UNIVERSIDAD MAYOR DE SAN ANDRÉS

FACULTAD DE TECNOLOGÍA

CARRERA DE ELECTRONICA Y TELECOMUNICACIONES

" DISEÑO DE UNA APLICACIÓN PARA LA GESTIÓN DE UNA ESTACIÓN METEOROLÓGICA MEDIANTE PROTOCOLO DE TRANSPORTE DE TELEMETRÍA POR MENSAJE DE COLAS (MQTT) SOBRE MODULO DE RADIO FRECUENCIA ESP32 "

Trabajo de Aplicación – Examen de Grado presentado para obtener el Grado de Licenciatura

POR: TITO BELMONTE RIOS

La Paz- Bolivia

JUNIO – 2022

UNIVERSIDAD MAYOR DE SAN ANDRÉS
FACULTAD DE TECNOLOGÍA
CARRERA ELECTRÓNICA Y TELECOMUNICACIONES

Trabajo de aplicación – Examen de Grado

" DISEÑO DE UNA APLICACIÓN PARA LA GESTIÓN DE UNA ESTACIÓN METEOROLÓGICA MEDIANTE PROTOCOLO DE TRANSPORTE DE TELEMETRÍA POR MENSAJE DE COLAS (MQTT) SOBRE MODULO DE RADIO FRECUENCIA ESP32 "

Presentado por: TITO BELMONTE RIOS

Para optar del grado académico de Licenciado en Electrónica y Telecomunicaciones

Nota numeral: …….

Nota Literal: …….

Ha sido …….

Lic. Julia Torrez Soria
Directora de la Carrera de Electrónica y Telecomunicaciones

Tribunal: M. Sc. Javier Nicolas Yujra Tarqui

Tribunal: Ing. Juan Alberto Aguilera Ríos

Tribunal: Lic. Juan Carlos Gutierrez Yujra

DEDICATORIA

Quiero dedicar el presente proyecto a mis padres Alberto Belmonte y Elena Rios, gracias a quienes tengo el honor de realizar este documento, quienes me incentivaron, apoyaron y siempre animaron para que siga el arduo camino de la superación a través del estudio continuo, en tiempos difíciles hasta llegar a la meta, el cual con su cariño han logrado hacer de mí una persona de bien.

AGRADECIMIENTO

A mi alma máter, la UMSA, donde la calidad y prestigio se inculcaron desde el primer día en sus pasillos.

A la Facultad de Tecnología y sobre todo a los docentes de la carrera de Electrónica y Telecomunicaciones, por su ayuda, consejos, paciencia y todo el conocimiento y experiencia transmitido, por su valioso tiempo mismos que hicieron posible desarrollo de este proyecto.

INDICE

ÍNDICE DETABLAS

ÍNDICE DE FIGURAS

RESUMEN

En el presente proyecto se realiza el diseño y la implementación de una estación meteorológica en tiempo real, mismo que pudiera ser implementado bajo una red de manera local, o en un entorno de red con acceso a internet.

Primeramente, expondremos todo lo referido al sistema, el concepto teórico; desde los módulos a usar, programación, instalación y las respectivas pruebas. El proyecto pretende, con los datos obtenidos, generar una base de datos para la monitorización de los parámetros de presión atmosférica, humedad relativa, temperatura, velocidad del viento y otros para tener un referencia del punto de monitorización específico, todo esto basado en IoT, con una conexión a una red local para tener a la mano, a través de un Smartphone o una pc el acceso a los datos de medición, mismos que estarán en la nube de internet, y con ellos poder tomar decisiones respecto a donde se desee implementar.

Como protocolo de comunicación entre los dispositivos y el servidor utilizaremos el MQTT que ha ganado gran popularidad por su simpleza y mínimo coste computacional. El cerebro de nuestra aplicación estará diseñado en Node Red el cual permite, mediante una interfaz gráfica muy sencilla, programar toda la lógica de nuestro proyecto. Este subsistema estará ubicado en la nube de internet, donde puedes ver los datos en tiempo real.

Se detallarán también los periféricos; el módulo ESP32, sensores, pantalla lcd, cables, conectores y otros que nos permitirán el manejo estructural y de información para que todo el sistema tenga las mejores condiciones de funcionamiento. Será necesario conocer el manejo del software utilizado de estos equipos si se quiere tener un adecuado manejo y el uso de todas las ventajas que ofrecen los equipos, además del mantenimiento y programación de los mismos.

Pero también tocaremos un punto muy importante dentro de este proyecto que es la de conocer el funcionamiento interno y no solo el manejo general, por tanto, al implementar este sistema, la verificación del equipo, modos de conexión, usos y tipo de funcionamiento, serán desglosadas a lo largo del tema.

CAPÍTULO I

MARCO REFERENCIAL

1.1 INTRODUCCION

El desarrollo de este proyecto surge con el afán de llevar a la práctica los conocimientos adquiridos durante la carrera universitaria y de materializarlo en una implementación útil que ayudara a mejorar la vida de las personas. Después de un extenso estudio, la tecnología del IoT (*Internet of Things*), fue sin dudas el candidato número uno que motivó a la realización de este trabajo.

La infinidad de implementaciones que está teniendo esta tecnología, demuestran que el IoT ya no es cosa del futuro, sino cada vez más del presente. Utilizarlo en todo tipo de productos y servicios es, por tanto, una forma de ser más eficientes, mejorar las comunicaciones y controlar cada fase y momento de la producción. Permite generar datos muy interesantes para determinar las evoluciones de los productos y servicios y poder, de este modo, mejorarlos o adaptarlos en el futuro. Estas soluciones no sólo las vemos en los grandes sectores industriales, sino que cada día es más frecuente su utilización en entornos domésticos.

El flujo de información de electrodomésticos y dispositivos inteligentes, diseñados para facilitarnos la vida en nuestros hogares, con la capacidad añadida de intercambiar información a Internet es lo que conocemos popularmente como domótica.

Precisamente, basados en la idea de la domótica, se desarrolla el contexto de este trabajo. Se pretende el desarrollo de una aplicación IoT que ofrezca una solución asequible y sencilla, que permita obtener información en tiempo real de diversos sensores instalados en un punto en específico, como control de área o ambiente.

1.2 PLANTEAMIENTO DEL PROBLEMA

- Falta de acceso a datos sobre condiciones climáticos en un punto específico, y no en áreas geográficas demasiado grande con respectivas mediciones.

- Por qué no se pueden visualizar los datos del ambiente en cual se desea controlar las condiciones necesarias de humedad, temperatura y otros.
- Cuando deseamos verificar las condiciones de un ambiente, o de un área en específico, muchas veces simplemente se deja a la experiencia y no a datos reales.
- Controlar varios puntos al mismo tiempo supone personal o sufrir posibles problemas a la hora de tomar decisiones sin tener datos disponibles en tiempo real.

1.3 JUSTIFICACIÓN

Debido a la necesidad de tener los datos del clima en uno o varios puntos específicos, es decir, a las condiciones meteorológicas en tiempo real para la toma de decisiones se plantea las siguientes justificaciones:

1.3.1 JUSTIFICACIÓN TECNOLÓGICA

La tecnología para la obtención de ciertos parámetros que ayudan a tener una mejor decisión sobre condiciones reales de un entorno más específico que el general, se hacen cada vez más necesario y por tanto contar con procesadores cada vez más potentes y sencillos de aplicar, garantizan seguridad y confiabilidad para predios preestablecidos y todo tipo de aplicaciones de control.

Pero no solo los sensores o módulos, que serán las más importantes, sino también todo el software y los periféricos que se utilizara serán los más convenientes por su facilidad, de programación e implementación.

1.3.2 JUSTIFICACIÓN SOCIAL

Toda aquella persona, podrá realizar un mejor control de cualquier ambiente, beneficiándose así mismos, porque tendrá a la mano, toda la información necearía para

realizar los ajustes que se crean convenientes, para mejorar su entorno. El control permanente y el monitoreo de las instalaciones hacen de este sistema muy necesaria.

Además de que los puntos de medición donde serán instaladas, serán sujetos de control permanente. Disminuyendo el riesgo de no tener estos datos a la mano.

Este proyecto permitirá tener un continuo monitoreo para la supervisión del o los ambientes, como de su control.

1.3.3 JUSTIFICACIÓN ECONÓMICA

La inversión que se realiza en los equipos que se necesitarán para el diseño de este sistema de control en base a la estación meteorológica es por demás necesaria; tanto para aquel que desee implementarlo como para los que requieran datos que ayuden en el mejor control de los mismos. Estos equipos garantizaran en un alto porcentaje la confiabilidad de los datos, además de reducir los riesgos del entorno, ya que se encarga del control del área, de una manera más precisa y en tiempo real.

1.3.4 JUSTIFICACIÓN ACADÉMICA

Para la instalación del sistema, el diseño de las interconexiones y protocolos de transferencia y recepción de datos, los conocimientos previos durante la etapa de formación fueron de mucha utilidad; materias como telecomunicaciones, electrónica, fibra óptica, la práctica en laboratorios y muchas otras hicieron que el desarrollo del proyecto sea mucho más adecuado, tomando todas las medidas necesarias para que todo el sistema se realice de manera eficiente.

1.4 OBJETIVOS

1.4.1 OBJETIVO GENERAL

Implementar el diseño de una estación meteorológica de propósito general a fin, de reducir los riesgos posibles al no tener datos del clima en tiempo real, además de que sea accesible para todos, además de desarrollar una aplicación IoT completa para la gestión de una estación meteorológica y el diseño de una aplicación con Node-RED, que permita controlar varios sensores mediante el estandarizado protocolo de comunicación MQTT.

1.4.2 OBJETIVOS ESPECIFICOS

Para solucionar las necesidades de obtención de datos del entorno se plantean los siguientes objetivos específicos.

- Dimensionar los equipos necesarios para la implementación de la estación meteorológica de tal forma que cumpla con los requisitos exigidos.
- Instalar la estación en un entorno a controlar de manera estratégica de modo que cumpla con las funciones de manera eficiente.
- Investigar sobre los sensores más utilizados en el mercado de la domótica, sus características y funcionalidades, con el objetivo de que los utilizados cumplan con los requisitos del proyecto.
- Relacionarse con las características de funcionamiento y pinout de los distintos sensores escogidos: DTH11, BMP180, YL-38 y el modulo adaptador serial I2C junto a la pantalla LCD 16x2, necesarios para programar el ESP32 DEVKITV1.
- Comprobar el buen funcionamiento de los equipos, realizando todas las pruebas necesarias para que la estación no tenga inconveniente alguno.

1.5 DELIMITACIONES

1.5.1 DELIMITACION TEMATICA

El diseño de la aplicación IoT para la gestión de la estación meteorológica, será de gran ayuda, en ambientes, espacios o áreas donde el control de los mismos sea de importancia tanto para el control del mismo como para el monitoreo de lugares donde se desea tener una referencia real, cumpliendo así los requisitos de monitoreo del o los usuarios.

1.5.2 DELIMITACION ESPACIAL

La estación meteorológica al ser un sistema capaz de brindar información almacenada en los servidores de la nube de internet, de un modo accesible podrá ser implementado en cualquier ambiente o área, tales como; espacios de ambientes controlados, áreas de agricultura referente al cultivo de invernaderos, ambientes hospitalarios, incluso usarlos de manera doméstica o en cualquier área del cual se necesite tener loa datos para luego poder controlarlos.

1.5.3 DELIMITACION TEMPORAL

Para el diseño de la de la aplicación IoT de la estación meteorológica, el tiempo a considerar estará en consecuencia ligada al modo de implementación del espacio o área que se desea controlar, lo necesariamente útil: la aplicación IoT, el uso del protocolo MQTT el uso de los componentes, los sensores y la conexión a red disponibles para el montaje dependerá del usuario final; por tanto será necesario hacer un estudio previo del lugar específico que se desea controlar de manera que no genere errores graves al momento de la toma de datos.

Este tiempo será el suficiente para diseñar o rediseñar de forma adecuada todo el sistema, adquirir y revisar los equipos necesarios, realizar el estructurado del cableado y todas las conexiones, el manejo adecuado de los equipos, un periodo de prueba y la puesta en marcha del sistema.

CAPÍTULO II

MARCO TEÓRICO CONCEPTUAL

2.1 PROTOCOLO MQTT

2.1.1 INTRODUCCION

MQTT son las siglas de Message Queuing Telemetry Transport; Traducido seria Transporte de Telemetría por mensaje de Colas. Se trata de un protocolo de mensajería ligero para usar en casos de clientes que necesitan una huella de código pequeña, que están conectados a redes no fiables o con recursos limitados en cuanto al ancho de banda. Se utiliza principalmente para comunicaciones de máquina a máquina (M2M) o conexiones del tipo de Internet de las cosas.

El protocolo MQTT se ha convertido en uno de los principales pilares del IoT por su sencillez y ligereza. Ambos son condicionantes importantes dado que los dispositivos de IoT, a menudo, tienen limitaciones de potencia, consumo, y ancho de banda. Además, se trata de un protocolo abierto, lo que da bastantes ventajas para comunicaciones en red ya que se ejecuta, generalmente, sobre el protocolo TCP/IP.

Es un protocolo simple y muy ligero para la transmisión de mensajes cortos de telemetría y de control, desde/hacía una red de sensores/actuadores, que tenga limitaciones evidentes en cuanto al consumo, velocidad de transmisión y procesamiento.

2.1.2 PROTOCOLO MQTT DENTRO DEL MODELO OSI

El protocolo MQTT se localiza en las capas superiores de OSI (Figura 1), y normalmente se apoya en TCP/IP. Esto significa que los participantes de una aplicación MQTT deben tener una pila TCP/IP.

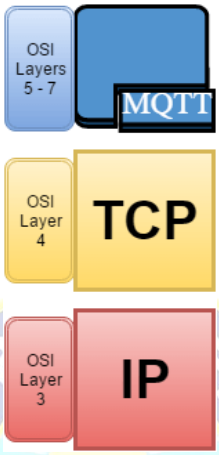


Figura 1 MQTT dentro del modelo OSI (Fuete: https://www.indelmar.com/wp-content/uploads/2017/11/MQTT-OSI.png)

2.1.3 HISTORIA

El protocolo MQTT fue creado originalmente por el Dr. Andy Stanford-Clark y Arlen Nipper en 1999. El propósito original de este método de comunicación era permitir que los dispositivos de monitoreo utilizados en la industria del petróleo y el gas enviaran sus datos a servidores remotos. En muchos casos, estos dispositivos de monitoreo se empleaban en ubicaciones remotas donde establecer cualquier tipo de línea fija, conexión por cable o enlace de transmisión de radio sería ya no simplemente difícil, sino incluso imposible. En ese momento, la única opción para encarar tales situaciones eran las comunicaciones por satélite, tremendamente costosas y que se facturaban en función de la cantidad de datos utilizada. Con miles de sensores en el campo, la industria necesitaba una forma de comunicación que pudiera proporcionar datos de manera suficientemente fiable para su uso, mientras empleaba un ancho de banda mínimo.

Aunque inicialmente era un formato propiedad de IBM, no es una invención de IBM, pero sí fue esta empresa la que ha contribuido en su impulso definitivo, en 2010 fue liberado y pasó a ser un estándar en 2014 según la OASIS

2.1.4 ARQUITECTURA DE MQTT

MQTT se ejecuta sobre TCP/IP utilizando una topología PUSH/SUBSCRIBE (Figura. 3). En la arquitectura MQTT, existen dos tipos de sistemas: clientes y brókeres. Un bróker es el servidor con el que se comunican los clientes: recibe comunicaciones de unos y se las envía a otros. Los clientes no se comunican directamente entre sí, sino que se conectan con el bróker. Cada cliente puede ser un editor, un suscriptor o ambos.

MQTT es un protocolo controlado por eventos, donde no hay transmisión de datos periódica o continua. Así se mantiene el volumen de transmisión al mínimo (Figura 2). Un cliente sólo publica cuando hay información para enviar, y un bróker sólo envía información a los suscriptores cuando llegan nuevos datos.

Para filtrar los mensajes que son enviados a cada cliente los mensajes se disponen en *topics* organizados jerárquicamente. Un cliente puede publicar un mensaje en un determinado *topic*. Otros clientes pueden suscribirse a este *topic*, y el *bróker* le hará llegar los mensajes suscritos. Los clientes inician una conexión TCP/IP con el *bróker*, el cual mantiene un registro de los clientes conectados. Esta conexión se mantiene abierta hasta que el cliente la finaliza. Por defecto, MQTT emplea el puerto 1883.

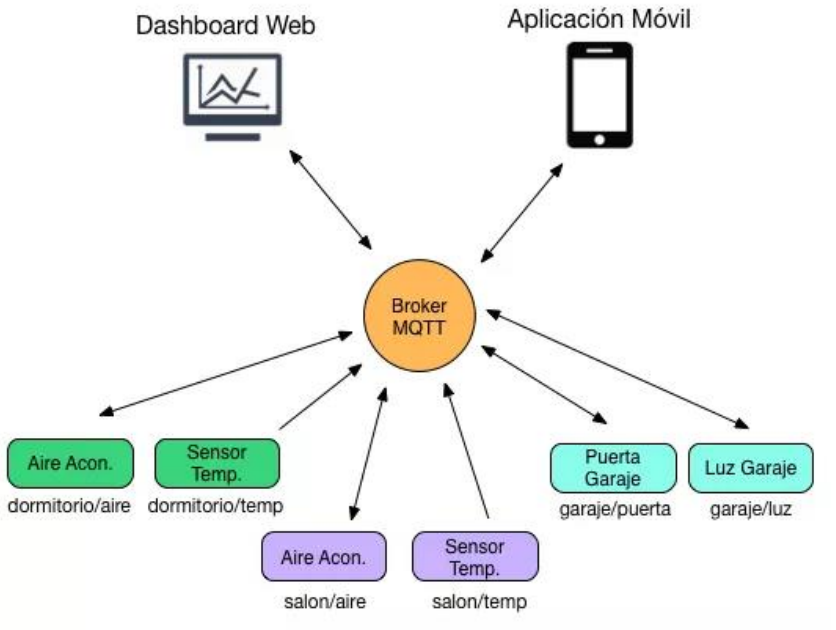


Figura 2 Arquitectura MQTT (Fuente: https://lh6.googleusercontent.com/)

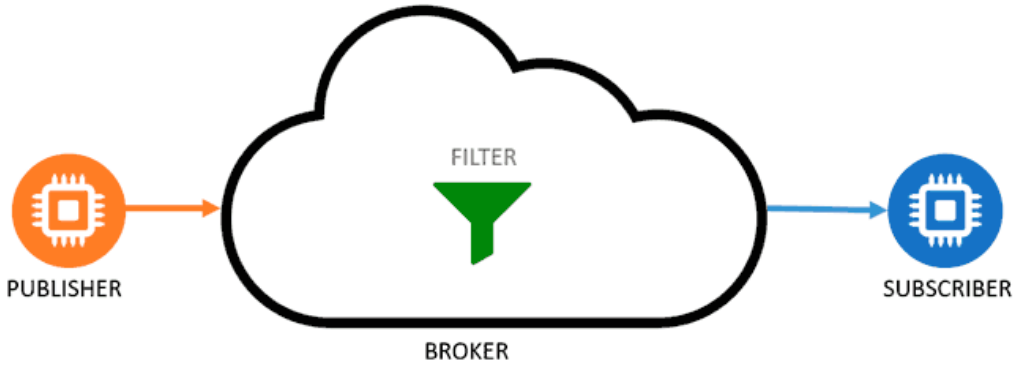


Figura 3 PUB/SUB a través del BROKER (Fuente:https://www.luisllamas.es/wp-content/uploads/2019/02/protocolos-iot-pubsub.png)

2.1.5 ARQUITECTURA DE LOS MENSAJES

Otra forma en que MQTT minimiza sus transmisiones es con un tamaño de mensaje pequeño y bien definido. Cada mensaje tiene un encabezado fijo de apenas 2 bytes. Se puede utilizar un encabezado opcional, pero eso incrementa el tamaño del mensaje. La carga útil del mensaje está limitada a únicamente 256 MB. Se distinguen dos estructuras:

- **Message Queue**: servicio de mensajería donde se genera una sola cola de mensajes para todos los clientes que inician una subscripción en el bróker. Éste último mantendrá los mensajes almacenados hasta que son entregados al cliente. Si el cliente o destinatario no estuviera conectado, se mantiene hasta que se conecta. Este tipo de servicios son como los usados en apps de mensajería instantánea como Telegram, Whatsapp, Messenger, etc.
- **Message Service**: es otro servicio en el que el bróker envía los menajes al cliente destinatario conectado, filtrando por el tipo de mensaje. Si el cliente o dispositivo receptor está desconectado, entonces se pierden los mensajes (aunque puede tener algún sistema de registro).

Cada mensaje consta de 3 partes: cabecera fija, cabecera variable y el contenido (Figura.4)

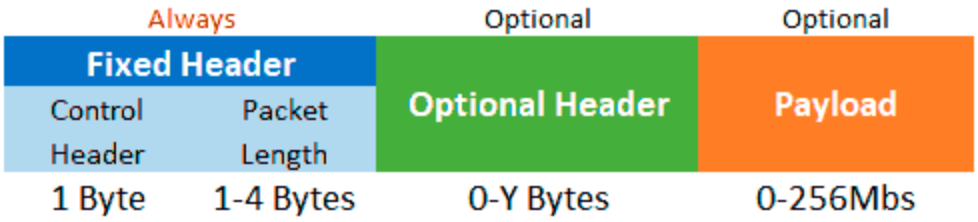


Figura 4 Estructura del mensaje en MQTT (Fuente: https://www.luisllamas.es/wp-content/uploads/2019/04/mqtt-message.png)

Para ello el cliente envía un mensaje CONNECT que contiene información necesaria (nombre de usuario, contraseña, client-id…). El bróker responde con un mensaje CONNACK, que contiene el resultado de la conexión (aceptada, rechazada, etc.) como se ve en la figura.5.

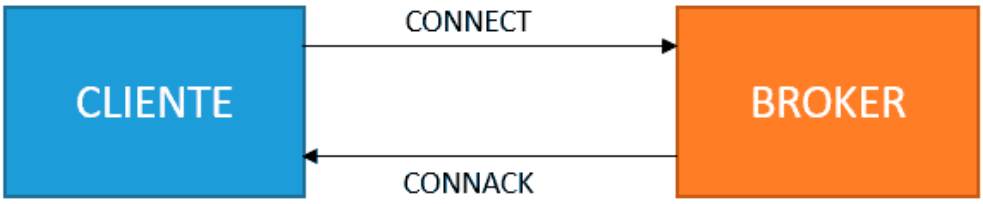


Figura 5 Conexión cliente-bróker (Fuente: https://www.luisllamas.es/wp-content/uploads/2019/04/mqtt-connect.png)

Para enviar los mensajes el cliente emplea mensajes PUBLISH (Figura 6), que contienen el topic y el payload

***Atributos del mensaje PUBLISH*:**

- ***topicName:*** Topic o tema de los mensajes**.** Tiene estructura jerárquica o ramificada separada con el signo "/". Una suscripción puede ser a un tema explícito o puede incluir comodines. Hay dos comodines disponibles: **+ (**para un solo nivel de jerarquía**)** o **# (**para todos los niveles restantes de jerarquía**)**. Los temas que comienzan con un símbolo $ están reservados para las estadísticas internas del bróker MQTT. A modo de seguridad, los tópicos deben de ser lo más personalizados posible de manera que no se produzcan conflictos.
- ***retainFlag:*** Este indicador define si el intermediario guarda el mensaje como el último valor válido conocido para un tema específico.
- ***Payload:*** contenido real del mensaje.
- ***packetId:*** identifica de forma exclusiva un mensaje a medida que fluye entre el cliente y el intermediario.
- ***dupFlag:*** indica si el mensaje es un duplicado y se reenvió porque el destinatario (cliente o bróker) no reconoció el mensaje original.

Figura 6 Publicación al bróker (Fuente: https://www.luisllamas.es/wp-content/uploads/2019/04/mqtt-suscribe.png)

- ***QoS:*** MQTT define tres niveles de calidad de servicio (QoS): 0,1 y 2. La calidad del servicio define la robustez con la que el bróker intentará garantizar que se reciba un mensaje. Si el cliente suscriptor define una QoS más baja que el cliente de publicación, el intermediario transmite el mensaje con la menor calidad de

servicio. Los niveles más altos de QoS son más confiables, pero implican una latencia más alta y tienen mayores requisitos de ancho de banda.

- ***QoS 0:*** ofrece la cantidad mínima de transmisión de datos. Con este nivel, cada mensaje se entrega a un suscriptor una vez, sin confirmación, por lo que no hay forma de saber si los suscriptores recibieron el mensaje. Este método a veces se denomina “lanzar y olvidar” o “una entrega como máximo”. Debido a que este nivel asume que la entrega está completa, los mensajes no se almacenan para entregarlos a los clientes desconectados que luego se vuelven a conectar.
- ***QoS 1****:* el bróker intenta entregar el mensaje y, luego, espera una respuesta de confirmación del suscriptor. Si no se recibe una confirmación dentro de un período de tiempo especificado, el mensaje se envía de nuevo. Usando este método, el suscriptor puede recibir el mensaje más de una vez si el bróker no recibe el acuse de recibo del suscriptor a tiempo. Esto se suele denominar “entregado al menos una vez”.
- ***QoS 2:*** el cliente y el bróker utilizan un protocolo de enlace de cuatro pasos para garantizar no sólo que el mensaje se reciba, sino que lo haga una única vez. También se conoce como “entrega exactamente una vez”.

QoS 0 puede ser la mejor opción para situaciones donde las comunicaciones son fiables pero limitadas. En aquellos casos en que las comunicaciones no sean fiables, pero donde las conexiones tampoco gozan de recursos limitados, QoS 2 sería la mejor opción. QoS 1 proporciona una solución a caballo entre ambos mundos, pero requiere que la aplicación que recibe los datos sepa cómo gestionar los duplicados.

Tanto en QoS 1 como en QoS 2, los mensajes se guardan o se ponen en cola para los clientes que están desconectados y que tienen una sesión persistente establecida. Estos mensajes se reenvían (de acuerdo con el nivel de QoS apropiado) una vez que el cliente vuelve a conectarse.

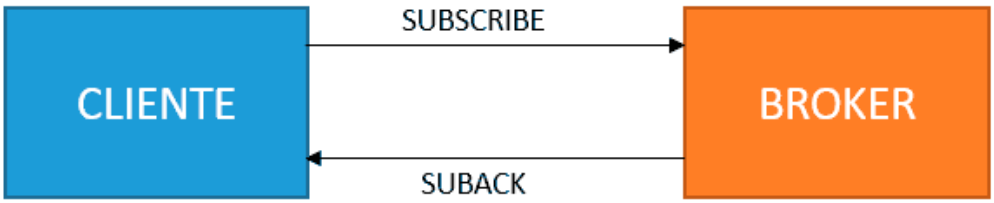


Figura 7 Suscripción al bróker (Fuente: https://www.luisllamas.es/wp-content/uploads/2019/04/mqtt-publish.png)

Para suscribirse y desuscribirse se emplean mensajes SUBSCRIBE y UNSUSCRIBE, que el servidor responde con SUBACK y UNSUBACK.

Para recibir mensajes sobre temas de interés, el cliente envía un mensaje SUBSCRIBE (Figura 7) al agente MQTT. Este mensaje de suscripción es muy simple y cuenta con dos atributos:

- ***packetId:*** que identifica a dicho paquete.
- ***listSubscriptions:*** Un mensaje SUBSCRIBE puede contener múltiples suscripciones para un cliente. Cada suscripción se compone de un tema y un nivel de QoS.

Para confirmar cada suscripción, el agente envía un mensaje de confirmación SUBACK al cliente.

- ***packetId:*** identifica el mensaje SUBACK. Este ID coincide con el del mensaje SUBSCRIBE.
- ***returnCode #:*** El intermediario envía un código de retorno para cada par tema y QoS que recibe en el mensaje SUBSCRIBE.

El mensaje UNSUBSCRIBE elimina las suscripciones existentes de un cliente en el bróker. Este comando es similar al mensaje SUBSCRIBE. Contiene un identificador de

paquete (***packetId)*** y una lista de temas (***topic #***) de los cuales quiere darse de baja independientemente de la QoS.

Para confirmar la cancelación de la suscripción, el bróker envía un mensaje de confirmación UNSUBACK al cliente. Este mensaje contiene solo el identificador de paquete (***packetId***) que coincide con el del mensaje UNSUBCRIBE.

El paquete de DESCONEXIÓN es el paquete de control final enviado desde el cliente al servidor. Indica que el cliente se está desconectando limpiamente. Las características principales de este paquete DISCONNECT es que no tiene payload y no tiene encabezado variable.

Después que el bróker envía un paquete DISCONNECT al cliente:

- Debe cerrar la conexión de red.
- No debe enviar más paquetes de control en esa conexión de red.

Cuando el bróker recibe un paquete DISCONNECT del cliente:

- Debe descartar cualquier paquete ***Will Message*** asociado con la conexión actual.
- Debe cerrar la conexión de red si el cliente aún no lo ha hecho.

Por otro lado, para asegurar que la conexión está activa los clientes mandan periódicamente un mensaje PINGREQ que es respondido por el servidor con un PINGRESP. Finalmente, el cliente se desconecta enviando un mensaje de DISCONNECT.

2.1.6 RETENCIÓN DE MENSAJES

Un mensaje retenido es un mensaje MQTT normal con el indicador retenido establecido en verdadero. El bróker almacena el último mensaje retenido y la QoS correspondiente para ese tema. Esto significa que el bróker mantendrá el mensaje incluso después de enviarlo a todos los suscriptores actuales. Si se realiza una nueva suscripción que coincide con el topic del mensaje retenido, el mensaje se enviará al cliente. Esto es útil si un tema

solo se actualiza con poca frecuencia, porque permite que un cliente recién suscrito no tenga que esperar mucho tiempo para recibir una actualización.

2.1.7 SEGURIDAD EN EL PROTOCOLO MQTT

La seguridad en MQTT se divide en diferentes capas, el nivel de red, el nivel de transporte y el nivel de aplicación.

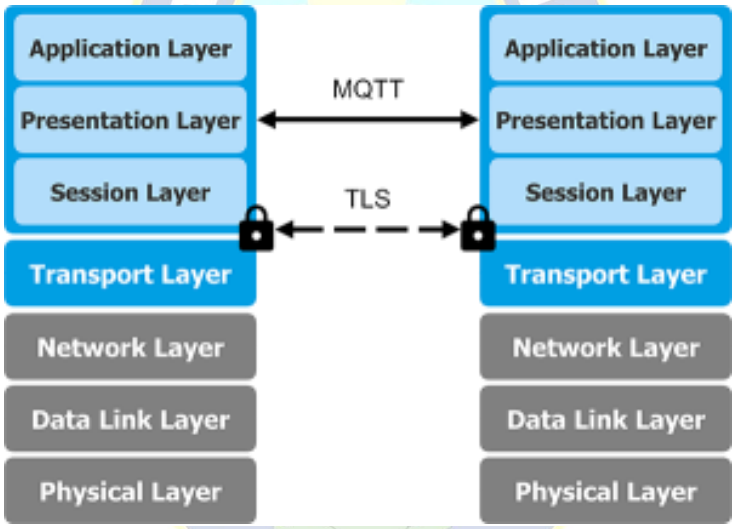


Figura 8 Seguridad MQTT (Fuente: https://www.wut.de/pics/collage/e-577ww-30-grww-000.png)

2.1.7.1 AUTENTICACIÓN Y CODIFICACIÓN

MQTT utiliza la autenticación de los clientes por nombre de usuario y contraseña. De ese modo se puede asignar autorizaciones en el lado del intermediario y garantizar, por ejemplo, que solo el sensor de temperatura local pueda publicar mensajes con el tópico f2/lugar/temp. El nombre de usuario y la contraseña se transfieren al establecer la conexión y se transmiten sin codificar. Por esa razón, conviene codificar toda comunicación por MQTT desde el principio. El protocolo se encuentra en las capas

superiores del modelo OSI, por lo que es muy sencillo realizar la codificación con TLS, siempre y cuando el terminal respectivo disponga de los recursos

2.1.7.2 NIVEL DE RED

Una forma de proporcionar una conexión segura y confiable es usar una red físicamente segura o VPN para toda comunicación entre clientes y servidores. Esta solución es adecuada para aplicaciones de puerta de enlace donde está conectada a dispositivos por un lado y con el agente a través de VPN por el otro.

2.1.7.3 NIVEL DE TRANSPORTE

Transport Layer Security (TLS) y Secure Sockets Layer (SSL) proporcionan un canal de comunicación seguro entre un cliente y un servidor (Figura 8). Este método es una forma segura y comprobada de garantizar que los datos no se puedan leer durante la transmisión. Con Transport Layer Security (TLS), la validación exitosa de un certificado de cliente se utiliza para autenticar al cliente en el servidor. El puerto 8883 está reservado exclusivamente para MQTT sobre TLS, para cifrar toda la comunicación.

2.1.7.4 NIVEL DE APLICACIÓN

En el nivel de transporte, la comunicación se cifra y las identidades se autentican. El protocolo MQTT (Figura 9) proporciona un identificador de cliente y credenciales de nombre de usuario / contraseña para autenticar dispositivos en el nivel de la aplicación. Estas propiedades son proporcionadas por el propio protocolo. La autorización o el control de lo que cada dispositivo puede hacer está definido por la implementación específica del bróker. Además, es posible utilizar el cifrado de carga útil (*payload*) en el nivel de la aplicación. Esto asegura que la información transmitida vaya cifrada sin la necesidad de un cifrado de transporte completo.

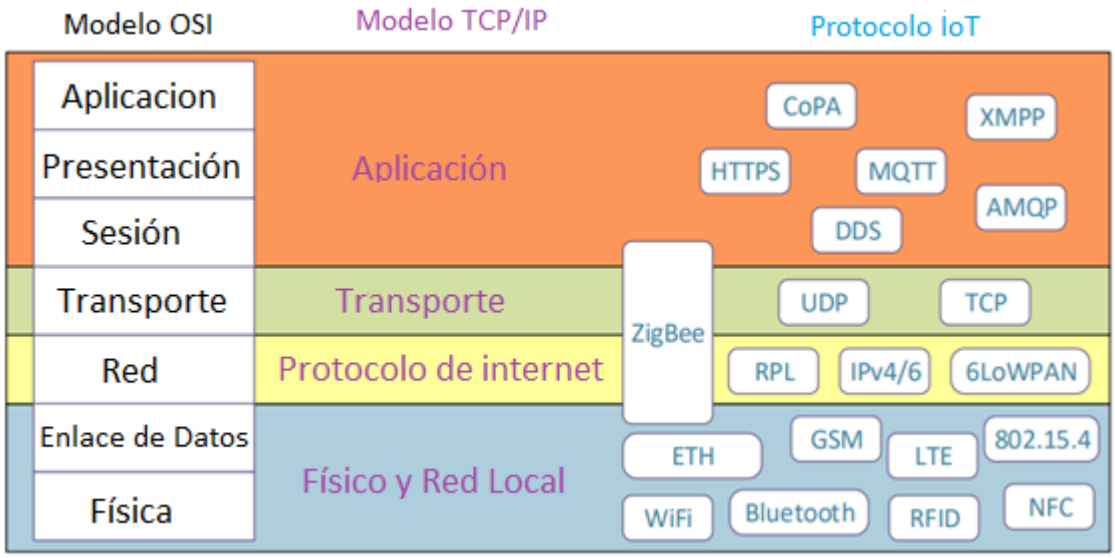


Figura 9 Comparación entre Protocolos de Comunicación (Fuente: https://www.gotoiot.com/pages/articles/iot_protocols_intro/images/image12.png)

2.2 MODULO ESP32-WROOM-32

El ESP-WROOM-32 (Figura 10) es un potente módulo que integra WiFi y Bluetooth, ideal para desarrollar productos de IoT. La integración de Bluetooth, Bluetooth LE y WiFi permite una amplia gama de aplicaciones, el uso de WiFi permite una comunicación de mediano alcance y conectarse a una red LAN y a través de un Modem Router con conexión a Internet, mientras que el Bluetooth nos permite conectarse directamente a otro dispositivo como un celular. El ESP32 es capaz de funcionar de forma fiable en entornos industriales, con una temperatura de funcionamiento que oscila entre -40 °C y +125 °C. Alimentado por circuitos de calibración avanzados, ESP32 puede eliminar dinámicamente las imperfecciones del circuito externo y adaptarse a los cambios en las condiciones externas.

Figura 10 ESP32-WROOM-32 (Fuente: https://media.naylampmechatronics.com/1510-superlarge_default/nodemcu-32-30-pin-esp32-wifi.jpg)

La corriente de reposo del chip ESP32 es inferior a 5 μA, por lo que es adecuado para aplicaciones de electrónica portátiles con batería. En el núcleo de este módulo está el SoC ESP32-D0WDQ6. El chip integrado está diseñado para ser escalable y adaptado. Hay dos núcleos de CPU que se pueden controlar individualmente, y la frecuencia del reloj es ajustable de 80 MHz a 240 MHz. El usuario también puede apagar el CPU y utilizar el co-procesador de baja potencia para supervisar constantemente los periféricos para detectar cambios de estado.

ESP32 integra un amplio conjunto de periféricos como sensores táctiles capacitivos, sensores Hall, amplificadores de bajo nivel de ruido, interfaz para SD, Ethernet, SPI, UART, I2S e I2C. Para flashear el chip es necesario utilizar un módulo conversor USB a serial TTL como el Módulo CP2102.

El módulo ESP-WROOM-32 trabaja a 3.3V en alimentación y GPIO por lo que ***NO se debe alimentar con 5V***. Se recomienda colocar un capacitor de 100uF en paralelo con la fuente de alimentación para filtrar los picos de corriente. Los pines de entradas/salidas (GPIO) trabajan a 3.3V por lo que para la conexión a sistemas de 5V es necesario utilizar conversores de nivel como: Conversor de nivel 3.3-5V 4CH o Conversor de nivel bidireccional 8CH - TXS0108E.

El SoC (System On a Chip) ESP32, integra un potente microcontrolador con arquitectura de 32 bits, conectividad WiFi y Bluetooth. El SoM (System on Module) ESP-WROOM-32 fabricado por Espressif integra en un módulo el SoC ESP32, memoria FLASH, cristal oscilador y antena WiFi en PCB.

La plataforma ESP32 permite el desarrollo de aplicaciones en diferentes lenguajes de programación, *frameworks*, librerías y recursos diversos. Los más comunes a elegir son: Arduino (en lenguaje C++), Esp-idf (Espressif IoT Development Framework) desarrollado por el fabricante del chip, Simba Embedded Programming Platform (en lenguaje Python), RTOS's (como Zephyr Project, Mongoose OS, NuttX RTOS), MicroPython, LUA, Javascript (Espruino, Duktape, Mongoose JS), Basic.

Finalmente se ha apostado por el uso del software Arduino IDE por las ventajas que ofrece. Al trabajar dentro del entorno Arduino podremos utilizar un lenguaje de programación conocido y hacer uso de un IDE sencillo de utilizar, además de hacer uso de toda la información sobre proyectos y librerías disponibles en internet. La comunidad de usuarios de Arduino es muy activa y da soporte a plataformas como el ESP32.

2.2.1 CARACTERÍSTICAS Y ESPECIFICACIONES DEL ESP32

En la Figura 11, es una imagen tomada de la hoja de datos, se muestran todos los bloques funcionales que conforman un SoC ESP32.

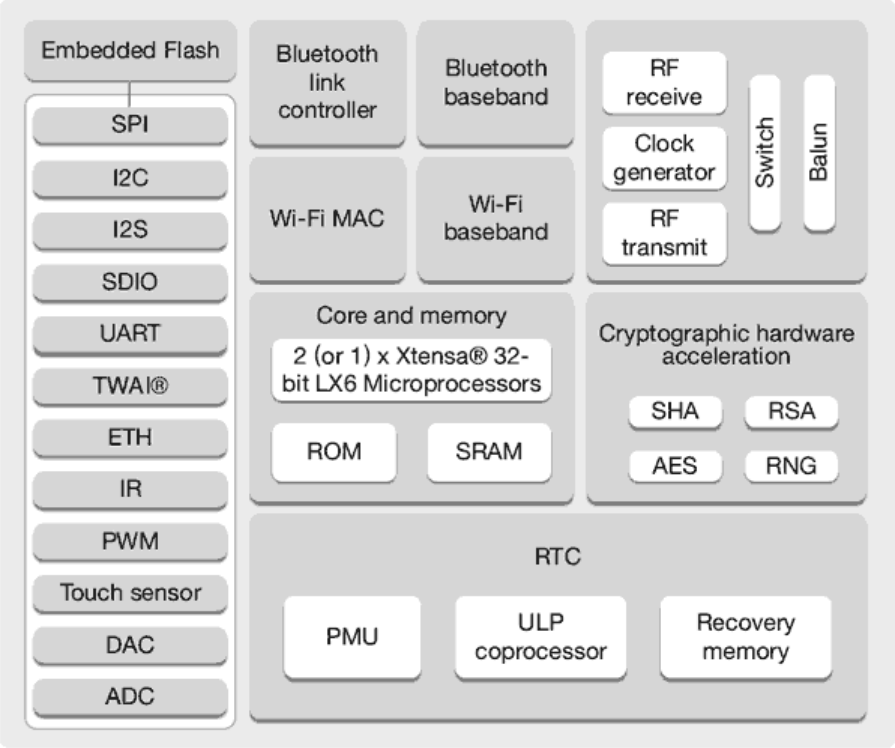


Figura 11 Arquitectura interna del SoC ESP32 (Fuente: https://pdf1.alldatasheet.com/4f9808b3-a05d-4cf5-9773-dced3f72f2ea)

2.2.1.1 CONECTIVIDAD INALÁMBRICA

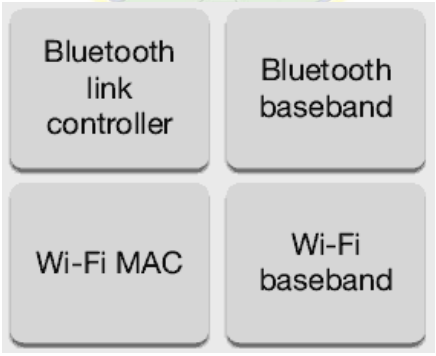


Figura 12 Bloque de conectividad Wi-Fi (Fuente: https://pdf1.alldatasheet.com/4f9808b3-a05d-4cf5-9773-dced3f72f2ea)

En la Figura 12 se ve el bloque del chip donde se puede ver que cuenta con conectividad WiFi, siendo compatible con 802.11 b/g/n en la banda de los 2.4GHz, alcanzando velocidades de hasta 150 Mbits/s. También incluye comunicación Bluetooth compatible con Bluetooth v4.2 y Bluetooth Low Energy (BLE).

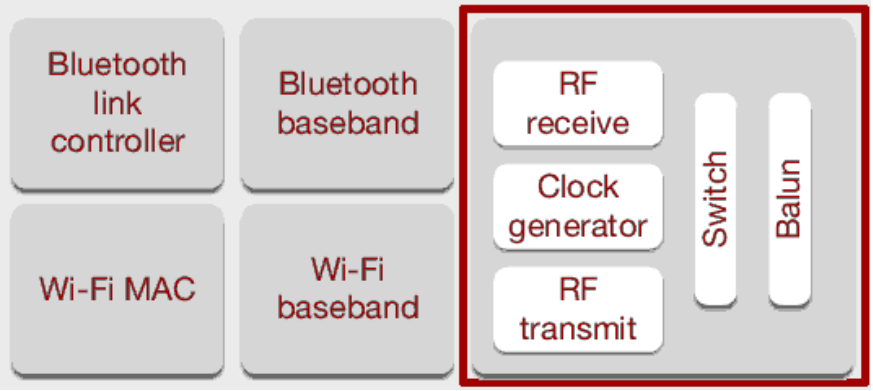


Figura 13 Bloque de RF (Fuente:https://pdf1.alldatasheet.com/4f9808b3-a05d-4cf5-9773-dced3f72f2ea)

El bloque de radio está estrechamente ligado a los módulos de comunicación inalámbrico (Figura 13). De hecho, este es el que realmente transmite y recibe la información, es decir, toma los datos digitales provenientes de los módulos Wifi y Bluetooth**;** y los convierte en señales electromagnéticas que viajan por el aire para comunicarse con tu teléfono móvil o el router. También realiza la operación inversa: traducir las ondas electromagnéticas generadas por otros dispositivos en datos digitales que los módulos Wifi y Bluetooth son capaces de interpretar.

2.2.1.2 NÚCLEO

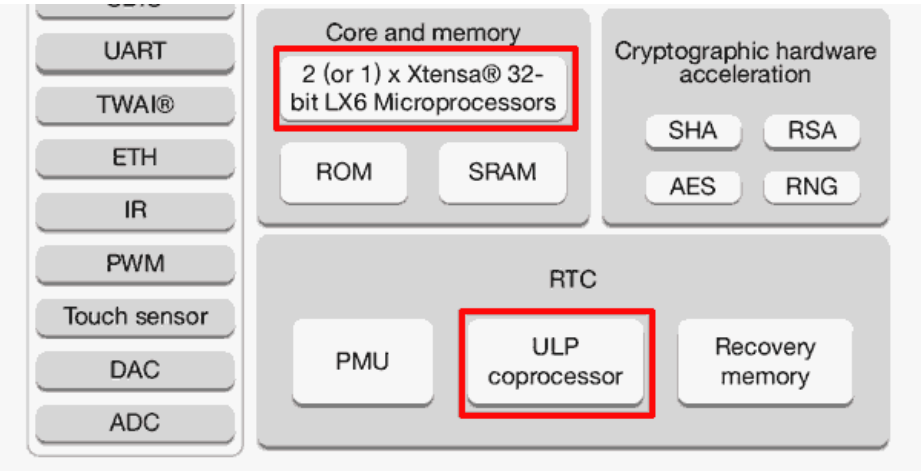


Figura 14 El núcleo de ESP32 (Fuente: https://pdf1.alldatasheet.com/4f9808b3-a05d-4cf5-9773-dced3f72f2ea)

El ESP32 cuenta con dos microprocesadores de bajo consumo Tensilica Xtensa de 32 bits LX6 (Figura14), además, cuenta con un co-procesador de ultra bajo consumo que es utilizado para realizar conversiones analógico-digital y otras operaciones mientras el dispositivo se encuentra funcionando en el modo de bajo consumo *deep sleep*. De esta forma, se consigue un consumo muy bajo por parte del SoC.

Es importante destacar que estos procesadores ofrecen grandes ventajas típicas de un procesador digital de señales:

- Frecuencia de operación: 240 MHz (ejecuta instrucciones 15 veces más rápido que una placa Arduino UNO)
- Permite realizar operaciones con números reales (números con coma) de forma muy eficiente.
- Permite realizar multiplicaciones de números grandes de forma instantánea.

Aunque estas características son transparentes cuando se está programando tienen gran repercusión en la eficiencia y tamaño del código que se graba al microcontrolador.

2.2.1.3 MEMORIAS

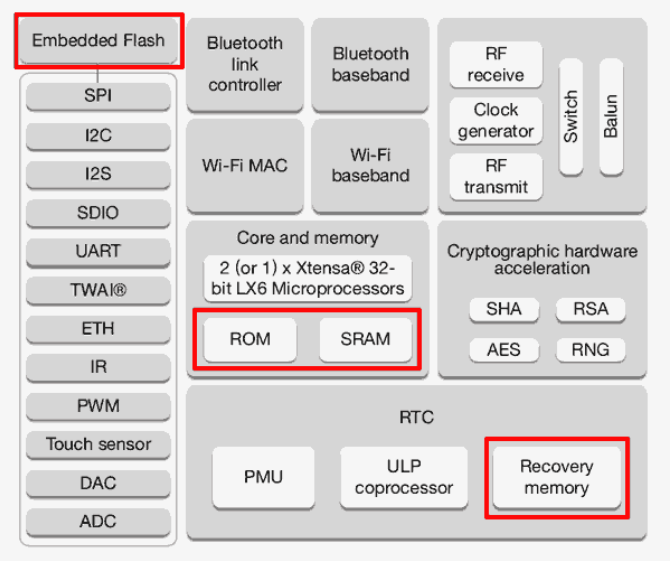


Figura 15 Bloques de memorias (Fuente: https://pdf1.alldatasheet.com/4f9808b3-a05d-4cf5-9773-dced3f72f2ea)

En la mayoría de los microcontroladores en que se basan las placas Arduino hay tres tipos de memorias:

- *Memoria de programa:* para almacenar el *sketch.*
- *Memoria SRAM*: para almacenar las variables que se utilizan en el código.
- *Memoria EEPROM:* para almacenar variables que no pierdan su valor aun cuando el dispositivo esté apagado.

En los ESP32 no ocurre así, en ellos se encuentran más tipos de memorias que se suelen clasificar en internas y externas (Figura 15).

Las memorias internas son aquellas que se encuentran ya incluidas en el SoC, y las externas son aquellas que se pueden adicionar para expandir la capacidad del sistema. Muchas placas de desarrollo basadas en ESP32 añaden memorias externas para lograr un sistema con mejores prestaciones.

En las memorias internas se encuentran:

- *Memoria ROM (448 KiB):* esta memoria es de solo escritura, es decir que no la puedes reprogramar. Aquí es donde se almacenan los códigos que manejan la pila Bluetooth, el control de la capa física de la Wifi, algunas rutinas de propósito general y el cargador de arranque (*bootloader*) para iniciar el código de la memoria externa.
- *Memoria SRAM interna (520 KiB):* esta memoria es utilizada por el procesador para almacenar tanto datos como instrucciones. Su ventaja es que, para el procesador, es mucho más fácil acceder a esta que a la SRAM externa.
- *RTC SRAM (16 KiB):* esta memoria es utilizada por el co-procesador cuando el dispositivo opera en modo *deep sleep*.
- *Efuse (1 Kilobit):* 256 bits de esta memoria son utilizados por el propio sistema y los 768 bits restantes están reservados para otras aplicaciones.
- *Flash empotrado (*Embedded flash*):* en esta memoria es donde se almacena el código de nuestra aplicación. La cantidad de memoria varía en dependencia del chip utilizado:
    - 0 MiB (chips ESP32-D0WDQ6, ESP32-D0WD y ESP32-S0WD)
    - 2 MiB (chip ESP32-D2WD)
    - 4 MiB (módulo SiP ESP32-PICO-D4)

Para los ESP32 que no poseen memoria empotrada o simplemente cuando la memoria es insuficiente para tu aplicación, es posible adicionar más memoria de forma externa:

- *Se pueden agregar hasta 16 MiB de memoria flash externa.* De esta forma puedes desarrollar aplicaciones más complejas.
- *También admite, hasta 8 MiB de memoria SRAM externa. Por* lo tanto, es difícil que te encuentres limitado en memoria al implementar una aplicación utilizando esta plataforma.

ESPECIFICACIONES TÉCNICAS	
Voltaje de Alimentación	3.3V DC (2.7~ 3.6V)
Voltaje lógico entradas/salidas (GPIO)	3.3V
Corriente de Operación	~80mA (fuente superior a 500mA)
SoM	ESP-WROOM-32
SoC	ESP32 (ESP32-D0WDQ6)
CPU	Dual core Tensilica Xtensa LX6 (32 bit)
Frecuencia de Reloj	240MHz
SRAM	520KB
Memoria Flash Externa	4MB
Pines Digitales GPIO	34 (incluyendo todos los periféricos)
UART	2
SPI	3
I2C	2
Capacitive touch sensors	10
Timers	3 (16-bit)
PWM Led	16 canales independientes (16-bits)
ADC	2 (12-bit)
DAC	2 (8-bit)
Wi-Fi	Protocolo 802.11 b/g/n/e/i (802.11n up to 150 Mbps)
	Certificación RF: FCC/CE/IC/TELEC/KCC/SRRC/NCC
	Rango de Frecuencia: 2.4 ~ 2.5 GHz
	Mode Station/SoftAP/SoftAP+Station/P2P
	Security WPA/WPA2/WPA2-Enterprise/WPS
Network Protocol	IPv4, IPv6, SSL, TCP/UDP/HTTP/FTP/MQTT

Bluetooth	Protocolos: V4.2 BR/EDR and BLE specification
	Radios: NZIF receiver with -97 dBm sensitivity, Class-1, class-2 and class-3 transmitter, AFH BluetooAudio: CVSD and SBC
Interfaz SD	Controlador SD/SDIO/MMC
	Protocolos infrarrojos
	Stack de Protocolo TCP/IP integrado

Tabla 1 Especificaciones técnicas del ESP-32 (Fuente: T. Belmonte)

2.2.1.4 PINOUT

El chip ESP32-WROOM-32 tiene un total de 38 pines, tal y como se muestra en la Figura 16, 4 de los cuales están relacionados con las entradas de alimentación (3 pines con conexión a tierra y 1 pin para la alimentación del dispositivo a 3V), 5 pines de entradas, 28 pines con función de entrada y salida y el pin NC sin una función especificada como se detalla en la Figura 6. Hay que señalar que el flash SPI tiene integrado en el módulo los pines: SCK / CLK, SDO / SD0, SDI / SD1, SHD / SD2, SWP / SD3 y SCS / CMD denotados en la placa de la GPIO6 a la GPIO11 y los cuales no se recomiendan para otros usos. También en la Tabla 1 se muestran las especificaciones técnicas del ESP32.

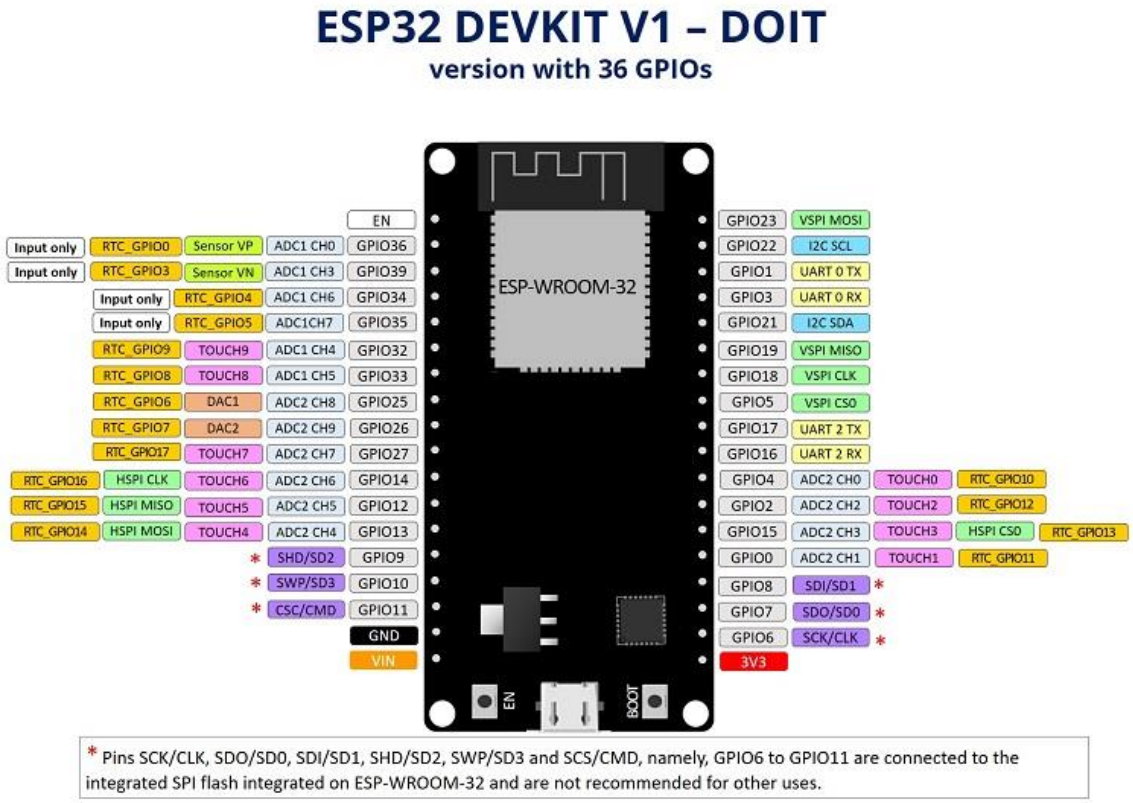


Figura 16 Pinout del chip ESP32-WROOM-32 (Fuente: https://i0.wp.com/randomnerdtutorials.com/wp-content/uploads/2018/08/ESP32-DOIT-DEVKIT-V1-Board-Pinout-36-GPIOs-updated.jpg?resize=750%2C538&quality=100&strip=all&ssl=1)

2.3 DHT11

Para la lectura de la humedad y la temperatura utilizando el chip ESP32 se decidió utilizar el sensor DHT11 (Figura 17) de la familia de los sensores DHT. El DHT11 es un sensor digital de temperatura y humedad relativa de bajo costo y fácil uso. Integra un sensor capacitivo de humedad y un termistor para medir el aire circundante, y muestra los datos mediante una señal digital en el pin de datos (no posee salida analógica). Utilizado en aplicaciones académicas relacionadas al control automático de temperatura, aire acondicionado, monitoreo ambiental en agricultura y más.

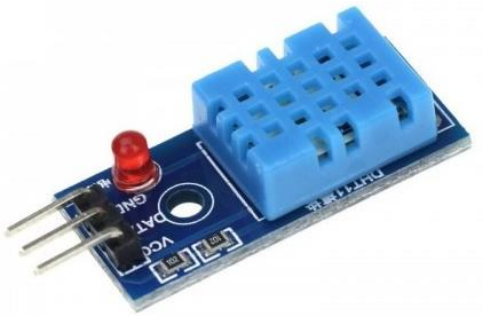


Figura 17 Sensor de humedad y temperatura DHT11

(Fuente: https://media.naylampmechatronics.com/2570-superlarge _default/sensor-de-temperatura-y-humedad-relativa-dht11.jpg)

Utilizar el sensor DHT11 con las plataformas Arduino/Raspberry Pi/Nodemcu es muy sencillo tanto a nivel de software como hardware. A nivel de software se dispone de librerías para Arduino con soporte para el protocolo "Single bus". En cuanto al hardware, solo es necesario conectar el pin VCC de alimentación a 3-5V, el pin GND a Tierra (0V) y el pin de datos a un pin digital en nuestro Arduino. Si se desea conectar varios sensores DHT11 a un mismo Arduino, cada sensor debe tener su propio pin de datos. Quizá la única desventaja del sensor es que sólo se puede obtener nuevos datos cada 2 segundos. Cada sensor es calibrado en fabrica para obtener unos coeficientes de calibración grabados en su memoria OTP, asegurando alta estabilidad y fiabilidad a lo largo del tiempo. El protocolo de comunicación entre el sensor y el microcontrolador emplea un único hilo o cable, la distancia máxima recomendable de longitud de cable es de 20m., de preferencia utilizar cable apantallado. Proteger el sensor de la luz directa del sol (radiación UV).

En comparación con el DHT22 y DHT21, este sensor es menos preciso, menos exacto y funciona en un rango más pequeño de temperatura / humedad, pero su empaque es más pequeño y de menor costo. Las especificaciones técnicas de este sensor, así como el pinout se detallan en la Tabla 2 y Tabla 3 respectivamente.

CARACTERÍSTICAS	
Voltaje de Operación:	3V - 5V DC
Rango de medición de temperatura:	0 a 50 °C
Precisión de medición de temperatura:	±2.0 °C
Resolución Temperatura:	0.1°C
Rango de medición de humedad:	20% a 90% RH.
Precisión de medición de humedad:	5% RH.
Resolución Humedad:	1% RH
Tiempo de muestreo:	1 seg.
Interface digital:	Single-bus (bidireccional)

Tabla 2 Especificaciones técnicas DHT11 (Fuente: T. Belmonte)

PINES	
1	Alimentación:+5V (VCC)
2	Datos (DATA)
3	Tierra (GND)

Tabla 3 Pinout DHT11 (Fuente: T. Belmonte)

2.4 EL DISPLAY LCD1602

El Display LCD1602 (Figura 18), permite mostrar texto/números/caracteres, además hacer *debugging* o correcciones en nuestros proyectos sobre todo cuando se trabaja con sensores y procesamiento de datos. Al funcionar con bajo voltaje**,** tanto para su alimentación como para la comunicación, lo cual es muy útil en proyectos basados en microcontroladores de 3.3V como ESP8266/ESP32/RaspberryPi/Teensy y la gama más avanzada de Arduino como Arduino Due.

Figura 18 Pantalla LCD 16X02 (Fuente: https://media.naylampmechatronics.com/3381-medium_default/display-alfanumerico-lcd-1602-3v.jpg)

El LCD 1602 posee 2 filas y 16 columnas de dígitos alfanuméricos, funciona con el controlador interno HD44780, que es un integrado muy utilizado y para el cual existe amplia documentación. Para conectar la pantalla LCD a nuestro Arduino/PIC se necesitan 6 pines: 2 de control y 4 de datos, verificando el pinout del mismo como se ve en la Tabla 4. En cuanto a la programación en Arduino ya se incluye por defecto la librería *LiquidCrystal***,** que incluye ejemplos de prueba.

Si bien es posible conectar directamente la pantalla LCD a nuestro Arduino, es una buena opción utilizar un adaptador LCD paralelo a serial I2C y de esa forma ahorrar pines, trabajando con solo 2 pines del puerto I2C. Se debe alimentar el modulo adaptador LCD a I2C con 3.3V.

PIN No	FUNCIÓN	NOMBRE
1	Tierra (0V)	Ground
2	Voltaje de Operación 3.3V	Vcc
3	Ajuste de contrastes mediante un potenciómetro (puede ser de 1k-10k)	VEE
4	Selección del registro, para 0 es comandos y en 1 es para datos	Register Select

5	Estado bajo para escribir y estado alto para leer el registro	Read/write
6	Envía datos a los pines de datos cuando recibe un flanco de bajada	Enable
7	Pines de datos 8-bit	DB0
8		DB1
9		DB2
10		DB3
11		DB4
12		DB5
13		DB6
14		DB7
15	Backlight VCC (3.3V)	Led+
16	Backlight Tierra (0V)	Led-

Tabla 4 Pinout LCD 1602 (Fuente: T. Belmonte)

2.5 MODULO ADAPTADOR LCD A I2C

Conectar nuestra pantalla LCD1602 con ESP32 es mucho más sencillo con la ayuda del Módulo adaptador de LCD a interfaz I2C (Figura 19) pues permite manejar nuestro LCD utilizando solo 2 pines (SDA y SCL). El módulo es compatible con los LCD 1602 y LCD 2004, está basado en el controlador I2C PCF8574 que es un expansor de entradas y salidas digitales controlado por I2C. Por el diseño del PCB este módulo se usa especialmente para controlar un LCD Alfanumérico.

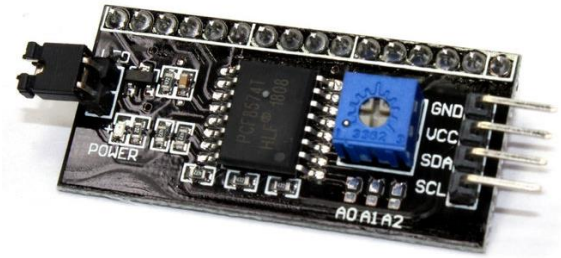


Figura 19 Modulo LCD a I2C (Fuente: https://media.naylampmechatronics.com/700-superlarge_default/modulo-adaptador-lcd-a-i2c-pcf8574.jpg)

La dirección I2C por defecto del módulo puede ser 0x3F o en otros casos 0x27. Es muy importante identificar correctamente la dirección I2C de nuestro modulo, pues de otra forma nuestro programa no funcionará correctamente. Para identificar la dirección especifica de nuestro módulo podemos utilizar un pequeño sketch de prueba llamado: I2C Scanner, el cual nos permite identificar la dirección I2C del dispositivo conectado al Arduino. Las especificaciones del conversor se muestran en la Tabla 5.

Si en nuestro proyecto contamos con varios dispositivos I2C, este módulo se conecta en el mismo bus y no utiliza más pines. Para cambiar la dirección I2C basta soldar unos puentes que trae el módulo, por ejemplo, para utilizar hasta 8 Conversores I2C + LCD y así poder manejar cada LCD de forma independiente.

CARACTERÍSTICAS	
Voltaje de alimentación:	5V DC
Chip Controlador:	PCF8574
Dirección I2C:	0x3F (en algunos modelos es 0x27)
Otros	Compatible con el protocolo I2C
	Jumper para luz de fondo (back light)
	Potenciómetro para ajuste de contraste

Tabla 5 Especificaciones técnicas Modulo LCD a I2C (Fuente: T. Belmonte)

2.6 YL-38

El módulo YL-38 (Figura 20), es capaz de detectar gotas de agua lluvia, por lo que puede ser utilizados para sistemas de detección que requieran realizar funciones cuando empieza a llover.

Este módulo consiste en una serie de pistas conductoras impresas sobre una placa de baquelita. La separación entre las pistas es muy pequeña. Lo que este módulo hace es crear un corto circuito cada vez que las pistas se mojan.

Figura 20 Módulo sensor YL-38

(Fuente:http://www.arduinove.com/image/cache/catalog/sensores/sensor-lluvia-228x228.png)

El agua hace que se cree un camino de baja resistencia entre las pistas con polaridad positiva y las pistas conectadas al GND. La corriente que fluye a través de estas pistas se ve limitada por resistencias de 10K en cada conductor, lo que impide que el corto circuito que se genera cuando se moja la placa vaya a estropear el micro controlador.

Se puede utilizar para todo tipo de monitoreo del clima, y se traduce en señales de salida y AO.

- El sensor utiliza el material doble de alta calidad FR - 04, área grande de 5.5 * 4.0 CM, tratamiento de niquelado y superficie, tiene oxidación de lucha, conductividad eléctrica y vida tiene un rendimiento superior.

- La salida del comparador, la limpieza de la señal, la buena forma de onda, la capacidad de conducción es fuerte, por más de 15 mA.
- Con ajuste de sensibilidad del potenciómetro.
- La tensión de funcionamiento de 3.3 V a 5 V.
- El formato de salida: salida de conmutador digital (0 y 1) y salida de voltaje analógico AO.
- Tiene un orificio para perno fijo, instalación conveniente.
- Tamaño de PCB de placa pequeña: aprox. 3,2 cm x 1,5 cm.
- El LM393, uso de un comparador de voltaje amplio

2.7 BMP180

El sensor de presión barométrica BMP180 de la Figura 21, permite medir la altura respecto al nivel del mar, su funcionamiento está basado en la relación entre presión del aire y la altitud. El BMP180 posee alta precisión y de bajo consumo de energía. Ofrece un rango de medición desde 300 a 1100 hPa (Hecto Pascal), con una precisión absoluta de hasta 0,03 hPa. Basado en tecnología piezo-resistiva de BOSCH con robustez EMC, alta precisión y linealidad, así como con estabilidad a largo plazo. Diseñado para ser conectado directamente a un microcontrolador a través de I2C utilizando solo 2 lineas.

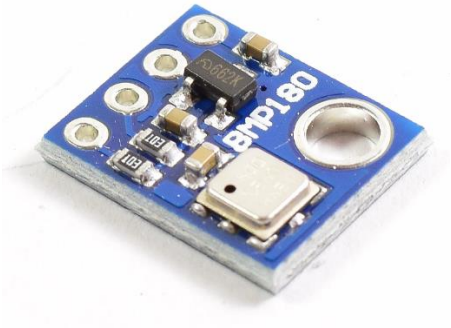


Figura 21 Sensor BMP180 (Fuente:https://media.naylampmechatronics.com/664-superlarge_default/sensor-de-presion-bmp180.jpg)

Este tipo de sensores pueden ser utilizados para calcular la altitud con gran precisión, por lo que es un sensor muy utilizado en sistemas de Autopiloto para Drones (UAVs). Sus características principales se detallan en la Tabla 6.

CARACTERÍSTICAS	
Voltaje de Operación:	3.3V - 5V DC
Interfaz de comunicación:	I2C (3.3V)
Rango de Presión:	300 a 1100 hPa (0.3-1.1bar)
Resolución:	1 Pa
Precisión absoluta:	1 hPa
Resolución de temperatura:	0.1°C
Precisión Temperatura:	Precisión Temperatura: 1°C
Frecuencia de Muestreo:	120 Hz (máx.)
Rango de altura medible:	0-9100 metros

Tabla 6 Datos técnicos BMP180 (Fuente: T. Belmonte)

2.8 NODE-RED

Node-RED es una poderosa herramienta de código abierto desarrollado por IBM para crear aplicaciones de Internet de las cosas (IoT), con el objetivo de simplificar el componente de programación, su modo de trabajo se puede ver en la Figura 23. Utiliza una programación visual que le permite conectar bloques de código conocidos como nodos, para realizar una tarea. Los nodos cuando están conectados entre sí se denominan flujos.

Proporciona un editor basado en navegador que facilita la conexión de flujos mediante la amplia gama de nodos de la paleta que se pueden implementar en su tiempo de ejecución con un solo clic.

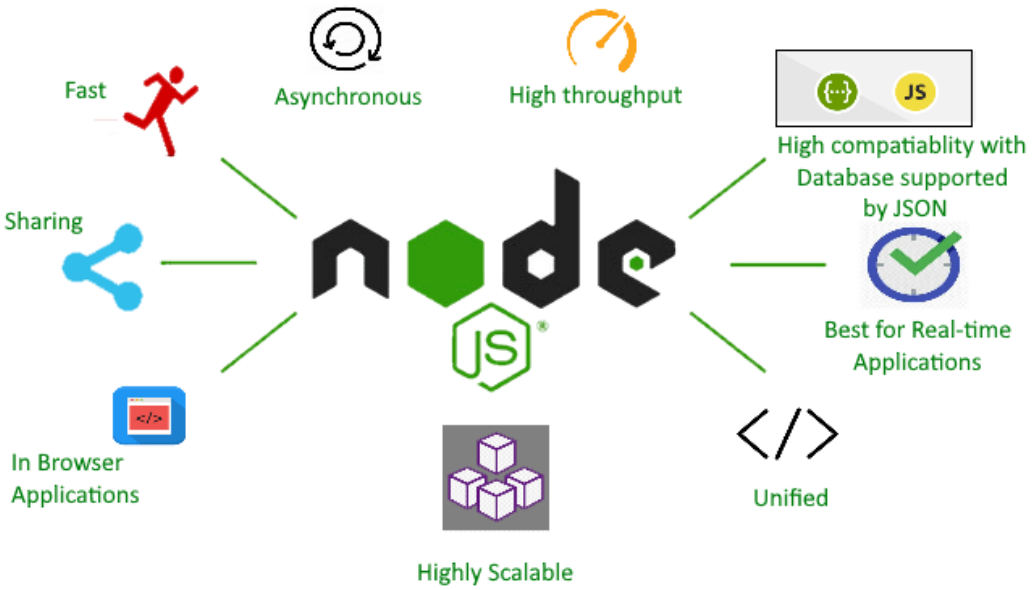


Figura 22 Complemento NodeJS para MQTT (Fuente: https://lh3.googleusercontent.com/-GxfVmK8uTWk/YTWj2y5RPOI/AAAAAAAAAK0/aBX5ZSMvv3g7Xev8UZJmRn4JF4XUZyebgCLcBGAsYHQ/s1600/1630905304770219-0.png)

El entorno de ejecución ligero se basa en Node.js y aprovecha al máximo su modelo sin bloqueo basado en eventos como se ve en la Figura 22. Esto lo hace ideal para ejecutarse en el borde de la red en hardware de bajo costo, así como en la nube.

Node.js se registra con el sistema operativo y cada vez que un cliente establece una conexión se ejecuta un callback. Dentro del entorno de ejecución de Node.js, cada conexión recibe una pequeña asignación de espacio de memoria dinámico, sin tener que crear un hilo de ejecución. A diferencia de otros servidores dirigidos por eventos, el bucle de gestión de eventos de Node.js no es llamado explícitamente, sino que se activa al final de cada ejecución de una función callback. El bucle de gestión de eventos se termina cuando ya no quedan eventos por atender.

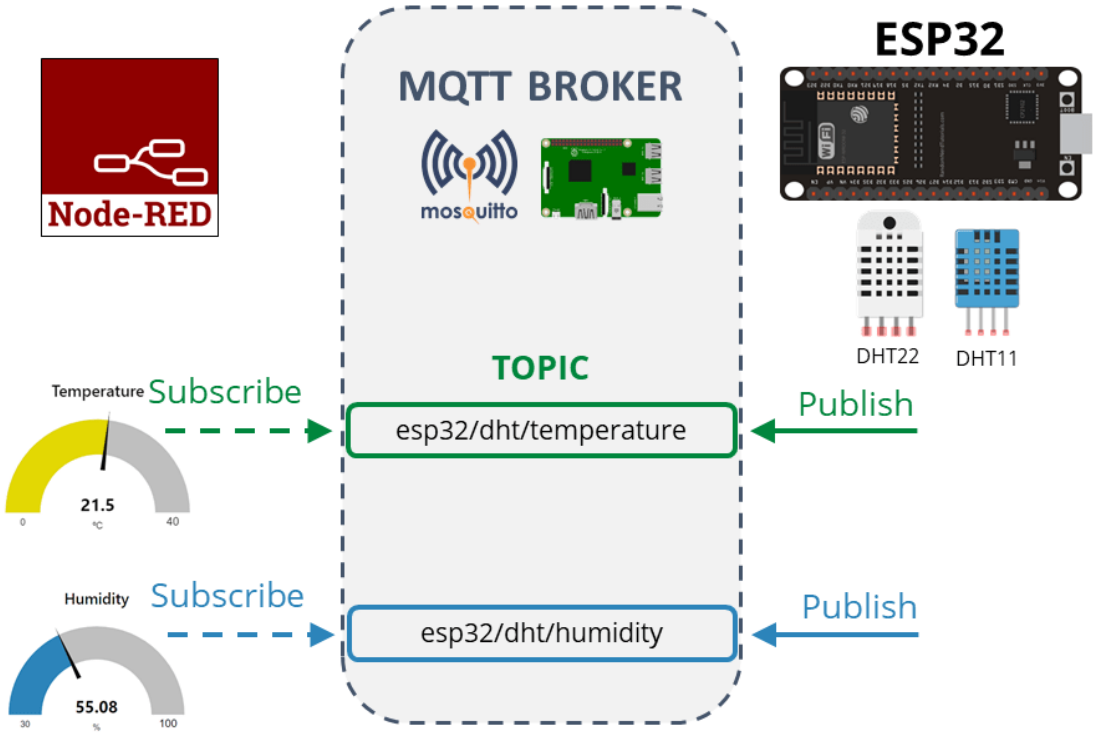


Figura 23 Esquema de trabajo Node-Red y MQTT (Fuente: https://i0.wp.com/randomnerdtutorials.com/wp-content/uploads/2020/04/ESP32-MQTT-DHT11-DHT22-Node-RED.png?quality=100&strip=all&ssl=1)

Ventajas de utilizar Node-RED:

- Permite acceder a las GPIO de diversas placas como: Raspberry Pi, ESP8266, ESP32, Arduino.
- Establece conexiones MQTT con otras placas (Arduino, ESP32, ESP8266, etc.)
- Crea una interfaz gráfica para los proyectos IoT de una manera muy sencilla.
- Comunica con servicios de terceros (IFTTT.com, Adafruit.io, Thing Speak, etc.)
- Captura datos de webs (pronóstico del tiempo, precios de acciones, correos electrónicos, etc.)
- Crear eventos activados por tiempo.
- Almacena y recupera datos de una base de datos.

2.8.1 SECCIONES PRINCIPALES DEL PROGRAMA:

Como se ve en la Figura 24, en el lado izquierdo, hay una amplia lista de bloques llamados nodos que están separados por su funcionalidad. Si selecciona un nodo, se puede saber cómo funciona en la pestaña de información. En el centro se encuentra el flujo que es donde se colocan los nodos.

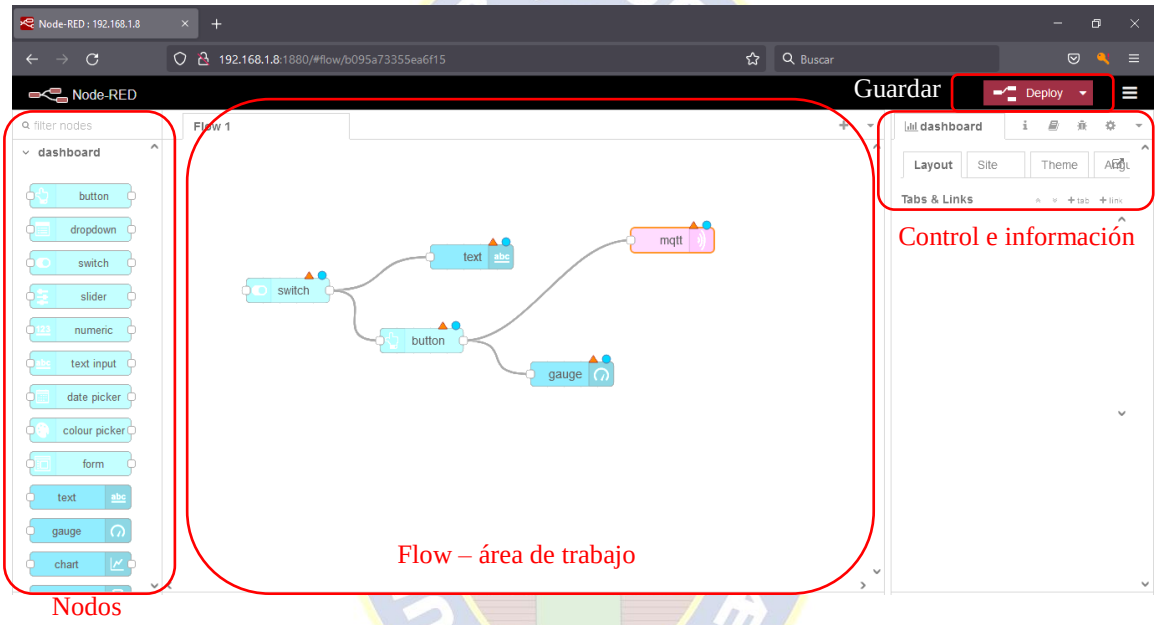


Figura 24 Principales secciones del Node-Red (Fuente: T. Belmonte)

Para crear una interfaz de usuario en Node-RED se utilizan los nodos de tipo ***Dashboard*** que proporcionan widgets mostrándose en la interfaz de usuario IU de la aplicación. La interfaz de usuario está organizada en pestañas y grupos. Dentro de cada pestaña existen grupos que dividen las pestañas en diferentes secciones, para poder organizar los widgets. Node-RED ofrece una amplia gama de opciones que nos permiten modificar el estilo y el formato de nuestra aplicación web, posibilitando ajustar la herramienta a nuestra medida.

CAPÍTULO III

INGENIERÍA DEL PROYECTO

3.1 DESCRIPCIÓN DEL PROYECTO

El diagrama en bloque del sistema de gestión de la estación meteorológica de la figura 25, describe de manera gráfica el funcionamiento del proyecto desarrollado.

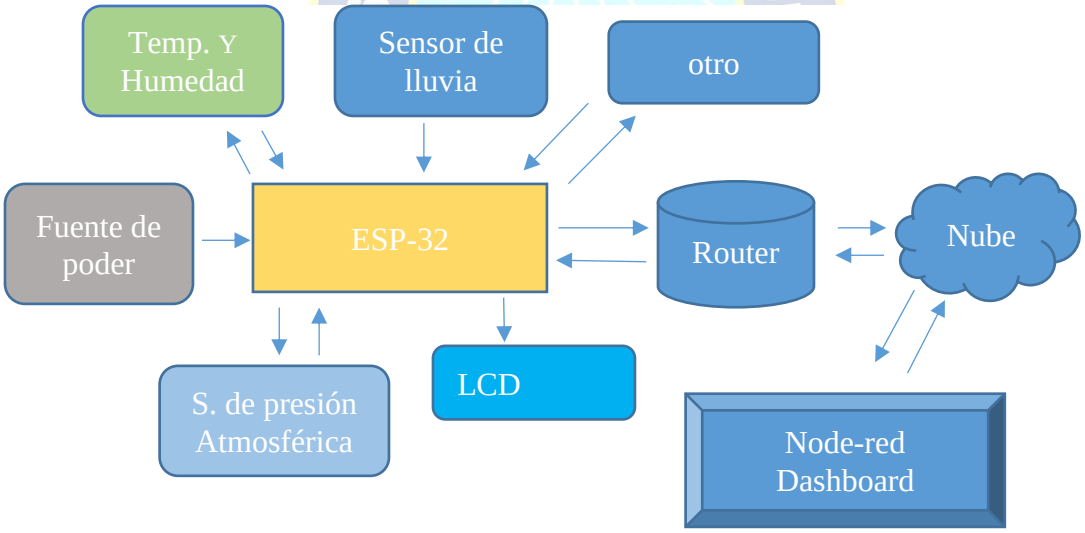


Figura 25 Diagrama en bloque de la Estación Meteorológica (Fuente: T. Belmonte)

Sin embargo, para poder desarrollar este proyecto de manera que, no presente dificultad al usar el protocolo MQTT sin ningún problema, se deben tomar en cuenta las siguientes consideraciones, respecto al uso adecuado de los programas que a continuación se describirá.

3.2 GESTOR MOSQUITTO PARA PROTOCOLO MQTT

Eclipse Mosquitto es un intermediario de mensajes de código abierto (con licencia EPL/EDL) que implementa las versiones 5.0, 3.1.1 y 3.1 del protocolo MQTT. Mosquitto es liviano y es adecuado para su uso en todos los dispositivos, desde computadoras de placa única de bajo consumo hasta servidores completos como se ve en la Figura 26.

Figura 26 Gestor mosquitto MQTT (Fuente: https://jelastic.com/blog/wp-content/uploads/2017/09/building-m2m-iot-projects.png)

El protocolo MQTT proporciona un método ligero para enviar mensajes utilizando un modelo de publicación/suscripción. Esto lo hace adecuado para la mensajería de Internet de las cosas, como sensores de baja potencia o dispositivos móviles como teléfonos, computadoras integradas o microcontroladores.

Mosquitto es un bróker MQTT OpenSource ampliamente utilizado debido a su ligereza lo que nos permite, fácilmente, emplearlo en gran número de ambientes, incluso si éstos son de pocos recursos y como está sobre TCP/IP (Figura 27), es aún más sencillo, por que usa una estructura mucho más ligera.

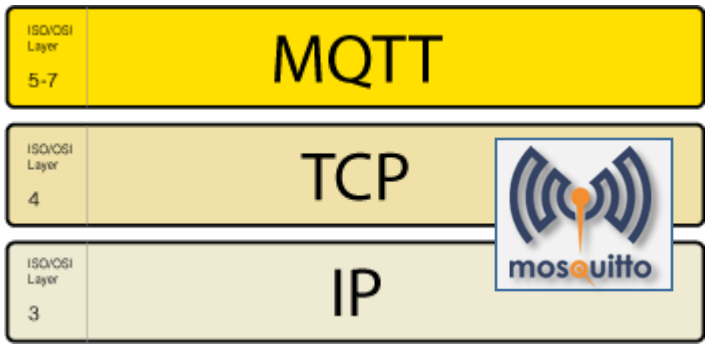


Figura 27 Mosquitto con TCP/IP (Fuente: https://slideplayer.es/slide/13630933/83/images/12/Donde+funciona+MQTT.jpg)

El gestor Mosquitto también proporciona una biblioteca C para implementar clientes MQTT y los muy populares clientes MQTT de línea de comandos mosquitto_pub y mosquitto_sub.

Para poder empezar a utilizar este gestor debemos poder instalarlo, se logra descargando el archivo de intalacion desde la pagina oficial y realizar una prueba sobre el gestor en cmd del equipo local (Fgura 30) y ademas se podra ver la version con la cual se esta conectando (Figura 28). Previamemte el gestor mosquitto para ser montada, debe instalarse su complemento , el Node.js, ya que se lo necesitara.

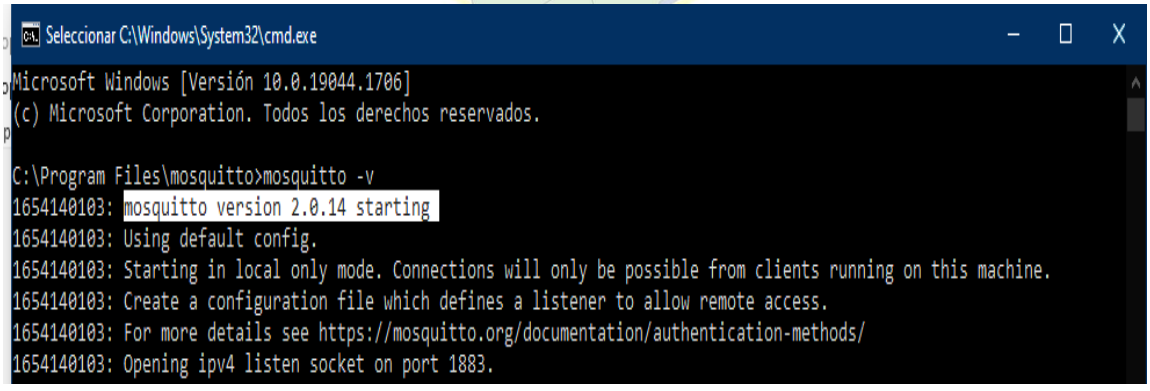


Figura 28 Prueba de inicio de Mosquitto (Fuente: T. Belmonte)

Para poder trabajar de manera adecuada se debe habilitar los puertos de trabajo por defecto del gestor y habilitar el puerto 1883 tanto de entrada como de salida. Esto se hace modificando el archivo *mosquitto.conf* de la carpeta donde fue instalada como se ve en la Figura 29. Luego se debe habilitar el puerto en el cortafuego del equipo y darle el permiso correspondiente. Este paso es muy importante, porque si se pasa por alto el programa no se ejecutará de manera correcta en el host bróker.

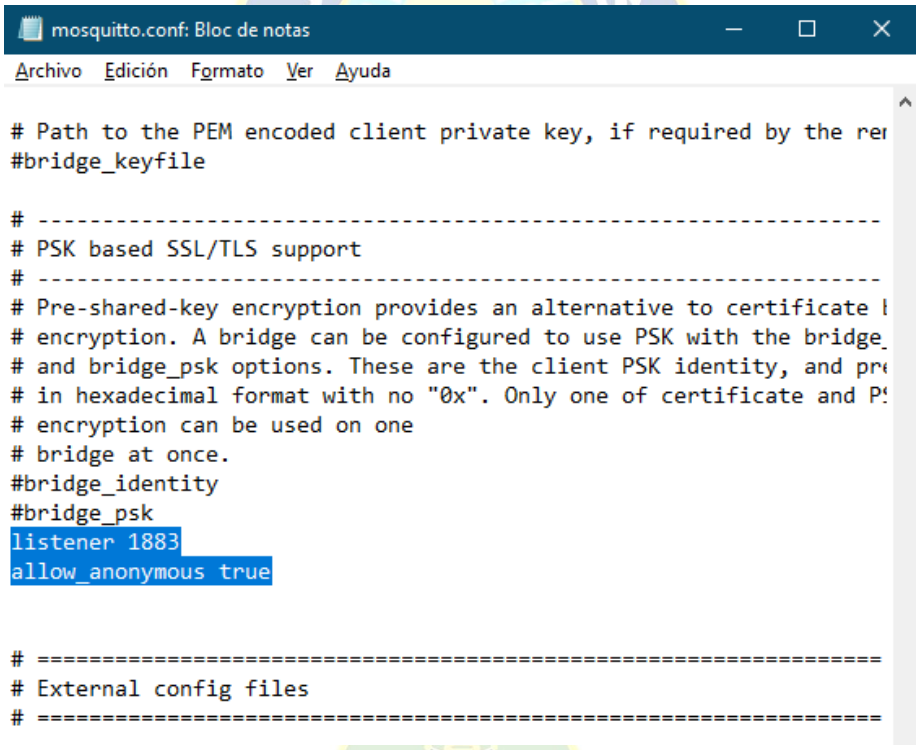


Figura 29 Modificación del archivo mosquitto.conf para el puerto 1883 (Fuente: T. Belmonte)

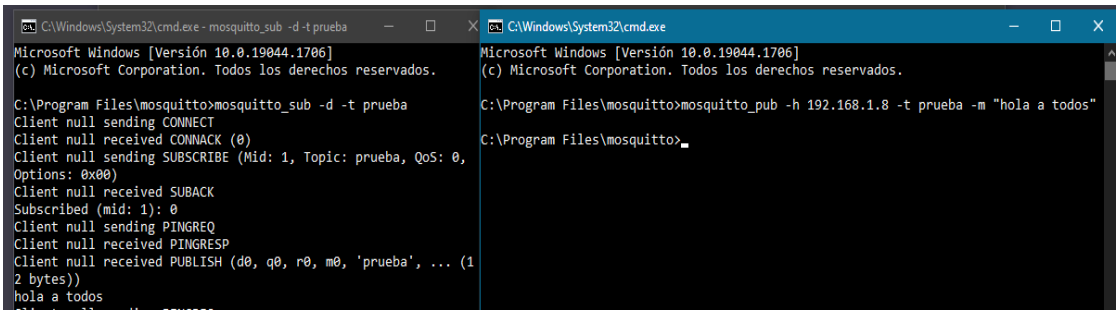


Figura 30 Prueba de conexión del host con Mosquito (Fuente: T. Belmonte)

3.3 ARDUINO IDE

Para la programación del chip ESP32 se decidió utilizar el conocido IDE (*entorno de desarrollo integrado*) de Arduino. Esta es una plataforma de código abierto que ha ganado gran popularidad por las facilidades de uso que brinda a los usuarios.

Arduino surgió en el Ivrea Interaction Design Institute como una herramienta fácil para la creación rápida de prototipos. Está basado en el lenguaje de programación de Java y se caracteriza por ser una aplicación multiplataforma, posibilitando su instalación tanto en Windows, como en macOS o Linux. El código fuente para el IDE de Arduino se publica bajo la Licencia Pública General de GNU. Admite los lenguajes C y C ++ utilizando reglas especiales de estructuración de códigos y suministra una biblioteca de software del proyecto Wiring.

3.3.1 SECCIONES DEL IDE DE ARDUINO

El entorno de desarrollo integrado de Arduino contiene un editor de texto para escribir código, un área de mensajes, una consola de texto, una barra de herramientas con botones para funciones comunes y una serie de menús como se detalla en la Figura 31**.** Se conecta al hardware Arduino y Genuino para cargar programas y comunicarse con ellos.

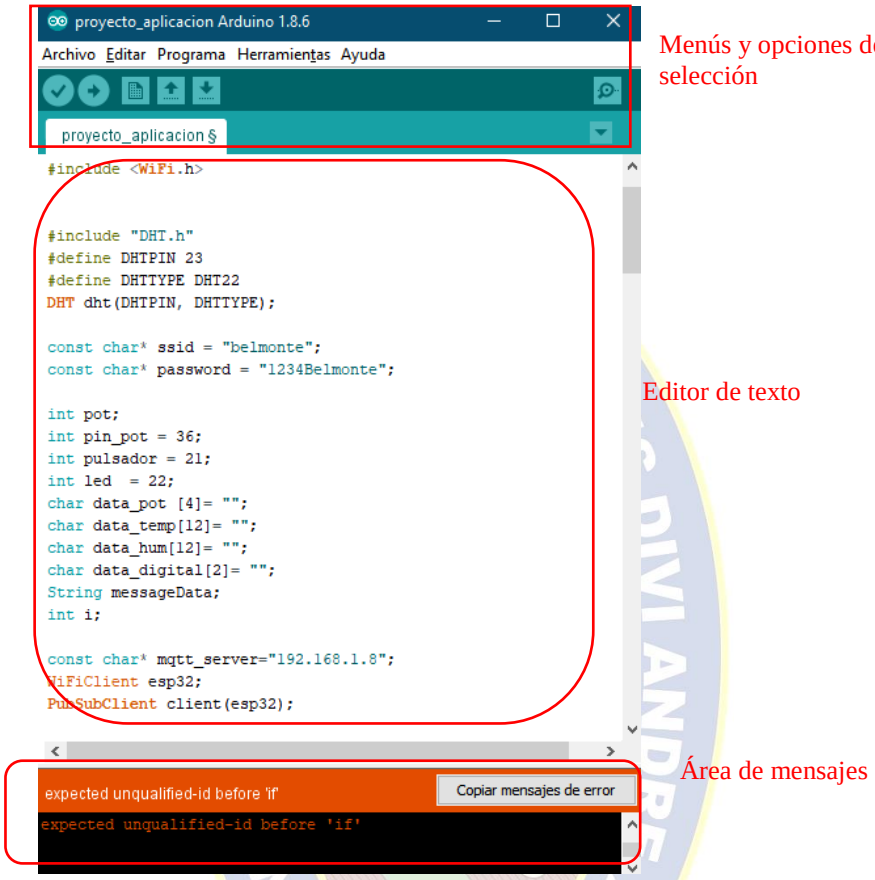


Figura 31 Secciones del IDE de Arduino (Fuente: T. Belmonte)

Los programas escritos con el software Arduino (IDE) se denominan bocetos o ***sketch***. Estos bocetos se escriben en el editor de texto y se guardan con la extensión de *archivo****.ino*** **(**generará en última instancia un archivo hexadecimal que luego se transfiere y se carga en el controlador de la placa**)**. El editor tiene funciones para cortar, pegar y para buscar y también para reemplazar texto. El área de mensajes proporciona información al guardar y exportar y también muestra errores. La consola (Figura 29) muestra la salida de texto del software Arduino (IDE), incluidos los mensajes de error completos y otra información.

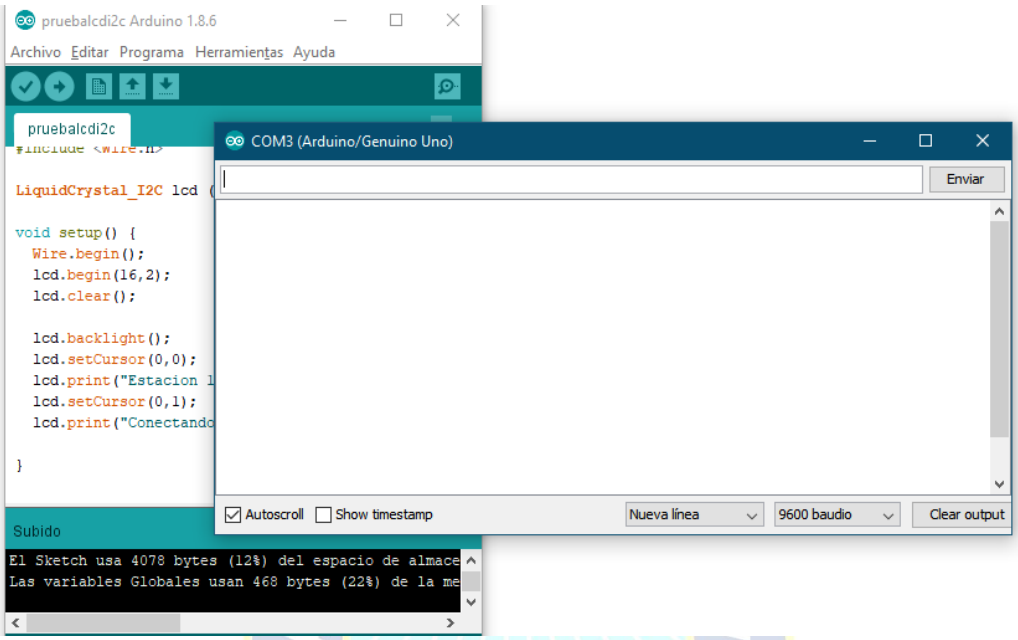


Figura 32 Monitor serie del IDE (Fuente: T. Belmonte)

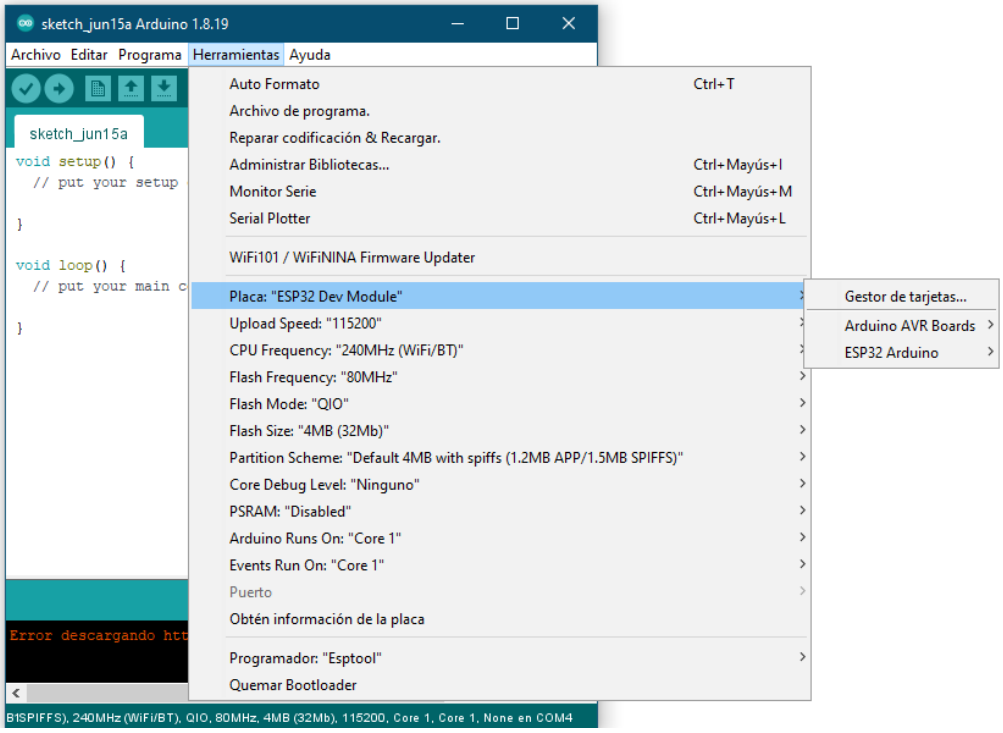


Figura 33 Selección del módulo en el menú herramientas (Fuente: T, Belmonte)

La opción de “Herramientas” (Figura 32) se utiliza principalmente para configurar proyectos de prueba. La sección del “Programador” de este panel se utiliza para grabar un cargador de arranque en el nuevo microcontrolador (Figura 33). Es de suma importancia seleccionar correctamente el modelo de la placa y el puerto COM serie en el IDE (Figura 34).

Se puede buscar el dispositivo serie USB en la sección de puertos del Administrador de dispositivos de Windows.

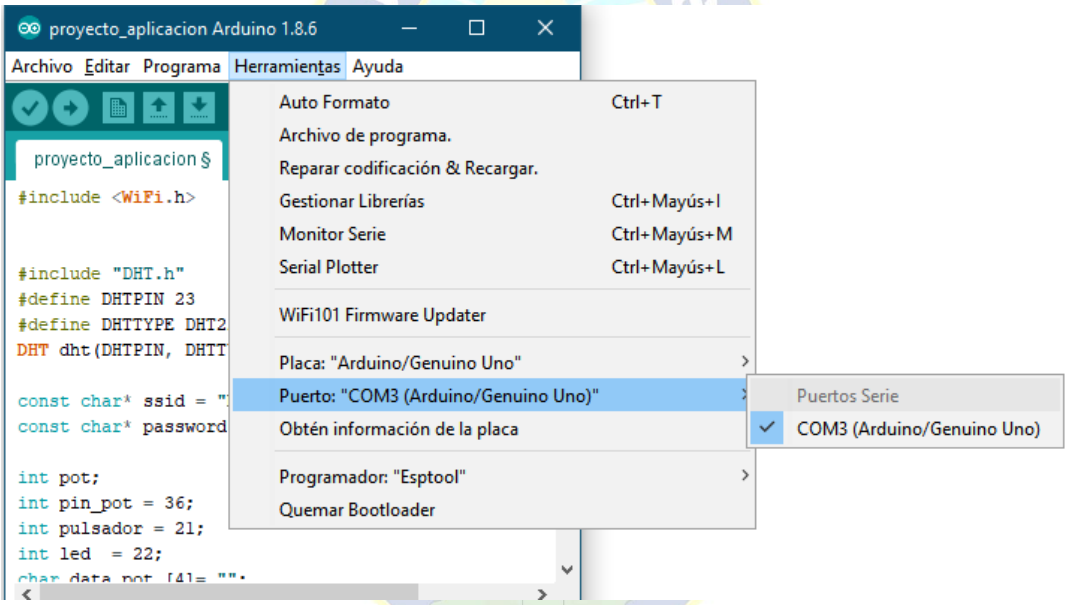


Figura 34 Selección del Puerto COM (Fuente: T. Belmonte)

Las bibliotecas son muy útiles para agregar la funcionalidad adicional al módulo Arduino. Hay una lista de bibliotecas que se pueden agregar haciendo clic en el botón Sketch en la barra de menú y yendo a “Incluir biblioteca” (Figura 35). Al hacer clic en” Incluir biblioteca “y agregar la biblioteca respectiva, aparecerá en la parte superior del boceto con un signo ***#incluir***. La mayoría de las bibliotecas están preinstaladas y vienen con el software Arduino. Sin embargo, también se pueden descargarlos de fuentes externas.

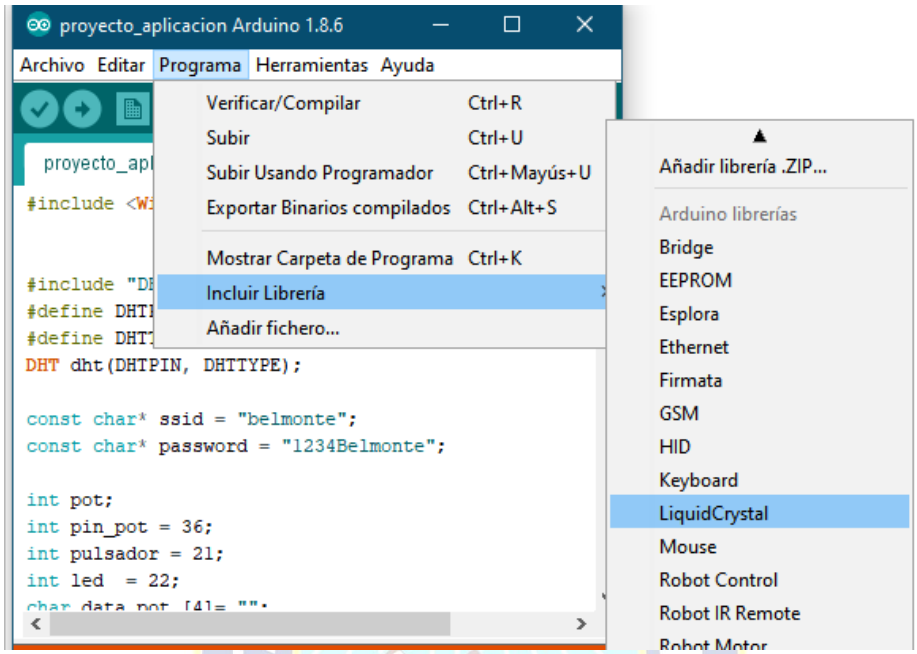


Figura 35 Incluir librerías del IDE (Fuente: T. Belmonte)

3.3.2 PROGRAMACIÓN EN EL IDE DE ARDUINO

La estructura del software consta de dos funciones principales (Figura 36):

Función de configuración (): La función ***setup ()*** se llama cuando se inicia un boceto. Se usa para inicializar las variables, modos de pin, comenzar a usar bibliotecas, etc. La función de configuración solo se ejecutará una vez, después de cada encendido o reinicio de la placa Arduino. Aquí, se puede definir la velocidad de salida en el ***Serial.begin*** () (declaración que abre el puerto serie para permitir que la placa envíe salida para que la muestre el monitor serie).

Función loop (): Después de llamar a la función setup (), que inicializa y establece los valores iniciales, la función loop () hace precisamente lo que sugiere su nombre y se repite consecutivamente, lo que permite que su programa cambie y responda. Se utiliza para controlar activamente la placa Arduino.

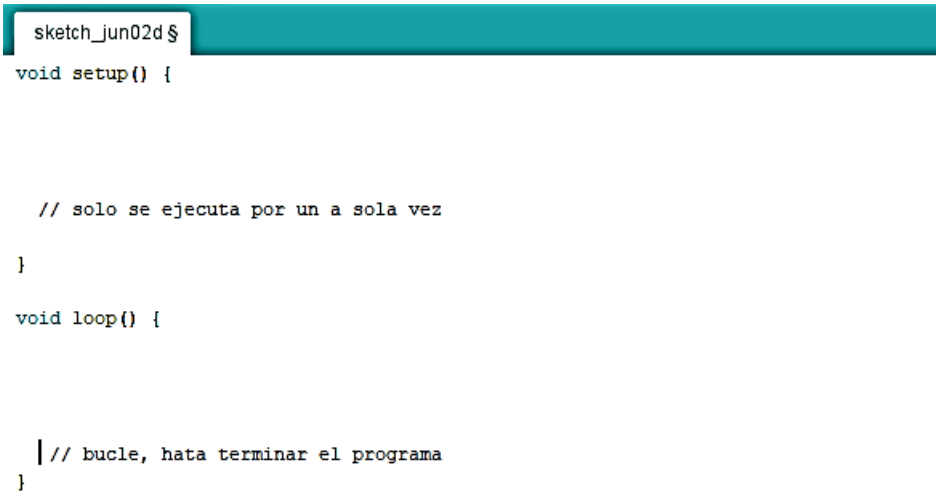


Figura 36 Estructura del IDE (Fuente: T. Belmonte)

Cuando este proceso se determine de manera correcta se realiza la estructuración del programa principal del ESP32 que hará que funcione tal y como pretendemos realizarlo debiendo tener cuidado con la programación. La Figura 37 muestra el diagrama desde la programación hasta cuando el programa es compilado y grabad en el chip.

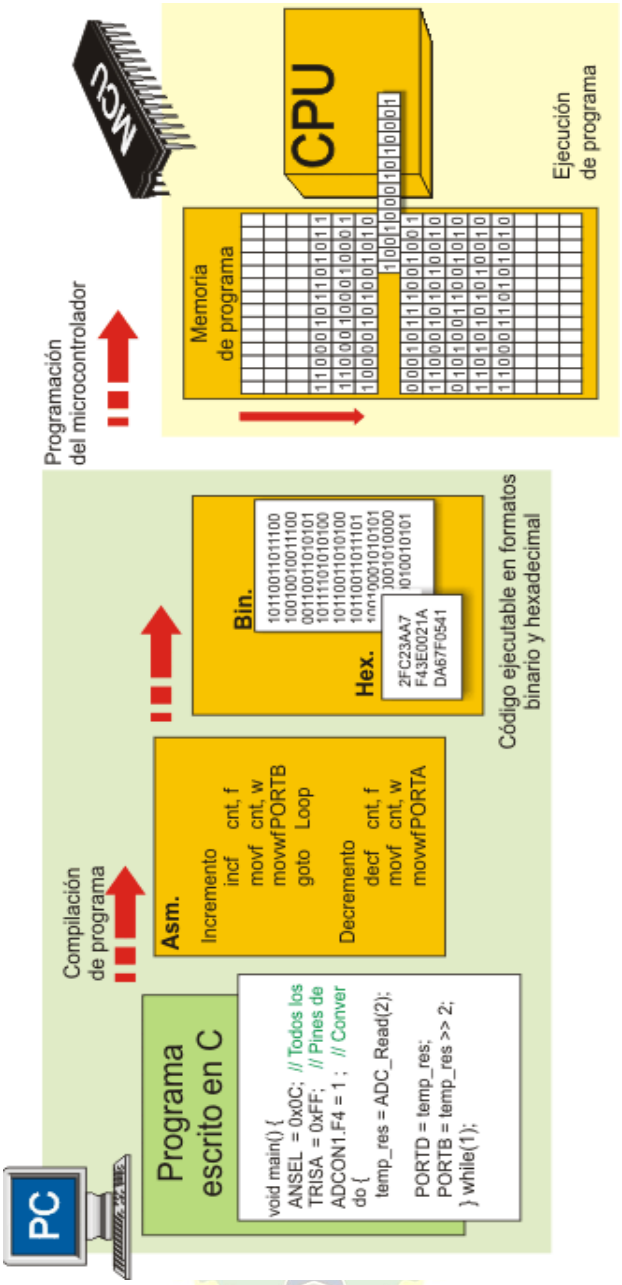


Figura 37 Diagrama de carga del sketch o programa (Fuente: https://cdn.mikroe.com/ebooks/img/37/2016/02/al-mundo-de-los-microcontroladores-chapter-02fig2-2.gif)

3.4 LA PROGRAMACIÓN

El programa está realizado en Arduino, ya que proporciona un entorno de programación sencillo y potente para programar, pero además incluye las herramientas necesarias para compilar el programa y "quemar" el programa ya compilado en la memoria flash del microcontrolador. Además, el IDE nos ofrece un sistema de gestión de librerías y placas muy práctico. Como IDE es un software sencillo que carece de funciones avanzadas típicas de otros IDEs, pero suficiente para programar.

El programa desarrollado para el funcionamiento de la aplicación en Node-Red a través del protocolo MQTT, es el siguiente:

```
//**ESTACION METEREOLOGICA**


// declarando librerías wifi - mqtt

#include <PubSubClient.h>
#include <WiFi.h>

#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);

/*#include <Wire.h>
#include <Adafruit_BMP085.h>

Adafruit_BMP085 MySensorBMP;

/*#include "Adafruit_I2CDevice.h"
#include  "Adafruit_BMP085.h "*/
#include "WiFi.h"
```

```
#include "DHT.h"
#define DHTPIN 23
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

const char* ssid = "NOMBRE DE RED";          //Entorno de red conectado al Broker
const char* password = "CONTRASEÑA";         //La contraseña

int pot;
int pin_pot = 36;
int pulsador = 21;
int led = 22;
char data_pot [4] ="";
char data_temp[12] ="";
char data_humi[12] ="";
//char data_pres[12] ="";
char data_digital[2] ="";
String messageData;
int i;

const char* mqtt_server = "192.168.XX.XX";    //ip del servidor

WiFiClient esp32;
PubSubClient client(esp32);

long lastMsg = 0;
long tdhtx = 0;
char msg[50];
```

```
int value = 0;

//------------------------CONEXIÓN WI-FI-------------------------------
void config_wifi() {
Serial.print("Conectandose a la red");
Serial.println(ssid);
WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
Serial.println("");
Serial.println("ESP32 conectado, su IP es: ");
Serial.println(WiFi.localIP());
}

//---------------------------------SUSCRIPCIÓN--------------------
void callback (String topic, byte* message,unsigned int length) //recibe los datos
{
Serial.println("Mensaje que llega del topic ");
Serial.print(topic);
Serial.print(". Message: ");

for (int i = 0; i < length; i++);
{
Serial.print((char)message[i]);
messageData += (char)message[i];
```

```
}
Serial.println();
}

//----------------------led----------------------

/*if(messageData == "true")
{digitalWrite(led,HIGH);}
if(messageData == "false")
{digitalWrite(led,LOW);}*/

//-------------------------CONEXIÓN A BROKER/SUSCRIB-------------------------
void reconnect() {
while (!client.connected()) {
Serial.print("Iniciando conexión con Broker..");
String clienteId = "ESP32";
if (client.connect(clienteId.c_str()))
{
Serial.println("Conectado");
client.subscribe("esp32iot");   //topic para suscribir
}else {
Serial.print("Failed. rc=");
Serial.print(client.state());
Serial.println(" esperando 3 segundos");
delay(3000);
}
}
}
```

```
//------------------------------EJECUCIÓN DEL SETUP--------------------------------

void setup() {

/*Serial.begin(9600);
if (!MySensorBMP.begin()) {
Serial.println("No sensor BMP180, verifique las conexiones...!");
while (1) {}
}*/

Serial.begin(115200);
config_wifi();
dht.begin();          //inicio dht11
client.setServer(mqtt_server,1883);
client.setCallback(callback);

lcd.init();
lcd.backlight();

lcd.setCursor(0,0);
lcd.print("Conect..Estación...");
}

//---------------------BUCLE-----------------

void loop() {
if (!client.connected()) {
reconnect();
}
```

```
client.loop();

/*Serial.print("Presion = ");
Serial.print(MySensorBMP.readPressure());
Serial.println(" Pa");
// La medida de una atmósfera de presión bajo condiciones estándar. Equivale a  1,013.25 milibares,
// 29.92 pulgadas de mercurio, 760 milímetros de mercurio, 14.7 libras por pulgadas cuadradas ó 1.033
// gramos por centímetro cuadrado.

// Calcula la altitud tomando como referencia la presion estandar barometrica
// Presion => 1013.25 milibar = 101325 Pascal
Serial.print("Altitud = ");
Serial.print(MySensorBMP.readAltitude());
Serial.println(" metros");

Serial.print("Presión a nivel del mar (calculado) = ");
Serial.print(MySensorBMP.readSealevelPressure());
Serial.println(" Pa");
// Es posible obtener una medición más exacta de la altura
// si se conoce el valor de la presión barométrica
// en tu ubicación por ejemplo asumamos que el valor es de
// 1012,5 milibares que es 101250 Pascales 6569993.929

Serial.print("Altitud real = ");
Serial.print(MySensorBMP.readAltitude(6569994));
Serial.println(" metros");
```

```
delay(2500);*/

long tiempo = millis();
if (tiempo - lastMsg > 500); {
lastMsg = tiempo;
}



messageData = "";
if (tiempo - tdhtx > 2000) {
tdhtx - tiempo;

//-----------LECTURA DE SENSORES ----------------------------------------

float h = dht.readHumidity();
float t = dht.readTemperature();
//    float p = bmp.readPressure();



if (isnan(h) || isnan(t)) {
Serial.println(F("Failed to read from DHT sensor!"));
return;
}

sprintf (data_temp, "%3.2f", t);
client.publish("temperatura", data_temp);
Serial.print("Temperatura: ");
Serial.println(data_temp);
```

```
sprintf (data_humi, "%3.2f", h);
client.publish("humedad", data_humi);
Serial.print("Humedad: ");
Serial.println(data_humi);

/*sprintf (data_pres, "%3.2f", p);
client.publish("presion", data_pres);
Serial.print("Presion: ");
Serial.println(data_pres);*/


lcd.clear();
lcd.setCursor(0,0);
lcd.print("Temp: ");
lcd.print(t);
/*lcd.print("Pre: ");
lcd.print(p);*/
lcd.setCursor(0,1);
lcd.print("Hum:   %");
lcd.print(h);

}
}
```

3.5 ESTRUCTURA GENERAL DE LA APLICACIÓN EN NODE- RED

En esta sección de la memoria se pretende explicar principalmente, la lógica utilizada para unir los nodos que componen el *backend* de la aplicación y como queda reflejada esta composición, en la interfaz gráfica del programa.

La aplicación agrupa en dos opciones las acciones que controlan las funciones de los sensores. Las opciones creadas son las siguientes:

Para ingresar a NODE RED se apunta al ip del bróker y una vez allí se conectará a través del puerto 1880.

Una vez allí se debe descargar los complementos del programa, el *node-red-dashboard* como se ve en la Figura 38, para así poder usar los gadgets y posteriormente programar de manera visual, uniendo los nodos de los gadgets y conectarse con el Bróker.

Este complemento habilitará la paleta del dashboard y se podrá usar sin problemas, porque con ellas se realizará el diseño del proyecto como se ve en la Figura 39.

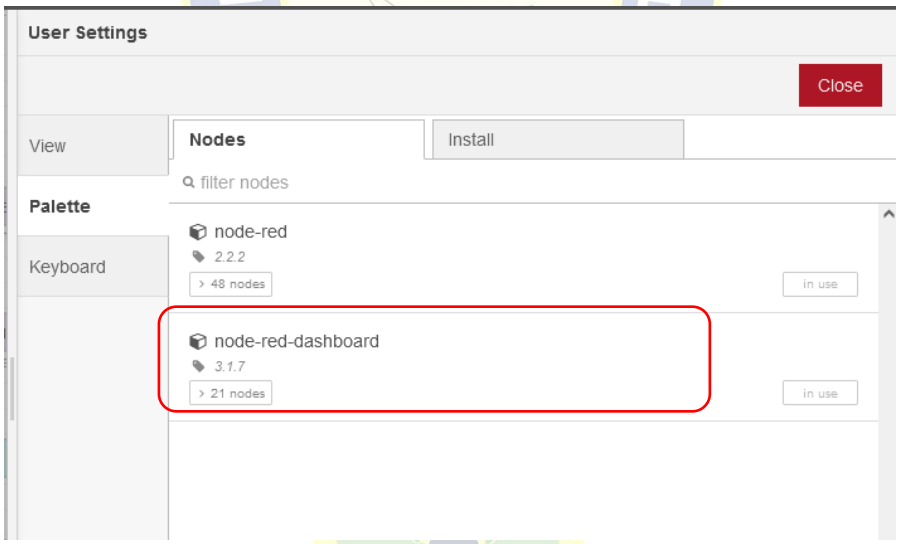


Figura 38 Descargando mi Dashboard (Fuente: T. Belmonte)

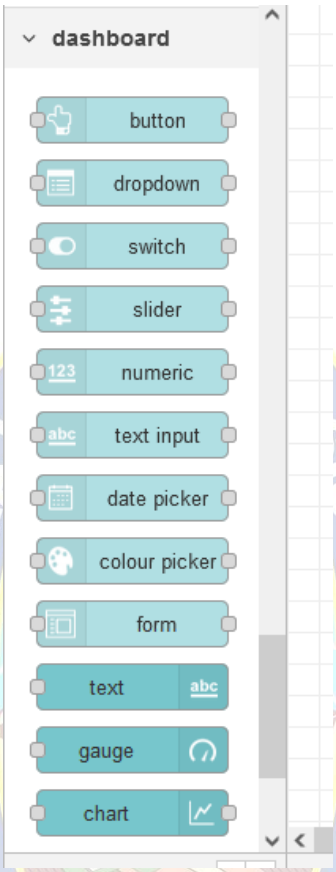


Figura 39 Habilitación del dashboard (Fuente: T. Belmonte)

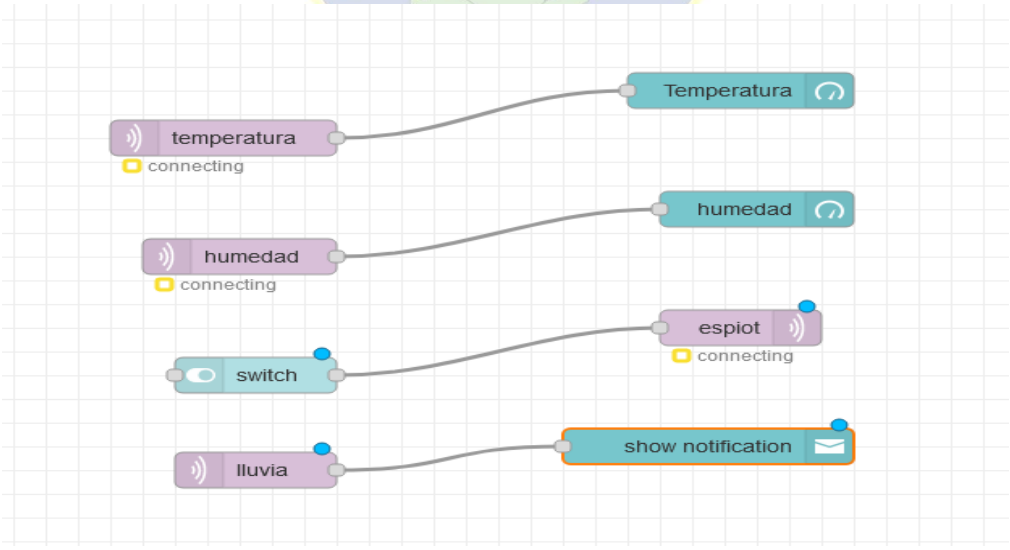


Figura 40 Dashboard de NODE-RED (Fuente: T. Belmonte)

Como se puede advertir en la Figura 40 del entorno de NODE RED la programación es más visual y esta es una de sus grandes ventajas y lo hace muy útil y sencillo de utilizar. Una vez allí podemos realizar las conexiones entre nodos y agrupándolas por grupos dentro el ***flow*** abierto y la configuración de los ***layout.*** La Figura 41 la Figura 43 muestra la configuración del bróker y la configuración de los nodos, y la Figura 42 muestra un diseño más elaborado

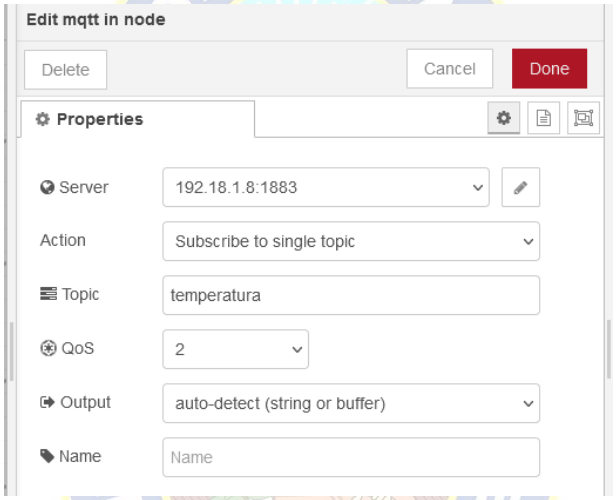


Figura 41 Configurando el Bróker ((Fuente: T. Belmonte)

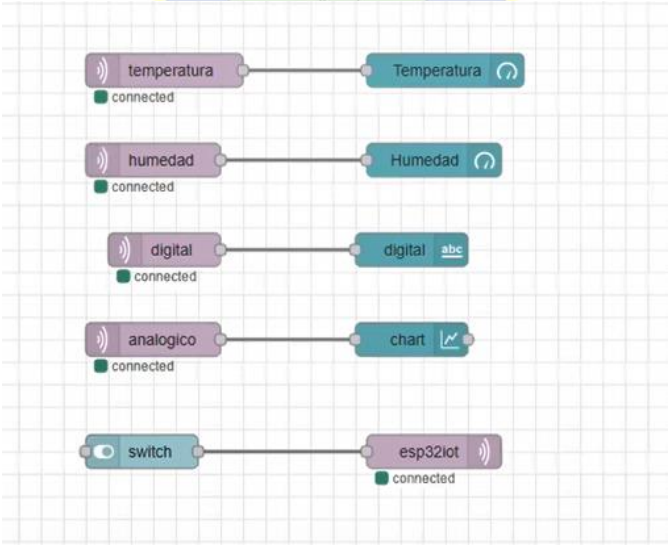


Figura 42 Otro diseño en NODE-RED (Fuente: https://lh6.googleusercontent.comp)

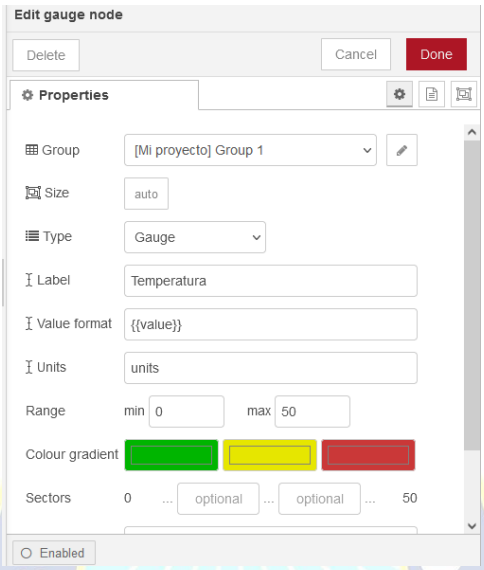


Figura 43 Configuración de los nodos (Fuente: T. Belmonte)

Realizada todas las conexiones dentro del NODE RED lo guardamos y lo corregimos si hay algún error, presionamos ***Deploy*** para luego ir al entorno gráfico.

En la Figura 44 se puede ver el entorno creado por node-red personalizado.

Figura 44 El interfaz dashboard del proyecto (Fuente: T. Belmonte)

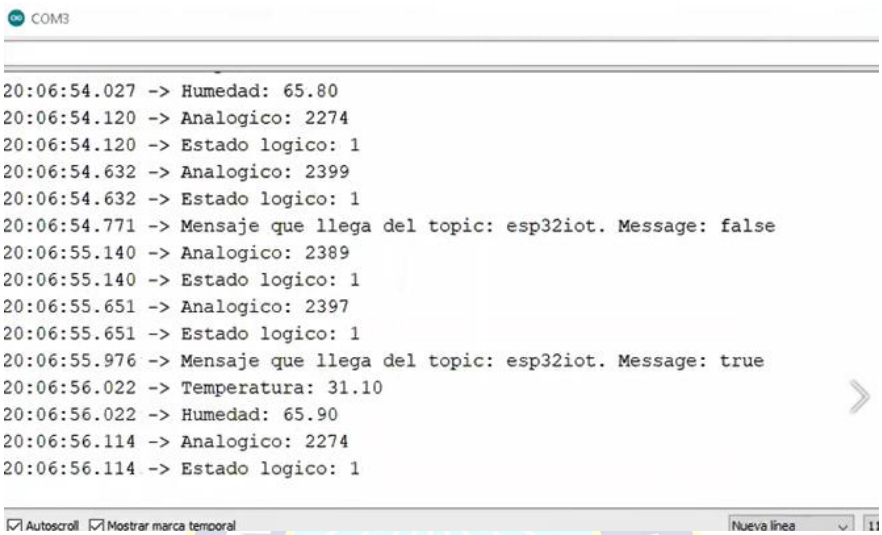


Figura 45 Verificación con el monitor serial del IDE (Fuente: T. Belmonte)

En la Figura 45 se puede ver los resultados de las mediciones mostrados en el monitor serial del IDE de Arduino, verificando que ambos resultados sean el mismo.

3.6 REALIZANDO LAS RESPECTIVAS MEDICIONES Y PRUEBAS.

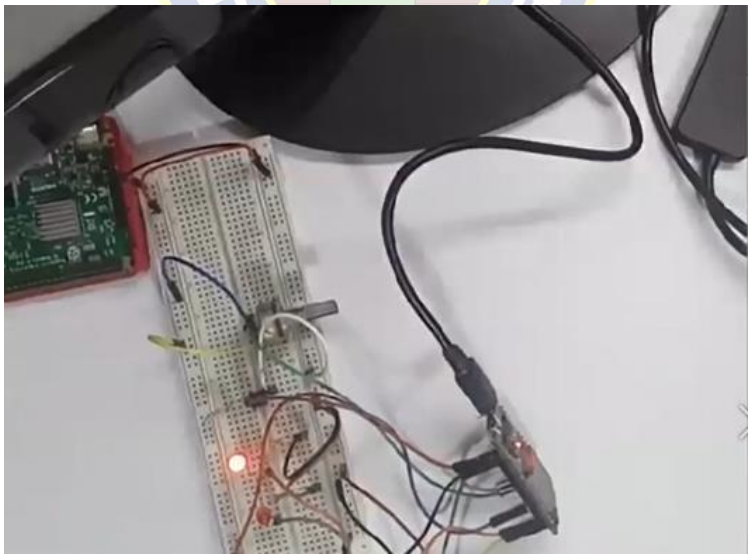


Figura 46 Pruebas del ESP32 (Fuente: T. Belmonte)

Figura 47 Cableado de los sensores (Fuente: T. Belmonte)

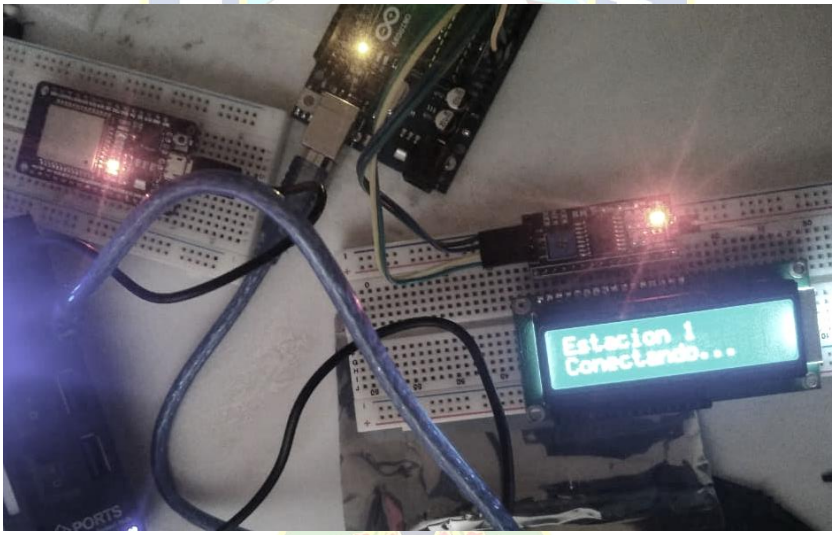


Figura 48 Funcionamiento (Fuente: T. Belmonte)

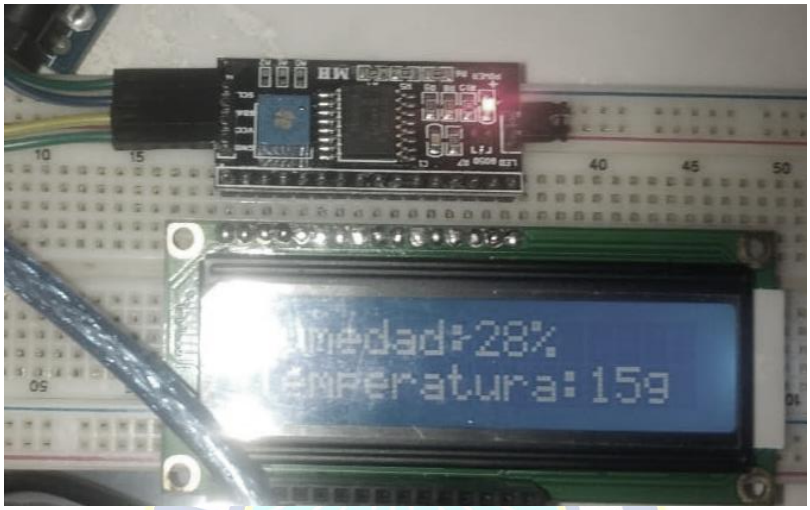


Figura 49 Mediciones (Fuente: T. Belmonte)

La estructura general del trabajo consta de varias partes. Hay una aplicación Node-RED que controla las salidas del ESP32 y recibe las lecturas del sensor conectado al ESP32, usando el protocolo de comunicación MQTT. Esta aplicación Node-RED se muestra de manera visual y puede ser observada direccionando al bróker o servidor.

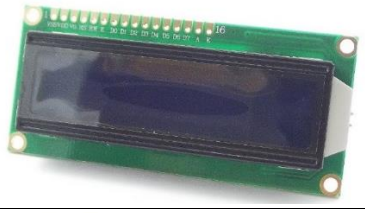
CAPÍTULO IV

ANÁLISIS DE COSTOS

Para este análisis de manera general se describirá, el costo de los todos los equipos (sensores y módulos), la estructura del programa y todo el software empleados durante todo el proceso de implementación de la estación meteorológica.

4.1 COSTO FIJOS

Para realizar el presente proyecto se utilizó los siguientes componentes, sensores y módulos. El proyecto del diseño funcional del mismo es el software con el hardware que se puede encontrar en el mercado de acuerdo a la tabla 7**.**

SENSOR/ MODULO	IMAGEN	COSTO (Bs.)
ESP32		90
LCD 16X2		30
I2C SERIAL		25

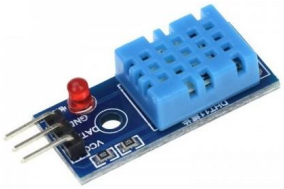
YL-38		15
BMP120		25
DHT11		30
	TOTAL	**(BS.) 215**

Tabla 7 Cotización de sensores y módulos (Fuente: T. Belmonte)

4.2 COSTOS VARIABLES

Respecto a los costos variables están dependerán al tipo de estructura que se desarrollara, de acuerdo a la cantidad de procesos y puntos de tomas de datos y los relacionados con el lugar de control además de otros factores como la variación de; marcas y/o modelos de los módulos y sensores, mantenimiento del sistema y otros.

Sin duda el proyecto ayudara en gran manera a gestionar las condiciones necesarias y por lo tanto variará de acuerdo a la precisión y al tipo de sensor que se desee incorporar.

CAPÍTULO V

CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

Al comienzo de este proyecto se proponía el desarrollo de un sistema capaz de monitorear diferentes parámetros físicos de medición ambiental y de realizar acciones relevantes en el área de control especifico. Tras la finalización del trabajo se ha conseguido realizar lo siguiente:

- En este proyecto se ha establecido la conexión entre el Bróker o servidor y los puntos de control realizada por los suscriptores y publicadores dentro el entorno de Node-Red, usando el protocolo MQTT.
- En el diseño se ha conseguido un sistema escalable y de fácil implementación, tanto el controlador ESP32, los sensores con sus complementos y los componentes electrónicos para hacer las pruebas, han sido conectados.
- Se puede acceder de manera muy sencilla a la aplicación web en Node-RED desde cualquier dispositivo con conexión a internet.
- Es fácilmente escalable ya que se puede ampliar el número de nodos sensores con el único límite de las direcciones IPs disponibles en la red, puesto que se puede reutilizar el mismo código para programar todos los controladores.
- El proyecto en sí, abre un abanico de posibilidades, pudiendo hacer que el mismo pueda ser desarrollado con mayor complejidad de acuerdo al uso que se le quiera dar dentro de lo requerido.
- Se puede afirmar que el sistema ofrece una gran versatilidad, ya que su funcionamiento es fácilmente extrapolable a otros ecosistemas como oficinas, escuelas, hospitales o cualquier otro centro en el cual se esté interesado instalar un sistema de monitorización.

5.2 RECOMENDACIONES

Después de realizar este proyecto debemos realizar las siguientes recomendaciones

- Debemos tener cuidado en restablecer archivos por defecto y realizar los cambios que no afecten tanto al entorno de desarrollo.
- El sistema funcionara siempre y cuando tenga conexión fiable a la nube internet, por eso se debe tener un buen acceso mediante las tecnologías que se proporciona en el mercado.
- Los accesos dentro del entorno de desarrollo que presenta Node-Red serán fiables siempre y cuando se realice la configuración adecuada, por grupos y nodos manteniendo la jerarquía, para un buen funcionamiento.
- Al configurar el Bróker dentro de Node-red se debe apuntar al IP del servidor o no se conectará.
- Es posible que al ingresar a la aplicación no le permita ingresar, esto se debe a que en una conexión wifi la IP cambia porque está configurado en modo dinámico, es necesario configurar de manera estática.
- Si el proyecto se desarrolla con más nodos, puntos de acceso e incluso con más puntos de control, es necesario contar con un bróker robusto que le permita mantener la conexión y que los datos se administren de manera más real.

CAPÍTULO V

BIBLIOGRAFÍA

---

[1]. Llamas, L. ¿Qué es MQTT? Su importancia como protocolo IoT. En Zona Geek. https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot. Recuperado el 23 de mayo de 2022.

[2]. Explicación de TI: MQTT (n. d.). https://www.paessler.com/es/it-explained/mqtt

[3]. Issac. A. MQTT: un protocolo abierto y su importancia en el IoT. En Hardware Libre. https://www.hwlibre.com/mqtt/#Protocolos_de_red. Recuperado el 24 de mayo de 2022.

[4]. Guerra, J. ESP32 wifi y bluetooth en un solo chip. En Programar Fácil. https://programarfacil.com/esp8266/esp32. Recuperado el 24 de junio de 2022

[5]. ESP32 Wifi (n. d.). https://naylampmechatronics.com

[6]. Node-Red (n. d.). https://www.nodered.org

[7]. Arduino ESP32. En Espressif. https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html. Recuperado el 26 de junio de 2022.

[8]. Ingeniero en casa (31 de agosto de 2019). Estación Meteorológica con think speack [video]. https://www.youtube.com/channel/UCurfoG9YbFyQmxCLlxZVArg

[9]. Equipo HiveMQ (20 de abril de 2015). Fundamentos de seguridad de MQTT. https://www.hivemq.com/blog/introducing-the-mqtt-security-fundamentals.

[10]. Ingenieros en casa (28 de agosto de 2019). Programar ESP32 con Arduino IDE [video]. https://youtu.be/wVRcAMWvWko

[11]. W&T conecta; interface para TCP/IP (n.d). MQTT: Comunicación en el internet de las cosas. https://www.wut.de/e-577ww-05-apes-000.php

[12]. Aprendiendo arduino (n.d.). Protocolo MQTT. https://aprendiendoarduino.wordpress.com/tag/mosquitto/

[13]. Nodejs. (n. d.). https://nodejs.org/es

[14]. Tomasi, W. 2003, Sistemas De Comunicaciones Electrónicas, Cuarta Edición, Ediciones Pearson

[15]. Sever Spanulescu, 2020, ESP32 Programación para Internet de las cosas, Segunda Edición, Edición Kindle

[16]. Gastón C. Hillar, 2017, Primera Edición, MQTT Essentials: un protocolo ligero de IoT, Edición Kindle

[17]. Sánchez, C. (24 de enero de 2020). Estructura Normas APA (7ma edición). https://normas-apa.org/estructura/

ANEXOS

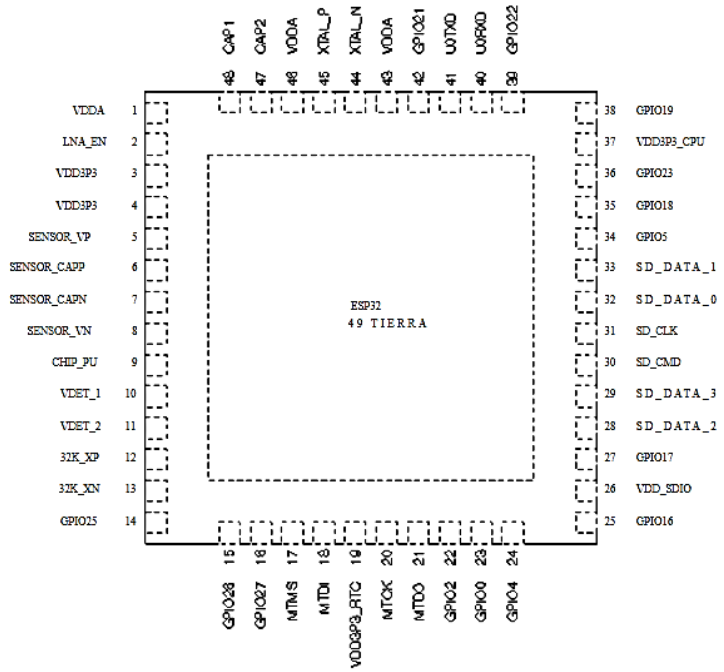
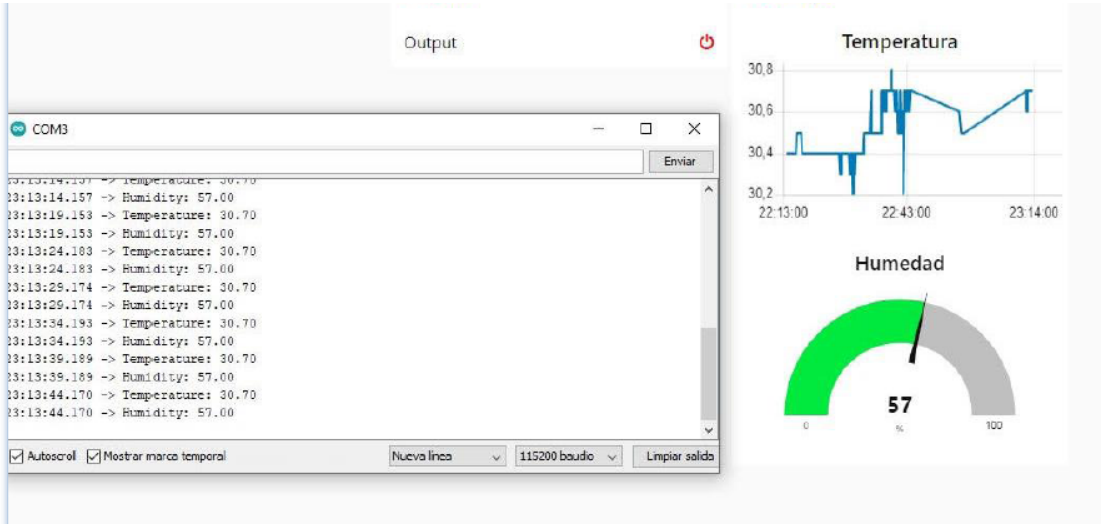


Figura 3: Diseño de clavijas ESP32 (QFN 5*5, vista superior)



Name	No.	Type	Function
Analog			
VDDA	1	P	Analog power supply (2.3 V – 3.6 V)
LNA_IN	2	I/O	RF input and output
VDD3P3	3	P	Analog power supply (2.3 V – 3.6 V)
VDD3P3	4	P	Analog power supply (2.3 V – 3.6 V)
VDD3P3_RTC			
SENSOR_VP	5	I	GPIO36, ADC1_CH0, RTC_GPIO0
SENSOR_CAPP	6	I	GPIO37, ADC1_CH1, RTC_GPIO1
SENSOR_CAPN	7	I	GPIO38, ADC1_CH2, RTC_GPIO2
SENSOR_VN	8	I	GPIO39, ADC1_CH3, RTC_GPIO3
CHIP_PU	9	I	High: On; enables the chip Low: Off; the chip powers off Note: Do not leave the CHIP_PU pin floating.

Name	No.	Type	Function
VDET_1	10	I	GPIO34, ADC1_CH6, RTC_GPIO4
VDET_2	11	I	GPIO35, ADC1_CH7, RTC_GPIO5
32K_XP	12	I/O	GPIO32, ADC1_CH4, RTC_GPIO9, TOUCH9, 32K_XP (32.768 kHz crystal oscillator input)
32K_XN	13	I/O	GPIO33, ADC1_CH5, RTC_GPIO8, TOUCH8, 32K_XN (32.768 kHz crystal oscillator output)
GPIO25	14	I/O	GPIO25, ADC2_CH8, RTC_GPIO6, DAC_1, EMAC_RXD0
GPIO26	15	I/O	GPIO26, ADC2_CH9, RTC_GPIO7, DAC_2, EMAC_RXD1
GPIO27	16	I/O	GPIO27, ADC2_CH7, RTC_GPIO17, TOUCH7, EMAC_RX_DV
MTMS	17	I/O	GPIO14, ADC2_CH6, RTC_GPIO16, TOUCH6, EMAC_TXD2, HSPICLK, HS2_CLK, SD_CLK, MTMS
MTDI	18	I/O	GPIO12, ADC2_CH5, RTC_GPIO15, TOUCH5, EMAC_TXD3, HSPIQ, HS2_DATA2, SD_DATA2, MTDI
VDD3P3_RTC	19	P	Input power supply for RTC IO (2.3 V – 3.6 V)
MTCK	20	I/O	GPIO13, ADC2_CH4, RTC_GPIO14, TOUCH4, EMAC_RX_ER, HSPID, HS2_DATA3, SD_DATA3, MTCK
MTDO	21	I/O	GPIO15, ADC2_CH3, RTC_GPIO13, TOUCH3, EMAC_RXD3, HSPICS0, HS2_CMD, SD_CMD, MTDO
GPIO2	22	I/O	GPIO2, ADC2_CH2, RTC_GPIO12, TOUCH2, HSPIWP, HS2_DATA0, SD_DATA0
GPIO0	23	I/O	GPIO0, ADC2_CH1, RTC_GPIO11, TOUCH1, EMAC_TX_CLK, CLK_OUT1,
GPIO4	24	I/O	GPIO4, ADC2_CH0, RTC_GPIO10, TOUCH0, EMAC_TX_ER, HSPIHD, HS2_DATA1, SD_DATA1

VDD_SDIO			
GPIO16	25	I/O	GPIO16, HS1_DATA4, U2RXD, EMAC_CLK_OUT
VDD_SDIO	26	P	Output power supply: 1.8 V or the same voltage as VDD3P3_RTC
GPIO17	27	I/O	GPIO17, HS1_DATA5, U2TXD, EMAC_CLK_OUT_180
SD_DATA_2	28	I/O	GPIO9, HS1_DATA2, U1RXD, SD_DATA2, SPIHD
SD_DATA_3	29	I/O	GPIO10, HS1_DATA3, U1TXD, SD_DATA3, SPIWP
SD_CMD	30	I/O	GPIO11, HS1_CMD, U1RTS, SD_CMD, SPICS0
SD_CLK	31	I/O	GPIO6, HS1_CLK, U1CTS, SD_CLK, SPICLK
SD_DATA_0	32	I/O	GPIO7, HS1_DATA0, U2RTS, SD_DATA0, SPIQ
SD_DATA_1	33	I/O	GPIO8, HS1_DATA1, U2CTS, SD_DATA1, SPID
VDD3P3_CPU			
GPIO5	34	I/O	GPIO5, HS1_DATA6, VSPICS0, EMAC_RX_CLK
GPIO18	35	I/O	GPIO18, HS1_DATA7, VSPICLK
GPIO23	36	I/O	GPIO23, HS1_STROBE, VSPID
VDD3P3_CPU	37	P	Input power supply for CPU IO (1.8 V – 3.6 V)
GPIO19	38	I/O	GPIO19, U0CTS, VSPIQ, EMAC_TXD0
GPIO22	39	I/O	GPIO22, U0RTS, VSPIWP, EMAC_TXD1
U0RXD	40	I/O	GPIO3, U0RXD, CLK_OUT2
U0TXD	41	I/O	GPIO1, U0TXD, CLK_OUT3, EMAC_RXD2
GPIO21	42	I/O	GPIO21, VSPIHD, EMAC_TX_EN

Analog			
VDDA	43	P	Analog power supply (2.3 V – 3.6 V)
XTAL_N	44	O	External crystal output
XTAL_P	45	I	External crystal input
VDDA	46	P	Analog power supply (2.3 V – 3.6 V)
CAP2	47	I	Connects to a 3 nF capacitor and 20 kΩ resistor in parallel to CAP1
CAP1	48	I	Connects to a 10 nF series capacitor to ground
GND	49	P	Ground

Note:

- ESP32-D2WD's pins GPIO16, GPIO17, SD_CMD, SD_CLK, SD_DATA_0 and SD_DATA_1 are used for connecting the embedded flash, and are not recommended for other uses.
- For a quick reference guide to using the IO_MUX, Ethernet MAC, and GIPO Matrix pins of ESP32, please refer to Appendix ESP32 Pin Lists.
- In most cases, the data port connection between the ESP32 and external flash is as follows: SD_DATA0/SPIQ = IO1/DO, SD_DATA1/SPID = IO0/DI, SD_DATA2/SPIHD = IO3/HOLD#, SD_DATA3/SPIWP = IO2/WP#.

Tabla 8 Pinout Datasheet ESP32 (www.alldatasheet.com)