

Universidad de Alcalá  
Escuela Politécnica Superior

Grado en Ingeniería Informática



Sistema IoT para la monitorización y control de  
consumos eléctricos

ESCUELA POLITECNICA  
SUPERIOR

**Autor:** David Ruiz González

**Tutor:** Óscar García Población

2024



UNIVERSIDAD DE ALCALÁ  
Escuela Politécnica Superior

**Grado en Ingeniería Informática**

Trabajo Fin de Grado

**Sistema IoT para la monitorización y control de consumos eléctricos**

**Autor:** David Ruiz González

**Tutor/es:** Óscar García Población

**TRIBUNAL:**

**Presidente:** Óscar Rodríguez Polo

**Vocal 1º:** Juan Ignacio García Tejedor

**Vocal 2º:** Óscar García Población

**FECHA:** << Fecha de depósito >>



## **Agradecimientos**

En primer lugar, a mi tutor Óscar, por prestarse para ser mi tutor para este proyecto, por toda la ayuda prestada y el tiempo dedicado, y por toda la paciencia y el compromiso que me ha brindado.

En especial me gustaría agradecer a mi familia, por estar en los buenos y los malos momentos, y acompañarme en cada paso del camino. A mis padres, M.<sup>a</sup> Carmen y Fran, por apoyarme y estar a mi lado siempre. Y a mi hermana María, por ser la mejor compañera que la vida podía darme. Les debo todo, mi educación, mis valores y todo el amor recibido. Sin ellos, no sería la persona que soy hoy.

No puedo olvidarme de las amistades que me ha dejado esta carrera. Los que conocí en las aulas de la Escuela Politécnica, y que ahora forman parte de mi día a día y no podría imaginar una vida sin ellos: Edu, Jaime, Adri, Isco, Mario, Alex, Heras y Miguel. Y los que conocí en Varsovia, durante mi estancia de Erasmus: Raúl, Blanca, Pol, Alba, Sergio, Alba e Inés. Pese a la distancia que nos separa, los momentos vividos nos unen de por vida, y siempre es una alegría cuando nos reencontramos.



# Resumen

Este Trabajo de Fin de Grado (TFG) presenta el desarrollo de un sistema Internet-of-Things (IoT) de monitorización y control de cargas eléctricas utilizando un microcontrolador ESP32. El sistema recoge datos de un sensor PZEM004T, que mide parámetros eléctricos. Estos datos se envían, tanto a una base de datos Influxdb mediante MQTT para su almacenamiento, como a una aplicación web, diseñada con React+Vite, que permite ver las mediciones en tiempo real, así como el control remoto de las cargas conectadas mediante relés.

**Palabras clave:** IoT, ESP32, MQTT, Websockets, Monitorización eléctrica, Control de cargas.



# Abstract

This Bachelor's Degree Final Project presents the development of an Internet-of-Things (IoT) system for monitoring and controlling electrical loads using an ESP32 microcontroller. The system collects data from a PZEM004T sensor, which measures electrical parameters. These data are sent both to an Influxdb database via MQTT for storage and to a web application, designed with React+Vite, that allows real-time viewing of the measurements as well as remote control of the connected loads via relays.

**Keywords:** IoT, ESP32, MQTT, Websockets, Electrical monitoring, Load control.



# Índice General

<i>Agradecimientos</i>	5
<i>Resumen</i>	7
<i>Abstract</i>	9
<i>Índice de Figuras</i>	13
<i>Índice de Tablas</i>	15
<b>Capítulo 1: Introducción</b>	17
1.1 Motivación y objetivos	17
1.2 Metodología	18
1.3 Estructura del documento	18
<b>Capítulo 2: Descripción de las tecnologías utilizadas</b>	21
2.1 Tecnología Web	21
2.1.2 React	21
2.1.3 Node.js	22
2.1.4 Vite	23
2.2 Electrónica	23
2.2.1 ESP32	23
2.2.2 PZEM-004T	25
2.2.3 XL9535-K4V5	26
2.3 Base de datos	27
2.3.1 Influxdb	27
2.3.2 Telegraf	28
<b>Capítulo 3: Diseño del sistema</b>	29
3.1 Descripción general de la arquitectura	29
3.2 Descripción detallada de la arquitectura	32
3.2.1 Capa de percepción	33
3.2.1.1 ESP32	33
3.2.2 Capa de transporte	46
3.2.2.1 MQTT	46
3.2.2.2 Websockets	47
3.2.3 Capa de procesamiento	48
3.2.3.1 Influxdb	49
3.2.3.2 Telegraf	51
3.2.4 Capa de aplicación	54
3.2.4.1 Aplicación web	54
<b>Capítulo 4: Pruebas y resultados</b>	65
4.1 Conexión Wi-Fi	65
4.2 Conexión SNTP y establecimiento de fecha y hora	65
4.3 Recogida y recepción de mediciones	66

<b>4.4 Comunicación mediante MQTT</b>	<b>66</b>
4.4.1 Conexión del ESP32 con el bróker	66
4.4.2 Conexión de Telegraf con el bróker	67
4.4.3 Envío de mensaje MQTT desde el ESP32	67
4.4.4 Recepción de mensaje MQTT en Telegraf	68
<b>4.5 Comunicación mediante Websockets</b>	<b>69</b>
4.5.1 Conexión del ESP32 con el servidor	69
4.5.2 Conexión de la aplicación web con el servidor	69
4.5.3 Envío de mensaje desde el ESP32	70
4.5.4 Recepción de mensaje en la aplicación web	70
4.5.5 Envío de mensaje desde la aplicación web	72
4.5.6 Recepción de mensaje en el ESP32	72
<b>4.6 Modificación del estado de un relé</b>	<b>72</b>
<b>Capítulo 5: Conclusiones y trabajo futuro</b>	<b>75</b>
5.1 Conclusiones	75
5.2 Trabajo a futuro	76
<b>Capítulo 6: Entregables</b>	<b>77</b>
<b>Bibliografía y referencias</b>	<b>79</b>
<b>Acrónimos y abreviaturas</b>	<b>83</b>
<b>Anexo I – Presupuesto</b>	<b>85</b>
<b>Anexo II – Manuales de usuario e Instalación</b>	<b>87</b>
Manual de instalación: Entorno de desarrollo ESP32	87
Manual de instalación: Node.js y npm	94
Manual de usuario: Aplicación web	97

# Índice de Figuras

Figura 1: ESP32	24
Figura 2: Kit de medición PZEM-004T	25
Figura 3: Módulo de relés XL9535-K4V5	26
Figura 4: Arquitectura de 4 capas del sistema.	30
Figura 5: Flujo de información en el sistema	31
Figura 6: Estructura simplificada del sistema.	31
Figura 7: Estructura detallada del sistema.	32
Figura 8: Mapa de pines ESP32	34
Figura 9: Flujo principal del programa	41
Figura 10: Flujo de la tarea principal del sistema	42
Figura 11: Conexiones entre ESP32 y PZEM-004T	43
Figura 12: Diagrama de secuencia - Interacción entre ESP32 y PZEM-004T	43
Figura 13: Conexiones entre ESP32 y XL9535-K4V5	45
Figura 14: Diagrama de secuencia - Interacción entre ESP32 y XL9535-K4V5	45
Figura 15: Pantalla Influxdb - Inicio de sesión con credenciales	50
Figura 16: Pantalla Influxdb - Visualización de datos	50
Figura 17: Pantalla Influxdb - Configuración Telegraf – Bucket	52
Figura 18: Pantalla Influxdb - Configuración Telegraf – Selección del origen de los datos	53
Figura 19: Navegador - Búsqueda VS Code	87
Figura 20: Página de VS Code - Instaladores	88
Figura 21: Página VS Code - Descarga + Tutorial guiado	88
Figura 22: Instalación VS Code - Términos de licencia de Microsoft	89
Figura 23: Instalación VS Code - Opciones adicionales	89
Figura 24: Instalación VS Code - Resumen de la instalación	90
Figura 25: Instalación VS Code - Final de la instalación	90
Figura 26: Configuración VS Code – Extensiones	91
Figura 27: Configuración VS Code – ESP-IDF	92
Figura 28: Configuración VS Code – ESP-IDF instalación	92
Figura 29: VS Code - ESP-IDF instalado correctamente	93
Figura 30: Instalación de Node.js - Descarga	94
Figura 31: Instalación de Node.js - Términos	94
Figura 32: Instalación de Node.js - Dirección de instalación	95
Figura 33: Instalación de Node.js - Opciones de instalación	95
Figura 34: Instalación de Node.js - Herramientas necesarias	96
Figura 35: Instalación de Node.js - Instalación exitosa	96
Figura 36: Consola de comandos de Node.js	97
Figura 37: Consola Node.js - Cambio de directorio	98
Figura 38: Consola Node.js - Aplicación web en funcionamiento	98
Figura 39: Explorador de archivos - Abrir carpeta con VS Code	99
Figura 40: VS Code - Abrir nuevo terminal	99
Figura 41: Aplicación web - Pantalla principal	100
Figura 42: Aplicación web - Pantalla principal con el menú oculto	101
Figura 43: Aplicación web - Pantalla Datos	102
Figura 44: Aplicación web - Gráfica Potencia	103
Figura 45: Aplicación web - Gráfica Voltaje	103
Figura 46: Aplicación web - Gráfica Intensidad	104
Figura 47: Aplicación web - Gráfica Energía	104
Figura 48: Aplicación web - Gráfica Frecuencia	105
Figura 49: Aplicación web - Gráfica Factor de Potencia	105
Figura 50: Aplicación web - Control sobre las cargas	106
Figura 51: Aplicación web - Sobre el proyecto	106



# Índice de Tablas

<i>Tabla 1: Especificaciones ESP32 Dev Kit C V4 NodeMCU .....</i>	25
<i>Tabla 2: Parámetros eléctricos y rangos de medición PZEM-004T .....</i>	26
<i>Tabla 3: Pruebas Wi-Fi - Resultados conexión.....</i>	65
<i>Tabla 4: Pruebas SNTP - Resultados conexión .....</i>	65
<i>Tabla 5: Pruebas Sensor - Resultados recepción de mediciones.....</i>	66
<i>Tabla 6: Pruebas MQTT - Resultados conexión ESP32 con bróker.....</i>	66
<i>Tabla 7: Pruebas MQTT - Resultados conexión Telegraf con bróker .....</i>	67
<i>Tabla 8: Pruebas MQTT - Resultados envío de mensaje desde ESP32 .....</i>	68
<i>Tabla 9: Pruebas MQTT - Resultados recepción de mensaje en base de datos .....</i>	68
<i>Tabla 10: Pruebas Websockets - Resultados conexión entre ESP32 y servidor .....</i>	69
<i>Tabla 11: Pruebas Websockets - Resultados conexión entre aplicación web y servidor .....</i>	70
<i>Tabla 12: Pruebas Websockets - Resultados envío de mensajes desde ESP32 .....</i>	70
<i>Tabla 13: Pruebas Websockets - Resultados recepción de mensajes en aplicación web .....</i>	71
<i>Tabla 14: Pruebas Websockets - Resultados envío de mensaje desde aplicación web .....</i>	72
<i>Tabla 15: Pruebas Websockets - Resultados recepción de mensaje en ESP32.....</i>	72
<i>Tabla 16: Pruebas Relé - Resultados modificación de estado .....</i>	74
<i>Tabla 17: Costes de la parte no tangible del sistema .....</i>	85
<i>Tabla 18: Costes de la parte tangible del sistema .....</i>	85



# Capítulo 1: Introducción

En esta sección se describirán los puntos clave del proyecto. Comenzando por detallar los objetivos que se busca alcanzar, seguido de la metodología empleada, y finalmente la estructura que siguen los capítulos y secciones del proyecto.

## 1.1 Motivación y objetivos

Mi principal motivación para aceptar esta propuesta de TFG fue que combina perfectamente dos temas muy presentes en la actualidad en el mundo de las Tecnologías de la Información y las Comunicaciones (TIC), estos son el desarrollo web y la domótica.

Por un lado, la **domótica** cada día tiene más presencia en los hogares. Tras cursar la asignatura Computación Ubiña, realizando las correspondientes prácticas, se despertó mi curiosidad por un sector de la informática que puede aplicarse en un hogar promedio, y que puede traer mucha comodidad y facilidad durante la realización de tareas cotidianas. Desde programar horarios de funcionamiento y apagado de electrodomésticos, al control remoto de calefacción o iluminación, la domótica presenta un sinfín de posibilidades en el entorno del hogar. Para aprovechar estas oportunidades basta con tener un ligero conocimiento de electrónica e informática, y realizar algo de investigación acerca del tema.

Por otro lado, no son necesarios profundos conocimientos de informática para darse cuenta de la constante presencia de los **entornos web** en el día a día. Dejando a un lado las páginas web que proporcionan cualquier fragmento de información que podamos requerir, el comercio y consumo en línea se encuentra en auge. Antes de 2019 podía verse una tendencia ascendente del número de negocios que comenzaba a ofrecer de forma online sus servicios, de forma parcial como los periódicos (que mantenían el formato en papel), o completamente en remoto como marcas de ropa que optan por el envío de sus productos, como es Nude Project. Durante la pandemia de Covid-19 en 2020, con el confinamiento y la situación sanitaria de riesgo posterior, aumentó el número de compras por internet por parte de la población española colocándola a la cabeza en el consumo online en Europa. Es, por tanto, de gran importancia para cualquier negocio actual que quiera tener un desempeño competitivo en el mercado, tener la capacidad de llegar a su público objetivo mediante las vías en auge.

En vista de que no todo el mundo comparte las mismas capacidades para tratar con la tecnología, el primer y más importante objetivo del proyecto es **diseñar un sistema fiable que se pueda implementar en cualquier hogar**, tratando de minimizar el coste total, y procurando que la dificultad del sistema no se traduzca en dificultad de instalación y configuración. El sistema podría alcanzar los hogares mediante dos vías:

- De forma particular, para la gestión privada por parte de los residentes, como plan de domotización del hogar;
- o como un servicio que forma parte de la instalación realizada por la compañía suministradora del servicio eléctrico.

Aunque no con tanto peso, también formarían parte de la lista de objetivos:

- La **obtención de conocimientos en desarrollo web y en electrónica**: Siendo dos campos con una creciente presencia en el mundo de la tecnología, durante el transcurso del grado no se ha puesto el foco sobre ellos más allá de tareas independientes de ciertas asignaturas.
- El **crecimiento personal**: la posibilidad de demostrarse a uno mismo que es capaz de realizar un proyecto con características visiblemente más complejas y exigentes que los realizados a lo largo de la titulación. Todo esto siendo posible mediante una organización apropiada del tiempo, y la voluntad para realizar el esfuerzo necesario que ello conlleva.

## 1.2 Metodología

La metodología que se ha empleado durante el desarrollo del proyecto para lograr el objetivo mencionado anteriormente consta de dos partes que se van alternando de forma constante a lo largo del mismo. La primera de ellas es **teórica**, trata de la **adquisición de conocimientos** necesarios. Tras seleccionar y detallar los componentes del proyecto, tuve que realizar trabajo de **investigación** para entender su funcionamiento, cómo prepararlos para que realicen las tareas requeridas en el sistema y cómo conectarlos entre sí, antes de poder comenzar a trabajar con cada uno de ellos. Principalmente, me serví de páginas web y foros de internet, así como de cursos y tutoriales (tanto de pago, como gratuitos) para obtener la información que he ido necesitando durante el desarrollo del sistema. La otra parte es la **práctica**, que consiste en la **configuración** de los componentes y la **programación** de estos para que funcionen como deberían en el esquema general del sistema. Como se menciona anteriormente, las dos partes se van intercalando, para cada nuevo componente que se va agregando al sistema, primero se investiga acerca de sus características, luego se busca la forma de prepararlo que más se ajuste a las necesidades y limitaciones del sistema, para finalmente pasar a la realización de la configuración y puesta a punto.

Este proyecto involucra **componentes SW**, como la aplicación web, y **HW**, como el propio chip ESP32. Ambos tipos han requerido diferente cantidad de esfuerzo y tiempo durante el desarrollo. Debido a que a lo largo de la carrera hemos cursado varias asignaturas donde se han tratado diferentes entornos y lenguajes de programación, el software involucrado no ha resultado problemático ni ha demandado demasiado tiempo a la hora de la investigación. En cambio, no son tantas las situaciones donde hemos empleado componentes electrónicos en las prácticas de las asignaturas, por ello he necesitado emplear un mayor tiempo y esfuerzo en comprender conceptos básicos, que por las razones mencionadas no tenía interiorizados.

## 1.3 Estructura del documento

Esta sección tiene como propósito mostrar la estructura básica de este documento, resumiendo brevemente los aspectos más importantes de cada sección, permitiendo tener una pequeña visión de lo que se va a tratar en ella. A continuación, se detallan los capítulos que conforman esta memoria del proyecto, explicando con brevedad su contenido:

### **Capítulo 1: Introducción.**

Presenta la motivación y los objetivos del proyecto, justificando la relevancia del trabajo realizado. Describe la metodología utilizada y proporciona una visión general de la estructura del documento, facilitando al lector una comprensión inicial del contenido.

## **Capítulo 2: Descripción de las tecnologías utilizadas.**

Detalla las tecnologías empleadas en el desarrollo del sistema, ofreciendo al lector una debida presentación sobre cada una de ellas. Las nociones teóricas presentadas facilitan el seguimiento del resto del documento, y la compresión del funcionamiento del sistema.

## **Capítulo 3: Diseño del sistema.**

Descripción de la arquitectura del sistema, comenzando desde una vista más general, para posteriormente adentrarse en detalle en cada uno de los componentes, su función en el sistema y la forma en la que fueron configurados. Los medios y protocolos utilizados para la comunicación entre componentes son también descritos en esta sección.

## **Capítulo 4: Pruebas y resultados.**

Descripción de las pruebas realizadas para comprobar si el funcionamiento del sistema es el esperado. Se detallan las condiciones iniciales del sistema, y se compara el resultado esperado con el resultado real producido por el sistema.

## **Capítulo 5: Conclusiones y trabajo futuro.**

Resumen de los principales hallazgos y logros del proyecto, destacando su impacto y las lecciones aprendidas. Descripción de posibles mejoras y direcciones para futuras versiones del sistema, ofreciendo una perspectiva de cómo el sistema podría evolucionar y expandirse.

## **Capítulo 6: Entregables.**

Lista de los elementos creados o generados durante la implementación del sistema, incluyendo una maqueta de consumos y un prototipo formado por los elementos hardware.

## **Bibliografía y referencias.**

Lista completa de las fuentes consultadas y citadas a lo largo del proyecto, ofreciendo un respaldo teórico y técnico. Incluye páginas web, documentación técnica, material audiovisual y otros recursos relevantes.

## **Anexos I – Presupuesto**

Presenta un detalle exhaustivo de los costos asociados con el proyecto, incluyendo materiales, herramientas, software y otros recursos necesarios para su desarrollo e implementación. Proporciona una visión clara de los recursos financieros invertidos.

## **Anexo II – Manuales de usuario e Instalación**

Instrucciones detalladas para la instalación y uso del sistema. Incluye guías para configurar el entorno de desarrollo del ESP32, instalar y configurar el bróker MQTT, configurar el servidor WebSocket, y configurar la base de datos Influxdb estableciendo Telegraf como agente para la obtención de datos. Ofrece también un manual de usuario para la aplicación web, facilitando su uso y administración.



# Capítulo 2: Descripción de las tecnologías utilizadas

Este capítulo reflejará conceptos fundamentales acerca de las tecnologías involucradas en el proyecto. El objetivo, por tanto, será proporcionar un marco teórico lo suficientemente consistente, que permita al lector conocer las nociones básicas necesarias acerca de las tecnologías escogidas para dar vida al proyecto.

## 2.1 Tecnología Web

Ya que uno de los principales motivos para realizar esta propuesta fue el poder realizar el diseño de una aplicación web, será esta tecnología con la que comience este marco teórico.

Las tecnologías web son un medio para que cualquier usuario que posea un dispositivo con acceso a internet pueda acceder a los recursos y conocimientos disponibles utilizando un navegador. Su escalabilidad, portabilidad y sencillez de uso han llevado a esta tecnología a ser una de las más abundantes y con las que más trabajan los desarrolladores en la actualidad [7].

Actualmente, son varias las opciones que tiene un desarrollador para diseñar e implementar una aplicación web. Obviando la cantidad de posibles entornos en los que se decida desarrollar la aplicación web, se dispone de infinidad de posibles combinaciones en las que se puede escoger la biblioteca, el *framework*<sup>1</sup>, y cualquier componente extra que pueda aportar rendimiento o reducir la complejidad del trabajo.

Para el desarrollo de la aplicación web se han utilizado React, Node.js y Vite.

### 2.1.2 React

React es una **biblioteca de código abierto**<sup>2</sup> de JavaScript enfocada en la construcción de interfaces de usuario (UI). Fue creada por un grupo de desarrolladores de Facebook con el objetivo principal de reducir la dificultad al trabajar con interfaces de usuario de gran complejidad, incrementando el rendimiento.

La característica más destacada de la biblioteca es que basa el desarrollo en un concepto conocido como «Virtual DOM» (Modelo de Objetos del Documento). El DOM tradicional permite al navegador interactuar con los elementos presentes en la web, mediante una **representación con estructura jerárquica** del documento HTML o XML, considerando el documento como un árbol donde cada elemento es un nodo. [1]

JavaScript normalmente manipula los componentes del DOM cada vez que hay cambios, actualizando y renderizando el conjunto completo, mientras que React aprovecha este «Virtual DOM» para realizar **únicamente las operaciones de actualización precisas**. Esto lo hace mediante una representación virtual de los elementos, que se compara con los componentes reales a la hora de realizar una actualización, permitiendo operar únicamente sobre los elementos que

1. En el ámbito de la programación, “framework” es un conjunto de herramientas y librerías que se utilizan para desarrollar aplicaciones más fácilmente y de manera más eficiente. [1]

2. Que un software sea referido como de código abierto o “open source” quiere decir que es un tipo de software cuyo código fuente está disponible para ser utilizado, modificado y distribuido por cualquier persona. [2]

han sufrido alguna modificación, y dejando los demás intactos. Esto supone una gran mejora de rendimiento y mayor fluidez a la hora de interactuar con la aplicación web, ya que **reduce el número de actualizaciones** y la cantidad de recursos necesarios. [1]

Además, que React sea una biblioteca de código abierto es una característica de gran valor para un desarrollador. El ser una de las bibliotecas más populares para el diseño de aplicaciones web en lenguaje JavaScript, hace que haya una **comunidad** compuesta por miles de desarrolladores de todo el mundo. Esto siempre es algo positivo, ya que habitualmente se dispone de gran cantidad de **material expuesto públicamente** que ayuda tanto a principiantes a ampliar su conocimiento, como a expertos a refinar aún más su habilidad.

### 2.1.3 Node.js

Creado por los desarrolladores originales de JavaScript en 2009, Node.js es un **entorno de ejecución de JavaScript** del lado del servidor de código abierto que permite ejecutar código JavaScript, facilitando el desarrollo de aplicaciones web escalables y de alto rendimiento. Es decir, es una plataforma que aporta todo lo necesario para la ejecución de un programa en dicho lenguaje fuera del navegador, permitiendo crear aplicaciones web rápidas y escalables. [11]

A diferencia de los enfoques tradicionales de servidores web que generan un nuevo hilo para cada conexión, consumiendo RAM y saturando rápidamente los recursos del sistema, Node.js funciona con **un único hilo**. Utiliza un modelo de **E/S no bloqueante**, lo que le permite gestionar decenas de miles de conexiones simultáneamente mediante un bucle de eventos. Este modelo se basa en un sistema de eventos, en el que el servidor maneja un único hilo que procesa los eventos uno tras otro. El funcionamiento podría resumirse de la siguiente forma:

1. Cuando se recibe una nueva solicitud, se crea un evento correspondiente y el servidor inicia su procesamiento.
2. Al encontrarse con una operación de E/S que bloquearía el hilo, no espera a que termine. En cambio, delega la continuación a una función de *callback*<sup>3</sup> y sigue con otros eventos.
3. Una vez que la operación de E/S se completa, el servidor ejecuta la función de *callback* tan pronto como puede.

Por lo tanto, el servidor nunca necesita generar múltiples hilos ni alternar entre ellos, **minimizando la sobrecarga** que pudiera darse en el sistema. [3]

Es por ello que las características que ofrece Node.js son atractivas a ojos de desarrolladores de cualquier nivel: comenzando por su **sencillez**, que promueve su utilización por parte de programadores menos experimentados; la **escalabilidad y velocidad** debido a que su arquitectura emplea menos hilos que otras alternativas, necesitando menos recursos; la **variedad de paquetes** de código abierto que facilitan la actividad de desarrollo; la posibilidad de crear aplicaciones en distintas plataformas; ofrece gestión tanto para *frontend*<sup>4</sup> como *backend*<sup>5</sup>.

Además de todo ello, Node.js cuenta con **Node Package Manager** (npm), un gestor de paquetes que facilita al usuario la instalación, actualización y manejo de paquetes y dependencias en proyectos de Node.js. Permite a los desarrolladores la gestión y el mantenimiento de paquetes cuando el almacenamiento local resulta insuficiente, o el número de colaboradores sobre el código

3. Un “callback” es una función que se pasa como argumento de otra función, y que se invoca en relación con unas condiciones determinadas (usualmente el final de la ejecución de la primera función). Se utilizan para manejar operaciones asíncronas y eventos. [13]

4. El término “frontend” se refiere a la parte de una aplicación o sitio web con la que los usuarios interactúan directamente. [12]

5. El término “backend” se refiere a la parte de una aplicación o sitio web que se ejecuta en el servidor y es gestionada la lógica del negocio y de la aplicación, así como del procesamiento de solicitudes y respuestas entre el cliente (frontend) y el servidor. [12]

aumenta. Actualmente, cuenta con más de un millón de paquetes disponibles, que permite a los desarrolladores **compartir y reutilizar código**. npm también permite la **definición de scripts**<sup>6</sup> personalizados para automatizar tareas comunes, aumentando la eficiencia y productividad durante el proceso de desarrollo. [14][15]

#### 2.1.4 Vite

Vite es una **herramienta de construcción y desarrollo** para proyecto web. Fue creada por Evan You, creador de la biblioteca para JavaScript Vue.js. El objetivo de su creador es proporcionar una herramienta para mejorar la experiencia del desarrollador a la hora de realizar proyectos web modernos, simplificando y acelerando el proceso de compilación.

Uno de los principales problemas que presenta JavaScript es la modularización, es decir, diseñar el proyecto creando fragmentos separados que cubren diferentes funcionalidades y uniéndolos se obtiene el resultado final. Hasta la llegada de los módulos ES, se dificultaba la creación de aplicaciones web muy complejas o con un número elevado de líneas de código fuente. Vite aprovecha estos **módulos ES** junto con otras herramientas para JavaScript para **maximizar el rendimiento y minimizar la dificultad** a la hora de enfrentar proyectos grandes y complejos en este lenguaje.

Una de las mejores características que representa el uso de Vite es la **mejora en el tiempo de inicio** del servidor, mediante la división en módulos de una aplicación en dependencias y código fuente, acelera el proceso de compilación de arranque cuando el servidor de desarrollo parte de cero. [16]

En cuanto a rendimiento y velocidad se refiere, Vite también es **responsable de agilizar** las usualmente lentas actualizaciones que se dan al modificar un archivo. Los refrescos se realizan sobre los componentes modificados, únicamente la parte que ha sufrido cambios, sin afectar al resto de la página. Este proceso se conoce como *Hot Module Replacement* (HMR) ya que el módulo afectado se “reemplaza” a sí mismo “en caliente”. Este método no es infalible, el rendimiento se reduce considerablemente según crece la aplicación, pero la mejora frente a métodos basados en empaquetado tradicionales es más que notable. [17]

## 2.2 Electrónica

El otro campo de la informática con gran importancia en el proyecto es la domótica, que no podría concebirse sin el uso de elementos electrónicos como sensores, actuadores o, los que cargan con la tarea de controlar todo el sistema, microcontroladores. A continuación, se comentarán las principales características de los elementos electrónicos involucrados en el sistema.

#### 2.2.1 ESP32

La denominación ESP32 no se refiere a un dispositivo concreto, si no a una **familia de placas de desarrollo con microcontrolador** creada por Espressif Systems como sucesor del ESP8266, otro SoC. Dentro de esta familia encontramos diversos modelos cada uno con diferentes

6. Un “script” es código que se ejecuta en el navegador web para interactuar con una aplicación web y realizar determinada función.  
[28]

especificaciones, que los hacen más apropiados para un tipo u otro de proyecto [18]. Sin embargo, todos tienen en común ciertas características, de entre las que destacan:

- Microprocesador de 32-bit de doble núcleo (o un solo núcleo), que opera entre 160 y 240 MHz.
- Memoria RAM con 520KiB.
- Conectividad Wi-Fi.
- Conectividad Bluetooth: v4.2 y BLE.
- Tres interfaces UART.
- Dos interfaces I2C.
- 16 canales de salida.
- 2 convertidores digital a analógico.
- Al menos 10 GPIOs con detección capacitativa. [19]

Estas y más características convierten a cualquier miembro de la familia ESP32 en una interesante solución aplicable en proyectos de todo ámbito y escala. Junto a la amplia variedad de periféricos compatibles con estas placas, el límite está en la imaginación del usuario.

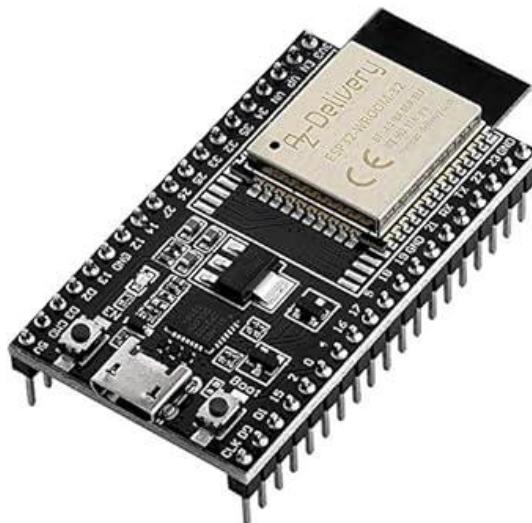


Figura 1: ESP32

La configuración y programación para estos dispositivos suele realizarse de dos formas. La primera de ellas es mediante el entorno de desarrollo ArduinoIDE. Este entorno no solo es compatible con las placas y tarjetas de la familiar Arduino, también incluye soporte para placas

ESP y ofrece infinidad de bibliotecas y opciones de personalización al usuario. La segunda forma, y recomendada por la propia compañía Espressif Systems, es mediante el uso de **ESP-IDF**, SDK oficial del fabricante. Para optar por este método, será necesario disponer de un entorno de desarrollo compatible con el SDK, siendo recomendados por la compañía ESP-IDE y **Visual Studio Code** con la extensión de ESP-IDF.

Para la realización de este proyecto, el modelo escogido es el ESP32 Dev Kit C V4 NodeMCU:

<b>Velocidad de CPU</b>	2,4 GHz
<b>Número de procesadores</b>	2
<b>Tecnología de conectividad</b>	Bluetooth, Wi-Fi, WLAN, I2C, UART, GPIO
<b>Estándar de comunicación inalámbrica</b>	Bluetooth, 802.11b

Tabla 1: Especificaciones ESP32 Dev Kit C V4 NodeMCU

Su **coste moderado**, las reseñas que lo presentaban como **fiable** y con **buen rendimiento**, junto con sus opciones de conectividad que cubren las necesidades del proyecto lo convertían en un candidato ideal, y que sin duda ha rendido a la altura de las expectativas.

## 2.2.2 PZEM-004T

El componente escogido para realizar las mediciones es el **módulo multi-función PZEM-004T**. Dispone de una interfaz UART TTL optoacoplada<sup>7</sup> que le permite la comunicación con un microprocesador, un PC o un módulo Wi-Fi. Permite **obtener medidas eléctricas** mediante la conexión a una línea 110/220 V, como la que suele haber en los hogares o electrodomésticos comunes.



Figura 2: Kit de medición PZEM-004T

Como se puede ver en la imagen, el kit PZEM-004T está formado por la placa **PZEM-004T** y un **transformador 100A**.

7. Significa que está diseñada para proporcionar aislamiento eléctrico entre los componentes conectados.

Las medidas que recoge, junto con los rangos de trabajo en los que opera son los siguientes:

<b>Potencia</b>	0 a 23 kW
<b>Energía</b>	0 y 9999.99 kWh
<b>Voltaje</b>	80 a 260 VAC
<b>Corriente</b>	0 a 100 A
<b>Frecuencia</b>	45 a 65 Hz
<b>Factor de potencia</b>	0.00 a 1.00

Tabla 2: Parámetros eléctricos y rangos de medición PZEM-004T

### 2.2.3 XL9535-K4V5

El componente encargado del control de las cargas es el **módulo de relé de expansión XL9535-K4V5**. Compuesta por la placa de expansión XL9535, que se encarga de controlar el accionamiento de los cuatro relés SRD-05VDC-SL-C. Este dispositivo permite el **control del flujo de corriente** en el circuito eléctrico.

Los relés, que funcionan con bobinas de 5 voltios, y tienen dos puntos de contactos, uno abierto denotado por NO (*Normally Open*) y otro cerrado denotado por NC (*Normally Closed*). Mediante esto y el uso de un punto de contacto común que recibe el flujo eléctrico y alterando los puntos conectados, se puede obtener un efecto similar al de un interruptor. [21]

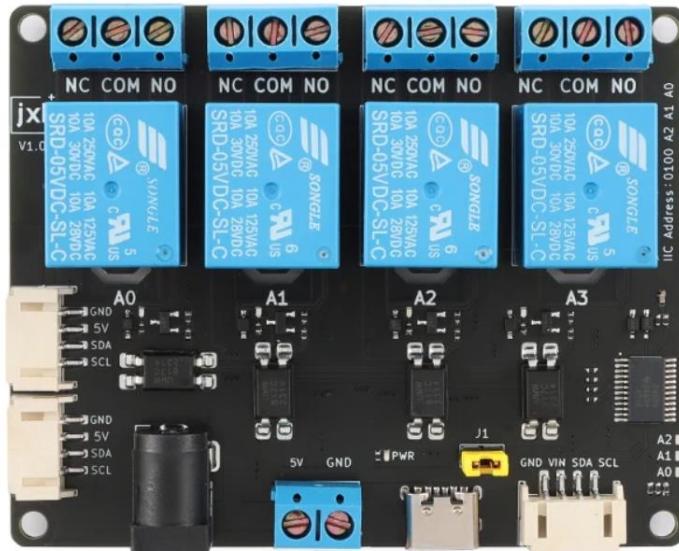


Figura 3: Módulo de relés XL9535-K4V5

Este dispositivo ofrece diferentes interfaces de conexión:

- 3 interfaces I2C: cada una con su línea de tierra (GND), 5V, SCL y SDA. Comparten el mismo bus.
- Interfaz USB-C

- DC Jack

Todas las entradas de alimentación están conectadas y soportan de 2,3 a 5,5 V.

## 2.3 Base de datos

Para todo sistema que recoja datos es esencial disponer de algún tipo de almacenamiento donde guardar las mediciones según se van realizando, ya sea para mantener un histórico para poder visualizar los datos del sistema desde que comenzó su funcionamiento, como para procesar y utilizar esos datos en otras partes del sistema. La tecnología para almacenamiento de datos escogida para este sistema está compuesta por los elementos que se describen a continuación.

### 2.3.1 Influxdb

Creada por la empresa InfluxData, Influxdb es un sistema de código abierto para **la gestión de bases de datos de series temporales**<sup>8</sup>, lo que significa que la entrada de datos está asociada a una marca temporal. Escrita en lenguaje Go (o Golang) permite la **supervisión y el procesamiento de datos**, siendo una opción atractiva y de garantías a la hora de trabajar con sensores y domotización del hogar.

Una de las características más determinantes es que este sistema se diseñó íntegramente para trabajar con bases de datos de series temporales. Permite la obtención, almacenamiento, gestión y visualización de los datos.

Cuenta con un lenguaje de consulta propio inspirado en SQL, llamado InfluxQL, diseñado como alternativa más familiar al tradicional lenguaje de tratamiento de bases de datos que el otro lenguaje de consulta disponible para Influxdb, llamado Flux. [22]

Una funcionalidad muy interesante que ofrece Influxdb es la posibilidad de gestión de las bases de datos mediante diferentes vías, permitiendo a usuarios más experimentados trabajar con consola instalando su CLI, y a los más novatos utilizar una UI. El sistema incluye inicio de sesión con credenciales, visualización de datos con diferentes gráficos y configuración de las bases de datos y los agentes involucrados.

Las principales ventajas que ofrece Influxdb a los desarrolladores son:

- **Velocidad de escritura alta**, permitiendo manejar grandes volúmenes de datos.
- **Escalabilidad** de las bases de datos, siendo capaz de gestionar millones de puntos de datos por segundo.
- **Flexibilidad** de almacenamiento: dado que no requiere de la estructura habitual regida por columnas, permitiendo la inserción de nuevos tipos de datos sin alterar bases de datos ya existentes.
- **Integración con herramientas** de análisis, procesado y visualización, como Grafana, para proporcionar dashboards interactivos y análisis de datos en tiempo real.

<sup>8</sup>. Sucesión de datos medidos en momentos determinados, que son ordenados de forma cronológica, separados en intervalos iguales o desiguales en el tiempo. [24]

- **Integración con agentes de recolección de datos**, como **Telegraf**, ofreciendo un amplio abanico de métodos y protocolos de inserción de datos, todos altamente configurables. [23]

### ***2.3.2 Telegraf***

Para la recopilación de datos, tenemos **Telegraf**, un **agente de servidor de código abierto** que trabaja como complemento de Influxdb. Dispone de **más de 300 plugins** escritos por miembros de la comunidad, y que permiten la recolección de métricas de **más de 150 orígenes distintos**. Permite la recepción de datos en formato CSV, JSON, o procedentes de Graphite.

La recepción y recolección de datos es altamente configurable y se puede adaptar a múltiples tipos de aplicación de diferentes ámbitos. [25]

# Capítulo 3: Diseño del sistema

Este capítulo tiene la función de describir la estructura del sistema para la monitorización y control de consumos eléctricos. En él, se describirán los elementos que intervienen, detallando la función que desempeñan, y las relaciones entre los elementos tanto hardware como software.

Este análisis tomará un **enfoque top-down**, es decir, donde se comenzará con una visión general del sistema para posteriormente adentrarse en los elementos individuales que lo conforman.

## 3.1 Descripción general de la arquitectura

Tras un análisis de las posibilidades para la arquitectura del sistema, comparando el ámbito, la escala actual y las posibilidades a futuro, una **arquitectura de 4 capas** cubre con suficiencia las necesidades del proyecto. Las capas que conforman el sistema son:

- **Capa de Aplicación:** Capa responsable de la lógica y la funcionalidad que presenta la aplicación web. Esta capa gestiona la UI con la que interactúa el usuario final. Esta interfaz tiene como propósito visualización de los datos en tiempo real, además del control remoto de las cargas. Se ha escogido React para el diseño e implementación de la **aplicación web**.
- **Capa de Procesamiento:** Capa encargada del procesamiento de los datos obtenidos por la capa de Percepción y transmitidos mediante la capa de Transporte. En el caso de este sistema, forman parte de esta capa: **Influxdb**, la base de datos escogida para el almacenamiento de los datos, y **Telegraf**, el agente de recolección de datos que suministra a la base de datos con información procesada.
- **Capa de Transporte:** La capa de transporte es la encargada de comunicar la capa de percepción con las capas superiores. En ella son definidos los protocolos y mecanismos de transporte utilizados para la transferencia de los datos entre la capa física (percepción) y la capa de procesamiento. Forma parte de esta capa el bróker de los mensajes MQTT, **Mosquitto Broker**, que envía los valores obtenidos desde el ESP32 hacia la base de datos, junto con el **servidor de Websockets**, que gestiona la comunicación entre el ESP32 y la aplicación web. Mediante WiFi, se establecerá la conexión con ambos y se podrán enviar y recibir mensajes utilizando MQTT y Websockets.
- **Capa de Percepción:** Se trata de la capa física, formada por **elementos hardware**. Esta normalmente incluye microcontroladores, sensores, dispositivos de E/S, actuadores y cualquier elemento que interactúe de forma directa con el entorno físico. Esta capa obtiene información del mundo real que es transmitida al resto del sistema, pudiendo luego actuar según sea el procesamiento y la respuesta a dicha información. Respecto a este sistema, en esta capa entran el **ESP32** como pieza central y fundamental para su correcto funcionamiento, el sensor **PZEM004T** y la placa de expansión con relés **XL9535-K4V5**. Para la configuración del ESP32 y la definición de los procesos que realizará se utilizará el lenguaje C++ en el entorno Visual Studio Code 2022 haciendo uso de la extensión ESP-IDF.

Mediante esta distribución la separación de responsabilidades queda claramente establecida y permite la **modularización**, y la **escalabilidad** para una posible expansión en el futuro.

Durante el análisis de qué arquitectura favorecía un mejor desarrollo del proyecto, se descartaron las arquitecturas de 5 y 7 capas ya que presentaban una complejidad innecesaria, y no era de interés entrar en modelo de negocio ni ámbitos similares.

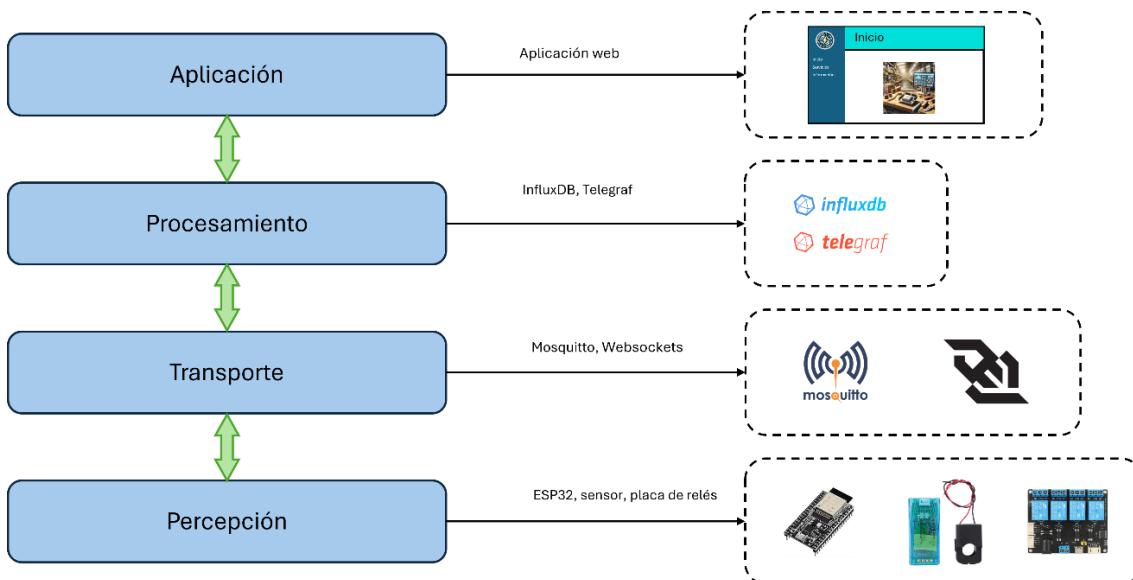


Figura 4: Arquitectura de 4 capas del sistema.

Tomando como referencia la arquitectura descrita y mostrada en la Figura 4, el flujo de los procesos de actividad y comunicación del sistema puede considerarse como se describe a continuación:

1. El sensor PZEM004T recoge las mediciones sobre el circuito eléctrico. Estos datos son enviados mediante conexión física al ESP32.
2. El ESP32 se conecta utilizando WiFi a la red y establece la comunicación con el bróker MQTT y con el servidor de Websockets para enviar un mensaje MQTT y JSON con los valores obtenidos por el sensor, respectivamente.
3. Respecto al bróker MQTT, este recibe los datos y manda un mensaje a todo cliente suscrito al tópico recibido. En este caso será la base de datos, teniendo como intermediario a Telegraf, un agente de servidor encargado de la recogida de información y su procesado para la posterior inserción en la base de datos.
  - 3.1. La base de datos recibe los datos, se insertan y se almacenan.
4. Respecto al servidor de Websockets, envía los datos recibidos a los clientes conectados, en este caso siendo la aplicación web la que recibe los valores.
  - 4.1. La aplicación web recibe los datos y los muestra al usuario en forma de tabla y de diferentes gráficas.
  - 4.2. En caso de que el usuario decida interactuar con las cargas, el flujo sería el inverso, iniciando la orden en la aplicación web, que pasaría por el servidor de Websockets, que

la enviaría a los clientes conectados, recibiéndola el ESP32 y actuando conforme a la información recibida.

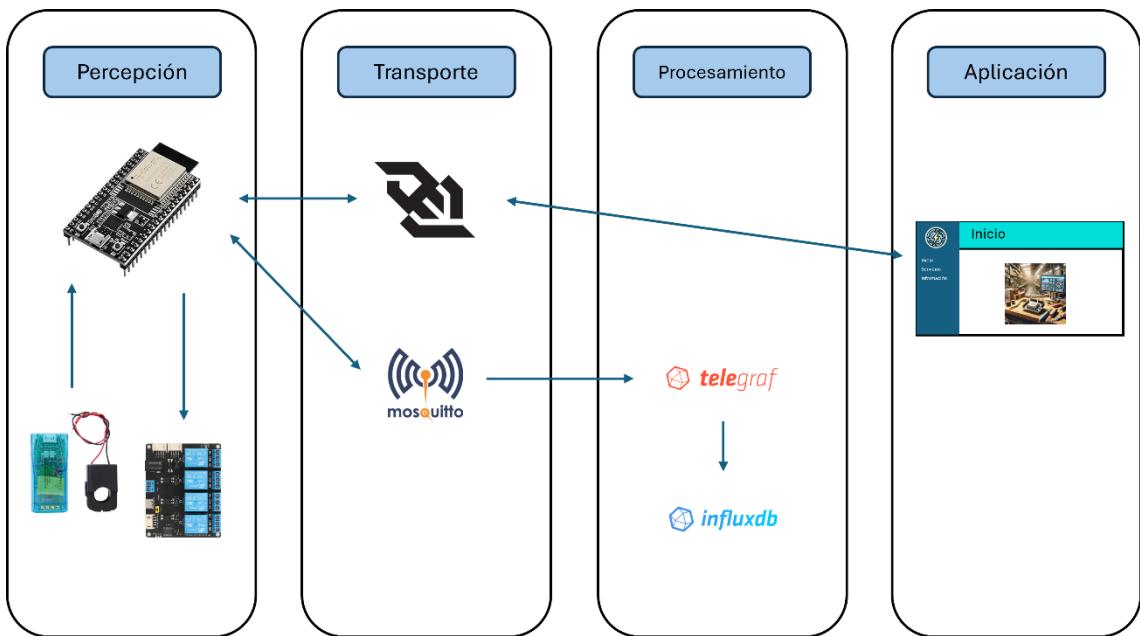


Figura 5: Flujo de información en el sistema

Por otro lado, puede decirse que la estructura de este proyecto se divide en **dos partes**, diferenciadas por el tipo de sus componentes. En primer lugar, los **componentes físicos (hardware)** forman el circuito encargado con la tarea de obtener las mediciones eléctricas deseadas, así como de llevar a cabo cualquier interacción pertinente sobre las cargas. En segundo lugar, tenemos una **parte intangible**, formada por elementos que se alojan en un PC o en un servidor, y que se encargan de la parte de almacenamiento y visualización de la información, y del envío de órdenes para el control de las cargas. A continuación, en la Figura 6 se muestra un esquema de la estructura del sistema:

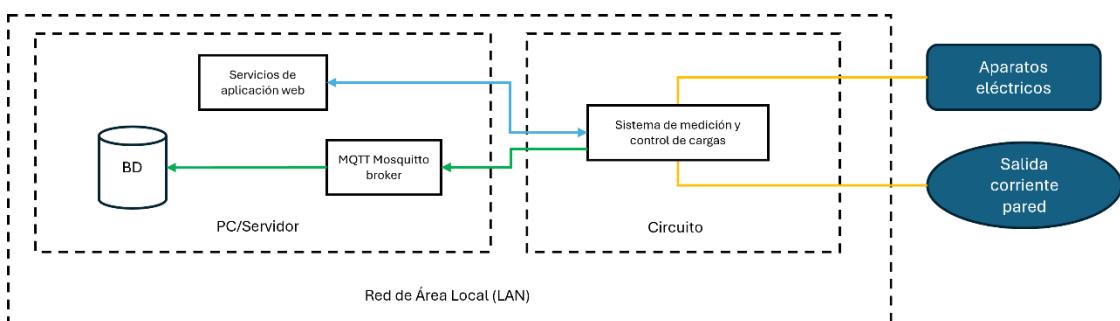


Figura 6: Estructura simplificada del sistema.

En la figura se pueden apreciar las dos partes mencionadas anteriormente. Estas se encuentran situadas dentro de la misma red local. Entrando en el significado de las conexiones, en naranja se designa la corriente eléctrica que estudia el circuito; en verde y de forma unidireccional, se representa el envío de mensajes desde el sistema de medición hacia la base de datos, siguiendo el modelo PUB/SUB de MQTT pasando primero por el bróker; finalmente, en azul se representa el intercambio de mensajes mediante websockets que realizan el sistema de medición y la aplicación web, donde el sistema de medición envía los valores obtenidos para que sean mostrados en la aplicación web, mientras que la aplicación web manda órdenes para el encendido o apagado de las cargas eléctricas. También puede apreciarse en la figura, cómo el sistema de medición interviene sobre el cableado antes del punto final de consumo. Esta actuación sobre el cableado intermedio puede establecerse de forma que se midan los valores para un único aparato (como se ha planteado en el prototipo realizado) o para múltiples puntos de consumo, bien sea una habitación de forma aislada o un domicilio completo.

### 3.2 Descripción detallada de la arquitectura

Una vez se han ofrecido unas nociones básicas acerca de la estructura del proyecto, procede realizar un análisis más detallado y explicar en mayor profundidad los componentes involucrados en las partes ya mencionadas. Comenzando por mostrar una representación más detallada de la estructura del sistema y las conexiones e interacciones entre componentes, reflejados en la Figura 7.

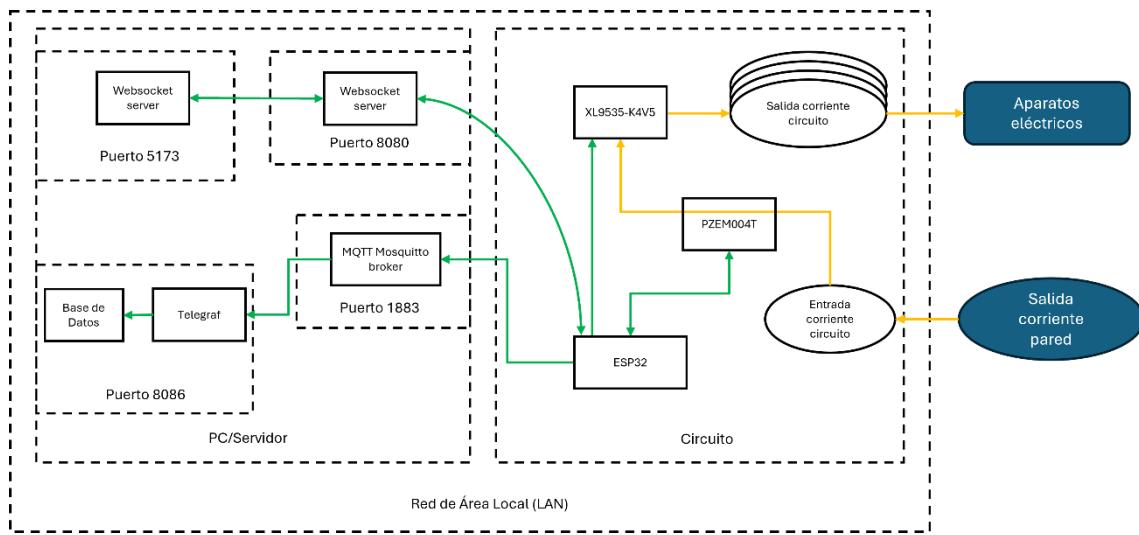


Figura 7: Estructura detallada del sistema.

El análisis en profundidad se realizará por capas, comenzando por la capa inferior hasta llegar a la más alta. Para ello se describirá la estructura, el funcionamiento y la configuración realizada sobre cada uno de los elementos que la conforman.

### **3.2.1 Capa de percepción**

Como ya se ha explicado anteriormente, esta capa es la parte del sistema en **contacto directo e interactúa con el entorno físico**. En ella suelen incluirse todo tipo de **componentes hardware** que permiten la obtención de información (como sensores) o que tengan efectos tangibles sobre el entorno en el que se encuentran (como servomotores, relés, ...). De este sistema, pueden englobarse dentro de esta capa tres componentes:

- El microcontrolador ESP32 Dev Kit C V4 NodeMCU.
- El kit PZEM-004T para medición de valores eléctricos.
- Módulo de relés expansible XL9535-K4V5.

Cada uno de ellos desempeña una función específica y fundamental que se detallará en los próximos apartados.

Además de los componentes, dentro de esta capa de percepción se explicarán las **conexiones** establecidas que permiten la interacción entre los citados componentes.

- Conexión mediante UART para comunicar el ESP32 y el sensor.
- Conexión mediante I2C para comunicar el ESP32 con el módulo de relés.

#### **3.2.1.1 ESP32**

Este microcontrolador de la empresa Espressif Systems, toma la base de modelos anteriores como el ESP8266, para añadir características extra muy atractivas para el desarrollador, como pueden ser conexión Wifi más rápida o distintas interfaces de conexión. En este proyecto se aprovecharán las siguientes características del ESP32:

- Conexión mediante **Wifi** a la red, permitiendo el envío de mensajes **MQTT** y el intercambio de mensajes con el servidor de **Websockets**.
- Conexión mediante **UART** al sensor **PZEM004T**.
- Conexión mediante el protocolo **I2C** a la placa de expansión con relés **XL9535-K4V5**.

El ESP32 toma un **papel fundamental** en el sistema, tal y como se puede ver en la Figura 7, ya que sirve como punto principal de la comunicación entre varios componentes:

- Recibe la información desde el sensor.
- Envía la información a la base de datos y a la aplicación web
- Recibe las órdenes de acción sobre las cargas.
- Envía las órdenes a la placa de expansión para que se hagan efectivas.

## Estructura del ESP32

El modelo empleado de ESP32 es el **ESP32 Dev Kit C V4 NodeMCU**. Este cuenta con un total de 38 pines, de los cuales 34 son *General Purpose Input/Output* (GPIO). De estos, 6 pueden utilizarse por pares para comunicación mediante UART, en el proyecto se usarán los pines 16 y 17 con este propósito. Otros sirven para comunicación I2C, en este caso se tomarán el pin 21 y el pin 22, uno servirá como *Serial Data* (SDA) y otro como *Serial Clock* (SCL), respectivamente.

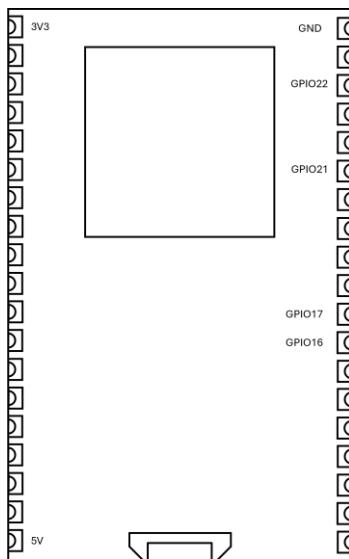


Figura 8: Mapa de pines ESP32

En la Figura 8 se puede ver la estructura del ESP32, donde se han señalado los pines que participan en la actividad del sistema. A continuación, se describirá la tarea de cada uno:

- **3v3 y 5v:** pines empleados para obtener 3,3V y 5V respectivamente, permitiendo alimentar otros componentes del circuito.
- **GND:** conexión a tierra.
- **GPIO16:** establecido como receptor de los mensajes enviados desde el sensor. Los mensajes mediante UART enviados desde el sensor serán transmitidos a este pin. Este pin estará conectado físicamente a la salida UART de transmisión del sensor, recibiendo los datos transmitidos.
- **GPIO17:** establecido como transmisor de los mensajes enviados hacia el sensor. Los mensajes mediante UART enviados hacia el sensor serán transmitidos a este pin. Este pin estará conectado físicamente a la salida UART de recepción del sensor, por él se transmitirá la orden de que el sensor debe enviar los datos.
- **GPIO21:** establecido como reloj para la comunicación mediante el bus I2C con la placa de relés. Su función es sincronizar el sistema mediante pulsos de reloj. Estará conectado físicamente con la interfaz I2C de la placa de relés.

- **GPIO22:** establecido como línea de datos en la comunicación mediante el bus I2C con la placa de relés. Por esta línea se moverán los datos de la comunicación, en este caso, las ordenes de encendido y apagado de cargas. Estará conectado físicamente con la interfaz I2C de la placa de relés.

Explicada la estructura física, la estructura lógica que contiene la funcionalidad del ESP32 se encuentra repartido entre varios archivos y directorios dentro del directorio raíz del proyecto. Examinando por consola el contenido del proyecto encontramos:

```
>> dir
```

Directorio: C:\projects\TFG\esp32

Mode	LastWriteTime	Length	Name
d----	14/06/2024 17:25		.vscode
d----	17/06/2024 21:18		build
d----	16/06/2024 18:40		main
d----	14/06/2024 17:25		managed_components
-a----	14/06/2024 17:25	30	.gitignore
-a----	14/06/2024 17:25	364	CMakeLists.txt
-a----	14/06/2024 17:25	429	dependencies.lock
-a----	16/06/2024 19:03	58186	sdkconfig
-a----	16/06/2024 18:27	58184	sdkconfig.old

De entre los elementos contenidos en el directorio raíz, algunos de ellos han sido creados de forma automática y no se han modificado posteriormente:

- **.vscode:** directorio creado por el entorno Visual Studio Code que contiene configuraciones para el proyecto, sobre las extensiones y el editor.
- **managed\_components:** directorio que se crea al incluir ciertos módulos que no se incluyen con ESP-IDF por defecto. Para este proyecto se ha añadido `esp_websockets_client.h`. Para incluirlo se utiliza la opción de ESP-IDF “IDF Component Registry” en VS Code, donde se busca e instala el paquete necesario.
- **build:** directorio creado durante la construcción del proyecto, en él se almacenan todos los archivos generados durante este proceso y que son esenciales para que el microcontrolador realice su actividad correctamente. Incluye archivos binarios,
- **CMakeLists.txt:** es un archivo de configuración para la herramienta CMake, encargada de definir cómo se debe compilar y enlazar el proyecto. Este archivo contiene la versión de CMake mínima necesaria, la ruta con las dependencias de CMake y el nombre del proyecto.
- **.gitignore:** archivo creado por Git para indicar qué archivos y directorios son ignorados por el control de versiones.

- **dependencies.lock**: archivo que bloquea las versiones de las dependencias del proyecto, asegurando que dichas versiones específicas sean utilizadas.
- **sdkconfig**: archivo que almacena la configuración del SDK de ESP-IDF. Incluye configuraciones de compilación, configuraciones de hardware y parámetros del sistema.
- **sdkconfig.old**: copia de seguridad del archivo **sdkconfig**. Permite revertir cambios, manteniendo y presentando una versión anterior .

El elemento restante es el directorio **main**, este contiene el código que define la funcionalidad de la aplicación. Está compuesto por: **main.c** como punto de inicio del programa; **CMakeLists.txt**, que define que archivos fuente se compilan; **idf\_component.yml** que se genera automáticamente al añadir el módulo `esp_websocket_client.h`; y 6 módulos compuestos de varios archivos, que cubren las diferentes funcionalidades del sistema.

Comenzando por **main.c**, este contiene la referencia para incluir dependencias necesarias, la definición de parámetros importantes y pines involucrados, la tarea principal del sistema `main_task()`, y el método principal del programa `app_main()`. Las configuraciones que se incluyen en `app_main()` son:

- 1- Inicialización de la memoria volátil, necesario para establecer conexión Wi-Fi, mediante `nvs_flash_init()`.
- 2- Inicialización de la conexión Wi-Fi mediante `wifi_connect_init()`, que incluye varias acciones definidas en el módulo de conexión Wi-Fi, y posterior conexión a la red de pruebas para la maqueta usando `wifi_connect_demo()`.
- 3- Configuración de la fecha y hora utilizando el protocolo SNTP para establecer conexión al servidor pool.ntp.org, mediante `ntp_setup()`.
- 4- Iniciar la conexión al bróker MQTT utilizando `mqtt_connect_init()`.
- 5- Iniciar la conexión al servidor de Websockets mediante `websockets_connect_init()`.
- 6- Establecer la configuración para la comunicación mediante I2C mediante `i2c_master_init()`, y apagando de inicio todos los relés, utilizando `i2c_relay_init(0)`.
- 7- Finalmente, ejecución de la tarea principal `main_task()`.

A excepción de la tarea principal, que sí está definida en `main.c`, todas las funciones empleadas para la inicialización y configuración del sistema y sus conexiones pertenecen a alguno de los 6 módulos definidos en el proyecto. Estos son:

- Módulo de conexión wifi, que incluye los archivos: `wifi_connect.h`, `wifi_connect.c` y `wifi_connect_err.c`.
  - `wifi_connect.h`: en él se declaran las funciones que se definen en `wifi_connect.c`.

- `wifi_connect.c`: además de incluir las dependencias necesarias, define cuatro funciones y un manejador de eventos.
    - La primera función (`wifi_connect_init()`) inicializa la interfaz de red y el bucle de eventos, establece la configuración para conexión Wi-Fi y la inicializa, establece el tipo de almacenamiento utilizado y enlaza el manejador de eventos con el bucle de eventos.
    - La segunda función (`wifi_connect_station()`) define la interfaz de red del dispositivo estableciéndolo como estación (STA), y conectándolo al punto de acceso (AP) mediante las credenciales pasadas como parámetro (ssid y contraseña de la red Wi-Fi, y timeout).
    - La tercera función (`wifi_connect_demo()`) tiene como propósito conectar a la red de pruebas para la maqueta, sin mostrar tener que pasar las credenciales desde fuera del propio archivo.
    - La última función (`wifi_disconnect()`) realiza la desconexión de la red y la limpieza de la interfaz.
    - Finalmente, el manejador de eventos (`wifi_event_handler()`) contempla acciones para los eventos de: obtención de dirección IP propia, inicio de conexión, establecimiento correcto de conexión, y desconexión, estableciendo reintentos de conexión para determinadas desconexiones.
  - `wifi_connect_err.c`: contiene los códigos de error que se muestran en caso de error durante alguno de los eventos relacionados con Wi-Fi.
- Módulo de configuración sntp, formado por: `sntp.h` y `sntp.c`.
  - `sntp.h`: en él se declaran las funciones que se definen en `sntp.c`.
  - `sntp.c`: contiene dos funciones que permiten la sincronización del reloj del sistema para ofrecer la fecha y la hora reales.
    - La primera función (`sntp_setup()`) establece el modo de sincronización y el servidor ntp al que conectar y al recibir la notificación de sincronización invoca a la segunda función.
    - La segunda función (`on_got_time()`) establece la zona horaria donde se encuentra el dispositivo para sincronizar correctamente la fecha y la hora.
- Módulo de conexión y comunicación por MQTT, formado por: `mqtt_connect.h` y `mqtt_connect.c`.
  - `mqtt_connect.h`: en él se declaran las funciones que se definen en `mqtt_connect.c`.
  - `mqtt_connect.c`: contiene un manejador de eventos, y las funciones que permiten establecer una conexión y enviar mensajes al bróker MQTT.

- La primera función (*mqtt\_connect\_init()*) permite la conexión con el bróker, declarando la configuración básica para crear un cliente, como la dirección del bróker (mqtt://192.168.216.210:1883) dentro de la red para la maqueta. Enlaza el manejador de eventos con los eventos relacionados con el cliente creado. Finalmente, inicia la conexión con el cliente creado.
  - La segunda función (*mqtt\_send()*) permite la publicación de datos a un tópico específico.
  - La tercera función (*mqtt\_send\_telemetry()*) reúne el envío de los datos de cada uno de los parámetros eléctricos en una sola función a la que se le pasa la estructura que contiene los datos.
  - La cuarta función (*mqtt\_disconnect()*) permite la desconexión del bróker.
  - Finalmente, el manejador de eventos (*mqtt\_event\_handler()*) actúa tras los eventos de conexión, desconexión, suscripción, eliminación de la suscripción, publicación, recepción de datos y error, mostrando en el log el evento correspondiente y en su caso un mensaje con el error.
- Módulo de conexión y comunicación mediante Websockets, formado por: `websockets_connect.h` y `websockets_connect.c`.
  - `websockets_connect.h`: en él se declaran las funciones que se definen en `websockets_connect.c`.
  - `websockets_connect.c`: contiene las funciones que permiten la conexión al servidor de Websockets y el envío y recepción de mensajes.
    - La primera función (*websockets\_connect\_init()*) crea la configuración del cliente que se conectará al servidor, indicando la URL y los *timeouts*. Crea el cliente con la configuración definida y enlaza el manejador de eventos al cliente. Finalmente, establece la conexión.
    - La segunda función (*message\_to\_json()*) recibe como parámetros un buffer de datos y los datos de valores eléctricos, en el que se insertará el mensaje en formato JSON que se enviará al servidor. En este mensaje se incluye la hora y fecha de creación del mensaje, junto con los valores eléctricos.
    - La tercera función (*relay\_status\_send()*) permite notificar del estado actual de un relé. De esta forma se crea un mensaje JSON con los parámetros de la función: buffer para almacenar el mensaje, numero del relé y estado del relé. El mensaje es enviado.
    - La cuarta función (*websockets\_send()*) hace uso de la segunda función para obtener el mensaje a enviar con los parámetros recibidos (los mismos que la segunda función) y envía el mensaje.

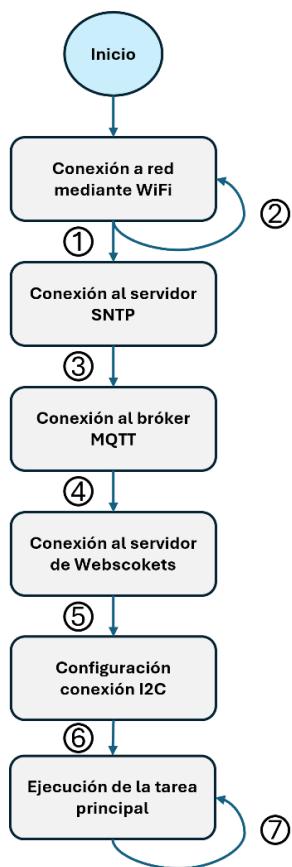
- La quinta función (*websockets\_disconnect()*) permite la desconexión limpia del servidor.
  - Finalmente, el manejador de eventos (*websocket\_event\_handler()*) actúa al detectar los eventos: conexión al servidor, desconexión al servidor y datos recibidos. Al recibir datos comprueba si no es un mensaje vacío, de no serlo se comprueba si es un mensaje de tipo ‘order’ procedente de la aplicación web, para modificar el estado de un relé. De serlo, y existir el relé indicado en el mensaje, se intenta modificar su estado (si el relé ya posee ese estado no hay cambio pese a ejecutar la operación).
- Módulo de conexión y comunicación mediante I2C, formado por: *i2c\_relay\_comm.h* y *i2c\_relay\_comm.c*.
  - *i2c\_relay\_comm.h*: en él se declaran las funciones que se definen en *i2c\_relay\_comm.c*.
  - *i2c\_relay\_comm.c*: contiene las funciones para la configuración de la conexión y comunicación mediante interfaces I2C.
    - La primera función (*i2c\_master\_init()*) crea la configuración que tendrá la comunicación mediante I2C, estableciendo el modo que toma el dispositivo (maestro), los pines que servirán como SCL (22) y SDA (21), habilita resistencias *pull-up* para ambas líneas, establece la configuración y finalmente instala el driver I2C. Se realiza la operación, recibiendo el mensaje de éxito o error. Finalmente, se libera el manejador de comandos.
    - La segunda función (*i2c\_relay\_write()*) crea un manejador de comandos I2C. Se inicia una nueva transacción I2C. En ella se añaden la dirección del dispositivo receptor y el registro donde se va a escribir. Se añaden los datos a escribir en el registro y se establece el final de la transacción.
    - La tercera función (*i2c\_set\_relay\_state()*) recibe el número del relé y el estado deseado, y utilizando operadores lógicos calcula el valor que debe adquirir el registro para modificar el estado del relé indicado sin afectar al resto. Despues escribe el nuevo valor en el registro y actualiza la variable que almacena el estado de los relés, mostrando un mensaje de éxito o error al final del proceso.
    - La cuarta función (*relay\_off()*) sirve para apagar un relé específico.
    - La quinta función (*relay\_on()*) sirve para encender un relé específico.
    - La sexta función (*i2c\_relay\_init()*) es usada para establecer el estado inicial de todos los relés del sistema, apagándolos.
- Módulo de conexión y comunicación con el sensor PZEM-004T, formado por: *pzem004t\_comm.h* y *pzem004t\_comm.c*.

- pzem004t\_comm.h: en él se declaran las funciones que se definen en pzem004t\_comm.c. Además, en él se define la estructura para almacenar los valores de las mediciones, la estructura *power\_values\_s*. Tendrá un atributo de tipo float por cada parámetro de la medición en total 7 (contando el parámetro de alarmas que alerta de cualquier problema).
- pzem004t\_comm.c: contiene funciones que permiten la comunicación mediante el interfaz UART utilizando tramas Modbus-RTU.
  - La primera función (*CRC16()*) se encarga de calcular el checksum de la trama Modbus utilizando CRC-16 como método de detección de errores.
  - La primera función (*update\_values\_cmd()*) crea un mensaje que indica el dispositivo esclavo al que va dirigido (1), la función que realizar (4: lectura de registros), el registro donde iniciar (0) y el número de registros a leer (10). Al final del mensaje se añade el *checksum* resultado de invocar a la primera función pasando como argumento el mensaje sin *checksum*.
  - La tercera función (*parse\_values()*) obtiene los valores de los parámetros eléctricos de las mediciones a partir de los bytes recibidos en la trama procedente del esclavo. Muestra por consola los valores al finalizar la conversión.

Finalmente, la tarea principal, que contiene el código para la solicitud, el procesamiento y el envío de los datos. La función ***main\_task()*** comienza con la creación y definición de la configuración que regirá la comunicación mediante UART. En esta configuración se detalla: la velocidad de transmisión de datos, 9600; el número de bits de datos por cada trama, 8 bits; el bit de paridad, no se utilizará; sólo 1 bit de parada; el control de flujo para la comunicación no utilizará las señales de control *Ready to Send/Clear to Send* (RTS/CTS); la fuente del reloj, la predeterminada del sistema. Una vez definida la configuración, se instala el driver UART, se establece la configuración definida y se detallan el puerto y los pines involucrados, el puerto UART 1 y los pines 16 (Recepción de datos, RXD) y 17 (Transmisión de datos, TXD). Con esto termina la configuración de UART, y se pasa a crear dos estructuras para almacenar datos. La primera es un buffer que servirá para almacenar temporalmente las órdenes de lectura de datos, y que tras recibir los datos serán sobrescritas con el mensaje recibido. La otra, del tipo *power\_values\_s*, donde se escribirán los valores una vez procesado el mensaje. A continuación, se encuentra el bucle más importante del programa. En él se da la constante sucesión de lectura de datos, procesamiento del mensaje, creación de mensaje para enviar, y envío por Websockets y MQTT. Finalmente se establece un tiempo de espera de dos segundos entre un par lectura-envío y el siguiente.

## Funcionamiento del ESP32

El programa comienza con la realización de una **serie de configuraciones**, que son necesarias para que posteriormente tenga la capacidad de realizar la tarea principal. Este flujo se describe en la Figura 9, mostrada a continuación:



Al iniciarse, lo primero que necesita el ESP32 es establecer una **conexión Wi-Fi** a la red correspondiente. Esto le permitirá tener acceso y conectarse más adelante al bróker MQTT y al servidor de Websockets (1). En caso de error o perdida de conexión se reconectará al punto de acceso (2).

Posteriormente, **sincronizará el reloj del sistema** conectándose a un servidor NTP mediante el protocolo SNTP. De esta forma se podrá obtener la fecha y la hora a la que se registra la recepción de las mediciones para incluir en los mensajes a la aplicación (3).

A continuación, **configurará el cliente MQTT** y establecerá conexión con el bróker de Eclipse Mosquitto en <http://192.168.216.210:1883> (4).

Tras conectar con el bróker, se configurará el cliente y establecerá **conexión con el servidor de Websockets** en <http://192.168.216.210:8080> (5).

La última configuración que se realiza es la correspondiente a la **comunicación mediante el interfaz I2C** con el módulo de relés (6).

Finalmente, se pone en marcha la **tarea principal** del sistema, que consiste en un bucle que se describirá a continuación (7).

Figura 9: Flujo principal del programa

La **tarea principal** del sistema consta de dos partes:

Primero una parte que realiza la **configuración necesaria** para llevar a cabo la **comunicación a través del interfaz UART**. En primer lugar, se definen los valores de la configuración, como el bit de paridad y la velocidad de transmisión (1). Después, se instala el driver UART, se establece la configuración definida, y se definen el puerto y los pines involucrados (2). Se crean dos estructuras para contener datos, uno que se utilizará para almacenar la orden de lectura antes de mandarla y los datos al recibirlos, y otro para almacenar los valores tras ser extraídos de la otra estructura procesados (3).

Una vez se ha realizado la configuración, se pasa a la segunda parte. Esta se compone de un **bucle sin final** que realiza **peticIÓN, procesamiento y envío de datos**. En primera instancia, se envía la orden de lectura, en caso de error o recibir un mensaje vacío como respuesta se vuelve a enviar la petición (5). En caso de recibir un mensaje correcto, se extraen los datos (4). Finalmente, se generan el mensaje JSON para enviar por Websockets y el mensaje MQTT, antes de volver a solicitar los datos comenzando de nuevo el proceso (6).

## Conexiones del ESP32 dentro de la capa de percepción

### **UART**

La comunicación entre el ESP32 y el sensor PZEM004T sucede mediante **conexiones físicas**. Estas se rigen por el protocolo *Universal Asynchronous Receiver Transmitter* mejor conocido por las siglas **UART**. Este tipo de conexión soporta comunicación simplex (una dirección de transmisión), semidúplex (ambos pueden comunicar, pero sólo uno a la vez) y dúplex completo (comunicación desde ambos extremos de forma simultánea).

La conexión física se lleva a cabo mediante dos cables que conectan:

- El pin 17 del ESP32 con el pin Rx del puerto TTL del sensor
- El pin 16 del ESP32 con el pin Tx del puerto TTL del sensor

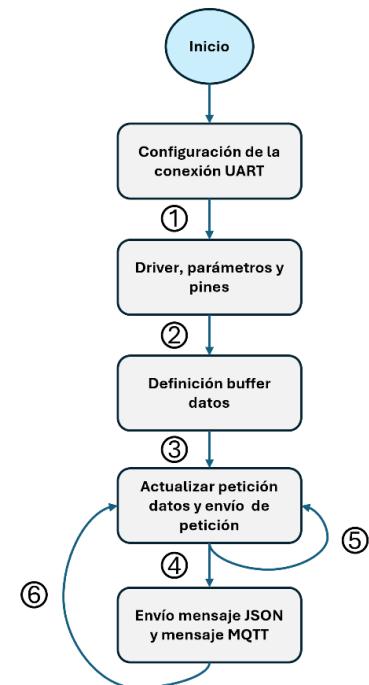


Figura 10: Flujo de la tarea principal del sistema

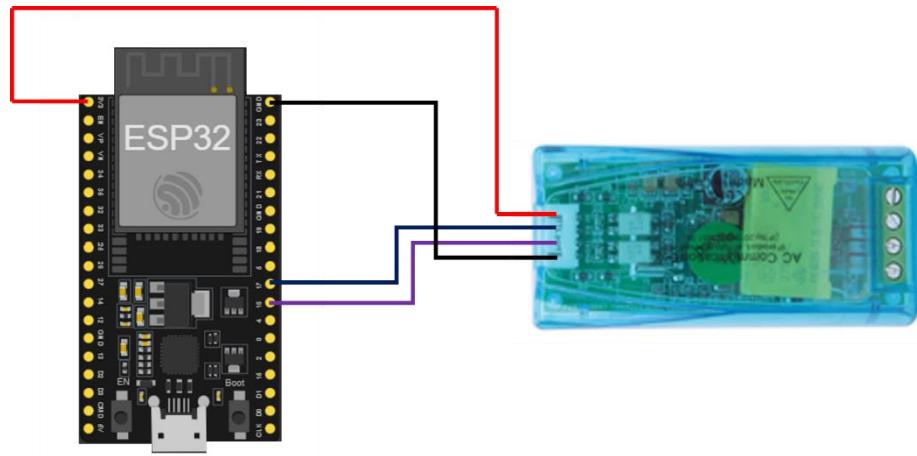


Figura 11: Conexiones entre ESP32 y PZEM-004T

Para el correcto funcionamiento del sensor será necesario además conectarlo a 3V3 y a línea de tierra (GND).

Una vez realizadas las conexiones, y con las configuraciones establecidas, la interacción entre ambos componentes es como sigue:

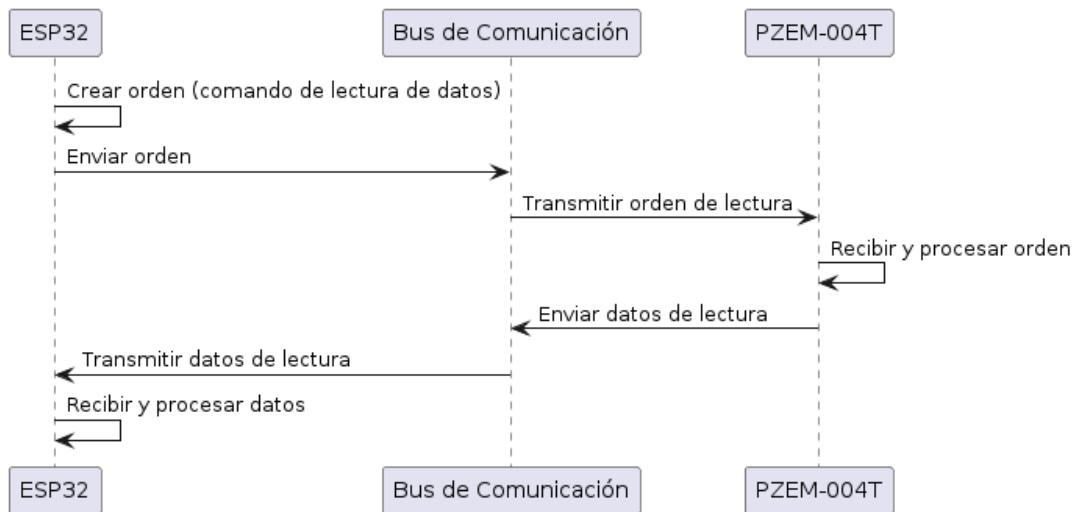


Figura 12: Diagrama de secuencia - Interacción entre ESP32 y PZEM-004T

En primer lugar, el ESP32 **crea una trama de bytes** que será escrita en el bus y llegará al sensor. El esquema seguido es el de maestro esclavo, por lo tanto, habrá una trama ligeramente distinta para cada dispositivo. La trama del maestro (ESP32) constará de 8 bytes y tendrá la siguiente estructura:

- **Dirección del esclavo** (1 byte): número hexadecimal entre 0x01 y 0XF7. El sensor tiene dirección 0x01.

- **Orden** (1 byte): comando de lectura que se corresponde con 0x04.
- **Dirección del primer registro** (2 bytes): Indica la dirección donde se encuentran los primeros datos que deben leerse y mandarse. Dado que se quieren leer todos los valores se inicia en el registro 0, al ser de dos bytes sería 0x00 0x00.
- **Número de registros** (2 bytes): indica el número de registros sobre los que actúa la operación. Se deben leer todos los registros, por tanto, 0x00 0x0A para leer los 10 registros con datos.
- **CRC Check** (2 bytes): bytes dedicados a la detección de posibles errores. Se calculan mediante CRC (*Cyclic Redundancy Check*), una serie de operaciones sobre el mensaje a enviar que generan un valor que se incluye en el mensaje. El receptor del mensaje realiza estas mismas operaciones sobre los datos recibidos, si coinciden con el cálculo incluido por el emisor, el mensaje es correcto, en caso contrario, se deberá solicitar un reenvío o actuar en consecuencia.

Una vez generada la trama, esta se envía a través del bus, que lo transporta hasta cualquier dispositivo conectado que esté escuchando, en este caso el sensor. El sensor recibe la orden, toma los datos y genera una nueva trama que enviar de vuelta hacia el ESP32. El formato de una trama de esclavo correcta es el siguiente:

- **Dirección del esclavo** (1 byte): debe coincidir con la incluida en la orden del maestro para ser tomada en cuenta.
- **Orden** (1 byte): debe coincidir con la orden de la trama del maestro para ser tomada en cuenta.
- **Datos:** el número de bytes de esta sección dependerá de el número de registros que tuvieran que ser leídos. En esta conexión deberán ser 10 bytes, uno por registro, siendo que algunas mediciones se dividen en 2 bytes.
- **CRC Check** (2 bytes): de nuevo, el valor incluido debe ser calculado por el receptor para asegurar que el mensaje no contiene errores.

Esta trama se envía a través del bus, y llega al ESP32 que recibe y procesa los datos, en este caso para ser mandados mediante MQTT hacia la base de datos.

## I2C

La otra comunicación dentro de la capa de percepción es entre el ESP32 y el módulo de relés XL9535-K4V5. Esta es posible mediante las siguientes conexiones:

- El pin 22 del ESP32, configurado para actuar como SCL, con el pin SCL de una de las interfaces I2C del módulo.
- El pin 21 del ESP32, configurado como línea de datos, con el pin SDA de una de las interfaces I2C del módulo.

Todas las interfaces I2C comparten el mismo bus, por lo que es indiferente a que pines llevar las conexiones, por motivos visuales y de limpieza se utilizarán pines de la misma interfaz y lo más cercanos posible al microcontrolador.

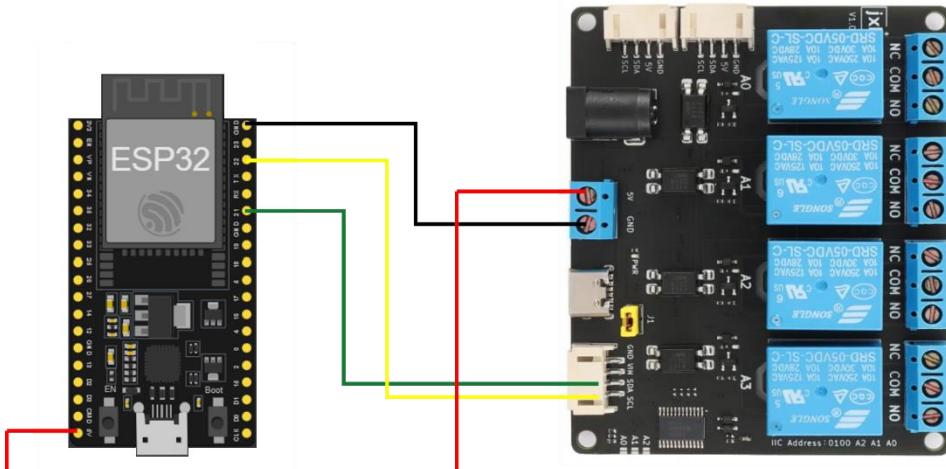


Figura 13: Conexiones entre ESP32 y XL9535-K4V5

Al igual que para la conexión mediante UART, es necesario dotar de alimentación (5V) y una línea de tierra (GND) al componente que se conecta al ESP32.

Igual que con la conexión UART, se puede describir la interacción entre ambos componentes mediante un diagrama de secuencia:

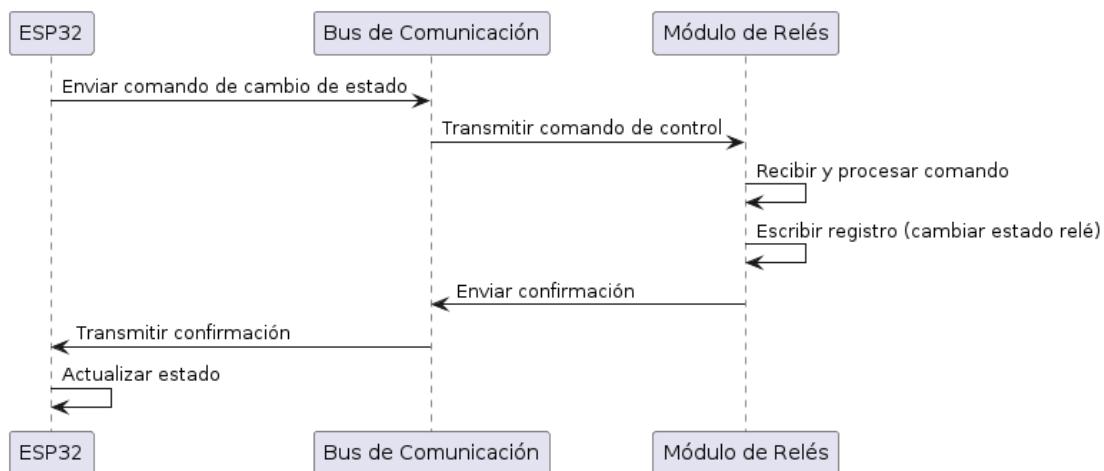


Figura 14: Diagrama de secuencia - Interacción entre ESP32 y XL9535-K4V5

De nuevo, tenemos que escribir una **orden que se transmite por el bus**. En este caso, para cambiar el estado de un relé se debe **modificar el registro 0x02** del módulo. Este contiene 4 bits de forma que cada bit corresponde a un relé, si el bit está a 1 el relé estará encendido y si está a 0, apagado. Para no alterar el estado de otros relés al cambiar uno concreto será necesario tener

en cuenta los 4 bits, modificando únicamente el bit correspondiente manteniendo los demás intactos. Por ejemplo, si se encuentran los relés 0 y 1 encendidos encontraremos ‘0011’ en el registro 0x02. Para encender, pongamos, el registro 3 el valor escrito en el registro 0x02 será ‘1011’.

Una vez definido el valor que debe tomar el registro, esta orden de modificar el valor del registro 0x02 se transmite por el bus y llega al módulo. Una vez allí, se modifica el valor, con el efecto físico correspondiente en los relés<sup>9</sup>. Entonces, el módulo devuelve la confirmación de que se ha escrito correctamente en el bus, y cuando el ESP32 la recibe, actualiza la variable usada para registrar el estado de los relés.

### 3.2.2 Capa de transporte

La capa de transporte se sitúa justo sobre la capa de percepción, se encarga de proporcionar una comunicación segura y transparente entre los diferentes elementos involucrados en el sistema. En ella se definen los protocolos empleados para el intercambio de información entre componentes. En este sistema se emplean dos protocolos:

- El protocolo de mensajería MQTT.
- El protocolo de intercambio de datos Websocket.

#### 3.2.2.1 MQTT

Para la comunicación entre el ESP32, que actúa como intermediario para el sensor, y Telegraf, que actúa como intermediario para la base de datos, se ha escogido el protocolo de mensajería MQTT.

A la hora de implementar el protocolo, se ha escogido el bróker de código abierto Eclipse Mosquitto. Una opción ligera y fácil de implementar que permite una fluida comunicación entre los dispositivos que se conecten a ella, tanto para publicación (envío de mensajes) como para suscripción (recepción de mensajes). Como bróker Mosquitto nos ofrece diferentes características bastante atractivas para nuestro sistema:

- **Creación de sistemas de tópicos** muy sencilla y dinámica, permitiendo crear nuevas entradas en tiempo real y facilitando la escalabilidad del sistema con posibles inclusiones de nuevos dispositivos en el futuro.
- **Retención de mensajes**, que permite al usuario acceder al último mensaje aún si su recepción se dio estando fuera de línea.
- Soporte para niveles de **Quality of Service (QoS)** 0 (entrega de mensaje máximo una vez), 1 (entrega de mensaje mínimo una vez hasta que la transferencia se complete correctamente) y 2 (entrega de mensaje exactamente una vez asegurando la recepción por parte de los suscriptores). El valor para esta característica la establece el cliente a la hora de hacer la publicación del mensaje.

9. Los relés cuentan con tres puntos de contacto: NC, NO Y COM. COM siempre está conectado a uno de los otros dos. Mediante cambios en la polaridad producidos por una bobina magnética, se cambia el canal al que se conecta COM, cortando el flujo de corriente por el circuito, o suministrando corriente al circuito. Por esto se dice que el relé conecta o desconecta las cargas.

Dentro de la configuración las modificaciones que se han realizado son escasas, ya que la configuración básica de Eclipse Mosquitto es suficiente para la actividad que va a realizar el sistema. Los cambios han de realizarse en el archivo de configuración **mosquitto.config** que deberá de incluirse como parámetro al iniciar el servicio del bróker, estos son:

- Dentro del apartado *Listeners*, cambiar la configuración de Mosquitto para escuchar conexiones MQTT en el puerto 1883 en todas las interfaces de red disponibles. Para ello basta con añadir la línea:

```
listener 1883
```

En caso de querer escuchar de una dirección IP concreta se deberá añadir al final de la línea la dirección, de la siguiente forma:

```
listener 1883 192.168.1.99
```

- En el apartado *Security* se encuentra el otro fragmento modificado de la configuración de mosquitto. En este apartado se eliminará la necesidad de acceder mediante credenciales, permitiendo el acceso anónimo al bróker. El propósito de este cambio es agilizar el proceso de cara a la maqueta, para una implementación segura se deberá incluir el acceso con usuario y contraseña verificados por el sistema para evitar cualquier incidente. La línea añadida es la siguiente:

```
allow_anonymous true
```

Una vez se ha finalizado la configuración, para poner en marcha el servicio del bróker hay que acceder mediante consola a la carpeta que contiene los archivos de Eclipse Mosquitto y una vez ahí, utilizar el siguiente comando:

```
.\mosquitto.exe -c .\mosquitto.conf
```

De este modo se iniciará el bróker aplicando la configuración que habíamos personalizado acorde a las necesidades del sistema. Una vez hecho, el bróker será completamente funcional, y estará a la espera de la suscripción de clientes y la publicación de mensajes.

### 3.2.2.2 Websockets

La conexión entre el ESP32 y la aplicación web se hace mediante el uso de Websockets. Para ello se necesita preparar un servidor de Websockets que sirva el propósito de hacer de intermediario entre ambos componentes del sistema. El primer paso para crear el servidor es instalar las dependencias necesarias, aprovechando el proyecto ya creado para la aplicación web. Se deberá utilizar el siguiente comando:

```
>> npm install ws
```

Tras instalar las dependencias, se creará con JavaScript un archivo con la funcionalidad, incluyéndolo en la carpeta raíz del proyecto de aplicación web. Para poder arrancar el servidor, se deberá modificar el archivo package.json, añadiendo en la sección *scripts* lo siguiente:

```
"start:ws": "node websocket-server.js"
```

De este modo se podrá utilizar el script **npm run start:ws** para iniciar el servidor de Websockets.

El archivo JavaScript con la lógica del servidor es **websocket-server.js**. En él se crea una instancia de WebSocketServer y se asigna al puerto 8080. En este archivo también se define qué debe hacer el servidor.

Se definen tres *event listener*<sup>10</sup> dentro de este archivo. El primero de ellos es del servidor, puede considerarse el más general, reacciona cuando un cliente realiza una conexión y salvo que se desconecten todos los clientes se mantiene a la escucha de eventos. Cuando el servidor detecta una nueva conexión muestra por pantalla un texto indicando la conexión exitosa. Es a raíz de la primera conexión exitosa que se ponen en marcha los otros dos *event listener*:

- Uno de ellos dedicado a la recepción de mensajes. Cuando se recibe un mensaje, se muestra por consola el mensaje y después, para cada cliente conectado, si se encuentra disponible para recibir mensajes, el mensaje se pone en circulación con destino el cliente.
- El otro dedicado a la desconexión de clientes. Muestra por consola un mensaje indicando la desconexión.

Cuando el servidor se ha iniciado correctamente se muestra un mensaje indicando la dirección y el puerto donde se encuentra. Dentro de la red utilizada para la maqueta la dirección del servidor de Websockets es <http://192.168.216.210:8080>.

### 3.2.3 Capa de procesamiento

La capa de procesamiento se encuentra entre la capa de aplicación y la capa de transporte. Es responsable de la **transformación de los datos** procedentes de la capa de percepción y entregados por la capa de transporte. Estas operaciones donde se manipulan los datos no están directamente relacionadas con la lógica de negocio, pero son vitales para brindar a la capa superior información fiable y precisa que utilizar en sus procesos. El **almacenamiento de datos** también se incluye en esta capa.

Los elementos del sistema que corresponden a esta capa son:

- El gestor de bases de datos **Influxdb**.
- El agente de recolección y procesamiento de datos **Telegraf**.

10. Función de JavaScript que espera a que ocurra un evento específico y ejecuta una acción en respuesta a ese evento. [26]

### 3.2.3.1 Influxdb

La gestión apropiada de los datos almacenados en el sistema es de vital importancia. El poder **acceder de forma rápida y sencilla a los datos** en cualquier circunstancia o la **fiabilidad** durante la inserción de los datos son características deseables para una base de datos. Si a esto sumamos la posibilidad de **visualización** de los datos presentados mediante distintas gráficas y herramientas, la posibilidad de añadir *plugins* y complementos que ayuden al **procesado** de los datos, y el acceso seguro mediante credenciales entre otras muchas opciones obtenemos Influxdb.

El atractivo que tiene este gestor de bases de datos, obviando las características ya mencionadas, es que está diseñado íntegramente para el trabajo con series temporales. El sistema tiene como propósito la medición constante de valores, por ello, es necesario una forma de almacenar los datos que facilite la visualización en el tiempo y la comparación entre datos. Influxdb se erige sin duda como una de las opciones óptimas para esta situación. Sin duda, lo que decanta la decisión es que además de la gestión a través de consola (CLI), Influx dispone de una interfaz de usuario intuitiva que permite realizar la configuración de manera gráfica.

Para usar Influxdb existen diferentes opciones. La compañía ofrece una suma limitada de créditos para la realización de pruebas con Influxdb Cloud, sin embargo, siguiendo la línea del proyecto de alojar los componentes de manera local en el PC, se ha escogido **Influxdb OSS v2**. Se trata de la **opción de código abierto** de Influxdb que permite tener el gestor totalmente localizado en el PC.

Para ello se debe descargar el instalador desde la página oficial de Influxdb: <https://docs.influxdata.com/influxdb/v2/install/?t=Windows>.

Y extraer los archivos y ubicarlos en una dirección específica utilizando los comandos proporcionados por la compañía:

```
>> Expand-Archive .\influxdb2-2.7.6-windows.zip -  
DestinationPath 'C:\Program Files\InfluxData\'  
  
>> mv 'C:\Program Files\InfluxData\influxdb2-2.7.6' 'C:\Program  
Files\InfluxData\influxdb'
```

Con esto ya se tendría el gestor de bases de datos alojado en el PC. Para pasar a configurarlo, lo primero es arrancar por primera vez el servicio. Bastará con desplazarse (o mantenerse) al directorio con los archivos recientemente instalados e iniciar el gestor.

```
>> cd -Path 'C:\Program Files\InfluxData\influxdb'  
  
>> ./influx
```

Una vez el gestor se encuentre operando, visitando <http://localhost:8086> en el navegador tendremos acceso a la UI de influx. La primera vez que entremos se deberá llevar a cabo la configuración, donde se establecerán el usuario y la contraseña, se definirá la organización a la que pertenece el usuario y se creará un *Bucket*. Los *Bucket* son ubicaciones donde se almacenan las series temporales, pueden configurarse para retener durante ciertos períodos de tiempo los datos y luego desecharlos. Tras introducir estos datos, se proporcionará un *operator API token*, una llave que otorga permisos totales de lectura y escritura sobre los recursos de todas las organizaciones. Es decir, concede permisos de administrador al usuario al que está ligada la llave.

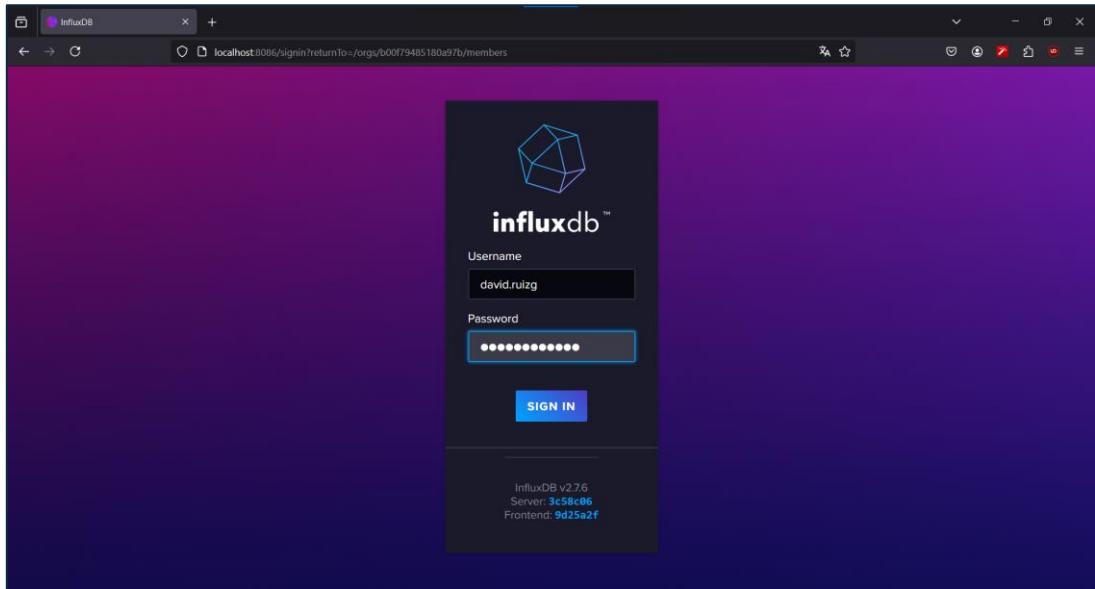


Figura 15: Pantalla Influxdb - Inicio de sesión con credenciales

Una vez introducidas correctamente las credenciales, se puede pasar a la configuración de los *Buckets*. En el menú de la izquierda de la pantalla se selecciona el apartado *Buckets* y pulsando el botón azul *Create Bucket* se inicia el proceso de creación, donde se establece el nombre que tomará y el tiempo de retención de los datos. Para este último se puede escoger entre nunca o un periodo concreto de tiempo. El *Bucket* utilizado en este proyecto tiene el nombre de TFG.

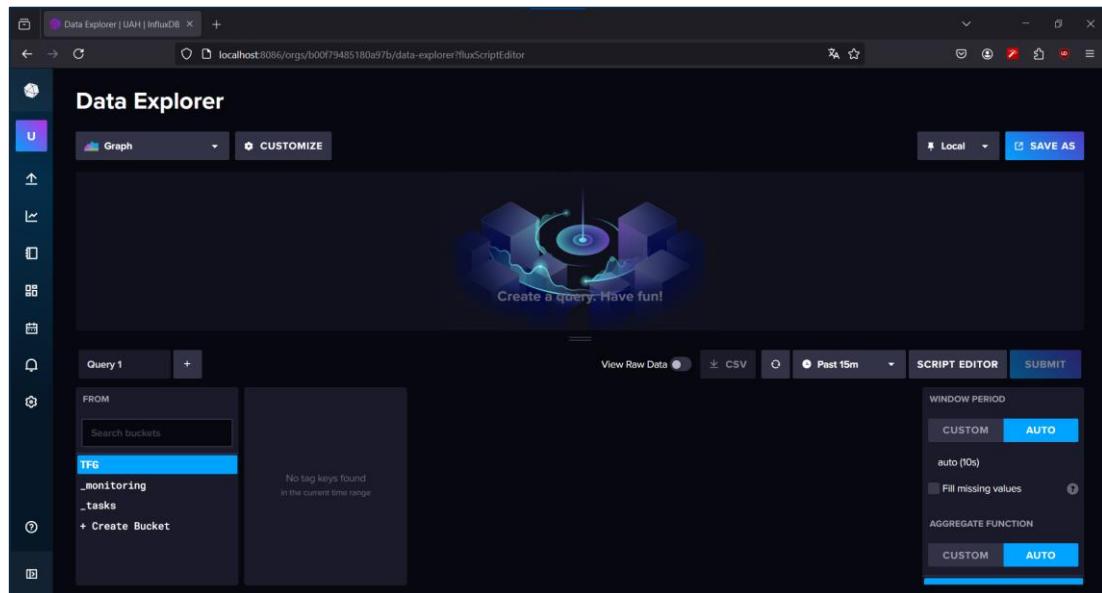


Figura 16: Pantalla Influxdb - Visualización de datos

En la sección *Data Explorer* del menú vertical izquierdo se podrán visualizar los datos mediante varias gráficas. Se pueden realizar diferentes configuraciones de filtros para controlar qué datos son mostrados.

Para realizar cualquier modificación sobre el gestor, como podría ser añadir usuarios, recuperar contraseñas y operar sobre los propios *Buckets* será necesario actuar a través de consola. Será necesario instalar **influx CLI**, siguiendo un proceso similar al de instalación de Influxdb:

- 1- Se descarga el paquete desde la página oficial: <https://docs.influxdata.com/influxdb/v2/tools/influx-cli/?t=Windows>.
- 2- Se expande y ubica mediante comandos:

```
>> Expand-Archive .\influxdb2-client-2.7.5-windows-amd64.zip -  
DestinationPath 'C:\Program Files\InfluxData\'  
  
>> mv 'C:\Program Files\InfluxData\influxdb2-client-2.7.5-  
windows-amd64' 'C:\Program Files\InfluxData\influx'
```

Cuando se configura por primera vez aparecerá un mensaje de Windows Defender con el siguiente contenido: *Windows Defender Firewall has blocked some features of this app*. Para asegurar el correcto funcionamiento del servicio debemos seleccionar la opción de “Redes privadas” y permitir el acceso.

La última parte necesaria para usar influx CLI de forma fluida es crear una configuración de la que se pueda extraer el *host*, el *API token* mencionado anteriormente, y la organización, y así no tener que incluirlo por cada comando utilizado. Para ello se utilizará el siguiente comando:

```
>> influx config create --config-name CONFIG_NAME \  
--host-url http://localhost:8086 \  
--org ORG\  
--token API_TOKEN \  
--active
```

Manteniendo la dirección y el puerto indicados en el host, pero modificando el nombre de la configuración por un nombre que la identifique, la organización y el *API token*. ORG se sustituye por UAH que es la organización definida durante la configuración de Influxdb para el proyecto. Por API\_TOKEN se debe sustituir una de las llaves registradas en el sistema. En función de la que se incluya en la configuración, influx CLI concederá diferentes permisos al usuario (los asociados a la llave utilizada).

### 3.2.3.2 Telegraf

Una vez se haya configurado el gestor Influxdb, y este sea plenamente funcional y esté activo, se podrán añadir diferentes **métodos de recolección de datos**. Uno de ellos es el agente de servidor **Telegraf**, que permite la recolección de datos de más de 150 fuentes diferentes. Para este sistema, se ha configurado Telegraf para que recoja **datos que llegarán mediante el protocolo MQTT**.

Para poder utilizar **Telegraf** lo primero es realizar la instalación. En este caso no hay que descargar ningún archivo manualmente, la descarga y la reubicación se realizan con los siguientes comandos:

```
>> wget https://dl.influxdata.com/telegraf/releases/telegraf-1.31.0_windows_amd64.zip -UseBasicParsing -OutFile telegraf-1.31.0_windows_amd64.zip  
>> Expand-Archive .\telegraf-1.31.0_windows_amd64.zip -DestinationPath 'C:\Program Files\InfluxData\telegraf'
```

La configuración comienza desde la interfaz de usuario de Influxdb en <http://localhost:8086>, donde en el menú vertical de la izquierda se accede al apartado *Load Data>Telegraf*. Una vez ahí, con el botón *Create Configuration* se inicia una nueva configuración. Lo primero es seleccionar el *Bucket* en el que se insertarán los datos recogidos.

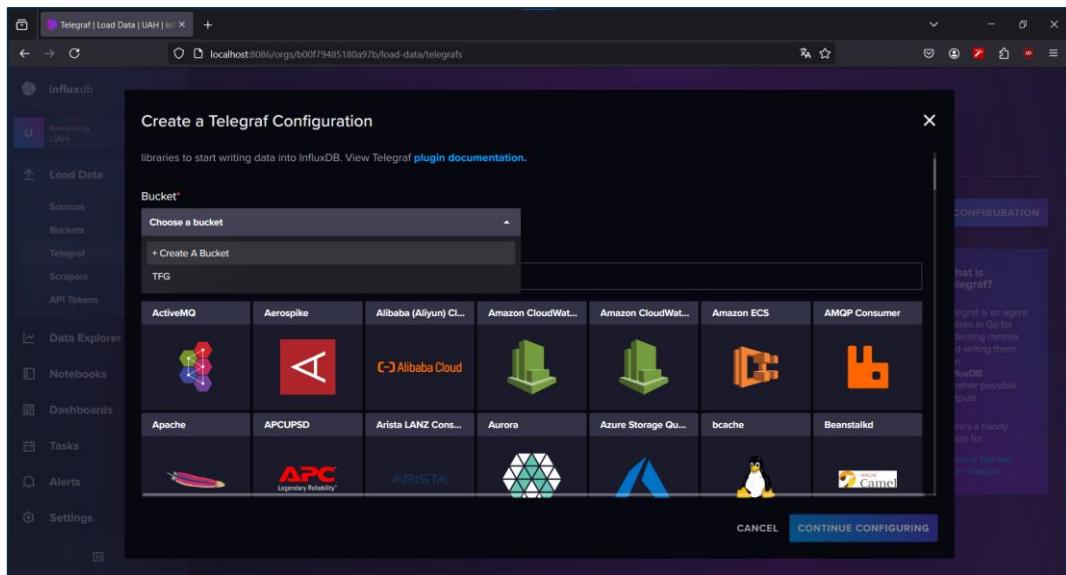


Figura 17: Pantalla Influxd - Configuración Telegraf – Bucket

Elegido el *Bucket*, se escoge el método de obtención de los datos, el *MQTT Consumer*, y se continua con la configuración.

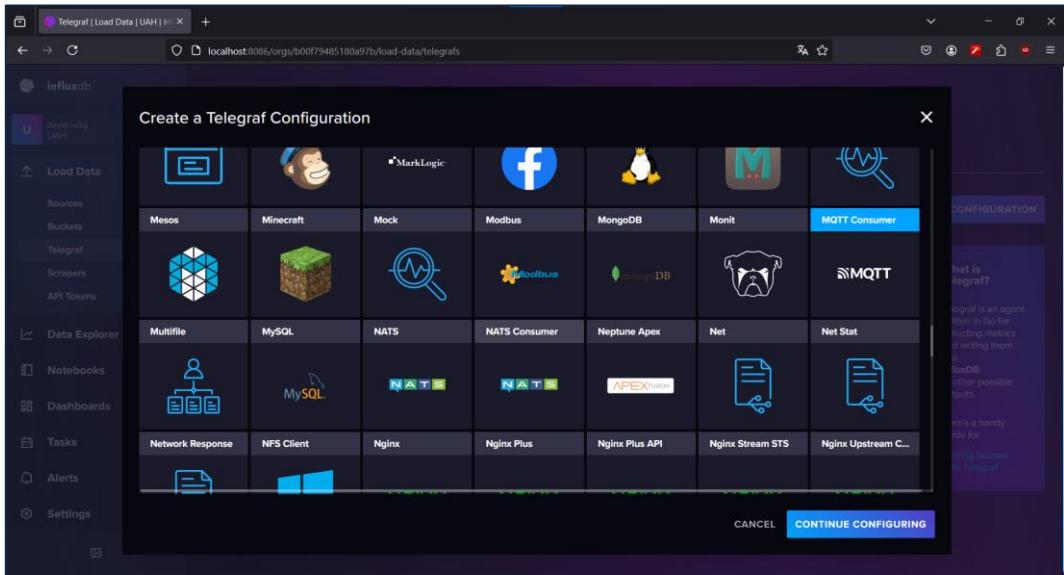


Figura 18: Pantalla Influxdb - Configuración Telegraf – Selección del origen de los datos

Para asegurar la correcta recepción de datos, dentro del archivo de texto con la configuración del agente se deberán realizar los siguientes cambios:

- Cambiar la dirección del servidor MQTT para ajustarse a la utilizada en el sistema.

```
servers = ["tcp://localhost:1883"]
```

Ya que el bróker MQTT reside en la misma máquina, aunque la configuración realizada permite también:

```
servers = ["tcp://192.168.216.210:1883"]
```

- Cambiar los tópicos a los que se suscribe el agente, para recibir datos cada vez que al bróker llegue un mensaje dirigido a esos tópicos:

```
topics = [
    "iotsys/#"
]
```

Pudiendo modificarse según los requerimientos específicos en cada situación, estos son los tópicos a los que se suscribe el sistema de almacenamiento de este proyecto.

- Se puede establecer también el QoS deseado para los mensajes:

```
qos = 1
```

- El último cambio necesario para asegurar la correcta entrada de datos es cambiar el formato y tipo de dato esperado:

```
data_format = "value"
data_type = "float"
```

Habiendo definido la configuración para Telegraf que se utilizará en el sistema, al confirmar aparecerán las instrucciones para terminar la configuración. Se mostrará un parámetro llamado INFLUX\_TOKEN que deberá utilizarse siempre que se inicie Telegraf. Para no tener que hacerlo, se podrá definir una variable de entorno que evite la referencia constante al parámetro. Con el siguiente comando, sustituyendo <INFLUX\_TOKEN> por su valor correspondiente se define la variable de entorno:

```
>> export INFLUX_TOKEN=<INFLUX_TOKEN>
```

Hecha la configuración, la forma de iniciar el agente para que comience la recolección de datos es a través de la consola del PC en la carpeta donde se han ubicado los archivos de Telegraf. Se utilizará el siguiente comando:

```
>> .\telegraf.exe --config
http://localhost:8086/api/v2/telegraf/0d0af2e8c0938000
```

### **3.2.4 Capa de aplicación**

Finalmente, tenemos la capa de aplicación. Es la más cercana al usuario final y tiene como función proporcionar los servicios necesarios para interactuar con el sistema, actuando como puente entre el usuario y las capas inferiores, sin otorgar acceso o visión de ellas. En esta capa se gestiona la lógica de negocio y se establecen las reglas específicas de la aplicación.

#### **3.2.4.1 Aplicación web**

Un sistema de monitorización necesita de una forma para poder visualizar los datos, ya sea para poder realizar un análisis sobre los valores sin procesar para generar datos significativos en un ámbito específico, o para tomar acción en respuesta a los valores mostrados. Además, si la solución también permite realizar estas acciones de respuesta, se obtienen varias ventajas:

- **Control remoto en tiempo real:** se permite al usuario responder a las condiciones del sistema sin requerir su presencia física.
- **Facilidad de gestión:** elimina la necesidad de tener diferentes aplicaciones para dos tareas estrechamente relacionadas, centralizando en una única interfaz ambas funcionalidades.

- **Retroalimentación inmediata:** los usuarios pueden ver los resultados de las acciones tomadas de forma instantánea en los continuos valores que recibe el sistema, pudiendo analizar si la acción tomada ha dado lugar a los resultados esperados.
- **Flexibilidad y escalabilidad:** estando ambas funciones unificadas, únicamente se necesita actualizar una aplicación, sin necesidad de rediseños en caso de querer incluir nuevos dispositivos en un futuro.
- **Mejora de la experiencia del usuario:** dando la oportunidad al usuario de interactuar con el sistema más allá de leer datos mostrados por una pantalla, permitiendo el control de las cargas, aumenta la satisfacción y el compromiso del usuario con el sistema.

Es por ello que, finalmente, la aplicación web combina ambas funcionalidades en un solo punto del sistema. De cara al usuario, esta decisión **no incrementa la complejidad** a la hora de usar la aplicación web, ya que ambas funciones son muy intuitivas de por sí, además de que la aplicación web contiene explicaciones muy claras sobre las posibilidades ofrecidas en cada vista.

## Estructura de la aplicación

La aplicación web sigue una arquitectura ***Single Page Application (SPA)***, esto quiere decir que ejecuta todo su contenido en una sola página, y los cambios se reflejan únicamente en los componentes modificados, sin necesidad de volver a renderizar la aplicación completa. Su estructura se enfoca en ser lo más sencilla posible, dividida en varios componentes, siguiendo el principio de modularidad. El motivo de buscar la sencillez reside en tratar de proporcionar el mejor rendimiento posible, además de un sistema que sea fácil de mantener y que en el futuro se pueda mejorar y escalar.

La aplicación, al igual que la funcionalidad del ESP32, ha sido programada al completo utilizando el entorno Visual Studio Code (VS Code) en la última versión disponible en junio de 2024. Además del entorno, ha sido necesario instalar Node.js junto con el gestor de paquetes npm, y se ha elegido<sup>11</sup> **Vite** como herramienta de compilación para el proyecto.

Con todo eso listo, crear el proyecto es tan simple como:

- 1- Abrir el entorno VS Code.
- 2- Abrir la carpeta donde se vaya a crear la carpeta raíz del proyecto.
- 3- Abrir un nuevo terminal dentro de la aplicación de VS Code.
- 4- Utilizar el siguiente comando: >> `npm create vite@latest`

Al introducir este comando, la consola pedirá un nombre para el proyecto (la opción predeterminada es *vite-project*). En este caso, el nombre elegido fue *Webapp*. A continuación, hay que decidir el *framework* o biblioteca que va a apoyar el desarrollo. Este proyecto ha sido creado con la biblioteca React. Para simplificar la escritura de código JavaScript, se utilizará JSX<sup>12</sup> que es el tipo de archivo JavaScript que se genera por defecto al usar Vite.

Con esto, automáticamente se definirá la **estructura básica del proyecto**, creando archivos y carpetas esenciales. Una vez ha finalizado este proceso, para instalar las dependencias y módulos

11. El uso de la palabra “elegido” en lugar de otra como “instalado” es debido a que para trabajar con Vite no es necesaria ninguna instalación adicional ya que lo gestiona npm durante la creación del proyecto.

12. JSX es una extensión de la sintaxis de JavaScript muy recomendada al usar React, por su notable simplificación de la complejidad y longitud del código. [29]

necesarios, dentro de la carpeta raíz del proyecto se deberá utilizar el comando: >> npm install. La estructura del proyecto quedará definida y posteriormente algunos de estos archivos serán modificados y se añadirán otros nuevos para cubrir las necesidades de la aplicación web.

Para realizar el análisis de la estructura final de la aplicación web, un buen punto de partida es la carpeta raíz de la aplicación. En ella encontramos elementos creados y definidos de forma automática por formar parte de la plantilla predeterminada del proyecto, y elementos modificados o creados por el desarrollador para la aplicación web.

Para ver el contenido de la carpeta utilizando la consola del PC primero nos desplazamos a la carpeta:

```
>> cd C:\projects\TFG\Webapp
```

Luego utilizando el comando **dir** se muestra, con estructura de árbol, los archivos y las subcarpetas contenidas en la raíz del proyecto:

```
>> dir

    Directorio: C:\projects\TFG\Webapp

Mode                 LastWriteTime       Length Name
----                 -----          -----
d-----               16/06/2024        18:22   node_modules
d-----               16/06/2024        18:53   src
-a----   14/06/2024        17:25      253  .gitignore
-a----   14/06/2024        17:25      389  index.html
-a----   14/06/2024        17:25    153685 package-
lock.json
-a----   14/06/2024        17:25      772  package.json
-a----   16/06/2024        19:34      212  vite.config.js
-a----   14/06/2024        17:25      666  websocket-
server.js
```

Puede verse que el contenido de la carpeta raíz son dos subcarpetas y seis archivos.

Comenzando por los archivos individuales, el archivo **.gitignore** es un archivo que se genera al utilizar repositorios de Github como se ha hecho en este proyecto, y su función es definir que archivos y directorios deben ser ignorados por Git. Su relevancia para la aplicación se limita a la subida y descarga de versiones al repositorio donde se almacena todo el código del TFG.

El siguiente archivo que aparece es **index.html**, este archivo es el punto de entrada principal de la aplicación web. Utiliza etiquetas HTML básicas para definir la estructura de la página y el punto de montaje para la aplicación React. Referencia al archivo **main.jsx** (que se analizará posteriormente) que inicia la aplicación.

Luego encontramos **package-lock.json**. Se trata de un archivo generado automáticamente al instalar las dependencias del proyecto. Asegura que las mismas versiones de los paquetes se instalen en todas las máquinas, proporcionando un entorno consistente para el desarrollo.

A este le sigue **package.json**, es el “corazón” del proyecto. En él se definen las dependencias del proyecto, los scripts de npm que pueden ejecutarse y otras configuraciones esenciales como nombre y versión del proyecto. Dentro de este archivo, destacan los scripts “dev” y “start:ws”.

- El script “dev” es el que da la orden de arranque a la aplicación web, el comando completo sería: >> npm run dev
- El script “start:ws” por otro lado, da la orden de arranque para el servidor de Websockets, este hace referencia al archivo `websocket-server.js`, que contiene las instrucciones de arranque del servidor. El comando completo sería: >> npm run start:ws

Estos comandos deben ejecutarse para poder usar el sistema en caso de que aplicación y servidor no estuvieran activos.

Por último, en cuanto a archivos se refiere, en la raíz encontramos dos archivos con extensión .js o JavaScript. El primero, **vite.config.js**, contiene la configuración usada por Vite, como el puerto donde se aloja la aplicación o la configuración de acceso para otros dispositivos en la misma red. El puerto designado por defecto se mantiene para este proyecto, es decir, el 5173, y se ha establecido la configuración para que otros dispositivos conectados a la misma red Wi-Fi puedan acceder a la aplicación web. El otro, **websocket-server.js**, mencionado previamente, es el archivo que define la configuración y el comportamiento del servidor de Websockets. Se describirá en detalle en la sección dedicada a la comunicación mediante Websockets.

Una vez analizada la función y el contenido de los archivos individuales del directorio raíz, los otros elementos importantes son las subcarpetas.

La primera de ellas es la carpeta **node\_modules**, esta se genera de forma automática al instalar las dependencias del proyecto. En ella se encuentran las dependencias, módulos, librerías y herramientas esenciales para el funcionamiento de la aplicación. Para este proyecto no ha sido necesario añadir ningún elemento adicional que no incluyese al acabar la instalación.

La segunda carpeta es **src**, que contiene el código fuente que da funcionalidad y forma a la aplicación web. Dentro encontramos: 9 archivos .jsx, 7 archivos .css y una carpeta (assets).

La carpeta **assets** contiene diferentes archivos estáticos que no se procesan antes de ser usados en la aplicación web. Esto incluye todas las imágenes que aparecen en la aplicación, el ícono mostrado en el navegador o el diagrama de una de las secciones.

Los archivos contenidos en la carpeta con extensión .css sirven para definir los estilos que se aplican sobre cada elemento de la aplicación, es decir, definen la apariencia final que el usuario puede ver. Cada archivo .css da forma al estilo del componente .jsx con el mismo nombre, es decir, los estilos para `Panel.jsx` están contenidos en `Panel.css`. Estos estilos establecen características como la ubicación de cada elemento, el margen que existe entre elementos, las dimensiones y la alineación de estructuras HTML. También controlan el comportamiento visual frente a eventos dinámicos como: el formato adquiere un botón al pulsarse, la reorganización de los componentes de la aplicación al ocultar un menú, la posición de los elementos al deslizar verticalmente o el tamaño y posición al agrandar o reducir la aplicación. El efecto de los estilos

definidos en CSS sobre la aplicación es meramente estético. Su propósito es hacer la UI más visual, atractiva e intuitiva para el usuario final. La funcionalidad la brindan los archivos .jsx.

Por último, tenemos los nueve archivos .jsx. Estos contienen el código que define la estructura y la lógica de la aplicación web. Cada uno importa los estilos CSS correspondientes a los componentes renderizados en el propio archivo.

- **main.jsx:** es el primer archivo en ser referenciado (al iniciar la aplicación, por index.html). Como ya se comentó, es el punto de partida de la aplicación web. Contiene dos de las líneas de código más importantes de toda la aplicación.

```
import ReactDOM from 'react-dom/client'
```

Importa el módulo que contiene los métodos que permiten interactuar con el DOM, sin esto no se podrían crear interfaces dinámicas y reactivas.

```
const root =
ReactDOM.createRoot(document.getElementById('root'))
```

Crea la raíz de React, un elemento que actúa como contenedor, y es donde se renderiza la aplicación web.

Finalmente, se encarga de renderizar el componente App dentro de la raíz, dando inicio al proceso de renderizado, y creando y montando el árbol de componentes con el que trabaja el DOM.

Nota: Para renderizar un componente JSX o un elemento o contenedor HTML, se debe hacer uso de la/s correspondiente/s etiqueta/s, pasando cualquier característica (*prop*) necesaria.

```
// Componente sin prop
<App/>

// Componente con prop
<Panel currentView={currentView} />

// Contenedor
<div>[Contenido]</div>
```

- **App.jsx:** es el componente principal de la aplicación, actúa como raíz del árbol de componentes. Tiene la función de gestionar las vistas mostradas en el panel central (Panel.jsx) y controlar la visibilidad del menú vertical (VertMenu.jsx).

Hace uso de *useState*, un *hook*<sup>13</sup> de React que permite el almacenamiento de estados. Un estado de *useState* se define de la siguiente forma:

```
const [currentView, setCurrentView] = useState('Inicio');
```

Se define la variable donde se almacena el estado (*currentView*), la función para cambiar el valor (*setCurrentView*) y el estado por defecto en el primer renderizado ('*Inicio*'). Este recurso se utiliza a lo largo de toda la aplicación para gestionar estados que son modificables por el usuario o como resultado de algún proceso.

Dentro de *App.jsx* se manejan dos estados, *currentView* que ha servido como ejemplo del uso de *useState* y el estado *isShown* con su correspondiente *setIsShown*.

- El primero almacena que vista se está mostrando en el panel central, entre 'Inicio', 'Datos', 'Graficas', 'Control Cargas', 'Sobre el proyecto' y 'Sobre mi', y siendo 'Inicio' el valor por defecto. Se pasa como característica de *Panel* para poder realizar los cambios.
- El segundo almacena si el menú vertical es visible o no. Se pasa como característica de *VertMenu* para realizar los cambios.

Además de usar los *hooks* de tipo *useState*, utiliza manejadores de eventos que dictan el comportamiento de la aplicación en caso de que suceda un evento esperado. Los manejadores de eventos se definen de la siguiente forma:

```
const handleMenuItemClick = (view) => {
  console.log(`Changing view to: ${view}`);
  setCurrentView(view);
}
```

Se le da un nombre relacionado con el evento, se definen los posibles parámetros que pueden pasarse y finalmente se decide el resultado esperado si sucede el evento.

En *App.jsx* se definen dos manejadores de eventos, el mencionado *handleMenuItemClick* del ejemplo, y *handleButtonClick*. El primero se pasa como característica de *VertMenu* y detecta cuando el usuario ha pulsado alguno de los botones que representan una sección de la aplicación, para cambiar el estado de la vista actual al que corresponde a la nueva sección a visitar. El segundo se pasa como característica de *MenuButton*, con el propósito de detectar cuando se pulsa el botón que cierra y abre el menú, cambiando las dimensiones del menú y del panel en consecuencia.

*App.jsx* se encarga de renderizar los tres componentes mencionados *VertMenu*, *Panel* y *MenuButton*, cada uno definido en un archivo con mismo nombre y extensión *.jsx*, dentro de la carpeta *src*.

- **VertMenu.jsx:** este componente representa el menú vertical de la aplicación que permite alternar las diferentes vistas. Para ello se utilizan los botones que contiene. Se le pasan dos características, *onClick* y *menuView*. La primera referencia al manejador de eventos *handleMenuItemClick* definido en *App.jsx* y se establece como respuesta al evento de clic de ratón sobre los botones que corresponden a secciones de la aplicación. La segunda indica el estado del menú, abatido o desplegado.

Este componente no cuenta con ningún almacenamiento de estado ni define manejadores de eventos. Su función es renderizar una imagen (logo), y un panel que consta de tres botones, con textos, 'Inicio', 'Servicios' e 'Información'. 'Inicio' cambia el estado de vista mostrada a 'Inicio', mientras que los otros dos no realizan función al

13. Un *hook* es una función especial de JavaScript que permite al desarrollador reutilizar ciertas características del ciclo de vida de los componentes de React. [27]

pulsarse. Estos sirven como título de secciones otros botones, ocultos en primera instancia, y que aparecen al colocar el ratón sobre cada título. En la sección ‘Servicios’ se despliegan ‘Datos’, ‘Graficas’ y ‘Control Cargas’. Cada uno cambia la vista actual a la sección con el mismo nombre del botón. Y de la sección ‘Informacion’, surgen ‘Sobre el proyecto’ y ‘Sobre mi’, que siguen el mismo principio.

- **MenuButton.jsx:** este componente representa el botón que de inicio tiene forma de flecha “<”. Se encarga de ocultar y mostrar el menú vertical. Tiene una única *prop*, *onClick*, que consiste en una referencia a *handleButtonClick* definida en *App.jsx*.

En este componente se define un *useState* y un manejador de eventos, además de dos variables que permiten el cambio en los estilos del botón

- *isShown* y *setIsShown*: almacenamiento y modificación del estado que recoge si el botón adopta la forma para cerrar el menú o la forma para abrir el menú.
  - *handleButtonClick*: manejador de eventos que modifica el estado de *isShown* y que activa la reacción del manejador de eventos correspondiente a la referencia *onClick*.
  - *buttonView* y *imageSrc*: variables que cambian su valor y que establecen el estilo que adopta el botón según se haya cerrado o abierto el menú.
- **Panel.jsx:** este componente representa el panel central de la aplicación, que contiene y muestra las diferentes vistas que forman la interfaz de usuario de la aplicación web. En este componente se utilizan tres tipos de *hook* distintos, dos nativos de React (*useState* y *useEffect*) y uno personalizado (*useWebSocket*).

En primer lugar, el ya mencionado *useState* para almacenar estados. De este tipo podemos encontrar cinco estados. El primero de ellos, *tableData* representa la tabla en la que se almacenan los datos recibidos de mediciones. Las filas de este estado se muestran en forma de tabla en la sección ‘Datos’, mientras que cada columna que corresponda a un parámetro eléctrico es representada como una gráfica en la vista ‘Graficas’, donde cada nueva medición se incluye como un nodo. Los otros cuatro estados *relayXState* (donde X se reemplaza por números entre 0 y 3, uno en referencia a cada botón) corresponden al estado de los relés. Si el relé se encuentra en estado apagado, el botón de ‘Control Cargas’ que lo gestiona mostrará el texto ‘Encender’, y cuando el relé está encendido, el texto ‘Apagar’.

El segundo *hook* es el *useEffect*. Sirve para permitir a componentes funcionales ejecutar efectos secundarios. Permite realizar tareas como solicitudes de datos, suscripciones, actualizaciones manuales o llamadas a una API [27]. Se estructura de la siguiente forma:

```
useEffect(() => { // Declaración
  // Inicio de la definición de efectos secundarios
  if (receivedData === null) {
    return;
  }
  if (receivedData.Type === 'data') {
    setTableData((prevData) => [...prevData, receivedData]);
  }
})
```

```

    }
    else if (receivedData.Type === 'state') {
        if (receivedData.State === 0) {
            (receivedData.Relay === 0) ? setRelay0State("control-
button off") :
                (receivedData.Relay === 1) ? setRelay1State("control-
button off") :
                    (receivedData.Relay === 2) ? setRelay2State("control-
button off") :
                        (receivedData.Relay === 3) ? setRelay3State("control-
button off") : console.log('Error: Relay not found.');
        }
        else if (receivedData.State === 1) {
            (receivedData.Relay === 0) ? setRelay0State("control-
button on") :
                (receivedData.Relay === 1) ? setRelay1State("control-
button on") :
                    (receivedData.Relay === 2) ? setRelay2State("control-
button on") :
                        (receivedData.Relay === 3) ? setRelay3State("control-
button on") : console.log('Error: Relay not found.');
        }
        else {
            console.log('Error: State not found.');
        }
    }
    // Fin de definición de efectos secundarios
    return () => {
        // Limpieza del efecto secundario (Opcional)
    };
},
[receivedData]); // Dependencia: si no hay se ejecuta
siempre, si [] sólo una vez, y si se añade dependencia cuando
cambie su valor

```

El *useEffect* mostrado como ejemplo es el que se puede encontrar definido en *Panel.jsx*, el efecto secundario que permite es, cuando cambia la dependencia (*receivedData*), si el mensaje tiene como tipo ‘data’, es decir, son datos, se añaden a la tabla, mientras que, si tiene valor ‘state’, representa el estado adquirido por un relé, y se debe cambiar el estado del botón que lo controla para mantener la consistencia y que se correspondan los estados de ambos.

El último tipo de *hook* se trata de uno personalizado, sigue los mismos principios que un *hook* de React además de que suele hacer uso de los estos últimos. Este en concreto, ha sido diseñado para permitir la conexión a un servidor de Websockets, pasando como parámetro su URL, y recibir mensajes mediante Websockets del servidor y como gestionarlos.

Además de estos recursos, se definen dos funciones. Una función (*sendMessage*) para enviar mensajes de tipo ‘order’ para apagar o encender la carga controlada por un relé. Y otra función cuyo propósito es renderizar una vista de las cinco descritas anteriormente, en función del valor de *currentView* pasado como característica al volver a renderizar su

componente padre. Aparte de texto, las vistas que renderizan algún elemento o componente destacable son:

- La vista ‘Datos’: renderiza una tabla con los datos de las mediciones.
  - La vista ‘Gráficas’: renderiza seis componentes *Graph*, definidos en *Graph.jsx*, que representan gráficas donde los nodos son cada uno de los valores ordenados cronológicamente para cada parámetro eléctrico.
  - La vista ‘Control’: renderiza cuatro botones, uno por relé, ajustando el estilo y acción del botón en función del estado real del relé.
  - La vista ‘Sobre mi’: renderiza un botón que descarga un *Curriculum Vitae* (CV) del autor del proyecto, y un panel horizontal que contiene referencias a redes sociales y métodos de contacto con el autor del proyecto, se encuentra situado en la parte inferior.
- **HorizMenu.jsx**: este componente representa el mencionado panel mostrado en ‘Sobre mi’ que muestra las redes sociales del autor del proyecto. En este componente se renderizan diferentes fragmentos de texto e imágenes.
  - **DownloadButton.jsx**: este componente corresponde a un botón que aparece en la sección ‘Sobre mi’ desde el cual puede descargarse un CV. En él se define un manejador de eventos (*handleCVClick*) que se activa cuando se pulsa el botón. Al activarse crea una referencia al archivo importado de la carpeta *assets* (el CV) y ejecuta la acción de descarga del elemento referenciado.
  - **useWebSocket.jsx**: se trata de un *hook* personalizado que pretende permitir la conexión a un servidor de Websockets, para la recepción y el envío de mensajes.

Se utiliza *useState* para definir dos estados. El estado *socket* como su nombre indica, se actualiza con el socket, si este está abierto y conectado, o si se ha cerrado la conexión. El otro estado es *receivedData*, empleado para almacenar el último dato de mediciones recibido en cada momento.

También se utiliza un *useEffect* que define los efectos secundarios cuando:

- Se establece conexión con el servidor: se muestra un mensaje por pantalla.
- Llega un mensaje del servidor: se muestra el mensaje por pantalla, se convierte el mensaje a JSON y se actualiza el valor del último dato recibido (*receivedData*).
- En caso de que se cierre la conexión se muestra por pantalla un mensaje indicando si ha sido un cierre limpio o debido a un error.
- En caso de error: se muestra un mensaje por pantalla indicando el error.

Al finalizar, se retorna el valor de *socket* y de *receivedData* como limpieza de efectos secundarios. De este modo, el socket se transmite a la clase padre permitiendo definir la función de envío de mensajes, y los datos recibidos se insertan en la tabla o actualizan el estado de los botones.

La dependencia que provoca que se ejecuten de nuevo los efectos secundarios es la dirección (URL) del servidor, en caso de cambiar se ejecutará de nuevo el *hook*.

- **Graph.jsx:** el último componente que pertenece a la carpeta `src`, representa una gráfica de líneas que puede ser de distintos tipos con distintos estilos. Tiene dos características, que son la tabla con los datos de las mediciones (`tableData`) y el tipo (`type`). El tipo permite distinguir sobre qué parámetro debe generarse la gráfica, y mediante una serie de condicionales que comparan los parámetros disponibles con el tipo recibido, se establecen las características de la tabla:
  - ‘data’: los datos mostrados en la tabla, según el tipo, se tomarán de diferentes columnas.
  - ‘borderColor’ y ‘backGroundColor’: colores de la línea y del relleno de los nodos, cambian por cada tipo para facilitar la distinción de un parámetro y otro de forma visual.

Con estos parámetros, y la hora de la medición como valor de la variable independiente (eje horizontal), se establecen los datos de la tabla.

Con la variable `options` se puede configurar el comportamiento y el tipo de la gráfica. De este modo, se establece :

- Que al colocar el ratón sobre un nodo se mostrará un cuadro con el valor de la medición y el parámetro en cuestión,
- el tamaño y la fuente de la letra,
- el texto de los títulos de los ejes y la gráfica completa, y su alineación.

Una vez establecido todo, este componente renderiza un componente importado de la biblioteca ‘react-chartsjs-2’ llamado *Line*, que se corresponde con el gráfico de líneas deseado.



# Capítulo 4: Pruebas y resultados

En esta sección se mostrarán los resultados obtenidos al realizar pruebas sobre las diferentes opciones y funcionalidades que ofrece el sistema. Para cada prueba se detallarán las condiciones del sistema al momento de iniciar la prueba y el resultado que se espera obtener siguiendo la teoría y el flujo de los procesos (detallados en sus respectivas capas). También se incluirá el resultado real obtenido permitiendo comprobar si se corresponde con lo esperado.

## 4.1 Conexión Wi-Fi

### Condiciones iniciales:

- El ESP32 debe estar conectado a una fuente de alimentación y con el programa cargado.

<b>Resultado esperado</b>	Si la configuración se ha realizado correctamente y se ha establecido la conexión, deberá mostrarse por consola un mensaje indicando la conexión. Indicando la dirección IP que ha sido asignada al dispositivo.
<b>Resultado obtenido</b>	<p>Inicio conexión:</p> <pre>I (840) WIFI_CONNECT: WIFI_EVENT_STA_START I (970) wifi:new:&lt;1,0&gt;, old:&lt;1,0&gt;, ap:&lt;255,255&gt;, sta:&lt;1,0&gt;, prof:1 I (1220) wifi:state: init -&gt; auth (b0) I (1230) wifi:state: auth -&gt; assoc (0) I (1260) wifi:state: assoc -&gt; run (10) I (1290) wifi:connected with Galaxy A715685, aid = 16, channel 1, BW20, bssid = 1e:8c:23:dc: I (1290) wifi:security: WPA2-PSK, phy: bgn, rssi: -34 I (1290) wifi:pm start, type: 1</pre> <p>Conexión exitosa y dirección IP:</p> <pre>I (1300) WIFI_CONNECT: WIFI_EVENT_STA_CONNECTED I (1320) wifi:dp: 2, bi: 102400, li: 4, scale listen interval from 307200 us to 409600 I (1320) wifi:AP's beacon interval = 102400 us, DTIM period = 2 I (2310) WIFI_CONNECT: IP_EVENT_STA_GOT_IP I (2310) esp_netif_handlers: sta ip: 192.168.216.101, mask: 255.255.255.0, gw: 192.168</pre>

Tabla 3: Pruebas Wi-Fi - Resultados conexión

## 4.2 Conexión SNTP y establecimiento de fecha y hora

### Condiciones iniciales:

- El ESP32 debe estar conectado a una fuente de alimentación y con el programa cargado. Debe estar conectado a una red Wi-Fi.

<b>Resultado esperado</b>	Si la configuración se ha realizado correctamente y se ha establecido la conexión, deberá mostrarse por consola un mensaje indicando la conexión.
<b>Resultado obtenido</b>	Conexión exitosa: <pre>I (2310) SNTP: Iniciando configuración SNTP... I (2320) SNTP: Configuración SNTP completada.</pre>

Tabla 4: Pruebas SNTP - Resultados conexión

## 4.3 Recogida y recepción de mediciones

### Condiciones iniciales:

- El ESP32 debe estar conectado a una fuente de alimentación y a una red Wi-Fi, con el programa cargado y conectado al kit de medición.
- El sensor debe estar conectado a la bobina, a una fuente de alimentación y al tramo de circuito estudiado. Se encenderá un led cuando este alimentado.
- La bobina debe rodear uno de los dos cables (Carga *L* o Neutro *N*) del circuito. Finalmente, el circuito debe estar conectado a una toma de pared.

<b>Resultado esperado</b>	Si las mediciones se han tomado y enviado correctamente, se espera que los valores recogidos se muestren por consola.
<b>Resultado obtenido</b>	Configuración exitosa y recepción de datos: <code>I (72773) MAIN: Leyendo datos del PZEM004T mediante UART... I (73833) MAIN: Recibido: 25 bytes Voltage: 235.60 Current: 0.00 Power: 0.00 Energy: 0.06 Frequency: 50.00 Power Factor: 0.00 Alarms: 0</code>

Tabla 5: Pruebas Sensor - Resultados recepción de mediciones

## 4.4 Comunicación mediante MQTT

### 4.4.1 Conexión del ESP32 con el bróker

### Condiciones iniciales:

- El ESP32 debe estar conectado a una fuente de alimentación y a una red Wi-Fi y con el programa cargado.

<b>Resultado esperado</b>	Si la configuración se ha realizado correctamente y se ha establecido la conexión, deberá mostrarse por consola un mensaje indicando la conexión.
<b>Resultado obtenido</b>	Conexión exitosa. <code>I (5283) main_task: Returned from app_main() I (5293) wifi:&lt;ba-add&gt;idx:0 (ifx:0, 1e:8c:23:dc:b5:c1), tid:0, ssn:3, winSize:64 I (5323) MQTT: MQTT_EVENT_CONNECTED</code>

Tabla 6: Pruebas MQTT - Resultados conexión ESP32 con bróker

#### **4.4.2 Conexión de Telegraf con el bróker**

**Condiciones iniciales:**

- Influxdb tiene que estar iniciado.
- Telegraf tiene que estar iniciado y debe estar conectado a la misma red que el bróker.

<b>Resultado esperado</b>	Si se ha establecido la conexión, deberá mostrarse por consola un mensaje indicando la conexión.
<b>Resultado obtenido</b>	Conexión establecida correctamente: 2024-06-20T10:16:42Z ! [agent] config: Interval:10s, Quiet:false, Hostname:"DESKTOP-02PTAJJ", Flush Interval:10s 2024-06-20T10:16:42Z ! [inputs.mqtt_consumer] Connected [tcp://localhost:1883]

Tabla 7: Pruebas MQTT - Resultados conexión Telegraf con bróker

#### **4.4.3 Envío de mensaje MQTT desde el ESP32**

**Condiciones iniciales:**

- El ESP32 debe estar conectado a una fuente de alimentación y a una red Wi-Fi y con el programa cargado. Tendrá que estar recibiendo datos que poder enviar, y estará conectado al bróker.

<b>Resultado esperado</b>	Aparecerá un mensaje por consola indicando el envío de cada mensaje. Si el envío se ha realizado correctamente, se tendrá que haber añadido un nuevo tópico por cada mensaje recibido (uno por parámetro eléctrico).
<b>Resultado obtenido</b>	Consola ESP32: <i>I (6354) MQTT: Enviando valores mediante MQTT</i> <i>I (6354) MQTT: Enviando valores de Potencia mediante MQTT</i> <i>I (6364) MQTT: Enviando valores de Voltaje mediante MQTT</i> <i>I (6374) MQTT: Enviando valores de Intensidad mediante MQTT</i> <i>I (6374) MQTT: Enviando valores de Energia mediante MQTT</i> <i>I (6384) MQTT: Enviando valores de Frecuencia mediante MQTT</i> <i>I (6384) MQTT: Enviando valores de Factor de Potencia mediante MQTT</i> Árbol de tópicos (MQTT Explorer):

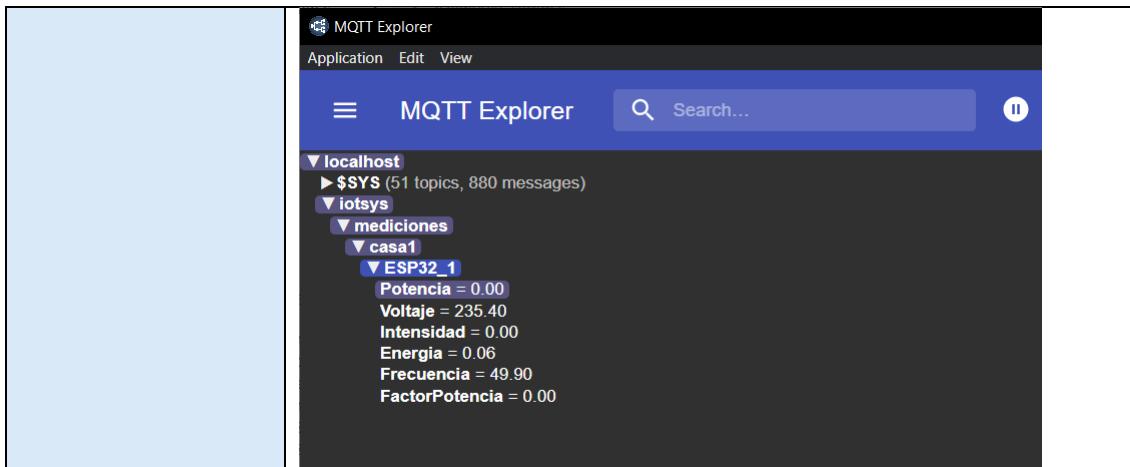


Tabla 8: Pruebas MQTT - Resultados envío de mensaje desde ESP32

#### 4.4.4 Recepción de mensaje MQTT en Telegraf

##### Condiciones iniciales:

- El ESP32 debe estar conectado a una fuente de alimentación y a una red Wi-Fi y con el programa cargado. Tendrá que estar recibiendo datos que poder enviar, y estará conectado al bróker.
- Influxdb tiene que estar iniciado.
- Telegraf tiene que estar iniciado y debe estar conectado al bróker, y en su configuración haberse suscrito al tópico correspondiente (iotsys/#).
- Los mensajes del ESP32 deben haber llegado correctamente al bróker.

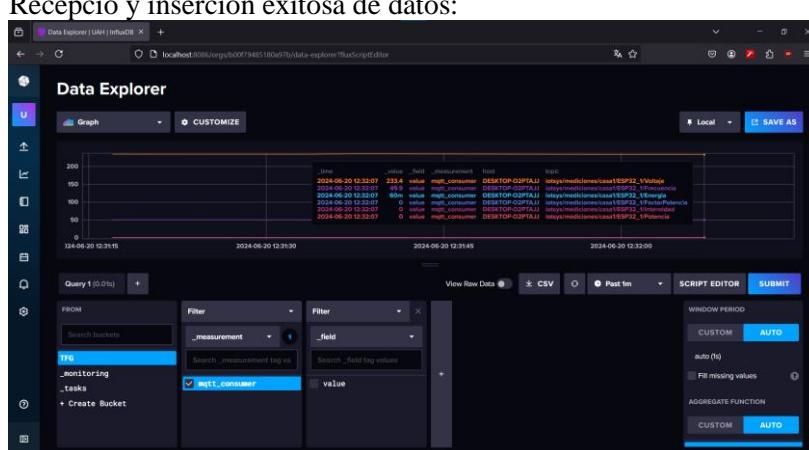
<b>Resultado esperado</b>	Si los mensajes han sido recibidos correctamente por Telegraf, los valores serán insertados y serán introducidos en el correspondiente <i>bucket</i> (TFG) y serán visibles desde la UI en <a href="http://192.168.216.210:8086">http://192.168.216.210:8086</a> accediendo con credenciales.
<b>Resultado obtenido</b>	Recepción y inserción exitosa de datos: 

Tabla 9: Pruebas MQTT - Resultados recepción de mensaje en base de datos

## 4.5 Comunicación mediante Websockets

### 4.5.1 Conexión del ESP32 con el servidor

**Condiciones iniciales:**

- El ESP32 debe estar conectado a una fuente de alimentación y a una red Wi-Fi y con el programa cargado.
- El servidor debe estar inicializado, esperando conexiones.

<b>Resultado esperado</b>	Si la configuración se ha realizado correctamente y se ha establecido la conexión, deberá mostrarse por consola un mensaje indicando la conexión y deberá aparecer una nueva conexión en el servidor.
<b>Resultado obtenido</b>	Consola ESP32: <pre>I (4078) wifi:&lt;ba-add&gt;idx:0 (ifx:0, 1e:8c:23:dc:b5:c1), tid:0, ssn:3, winSize:64 I (5288) main_task: Returned from app_main() I (5668) WEBSOCKET: WEBSOCKET_EVENT_CONNECTED</pre> Servidor: <pre>&gt; tfg@0.0.0 start:ws &gt; node websocket-server.js  WebSocket server is running on ws://localhost:8080 Client connected ■</pre>

Tabla 10: Pruebas Websockets - Resultados conexión entre ESP32 y servidor

### 4.5.2 Conexión de la aplicación web con el servidor

**Condiciones iniciales:**

- La aplicación web debe estar iniciada y debe tener acceso a la misma red en la que se encuentra el servidor.
- El servidor debe estar inicializado, esperando conexiones.

<b>Resultado esperado</b>	Si la conexión se ha realizado correctamente, en la consola de la aplicación se mostrará un mensaje que dice “Websocket connection opened” y en el servidor aparecerá una nueva conexión.
<b>Resultado obtenido</b>	Consola aplicación web:

The screenshot shows a browser's developer tools console with the 'Console' tab selected. The log output is as follows:

```
[vite] connecting...
[vite] connected.
ⓘ Download the React DevTools for a better development experience
Ruta imagen botón: /src/assets/hideMenu.png
Rendering content for: Inicio
WebSocket connection opened
```

**Servidor:**

```
✖ npm run start:ws
> tfg@0.0.0 start:ws
> node websocket-server.js
WebSocket server is running on ws://localhost:8080
Client connected
```

Tabla 11: Pruebas Websockets - Resultados conexión entre aplicación web y servidor

#### 4.5.3 Envío de mensaje desde el ESP32

##### Condiciones iniciales:

- El ESP32 debe estar conectado a una fuente de alimentación y a una red Wi-Fi y con el programa cargado. Debe estar recibiendo datos y conectado al servidor.
- El servidor debe estar inicializado, y la conexión con el ESP32 debe haber sido registrada.

<b>Resultado esperado</b>	Si el mensaje se ha creado y enviado correctamente se mostrará el contenido del mensaje en la consola del ESP32, y si se recibe correctamente aparecerá en la consola del servidor, y será reenviado a los clientes conectados.
<b>Resultado obtenido</b>	<p>Consola ESP32:</p> <pre>Voltage: 234.90 Current: 0.00 Power: 0.00 Energy: 0.06 Frequency: 50.00 Power Factor: 0.00 Alarms: 0 1 (337860) WEBSOCKET: Enviando mensaje mediante websockets: {"Type": "data", "ID": "ESP32_1", "Potencia": 0.00, "Voltaje": 234.90, "Intensidad": 0.00, "Energia": 0.06, "Frecuencia": 50.00, "FactorPotencia": 0.00, "Fecha": "20-06-2024", "Hora": "12:57:39"}</pre> <p>Servidor:</p> <pre>Received: {"Type": "data", "ID": "ESP32_1", "Potencia": 0.00, "Voltaje": 234.90, "Intensidad": 0.00, "Energia": 0.06, "Frecuencia": 50.00, "FactorPotencia": 0.00, "Fecha": "20-06-2024", "Hora": "12:57:39"}</pre>

Tabla 12: Pruebas Websockets - Resultados envío de mensajes desde ESP32

#### 4.5.4 Recepción de mensaje en la aplicación web

##### Condiciones iniciales:

- La aplicación web está iniciada y conectada al servidor.

- El servidor debe estar inicializado, y la conexión con la aplicación web debe haber sido registrada. Debe haber recibido o estar recibiendo mensajes desde el ESP32.

<b>Resultado esperado</b>	Si el mensaje ha llegado al servidor, este es reenviado a los clientes conectados. Si la aplicación web recibe el mensaje, se mostrará el contenido por consola y se actualizarán los valores de la tabla y de las gráficas.																																																																																										
<b>Resultado obtenido</b>	<p>Consola aplicación web:</p> <pre>Message from server: {"Type": "data", "ID": "ESP32_1", "Potencia": 0.00, "Voltaje": 236.20, "Intensidad": 0.00, "Frecuencia": 49.9} Rendering content for: Inicio Message from server: {"Type": "data", "ID": "ESP32_1", "Potencia": 0.00, "Voltaje": 236.30, "Intensidad": 0.00, "Frecuencia": 49.9} Rendering content for: Inicio Message from server: {"Type": "data", "ID": "ESP32_1", "Potencia": 0.00, "Voltaje": 236.50, "Intensidad": 0.00, "Frecuencia": 49.9} Rendering content for: Inicio</pre> <p>Tabla aplicación web:</p> <table border="1"> <thead> <tr> <th>ID</th> <th>Potencia (W)</th> <th>Voltaje (V)</th> <th>Intensidad (A)</th> <th>Energía</th> <th>Frecuencia</th> <th>Factor de potencia</th> <th>Fecha</th> <th>Hora</th> </tr> </thead> <tbody> <tr><td>ESP32_1</td><td>0</td><td>235.7</td><td>0</td><td>0.06</td><td>49.9</td><td>0</td><td>20-06-2024</td><td>13:03:42</td></tr> <tr><td>ESP32_1</td><td>0</td><td>235.9</td><td>0</td><td>0.06</td><td>49.9</td><td>0</td><td>20-06-2024</td><td>13:03:45</td></tr> <tr><td>ESP32_1</td><td>0</td><td>236.2</td><td>0</td><td>0.06</td><td>50</td><td>0</td><td>20-06-2024</td><td>13:03:48</td></tr> <tr><td>ESP32_1</td><td>0</td><td>236.5</td><td>0</td><td>0.06</td><td>50</td><td>0</td><td>20-06-2024</td><td>13:03:52</td></tr> <tr><td>ESP32_1</td><td>0</td><td>236.3</td><td>0</td><td>0.06</td><td>49.9</td><td>0</td><td>20-06-2024</td><td>13:03:55</td></tr> <tr><td>ESP32_1</td><td>0</td><td>236.2</td><td>0</td><td>0.06</td><td>49.9</td><td>0</td><td>20-06-2024</td><td>13:03:58</td></tr> <tr><td>ESP32_1</td><td>0</td><td>236.3</td><td>0</td><td>0.06</td><td>49.9</td><td>0</td><td>20-06-2024</td><td>13:04:01</td></tr> <tr><td>ESP32_1</td><td>0</td><td>236.5</td><td>0</td><td>0.06</td><td>49.9</td><td>0</td><td>20-06-2024</td><td>13:04:04</td></tr> <tr><td>ESP32_1</td><td>0</td><td>236.3</td><td>0</td><td>0.06</td><td>50</td><td>0</td><td>20-06-2024</td><td>13:04:07</td></tr> </tbody> </table> <p>Gráficas aplicación web:</p>	ID	Potencia (W)	Voltaje (V)	Intensidad (A)	Energía	Frecuencia	Factor de potencia	Fecha	Hora	ESP32_1	0	235.7	0	0.06	49.9	0	20-06-2024	13:03:42	ESP32_1	0	235.9	0	0.06	49.9	0	20-06-2024	13:03:45	ESP32_1	0	236.2	0	0.06	50	0	20-06-2024	13:03:48	ESP32_1	0	236.5	0	0.06	50	0	20-06-2024	13:03:52	ESP32_1	0	236.3	0	0.06	49.9	0	20-06-2024	13:03:55	ESP32_1	0	236.2	0	0.06	49.9	0	20-06-2024	13:03:58	ESP32_1	0	236.3	0	0.06	49.9	0	20-06-2024	13:04:01	ESP32_1	0	236.5	0	0.06	49.9	0	20-06-2024	13:04:04	ESP32_1	0	236.3	0	0.06	50	0	20-06-2024	13:04:07
ID	Potencia (W)	Voltaje (V)	Intensidad (A)	Energía	Frecuencia	Factor de potencia	Fecha	Hora																																																																																			
ESP32_1	0	235.7	0	0.06	49.9	0	20-06-2024	13:03:42																																																																																			
ESP32_1	0	235.9	0	0.06	49.9	0	20-06-2024	13:03:45																																																																																			
ESP32_1	0	236.2	0	0.06	50	0	20-06-2024	13:03:48																																																																																			
ESP32_1	0	236.5	0	0.06	50	0	20-06-2024	13:03:52																																																																																			
ESP32_1	0	236.3	0	0.06	49.9	0	20-06-2024	13:03:55																																																																																			
ESP32_1	0	236.2	0	0.06	49.9	0	20-06-2024	13:03:58																																																																																			
ESP32_1	0	236.3	0	0.06	49.9	0	20-06-2024	13:04:01																																																																																			
ESP32_1	0	236.5	0	0.06	49.9	0	20-06-2024	13:04:04																																																																																			
ESP32_1	0	236.3	0	0.06	50	0	20-06-2024	13:04:07																																																																																			

Tabla 13: Pruebas Websockets - Resultados recepción de mensajes en aplicación web

#### **4.5.5 Envío de mensaje desde la aplicación web**

##### **Condiciones iniciales:**

- La aplicación web está iniciada y conectada al servidor.
- El servidor debe estar inicializado, y la conexión con la aplicación web debe haber sido registrada.

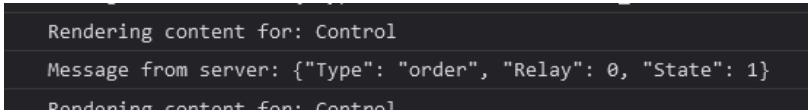
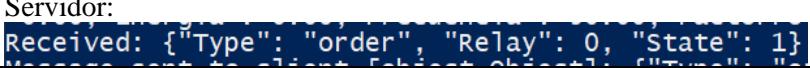
<b>Resultado esperado</b>	Si el mensaje se ha creado y enviado correctamente se mostrará el contenido del mensaje en la consola de la aplicación web, y si se recibe correctamente aparecerá en la consola del servidor, y será reenviado a los clientes conectados.
<b>Resultado obtenido</b>	Consola ESP32:  Servidor: 

Tabla 14: Pruebas Websockets - Resultados envío de mensaje desde aplicación web

#### **4.5.6 Recepción de mensaje en el ESP32**

##### **Condiciones iniciales:**

- El ESP32 debe estar conectado a una fuente de alimentación y a una red Wi-Fi y con el programa cargado. Debe estar recibiendo datos y conectado al servidor.
- El servidor debe estar inicializado, y la conexión con el ESP32 debe haber sido registrada. Debe haber recibido o estar recibiendo mensajes de la aplicación web.

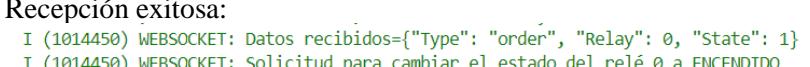
<b>Resultado esperado</b>	Si el mensaje ha llegado al servidor, este es reenviado a los clientes conectados. Si el ESP32 recibe el mensaje, se mostrará el contenido por consola y se iniciará el proceso para el cambio de estado del relé.
<b>Resultado obtenido</b>	Recepción exitosa: 

Tabla 15: Pruebas Websockets - Resultados recepción de mensaje en ESP32

#### **4.6 Modificación del estado de un relé**

##### **Condiciones iniciales:**

- El ESP32 debe estar conectado a una fuente de alimentación y a una red Wi-Fi y con el programa cargado. Debe estar recibiendo mensajes del servidor y estar conectado físicamente al módulo de relés.
- El módulo debe estar conectado al ESP32, con las líneas SCL y SDA, alimentación y tierra.

<b>Resultado esperado</b>	Si la configuración de la comunicación I2C se ha realizado correctamente se mostrará un mensaje por consola. Una vez recibida la orden en el ESP32 se iniciará la acción de cambiar el estado del relé. Si el cambio es efectivo el LED correspondiente al relé invertirá su estado (si estaba apagado se encenderá y viceversa) indicando el estado del relé. Esto provocará el envío de un mensaje de vuelta hacia la aplicación web a través del servidor, que modificará la apariencia del botón utilizado (pasando de “Encender” a “Apagar” y viceversa).
<b>Resultado obtenido</b>	<p>Configuración I2C:</p> <p>I (5310) PLACA_RELÉS: Estableciendo configuración I2C...  I (5310) PLACA_RELÉS: Inicializando driver I2C...</p> <p>Orden de modificación de estado:</p> <p>I (75840) WEBSOCKET: Datos recibidos={"Type": "order", "Relay": 3, "State": 1}  I (75840) WEBSOCKET: Solicitud para cambiar el estado del relé 3 a ENCENDIDO</p> <p>Modificación de estado:</p> <p>I (75840) PLACA_RELÉS: Relé 3 ENCENDIDO</p>  <p>Notificación de cambio:</p> <pre>Message from server: {"Type": "order", "Relay": 3, "State": 0}</pre> <p>Cambio botón:</p>



Tabla 16: Pruebas Relé - Resultados modificación de estado

# **Capítulo 5: Conclusiones y trabajo futuro**

En este capítulo se hablará sobre las conclusiones extraídas tras haber completado la implementación del sistema y sobre las posibilidades que presenta el sistema de cara al futuro, incluyendo cambio y mejoras.

## **5.1 Conclusiones**

En primer lugar, quiero expresar que la realización de este trabajo ha sido un proceso muy interesante y enriquecedor. Pese a las dificultades encontradas, los problemas que han ido surgiendo y los errores cometidos en el transcurso de incontables horas de trabajo, sin duda alguna, ha sido una experiencia en la que me he divertido y que he disfrutado inmensamente. También ha tenido momentos de creatividad que brindaban alegría y levantaban el ánimo, como el diseño de la interfaz de la web, comprobar el funcionamiento de un componente y que sea el esperado, o el montaje del circuito físico.

En cuanto a la consecución de objetivos, creo que con creces se han cubierto todos los puntos establecidos, detallados al inicio de este documento. El primer y más importante objetivo era construir un sistema sencillo de implementar, barato y fácil de comprender y usar. Sobre el coste del proyecto, se comenta en profundidad en el Anexo I – Presupuesto, queda por debajo de los 35€, y todo el coste procede de la parte física, ya que el software empleado es gratuito en su totalidad, escogiendo herramientas de código abierto en los casos que se ha podido. Sobre la sencillez de implementación, varios familiares que han participado leyendo el documento creen que, recibiendo el código ya cargado en el ESP32 y la aplicación web ya creada, serían capaces de montar el circuito físico, y siguiendo las instrucciones del documento inicializar los diferentes componentes con mayor o menor dificultad. Sobre la comprensión y la usabilidad, se ha mostrado este documento a gente del entorno familiar y amistades sin estudios relacionados con el campo de la informática y la opinión más común entre los participantes es que el sistema es muy intuitivo de utilizar gracias a la simpleza de la UI que presenta la aplicación web.

El segundo objetivo consistía en obtener los conocimientos sobre dos campos que no se han tratado mucho durante el estudio del grado. Comenzando por el campo de la tecnología web, no solo he adquirido los conocimientos necesarios para crear de cero una aplicación web plenamente funcional y con características que jamás hubiera imaginado que podría incluir, si no que también he descubierto una pasión hasta ahora desconocida por el diseño web. Ha resultado ser un tema que encuentro realmente interesante, que disfruto aprendiendo y llevando la teoría a la práctica, y sobre el que sin duda seguiré profundizando en un futuro. En cuanto a la electrónica, pese a no tener una gran habilidad para el trabajo manual, ha sido una actividad entretenida, aunque a ratos algo caótica, y que también he encontrado interesante.

El último objetivo representaba la superación personal, la demostración a uno mismo de que puede desempeñar una tarea de este calibre si se enfoca y pone voluntad en ello. Creo que este objetivo lo he cumplido con creces. He sido capaz de dedicarle al proyecto el tiempo que ha necesitado, organizándome para poder compaginar el desarrollo con otras actividades de mi vida, sin que el sistema disminuyese en calidad.

Respecto al proyecto en sí, debo decir que no ha representado un reto tan inmenso como cuando escogí el tema y tuve la primera reunión con el profesor sobre el sistema, sus componentes y conexiones. Quiero destacar que en todo momento he tratado de mantener un nivel mínimo de

limpieza al crear código, con la vista puesta en futuros proyectos donde sea necesario el mantenimiento constante del código o su actualización, o la colaboración con otros desarrolladores, donde seguir buenas prácticas siempre será de ayuda.

En resumen, la realización de este proyecto ha sido un proceso que he disfrutado enormemente, cumpliendo los objetivos impuestos y superando las expectativas que tenía en un principio.

## 5.2 Trabajo a futuro

Dentro de esta sección se expondrán posibles cambios o mejoras a realizar en un futuro y que permitan que el sistema siga evolucionando y adaptándose a los tiempos actuales donde la tecnología sufre avances constantemente.

Una de las principales propuestas sería la de migrar los componentes alojados en el PC a un servidor en la nube. De esta forma se permite la gestión y el acceso remoto, reduciendo el gasto de recursos por parte del usuario. De la mano de este cambio, se podría incluir un *bucket* en el gestor de datos que almacene credenciales (pares usuario-contraseña). De este modo, implementando un inicio de sesión en la aplicación web, cada usuario vería una versión de la aplicación web acorde a su instalación. Este cambio centralizaría el mantenimiento y la gestión de los componentes intangibles, conllevando modificaciones para evitar tener que implementar un conjunto de componentes para cada usuario y un gasto desproporcionado de recursos.

Otra línea para futuras mejoras viene de que en el mundo de la domótica hay infinidad de componentes y dispositivos que sería atractivo incluir en este sistema, aportando funcionalidad o ampliando alguna ya existente. Por ejemplo, podrían implementarse servomotores en el sistema físico, también controlados por el ESP32 recibiendo las órdenes desde la aplicación web. Estos permitirían el apagado o encendido de interruptores de pared, presentes en todos los hogares, de forma remota.

Ya que el sistema puede controlar cargas de salidas específicas de corriente, podrían incluirse sensores de diversos parámetros como temperatura o luz, que manden sus mediciones a la aplicación web como ya sucede con el sensor de medidas eléctricas. De este modo, con electrodomésticos conectados a salidas de corriente controladas por el sistema, se pueden dar situaciones como, por ejemplo, ver en la aplicación que se detecta un aumento de temperatura, y en consecuencia alimentar la salida conectada al aire acondicionado para que comience a funcionar.

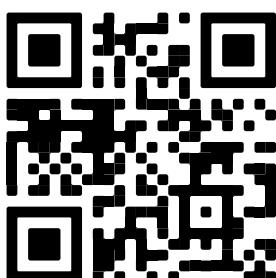
Viendo las características del sistema es fácil ver que sin duda, el sistema diseñado tiene potencial para ampliar por diferentes vías su funcionalidad, según las necesidades específicas y los deseos de cada usuario.

# Capítulo 6: Entregables

El propósito de esta sección es indicar los elementos desarrollados en el sistema, una vez se ha completado su implementación.

- Prototipo con el controlador IoT ESP32, un kit de medición PZEM-004T y una placa de expansión con Relés, basada en el expensor de entradas y salidas de propósito general XL9535, conectadas ambas mediante un bus I2C.
- Aplicación web para la monitorización y el gobierno de las distintas cargas eléctricas conectadas.
- Maqueta con consumos simulados.
- Subidos en un repositorio de Github se encuentran los siguientes elementos:
  - Documento de análisis y diseño del sistema
  - Código fuente:
    - Aplicación web.
    - Programa cargado en el ESP32
  - Archivos de configuración de los siguientes servicios que estarán alojados en un PC:
    - Sistema de suscripción y publicación de mensajes MQTT.
    - Sistema de intercambio de mensajes mediante Websockets (servidor de Websockets).
    - Gestor de bases de datos Influxdb y agente de recolección Telegraf.

Se puede acceder al repositorio de Github escaneando el siguiente QR:



O bien visitando en un navegador el siguiente enlace:  
<https://github.com/DavidRuizg/TFG>



# Bibliografía y referencias

- [1] Lucena, P. (2023, 6 mayo). ¿Qué es el framework? | 2024. Maestrías y MBA. <https://www.cesuma.mx/blog/que-es-el-framework.html#:~:text=Un%20framework%20es%20un%20conjunto%20de%20reglas%20y%20convenciones%20que,utilizar%20como%20punto%20de%20partida>.
- [2] Degni, R. (2024, 11 marzo). Una completa introducción a la librería React. Codemotion Magazine. <https://www.codemotion.com/magazine/es/lenguajes-de-programacion/una-completa-introduccion-a-la-libreria-react/#:~:text=React%20es%20una%20librer%C3%A1%20de,complejas%20y%20de%20alto%20rendimiento>.
- [3] Lucas, J. (2019, 4 septiembre). Qué es NodeJS y para qué sirve. OpenWebinars.net. <https://openwebinars.net/blog/que-es-nodejs/>.
- [4] Harsh, K. (2022, 14 octubre). ¿Qué es la Arquitectura de las Aplicaciones Web? Desglosando una Aplicación Web. Kinsta®. <https://kinsta.com/es/blog/arquitectura-aplicaciones-web/#tipos-de-arquitectura-de-aplicaciones-web>.
- [5] Mora, S. L. (2022, 4 octubre). ¿Qué son las Single-Page Application (SPA)? El desarrollo elegido por Gmail y LinkedIn. DIGITAL55. <https://digital55.com/blog/que-son-single-page-application-spa-desarrollo-elegido-por-gmail-linkedin/>.
- [6] midulive. (2023, 4 abril). CURSO REACT 2024 - Aprende desde cero [Vídeo]. YouTube. [https://www.youtube.com/watch?v=7iobxzd\\_2wY](https://www.youtube.com/watch?v=7iobxzd_2wY)
- [7] Tecnoweb2.com. (s. f.). Las tecnologías web | tecnoweb2.com. <https://www.tecnoweb2.com/tecnologias-web>
- [8] Morales, J. A. R. (2022, 12 febrero). Esp32 características y pines. PASIÓN ELECTRÓNICA. <https://pasionelectronica.com/esp32-caracteristicas-y-pines/>.
- [9] Llamas, L. (2023, 18 agosto). Pinout y detalles del hardware del ESP32. Luis Llamas. <https://www.luisllamas.es/esp32-detalles-hardware-pinout/>.
- [10] ¿Qué es el código abierto? - Explicación del código abierto - AWS. (s. f.). Amazon Web Services, Inc. <https://aws.amazon.com/es/what-is/open-source/>.
- [11] Kinsta. (2023, 15 junio). Qué es Node.js y por qué deberías usarlo. Kinsta®. <https://kinsta.com/es/base-de-conocimiento/que-es-node-js/#caracteristicas-de-nodejs>.
- [12] Front End frente a back-end: diferencia entre el desarrollo de aplicaciones - AWS. (s. f.). Amazon Web Services, Inc. <https://aws.amazon.com/es/compare/the-difference-between-frontend-and-backend/#:~:text=El%20front%20end%20es%20aquellos,permiten%20que%20la%20aplicaci%C3%B3n%20funcione>.

- [13] Equipo editorial de IONOS. (2020, 1 octubre). Callback: ¿qué son las funciones callback? IONOS Digital Guide. <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-un-callback/>.
- [14] Metwalli, S. A. (2023, 28 marzo). What is NPM? Built In. [https://builtin.com/software-engineering-perspectives/npm#:~:text=Node%20package%20manager%20\(npm\)%20is,an%20robotics%20to%20mobile%20apps](https://builtin.com/software-engineering-perspectives/npm#:~:text=Node%20package%20manager%20(npm)%20is,an%20robotics%20to%20mobile%20apps).
- [15] Arsys. (2023, 8 marzo). ¿Qué es NPM? JavaScript para principiantes. <https://www.arsys.es/blog/que-es-npm-javascript-para-principiantes#:~:text=NPM%20es%20una%20herramienta%20fundamental,con%20lo%20que%20llamamos%20dependencias>.
- [16] ¿Por qué Vite? (s. f.). Vitejs. <https://es.vitejs.dev/guide/why>.
- [17] ¿Qué son los módulos (ESM)? - Javascript en español. (s. f.). Lenguaje JS. <https://lenguajejs.com/javascript/modulos/que-es-esm/>.
- [18] Carmenate, J. G. (2022, 7 marzo). ESP32 Wifi + Bluetooth en un solo lugar. Programarfacil Arduino y Home Assistant. <https://programarfacil.com/esp8266/esp32/>.
- [19] Carranza, S. (2022, 17 septiembre). CONOCIENDO AL ESP32. TodoMaker. <https://todomaker.com/blog/conociendo-al-esp32/> ¿Por qué Vite? (s. f.). Vitejs. <https://es.vitejs.dev/guide/why> ¿Qué son los módulos (ESM)? - Javascript en español. (s. f.). Lenguaje JS. <https://lenguajejs.com/javascript/modulos/que-es-esm/>.
- [20] Optoacopladores | RS. (s. f.). <https://es.rs-online.com/web/c/displays-y-optoelectronica/ptoacopladores-y-fotodetectores/ptoacopladores/>.
- [21] Mcauser. (s. f.). GitHub - mcauser/micropython-xl9535-kxv5-relay: A MicroPython library for xl1 XL9535-KxV5 I2C relay boards. GitHub. <https://github.com/mcauser/micropython-xl9535-kxv5-relay>.
- [22] Martínez, J. (2022, 25 febrero). Qué es Influxdb y primeros pasos. OpenWebinars.net. <https://openwebinars.net/blog/que-es-Influxdb-y-primeros-pasos/>.
- [23] Alfaiot-Webmaster. (2024, 24 enero). Qué es Influxdb. AlfaIOT. <https://alfaiot.com/iot/Influxdb/>.
- [24] SPSS Modeler Subscription. (s. f.). <https://www.ibm.com/docs/es/spss-modeler/saas?topic=models-time-series-data>.
- [25] InfluxData. (2021, 10 diciembre). Influxdb: Open Source Time Series Database | InfluxData. <https://www.influxdata.com/time-series-platform/telegraf/>.
- [26] ¿Qué es un agente de escucha de eventos? - Explicación del agente de escucha de eventos de Javascript - AWS. (s. f.). Amazon Web Services, Inc. <https://aws.amazon.com/es/what-is/event-listener/>.

[27] Using the Effect Hook – React. (s. f.). React. <https://legacy.reactjs.org/docs/hooks-effect.html>.

[28] Casero, A. (2024, 8 abril). ¿Qué es un script en JavaScript? | KeepCoding Bootcamps. KeepCoding Bootcamps. <https://keepcoding.io/blog/que-es-un-script-en-javascript/#:~:text=Un%20script%20en%20JavaScript%20es,web%20y%20realizar%20diversas%20acciones>.

[29] Kinsta. (2023a, mayo 29). Guía sin Complicaciones de la Sintaxis JSX - Kinsta®. Kinsta®. [https://kinsta.com/es/base-de-conocimiento/que-es-javascript/#:~:text=JSX%20\(JavaScript%20XML\)%20es%20una,cierre%2C%20atributos%20y%20elementos%20anidados](https://kinsta.com/es/base-de-conocimiento/que-es-javascript/#:~:text=JSX%20(JavaScript%20XML)%20es%20una,cierre%2C%20atributos%20y%20elementos%20anidados).



# Acrónimos y abreviaturas

Acrónimos y abreviaturas	Significados
<b>AP</b>	Punto de Acceso (Access Point)
<b>CLI</b>	Command Line Interface
<b>CRC</b>	Cyclic Redundancy Check
<b>ESP32</b>	Módulo de microcontrolador con Wi-Fi y Bluetooth
<b>GND</b>	Línea de tierra (Ground)
<b>I2C</b>	Inter-Integrated Circuit
<b>IDE</b>	Entorno de Desarrollo Integrado (Integrated Development Environment)
<b>IP</b>	Protocolo de Internet (Internet Protocol)
<b>JSON</b>	JavaScript Object Notation
<b>MQTT</b>	Message Queuing Telemetry Transport
<b>NTP</b>	Network Time Protocol
<b>PC</b>	Computadora Personal
<b>PZEM-004T</b>	Sensor de mediciones eléctricas
<b>RAM</b>	Memoria de Acceso Aleatorio (Random Access Memory)
<b>SCL</b>	Serial Clock Line
<b>SDA</b>	Serial Data Line
<b>SNTP</b>	Simple Network Time Protocol
<b>SPI</b>	Serial Peripheral Interface
<b>STA</b>	Estación (Station)
<b>TFG</b>	Trabajo de Fin de Grado
<b>UART</b>	Universal Asynchronous Receiver-Transmitter
<b>UI</b>	Interfaz de Usuario (User Interface)
<b>XL9535-K4V5</b>	Módulo de relé de expansión



# Anexo I – Presupuesto

Una vez finalizado el montaje y la implementación del sistema, el coste final incurrido es bastante moderado.

En cuanto a la parte lógica, ninguno de los softwares y programas utilizados conllevaba un pago o suscripción. Se ha tratado en todo momento de escoger herramientas de código abierto, tanto por las ventajas que conlleva tener una comunidad de desarrollo a la hora de utilizar la herramienta, como por el nulo coste para utilizarlas.

Por tanto, el coste incurrido en la parte logística es:

Visual Studio Code	0€
ESP-IDF	0€
Influxdb	0€
Telegraf	0€
Eclipse Mosquitto	0€
React	0€
Vite	0€

Tabla 17: Costes de la parte no tangible del sistema

La adquisición de algunos de los componentes físicos, sí que ha requerido de inversión económica, aunque desde las fuentes correctas, el gasto no resulta muy elevado. Algunos de los componentes no han tenido que adquirirse porque fueron utilizados en pasados cursos, como el ESP32 o el cableado para este. Aun así, se buscará su precio en el mercado de cara a reflejar el coste que tendría el proyecto comenzando de cero sin ningún material. Los precios han sido obtenidos de Aliexpress, Amazon y Leroy Merlin.

ESP32	3,16€ (Aliexpress)/11,99€ (Amazon)
Placa de inserción	5,49€
Kit PZEM-004T	5,17€ (Aliexpress)/18,99 (Amazon)
XL9535-K4V5	4,96€
Cables macho/macho	1,20€
Cables macho/hembra	1,03€
Clavija macho (1 unidad)	1,00€
Clavija hembra (4 unidades)	1,00€/u.
Regleta 12 conexiones	0,51€
Manguera 5M	7,29€
Cable microUSB a USB-A	0,97€

Tabla 18: Costes de la parte tangible del sistema

Por tanto, según la procedencia elegida en algunos de los elementos, el coste final del proyecto (despreciando costes de luz y herramientas como el PC utilizado) ascendería a:

- Procedentes de Aliexpress: 34,60€
- Procedentes de Amazon: 57,26€

Dado que ambas vías han demostrado ser fiables, al menos en la experiencia con este proyecto, se recomienda optar por la vía más económica.



## Anexo II – Manuales de usuario e Instalación

En esta sección se cubrirán dos puntos:

- Explicación detallada el proceso de instalación de algunos elementos esenciales para el desarrollo y la implementación del proyecto,
- Instrucciones de uso de la aplicación web y toda su funcionalidad.

### Manual de instalación: Entorno de desarrollo ESP32

En este manual se explicará paso por paso cómo instalar el entorno de desarrollo Visual Studio Code, y una vez instalado, como configurarlo con todo lo necesario para poder realizar el desarrollo de este proyecto, si se quisiera hacer desde cero.

#### *Instalación del entorno*

El primer paso hacia nuestro objetivo de tener un entorno plenamente funcional es descargar el instalador del entorno. Para ello bastará con buscar en un navegador “Visual Studio Code descargar” y aparecerán varias páginas para ello. Seleccionaremos la primera opción, página oficial del software.

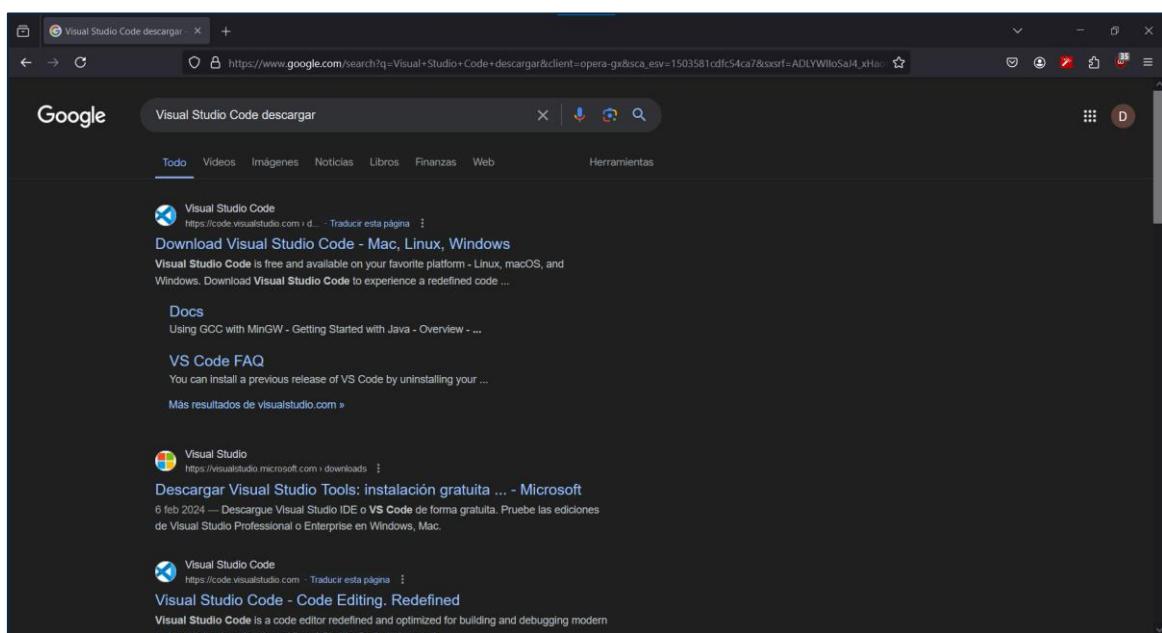


Figura 19: Navegador - Búsqueda VS Code

Una vez dentro deberemos escoger el instalador correspondiente para el sistema operativo de nuestro dispositivo. En este manual se realizará la instalación en Windows.

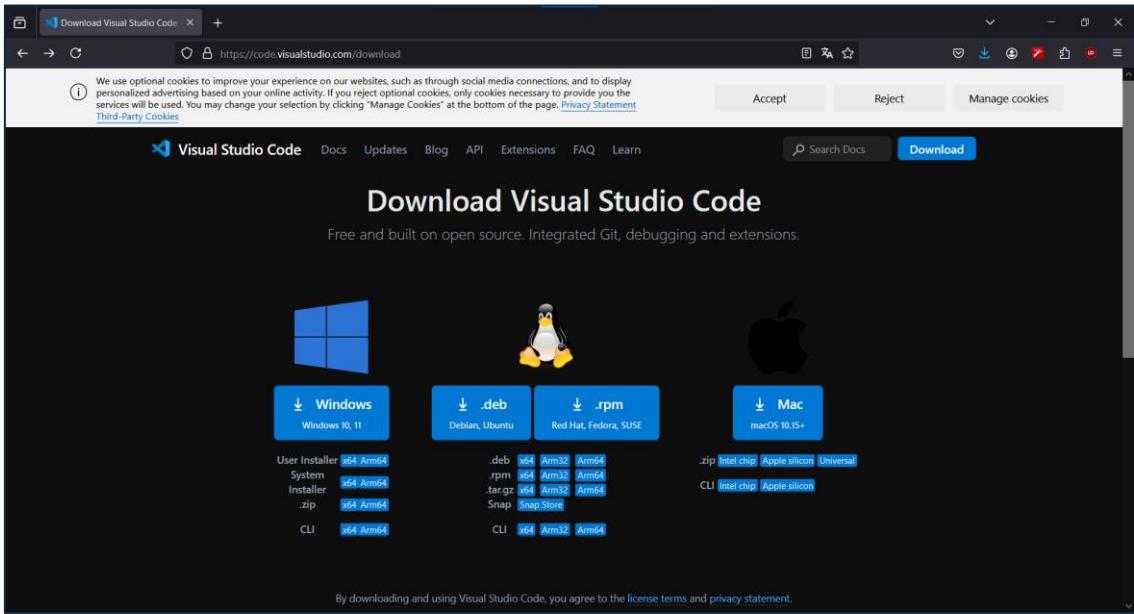


Figura 20: Página de VS Code – Instaladores

Una vez elegido el instalador deseado, se pulsa el botón correspondiente. Comenzará la descarga de forma automática y se nos redirigirá a una página con los pasos a seguir para comenzar a usar el entorno.

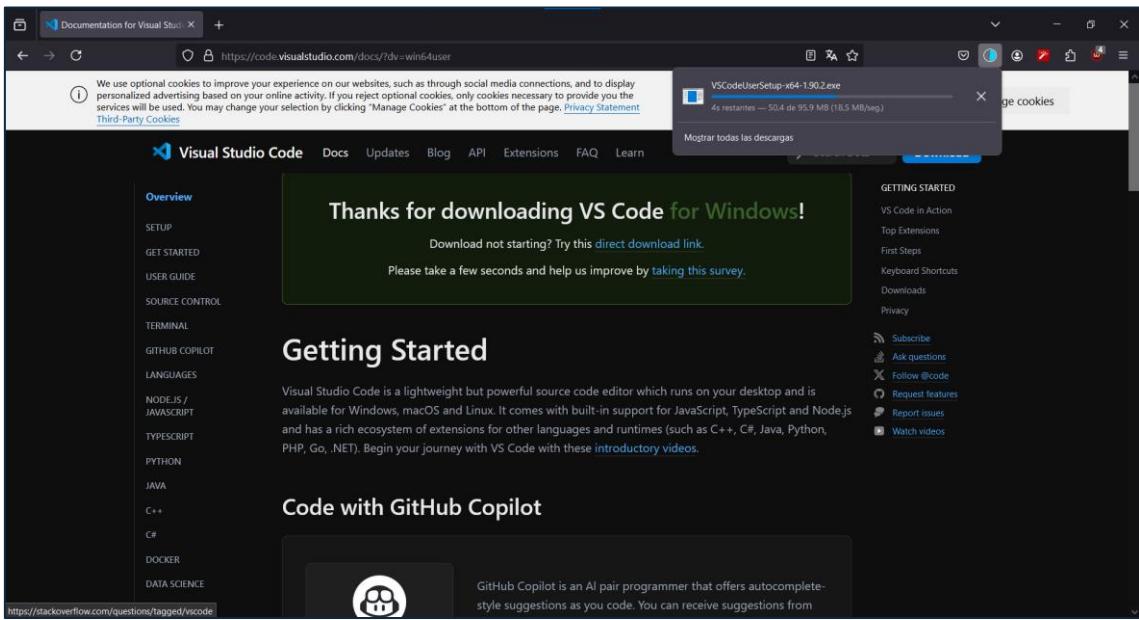


Figura 21: Página VS Code - Descarga + Tutorial guiado

Buscaremos en los archivos del sistema el instalador descargado, normalmente aparecerá en la carpeta “Descargas”, y lo ejecutaremos. La primera pantalla que aparece muestra los términos de la licencia del software. Tendremos que aceptar el acuerdo para poder continuar.

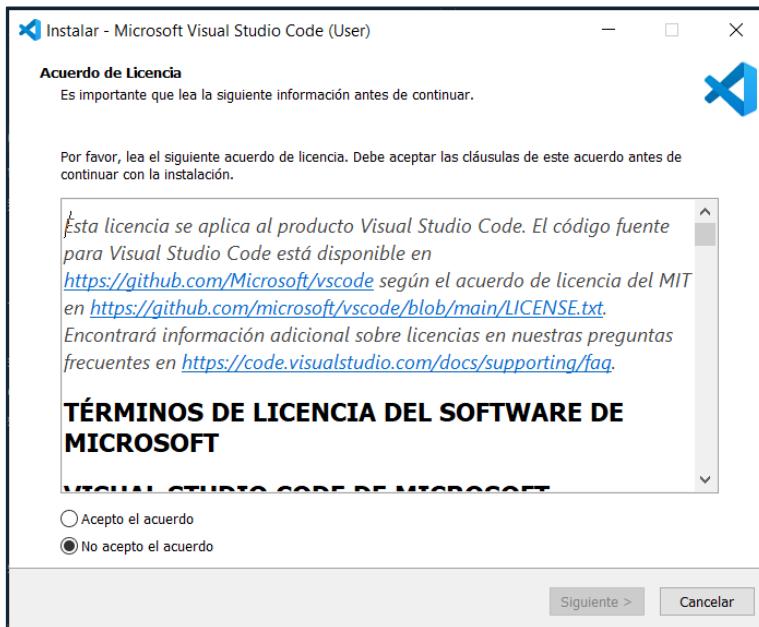


Figura 22: Instalación VS Code - Términos de licencia de Microsoft

A continuación, se nos mostrarán distintas opciones adicionales que harán el uso del entorno mucho más sencillo y fluido. Queda a juicio del usuario qué casillas marcar. Aun así, se recomienda marcar todas las casillas del apartado “Otros”.

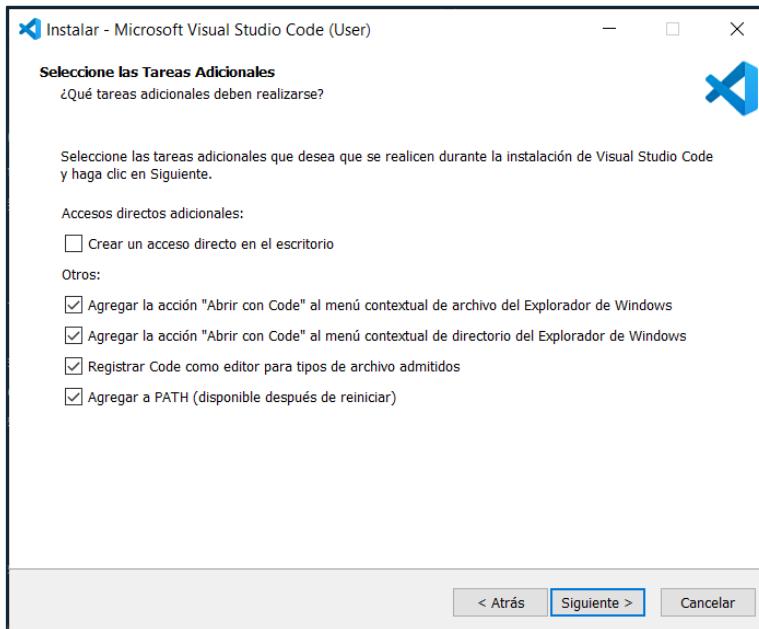


Figura 23: Instalación VS Code - Opciones adicionales

Aparecerá un resumen de la instalación, y por fin veremos el botón “Instalar”.

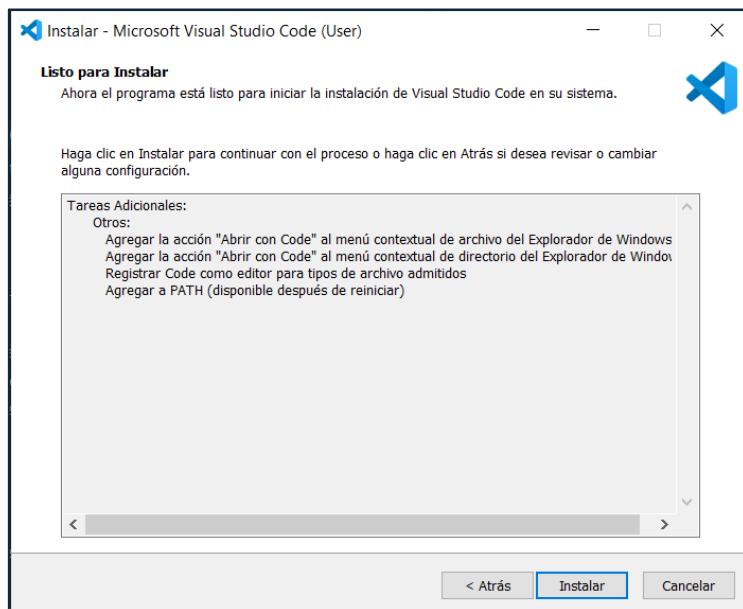


Figura 24: Instalación VS Code - Resumen de la instalación

Si todo ha salido correctamente deberíamos tener ante nosotros el siguiente panel:

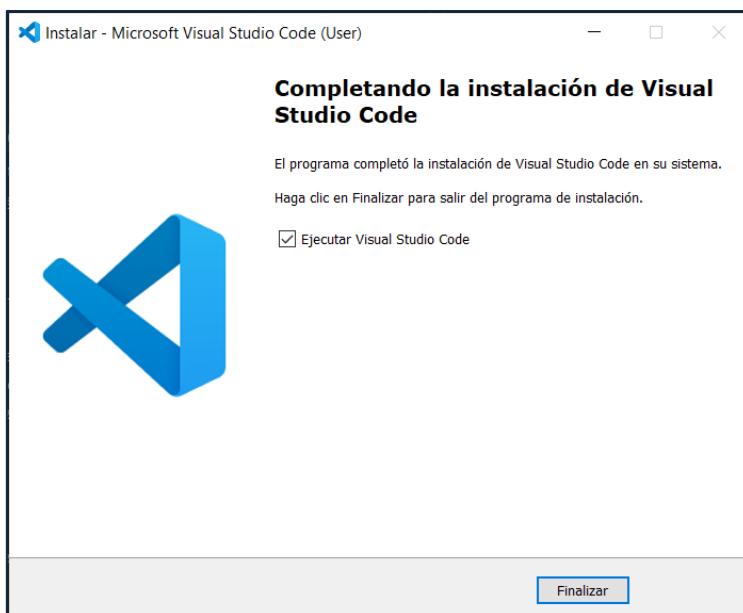


Figura 25: Instalación VS Code - Final de la instalación

Si se pulsa “Finalizar” dejando marcada la casilla se ejecutará Visual Studio Code. Ya está instalado el entorno de desarrollo. El siguiente paso es configurarlo con todas las características necesarias.

## Configuración del entorno

Si abrimos el entorno, lo primero que aparecerá será la ventana de bienvenida con diferentes opciones como “Nuevo archivo”, “Abrir archivo” o “Abrir carpeta”, además de ofrecer varios tutoriales. Lo que debemos hacer para configurar correctamente el entorno es dirigirnos a la sección “Extensiones” del menú izquierdo.

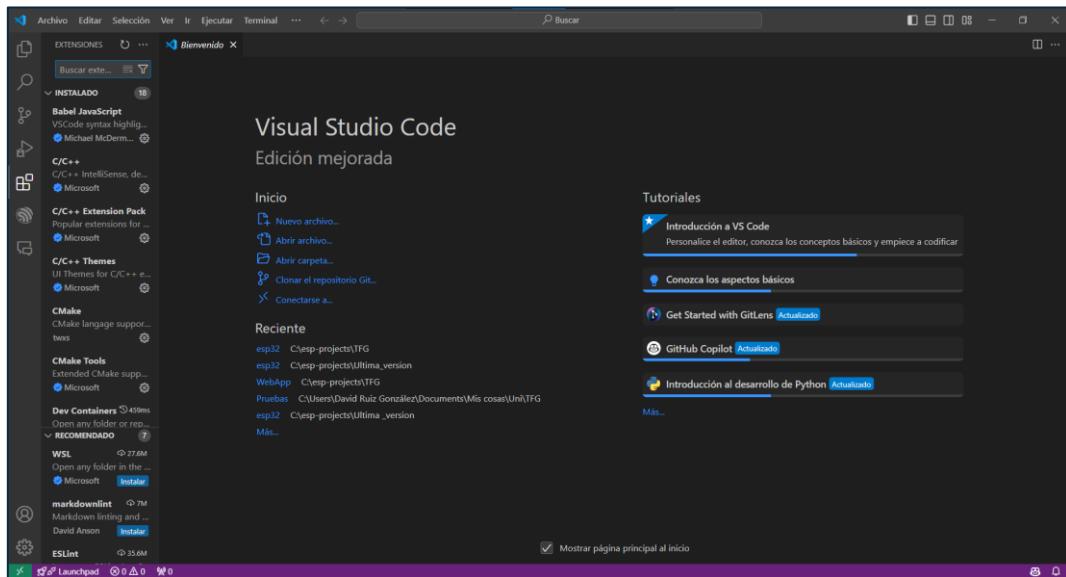


Figura 26: Configuración VS Code – Extensiones

Una vez en la sección, la lista de extensiones necesarias que habrá que instalar es:

- C/C++
- CMake Tools
- ESP-IDF

De forma opcional, pero muy recomendada, se pueden instalar las siguientes dependencias, que sin duda son de ayuda durante el desarrollo:

- Spanish Language Pack for Visual Studio Code (En caso de no querer trabajar con el entorno en inglés)
- Babel JavaScript
- npm Intellisense

Una vez instaladas en su última versión y habilitadas, la única extensión que requiere de pasos extra para su configuración es ESP-IDF. Haciendo clic sobre el ícono de Espressif del menú izquierdo (que habrá aparecido tras añadir la extensión) se mostrará una ventana de bienvenida. En caso de ser la primera vez tendrá este aspecto:

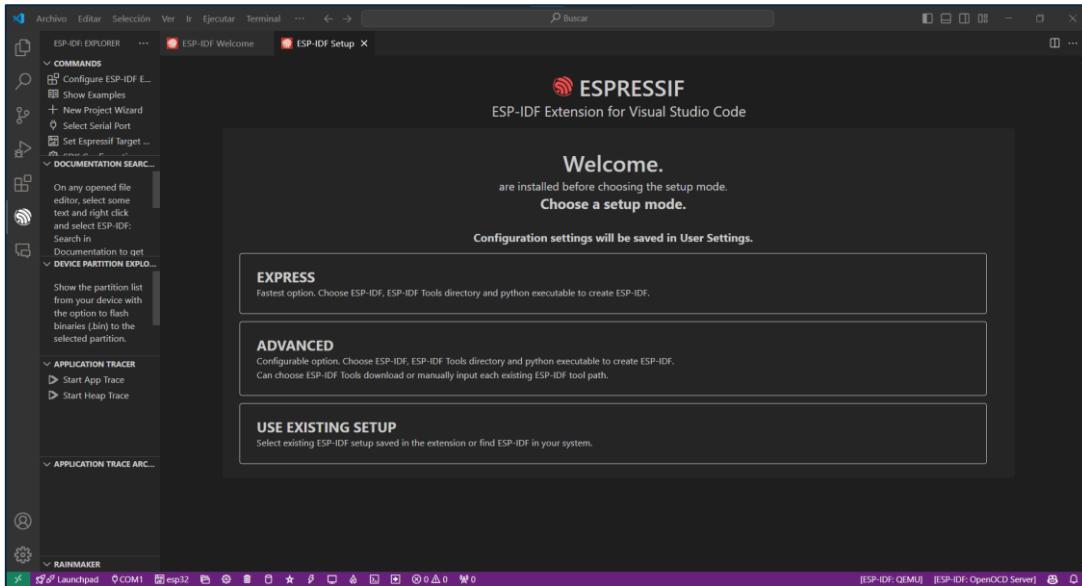


Figura 27: Configuración VS Code – ESP-IDF

La mejor opción si no conoces a fondo todos los detalles que pueden modificarse para la instalación es Express. Se nos pedirá que escogamos el servidor de descarga (Github), la versión de ESP-IDF (la que el usuario considere) y las carpetas donde se alojarán ESP-IDF y ESP-IDF tools.

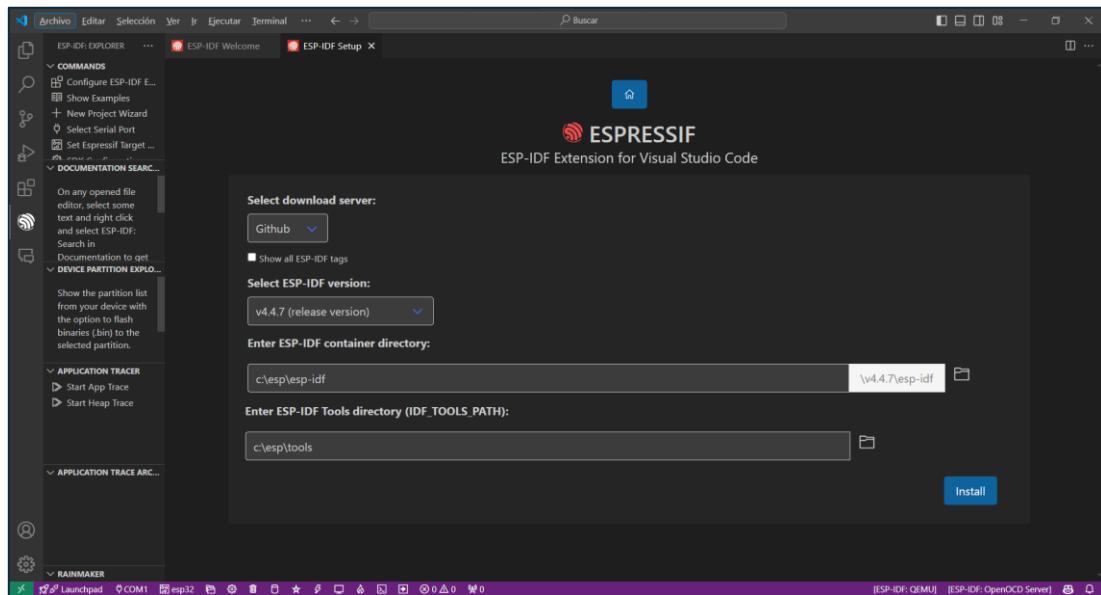


Figura 28: Configuración VS Code – ESP-IDF instalación

Cuando todo esté en orden se podrá pulsar el botón “Install”, que comenzará la instalación de módulos, librerías, dependencias y todo lo necesario para la programación de microcontroladores compatibles con el SDK de Espressif Systems.

Ya se podrá, por ejemplo, utilizar plantillas o crear nuevos proyectos desde cero.

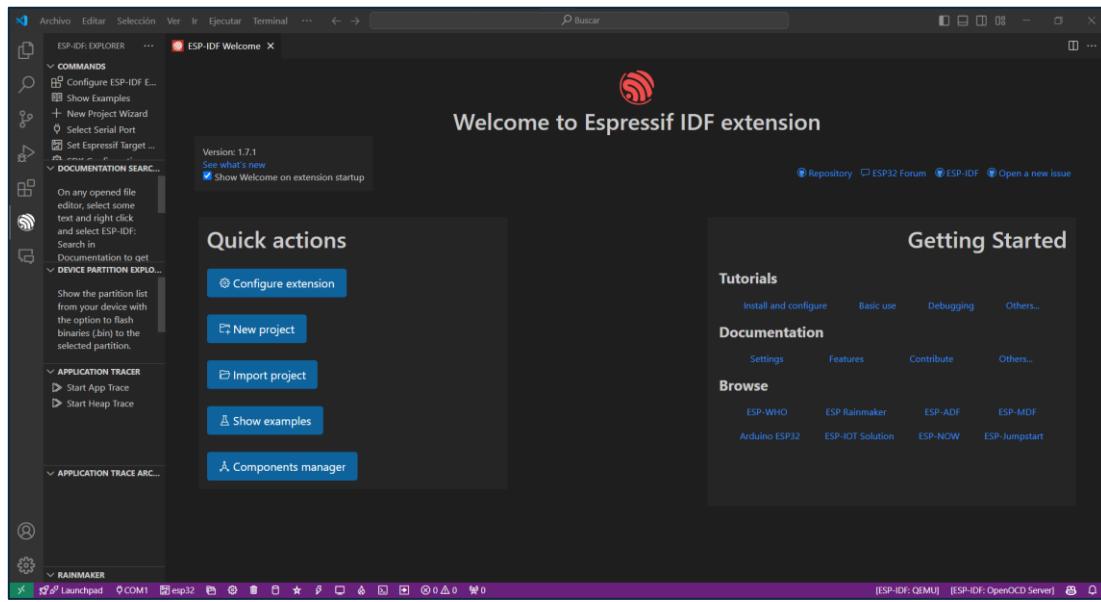


Figura 29: VS Code - ESP-IDF intalado correctamente

# Manual de instalación: Node.js y npm

Para comenzar la instalación de Node.js para Windows, lo primero es ir a la página oficial y descargar el instalador.

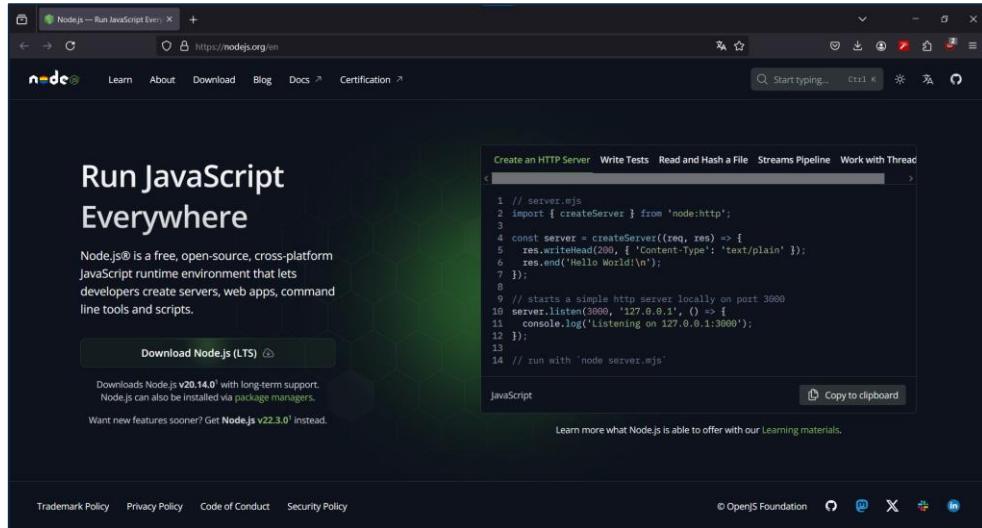


Figura 30: Instalación de Node.js - Descarga

Una vez descargado se busca el archivo y se ejecuta, abriendo así el asistente de instalación. Una vez abierto se aceptarán los términos.

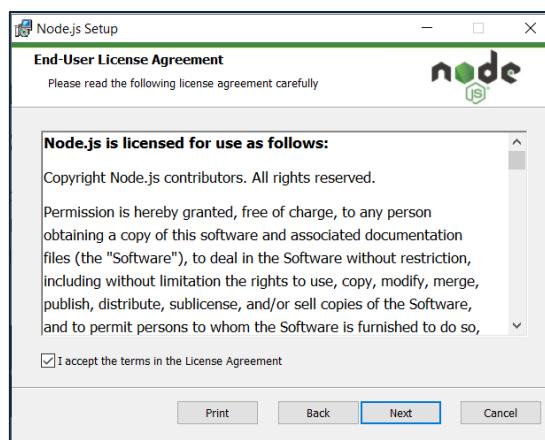


Figura 31: Instalación de Node.js – Términos

Se procederá a escoger la ubicación de la instalación.

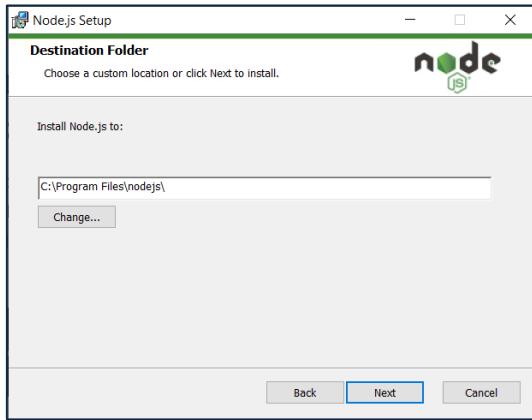


Figura 32: Instalación de Node.js - Dirección de instalación

Y se escogerá la instalación deseada, en este punto se puede ver que nos ofrece varias opciones de instalación, añadiendo o quitando elementos. Uno de ellos es el gestor de paquetes npm que necesitamos para manejar y gestionar dependencias y ejecutar *scripts* de la aplicación web. Por lo tanto, lo más recomendable es dejar las opciones por defecto, que incluyen Node.js y npm.

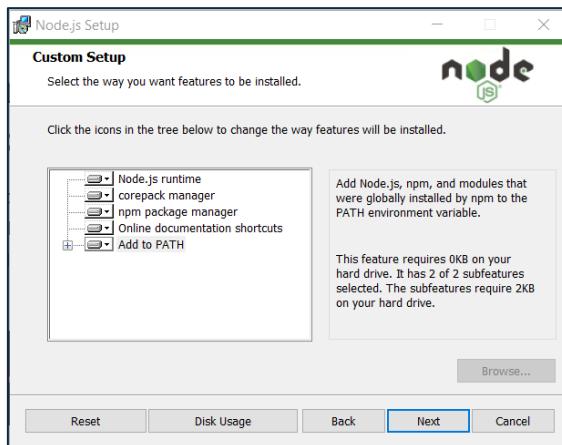


Figura 33: Instalación de Node.js - Opciones de instalación

Lo siguiente que se nos pregunta es si se deberán instalar automáticamente las herramientas necesarias para la instalación. Normalmente, si es la primera instalación, es raro que estas herramientas se encuentren ya en el sistema. Por ello se deberá marcar la casilla.

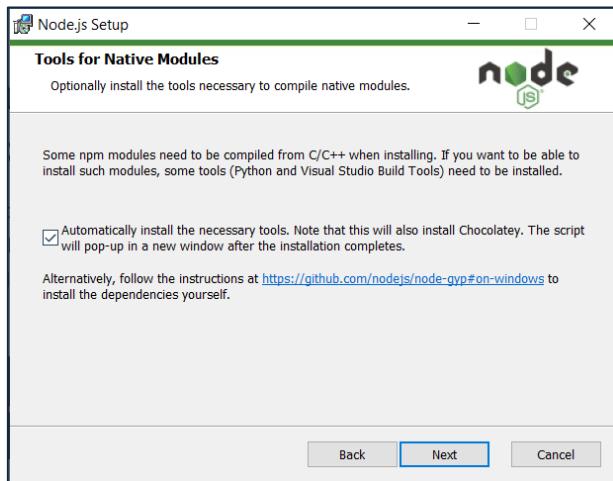


Figura 34: Instalación de Node.js - Herramientas necesarias

Tras esta última configuración, por fin se nos permitirá comenzar con la instalación, que deberemos autorizar con permisos de administrador del sistema. De haberse instalado correctamente, veremos el siguiente panel:

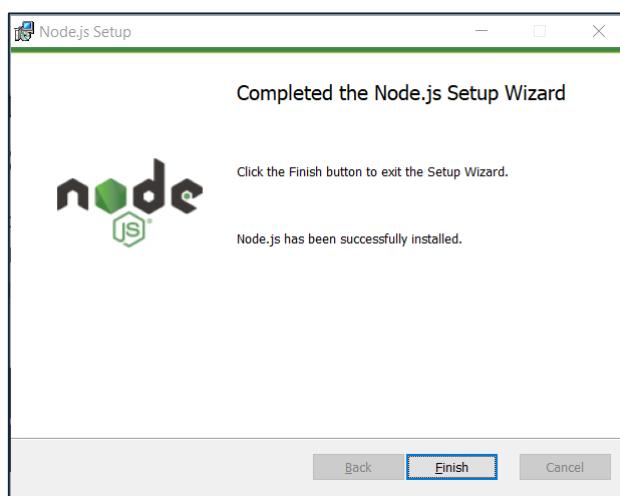


Figura 35: Instalación de Node.js - Instalación exitosa

# Manual de usuario: Aplicación web

Esta sección está dedicada a explicar al usuario corriente acerca de la funcionalidad disponible en la aplicación web, y cómo debe usarla para sacar su máximo rendimiento. Por usuario corriente se entiende cualquier individuo en posesión de un PC y que posea los conocimientos suficientes para operar con el dispositivo, sin ser necesarios conocimientos avanzados de informática.

## *Iniciar la aplicación*

En primer lugar, se deberá arrancar la aplicación web en caso de que no esté ya en funcionamiento, para ello se deberá abrir una consola de comandos Node.js. Para ello tenemos dos opciones:

- Utilizar la consola de Node.js, proporcionada por la instalación del entorno en el PC.
- Utilizar un terminal en Visual Studio Code.

La primera opción es abrir la consola de comandos que proporciona la instalación de Node.js en el PC. Es tan sencillo como dirigirse a la barra de búsqueda de Windows y escribir “Node.js command prompt”. El resultado esperado es el descrito en la Figura 36. Una vez aparezca el resultado deseado abrimos la consola haciendo clic con el ratón sobre la opción correspondiente (o pulsado la tecla Intro si la selección se encuentra sobre la opción).

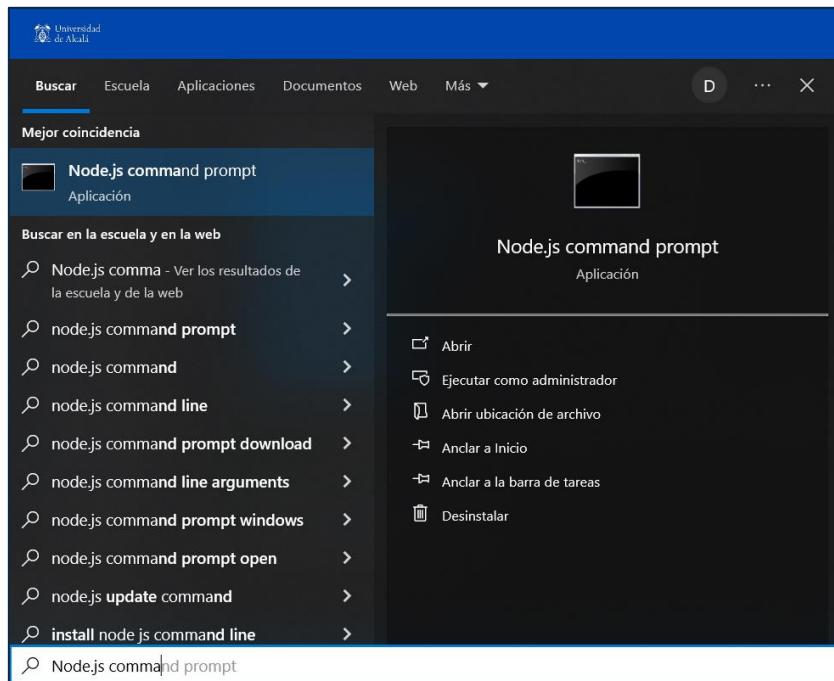
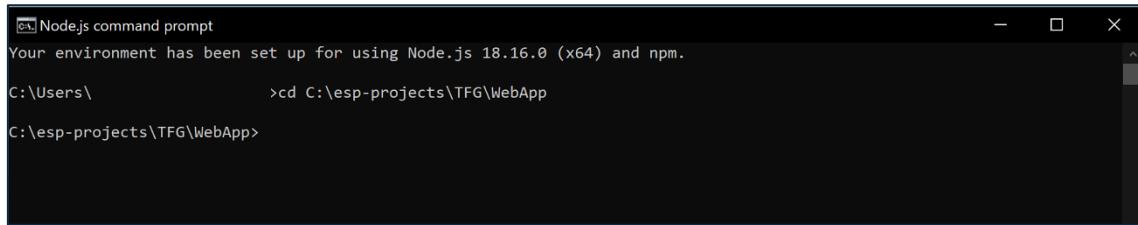


Figura 36: Consola de comandos de Node.js

De este modo, veremos cómo se abre la consola, y el siguiente paso sería desplazarnos hasta el directorio raíz que contiene los archivos del proyecto de la aplicación web. Para ello se utilizará el comando **cd**, incluyendo la ruta.



A screenshot of a Windows Command Prompt window titled "Nodejs command prompt". The window shows the following text:  
Your environment has been set up for using Node.js 18.16.0 (x64) and npm.  
C:\Users\ [user] >cd C:\esp-projects\TFG\WebApp  
C:\esp-projects\TFG\WebApp>

Figura 37: Consola Node.js - Cambio de directorio

A continuación, se debe escribir el comando para inicializar la aplicación web: >> npm run dev. Si se han seguido correctamente los pasos, ante el usuario debería encontrarse una pantalla similar a la de la Figura 38. La pantalla mostrada sirve como menú de operación sobre la aplicación web, con limitadas pero útiles opciones. Lo primero que se ve es la versión utilizada de Vite, junto con las direcciones que se deben utilizar para acceder a la aplicación web.



A screenshot of a Windows Command Prompt window titled "Seleccionar C:\Windows\system32\cmd.exe". The window shows the following text:  
VITE v5.2.12 ready in 159 ms  
[?] Local: http://localhost:5173/  
[?] Network: http://172.25.144.1:5173/  
[?] Network: http://192.168.216.210:5173/  
[?] press h + enter to show help

Figura 38: Consola Node.js - Aplicación web en funcionamiento

Si en la consola se escribe “h” se mostrarán las opciones disponibles, que son:

- Pulsar r + Intro para reiniciar el servidor
- Pulsar u + Intro para mostrar las direcciones
- Pulsar o + Intro para abrir en el navegador
- Pulsar c + Intro para limpiar la consola
- Pulsar q + Intro para salir y cerrar la aplicación

La otra opción disponible es utilizar el terminal del entorno de desarrollo Visual Studio Code. Para ello lo primero que hay que hacer es abrir la carpeta raíz del proyecto en el Explorador de archivos de Windows. A continuación, haciendo clic con el botón derecho del ratón sobre cualquier punto de la carpeta (no sobre los archivos) se mostrará un panel de opciones. En este panel la opción a seleccionar es “Abrir con Code”, precedida del icono azul característico del entorno de programación.

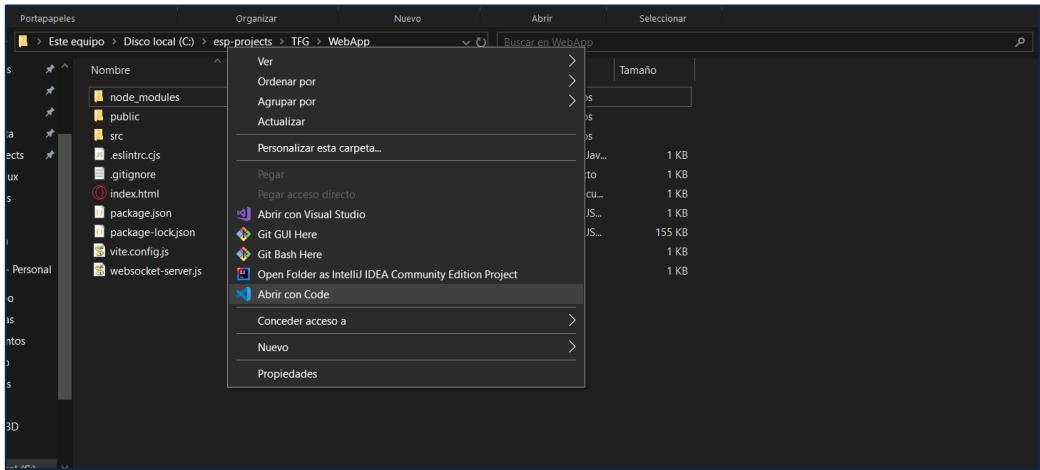


Figura 39: Explorador de archivos - Abrir carpeta con VS Code

Una vez seleccionada, se mostrará frente al usuario una pantalla como la descrita en la Figura 40. En el apartado “Terminal” del menú horizontal de la parte superior de la pantalla, elegir la opción “Nuevo Terminal”.

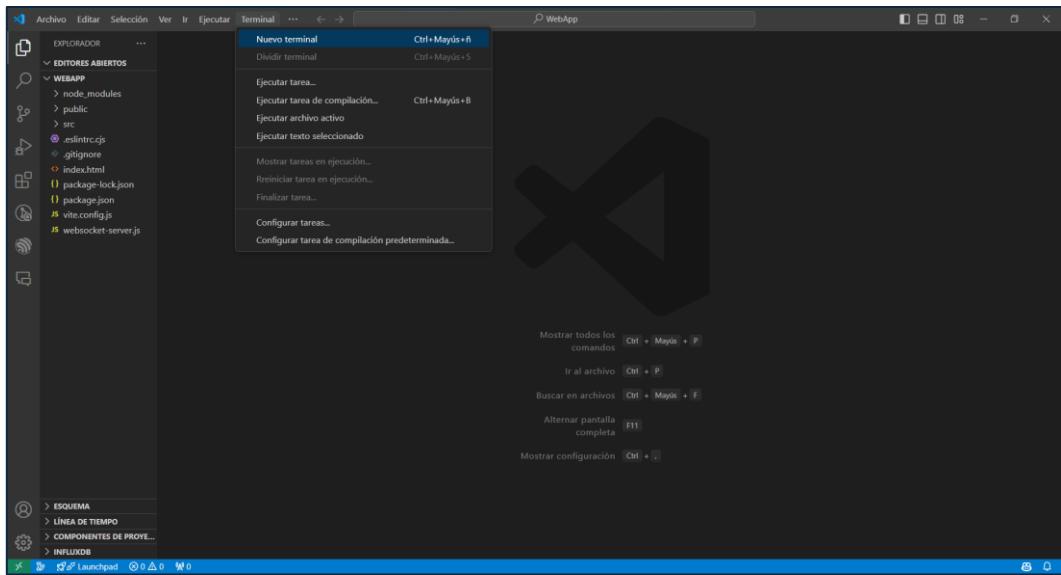


Figura 40: VS Code - Abrir nuevo terminal

Si se ha seguido los pasos correctamente, se habrá creado una consola en la parte inferior. Al abrir la carpeta raíz del proyecto con el entorno se evita el paso de tener que desplazarse con el comando **cd**, pudiendo directamente iniciar la aplicación con **npm run dev**.

Nota: “En caso de estar iniciado el servidor de Websockets, la aplicación web se conectará automáticamente al iniciarse, en caso contrario, si se inicia el servidor después de la aplicación,

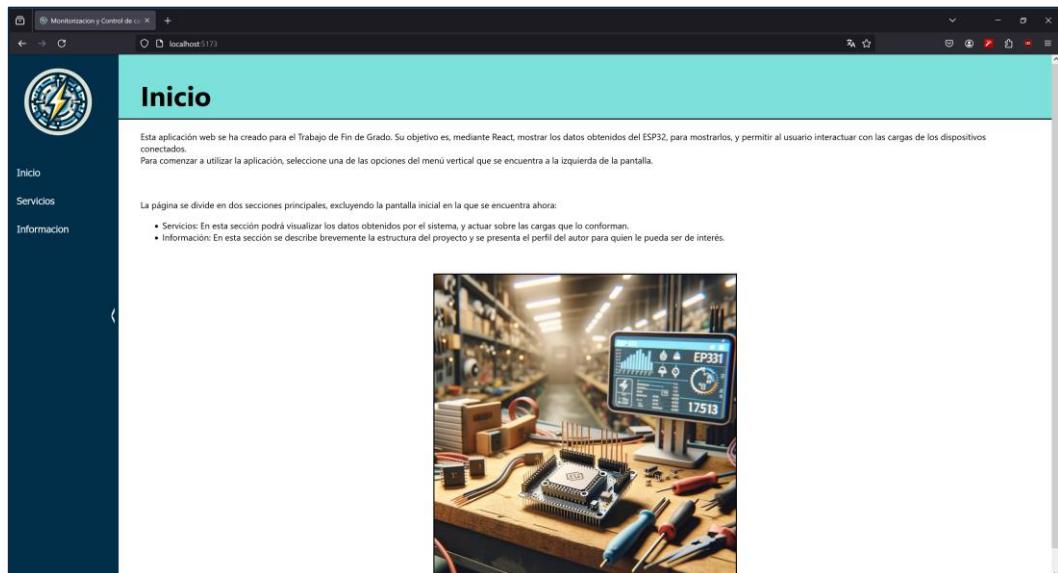
bastará con esperar a que esté funcionando el servidor y con refrescar la aplicación se intentará la conexión.”

## ***Apertura y uso de la aplicación***

Una vez tengamos iniciada y corriendo la aplicación web, para cargar la aplicación, el usuario deberá ir al navegador de su preferencia desde cualquier dispositivo que esté conectado a la misma red que el PC que aloja la aplicación. Una vez abierto, deberá buscar en el navegador la dirección de la aplicación, en este caso <http://192.168.216.210:5173/>.

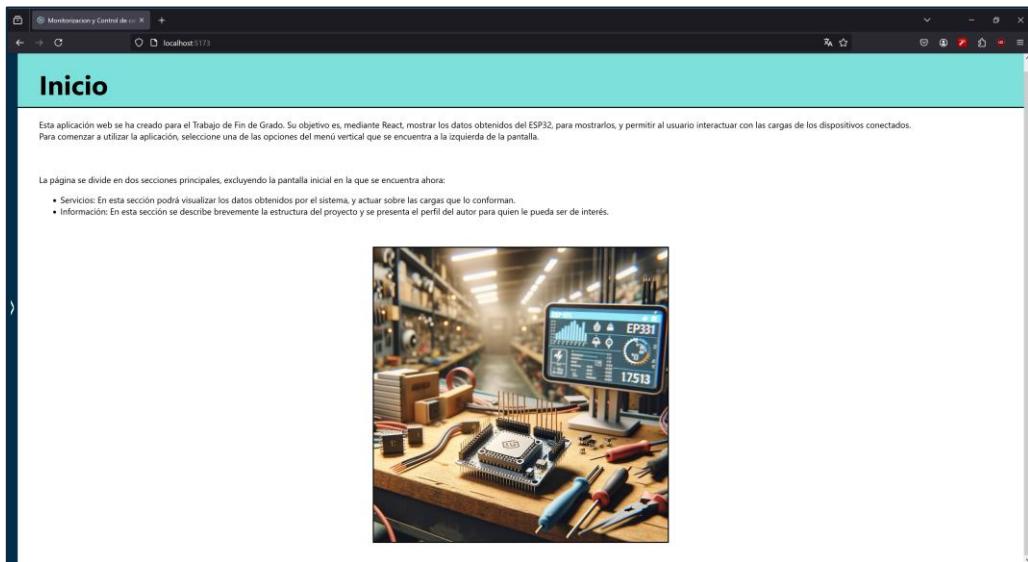
Nota: “En caso de alojar la aplicación en un servidor remoto, no será necesario compartir red con el servidor y bastará con abrir el navegador y utilizar la dirección de búsqueda correspondiente.”

Cuando la búsqueda se haya realizado, el usuario se encontrará frente a la pantalla descrita en la Figura 41, la página de inicio desde la que parte la aplicación web cada vez que se inicializa o recarga.



*Figura 41: Aplicación web - Pantalla principal*

A la izquierda de la pantalla se puede apreciar un menú vertical, el cual muestra las diferentes secciones en las que se divide la aplicación. Mediante el uso del ratón y este menú podremos navegar por la aplicación libremente. Servicios e información representan títulos de sección, y al pasar el ratón sobre ellos, se desplegarán otros botones, que dan acceso a nuevas secciones. Este menú cuenta con dos posiciones, desplegado y recogido, siendo desplegada la posición por defecto, y cambiando a la posición recogida pulsando el botón en forma de flecha hacia la izquierda (“<”) del menú.



*Figura 42: Aplicación web - Pantalla principal con el menú oculto*

El otro componente principal de la aplicación es el panel central que ocupa el resto de la pantalla, comenzando donde acaba el borde derecho del menú vertical

- La sección **Inicio** es en la que se encuentra actualmente el usuario. Cuenta con una única vista, que es la que se muestra al cargar y refrescar la aplicación.
- La sección **Servicios**, en ella se encuentra la funcionalidad relacionada con la monitorización. Esta está formada a su vez por tres vistas:
  - La vista **Datos**, que muestra una tabla con las mediciones en tiempo real según se van recibiendo.
  - La vista **Gráficas**, en la que se presenta una gráfica por cada parámetro eléctrico de la medición. Estas se actualizan en tiempo real, añadiendo nodos nuevos según se reciban los datos.
  - La vista **Control Cargas** es donde se permite al usuario actuar de forma directa sobre el sistema. Si el usuario activa o desactiva una o varias cargas, el resultado se reflejará físicamente de forma inmediata.
- Finalmente, la sección **Información**, en la que se ofrece información acerca del proyecto y su autor.
  - La vista **Sobre el proyecto** contiene información referente a la estructura del proyecto, incluyendo un breve resumen de funcionalidad de cada elemento del sistema, así como un diagrama general de este.
  - La vista **Sobre mí** sirve como presentación para conocer un poco más acerca del autor del proyecto, sus conocimientos y habilidades.

Eligiendo **Datos** en el menú vertical pasaremos a una de las secciones de la aplicación en las que la interacción que realiza el usuario es visualización de información, siendo la otra la sección **Gráficas**.

La sección de datos se compone de dos elementos: un fragmento de texto a modo de introducción sobre la sección, donde se indican los parámetros recogidos por el sistema, y una tabla donde aparecen las mediciones en el momento en que son recibidas por la aplicación, de modo que la primera entrada de la tabla representa el dato más antiguo recibido, y la entrada que encontraremos más abajo en la tabla se corresponde con el dato más reciente recibido. Para confirmar esto basta con revisar las columnas de la tabla que reflejan los valores temporales de la medición (la fecha y hora representan el momento de medición, no el de recepción en la aplicación).

ID	Potencia (W)	Voltaje (V)	Intensidad (A)	Energía	Frecuencia	Factor de potencia	Fecha	Hora
ESP32_1	6.7	238.3	0.05	0.06	50	0.53	16-06-2024	18:50:19
ESP32_1	6.7	238.9	0.05	0.06	50	0.53	16-06-2024	18:50:22
ESP32_1	6.7	238.9	0.05	0.06	50	0.53	16-06-2024	18:50:25

Figura 43: Aplicación web - Pantalla Datos

Como puede verse en la Figura 43, la tabla está compuesta de 9 columnas, que incluyen:

- La cadena alfanumérica de identificación de la placa que envía las mediciones,
- los seis parámetros eléctricos obtenidos por cada medición,
- y la fecha y hora correspondientes al momento en que se realizó la medición.

En caso de ver que no aparecen o entran datos, y se tiene la seguridad de que se están enviando, refrescar la página debería solucionar cualquier error de conexión con el servidor de Websockets.

La otra sección correspondiente a visualización de datos es la correspondiente a **Gráficas**. Como su nombre indica es una vista compuesta por una gráfica por cada parámetro eléctrico de los datos recibidos por la aplicación. Por cada entrada que se añade en la tabla, se toman los parámetros eléctricos, y cada uno se introduce como el último nodo dentro de la gráfica correspondiente, actualizando visualmente la función representada en el gráfico.

Todas las gráficas toman como variable independiente (eje horizontal) la hora en que se hizo la medición, y como variable dependiente (eje vertical) un rango de valores de la unidad correspondiente al parámetro registrado en la gráfica.

En caso de la **Potencia** eléctrica la unidad empleada son los Vatios (W).



Figura 44: Aplicación web - Gráfica Potencia

Para el **Voltaje** la unidad empleada son los Voltios (V).

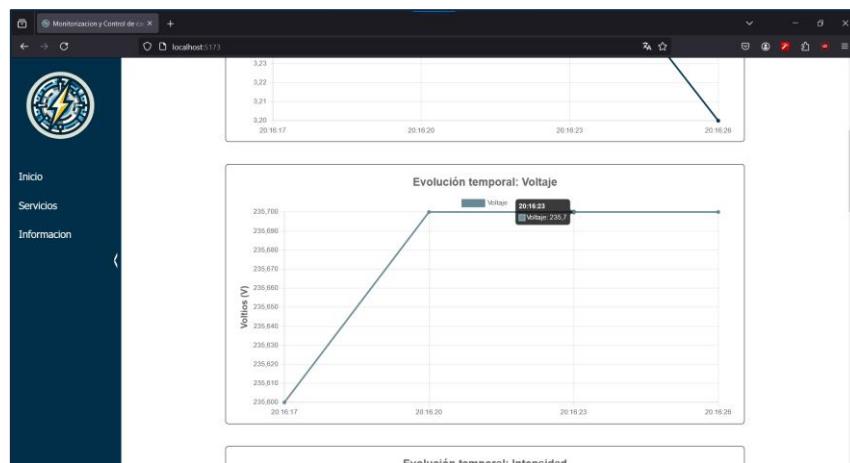


Figura 45: Aplicación web - Gráfica Voltaje

En el caso de la **Intensidad**, la unidad escogida son los Amperios (A).

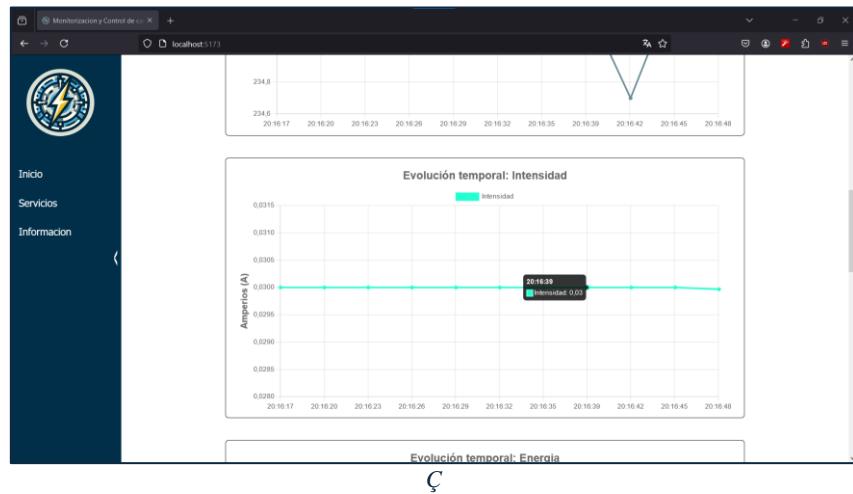


Figura 46: Aplicación web - Gráfica Intensidad

La unidad para medir la **Energía** serán los Kilovatios por hora (KWh).

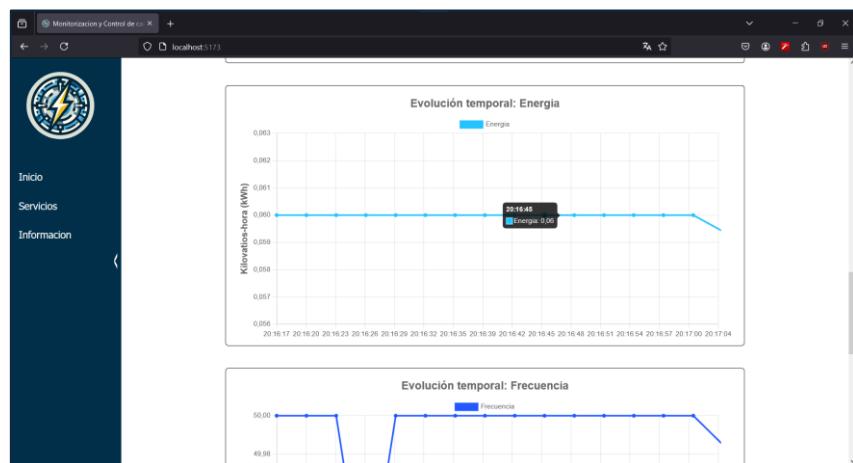


Figura 47: Aplicación web - Gráfica Energía

La **Frecuencia** se registrará en Hercios (Hz).

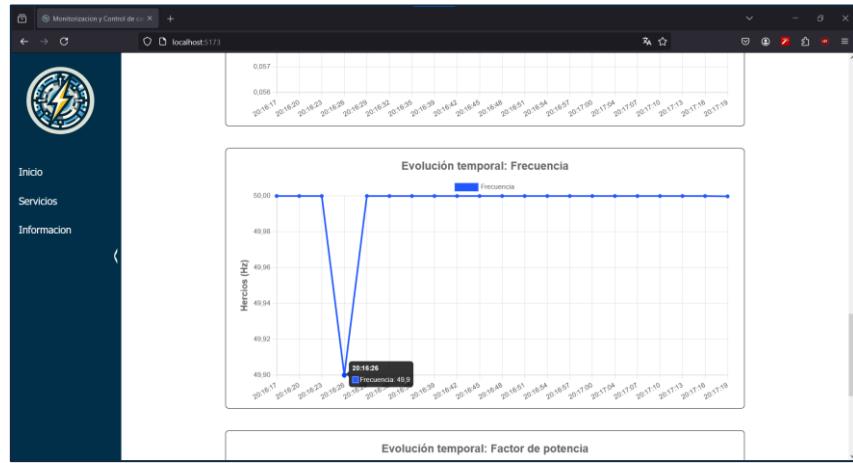


Figura 48: Aplicación web - Gráfica Frecuencia

Y finalmente, el **Factor de potencia** se mide como un valor decimal que oscila entre 0 y 1.

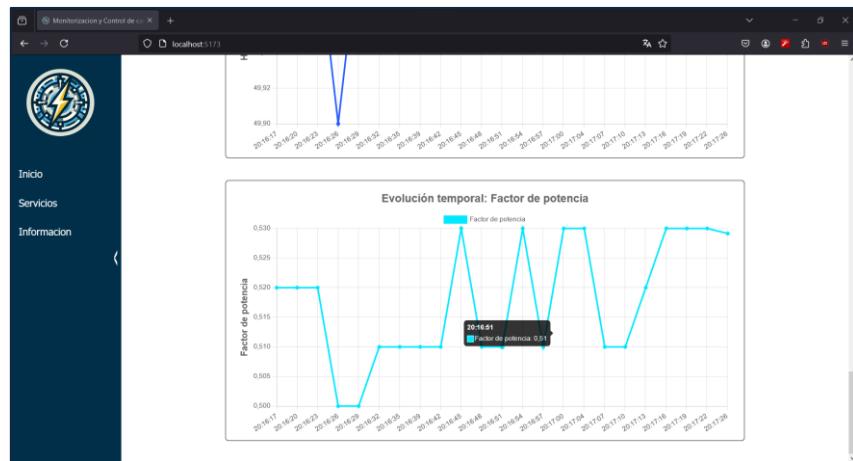


Figura 49: Aplicación web - Gráfica Factor de Potencia

Una vez descritas las dos secciones de visualización de datos, la única vista restante de la sección **Datos** es el apartado de **Control Cargas**. Como se menciona al inicio del manual, esta es la única sección con capacidad de acción sobre el circuito. Pulsando los botones “Apagar” y “Encender” se desactiva o se activa, respectivamente, la carga bajo el mando de dichos botones.

La vista se compone de un breve fragmento a modo de introducción y de un panel con iconos representando las salidas del circuito. El encender o apagar las cargas mediante los botones permite o corta el flujo de la corriente hacia las salidas del circuito.

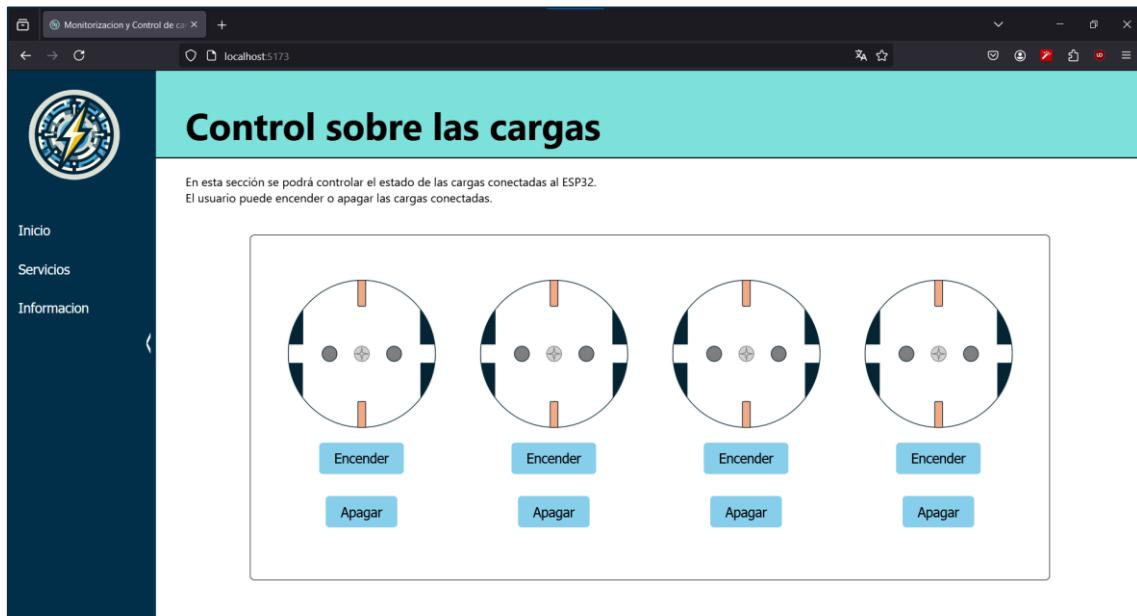


Figura 50: Aplicación web - Control sobre las cargas

La sección restante, **Información**, tiene como propósito ofrecer:

- Detalles sobre el sistema que el usuario está usando.
- Información sobre el autor que diseñó e implementó la solución que tiene frente a sí.

Comenzando por la vista **Sobre el proyecto**, en ella se puede ver un diagrama representando la estructura del sistema, con los componentes como nodos, y flechas indicando las conexiones y el flujo de información en el sistema, y sobre el diagrama, se incluye un texto donde se resume esta estructura.

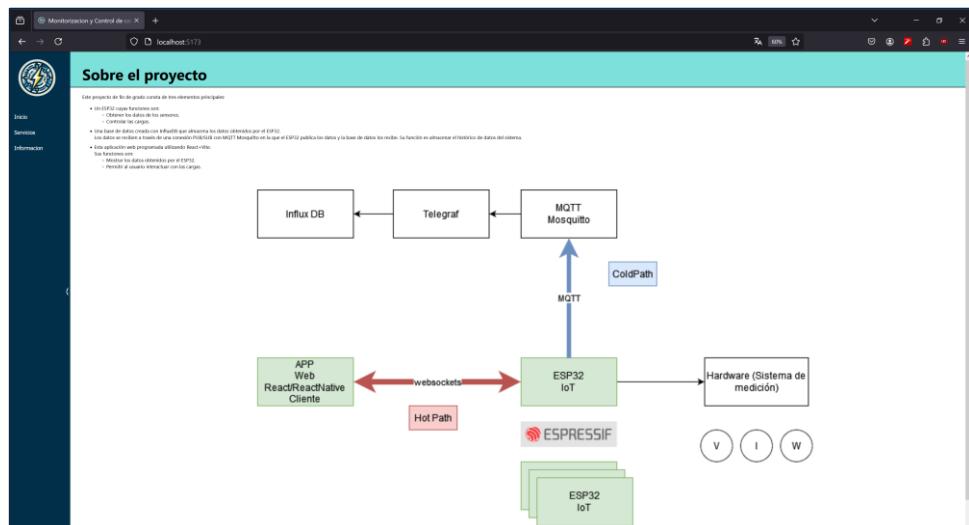


Figura 51: Aplicación web - Sobre el proyecto

Como última vista de la aplicación tenemos **Sobre mí**, un pequeño apartado dedicado a dar a conocer un poco a la persona que hay detrás de horas de investigación, diseño e implementación del sistema. En ella se reflejan los idiomas conocidos, y las aptitudes y conocimientos relacionados con la programación. Se incluye también un botón que permite descargar un CV con un perfil mucho más detallado. Finalmente, en la parte inferior se mencionan métodos de contacto y redes sociales.

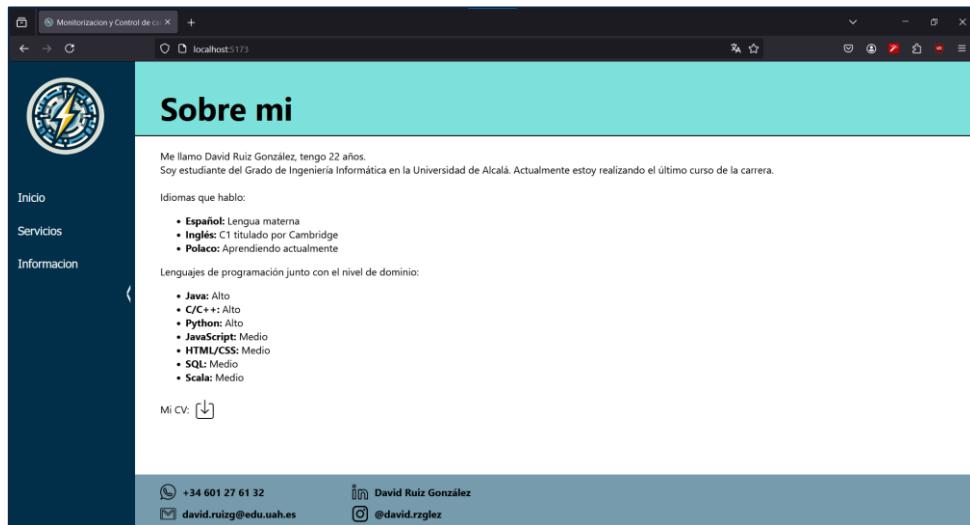


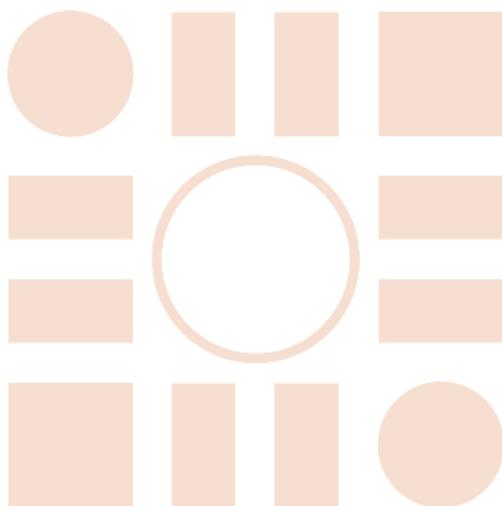
Figura 52: Aplicación web - Sobre mi

Con esto termina el manual de usuario de la aplicación web, habiendo cubierto toda la funcionalidad implementada hasta el momento. En caso de que en el futuro se publiquen nuevas versiones con actualizaciones, se ampliará y revisará este documento.





Universidad de Alcalá  
Escuela Politécnica Superior



ESCUELA POLITECNICA  
SUPERIOR

