
340                     android:textSize="14sp"
341                     android:gravity="center" />
342
343                 <LinearLayout
```

```
344                     android:orientation="horizontal"
345                     android:layout_width="match_parent"
346                     android:layout_height="match_parent"
347                     android:paddingTop="15dp">
348
349                     <TextView
350                         android:text="Max. temperatura subir (C):"
351                         android:layout_width="207dp"
352                         android:layout_height="41dp"
353                         android:id="@+id/textView13"
354                         android:textSize="14sp" />
355
356                     <EditText
357                         android:layout_width="wrap_content"
358                         android:layout_height="wrap_content"
359                         android:ems="10"
360                         android:id="@+id/editText2"
361                         android:layout_weight="1"
362                         android:textSize="14sp"
363                         android:gravity="center"
364                         android:inputType="number" />
365                 </LinearLayout>
366             </LinearLayout>
367
368             <LinearLayout
369                 android:orientation="horizontal"
370                 android:layout_width="match_parent"
371                 android:layout_height="match_parent"
372                 android:paddingTop="15dp">
373
374                 <TextView
375                     android:text="Min. humedad bajar (%RH):"
376                     android:layout_width="227dp"
377                     android:layout_height="30dp"
378                     android:id="@+id/textView16"
379                     android:textSize="14sp" />
380
381                 <EditText
382                     android:layout_width="101dp"
383                     android:layout_height="wrap_content"
384                     android:inputType="number"
385                     android:ems="10"
386                     android:id="@+id/editText5"
387                     android:layout_weight="1"
388                     android:gravity="center"
389                     android:textSize="14sp" />
390             </LinearLayout>
391
392             <LinearLayout
393                 android:orientation="horizontal"
394                 android:layout_width="match_parent"
395                 android:layout_height="match_parent"
396                 android:paddingTop="15dp"
397                 android:weightSum="1">
398
399                 <TextView
400                     android:text="Max. luz subir (lux):"
401                     android:layout_width="227dp"
402                     android:layout_height="30dp"
403                     android:id="@+id/textView17"
```

```
404                 android:textSize="14sp" />
405
406             <EditText
407                 android:layout_width="101dp"
408                 android:layout_height="wrap_content"
409                 android:inputType="number"
410                 android:ems="10"
411                 android:id="@+id/editText6"
412                 android:gravity="center"
413                 android:textSize="14sp" />
414         </LinearLayout>
415
416         <LinearLayout
417             android:orientation="horizontal"
418             android:layout_width="wrap_content"
419             android:layout_height="wrap_content"
420             android:paddingTop="15dp"
421             android:weightSum="1">
422
423             <TextView
424                 android:text="Min.␣luz␣bajar␣(lux):"
425                 android:layout_width="227dp"
426                 android:layout_height="30dp"
427                 android:id="@+id/textView18"
428                 android:textSize="14sp"
429                 android:layout_weight="0.45" />
430
431             <EditText
432                 android:layout_width="101dp"
433                 android:layout_height="wrap_content"
434                 android:inputType="number"
435                 android:ems="10"
436                 android:id="@+id/editText7"
437                 android:textSize="14sp"
438                 android:gravity="center" />
439         </LinearLayout>
440
441         <LinearLayout
442             android:orientation="horizontal"
443             android:layout_width="match_parent"
444             android:layout_height="match_parent"
445             android:paddingTop="15dp">
446
447             <TextView
448                 android:text="Modo␣vacaciones:"
449                 android:layout_width="227dp"
450                 android:layout_height="30dp"
451                 android:id="@+id/textView19"
452                 android:layout_weight="1"
453                 android:textSize="14sp" />
454
455             <TextView
456                 android:layout_width="101dp"
457                 android:layout_height="30dp"
458                 android:id="@+id/textView20"
459                 android:layout_weight="1"
460                 android:textSize="14sp" />
461         </LinearLayout>
462
463         <LinearLayout
```

```
464                 android:orientation="horizontal"
465                 android:layout_width="match_parent"
466                 android:layout_height="match_parent"
467                 android:paddingTop="15dp">
468
469                 <Button
470                     android:text="Activar"
471                     android:layout_width="wrap_content"
472                     android:layout_height="wrap_content"
473                     android:id="@+id/button"
474                     android:layout_weight="1"
475                     android:textSize="14sp" />
476
477                 <Button
478                     android:text="Desactivar"
479                     android:layout_width="wrap_content"
480                     android:layout_height="wrap_content"
481                     android:id="@+id/button2"
482                     android:layout_weight="1"
483                     android:textSize="14sp" />
484             </LinearLayout>
485
486             <LinearLayout
487                 android:orientation="horizontal"
488                 android:layout_width="match_parent"
489                 android:layout_height="match_parent"
490                 android:paddingTop="15dp">
491
492                 <TextView
493                     android:text="Hora1:"
494                     android:layout_width="227dp"
495                     android:layout_height="30dp"
496                     android:id="@+id/textView21"
497                     android:textSize="14sp" />
498
499                 <EditText
500                     android:layout_width="101dp"
501                     android:layout_height="wrap_content"
502                     android:inputType="number"
503                     android:ems="10"
504                     android:id="@+id/editText8"
505                     android:layout_weight="1"
506                     android:textSize="14sp"
507                     android:gravity="center" />
508             </LinearLayout>
509
510             <LinearLayout
511                 android:orientation="horizontal"
512                 android:layout_width="match_parent"
513                 android:layout_height="match_parent"
514                 android:paddingTop="15dp">
515
516                 <TextView
517                     android:text="Hora2:"
518                     android:layout_width="227dp"
519                     android:layout_height="30dp"
520                     android:id="@+id/textView22"
521                     android:textSize="14sp" />
522
523                 <EditText
```

```
                    android:layout_width="101dp"
                    android:layout_height="wrap_content"
                    android:inputType="number"
                    android:ems="10"
                    android:id="@+id/editText9"
                    android:layout_weight="1"
                    android:textSize="14sp"
                    android:gravity="center" />
            </LinearLayout>

        </LinearLayout>
    </ScrollView>

</RelativeLayout>
```

Bibliografía

[1] El Internet de las Cosas, En un mundo conectado de objetos inteligentes. Fundación de la Innovación Bankinter.

[2] Artículo: Wireless Communications Standars: The battle for the IoT.

[3] Artículo: MQTT and CoAP: Underlying Protocols for the IoT.

[4] 5 Things to Know About MQTT- The protocol for Internet of Things. Rahul Gupta. https://www.ibm.com/developerworks/community/blogs/5things/entry/5_things_to_know_about_mqtt_

[5] Lógica Difusa, Una introducción práctica. Carlos González Morcillo.

[6] Página oficial de Mosquitto: https://mosquitto.org/

[7] Página web de desarrollo de Android, https://developer.android.com/index.html

[8] Página oficial Arduino: https://www.arduino.cc/

[9] Espressif systems: Esp8266. https://espressif.com/en/products/hardware/esp8266ex/overview

[10] HiveMQ, entreprise MQTT broker. http://www.hivemq.com/