

2023

# PROTOTIPOS IOT PARA RASPBERRY

MARIAM GARCÍA MARTÍN  
PERE GRANE ALQUEZAR

## Contenido

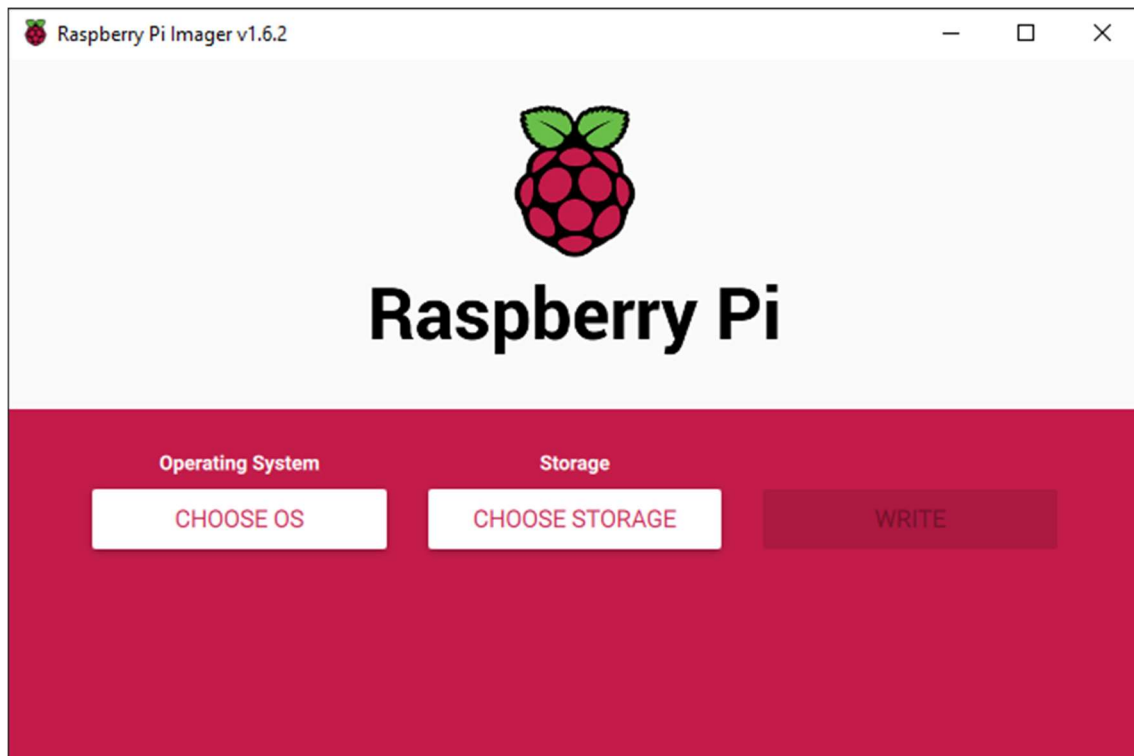
• INTRODUCCION:.....	2
• INSTALACION DEL SISTEMA OPERATIVO:.....	2
• CONFIGURACION DE RASPBERRY PI COMO ACCESS POINT WIFI.....	6
Preparación: .....	6
Configuración de red:.....	6
Configurar NAT:.....	9
• INSTALACION DE NODE-RED: .....	11
• INSTALACION DEL BROKER MQTT MOSQUITO: .....	13
¿Qué es MQTT? .....	13
Como instalar y configurar Mosquitto: .....	14
• INSTALAR ARDUINO IDE EN WINDOWS: .....	15
• CREACION DE UN FLOW MQTT EN NODE-RED: .....	17
Código del cliente ESP32: .....	30
• BRUJULA QMC5883L .....	34
Programación: .....	34
Explicación del código: .....	35
• PLACA SOLAR:.....	37
Paneles solares:.....	37
Módulo de cargador TP4056.....	38
Regulador de voltaje: .....	39
Diodo Schottky .....	40
Circuito de Monitoreo de Nivel de Tensión de Batería.....	40
Consumo de energía del módulo ESP32 WROOM .....	41
• CONCLUSIONES: .....	41
• BIBLIOGRAFIA:.....	42

- INTRODUCCION:

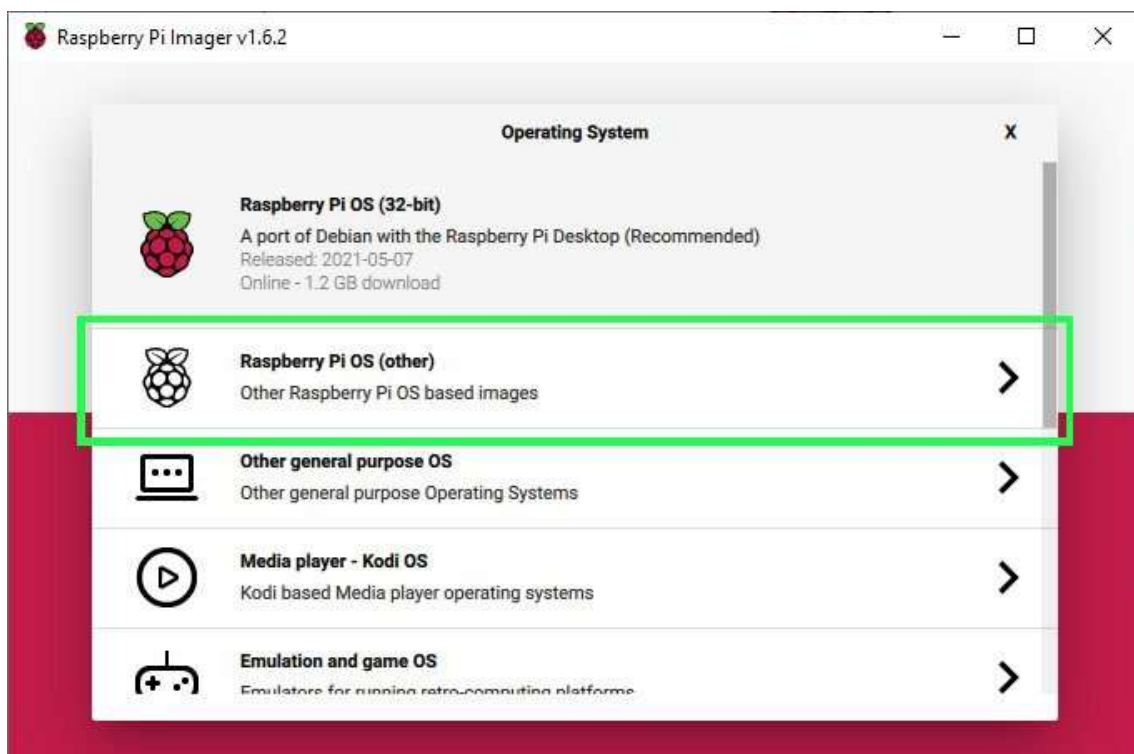
En el fascinante mundo de la Internet de las Cosas (IoT), nos complace presentar nuestro proyecto que integra de manera innovadora los conocimientos adquiridos a lo largo de nuestro curso. Mediante el uso de la versátil Raspberry Pi, hemos explorado las infinitas posibilidades que ofrece esta plataforma para crear soluciones tecnológicas inteligentes y prácticas. Aunque, en un principio, nuestra idea podría haber tomado una dirección diferente, nos complace informar que el resultado final ha superado nuestras expectativas. Este proyecto no solo ha demostrado la versatilidad de la Raspberry Pi, sino también la capacidad de adaptación y creatividad de nuestro equipo. A lo largo de este viaje, hemos enfrentado desafíos, hemos aprendido valiosas lecciones y, sobre todo, hemos logrado construir una solución satisfactoria que refleja nuestro compromiso con la excelencia y la innovación en el campo de la IoT. El núcleo de nuestro proyecto se basa en la integración armoniosa de dispositivos y sensores, utilizando la Raspberry Pi como la columna vertebral de nuestra infraestructura. Desde la recopilación de datos hasta su procesamiento y presentación, cada fase ha sido cuidadosamente diseñada para maximizar la eficiencia y la utilidad de nuestra implementación. Hemos seleccionado cuidadosamente una variedad de componentes para potenciar nuestro proyecto, desde sensores de temperatura y humedad hasta actuadores controlados remotamente. La interconexión de estos elementos, gestionada de manera inteligente por la Raspberry Pi, permite un monitoreo y control de efectivos, ofreciendo una solución integral a los desafíos planteados en nuestro escenario inicial. A lo largo de este emocionante viaje, hemos logrado logros significativos, desde la programación eficiente de la Raspberry Pi hasta la creación de una interfaz de usuario intuitiva. Sin embargo, también hemos enfrentado desafíos que han estimulado nuestro pensamiento creativo y nuestra resolución de problemas, demostrando la esencia misma de la ingeniería y la innovación.

- INSTALACION DEL SISTEMA OPERATIVO:

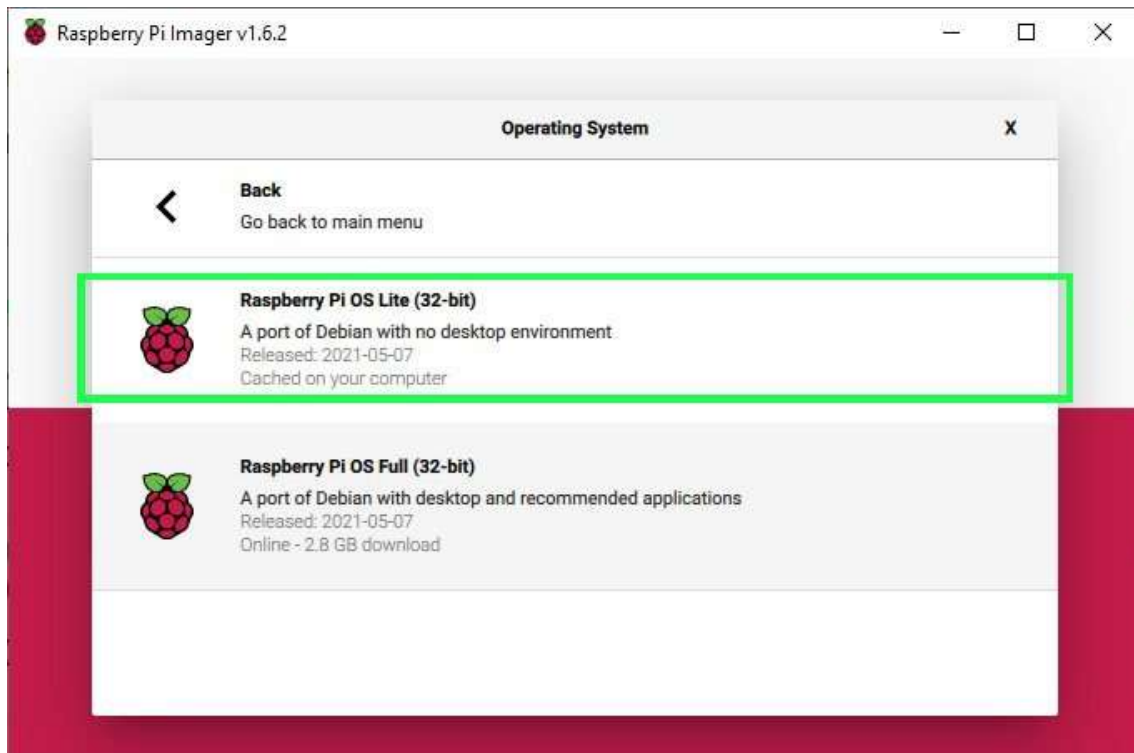
Lo primero que haremos es instalar el sistema operativo base en nuestra Raspberr Pi. Para eso, usaremos el Raspberry Pi Imager oficial que podemos descargar desde su web.



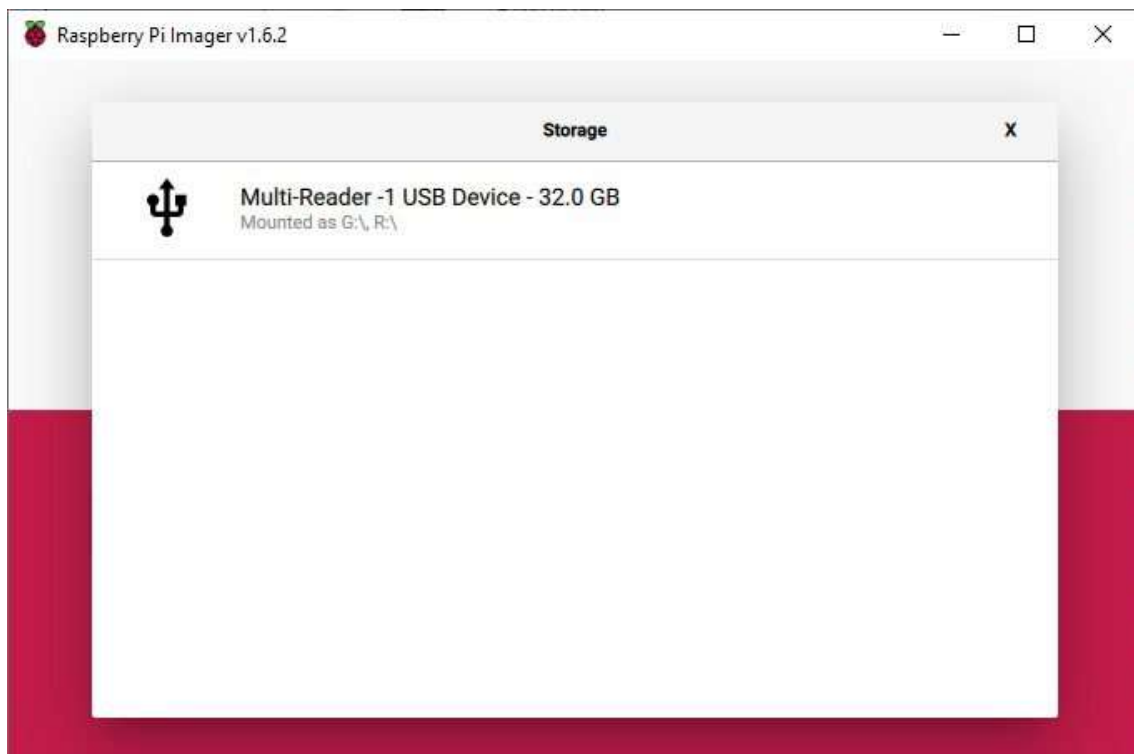
Desde éste programa podemos instalar cualquiera de las versiones disponibles del sistema operativo oficial de la fundación Raspberry. Para eso, solo es necesario hacer clic en el botón **CHOOSE OS** y veremos un cuadro de selección.



Seleccionamos **Raspberry Pi OS (other)** y luego **Raspberry Pi OS Lite (32-bit)**



Luego debemos seleccionar la unidad donde está montada la tarjeta SD.

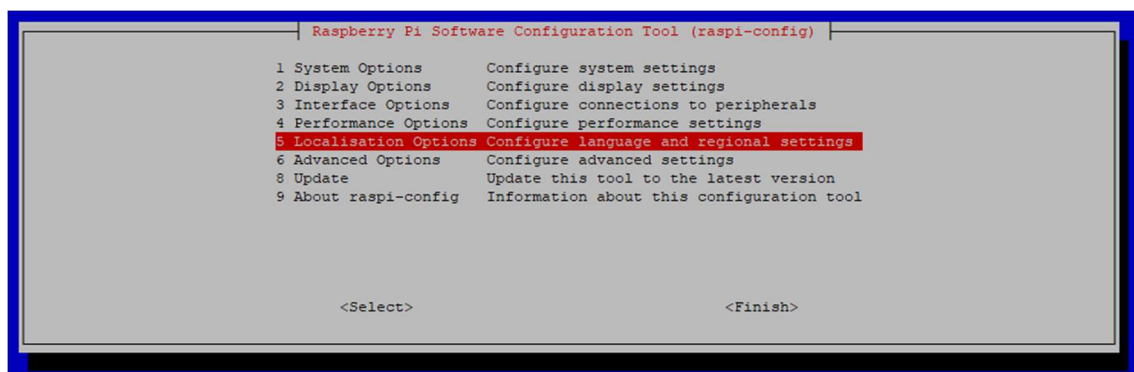
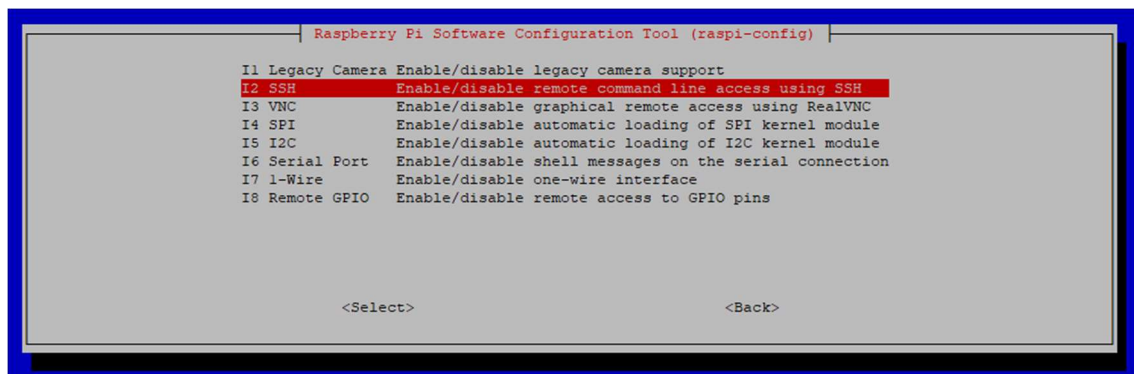
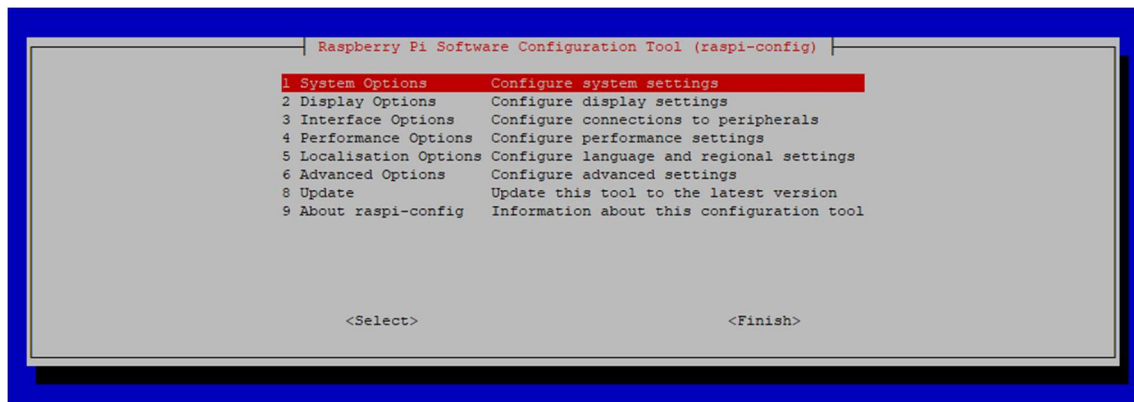


Una vez hemos instalado el sistema operativo, con el comando **raspi-config** podemos configurar nuestra Raspberry Pi sin tener que modificar manualmente los ficheros necesarios.

Esta herramienta nos permitirá, entre otras cosas, cambiar el idioma, la región, el password, etc.

También podemos habilitar y configurar la conexión remota a través de SSH, con lo que convertiremos nuestra Raspberry en un equipo **headless**, es decir, que puede funcionar sin monitor, teclado ni ratón.

### sudo raspi-config



- CONFIGURACION DE RASPBERRY PI COMO ACCESS POINT WIFI

### Preparación:

El adaptador de red inalámbrica que lleva incorporado Raspberry Pi 3B o posteriores puede configurarse para transmitir su SSID, aceptar conexiones WiFi y repartir direcciones IP (utilizando DHCP). Para comprobarlo podemos utilizar el comando `iw list`:

```
Wiphy phy0
  wiphy index: 0
  max # scan SSIDs: 10
  max scan IEs length: 2048 bytes
  max # sched scan SSIDs: 16
  max # match sets: 16
  Retry short limit: 7
  Retry long limit: 4
  Coverage class: 0 (up to 0m)
  Device supports roaming.
  Supported Ciphers:
    * WEP40 (00-0f-ac:1)
    * WEP104 (00-0f-ac:5)
    * TKIP (00-0f-ac:2)
    * CCMP-128 (00-0f-ac:4)
  Available Antennas: TX 0 RX 0
  Supported interface modes:
    * IBSS
    * managed
    * AP
    * P2P-client
    * P2P-GO
    * P2P-device
```

Una vez comprobada la compatibilidad del adaptador de red, será necesario instalar `hostapd` (Host access point daemon) mediante el gestor de paquetes. Este software es necesario para convertir el adaptador de red en un servidor de punto de acceso y autenticación:

**`sudo apt-get install hostapd`**

También será necesario instalar el paquete `dnsmasq` que proporcionará capacidades de servidor de DNS local a la nueva red.

**`sudo apt-get install dnsmasq`**

### Configuración de red:

Agregamos la siguiente configuración al archivo `/etc/dhcpd.conf` para configurar las direcciones IP estáticas en `eth0` y `wlan0`:

**`sudo nano /etc/dhcpd.conf`**

```
# static IP configuration:
interface eth0
static ip_address=192.168.1.154/24
#static ip6_address=fd51:42f8:caae:d92e::ff/64
static routers=192.168.1.1
static domain_name_servers=8.8.8.8. 8.8.4.4

interface wlan0
static ip_address=192.168.5.1/24
static domain_name_servers=8.8.8.8. 8.8.4.4
```

Cambiamos el archivo `/etc/network/interfaces` de la siguiente manera:

**sudo nano /etc/network/interfaces**

```
auto lo
iface lo inet loopback

allow-hotplug eth0
iface eth0 inet manual

allow-hotplug wlan0
iface wlan0 inet manual

iface wlan0 inet static
    address 192.168.5.1
    netmask 255.255.255.0
```

Tenemos que configurar Hostapd para decirle que emita un SSID en particular y permita conexiones WiFi en un determinado canal.

**sudo nano /etc/hostapd/hostapd.conf**

```
interface=wlan0
driver=nl80211
ssid=RaspiVELETA
hw_mode=g
channel=7
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=veletal2345
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```



Desafortunadamente, Hostapd no sabe dónde encontrar este archivo de configuración, por lo que necesitamos proporcionar su ubicación al script de inicio hostapd.

**sudo nano /etc/default/hostapd**

```
#  
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Ahora configuramos Dnsmasq para asignar automáticamente direcciones IP a medida que los nuevos dispositivos se conecten a nuestra red. Por si acaso haremos una copia de seguridad de ese archivo.

**sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.bak**

Y lo editamos:

**sudo nano /etc/dnsmasq.conf**

```
GNU nano 5.4  
interface=wlan0  
dhcp-option=3,192.168.5.1  
dhcp-range=192.168.5.100,192.168.5.200,255.255.255.0,24h
```

Nos aseguramos de reiniciar los servicios después de realizar estas configuraciones:

**sudo service dhcpcd restart**

**sudo service hostapd restart**

**sudo service dnsmasq restart**

Reiniciamos nuestra Raspberry Pi y observamos cómo nos aparece nuestra red inalámbrica.



## Configurar NAT:

El siguiente paso es habilitar el routing con esta configuración para que la Raspberry Pi se comporte como router siendo capaz de conectarlo a otra red a través de Ethernet y hacer que los dispositivos conectados a WiFi puedan hablar con esa red. Al hacer esto, podemos compartir una conexión a Internet del Pi.

Abrimos el archivo de configuración de sysctl para habilitar el reenvío de paquetes IP:

**sudo nano /etc/sysctl.conf**

Dentro del archivo, buscamos la siguiente línea y nos aseguramos de que no esté comentada (sin el signo # al principio de la línea):

```
# Uncomment the next line to enable packet forwarding for IPv4  
net.ipv4.ip_forward=1
```

Para aplicar la configuración sin reiniciar, ejecutamos el siguiente comando:

**sudo sysctl -p /etc/sysctl.conf**

Esto habilitará el reenvío de paquetes IP en tu Raspberry Pi.

Si la red ethernet dispone de acceso a internet a través de un router se podrá habilitar el enmascarado de IPs de la red inalámbrica de modo que el punto de acceso actúe como NAT para dicha red, quitado así la necesidad de reconfigurar el router que da acceso a internet para habilitar a los dispositivos de la nueva red. Para ello se hará uso de una regla de iptables.

**sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE**

Nos aseguramos de que el servidor DHCP en dnsmasq está configurado para proporcionar la dirección IP del Raspberry Pi (192.168.5.1) como puerta de enlace predeterminada a los dispositivos conectados. Esto se establece en el archivo /etc/dnsmasq.conf.

**sudo nano /etc/dnsmasq.conf**

```
GNU nano 5.4
interface=wlan0
dhcp-option=3,192.168.5.1
dhcp-range=192.168.5.100,192.168.5.200,255.255.255.0,24h
```

Finalmente, verificamos que el enrutamiento IP esté habilitado en tu Raspberry Pi ejecutando:

**cat /proc/sys/net/ipv4/ip\_forward**

Si nos muestra "1", indica que el reenvío de paquetes IP está habilitado.

```
pgrane@raspberrypi:~ $ cat /proc/sys/net/ipv4/ip_forward
1
```

Podemos comprobar el estado de los servicios con los siguientes comandos:

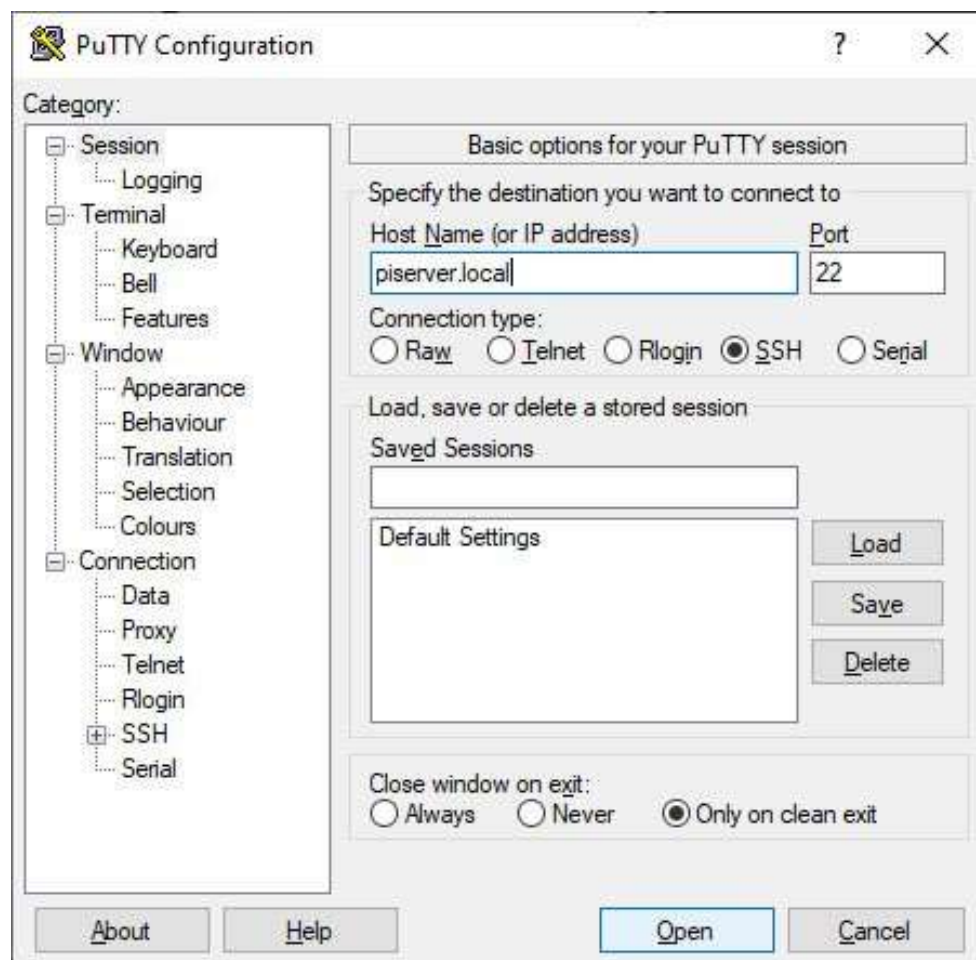
**ip addr show wlan0**

**sudo systemctl status hostapd**

**sudo systemctl status dnsmasq**

- INSTALACION DE NODE-RED:

Lo primero que haremos es conectarnos a nuestra Raspberry Pi mediante SSH usando el programa **PUTTY** que es un cliente SSH gratuito.



Solo hay que poner el nombre de Hostname que hayamos puesto en nuestra Raspberry Pi, o su IP, y nos podremos conectar directamente utilizando el puerto por defecto **22**.

```
pi@piserver: ~  
login as:pi  
pi@piserver's password:  
Linux piserver 5.10.17-v7l+ #1421 SMP Thu May 27 14:00:13 BST 2021 armv7l  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Sun Jul 4 00:33:52 2021 from 192.168.1.168  
pi@piserver:~$
```

Prepararemos el sistema con los paquetes que necesita. Primero vamos a instalar las herramientas de compilación y el soporte Curl:

**sudo apt install build-essential git curl**

Se descargarán los paquetes necesarios y una vez hecho, podemos pasar a instalar Node-Red ejecutando el siguiente comando:

**sudo bash <(curl -sL <https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered>)**

Éste comando realizará de forma automática las siguientes tareas:

1. Quitar eventuales versiones de Node-Red ya instaladas
2. Si detecta Node.js ya instalado, se asegura que la versión sea la v12 o superior.
3. Instala la última versión de Node-Red usando el gestor de paquetes npm
4. Opcionalmente, instala nodos interesantes específicos para Node-red
5. Instala Node-red como servicio del sistema

Ahora podemos iniciar Node-red mediante diferentes comandos que podremos ejecutar desde la consola:

1. node-red-start - Inicia Node-Redy muestra el log de lo que hace. Presionando CTRL C lo podemos parar. Esto lo incia como servicio. Es decir, se ejecuta en segundo plano.

2. `node-red-stop` - Detiene el servicio Node-red
3. `node-red-restart` - Reinicia el proceso de Node-red
4. `node-red-log` - Muestra el log del servicio para localizar posibles fallos

Para que se inicie automáticamente cuando la Raspberry Pi se encienda debemos iniciar el servicio en el arranque del sistema con el siguiente comando:

**`sudo systemctl enable nodered.service`**

De este modo podremos acceder a su panel web mediante la dirección <http://<hostname>:1880>

En "hostname" debemos indicar o bien el nombre del servidor o bien su dirección IP.

- **INSTALACION DEL BROKER MQTT MOSQUITO:**

## ¿Qué es MQTT?

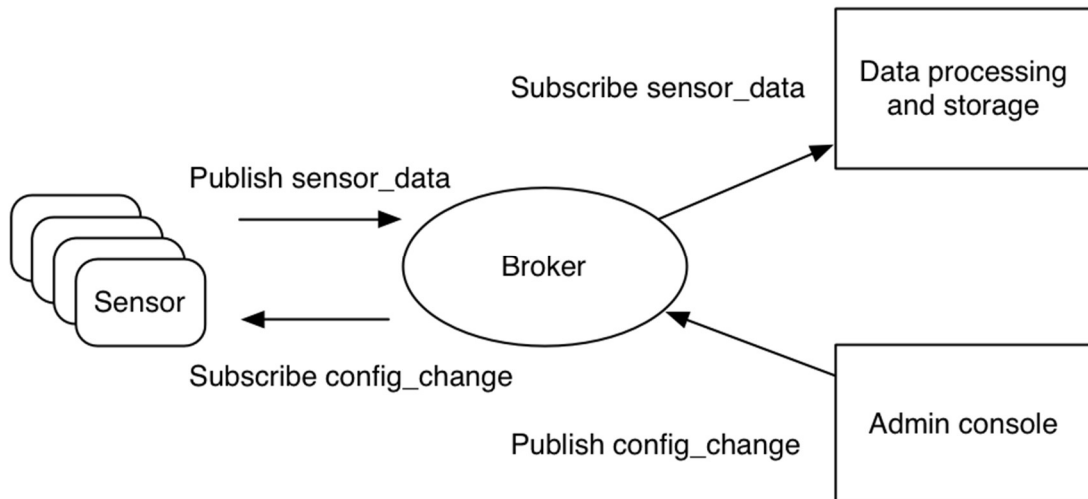
MQTT son las siglas MQ Telemetry Transport, aunque en primer lugar fue conocido como Message Queing Telemetry Transport. Es un protocolo de comunicación M2M (machine-to-machine) de tipo message queue.

Está basado en la pila TCP/IP como base para la comunicación. En el caso de MQTT cada conexión se mantiene abierta y se "reutiliza" en cada comunicación.

El funcionamiento del MQTT es un servicio de mensajería push con patrón publicador/suscriptor (pub-sub) donde los clientes se conectan con un servidor central denominado broker.

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 broker le hará llegar los mensajes suscritos.

Los clientes inician una conexión TCP/IP con el broker, 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.



MQTT aporta una serie de características como su sencillez y ligereza. Esto lo hace adecuado para aplicaciones IoT, donde frecuentemente se emplean dispositivos de escasa potencia. Además, esta menor necesidad de recursos se traduce en un menor consumo de energía. Otra consecuencia de la ligereza del protocolo MQTT es que requiere un ancho de banda mínimo, lo cual es importante en redes inalámbricas, o conexiones con posibles problemas de calidad.

## Como instalar y configurar Mosquitto:

El Broker MQTT más popular es Mosquitto. Es un Broker MQTT Open Source que es compatible con Windows, Linux y Mac.

Lo primero que hacemos es asegurarnos de que tenemos todos los paquetes actualizados.

**sudo apt update**

**sudo apt upgrade**

Una vez hecho esto, pasamos a instalar Mosquitto:

**sudo apt-get install mosquitto mosquitto-clients**

Aquí se descargarán los paquetes necesarios y ya lo tendremos instalado.

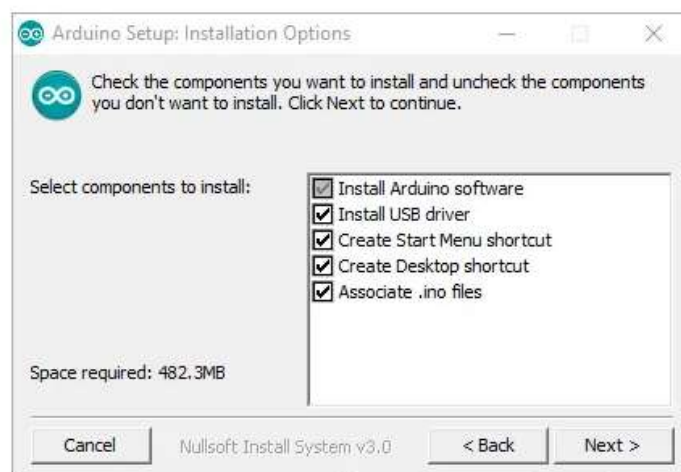
- **INSTALAR ARDUINO IDE EN WINDOWS:**

Lo primero que haremos es descargarnos el programa de su web oficial, y una vez descargado ejecutaremos el instalador.

Aceptamos los términos y condiciones de la licencia.

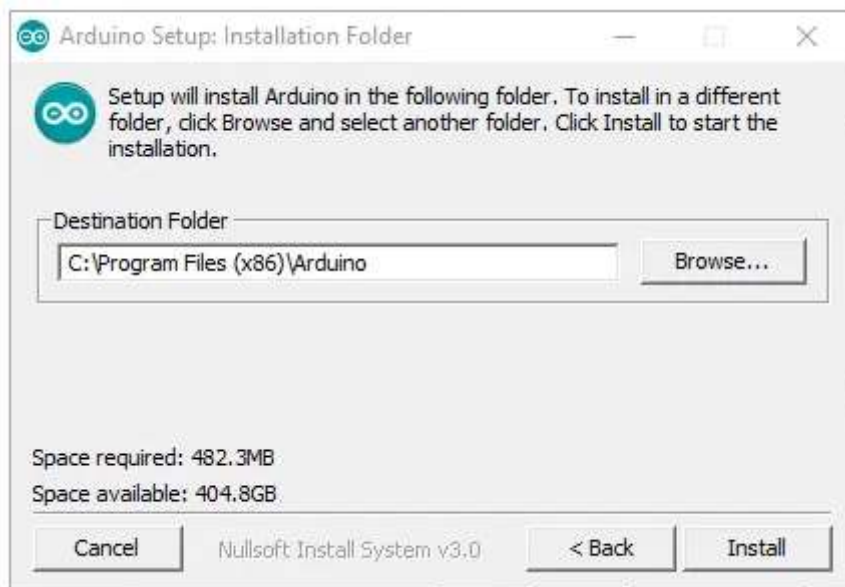


Selecciona todas las opciones para que instale todos los complementos y drivers necesarios.

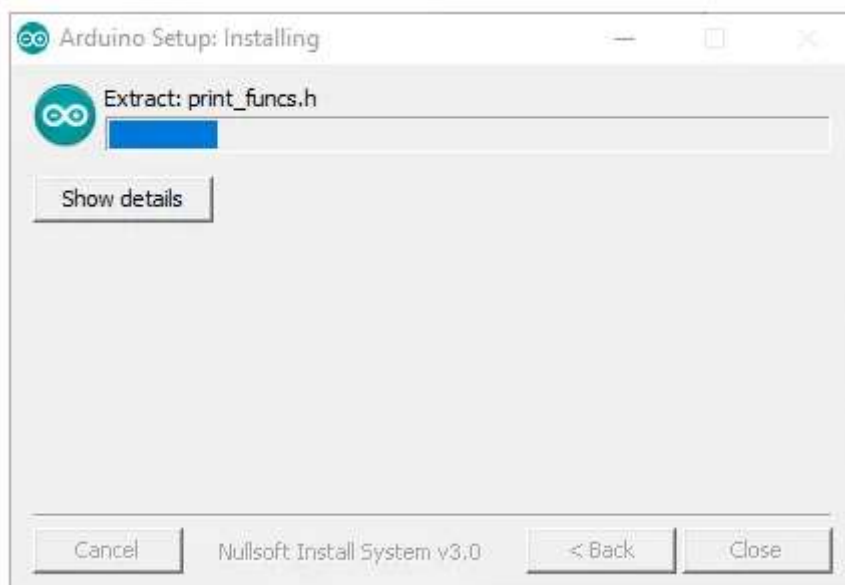


Selecciona la ruta de instalación y presiona “install”.

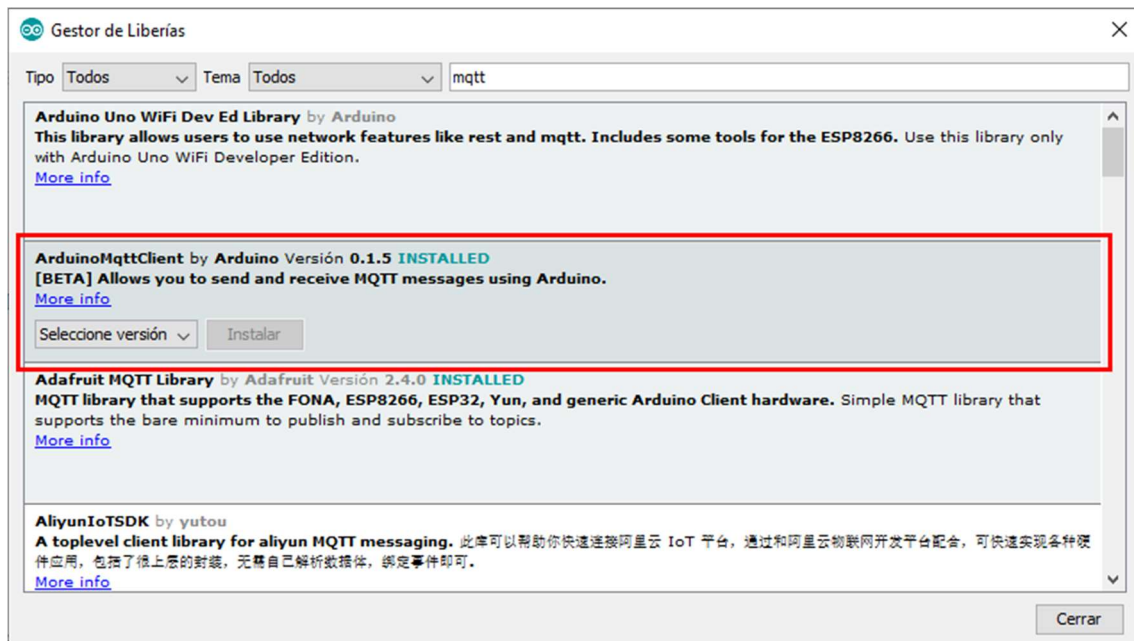




Espera hasta que termine el proceso de instalación.



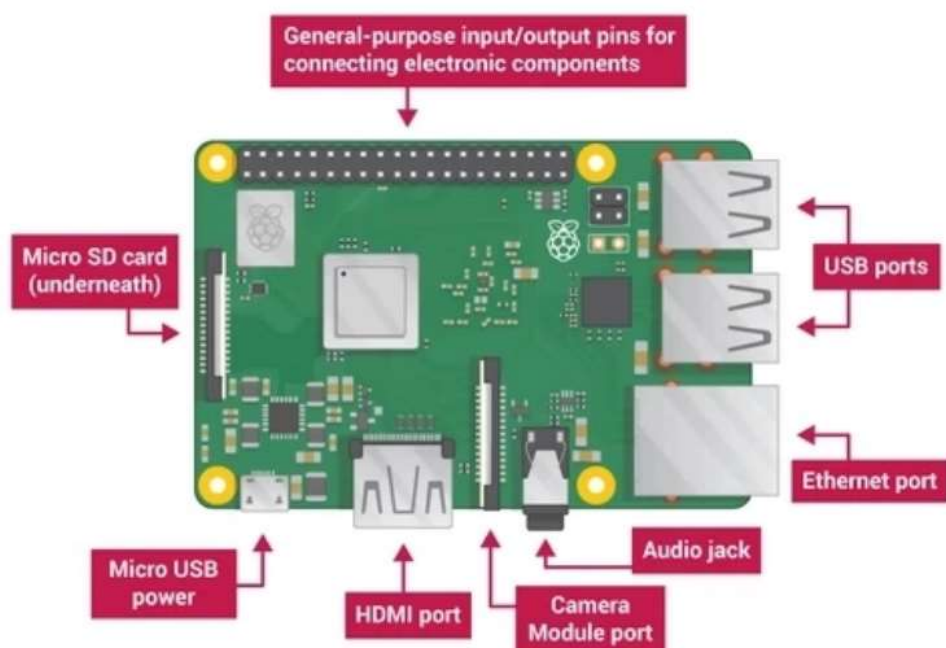
Ahora podremos instalar la librería MQTT desde el gestor de librerías para poder comunicarnos con el servidor MQTT.

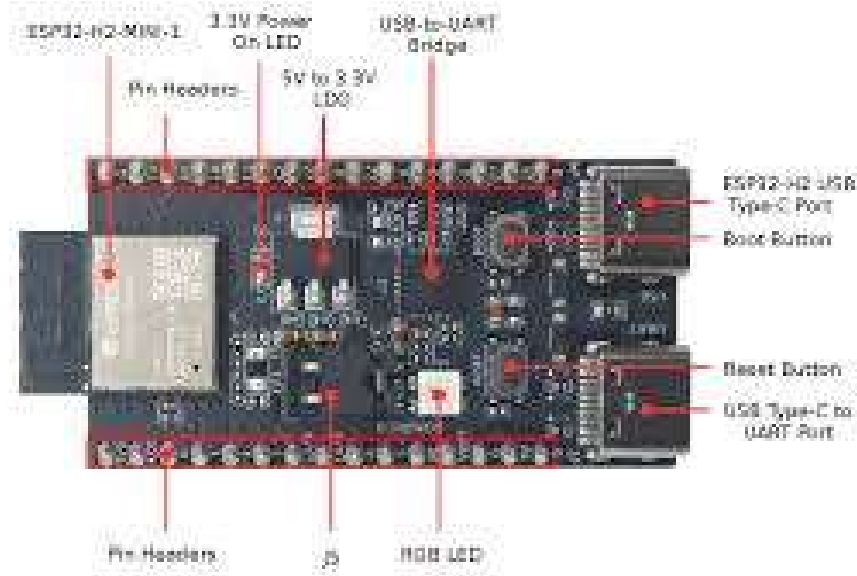


- CREACION DE UN FLOW MQTT EN NODE-RED:

El flow es la parte lógica de Node-Red y que va a manejar los mensajes MQTT entre nuestra Raspberry, que configuraremos como Broker, y otro dispositivo WIFI, un ESP32, que controlará los sensores y actuará de cliente, alimentado por una pila.

**ESP32** es un microcontrolador de la empresa **Espressif Systems**.





Los sensores que conectaremos a nuestro ESP32 son los siguientes:

El **BMP085** Sensor de Presión Atmosférica permite lecturas de presión atmosférica y temperatura a través del bus I2C. Como la presión atmosférica cambia con la altura, también se puede utilizar este sensor como un altímetro.

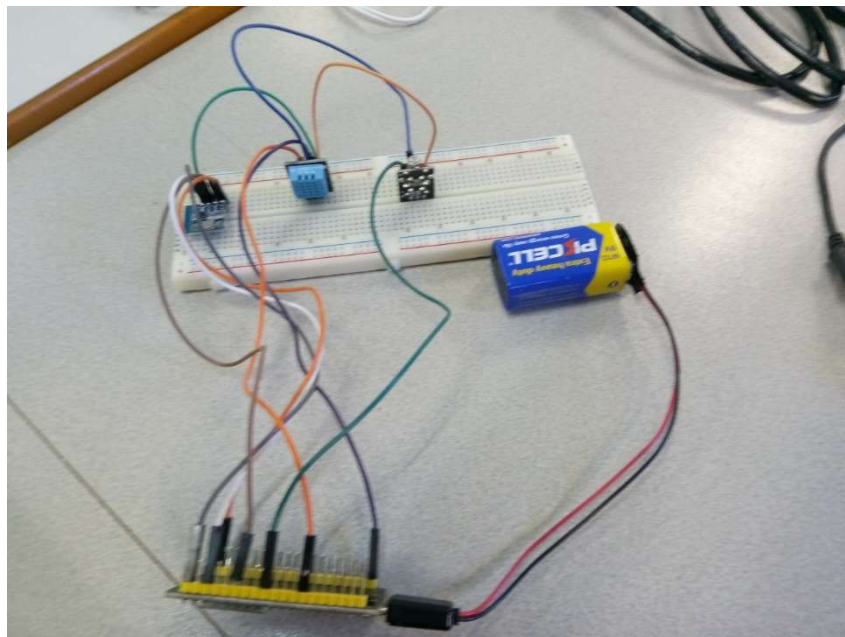
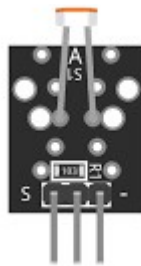


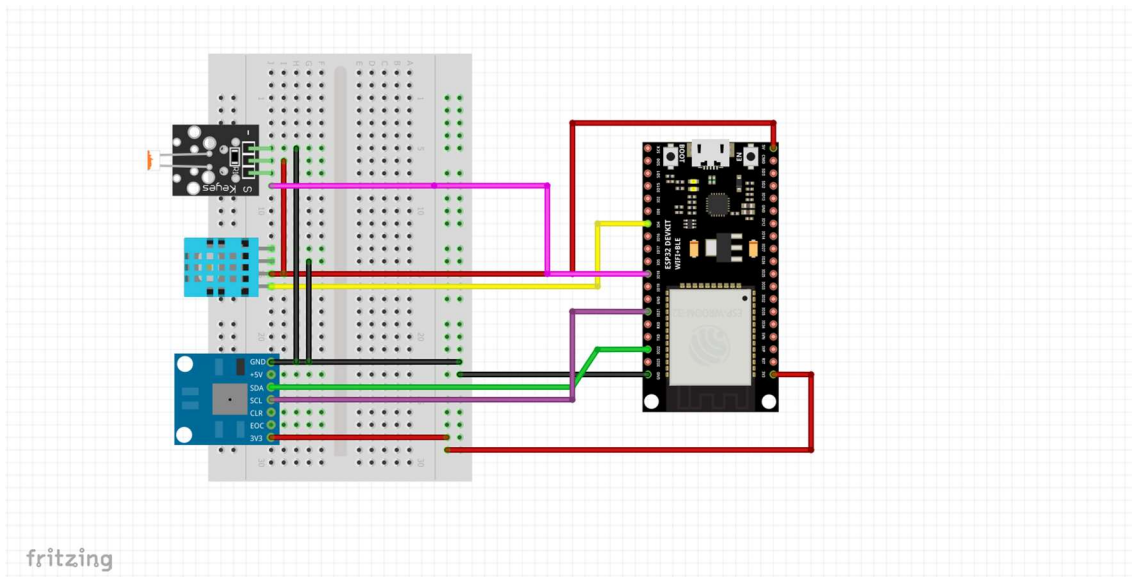
El **DHT11** es un sensor digital de temperatura y humedad relativa de bajo costo y fácil uso. Utiliza un sensor capacitivo de humedad y un termistor para medir el aire circundante, y muestra los datos mediante una señal digital en el pin de datos (no posee salida analógica).

Debido a que el sensor BMP085 también mide la temperatura, usaremos el DHT11 para medir solamente la humedad.

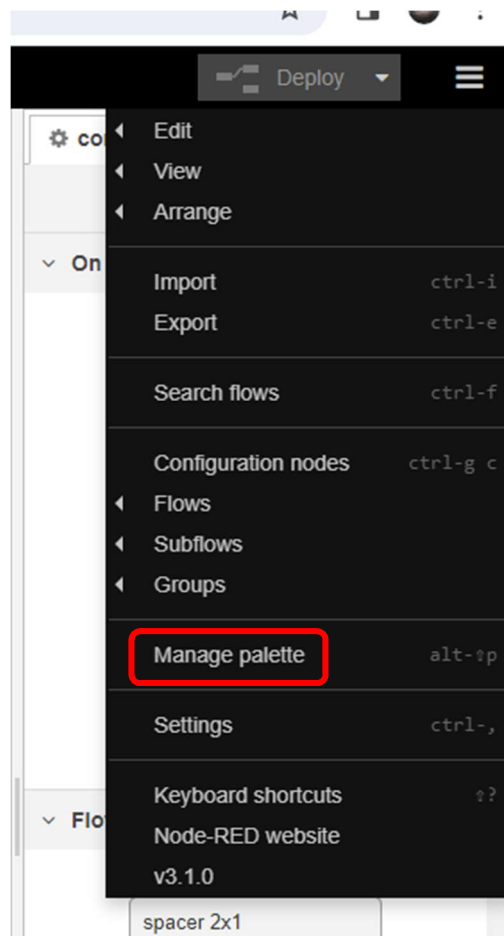


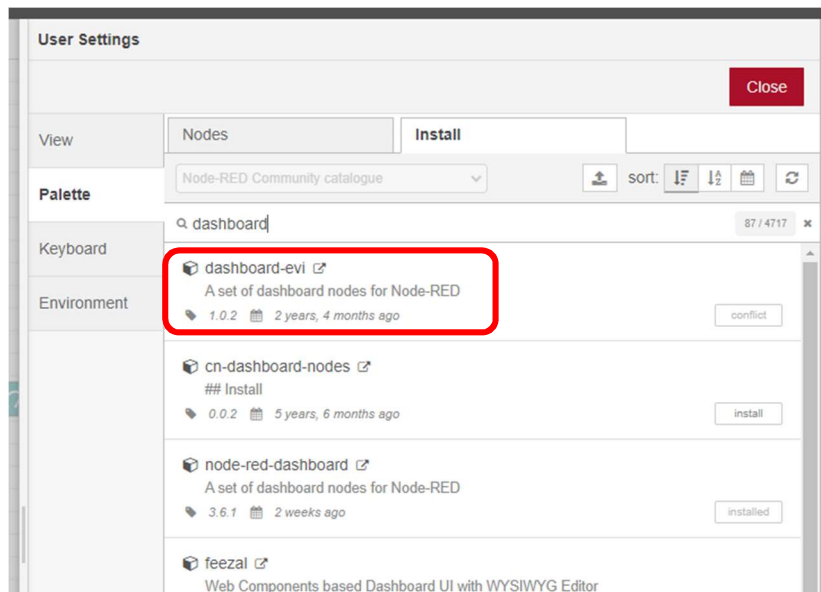
La fotorresistencia **KY-018** (también conocida como LDR o fotodiodo) es un sensor electrónico utilizado para medir el brillo de la luz. Consiste en un fotodiodo acoplado a una resistencia. Cuando la luz incide sobre la fotorresistencia, el fotodiodo se activa y la resistencia del sensor cambia.





El primer paso será conectarnos vía web introduciendo la “ip del servidor”:1880, e instalar la palette Dashboard para poder usar nodos que no están instalados por defecto.





Ahora ya podemos crear el flow usando bloques visuales. En un lado el nodo MQTT In, y el otro un indicador o gauge.

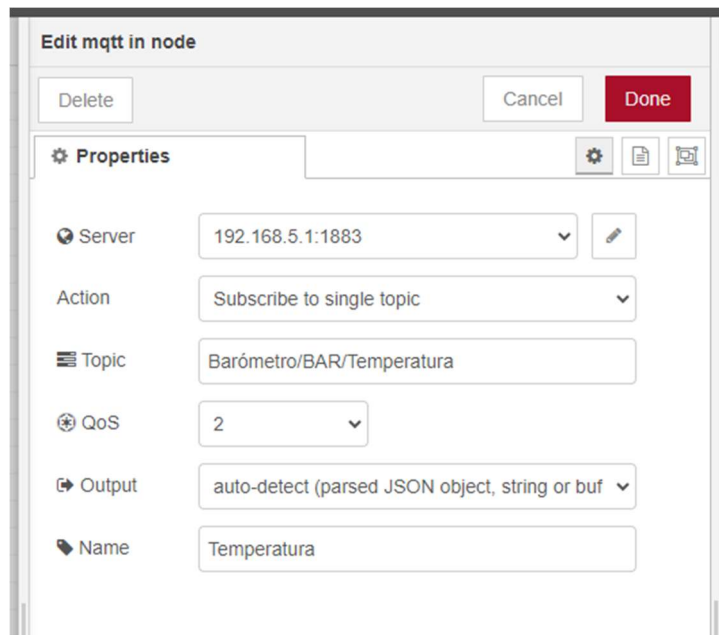


Editaremos nuestros nodos en función de nuestras necesidades. En el nodo MQTT In debemos indicar la ip de nuestro servidor o bróker, en este caso la raspberry, y generamos un tópic al que debe suscribirse.

También configuraremos el indicador QoS. El indicador QOS (calidad de servicio) indica la consistencia con la que los mensajes de este tema se tienen que entregar a los clientes y solo tiene tres valores posibles:

- **0:** No confiable, el mensaje se entrega como mucho una vez, si el cliente no está disponible en ese momento se perderá el mensaje.
- **1:** el mensaje se debería entregar al menos una vez.
- **2:** el mensaje se debería entregar exactamente una vez.

Finalmente pondremos un nombre al bloque y tendremos el sistema funcionando.



**Edit mqtt in node**

Delete Cancel Done

**Properties**

Server 192.168.5.1:1883

Action Subscribe to single topic

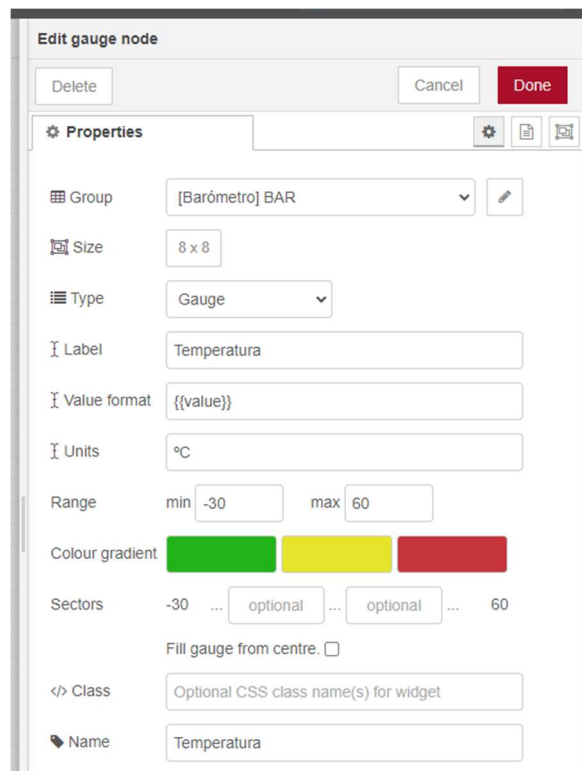
Topic Barómetro/BAR/Temperatura

QoS 2

Output auto-detect (parsed JSON object, string or buf)

Name Temperatura

También configuraremos cada indicador o gauge.



**Edit gauge node**

Delete Cancel Done

**Properties**

Group [Barómetro] BAR

Size 8 x 8

Type Gauge

Label Temperatura

Value format {{value}}

Units °C

Range min -30 max 60

Colour gradient

Sectors -30 ... optional ... optional ... 60

Fill gauge from centre. ☐

Class Optional CSS class name(s) for widget

Name Temperatura

Con Node-Red en marcha en nuestra raspberry, podemos observar que nuestro ESP32 se conecta a la red WIFI de nuestra propia Pi que actúa como Access Point, y empieza a enviar los datos de los sensores a los que está suscrito.



```

8 Nov 11:42:43 - [info]
Welcome to Node-RED
=====
8 Nov 11:42:43 - [info] Node-RED version: v3.1.0
8 Nov 11:42:43 - [info] Node.js version: v14.21.3
8 Nov 11:42:43 - [info] Linux 6.1.21-v7+ arm LE
8 Nov 11:42:45 - [info] Loading palette nodes
8 Nov 11:42:48 - [info] Dashboard version 3.6.1 started at /ui
8 Nov 11:42:50 - [warn] -----
8 Nov 11:42:50 - [warn] [node-red-node-rbe/rbe] 'rbe' already registered by module node-red
8 Nov 11:42:50 - [warn] -----
8 Nov 11:42:50 - [info] Settings file : /home/pgrane/.node-red/settings.js
8 Nov 11:42:50 - [info] Context store : 'default' [module=memory]
8 Nov 11:42:50 - [info] User directory : /home/pgrane/.node-red
8 Nov 11:42:50 - [warn] Projects disabled : editorTheme.projects.enabled=false
8 Nov 11:42:50 - [info] Flows file : /home/pgrane/.node-red/flows.json
8 Nov 11:42:50 - [info] Server now running at http://127.0.0.1:1880/
8 Nov 11:42:50 - [warn]

-----
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
-----

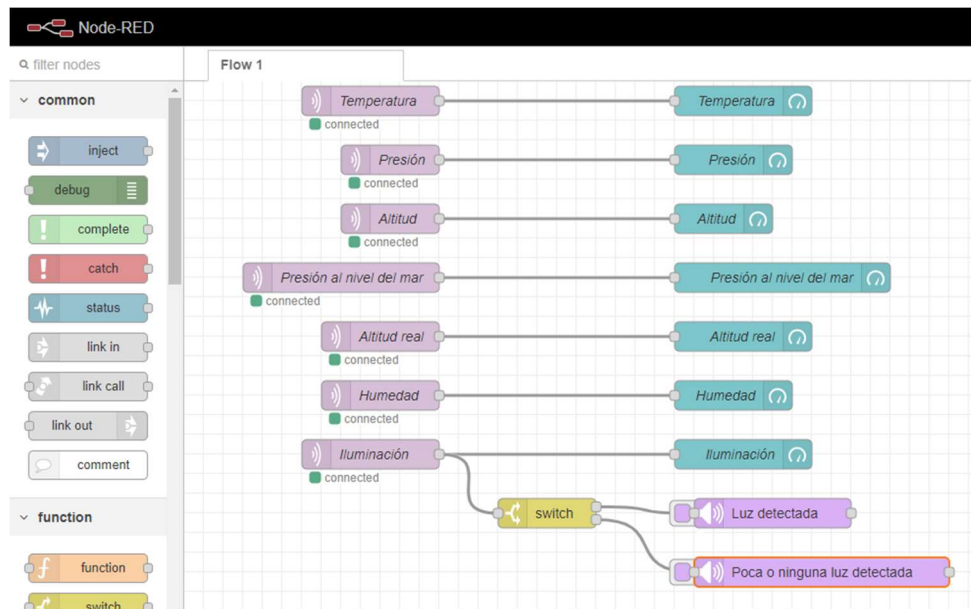
8 Nov 11:42:50 - [info] Starting flows
8 Nov 11:42:51 - [info] Started flows
8 Nov 11:42:51 - [info] [mqtt-broker:d4df0b6502d9ca86] Connected to broker: mqtt://192.168.5.1:1883

```

```

Conectando a Wi-Fi...
Conectado a Wi-Fi.
Conectando a MQTT...
Conectado a MQTT.
Session present: 0
Publicando en el tema Barómetro/BAR/Temperatura, QoS 2, packetId: 43 Mensaje: 21.50 °C
Publicando en el tema Barómetro/BAR/Presión, QoS 2, packetId: 44 Mensaje: 99590.00 Pa
Publicando en el tema Barómetro/BAR/Altitud, QoS 2, packetId: 45 Mensaje: 145.63 metros
Publicando en el tema Barómetro/BAR/Presion al nivel del mar, QoS 2, packetId: 46 Mensaje: 99598.00 Pa
Publicando en el tema Barómetro/BAR/Altitud_real, QoS 2, packetId: 47 Mensaje: 159.63 metros
Publicando en el tema Barómetro/BAR/Iluminación, QoS 2, packetId: 48 Mensaje: 1

```



Conjuntamente, a nuestro nodo MQTT In de iluminación, le conectaremos un nodo Switch.



El nodo Switch nos permite enrutar mensajes según ciertas condiciones o reglas. Actúa como una herramienta de toma de decisiones dentro del flujo, permitiéndonos definir reglas para dirigir mensajes a diferentes ramas de salida.

En nuestro caso, ya que nuestro fotoresistor RY-018 detecta la ausencia o no de luz, enviando un 0 cuando no hay luz, y un 1 cuando si la hay, creamos las reglas para cuando la lectura es igual a 1, y para cuando es igual a cero, generándonos dos salidas del nodo.

The screenshot shows the 'Edit switch node' interface. It includes a 'Name' field, a 'Property' dropdown set to 'msg. payload', and two conditional rules. The first rule is '== a\_z 1' leading to output '1', and the second is '== a\_z 0' leading to output '2'. At the bottom, there is an 'add' button and a dropdown menu set to 'stopping after first match'.

En cada una de estas salidas conectaremos un nodo Play Sound para obtener la lectura con un mensaje de voz que configuraremos para cada caso en particular.

Para ello crearemos dos archivos .mp3 con el mensaje de voz en la propia raspberry con el siguiente archivo en Python usando gTTS o Google Text-to-Speech que es una librería de Python y un CLI para convertir o sintetizar desde un archivo de texto a voz en mp3:

```
from gtts import gTTS

import os

language = 'es'

mytext1 = 'Luz detectada'

myobj = gTTS(text=mytext1, lang=language, slow=False)

myobj.save("uno.mp3")

os.system("cvlc --play-and-exit uno.mp3")

mytext2 = 'Poca o ninguna luz detectada'

myobj = gTTS(text=mytext2, lang=language, slow=False)

myobj.save("cero.mp3")

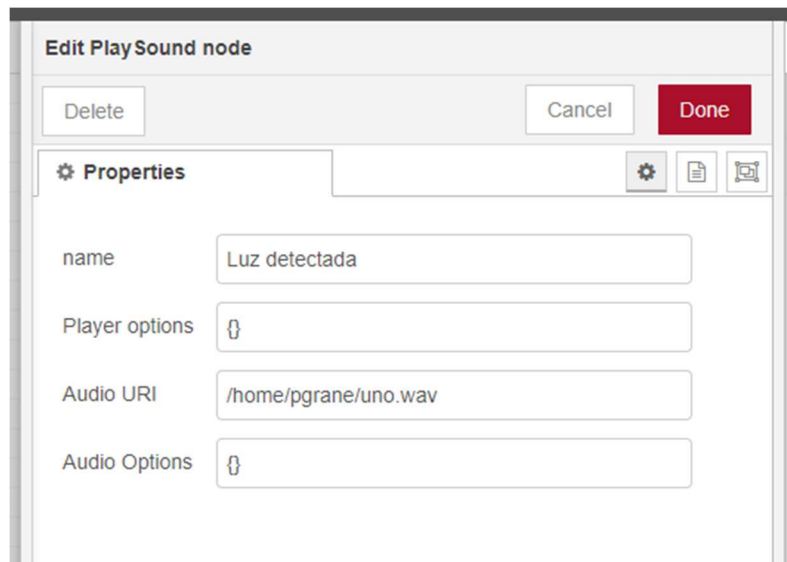
os.system("cvlc --play-and-exit cero.mp3")
```

Debido a que Aplay por defecto no reproduce ficheros .mp3, cambiaremos el formato a .wav para poder reproducirlo.

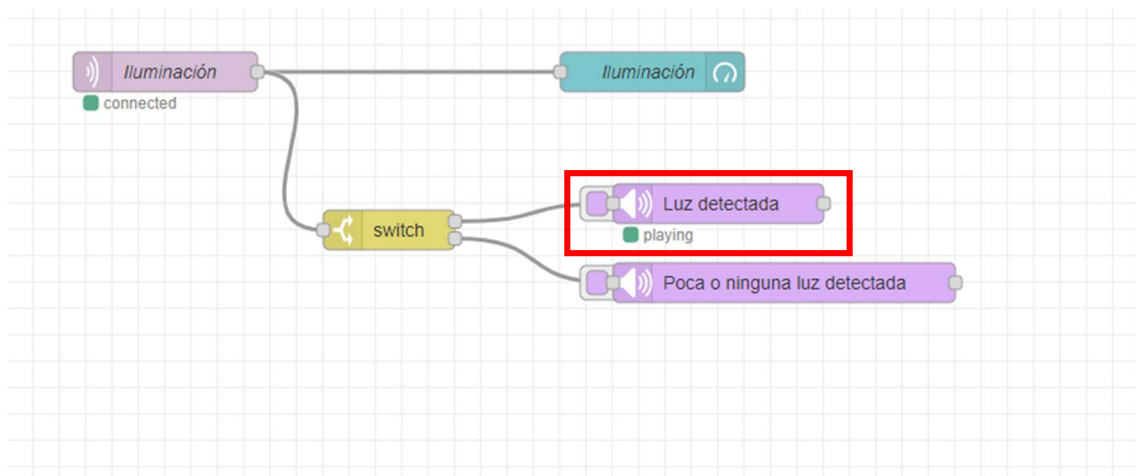
```
ffmpeg -i uno.mp3 uno.wav

ffmpeg -i cero.mp3 cero.wav
```

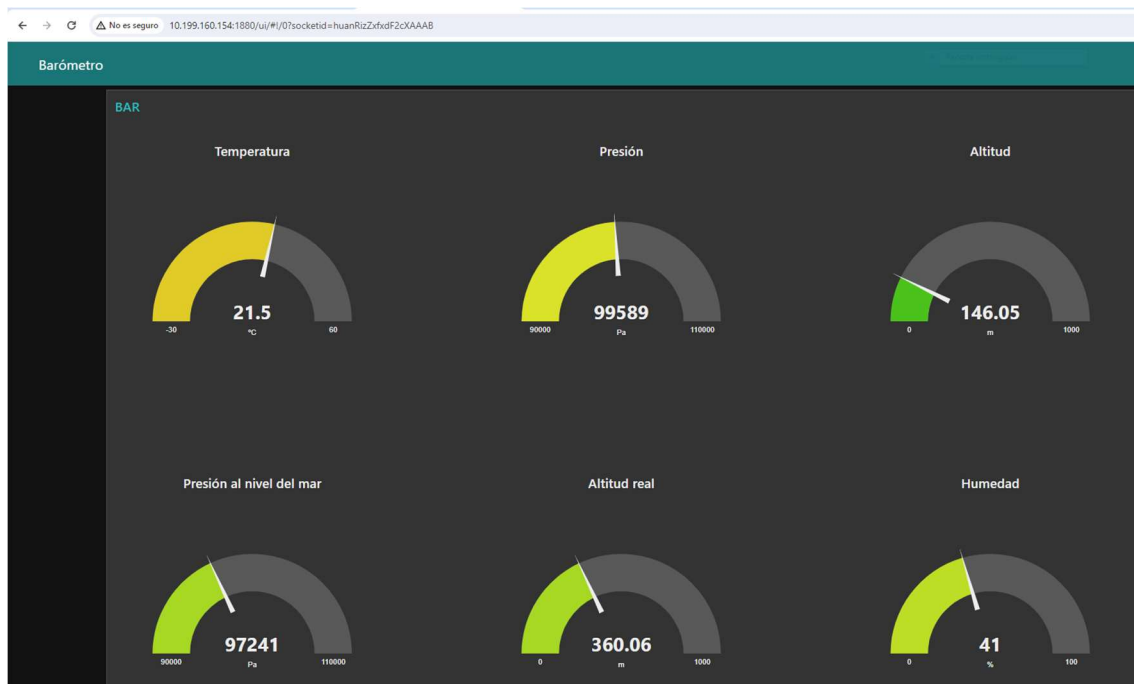
Ahora que tenemos los dos ficheros convertido a .wav, podemos introducir la ruta donde están almacenados para que los ejecute y reproduzca.



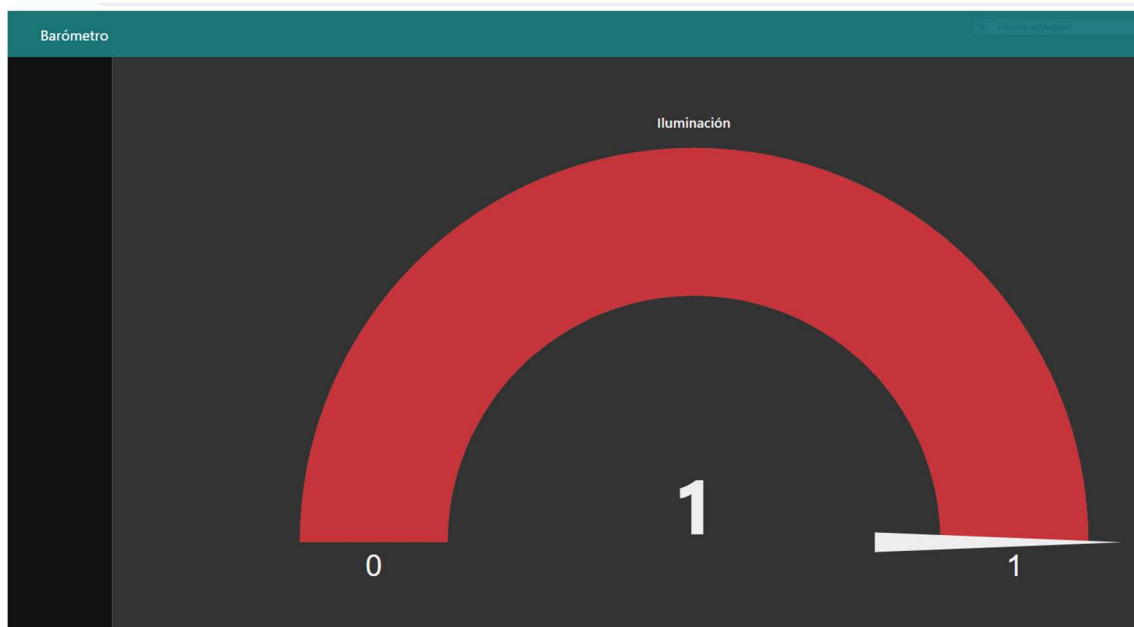
De este modo, cuando el bróker reciba los datos del cliente, reproducirá el audio.

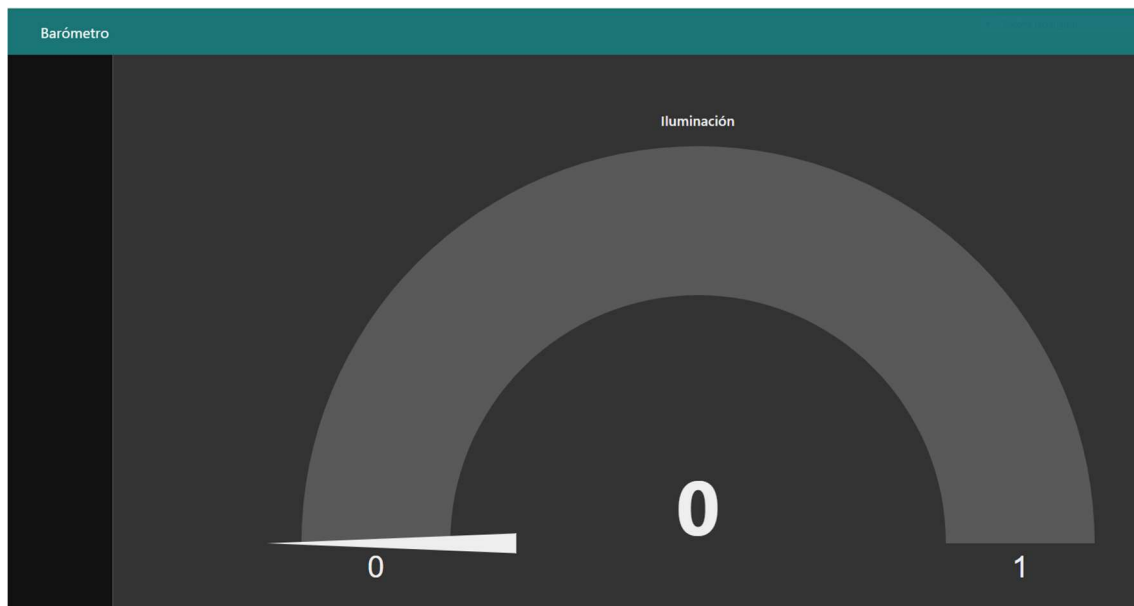


Una vez terminada toda la configuración, ya podremos acceder al **Dashboard** introduciendo la "ip del servidor":1880/ui para ver los indicadores configurados anteriormente.

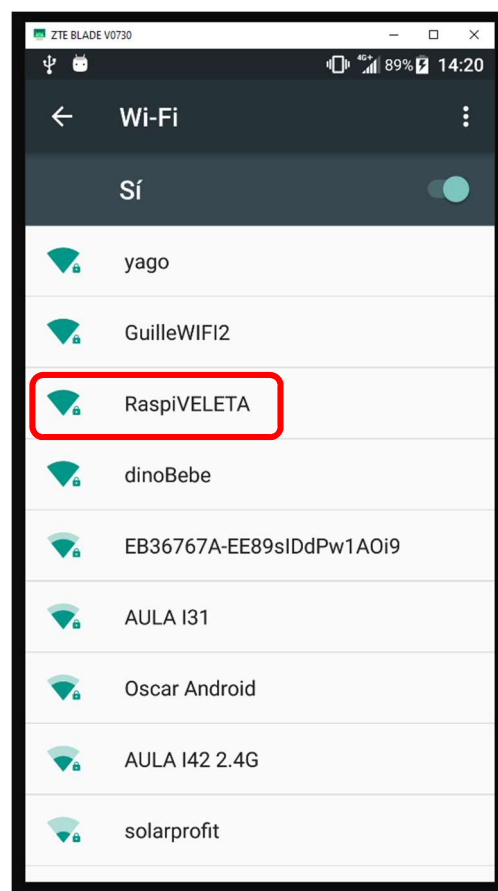


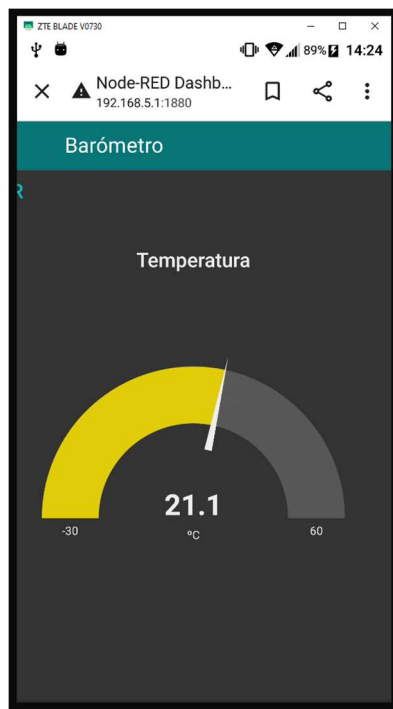
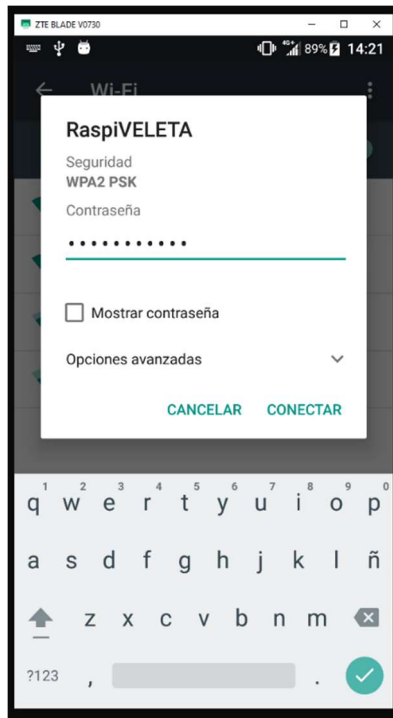
En el caso de la foto resistor RY-018, podemos comprobar el estado del indicador en función de si hay o no hay luz.





También podemos conectarnos a la Dashboard desde otros dispositivos como un teléfono móvil, por ejemplo.





## Código del cliente ESP32:

```
#include <Wire.h>
#include <Adafruit_BMP085.h>
#include <WiFi.h>
#include <AsyncMqttClient.h>
#include <Ticker.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>

#define WIFI_SSID "RaspiVELETA"
#define WIFI_PASSWORD "veleta12345"
#define MQTT_HOST IPAddress(192, 168, 5, 1)
#define MQTT_PORT 1883

const char* mqtt_topic_temperature = "Barómetro/BAR/Temperatura";
const char* mqtt_topic_pressure = "Barómetro/BAR/Presión";
const char* mqtt_topic_altitude = "Barómetro/BAR/Altitud";
const char* mqtt_topic_sea_level_pressure =
"Barómetro/BAR/Presion_al_nivel_del_mar";
const char* mqtt_topic_real_altitude = "Barómetro/BAR/Altitud_real";
const char* mqtt_topic_humidity = "Barómetro/BAR/Humedad"; // Nuevo tema
para la humedad
const char* mqtt_topic_ilum = "Barómetro/BAR/Iluminación"; // Tema para
el sensor KY-018

Adafruit_BMP085 bmp;
DHT dht(4, DHT11); // Conecta el sensor DHT11 al pin 4 de la ESP32

const int KY018Pin = 18; // Pin GPIO al que está conectado el módulo KY-
018
int ilum;

float temperature, pressure, altitude, seaLevelPressure, realAltitude,
humidity;

AsyncMqttClient mqttClient;
Ticker mqttReconnectTimer;
Ticker wifiReconnectTimer;

unsigned long previousMillis = 0;
const long interval = 10000;

void onWifiEvent(WiFiEvent_t event, WiFiEventInfo_t info) {
  if (event == SYSTEM_EVENT_STA_GOT_IP) {
    Serial.println("Conectado a Wi-Fi.");
    connectToMqtt();
  } else if (event == SYSTEM_EVENT_STA_DISCONNECTED) {
```

```

        Serial.println("Desconectado de Wi-Fi.");
        mqttReconnectTimer.detach();
        wifiReconnectTimer.once(2, connectToWifi);
    }
}

void connectToWifi() {
    Serial.println("Conectando a Wi-Fi...");
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}

void connectToMqtt() {
    Serial.println("Conectando a MQTT...");
    mqttClient.connect();
}

void onMqttConnect(bool sessionPresent) {
    Serial.println("Conectado a MQTT.");
    Serial.print("Session present: ");
    Serial.println(sessionPresent);
}

void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
    Serial.println("Desconectado de MQTT.");

    if (WiFi.isConnected()) {
        mqttReconnectTimer.once(2, connectToMqtt);
    }
}

void setup() {
    Serial.begin(115200);
    Serial.println();

    if (!bmp.begin()) {
        Serial.println("No se encontró un sensor BMP085 válido, ¡verifica la
conexión!");
        while (1);
    }

    dht.begin();

    pinMode(KY018Pin, INPUT); // Inicializa el pin del sensor KY-018 como
entrada

    WiFi.onEvent(onWifiEvent);

    mqttClient.onConnect(onMqttConnect);
    mqttClient.onDisconnect(onMqttDisconnect);
}

```



```

mqttClient.setServer(MQTT_HOST, MQTT_PORT);

connectToWifi();
}

void loop() {
    unsigned long currentMillis = millis();

    if (currentMillis - previousMillis >= interval) {
        previousMillis = currentMillis;

        temperature = bmp.readTemperature();
        pressure = bmp.readPressure();
        altitude = bmp.readAltitude();
        seaLevelPressure = bmp.readSealevelPressure();
        realAltitude = bmp.readAltitude(101500);
        humidity = dht.readHumidity();

        int digitalValue = digitalRead(KY018Pin);

        if (digitalValue == LOW) {
            ilum = 1; // Poca o ninguna luz detectada
        } else {
            ilum = 0; // Luz detectada
        }

        // Envía los datos de los sensores a través de MQTT
        uint16_t packetIdTemp = mqttClient.publish(mqtt_topic_temperature, 1,
true, String(temperature).c_str());
        Serial.printf("Publicando en el tema %s, QoS 2, packetId: %i ",
mqtt_topic_temperature, packetIdTemp);
        Serial.printf("Mensaje: %.2f °C\n", temperature);

        uint16_t packetIdPressure = mqttClient.publish(mqtt_topic_pressure,
1, true, String(pressure).c_str());
        Serial.printf("Publicando en el tema %s, QoS 2, packetId: %i ",
mqtt_topic_pressure, packetIdPressure);
        Serial.printf("Mensaje: %.2f Pa\n", pressure);

        uint16_t packetIdAltitude = mqttClient.publish(mqtt_topic_altitude,
1, true, String(altitude).c_str());
        Serial.printf("Publicando en el tema %s, QoS 2, packetId: %i ",
mqtt_topic_altitude, packetIdAltitude);
        Serial.printf("Mensaje: %.2f metros\n", altitude);

        uint16_t packetIdSeaLevelPressure =
mqttClient.publish(mqtt_topic_sea_level_pressure, 1, true,
String(seaLevelPressure).c_str());

```

```

    Serial.printf("Publicando en el tema %s, QoS 2, packetId: %i ",
mqtt_topic_sea_level_pressure, packetIdSeaLevelPressure);
    Serial.printf("Mensaje: %.2f Pa\n", seaLevelPressure);

    uint16_t packetIdRealAltitude =
mqttClient.publish(mqtt_topic_real_altitude, 1, true,
String(realAltitude).c_str());
    Serial.printf("Publicando en el tema %s, QoS 2, packetId: %i ",
mqtt_topic_real_altitude, packetIdRealAltitude);
    Serial.printf("Mensaje: %.2f metros\n", realAltitude);

    uint16_t packetIdIllum = mqttClient.publish(mqtt_topic_illum, 1, true,
String(illum).c_str());
    Serial.printf("Publicando en el tema %s, QoS 2, packetId: %i ",
mqtt_topic_illum, packetIdIllum);
    Serial.printf("Mensaje: %d\n", illum);
}
}

```

- BRUJULA QMC5883L

Programación:

```
#include <Wire.h>
#include <QMC5883LCompass.h>

const int buttonPin = 18; // Button connected to GPIO 2
const int vibradorPin = 23; // Built-in LED on GPIO 13
String direccio = "";
QMC5883LCompass compass;
int azimut;
unsigned long previousMillis = 0;
unsigned long currentMillis = millis();
const long interval = 500;

void setup() {
  Serial.begin(9600);
  compass.init();
  pinMode(buttonPin, INPUT);
  pinMode(vibradorPin, OUTPUT);
  attachInterrupt(digitalPinToInterrupt(buttonPin),
handleButtonInterrupt, FALLING);
}

void loop() {
  currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    compass.read();
    azimut = compass.getAzimuth();

    if (azimut < 0) {
      azimut = azimut + 360;
    }

    if ((azimut >= 157.5) && (azimut < 202.5)) {
      direccio = "Sur";
    } else if ((azimut >= 202.5) && (azimut < 247.5)) {
      direccio = "Suroeste";
    } else if ((azimut >= 247.5) && (azimut < 292.5)) {
      direccio = "Oeste";
    } else if ((azimut >= 292.5) && (azimut < 337.5)) {
      direccio = "Noroeste";
    } else if ((azimut >= 337.5) || (azimut < 22.5)) {
      direccio = "Norte";
    }
  }
}
```

```

    } else if ((azimut >= 22.5) && (azimut < 67.5)) {
        direccio = "Noreste";
    } else if ((azimut >= 67.5) && (azimut < 112.5)) {
        direccio = "Este";
    } else if ((azimut >= 112.5) && (azimut < 157.5)) {
        direccio = "Sureste";
    }
    Serial.println(direccio);
    if (digitalRead(buttonPin) == HIGH && direccio == "Norte") {
        digitalWrite(vibradorPin, HIGH); // Activa el vibrador si el botón
        // está presionado y apuntando al norte
    } else {
        digitalWrite(vibradorPin, LOW); // Apaga el vibrador en otros
        // casos
    }
}

void handleButtonInterrupt() {
    // Esta función no es necesaria en este caso.
}

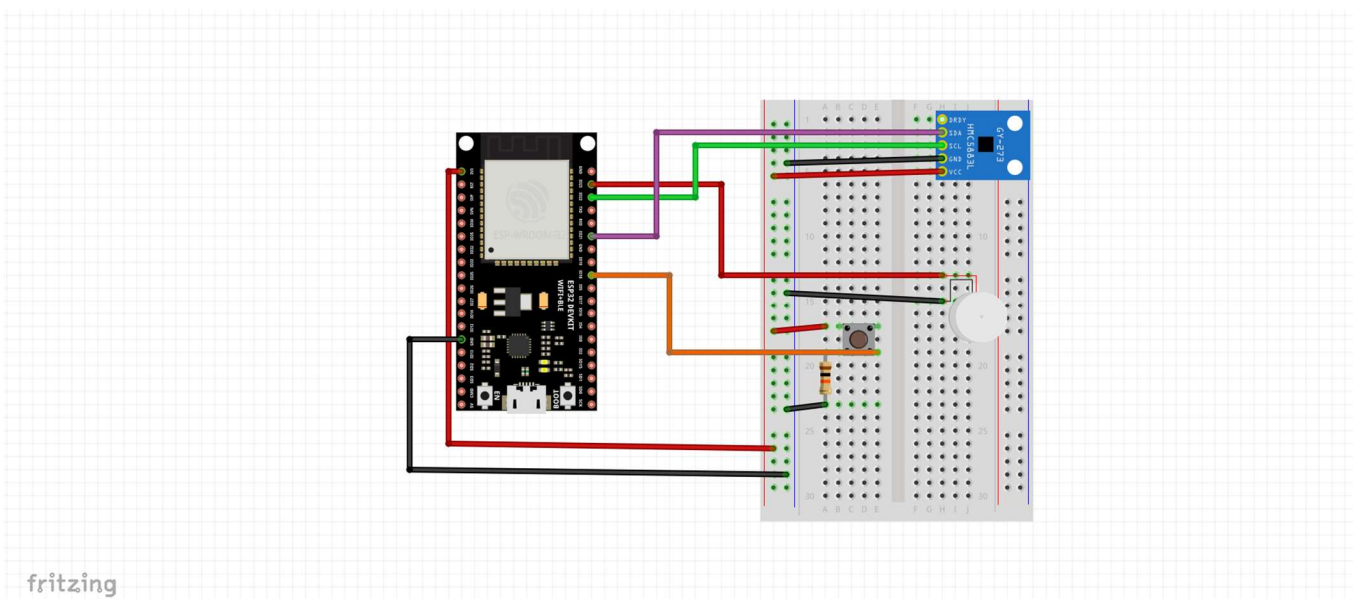
```

## Explicación del código:

Este código es un programa para la ESP32 que utiliza un magnetómetro QMC5883L para determinar la dirección del norte, si se presiona un botón conectado a GPIO 2 (pin 18), activa un vibrador conectado a GPIO 13 (pin 23) si el dispositivo está apuntando hacia el norte.

1. Se incluyen las bibliotecas necesarias para utilizar el sensor QMC5883L y la comunicación I2C con la función `#include <Wire.h>` y `#include <QMC5883LCompass.h>`.
2. Se definen algunas constantes y variables:
  - `buttonPin` se configura en 18, que es el pin al que se conecta un botón.
  - `vibradorPin` se configura en 23, que es el pin al que se conecta un vibrador.
  - `direccio` es una cadena de texto que se utiliza para almacenar la dirección (Norte, Sur, Este, Oeste, etc.) calculada por el magnetómetro.
  - `compass` es un objeto de la clase `QMC5883LCompass` que se utiliza para interactuar con el sensor QMC5883L.
  - `a` es una variable que se utiliza para almacenar la lectura del `azimut` (dirección en grados) del magnetómetro.

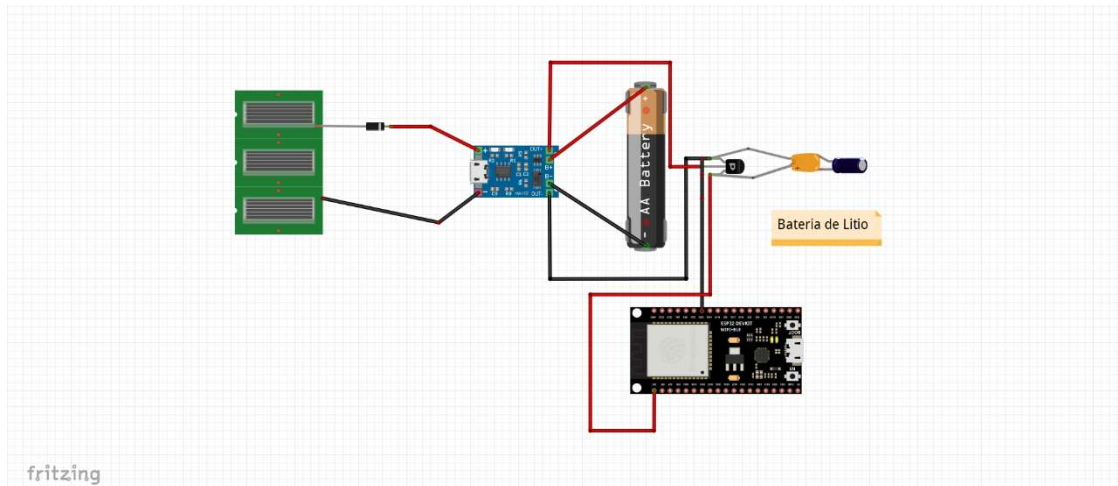
- `previousMillis` y `currentMillis` se utilizan para controlar el tiempo de actualización del sensor.
  - `interval` se establece en 500 milisegundos (0.5 segundos) y define cada cuánto tiempo se actualizará la dirección.
3. En la función `setup()`, se realiza la inicialización:
    - Se inicia la comunicación serie a una velocidad de 9600 baudios para la salida de datos de depuración.
    - Se inicializa el sensor QMC5883L utilizando `compass.init()`.
    - Se configura `buttonPin` como entrada (botón).
    - Se configura `vibradorPin` como salida (para el vibrador).
    - Se adjunta una interrupción al pin `buttonPin` para detectar la pulsación del botón y llamar a la función `handleButtonInterrupt` cuando se presiona el botón.
  4. En la función `loop()`, se ejecuta el bucle principal del programa:
    - Se verifica si ha pasado el tiempo definido por `interval` desde la última actualización del sensor.
    - Se lee el sensor QMC5883L con `compass.read()` y se obtiene el azimut (dirección en grados) con `compass.getAzimuth()`.
    - Se convierte el valor del azimut a una dirección textual (Norte, Sur, Este, Oeste, etc.) y se almacena en la variable `direccion`.
    - Se imprime la dirección en la consola serie.
    - Se verifica si el botón está presionado y si la dirección es "Norte". Si es así, se activa el vibrador; de lo contrario, se apaga.
    - Esto permite que el vibrador se active cuando el dispositivo esté apuntando al norte y el botón esté presionado.
  5. La función `handleButtonInterrupt()`.



- PLACA SOLAR:

La alimentación de la placa de desarrollo ESP32 será realizada con paneles solares, una batería de litio 18650 y el módulo cargador de batería TP4056.

El siguiente diagrama muestra cómo funciona el circuito para alimentar el ESP32 con paneles solares.



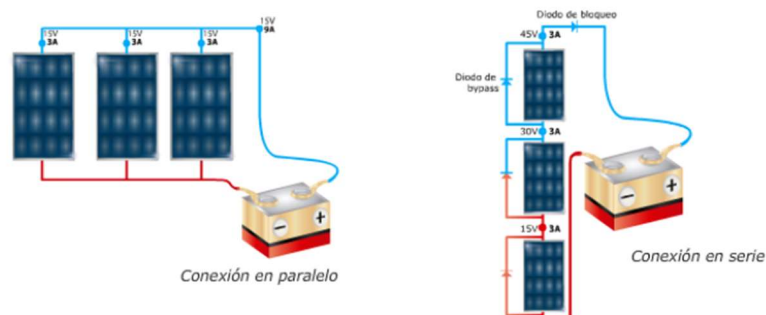
1. Los paneles solares emiten alrededor de 5V con sol directo.
2. Los paneles solares cargan la batería de litio a través del módulo cargador de baterías TP4056. Este módulo se encarga de cargar la batería y evitar la sobrecarga.
3. La batería de litio produce 4,2 V cuando está completamente cargada.
4. Debe usar un circuito regulador de voltaje de baja caída (MCP1700-3302E) para obtener 3.3 V de la salida de la batería.
5. La salida del regulador de voltaje alimentará el ESP32 a través del pin de 3,3 V.

## Paneles solares:

Los paneles solares que estamos utilizando tienen un voltaje de salida alrededor de 5V. Si deseamos que la batería se cargue más rápido, podemos usar varios paneles solares en paralelo.

Para cablear los paneles solares en serie, se ha soldado el terminal (+) de un panel solar al terminal (-) del otro panel solar.

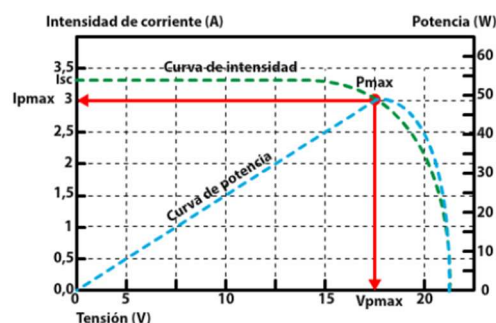
Al cablear paneles solares en serie, obtenemos la suma de los voltajes de salida y la misma corriente (para paneles solares idénticos).



Es importante tener en cuenta las características eléctricas ya que son éstas las que definen el comportamiento de los módulos.

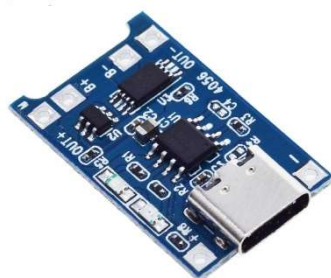
Las características eléctricas son intensidad de cortocircuito, tensión nominal, tensión a circuito abierto, intensidad de potencia máxima, tensión de potencia máxima, potencia máxima.

La curva V/I relaciona indirectamente la potencia de salida con la corriente de carga, ya que asocia en los valores de V e I para diferentes cargas.



## Módulo de cargador TP4056

El módulo cargador de batería de litio TP4056 viene con protección de circuito y evita la sobretensión de la batería y la conexión de polaridad inversa.



El módulo TP4056 enciende un LED rojo cuando está cargando la batería y enciende un LED azul cuando la batería está completamente cargada.

Conectamos los paneles solares al módulo cargador de batería de litio TP4056 como se muestra en el diagrama esquemático a continuación. Conecte los terminales positivos a la almohadilla marcada con IN+ y los terminales negativos a la almohadilla marcados con IN-.

A continuación, conecte el terminal positivo del soporte de la batería al B+ y el terminal negativo del soporte de la batería al B-.

OUT+ y OUT- son las salidas de la batería. Estas baterías de litio producen hasta 4,2 V cuando están completamente cargadas (aunque tienen 3,7 V marcados en la etiqueta).

## Regulador de voltaje:

Usar un regulador de voltaje lineal típico para bajar el voltaje de 4.2V a 3.3V no es una buena idea, porque a medida que la batería se descarga a, por ejemplo, 3.7V, su regulador de voltaje dejaría de funcionar, porque tiene un alto voltaje de corte.

Para reducir el voltaje de manera eficiente con baterías, debe usar un regulador de baja caída, o LDO para abreviar, que pueda regular el voltaje de salida.



Los LDO deben tener un condensador cerámico y un condensador electrolítico conectados en paralelo a GND y Vout para suavizar los picos de voltaje. Aquí estamos usando un condensador electrolítico de 100 uF y un condensador cerámico de 100 nF.

El pin Vout del regulador de voltaje debe emitir 3.3V. Ese es el pin que alimentará el ESP32.

Finalmente, después de asegurarse de que está obteniendo el voltaje correcto en el pin Vout del regulador de voltaje, puede alimentar el ESP32. Conecte el pin Vout al pin de 3.3V del ESP32 y GND a GND.



## Diodo Schottky

El SR240 es un diodo Schottky. Su elección se debe a que este tiene una caída de tensión menor que los diodos convencionales de silicio. En el SR240 la caída de tensión es además especialmente pequeña, en torno a los 0.55V. De este modo podremos proteger el panel solar a la vez que no disminuimos demasiado su rendimiento. Además, puede trabajar con intensidades de hasta 2A. Así nos aseguramos de que será capaz de funcionar sin problemas también en el caso de que el panel trabaje al máximo.

## Circuito de Monitoreo de Nivel de Tensión de Batería

Cuando tienes tu ESP32 alimentado con baterías o energía solar como en este caso, puede ser muy útil para controlar el nivel de la batería. Una manera de hacerlo es leer el voltaje de salida de la batería usando un pin analógico del ESP32.

Sin embargo, la batería que utilizamos aquí produce un máximo de 4.2V cuando está completamente cargada, pero los GPIO de ESP32 funcionan a 3.3V. Por lo tanto, tenemos que añadir un divisor de tensión para que podamos leer el voltaje de la batería.

La fórmula del divisor de tensión es la siguiente:

$$V_{out} = (V_{in} * R2) / (R1 + R2)$$

El ESP32 está diseñado para funcionar con un voltaje de alimentación de 3.3V. Si se alimenta con un voltaje superior a 3.3V, puede dañar el dispositivo. Por lo tanto, es importante asegurarse de que el voltaje de entrada no supere los 3.3V.

La corriente necesaria para alimentar el ESP32 depende del uso que se le dé al dispositivo. En general, se recomienda una corriente de al menos 500 mA para alimentar el ESP32. Si se utilizan periféricos adicionales, como sensores o módulos de comunicación, es posible que se requiera una corriente mayor.

Cuando se alimenta el ESP32 utilizando paneles solares o baterías, es importante ahorrar energía. Para ello, puede utilizar en modo “deepsleep” ESP32.

El modo de sueño profundo consiste en un modo de funcionamiento donde el ESP32 apaga casi todos sus periféricos para ahorrar energía eléctrica (solo se encienden el RTC y el procesador ULP). Al estar apagadas la mayoría de los periféricos, el consumo de energía eléctrica del ESP32 cae drásticamente, permitiendo su uso en sistemas embebidos alimentados por baterías según el ciclo de funcionamiento.

En modo de suspensión profunda, el consumo de corriente eléctrica puede alcanzar al menos 10  $\mu$ A. Sólo para tener una base de comparación, en pleno funcionamiento, el consumo de corriente eléctrica puede alcanzar los 50 mA (sin utilizar WiFi).

En otras palabras, “durmiendo” (en sueño profundo), el ESP32 consume 5.000 veces menos corriente eléctrica (y, en consecuencia, energía eléctrica) en comparación con el modo de funcionamiento completo, lo que se traduce en un ahorro de energía eléctrica muy significativo.

## Consumo de energía del módulo ESP32 WROOM

Algunos valores de consumo de energía determinados del chip ESP32:

Modo ESP32	Consumo
“Deepsleep”	7 $\mu$ A
“Lightsleep”	1 mA
Normal (240 MHz)	50 mA
Reloj del procesador reducido (3 MHz)	3,8 mA
Funcionamiento WiFi	80-180 mA

- CONCLUSIONES:

Al concluir este proyecto de IoT con Raspberry Pi, reflexionamos sobre los logros alcanzados y las puertas que hemos abierto hacia futuras exploraciones. Aunque reconocemos que con más tiempo podríamos haber llevado a cabo un proyecto aún más extenso, estamos satisfechos con los resultados obtenidos hasta ahora.

Queremos resaltar la importancia del entorno educativo proporcionado por el Centro de Formación Cifo La Violeta por contar con gran variedad de material didáctico para realizar las prácticas y el propio proyecto.

Queríamos dar las gracias a nuestro profesor Joan Masdemont por el esfuerzo y cariño mostrado tanto a nosotros como al resto de compañeros. Valoramos profundamente el tiempo y la energía invertidas durante todo el curso.

- BIBLIOGRAFIA:

<https://es.wikipedia.org/wiki/Wikipedia:Portada>

<https://www.luisllamas.es/>

<https://programarfacil.com/blog/arduino-blog/curso-de-arduino/>

<https://www.youtube.com/watch?v=eBVvD85MI2c>

<https://revhardware.com/>

<https://lab.bricogeek.com/>

<http://bitsmi.com/>