



TRABAJO FIN DE MÁSTER  
INGENIERÍA INFORMÁTICA

# Monitorización del consumo eléctrico en tiempo real

---

Micropython, Esp32 y MQTT

**Autor**  
Adrián Bonel Bolívar

**Director**  
Alvaro Gomez Pau

SISTEMAS ENCASTADOS

Barcelona, Noviembre de 2022







# Monitorización del consumo eléctrico en tiempo real

Adrián Bonel Bolívar

**Palabras clave:** Micropython, Software embebido, Esp32, Acs712, MQTT, Flask, Python, App Web

## Resumen

A día de hoy, el coste de la electricidad ha aumentado a unos niveles que nadie hubiera imaginado hace tan solo unos años. Muchas familias cada vez más, necesitan ir con mucho cuidado sobre cuándo poner ciertos electrodomésticos en casa, para elegir la hora donde el precio está más bajo y así poder ahorrar en la factura de la luz. Saber cuánto consume cada electrodoméstico y como se refleja en el coste de la factura puede llegar a ser de gran utilidad.

Aquí es donde podemos situar el trabajo de este proyecto. Aunque a día de hoy existen sistemas inteligentes con los que gestionar y monitorizar la energía que se consume en el hogar, normalmente estos dispositivos son independientes unos de otros y no encontramos muchos que recopilen los datos. El trabajo de este proyecto, no solo se va a centrar en medir el consumo eléctrico de los dispositivos individualmente, si no que el objetivo será que toda esa información se pueda consultar desde una misma aplicación.

Planteamos por tanto el despliegue de sistemas embebidos capaces de medir el consumo eléctrico y que a su vez sean capaces de comunicarse con un servidor central donde recopilar todos los datos. El despliegue de estos sistemas se hará en un entorno real, donde se probará el funcionamiento de los dispositivos y se analizará el rendimiento de los mismos.



# Real time measurement of power consumption

Adrián Bonel Bolívar

**Keywords:** Micropython, Embedded software, Esp32, Acs712, MQTT, Flask, Python, Web app

## Abstract

Today, the cost of electricity has risen to levels that no one would have imagined just a few years ago. Increasingly, many families need to be very careful about when to put certain devices in the house, to choose the time when the price is lower and thus be able to save on the electricity bill. Knowing how much each device consumes and how it is reflected in the cost of the bill can be very useful.

This is where we can place the work of this project. Although today there are intelligent systems with which to manage and monitor the energy consumed in the home, these devices are usually independent of each other and we do not find many that collect the data. The work of this project will not only focus on measuring the power consumption of individual devices, but the goal will be that all this information can be consulted from a single application.

We therefore propose the deployment of embedded systems capable of measuring electricity consumption and at the same time be able to communicate with a central server to collect all the data. The deployment of these systems will be done in a real environment, where the operation of the embedded systems will be tested and their performance will be analyzed.



---

Yo, **Adrián Bonel Bolívar**, alumno del Máster en Ingeniería Informática de la **Universitat Oberta de Catalunya**, con DNI 76668939A, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Máster en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Adrián Bonel Bolívar

Barcelona a 30 de Noviembre de 2022.



# Índice general

|   |          |
|---|----------|
| <b>1. Introducción</b>                            | <b>1</b> |
| 1.1. Introducción y motivación . . . . .          | 1        |
| 1.2. Objetivos . . . . .                          | 2        |
| 1.3. Planificación . . . . .                      | 2        |
| 1.4. Material y métodos . . . . .                 | 2        |
| 1.5. Estructura del documento . . . . .           | 3        |
| <b>2. Fundamentación teórica</b>                  | <b>1</b> |
| 2.1. Consumo energético . . . . .                 | 1        |
| 2.1.1. Consumo energético en el hogar . . . . .   | 1        |
| 2.1.2. Calculo del consumo energético . . . . .   | 5        |
| 2.1.3. Sensores de consumo energético . . . . .   | 7        |
| 2.2. Plataformas de desarrollo . . . . .          | 9        |
| 2.2.1. Arduino . . . . .                          | 9        |
| 2.2.2. Raspberry Pi . . . . .                     | 10       |
| 2.2.3. NodeMCU ESP8266 . . . . .                  | 10       |
| 2.2.4. ESP32 . . . . .                            | 11       |
| 2.3. Micropython . . . . .                        | 11       |
| 2.4. Comunicaciones inalámbricas en IoT . . . . . | 12       |
| 2.4.1. MQTT . . . . .                             | 12       |
| 2.4.2. Zigbee . . . . .                           | 13       |
| <b>3. Implementación del sensor de corriente</b>  | <b>1</b> |
| 3.1. Descripción general . . . . .                | 1        |
| 3.2. Setup experimental . . . . .                 | 2        |
| 3.2.1. Preparar ESP32 para Micropython . . . . .  | 3        |
| 3.2.2. Micropython en ESP32 . . . . .             | 5        |
| 3.2.3. Librerías utilizadas . . . . .             | 6        |
| 3.3. Implementación . . . . .                     | 7        |
| 3.3.1. Diagrama de estados . . . . .              | 7        |
| 3.3.2. Código . . . . .                           | 9        |

|           |   |          |
|-----------|---|----------|
| 3.4.      | Test de funcionamiento . . . . .            | 13       |
| 3.4.1.    | Corriente continua . . . . .                | 13       |
| 3.4.2.    | Corriente alterna . . . . .                 | 16       |
| <b>4.</b> | <b>Implementación de la aplicación web</b>  | <b>1</b> |
| 4.1.      | Descripción general . . . . .               | 1        |
| 4.2.      | Implementación backend . . . . .            | 1        |
| 4.2.1.    | Aplicación en python3 . . . . .             | 1        |
| 4.3.      | Implementación frontend . . . . .           | 7        |
| 4.3.1.    | Diseño . . . . .                            | 7        |
| 4.3.2.    | Implementación . . . . .                    | 9        |
| <b>5.</b> | <b>Conclusiones</b>                         | <b>1</b> |
| 5.1.      | Objetivos marcados . . . . .                | 1        |
| 5.1.1.    | Fase 1 . . . . .                            | 1        |
| 5.1.2.    | Fase 2 . . . . .                            | 2        |
| 5.1.3.    | Fase 3 . . . . .                            | 2        |
| 5.2.      | Trabajo futuro . . . . .                    | 3        |
| 5.2.1.    | Mejoras en el sensor de corriente . . . . . | 3        |
| 5.2.2.    | Mejoras en la aplicación web . . . . .      | 4        |

# Índice de figuras

|  |    |
|--|----|
| 1.1. Diagrama de Gantt para la planificación inicial . . . . .     | 2  |
| 2.1. Contador de luz[28] . . . . .                                 | 2  |
| 2.2. Medidor de consumo de energía[30] . . . . .                   | 3  |
| 2.3. Medidor de consumo en cuadro electrico[31] . . . . .          | 4  |
| 2.4. Monitor de energia inteligente[32] . . . . .                  | 5  |
| 2.5. Forma de una onda Sinusoidal[27] . . . . .                    | 6  |
| 2.6. Sensor INA219[33] . . . . .                                   | 8  |
| 2.7. Sensor ZMPT101B[34] . . . . .                                 | 8  |
| 2.8. Sensor ACS712[26] . . . . .                                   | 9  |
| 2.9. Arduino pin layout[35] . . . . .                              | 10 |
| 2.10. Raspberry Pi pin layout[36] . . . . .                        | 10 |
| 2.11. Node MCU ESP8266 pin layout[37] . . . . .                    | 11 |
| 2.12. ESP32 pin layout[38] . . . . .                               | 11 |
| 2.13. Ejemplo de dispositivos comunicados por MQTT[39] . . . . .   | 13 |
| 2.14. Ejemplo de dispositivos comunicados por Zigbee[40] . . . . . | 14 |
| 3.1. Divisor de voltaje[41] . . . . .                              | 2  |
| 3.2. Setup experimental . . . . .                                  | 3  |
| 3.3. Prompt de Micropython . . . . .                               | 4  |
| 3.4. Opciones de Aropy . . . . .                                   | 5  |
| 3.5. Diagrama de estados Init . . . . .                            | 7  |
| 3.6. Diagrama de estados función Main . . . . .                    | 8  |
| 3.7. Diagrama de la función de lectura del ACS712 . . . . .        | 9  |
| 3.8. Salida del ACS712 con corriente continua de 0A . . . . .      | 14 |
| 3.9. Salida del ACS712 con corriente continua de 1A . . . . .      | 14 |
| 3.10. Salida del ACS712 con corriente continua de 1.7A . . . . .   | 15 |
| 3.11. Salida del ACS712 con corriente continua de 3A . . . . .     | 15 |
| 3.12. Nada conectado al enchufe . . . . .                          | 16 |
| 3.13. Secador de pelo solo con la función de aire frio . . . . .   | 17 |

---

|   |    |
|---|----|
| 3.14. Secador de pelo solo con la función de aire caliente a nivel bajo de calor. . . . .   | 17 |
| 3.15. Secador de pelo solo con la función de aire caliente a nivel máximo de calor. . . . . | 18 |
| 4.1. Interfaz para añadir un sensor . . . . .   | 8  |
| 4.2. Interfaz para mostrar los sensores añadidos . . . . .                                  | 8  |
| 4.3. Interfaz para mostrar los datos de un sensor . . . . .                                 | 9  |

# Capítulo 1

## Introducción

### 1.1. Introducción y motivación

A día de hoy, el coste de la electricidad ha aumentado a unos niveles que nadie hubiera imaginado hace tan solo unos años. Muchas familias cada vez más, necesitan ir con mucho cuidado sobre cuándo poner ciertos electrodomésticos en casa, para elegir la hora donde el precio está más bajo y así poder ahorrar en la factura de la luz. Saber cuánto consume cada electrodoméstico y como se refleja en el coste de la factura puede llegar a ser de gran utilidad.

Y aquí es donde podemos situar el trabajo de este proyecto. Aunque a día de hoy existen sistemas inteligentes con los que gestionar y monitorizar la energía que se consume en el hogar, normalmente estos dispositivos son independientes unos de otros y no encontramos muchos que recopilen los datos. Estos dispositivos que ya existen en el mercado los analizaremos más adelante. El trabajo de este proyecto, no solo se va a centrar en medir el consumo eléctrico de los dispositivos individualmente, si no que el objetivo será que toda esa información se pueda consultar desde una misma aplicación.

En este proyecto se plantea el despliegue de diferentes sistemas embebidos, que se comunicaran con un servidor que recopile toda la información.

## 1.2. Objetivos

1. Estudio de viabilidad de la plataforma de desarrollo. Uso de micropython y estudio de la existencia de librerías para la comunicación inalámbrica por MQTT.
2. Estudio de los diferentes dispositivos que se pueden utilizar para medir el consumo eléctrico.
3. Puesta a punto de raspberry pi para el despliegue de un broker MQTT.
4. Integración de los sensores y dispositivos con la plataforma elegida.
5. Aplicación web en FLask que permita la visualización de los datos recogidos por los dispositivos en tiempo real.

## 1.3. Planificación

La planificación inicial y el tiempo estimado a emplear en cada tarea fue el siguiente: Hasta el momento, se han cumplido la mayoría de objetivos

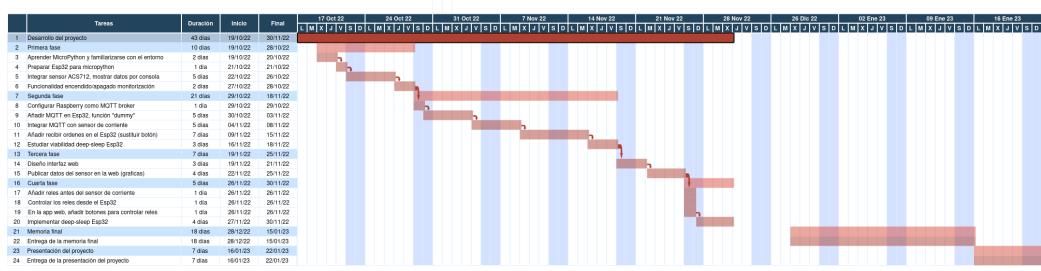


Figura 1.1: Diagrama de Gantt para la planificación inicial

planificados, aunque se han tenido que retrasar algunas tareas debido a la falta de tiempo. A continuación se muestra la planificación actualizada: Insertar imagen con nueva planificación

## 1.4. Material y métodos

### Software

- Visual Studio Code como IDE de programación.

- Ordenador con Ubuntu como SO.
- Esptool como herramienta para trabajar con la flash de los ESP32.
- Para la creación de este documento se ha utilizado Texmaker como plataforma de creación y edición de documentos L<sup>A</sup>T<sub>E</sub>X
- Picocom como software para la comunicación por serial.

## Equipos

- Raspberry Pi 2B+ como broker MQTT.
- x3 ESP32.
- x3 sensores de corriente ACS712.
- x3 pantallas OLED SSD1306.
- x3 cables USB - Micro USB.
- Multímetro
- Fuente de alimentación.
- x6 Mini breadboards.
- Otros componentes electrónicos.

## 1.5. Estructura del documento

El documento se divide en 5 capítulos, que se describen a continuación:

1. Capítulo 1. Introducción. En este capítulo se describen los objetivos del proyecto, así como la planificación inicial y la planificación actualizada. También se describen los materiales y métodos utilizados para la realización del proyecto.
2. Capítulo 2. Estudio del problema. En este capítulo se describen los diferentes dispositivos que se pueden utilizar para medir el consumo eléctrico. También se describen los diferentes protocolos de comunicación que se pueden utilizar para la comunicación entre los dispositivos y el servidor.

3. Capítulo 3. Implementación del sistema de monitorización de corriente. En este capítulo se describe la solución final, los materiales utilizados, el setup experimental y la implementación en Micropython.
4. Capítulo 4. Implementación de la aplicación web. En este capítulo se describe la implementación de la aplicación web en Python usando Flask, SocketIO y MQTT.
5. Capítulo 5. Conclusiones y trabajo futuro. En este capítulo se describen las conclusiones del proyecto y las posibles mejoras que se podrían realizar en el futuro.

# **Capítulo 2**

## **Fundamentación teórica**

### **2.1. Consumo energético**

#### **2.1.1. Consumo energético en el hogar**

##### **Contadores inteligentes**

A día de hoy la mayoría de las compañías eléctricas están sustituyendo los contadores en todas las viviendas por nuevos contadores inteligentes que te muestran el consumo eléctrico de toda la vivienda en tiempo real.

Ventajas:

- En teoría, la renovación de los contadores corre a cargo de las compañías eléctricas, por lo que esta opción no tiene ningún coste extra.

Inconvenientes:

- La medida que nos da es genérica y no podemos saber cual es el consumo individual de ciertos dispositivos en nuestra red.
- En algunos bloques de viviendas, los contadores no se encuentran dentro de las viviendas si no en zonas comunes, por lo que no es tan fácil consultar los datos.



Figura 2.1: Contador de luz[28]

### **Factura de la luz**

En la factura eléctrica podemos consultar el desglose del consumo general por tramos.

Ventajas:

- Desglose por tarifas del consumo.

Inconvenientes:

- No podemos conocer en tiempo real el consumo eléctrico de la vivienda.
- No podemos saber cuál es el consumo individual de dispositivos en nuestra red.

### **Calculo manual del consumo**

Podemos calcular cuánto gasta un dispositivo de manera fácil conociendo su potencia y el tiempo que está encendido. Mas adelante veremos mas en profundidad como hacerlo de diferentes formas.

Inconvenientes:

- Solo es posible saber el consumo teórico.

### Medidor de consumo electrico individual

Medidor de consumo para monitorizar la energía de un dispositivo conectado a la red eléctrica.

Ventajas:

- Protección por sobrecarga eléctrica incorporado.
- Pantalla incorporada para mostrar la lectura.

Inconvenientes:

- No es posible utilizar los datos de consumo fuera de la pantalla incorporada.
- No es posible consultar un histórico de consumo.



Figura 2.2: Medidor de consumo de energía[30]

### Medidor cuadro electrico

Permite medir la energía eléctrica de manera precisa y fiable.

Ventajas:

- Está pensado para instalarse directamente en el cuadro eléctrico.
- Permite medir amperajes muy altos.

Inconvenientes:

- Al instalarse en el cuadro eléctrico, la medida que nos da es genérica y no podemos saber cual es el consumo individual de ciertos dispositivos en nuestra red electrica.



Figura 2.3: Medidor de consumo en cuadro electrico[31]

### **Monitor inteligente de energía**

Sistema que agrega sensores individuales para monitorizar de forma precisa electrodomésticos y otros tipos de dispositivos. Integración con una app Android para consultar los datos. Esta opción sería lo más parecido al sistema que este proyecto plantea.

Ventajas:

- Sensores de hasta 50A.
- Cuenta con una aplicación para consultar los datos de cada sensor.

Inconvenientes:

- Los sensores se comunican con el panel principal por cable.
- Coste elevado.



Figura 2.4: Monitor de energía inteligente[32]

### 2.1.2. Calculo del consumo energético

Lo primero que necesitamos entender es como podemos calcular el consumo electrico de cualquier dispositivo. Sabiendo los datos teoricos, es algo bastante simple, tan solo necesitamos aplicar la siguiente formula:

$$P = I \cdot V \quad (2.1)$$

Donde P es el consumo en vatios, I es la intensidad en amperios y V es la tensión en voltios. La tensión es facil saberla. En España y en la mayoria de paises europeos, la tensión es de 230V. Pero la intensidad es algo mas complicado. Para saber la intensidad necesitamos saber la potencia del dispositivo. Para ello, podemos consultar la etiqueta de consumo de cada dispositivo. En la etiqueta de consumo, podemos ver la potencia en vatios, la intensidad en amperios y la tensión en voltios. Pero esta potencia que nos indica el dispositivo no es la real. Normalmente es la potencia maxima que puede llegar a consumir el dispositivo, pero no la que realmente consume. Para calcular la potencia real, podemos hacer uso de un multímetro y asi obtener la intensidad real, y con esto calcular la potencia que consume. En nuestra solución no podremos hacer uso de un multímetro, asi que necesitamos entender como hace el multímetro para calcular la intensidad.

### Onda sinusoidal

La corriente alterna viaja por el cableado en formas de ondas Sinusoidales. La forma de onda de la corriente alterna es la siguiente:

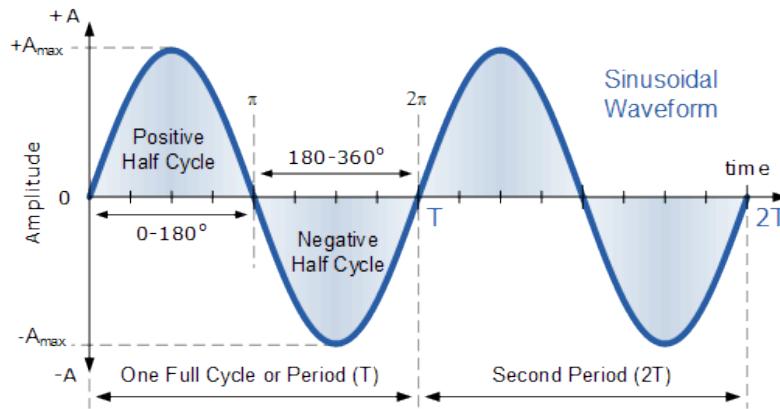


Figura 2.5: Forma de una onda Sinusoidal[27]

Aquí podemos diferenciar tres características de la onda sinusoidal[2]:

- Amplitud: Es la distancia entre el punto mas alto y el mas bajo de la onda. Esta es la intensidad de la señal de corriente.
- Frecuencia: Es el numero de veces que la onda se repite en un segundo. En España, la frecuencia es de 50Hz.
- Periodo: Es el tiempo que tarda en completarse una onda.

Para poder calcular correctamente el consumo de un dispositivo, tenemos que medir correctamente la amplitud de la onda, es decir, el maximo y minimo de la onda. Así es como los amperímetros y los multímetros calculan la intensidad.

### Factor de potencia

El factor de potencia[3] se utiliza como expresión para indicar la relación entre la potencia activa y la potencia aparente. La potencia aparente es la potencia que se mide en el medidor de la compañía eléctrica. La potencia activa es la potencia que realmente consume el dispositivo. Si un circuito eléctrico fuera cien por cien eficiente, la potencia activa y la potencia aparente serían iguales. Pero en la realidad, los circuitos no son cien por cien eficientes, y por tanto, la potencia activa es menor que la potencia aparente.

$$FP = \frac{P_{activa}}{P_{aparente}} \quad (2.2)$$

El factor de potencia es un valor que oscila entre 0 y 1. Cuanto mas cercano a 1, mas eficiente es el circuito. Cuanto mas cercano a 0, menos eficiente es el circuito. En aparatos electricos resistivos (como por ejemplo un radiador), el factor de potencia es muy cercano a 1 ya que convierten toda la energia electrica que reciben en calor. En aparatos electricos inductivos, como por ejemplo un motor, el factor de potencia es mucho mas bajo ya que no convierten toda la energia electrica que reciben en trabajo.

### Media cuadratica (RMS)

Para este proyecto, no nos interesa el el peak-to-peak de la onda, sino la media cuadratica[4] de la onda para poder saber la intensidad continua que consume el dispositivo. Para calcular la media cuadratica, podemos hacer uso de la siguiente formula:

$$V_{peak\_to\_peak} = V_{max} - V_{min} \quad (2.3)$$

$$VoltageRMS = \left( \frac{V_{peak\_to\_peak}}{2} \right) \cdot \sqrt{2} \quad (2.4)$$

### 2.1.3. Sensores de consumo energético

A continuacion vamos a ver los diferentes tipos de sensores que podemos utilizar para medir el consumo de un dispositivo en nuestro proyecto.

#### INA219

El sensor INA219[5] es un sensor de corriente y voltaje de alta precision. Este sensor es capaz de medir hasta 26V. Este sensor es muy util para medir el consumo de un dispositivo, ya que es capaz de medir tanto la intensidad como la tensión, como por ejemplo para monitorizar la duración de baterias.

Caracteristicas:

- Mide tensiones entre 0 y 26V.

- Interfaz de comunicación I2C.
- Alta precisión.
- Opciones de calibración para ajustar la sensibilidad.

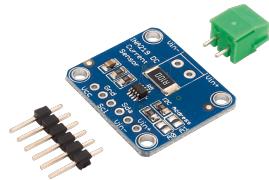


Figura 2.6: Sensor INA219[33]

### ZMPT101B

El sensor ZMPT101B[6] es capaz de medir hasta 1000V.

Características:

- Alto aislamiento.
- Rango amplio
- Alta precisión
- Resultados estables



Figura 2.7: Sensor ZMPT101B[34]

### ACS712

El sensor ACS712[7] es un sensor barato y preciso para medir la corriente tanto continua como alterna. Existen 3 modelos de este sensor, uno para medir corrientes de 5A, otro para medir corrientes de 20A y otro para medir corrientes de 30A.

Características:

- Alta precisión.
- Señal de salida con bajo ruido.



Figura 2.8: Sensor ACS712[26]

## 2.2. Plataformas de desarrollo

Para este proyecto, buscamos una plataforma de desarrollo de software embebido que disponga de ADCs y de una interfaz de comunicación inalámbrica para poder enviar los datos a un servidor. A continuación vamos a ver las diferentes opciones que tenemos.

### 2.2.1. Arduino

Arduino[8] es una plataforma de desarrollo de software embebido basada en microcontroladores. Arduino es una plataforma muy popular y tiene una gran comunidad de usuarios. Arduino tiene una gran cantidad de shields, que son placas de expansión que se pueden conectar a la placa base de Arduino. No dispone de WiFi integrado, pero podríamos conectarle un módulo WiFi. El Arduino UNO dispone de 6 entradas analógicas.

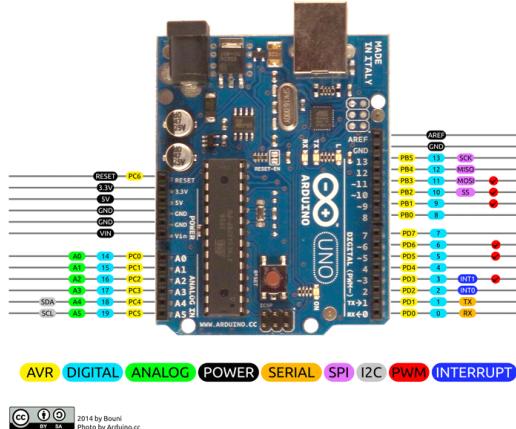


Figura 2.9: Arduino pin layout[35]

### 2.2.2. Raspberry Pi

La Raspberry Pi[9] es un mini ordenador de bajo coste (aunque mas caro que el resto de plataformas que vamos a ver). No incluye ningun convertidor analogico digital, por lo que tendriamos que usar un modulo externo. Si dispone de WiFi.

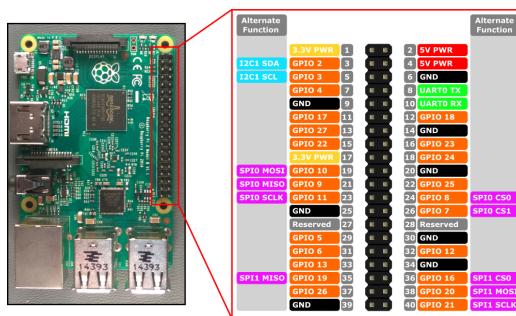


Figura 2.10: Raspberry Pi pin layout[36]

### 2.2.3. NodeMCU ESP8266

El NodeMCU ESP8266[10] es una plataforma de codigo abierto que como su propio nombre indica, incluye un modulo WiFi ESP8266. Es un proyecto ya abandonado pero que tiene una gran comunidad de usuarios que lo siguen manteniendo. Esta plataforma tan solo dispone de un ADC, pero

de resolución de 10 bits.

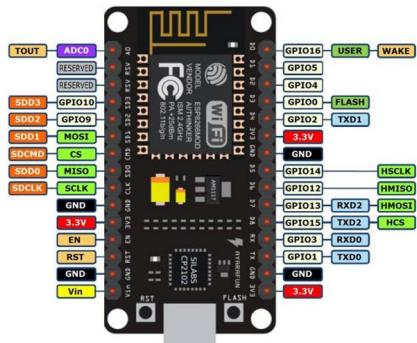


Figura 2.11: Node MCU ESP8266 pin layout[37]

#### 2.2.4. ESP32

El ESP32[1] es un microcontrolador con WiFi integrado. Dispone de hasta 18 entradas analogicas. Es el sucesor del ESP8266, por lo que es una plataforma mas moderna.

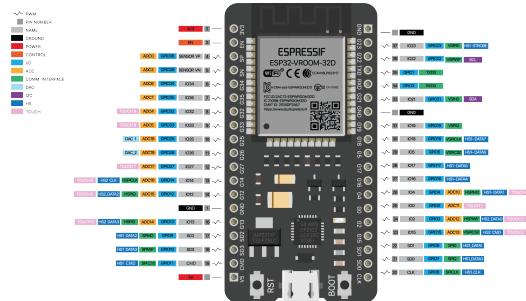


Figura 2.12: ESP32 pin layout[38]

## 2.3. Micropython

Micropython [12] es una implementación sencilla y eficiente de Python 3 para microcontroladores y otros entornos restringidos. Incluye un pequeño subconjunto de la biblioteca estándar de Python, y se puede extender

fácilmente con módulos escritos en C. Micropython es un lenguaje de programación de alto nivel, por lo que es muy sencillo de programar. El firmware de Micropython es muy ligero, por lo que tan solo necesitamos 256Kb de memoria flash y 16Kb de RAM para poder ejecutarlo.

Ejemplo de como cambiar el valor de un pin digital[13]:

```

1 from machine import Pin
2
3 p0 = Pin(0, Pin.OUT)      # create output pin on GPIO0
4 p0.on()                  # set pin to "on" (high) level
5 p0.off()                 # set pin to "off" (low) level
6 p0.value(1)              # set pin to on/high
7
8 p2 = Pin(2, Pin.IN)      # create input pin on GPIO2
9 print(p2.value())        # get value, 0 or 1
10
11 p4 = Pin(4, Pin.IN, Pin.PULL_UP) # enable internal pull-up resistor
12 p5 = Pin(5, Pin.OUT, value=1) # set pin high on creation
13 p6 = Pin(6, Pin.OUT, drive=Pin.DRIVE_3) # set maximum drive strength

```

## 2.4. Comunicaciones inalámbricas en IoT

Uno de los objetivos principales de este proyecto es poder enviar los datos que recoge cada sensor de corriente a un servidor para que se puedan procesar, analizar y mostrar en conjunto. Para ello, necesitamos alguna forma de comunicación inalámbrica.

### 2.4.1. MQTT

MQTT (Message Queuing Telemetry Transport) [14] es un protocolo de mensajería ligero que se utiliza para la comunicación entre dispositivos en IoT. MQTT es un protocolo de publicación/ suscripción, por lo que los dispositivos se suscriben a un topic y los mensajes se publican en ese topic.

Características principales:

- Ligero y eficiente. Los clientes necesitan pocos recursos para poder usar el protocolo, lo que lo hace ideal para usarlo en microcontroladores.
- Comunicación bidireccional. Los clientes pueden publicar y suscribirse a topics.

- Es un protocolo muy escalable.
- Seguridad. MQTT permite el uso de TLS para encriptar la comunicación o el uso de protocolos de autenticación.
- Fiabilidad en la entrega de mensajes. MQTT permite especificar el nivel de fiabilidad(Quality of Service) en diferentes niveles.

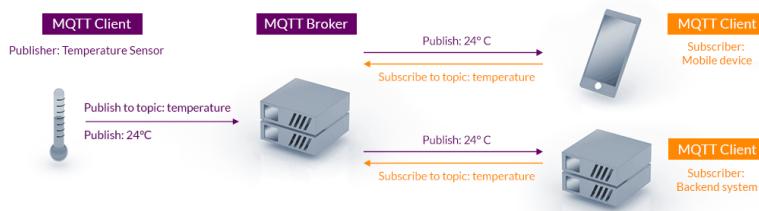


Figura 2.13: Ejemplo de dispositivos comunicados por MQTT[39]

Es importante destacar la importancia del broker en MQTT. El broker es el encargado de recibir los mensajes de los clientes y reenviarlos a los clientes suscritos al topic.

#### 2.4.2. Zigbee

Zigbee[15] es un estandar de comunicación inalámbrica que se utiliza para la comunicación entre dispositivos en IoT. Zigbee es un protocolo de red mesh, por lo que los dispositivos se comunican entre ellos y no con un broker central a diferencia de MQTT.

En Zigbee, podemos encontrar tres clases diferentes de dispositivos:

- Zigbee coordinator: Es el dispositivo mas capaz encargado de administrar la red y de coordinar la comunicación entre los dispositivos.
- Zigbee router: Es un dispositivo que se encarga de reenviar los mensajes de los dispositivos a otros dispositivos ademas de correr su propia aplicación
- Zigbee end device: Contiene lo basico para poder comunicarse con un nodo padre y correr su propia aplicación.

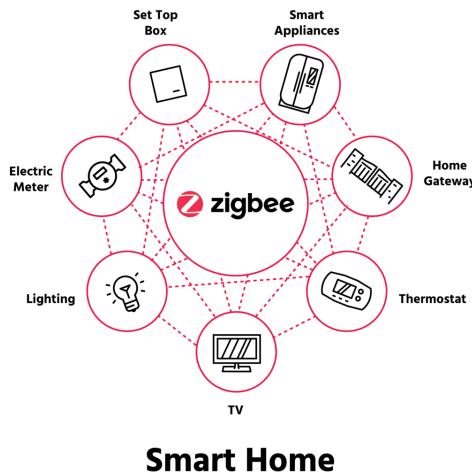


Figura 2.14: Ejemplo de dispositivos comunicados por Zigbee[40]

Características principales:

- Es un protocolo de red mesh.
- Protocolo de bajo consumo, baja latencia y bajo coste.
- Es un protocolo de baja potencia, por lo que es ideal para usarlo en dispositivos que necesitan una comunicación a corta distancia.
- Hasta 65000 nodos en una red.
- Soporta encriptación AES-128.

# Capítulo 3

## Implementación del sensor de corriente

### 3.1. Descripción general

Para el desarrollo de este proyecto estamos buscando una plataforma de bajo coste que permita comunicación inalámbrica entre dispositivos y que permita el uso de Micropython. Para ello hemos elegido la placa ESP32 de Espressif, que es de bajo coste y tiene un microcontrolador Tensilica LX6 dual con dos núcleos de procesamiento de 32 bits, 512 KB de SRAM y 4 MB de memoria flash. Además, tiene un módulo Wi-Fi y Bluetooth integrado. Se alimenta a 5V y tiene un consumo[16] de entre 160mA y 260mA con el WiFi activo.

Como sensor de corriente se va a utilizar el ACS712 de 5A y 30A para poder realizar diferentes mediciones y poder comparar los resultados.

Como broker MQTT se va a utilizar Mosquitto, que es un broker de código abierto que se puede instalar en cualquier sistema operativo. En este caso, en una Raspberry Pi 2B con Raspbian.

## 3.2. Setup experimental

Lo primero a tener en cuenta es que el sensor ACS712 necesita una tensión de alimentación de 5V y una corriente de 20mA. El pin de salida del sensor es un pin analógico que da una tensión entre 0 y 5V. Si conectaramos este pin directamente a un ADC del ESP32, podriamos llegar a dañar la placa puesto que los ADCs del ESP32 estan diseñados para trabajar con hasta un máximo de 3.3V (a nivel practica es un poco mas de 3.3V). Para evitar esto, se tiene que utilizar un divisor de voltaje que reduzca la tensión de salida del sensor a un valor que pueda leer el ADC. Para ello se ha utilizado una resistencia de 10k y otro de 20k para que el voltaje máximo de salida del sensor sea de 3.3V.

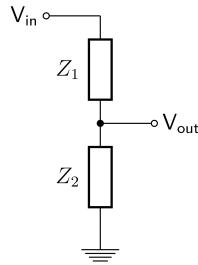


Figura 3.1: Divisor de voltaje[41]

El segundo componente a añadir es un filtro pasabajos para eliminar el ruido de la señal. Para ello se ha utilizado un capacitor de 100nF conectado entre el pin de salida del sensor y ground para que la señal pase por el capacitor y se elimine el ruido.

Lo siguiente es tener una pantalla LED o LCD para poder mostrar por ahí los datos que se van midiendo del sensor de corriente. Para ello se ha utilizado el modulo SSD1306 de 128x32 pixeles. Este modulo se comunica con el ESP32 mediante el protocolo I2C. Para poder comunicar el ESP32 con el modulo SSD1306 se ha utilizado la libreria de Adafruit para el ESP32 en Micropython.

Por último, queremos disponer de un botón de reset en el ESP32 en caso de que la configuración que guardemos del servidor MQTT cambie y queramos reiniciar la configuración interna. El ESP32 tiene dos botones que vienen en la placa, pero ninguno de ellos se puede cambiar su función por

software, por lo que debemos añadir un botón externo conectado a un pin del ESP32.

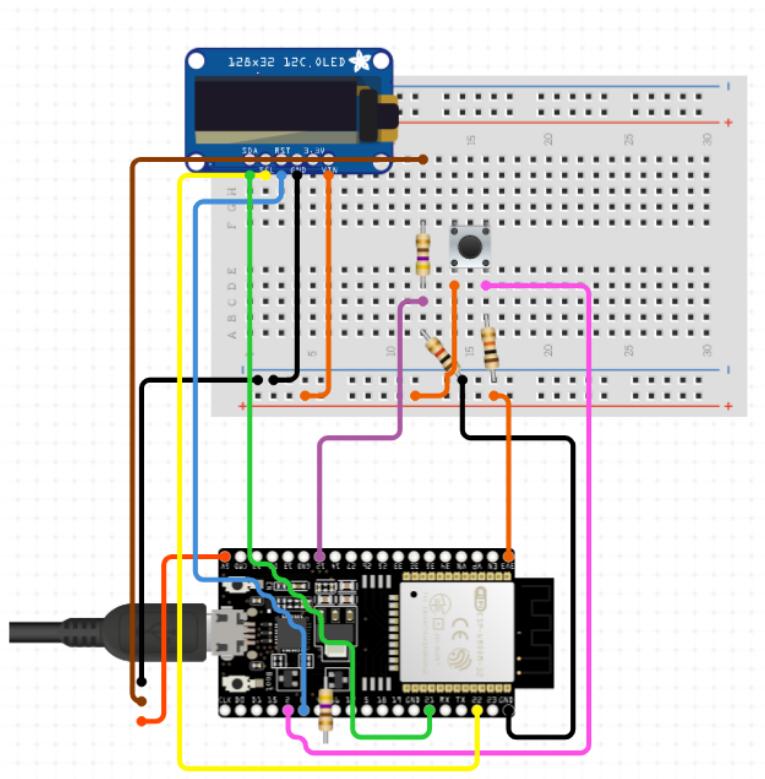


Figura 3.2: Setup experimental

### 3.2.1. Preparar ESP32 para Micropython

Antes de poder usar el ESP32 para ejecutar python, debemos de grabar el firmware correspondiente a la placa. El proceso es bastante sencillo utilizando la herramienta Esptool[17]:

- En lugar de instalar la herramienta usando pip de python, recomiendo descargar el código fuente desde la página de github[18] y ejecutar el archivo esptool.py directamente ya que al menos en mi experiencia da menos problemas.
  - Una vez con la herramienta instalada, lo primero es borrar completamente la memoria flash de la placa. Ejecutaremos el siguiente

comando(en mi caso, ttyUSB0 para indicar el puerto al que esta conectado):

```
1 esptool.py --port /dev/ttyUSB0 erase_flash
```

- Ahora, debemos de grabar el firmware de Micropython en la placa. Para ello, descargaremos el firmware (.bin) de la pagina oficial[19] y ejecutaremos el siguiente comando:

```
1 esptool.py --port /dev/ttyUSB0 write_flash -z 0x1000 esp32.bin
```

- Si todo ha ido bien, ya podremos ejecutar Micropython en la placa. Podemos comprobarlo conectandonos por el puerto serie a la placa (en el ejemplo uso picocom pero se puede utilizar cualquier software que permita la conexion por serial) y viendo si la placa nos responde con el prompt de Micropython:

```
1 $ picocom /dev/ttyUSB0 -b115200
```

```

local echo is : no
noinit is : no
noreset is : no
hangup is : no
nolock is : no
send_cmd is : sz -vv
receive_cmd is : rz -vv -E
imap is :
onmap is :
emap is : crcrlf,delbs,
logfile is : none
initstring : none
exit_after is : not set
exit is : no

Type [C-a] [C-h] to see available commands
Terminal ready

<function>
>>> import machine
>>> import ubinascii
>>> ubinascii.hexlify(machine.unique_id())
b'c049efccf254'
>>> 
```

Figura 3.3: Prompt de Micropython

### 3.2.2. Micropython en ESP32

Despues de instalar el firmware de Micropython en la placa, podremos observar que hay un archivo llamado boot.py que se ejecuta al arrancar la placa. En este archivo se puede configurar la placa para que se conecte a una red WiFi o para que se conecte a un servidor MQTT.

Despues de este boot.py, la placa intentará ejecutar el archivo main.py si existe. En este archivo se puede programar el comportamiento de la placa.

Al igual que en python, podemos hacer uso de diferentes librerias. Para poder usar librerias que no vengan por defecto en el firmware de Micropython, debemos de copiar los archivos .py a la flash de la placa con la herramienta Aypy de Adafruit[20]. Con Aypy podemos desde copiar un archivo a la flash de la placa o recuperar un archivo de la flash de la placa a nuestro ordenador.  
Uso de Aypy:

```
Usage: ampy [OPTIONS] COMMAND [ARGS]...
      ampy - Adafruit MicroPython Tool

      Aypy is a tool to control MicroPython boards over a serial connection.
      Using ampy you can manipulate files on the board's internal filesystem and
      even run scripts.

Options:
  -p, --port PORT      Name of serial port for connected board. Can optionally
                       specify with AMPY_PORT environment variable. [required]
  -b, --baud BAUD     Baud rate for the serial connection (default 115200).
                       Can optionally specify with AMPY_BAUD environment
                       variable.
  -d, --delay DELAY   Delay in seconds before entering RAW MODE (default 0).
                       Can optionally specify with AMPY_DELAY environment
                       variable.
  --version            Show the version and exit.
  --help               Show this message and exit.

Commands:
  get    Retrieve a file from the board.
  ls     List contents of a directory on the board.
  mkdir  Create a directory on the board.
  put    Put a file or folder and its contents on the board.
  reset  Perform soft reset/reboot of the board.
  rm    Remove a file from the board.
  rmdir  Forcefully remove a folder and all its children from the board.
  run    Run a script and print its output.
adrian@ubuntu:~$
```

Figura 3.4: Opciones de Aypy

### 3.2.3. Librerias utilizadas

#### Libreria para el SSD1306

Para usar el modulo SSD1306, se ha utilizado la libreria [21] en Micropython. Esta libreria nos permite inicializar el modulo SSD1306 y escribir texto en la pantalla de manera sencilla.

#### Libreria para MQTT

Para usar MQTT, se ha utilizado la libreria umqtt.simple[22] en Micropython. Esta libreria nos permite conectarnos a un servidor MQTT y publicar y suscribirse a diferentes topics.

### 3.3. Implementación

#### 3.3.1. Diagrama de estados

##### Init

La inicialización del programa empieza por conectarse a la red WiFi y al broker de MQTT. A continuación comprobamos si ya hemos establecido conexión previamente con la aplicación web. Si no se ha establecido previamente conexión con la aplicación web, empezaremos a publicar periódicamente el ID interno del ESP32 y esperaremos a que la aplicación web nos devuelva el ACK. Lo siguiente es comprobar si hemos recibido la configuración de la aplicación web (la configuración a recibir es el tipo de sensor y el voltaje al que estamos conectados). Por último calibraremos el sensor, es decir, leemos el voltaje de salida del ACS712 durante un periodo de tiempo sin que haya nada conectado al sensor y lo guardamos como voltaje de referencia para poder calcular la corriente en el futuro.

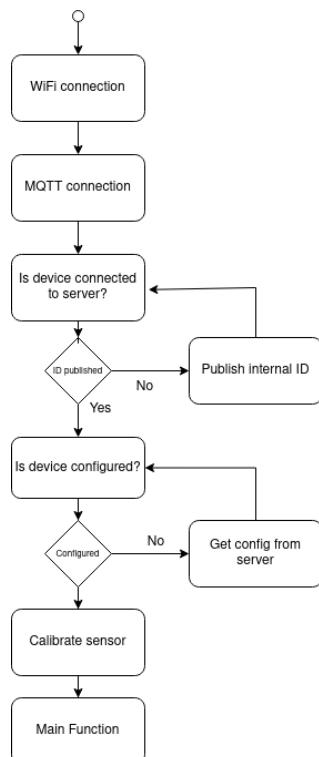


Figura 3.5: Diagrama de estados Init

### Main

La función principal es bastante sencilla. En cada iteración, lanzamos una llamada a la librería de MQTT para comprobar si hay mensajes en la cola recibidos (esto hará que se lance el callback de recepción de mensajes MQTT si es que hay alguno). Lo siguiente es leer el voltaje de salida del sensor de corriente y calcular el amperaje y consumo (en Watts). Estos watos calculados los mostramos por el dispositivo OLED que tenemos conectado. Por último, publicamos el valor del amperaje y consumo en el topic de MQTT correspondiente (watts/ID\_Sensor y amps/ID\_Sensor)

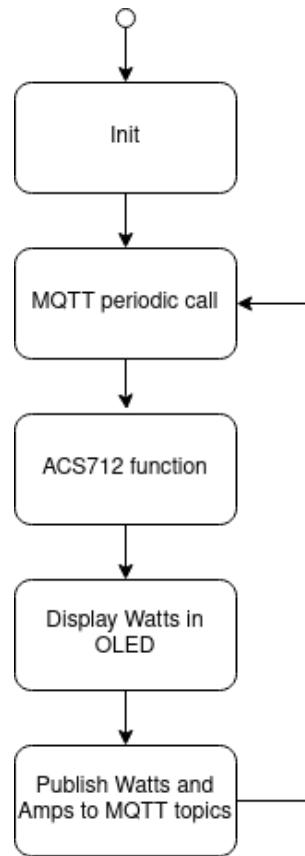


Figura 3.6: Diagrama de estados función Main

### Lectura del sensor

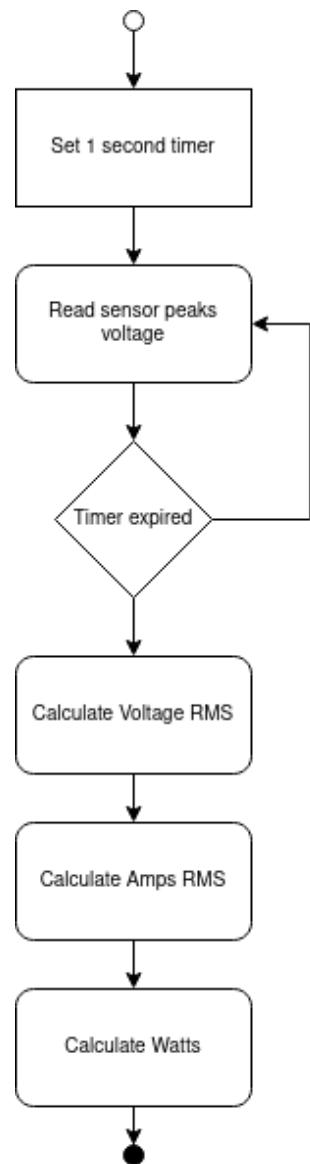


Figura 3.7: Diagrama de la función de lectura del ACS712

### 3.3.2. Código

#### Función Init

```

1 display_message('Connecting to WiFi...')
2 connect_wifi(SSID, PASSWORD)
3
4 #display_message('Connecting to MQTT broker...')
5 MQTT_CLIENT.connect()
6 MQTT_CLIENT.mqtt.set_callback(subscribe_callback)
7 if MQTT_CLIENT.is_broker_acknowledged() == False:
8     MQTT_CLIENT.publish_clientID()
9
10 #subscribe to reset topic
11 MQTT_CLIENT.subscribe('reset/' + MQTT_CLIENT.client_id.decode("utf-8"))
12 #subscribe to sensor_config topic to receive the sensor voltage
13 MQTT_CLIENT.subscribe('sensor_config/' + MQTT_CLIENT.client_id.decode("utf-8"))
14
15 display_message('Configuring sensor...')
16 if CURRENT_SENSOR.is_sensor_configured() == False:
17     MQTT_CLIENT.get_sensor_config_from_broker()
18
19 #subscribe to restart esp32 topic
20 MQTT_CLIENT.subscribe('restart/' + MQTT_CLIENT.client_id.decode("utf-8"))
21
22 display_message('Calibrating...')
23 CURRENT_SENSOR.calibrateSensorAC()
24
25 #subscribe to calibrate topic
26 MQTT_CLIENT.subscribe('calibrate/' + MQTT_CLIENT.client_id.decode("utf-8"))

```

### Función de calibración del sensor

```

1 def calibrateSensorAC(self, seconds=10):
2     """
3         Calibrate the sensor to get the average voltage
4         reading when there is no load.
5     """
6     print("[testCalibrateSensor]Calibrating sensor...")
7     print("[testCalibrateSensor]Please disconnect the sensor \
8             from the load NOW!")
9     for i in reversed(range(5)):
10         print("[testCalibrateSensor]Starting calibration in " +
11               str(i+1) + " seconds...")
12         utime.sleep(0.5)
13     print("[testCalibrateSensor]Calibrating...")
14     counter = 0
15     voltage_RMS = 0
16     average_voltage = 0
17     start = time.ticks_ms()
18     while time.ticks_ms() - start < (1000 * seconds):
19         min_value = 4095
20         max_value = 0
21         voltage = 0
22         for i in range(100):
23             adc_raw = self.adc.read()
24             voltage += adc_raw
25             if adc_raw < min_value:
26                 min_value = adc_raw

```

```

28         max_value = adc_raw
29         time.sleep_us(500)
30         average_voltage += ((voltage * 3.3) / 4095) / 100
31         counter = counter + 1
32         max_value_V = ((max_value * 3.3) / 4095)
33         min_value_V = ((min_value * 3.3) / 4095)
34         peak_to_peak = (max_value_V - min_value_V)
35         voltage_RMS += peak_to_peak * 0.3536
36     end = time.ticks_ms()
37     print("[readAmps2]Time to read ", counter, " values: ",
38           time.ticks_diff(end, start), "ms")
39     voltage_RMS = voltage_RMS / counter
40     if voltage_RMS > 0.020:
41         self.default_VRMS = voltage_RMS
42     else:
43         self.default_VRMS = 0.020
44     self.default_output_voltage = average_voltage / counter
45 #add 5mV to the default output voltage to avoid negative values
46     self.default_VRMS = self.default_VRMS + 0.005
47     print("[calibrateSensorAC]Sensor calibrated. Default output \
48           voltage: ", self.default_output_voltage, "V")
49     print("[calibrateSensorAC]Sensor calibrated. Default VRMS: ",
50           self.default_VRMS, "V")

```

## Función Main

```

1 def init():
2
3     while True:
4         MQTT_CLIENT.check_msg()
5         watts, amps = CURRENT_SENSOR.getACWatts(logging=False)
6         display_watts(watts)
7         MQTT_CLIENT.publish('amps/' +
8             MQTT_CLIENT.client_id.decode("utf-8"), str(amps))
9         MQTT_CLIENT.publish('watts/' +
10            MQTT_CLIENT.client_id.decode("utf-8"), str(watts))
11         print('-----')

```

## Funciones de lectura del sensor

```

1 def readSensorVPP(self, logging=False):
2     """
3         Capture the peak to peak voltage of the current ADC signal.
4         Period of measurement is 50ms. The ADC is sampled every 500us.
5         100 samples are taken. AC current in spain is 50Hz.
6         We should be able to capture at least 2 full cycles of the
7         sinusoidal signal with its peaks.
8     """
9     min_value = 4095
10    max_value = 0
11    average_value = 0
12    start = time.ticks_ms()
13    for i in range(100):

```

```

14         adc_raw = self.adc.read()
15         average_value += adc_raw
16         if adc_raw < min_value:
17             min_value = adc_raw
18         elif adc_raw > max_value:
19             max_value = adc_raw
20         time.sleep_us(500)
21     end = time.ticks_ms()
22     max_value_V = ((max_value * 3.3) / 4095)
23     min_value_V = ((min_value * 3.3) / 4095)
24     average_value_V = ((average_value / 100) * 3.3) / 4095
25     peak_to_peak = (max_value_V - min_value_V)
26     if logging:
27         print("[readSensorVPP]Time to read 100 samples: ",
28               time.ticks_diff(end, start), "ms")
29         print("[readSensorVPP]Average value: ", average_value_V, "V")
30         print("[readSensorVPP]Max value: ", max_value_V, "V; Min \
31               value: ", min_value_V, "V; VPP: ", peak_to_peak, "V")
32
33     return peak_to_peak, average_value_V
34
35 def readRMSVoltage(self, logging=False):
36     voltage_RMS = 0
37     voltage_RMS_nofilter = 0
38     average_voltage = 0
39     samplesVPP = list()
40     samplesV = list()
41     start = time.ticks_ms()
42     while time.ticks_ms() - start < 1000:
43         voltagePP, voltage = self.readSensorVPP(logging=False)
44         samplesVPP.append(voltagePP * 0.3536)
45         samplesV.append(voltage)
46     end = time.ticks_ms()
47     for i in range(len(samplesVPP)):
48         voltage_RMS_nofilter += samplesVPP[i]
49         average_voltage += samplesV[i]
50     """
51     for sample in samplesVPP:
52         voltage_RMS_nofilter += sample
53     """
54     voltage_RMS_nofilter = voltage_RMS_nofilter / len(samplesVPP)
55     average_voltage = average_voltage / len(samplesV)
56     #apply sw filter
57     samplesVPP = self.RMSFilter(samplesVPP, logging=False)
58     for sample in samplesVPP:
59         voltage_RMS += sample
60     voltage_RMS = voltage_RMS / len(samplesVPP)
61
62     if logging:
63         print("[readRMSVoltage]Time to read ", len(samplesV),
64               " values: ", time.ticks_diff(end, start), "ms")
65         print("[readRMSVoltage]Voltage RMS: ", voltage_RMS * 1000,
66               "mV", " default VRMS: ", self.default_VRMS * 1000, "mV")
67         print("[readRMSVoltage]Voltage RMS no filter: ",
68               voltage_RMS_nofilter * 1000, "mV", " default VRMS: ",
69               self.default_VRMS * 1000, "mV")
70         print("[readRMSVoltage]Average voltage: ",
71               average_voltage * 1000, "mV", " default output voltage: ",
72               self.default_output_voltage * 1000, "mV")
73
74     return voltage_RMS, average_voltage
75

```

```

76 def readRMSAmps(self, logging=False):
77     voltage_RMS, voltage = self.readRMSVoltage(logging=logging)
78     #decide whether to apply an error correction or not
79     if self.checkZeroRange(voltage_RMS, voltage, logging=logging):
80         Amps_RMS = 0
81     else:
82         Amps_RMS = (voltage_RMS * 1000) / self.scale_factor
83
84     amps = abs(((voltage - self.default_output_voltage) * 1000)
85                / self.scale_factor)
86     if logging:
87         print("[readRMSAmps] Current: ", Amps_RMS, "A RMS")
88         print("[readRMSAmps] Current: ", amps, "A")
89
90     return Amps_RMS, amps
91
92 def getACWatts(self, logging=False):
93     """
94     Calculate the power consumption in Watts.
95     """
96     Amps_RMS, amps = self.readRMSAmps(logging=logging)
97     if self.type == "AC":
98         watts = self.referenceVoltage * Amps_RMS
99         ret_amps = Amps_RMS
100    elif self.type == "DC":
101        watts = self.referenceVoltage * amps
102        ret_amps = amps
103    else:
104        watts = 0
105    if logging:
106        print("[getACWatts]Watts: ", watts, "W", "with sensor: ",
107              self.type)
108
109    return watts, ret_amps

```

El código completo puede ser consultado en el siguiente enlace[23].

## 3.4. Test de funcionamiento

### 3.4.1. Corriente continua

Para probar diferente amperajes se ha utilizado una fuente de alimentación que permite variar la corriente de salida entre 0 y 3A. La tensión de la fuente se ha fijado en 12V.

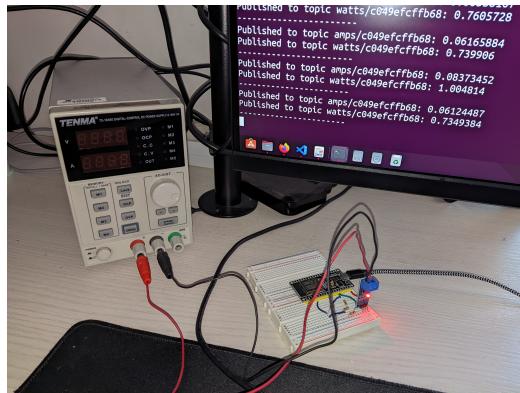


Figura 3.8: Salida del ACS712 con corriente continua de 0A

### No load

Como podemos observar, tras la conversión de la señal del ACS712 a amperios, tenemos valores muy cercanos a 0A que son prácticamente despreciables.

### 1A

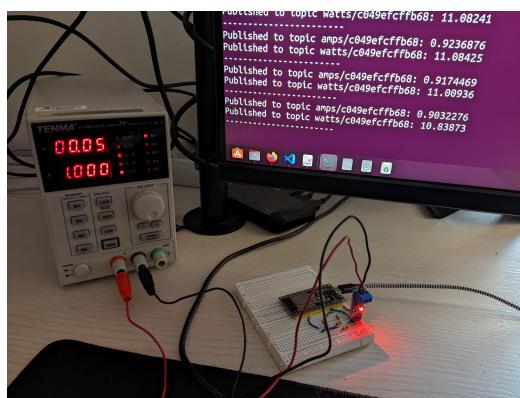


Figura 3.9: Salida del ACS712 con corriente continua de 1A

Generando 1A de corriente continua, medimos 0.91A con el sensor ACS712.

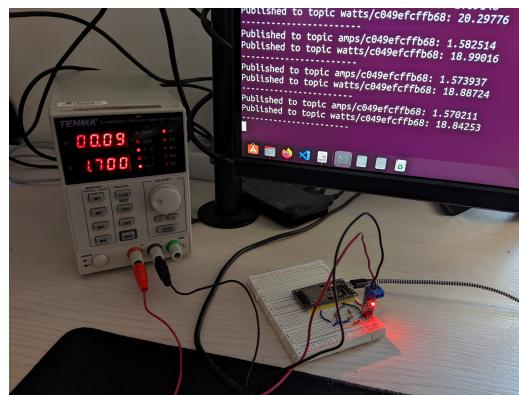
**1.7A**

Figura 3.10: Salida del ACS712 con corriente continua de 1.7A

Generando 1.7A de corriente continua, medimos 1.58A con el sensor ACS712.

**3A**

Figura 3.11: Salida del ACS712 con corriente continua de 3A

Generando 3A de corriente continua, medimos 2.82A con el sensor ACS712.

### 3.4.2. Corriente alterna

Como no dispongo de un generador de corriente alterna, he utilizado diferentes aparatos electricos para poder medir el consumo y el resultado del ACS712 se ha comparado con un amperimetro. El ACS712 esta conectado midiendo la corriente que pasa por el cable de una regleta de enchufes.

#### No load

Para la prueba de no tener ningun dispositivo conectado, simplemente se ha medido sin tener ningun enchufe conectado a la regleta.

Como podemos observar, ACS712 nos da un valor de 0A al igual que el



Figura 3.12: Nada conectado al enchufe

amperimetro.

#### Secador aire frío

La primera prueba con baja corriente se ha puesto un secador de pelo con la función de solo aire frio. Como solo esta funcionando el motor del secador, podemos observar como hay mas diferencia entre lo que mide el ACS712 y el amperimetro.

El amperimetro mide 0.701A y el ACS712 0.961A.

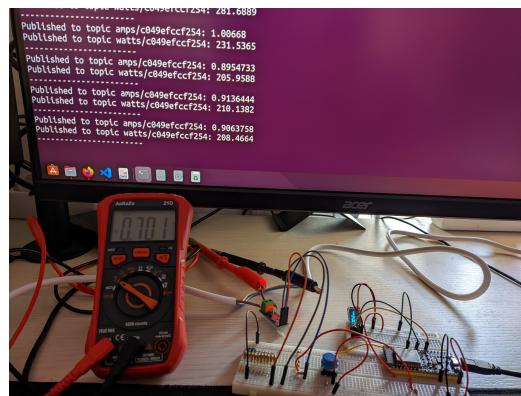


Figura 3.13: Secador de pelo solo con la función de aire frio

### Secador aire caliente al minimo

La siguiente prueba la hacemos con el secador de pelo con la función de aire caliente pero a nivel bajo de calor.

Al estar usando el secador las resistencias para generar aire caliente,



Figura 3.14: Secador de pelo solo con la función de aire caliente a nivel bajo de calor.

podemos observar como aqui si que medimos con el ACS712 casi igual que con el amperimetro. El amperimetro mide 2.8A y el ACS712 2.75A.

### Secador aire caliente al máximo

La última prueba la hacemos con el secador de pelo con la función de aire caliente al máximo nivel de calor.

El ACS712 podemos observar como mide 4.95A y el amperimetro 4.963A.

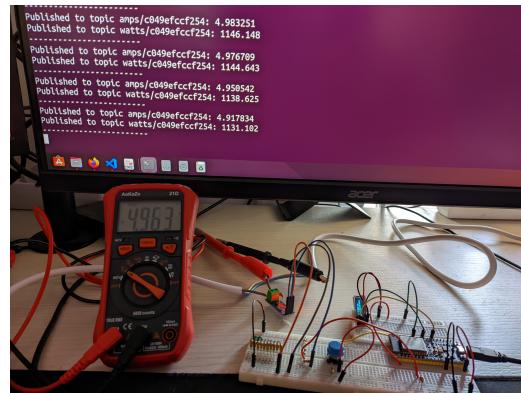


Figura 3.15: Secador de pelo solo con la función de aire caliente a nivel máximo de calor.

# Capítulo 4

## Implementación de la aplicación web

### 4.1. Descripción general

El objetivo de la aplicación web es que sirva de interfaz para mostrar los datos recopilados de todos los sensores que se encuentren configurados en la red.

### 4.2. Implementación backend

El backend de la aplicación web se ha implementado en Python utilizando el framework Flask, SocketIO y Paho para MQTT. Flask permite crear aplicaciones web de forma rápida y sencilla. SocketIO permite que la aplicación web sea capaz de recibir mensajes en tiempo real y comunicarse de forma facil con el cliente. Paho es una librería para MQTT que permite conectarse a un broker MQTT y publicar y suscribirse a topics.

#### 4.2.1. Aplicación en python3

##### Configuración de Flask

Con la siguiente configuración, creamos una aplicación FLask pudiendo indicarle a la vez toda la configuración MQTT a usar.

```
1 app = Flask(__name__)
```

```

2     app.config['SECRET'] = 'my secret key'
3     app.config['TEMPLATES_AUTO_RELOAD'] = True
4     app.config['MQTT_BROKER_URL'] = '192.168.1.200'
5     app.config['MQTT_BROKER_PORT'] = 1883
6     app.config['MQTT_CLIENT_ID'] = 'APP_SERVER'
7     app.config['MQTT_CLEAN_SESSION'] = True
8     app.config['MQTT_USERNAME'] = 'pi'
9     app.config['MQTT_PASSWORD'] = 'nairda'
10    app.config['MQTT_KEEPALIVE'] = 5
11    app.config['MQTT_TLS_ENABLED'] = False
12    app.config['MQTT_LAST_WILL_TOPIC'] = 'home/lastwill'
13    app.config['MQTT_LAST_WILL_MESSAGE'] = 'bye'
14    app.config['MQTT_LAST_WILL_QOS'] = 2

```

## Background tasks

Esta clase se encarga de ejecutar las tareas en segundo plano. Una tarea irá actualizando periodicamente el coste de la electricidad, y la otra tarea se ejecutara cuando sea necesario enviar la configuración a un sensor por MQTT.

```

1 class backgroundTask():
2     def __init__(self):
3         self.sendConfigToSensor = False
4
5     def stop_sendConfigToSensor(self):
6         logging.debug('stop_sendConfigToSensor called')
7         self.sendConfigToSensor = False
8
9     def start_sendConfigToSensor(self, json_data):
10        self.sendConfigToSensor = True
11        data = json.loads(json_data)
12        logging.info("Starting background thread to send config to sensor")
13        while self.sendConfigToSensor:
14            logging.info("Running loop to send config to sensor")
15            logging.info('Sending config to sensor %s, \n',
16                         data['sensor_type'] + ',' + data['voltage'])
17            mqtt.publish('sensor_config/' + data['sensor_id'], \
18                         data['sensor_type'] + ',' + data['voltage'])
19            socketio.sleep(1)
20
21     def update_cost_electricity(self):
22
23         while True:
24             costElectricity.load_current_data()
25             logging.debug("Reloading cost electricity data :",
26                           costElectricity.current_data)
27             #notify socket new data
28             socketio.emit('cost_electricity', costElectricity.current_data)
29             socketio.sleep(60)

```

## Clase para gestionar los sensores

Esta clase se encarga de gestionar los sensores, almacenando los datos de los sensores en un JSON. Tenemos una para añadir un sensor al JSON, otra función para leer los sensores del JSON y otra para eliminar un sensor del JSON.

```

1 class handleSensors():
2     def __init__(self):
3         self.JSON_FILE = 'sensors.json'
4         self.sensors_data = None
5
6     def read_saved_sensors(self):
7         if os.path.isfile(self.JSON_FILE):
8             with open(self.JSON_FILE, "r") as f:
9                 self.sensors_data = json.load(f)
10                print(self.sensors_data)
11            else:
12                with open(self.JSON_FILE, "w", encoding='utf-8') as f:
13                    json.dump({}, f, ensure_ascii=False, indent=4)
14                    logging.debug("No sensors found")
15
16    def add_sensor(self, sensor):
17        with open(self.JSON_FILE, "r") as f:
18            json_sensors = json.load(f)
19            sensorFound = False
20            for sensor_id, sensor_data in json_sensors.items():
21                print("sensor_id: ", sensor_id, "sensor_type: ",
22                      sensor_data["sensor_type"])
23
24            if not sensorFound:
25                logging.debug("New sensor added: %s", sensor["sensor_id"])
26                json_sensors[sensor["sensor_id"]] = sensor
27                self.sensors_data = json_sensors
28                with open(self.JSON_FILE, "w", encoding='utf-8') as f:
29                    json.dump(json_sensors, f, ensure_ascii=False, indent=4)
30                    #after adding, subscribe to watts topic
31                    mqtt.subscribe('watts/' + sensor["sensor_id"])
32
33    def remove_sensor(self, sensor_id):
34        with open(self.JSON_FILE, "r") as f:
35            json_sensors = json.load(f)
36            json_sensors.pop(sensor_id)
37            self.sensors_data = json_sensors
38            with open(self.JSON_FILE, "w", encoding='utf-8') as f:
39                json.dump(json_sensors, f, ensure_ascii=False, indent=4)
40                #after removing, unsubscribe to watts topic
41                mqtt.unsubscribe('watts/' + sensor_id)

```

## Funciones de la APP FFlask

Cada vez que un cliente pida una dirección de la web, estas son las funciones que se ejecutan, que tan solo llaman a renderizar plantillas html.

```

1 @app.route('/')
2 def index():
3     return render_template('index.html')
4
5 @app.route('/add_sensor')
6 def add_sensor():
7     return render_template('add_sensor.html')
8
9 @app.route('/sensors')
10 def sensors():
11     print("sensors " + str(handleSensors.sensors_data))
12     return render_template('sensors.html',
13                           sensors=handleSensors.sensors_data)
14
15 @app.route('/sensors/<path:sensor_id>')
16 def sensor(sensor_id):
17     return render_template('display_sensor_data.html',
18                           data=handleSensors.sensors_data[sensor_id])

```

---

### Llamadas a socketio

Estas son las llamadas a socketio que se ejecutan cuando se produce un evento. Para cada evento, se ejecutan diferentes acciones. Los eventos los pueden haber generado los clientes o la propia aplicación llamando a socketio.emit('EVENTO').

```

1 @socketio.on('publish')
2 def handle_publish(json_str):
3     data = json.loads(json_str)
4     mqtt.publish(data['topic'], data['message'], data['qos'])
5
6
7 @socketio.on('subscribe')
8 def handle_subscribe(json_str):
9     data = json.loads(json_str)
10    mqtt.subscribe(data['topic'], data['qos'])
11
12 @socketio.on('submit_sensor')
13 def handle_submit_sensor(json_str):
14     data = json.loads(json_str)
15     logging.info("New sensor added: %s", data['sensor_id'])
16     mqtt.publish('ack/' + data['sensor_id'], "ack", 0)
17     backgroundTask.start_sendConfigToSensor(json_str)
18     handleSensors.add_sensor(data)
19
20 @socketio.on('stop_sending_config')
21 def handle_stop_sending_config():
22     backgroundTask.stop_sendConfigToSensor()
23
24 @socketio.on('reset_sensor')
25 def handle_reset_sensor(str):

```

---

```

26     logging.debug("Reset sensor: %s", str)
27     mqtt.publish('restart/' + str, "restart", 0)
28
29 @socketio.on('calibrate_sensor'):
30 def handle_calibrate_sensor(str):
31     logging.debug("Recalibrating sensor: %s", str)
32     mqtt.publish('calibrate/' + str, "reset", 0)
33
34 @socketio.on('delete_sensor'):
35 def handle_delete_sensor(str):
36     logging.debug("Deleting sensor: %s", str)
37     handleSensors.remove_sensor(str)
38     mqtt.publish('reset/' + str, "reset", 0)
39
40 @socketio.on('unsubscribe_all'):
41 def handle_unsubscribe_all():
42     mqtt.unsubscribe_all()
43
44 @socketio.on('unsubscribe_sensor_id'):
45 def handle_unsubscribe_sensor_id():
46     mqtt.unsubscribe('clientID/broker')
47
48 @socketio.on('connect'):
49 def handle_connect():
50     logging.info("Client connected")
51     socketio.start_background_task(backgroundTask.update_cost_electricity)

```

## Funciones de MQTT

Cada vez que recibamos un mensaje en uno de los topics a los que se haya suscrito la app, se ejecutara la función on\_message. Para cuando MQTT se conecta al broker, se ejecuta la función on\_connect que lo unico que hace es suscribirse al topic de cada sensor para recibir los datos de consumo.

```

1 @mqtt.on_message()
2 def handle_mqtt_message(client, userdata, message):
3     data = dict(
4         topic=message.topic,
5         payload=message.payload.decode(),
6         qos=message.qos,
7     )
8     print("message received: ", data)
9     socketio.emit('mqtt_message', data=data)
10
11 @mqtt.on_connect()
12 def handle_mqtt_connect(client, userdata, flags, rc):
13     for sensor, data in handleSensors.sensors_data.items():
14         logging.debug("Subscribing to sensor: %s", sensor)
15         mqtt.subscribe('watts/' + sensor, 0)

```

### Clase para obtener precio KWh

Para obtener el precio del KWh se ha utilizado la API de la web de <https://api.preciodelaluz.org/>. Esta API devuelve el precio del KWh en tiempo real.

```

1 import requests
2 import json
3
4
5 class costElectricity:
6     """
7         Class to get the price of electricity in Spain
8     """
9
10    def __init__(self):
11        self.url_complete = 'https://api.preciodelaluz.org/v1/prices/all?zone=PCB'
12        self.url_average = 'https://api.preciodelaluz.org/v1/prices/avg?zone=PCB'
13        self.url_max = 'https://api.preciodelaluz.org/v1/prices/max?zone=PCB'
14        self.url_min = 'https://api.preciodelaluz.org/v1/prices/min?zone=PCB'
15        self.url_current = 'https://api.preciodelaluz.org/v1/prices/now?zone=PCB'
16        self.url_eco = 'https://api.preciodelaluz.org/v1/prices/cheapest?zone=PCB&n='
17
18        self.complete_data = None
19        self.average_data = None
20        self.max_data = None
21        self.min_data = None
22        self.current_data = None
23
24
25
26
27
28
29    def get_url_data(self, url):
30        response = requests.get(url)
31        if response.status_code == 200:
32            return json.loads(response.text)
33        else:
34            print("Error getting data from " + url)
35            return None
36
37
38    def load_complete_data(self):
39        self.complete_data = self.get_url_data(self.url_complete)
40
41    def load_average_data(self):
42        self.average_data = self.get_url_data(self.url_average)
43
44    def load_max_data(self):
45        self.max_data = self.get_url_data(self.url_max)
46
47    def load_min_data(self):
48        self.min_data = self.get_url_data(self.url_min)
49
50    def load_current_data(self):
51        self.current_data = self.get_url_data(self.url_current)
52
53    def update_everything(self):

```

```

54     self.load_complete_data()
55     self.load_average_data()
56     self.load_max_data()
57     self.load_min_data()
58     self.load_current_data()
59
60     """ Returns a list of the n cheapest prices in the day """
61 def get_eco_price(self, n):
62     return self.get_url_data(self.url_eco + str(n))

```

## Inicialización de la aplicación

```

1 if __name__ == '__main__':
2     handleSensors.read_saved_sensors()
3     costElectricity.load_current_data()
4     socketio.run(app, host='localhost', port=5000, use_reloader=False, debug=True)

```

El código completo se puede consultar en el repositorio de GitHub[25]

## 4.3. Implementación frontend

### 4.3.1. Diseño

#### Añadir sensor

Cosas a destacar de la interfaz de añadir un sensor:

- El botón de búsqueda hará que si hay algún sensor publicando su ID en el topic correspondiente, se añadirá automáticamente al campo "Sensor ID".
- Podemos elegir el tipo de sensor que estamos añadiendo, es decir, podremos elegir entre el sensor ACS712 de 5A, el de 20A o el de 30A.
- Otra cosa que debemos configurar bien es el "Load voltage", es decir, la tensión a la que se le aplica la carga al sensor. Esto es importante ya que el sensor ACS712 necesita saber la tensión a la que se le aplica la carga para poder calcular el consumo eléctrico correctamente.
- Una vez le demos al botón de Añadir, el servidor enviará la configuración al esp32 que corresponda.

The screenshot shows a web-based configuration interface for adding a new sensor. At the top, there are three buttons: 'Show global data', 'Add sensor' (which is highlighted in blue), and 'List sensors'. Below these, a title 'New sensor: Fill configuration' is displayed. The form contains five input fields: 'Sensor ID' (with a 'Search' button next to it), 'Sensor name', 'Sensor type' (set to '5A'), 'Load voltage' (set to '12'), and 'Sensor location'. A large blue 'Add' button is located at the bottom right of the form area.

Figura 4.1: Interfaz para añadir un sensor

### Mostrar sensores

Aquí simplemente saldrá una lista de los sensores que haya configurados.

The screenshot shows a web-based interface for listing configured sensors. At the top, there are three buttons: 'Show global data', 'Add sensor' (disabled and greyed out), and 'List sensors' (highlighted in blue). Below these, the text 'Sensors configured:' is displayed, followed by a bulleted list: '• Regleta' and '• 12V power supply'. The background is white with a light gray header bar.

Figura 4.2: Interfaz para mostrar los sensores añadidos

### Mostrar datos de un sensor

En esta interfaz se mostrarán los datos del sensor elegido. Por un lado tenemos una grafica que va mostrando en tiempo real los datos que se van recibiendo del sensor en Watts. Por otro lado podemos ver una tabla con los datos del consumo actual, el coste en euros del consumo actual, el precio del KWh y el consumo total en KWh. Tambien tenemos un cuadro con la información del sensor y unos botones para enviar un reset al esp32, que se calibre de nuevo el sensor y para eliminar el sensor configurado. Por ultimo se muestra una especie de consola con los ultimos 20 valores recibidos del sensor.

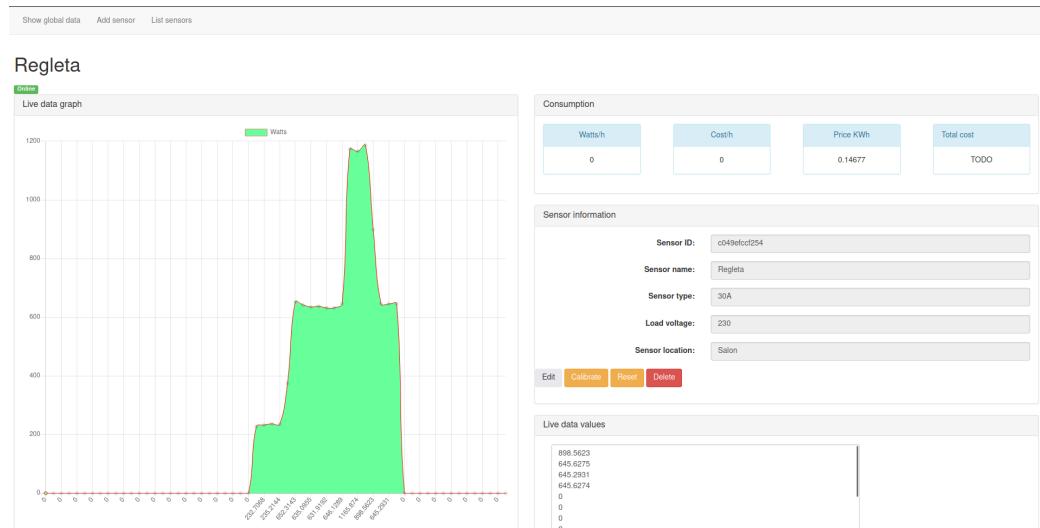


Figura 4.3: Interfaz para mostrar los datos de un sensor

### 4.3.2. Implementación

Para la implementación del frontend se ha usado html, javascript y bootstrap. El código completo se puede consultar en el repositorio de GitHub[25] en la carpeta de templates.



# Capítulo 5

## Conclusiones

En este último capítulo vamos a recopilar todo lo aprendido en el transcurso de este proyecto, vamos a describir las conclusiones sacadas en cada uno de los escenarios en los que hemos podido probar el sistema desarrollado y vamos a hacer una valoración final del proyecto.

### 5.1. Objetivos marcados

A continuación voy a describir los objetivos marcados al inicio del proyecto y voy a indicar uno por uno si se han cumplido. Para ello lo voy a dividir en las diferentes fases en las que fue pensado inicialmente.

#### 5.1.1. Fase 1

1. Conocer en profundidad cómo funciona MicroPython
  - a) Conocer las limitaciones de MicroPython. ✓
  - b) Escribir varios programas de prueba para familiarizarse con el entorno. ✓
2. Preparar el Esp32 para poder ejecutar MicroPython.
  - a) Borrar flash interna. ✓
  - b) Escribir MicroPython firmware en el microcontrolador. ✓
3. Integrar el sensor de corriente ACS712 con el Esp32.

- a) Programa que muestre a un LCD o por consola el consumo cada segundo. ✓
- b) Añadir una función para encender o apagar la monitorización del sensor con un botón.✓

### **5.1.2. Fase 2**

1. Configurar la Raspberry Pi como broker MQTT
2. Configurar el esp32 como publisher de MQTT.
  - a) Hacer una primera aproximación con un programa que publique el estado (encendido o apagado) de un led. ✓
  - b) Integrar MQTT en el programa de la tarea anterior con el sensor de corriente. El esp32 publicará cada segundo los datos del ACS712. ✓
  - c) Hacer que el Esp32 sea capaz de recibir órdenes de la raspberry para sustituir el botón de la primera tarea por un comando para apagar o encender la monitorización. ✓
  - d) Estudiar la viabilidad de implementar el deep-sleep del esp32 para que no consuma nada mientras no estamos monitorizando. Estudiar las diferentes fuentes disponibles para despertar al esp32.X
3. Crear aplicación temporal que muestre los datos recibidos de los diferentes publishers por la consola. ✓

### **5.1.3. Fase 3**

1. Diseñar interfaz de la app web. ✓
2. Mostrar en tiempo real los datos publicados por los sensores en la app web. ✓

## 5.2. Trabajo futuro

### 5.2.1. Mejoras en el sensor de corriente

La primera linea de continuación de este proyecto seria comparar los resultados obtenidos con el ACS712 con otro sensor de corriente como el ZMPT101B. Aunque los resultados obtenidos con el ACS712 en comparación con el amperímetro son bastante buenos, seria interesante compararlos igualmente con otro sensor de corriente.

Actualmente, el sistema es tan solo un prototipo que se alimenta por USB. Para poder hacer un sistema mas real, seria necesario añadir una fuente de alimentación[24] que convierta los 230V de corriente alterna a 5V de corriente continua.

Otra mejora que se podria añadir es la de poder controlar si encender o apagar los dispositivos conectados al sistema remotamente. Para ello, podriamos añadir un relé que se controlase mediante MQTT. Y a su vez, si no estamos alimentando con el rele, podriamos investigar como implementar el deep-sleep del ESP32 para que se quede sin consumir mientras esta en idle.

Tal y como se puede leer en el datasheet del ESP32, cuando el WiFi esta encendido, se produce un ruido sobre los ADCs que puede llegar a falsear los datos de medición del consumo electrico. Por ello, seria interesante añadir un ADC externo mas preciso que no se vea afectado por el ruido del WiFi y comparar los resultados para ver si merece la pena usar el ADC externo.

Por ultimo, actualmente la configuración del WiFi (SSID y contraseña) y la IP del broker MQTT estan hardcodeados en el codigo. Esto es funcional a nivel de prototipo, pero para un sistema real seria necesario poder configurar estos datos de manera remota. Para ello, se podria añadir que por defecto el ESP32 se pusiera en modo punto de acceso WiFi a la vez que lanzara un servidor web para poder configurar los datos de conexión.

### 5.2.2. Mejoras en la aplicación web

Respecto a la aplicación Web, lo principal seria añadir la funcionalidad del panel central que muestre los datos de todos los sensores a la vez.

Otra mejora importante seria añadir el historico de datos. Para ello deberiamos de configurar una base de datos donde guardasemos por ejemplo los datos de las ultimas 24h. De esta manera, al refrescar la pagina no deberiamos observar como perdemos los datos y solo aparecen los nuevos recibimos, si no que podriamos ver una continuación de los datos anteriores.

# Bibliografía

- [1] How to measure electrical power <https://www.ksb.com/en-global/centrifugal-pump-lexicon/article/power-measurement-1116920>
- [2] Electrical waves <https://www.electronics-tutorials.ws/waveforms/waveforms.html>
- [3] What is power factor and why is it important? <https://www.fluke.com/en/learn/blog/power-quality/power-factor-formula>
- [4] An Explanation of RMS Power Measurement <https://www.electronicdesign.com/power-management/article/21120866/whats-the-difference-between-rms-and-peak-watts>
- [5] INA219 Current sensor <https://www.ti.com/lit/ds/symlink/ina219.pdf>
- [6] ZMPT101B Current sensor <https://datasheetspdf.com/mobile/1031464/ETC/ZMPT101B/1>
- [7] ACS712 Current sensor <https://cdn-reichelt.de/documents/datenblatt/A200/ACS712.pdf>
- [8] Arduino <https://www.arduino.cc/>
- [9] Raspberry Pi <https://www.raspberrypi.org/>
- [10] Esp8266 [https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf)
- [11] ESP32 <https://www.espressif.com/en/products/socs/esp32>
- [12] Micropython <https://micropython.org/>
- [13] Ejemplo en micropython <https://docs.micropython.org/en/latest/esp32/quickref.html#pins-and-gpio>

- [14] MQTT <https://mqtt.org/>
- [15] Zigbee <https://en.wikipedia.org/wiki/Zigbee>
- [16] Insight Into ESP32 Sleep Modes & Their Power Consumption <https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/>
- [17] Esptool <https://docs.espressif.com/projects/esptool/en/latest/esp32/>
- [18] Esptool Github <https://github.com/espressif/esptool>
- [19] Micropython downloads <https://micropython.org/download/#esp32>
- [20] Aropy Adafruit <https://learn.adafruit.com/micropython-basics-load-files-and-run-code/install-ampy>
- [21] SSD1306 Micropython driver <https://github.com/stlehmann/micropython-ssd1306/blob/master/ssd1306.py>
- [22] UMQTT.simple Micropython driver <https://github.com/micropython/micropython-lib/tree/master/micropython/umqtt.simple/umqtt>
- [23] Enlace a proyecto en github del sensor de corriente [https://github.com/b0nel/TFM\\_CurrentSensors/tree/main/esp32](https://github.com/b0nel/TFM_CurrentSensors/tree/main/esp32)
- [24] Fuente alimentación 5V <https://www.az-delivery.de/en/products/copy-of-220v-zu-5v-mini-netzteil>
- [25] Enlace a proyecto en github de la app web [https://github.com/b0nel/TFM\\_CurrentSensors/tree/main/app\\_web](https://github.com/b0nel/TFM_CurrentSensors/tree/main/app_web)

## Imágenes

- [26] Sensor de corriente ACS712 [https://hifisac.com/web/image/product.template/2507/image\\_1024?unique=6bb2274](https://hifisac.com/web/image/product.template/2507/image_1024?unique=6bb2274)
- [27] Forma de onda <https://www.electronics-tutorials.ws/wp-content/uploads/2018/05/waveforms-tim1.gif>

- [28] Contador de luz [https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.edistribucion.com%2Fes%2Fcontador-inteligente.html&psig=A0vVaw1P7NHoZnSfYabSYum\\_6eGT&ust=1669552774663000&source=images&cd=vfe&ved=0CBAQjRxqFwoTCNi\\_wvLuy\\_sCFQAAAAAdAAAAABAE](https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.edistribucion.com%2Fes%2Fcontador-inteligente.html&psig=A0vVaw1P7NHoZnSfYabSYum_6eGT&ust=1669552774663000&source=images&cd=vfe&ved=0CBAQjRxqFwoTCNi_wvLuy_sCFQAAAAAdAAAAABAE)
- [29] Ejemplo de factura de la luz [https://www.google.com/url?sa=i&url=https%3A%2F%2Fenchufesolar.com%2Fblog%2Ftu-factura-de-la-luz-explicada-paso-a-paso%2F&psig=A0vVaw26bJ-01gl0Srq4XB1XktT3&ust=1669552975152000&source=images&cd=vfe&ved=0CBAQjRxqFwoTCJjw99Hvy\\_sCFQAAAAAdAAAAABAJ](https://www.google.com/url?sa=i&url=https%3A%2F%2Fenchufesolar.com%2Fblog%2Ftu-factura-de-la-luz-explicada-paso-a-paso%2F&psig=A0vVaw26bJ-01gl0Srq4XB1XktT3&ust=1669552975152000&source=images&cd=vfe&ved=0CBAQjRxqFwoTCJjw99Hvy_sCFQAAAAAdAAAAABAJ)
- [30] Enchufe medidor de consumo electrico [https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.elconfidencial.com%2Ftecnologia%2F2021-08-19%2Fmedidores-consumo-electrico-para-ahorrar-energia\\_3237006%2F&psig=A0vVaw0NNrUmh-RooxDxCa3Y-p5&ust=1669553501475000&source=images&cd=vfe&ved=0CBAQjRxqFwoTCNCk1uHxy\\_sCFQAAAAAdAAAAABAD](https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.elconfidencial.com%2Ftecnologia%2F2021-08-19%2Fmedidores-consumo-electrico-para-ahorrar-energia_3237006%2F&psig=A0vVaw0NNrUmh-RooxDxCa3Y-p5&ust=1669553501475000&source=images&cd=vfe&ved=0CBAQjRxqFwoTCNCk1uHxy_sCFQAAAAAdAAAAABAD)
- [31] Medidor consumo cuadro electrico [https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.amazon.de%2F-%2Fen%2FDigital-Electricity-Top-hat-1-Phase-AC230V%2Fd%2FB08T62R7Y5&psig=A0vVaw0SbT0RjfTplhgZSh5ESXo7&ust=1669553760554000&source=images&cd=vfe&ved=0CBAQjRxqFwoTCNimOcjyy\\_sCFQAAAAAdAAAAABAE](https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.amazon.de%2F-%2Fen%2FDigital-Electricity-Top-hat-1-Phase-AC230V%2Fd%2FB08T62R7Y5&psig=A0vVaw0SbT0RjfTplhgZSh5ESXo7&ust=1669553760554000&source=images&cd=vfe&ved=0CBAQjRxqFwoTCNimOcjyy_sCFQAAAAAdAAAAABAE)
- [32] Monitor de energia inteligente [https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.amazon.com%2F-%2Fes%2Finteligente-electricidad-Medici%25C3%25B3n-Conserva-tranquilidad%2Fd%2FB086BNQ1X4&psig=A0vVaw1h014UGx5ZK099RTKsxWgl&ust=1669554058157000&source=images&cd=vfe&ved=0CBAQjRxqFwoTCLiPxdbzy\\_sCFQAAAAAdAAAAABAE](https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.amazon.com%2F-%2Fes%2Finteligente-electricidad-Medici%25C3%25B3n-Conserva-tranquilidad%2Fd%2FB086BNQ1X4&psig=A0vVaw1h014UGx5ZK099RTKsxWgl&ust=1669554058157000&source=images&cd=vfe&ved=0CBAQjRxqFwoTCLiPxdbzy_sCFQAAAAAdAAAAABAE)
- [33] Sensor INA219 [https://cdn-reichelt.de/bilder/web/xxl\\_ws/A300/DEB0\\_SENS\\_POWER\\_01.png](https://cdn-reichelt.de/bilder/web/xxl_ws/A300/DEB0_SENS_POWER_01.png)
- [34] Sensor ZMPT101B <https://mvelectronica.s3.us-east-2.amazonaws.com/productos/ZMPT/60997a27ba948.webp>
- [35] Arduino [https://components101.com/sites/default/files/component\\_pin/Arduino-Uno-Pin-Diagram.png](https://components101.com/sites/default/files/component_pin/Arduino-Uno-Pin-Diagram.png)

- [36] Raspberry Pi [https://learn.microsoft.com/en-us/windows/iot-core/media/pinmappingsrpি/rp2\\_pinout.png](https://learn.microsoft.com/en-us/windows/iot-core/media/pinmappingsrpি/rp2_pinout.png)
- [37] Esp8266 [https://components101.com/sites/default/files/component\\_pin/NodeMCU-ESP8266-Pinout.jpg](https://components101.com/sites/default/files/component_pin/NodeMCU-ESP8266-Pinout.jpg)
- [38] Esp32 <https://raw.githubusercontent.com/AchimPieters/esp32-homekit-camera/master/Images/ESP32-38%20PIN-DEVBOARD.png>
- [39] Ejeplo MQTT <https://mqtt.org/assets/img/mqtt-publish-subscribe.png>
- [40] Ejemplo Zigbee [https://csa-iot.org/wp-content/uploads/2021/12/Zigbee\\_SmartHome-996x1024.png](https://csa-iot.org/wp-content/uploads/2021/12/Zigbee_SmartHome-996x1024.png)
- [41] Divisor de tension [https://upload.wikimedia.org/wikipedia/commons/thumb/3/31/Impedance\\_voltage\\_divider.svg/1200px-Impedance\\_voltage\\_divider.svg.png](https://upload.wikimedia.org/wikipedia/commons/thumb/3/31/Impedance_voltage_divider.svg/1200px-Impedance_voltage_divider.svg.png)

