ULPGC
Universidad de
Las Palmas de
Gran Canaria

ESCUELA DE INGENIERÍA DE

TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

DESARROLLO DE LOS SISTEMAS DE MEDICIÓN, COMUNICACIÓN Y PRESENTACION EN EL PROYECTO COLONIUM

Titulación: Grado en Ingeniería en Tecnologías de la Telecomunicación

Mención: Sistemas de Telecomunicaciones

Autor/a: Jesús Borobia Sánchez

Tutor/a: Miguel Ángel Quintana Suárez

Fecha: junio 2023

ULPGC
Universidad de
Las Palmas de
Gran Canaria

ESCUELA DE INGENIERÍA DE

TELECOMUNICACIÓN Y ELECTRÓNICA

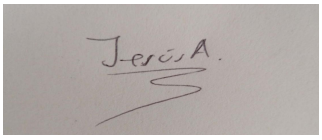
TRABAJO FIN DE GRADO

DESARROLLO DE LOS SISTEMAS DE MEDICIÓN, COMUNICACIÓN Y PRESENTACION EN EL PROYECTO COLONIUM

HOJA DE FIRMAS

Alumno:

Jesús Borobia Sánchez



Tutor:

Miguel Ángel Quintana Suárez

Firmado por QUINTANA SUAREZ, MIGUEL ANGEL (FIRMA) el día 19/06/2023 con un certificado emitido por AC DNIE 005

Fecha: junio 2023

ULPGC
Universidad de
Las Palmas de
Gran Canaria

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

DESARROLLO DE LOS SISTEMAS DE MEDICIÓN, COMUNICACIÓN Y PRESENTACION EN EL PROYECTO COLONIUM

HOJA DE EVALUACIÓN

Calificación:_____

Presidente	**Secretario**	**Vocal**
Fdo.:	Fdo.:	Fdo.:

Fecha: junio 2023

Agradecimientos

En primer lugar a mi familia y amigos, que me han apoyado desde la lejanía de mi casa dándome fuerzas y ánimos cuando más lo necesitaba. También quisiera agradecer a toda la gente que he conocido durante este año en la isla, gente buena que se ha preocupado por mí y me han acogido como a uno más. A mis compañeros de clase y demás gente de la universidad, que de alguna u otra forma me hayan ayudado durante este curso. También querría dar las gracias por supuesto al Colegio Sagrado Corazón de Tafira y a Antonio Ramírez, que me dieron la oportunidad de realizar este proyecto y que me han dado los ánimos y todas las herramientas a su disposición para seguir con la tarea propuesta y solventar los tropiezos del camino. Finalmente también quisiera agradecerle a mi tutor, Miguel Ángel Quintana, por ayudarme cuando ha sido necesario, enseñándome y proponiéndome soluciones como ingeniero. Sin la ayuda de todas estas personas y el esfuerzo personal, este documento tendría las páginas en blanco.

Resumen

Desde el colegio Sagrado Corazón de Tafira se propone el proyecto Colonium. Este proyecto consiste en lanzar al océano un barco de madera a escala de una típica nave de Lanzarote llamada Goleta. Durante su trayecto el barco irá realizando periódicamente una serie de medidas, como son: temperatura del aire y del agua, velocidad del viento, posición...etc. El barco se comunicará vía satélite para enviar los datos recogidos a un servidor y que éstos puedan ser puestos a disposición de la comunidad educativa del colegio en un portal web.

En este TFT se realizarán los diseños de cada uno de los bloques necesarios para que el sistema completo funcione. Los bloques a desarrollar son: el bloque de medición, donde se realizarán las medidas a través de los sensores; el bloque de comunicación donde se transmitirá la información recogida previamente vía satélite; y el bloque de presentación, donde se implementará un portal web capaz de recibir y guardar los datos enviados desde el barco, así como mostrarlos de manera adecuada través de la web.

Abstract

From the school Sagrado Corazón de Tafira the Colonium project is proposed. This project consists of launching into the ocean a wooden ship to scale of a typical ship of Lanzarote called Goleta. During its voyage the ship will periodically take a series of measurements, such as: air and water temperature, wind speed, position...etc. The ship will communicate via satellite to send the collected data to a server so that they can be made available to the educational community of the school in a web portal.

In this TFT the designs of each of the blocks necessary for the complete system to work will be made. The blocks to be developed are the measurement block, where the measurements will be taken through the sensors; the communication block where the information previously collected via satellite will be transmitted; and the presentation block, where a web portal capable of receiving and storing the data sent from the ship, as well as displaying them properly through the web, will be implemented.

Contenido

Agradecimientos I

Resumen III

Abstract V

Contenido VII

Índice de figuras XI

Índice de tablas XIII

Índice de ecuaciones XIII

Índice de acrónimos XV

Capítulo 1 Introducción 1

1.1 Introducción 1

1.2 Objetivos 2

1.3 Medios Materiales 2

1.4 Estructura del documento 3

Capítulo 2 Fundamentos teóricos 5

2.1 Sistemas empotrados 5

2.2 Comunicación satelital y red Iridium 7

2.3 Fundamentos de WordPress y programación web 11

Capítulo 3 Diseño del sistema 15

3.1 Descripción general del sistema propuesto 15

3.2 Componentes del sistema 16

3.2.1 Componentes del bloque de medición 17

3.2.2 Componentes del bloque de comunicaciones 23

3.2.3 Componentes del bloque de presentación 24

3.3 Selección de la ESP32 y sus características 24

Capítulo 4 Bloque de medición 29

4.1 Funcionamiento de los sensores utilizados 29

4.2 Watchdog y modo de hibernación 30

4.3 Rutina de funcionamiento 31

Capítulo 5 Bloque de comunicación 37

5.1 Consideraciones iniciales 37

5.2 Módulo ROCKBLOCK+ 38

5.3 Implementación y envío de datos 39

Capítulo 6 Bloque de presentación 43

6.1 Configuración básica de WordPress 43

6.2 Definición de la tabla en la base de datos 45

6.3 API para recepción de datos 48

6.4 Presentación de los datos 51

Capítulo 7 Resultados y conclusiones 55

7.1 Resultados 55

7.2 Conclusiones 57

Bibliografía 59

ANEXOS 61

Anexo I. Código final ESP32 con bloque Testing 62

Anexo II. Formulario para añadir datos de forma manual. 68

Anexo III. Endpoint y recepción solicitudes HTTP POST 69

Anexo IV. Generación de Gráfica. Ejemplo con Temp1 71

Anexo V. Generación del mapa. 73

PRESUPUESTO 75

P.1 .Introducción 76

P.2 .Recursos hardware 76

P.3 .Recursos software 77

P.4. Recursos humanos 77

P.5. Redacción del documento 78

P.6. Aplicación de impuestos y coste final 78

Índice de figuras

Figura 1. Esquema básico en un enlace de telecomunicación 8
Figura 2. Diagrama de bloques del sistema 16
Figura 3. Diagrama del bloque de alimentación 17
Figura 4. Esquema e imagen del sensor DS18B20 18
Figura 5. Anemómetro de tres tazas 20
Figura 6. Módulo Beitian-BN880 21
Figura 7. Módulo ROCKBLOCK+ y pinout USB 23
Figura 8. Pinout ESP32 26
Figura 9. Conexionado de los bloques de medición y comunicación 30
Figura 10. Máquina de estados del bloque de medición 34
Figura 11. Envío de datos por Bluetooth 36
Figura 12. Parámetros enviados por el ROCKBLOCK+ 38
Figura 13. Bloque de testing para comunicación con ROCKBLOCK 41
Figura 14. Recepción mensajes ROCKBLOCK a email. 42
Figura 15. Menú de servicios de Infinity Host 43
Figura 16. Panel de Instalación de WordPress 44
Figura 17. Ejemplo Plantilla WordPress 45
Figura 18. Menú de configuración phpMyAdmin 46
Figura 19. Configuración de variables de la base de datos 47
Figura 20. Archivos del host 48
Figura 21. Posibles errores al acceder sin permiso 49
Figura 22. Solicitud HTTP Post en POSTMAN 50
Figura 23. Gráfica con datos añadidos manualmente 52
Figura 24. Mapa con coordenadas y desplegable para visualización de datos 53
Figura 25. Datos recibidos a la web final 56
Figura 26. Gráfica con resultados finales 56

Índice de tablas

Tabla 1. Características sensor D18B20 18
Tabla 2. Características anemómetro 19
Tabla 3. Características Beitian 21
Tabla 4. Características ESP32 26
Tabla 5. Cálculo de consumo de tokens 40
Tabla 6. Columnas de la base de datos 46
Tabla 7. Recursos hardware y vida útil 76
Tabla 8. Coste de recursos hardware 76
Tabla 9. Recursos software y vida útil. 77
Tabla 10. Coste de recursos software 77
Tabla 11. Costes de redacción del documento 78
Tabla 12. Coste total tras impuestos 78

Índice de ecuaciones

Ecuación 1. Teorema de Shannon-Hartley 8
Ecuación 2. Ecuación de la relación portadora/ruido 9
Ecuación 3. Relación energía de bit con densidad de ruido 9
Ecuación 4. Eb/No en escala decibélica 9
Ecuación 5. Ecuación PIRE 9
Ecuación 6. Figura de mérito 10

Índice de acrónimos

TFT: Trabajo de Fin de Título

GPS: *Global Positioning System*

IoT: *Internet of Things*

PIRE: Potencia isotrópica radiada efectiva

API: *Application programming interface*

MPPT: *Maximum Power Point Tracking*

PHP: *Hypertext Pre-Processor*

HTML: *Hypertext Markup Language*

SQL: *Structured Query Language*

I2C: *Inter-Integrated Circuit*

UART: *Universal Asynchronous Receiver / Transmitter*

RTC: *Real Time Clock*

Capítulo 1 Introducción

1.1 Introducción

Este TFT parte de la base del realizado por Gabriel Ojeda Suárez [1], en él se describían los antecedentes y el impacto del proyecto sobre el alumnado. En ese TFT se diseñaba de forma parcial el sistema de recolección y transmisión de datos en los sensores además de plantear distintas tareas desde el punto de vista del proyecto Colonium como proyecto educativo y el análisis de la travesía de la Goleta.

Se rediseñará el software utilizado por el microcontrolador para adaptarlo a un sistema de control en tiempo real, prestando especial atención a los ciclos de hibernación y recuperación frente a bloqueos o errores inesperados. Además de completarlo con la funcionalidad de comunicación satelital y publicación de resultados a través de un portal web.

En primer lugar, se analizará el sistema de medición que realizará el barco durante su trayecto. Se utilizará un microcontrolador de la familia del ESP32 junto a varios sensores para medir las magnitudes como la temperatura ambiente, la temperatura del agua, la velocidad del viento o la posición GPS. En este sistema se debe tener en cuenta la alimentación del sistema para optimizar el uso de la energía. La comunicación entre el microcontrolador utilizado y los diferentes sensores se realizará mediante una comunicación a través del protocolo correspondiente.

En segundo lugar, se debe establecer la comunicación satelital. En este caso se utilizará la red Iridium para enviar los mensajes con los datos recogidos desde el barco por lo que se añadirá un módulo Iridium al barco. Mediante un módulo de comunicación ROCKBLOCK se establecerá la conexión necesaria para enviar los datos vía satélite.

Finalmente, el desarrollo web será mediante WordPress e incluirá *plugins* dedicados para mostrar los datos recogidos en diversas gráficas, así como la ubicación GPS y comentarios sobre los datos, así como otras posibles mejoras. Los alumnos y demás usuarios podrán acceder a los datos para utilizarlos en sus propios trabajos.

1.2 Objetivos

El objetivo de este proyecto es la implementación de los 3 bloques: medición, comunicación satelital y presentación, teniendo en cuenta las especificaciones de los componentes a utilizar y sus limitaciones.

O1. Diseño e implementación del bloque de toma de medidas en el barco.

Se justificará la elección de los distintos elementos de este bloque así como se realizarán las conexiones entre ellos y se programará la rutina de ejecución para su correcto funcionamiento.

O2. Envío y recepción de datos vía satélite.

Se justificará la elección de los componentes utilizados para el bloque de comunicación además de programar su comportamiento, este bloque permitirá la comunicación por satélite y la posterior recepción de los mensajes en el formato deseado.

O3. Tratamiento de datos y diseño web.

Se analizará el flujo de datos del enlace y se aplicará algún mecanismo de optimización de los datos a enviar. Se estudiará cómo almacenarlos y mostrarlos a través de una web.

O4. Presentación y consideraciones con respecto al proyecto Colonium

Se presentará el sistema en su conjunto, teniendo en cuenta las restricciones del medio en el que funcionará y habiendo conectado los bloques, comprobando la funcionalidad completa.

1.3 Medios Materiales

Para la realización de este TFT se requiere de recursos hardware y software. Comenzando por los hardware, se necesitará un PC en el cual poder acceder a información a través de internet y a recursos software como un IDE donde programar el microcontrolador o herramientas de redacción de documentos, así como una plataforma de *host* para WordPress. El resto de los recursos hardware serán necesarios para los distintos bloques del sistema, el panel fotovoltaico junto al MPPT y la batería servirán para alimentar el sistema completo. También se utilizará el microcontrolador ESP32 al que se le conectarán los sensores DS18B20 (sensor de temperatura), un anemómetro y un Módulo GPS.

Sin los elementos que se encuentran en la siguiente lista sería imposible la realización de este TFT tal y como se plantea.

- Recurso Hardware:
  - Ordenador personal ASUS
  - Panel Fotovoltaico
  - MPPT
  - Batería 12V
  - ESP32-Espressif
  - Anemómetro
  - Sensor de temperatura DS18B20
  - Módulo GPS Beitian
  - Unidad Rockblock+
- Recurso Software:
  - Visual Studio Code con extensión PlatformIO
  - Infinity Host
  - WordPress
  - Microsoft Office
  - Herramientas de búsqueda bibliográfica (Mendeley)

1.4 Estructura del documento

Tras comentar la estructura general del proyecto así como sus objetivos y nombrar los principales componentes, en el capítulo 2 se comentarán los fundamentos teóricos aplicados en cada uno de los bloques del proyecto.

En el Capítulo 3 se realizará diseño del sistema completo, eligiendo los componentes y comparando con otras alternativas posibles, justificando las elecciones tomadas.

En el Capítulo 4 se explicará la implementación del bloque de medición del sistema, explicando el código utilizado para controlar el comportamiento de cada componente y los métodos de control de errores.

En el Capítulo 5 se estudiará la implementación del bloque de comunicación en el código, entendiendo el formato de los mensajes enviados y las posibles formas de transmisión.

En el Capítulo 6 se estudiará la implementación del bloque de presentación, se explicarán los *plugins* utilizados y la configuración del WordPress para la creación de una web accesible para cualquier usuario.

En el Capítulo 7 se recogerán los resultados obtenidos para el sistema final y se obtendrán las conclusiones.

En los Anexos se pueden encontrar todos los fragmentos de código que han sido necesarios para la implementación de los distintos bloques.

Finalmente se puede encontrar el presupuesto para la realización de este TFT teniendo en cuenta gastos en hardware, software y mano de obra.

Capítulo 2 Fundamentos teóricos

En el Capítulo 2 se estudiarán los fundamentos teóricos que se aplicarán posteriormente en el proyecto.

2.1 Sistemas empotrados

Un sistema empotrado es un sistema informático de uso específico, este sistema es una combinación entre *software* y *hardware* que llevarán a cabo la funcionalidad del sistema. Se denominan sistemas empotrados o embebidos puesto que normalmente se encuentran formando parte de un sistema completo.

Este tipo de sistemas normalmente están basados en un microcontrolador, que podrá ser programado. El usuario puede modificar el comportamiento del microcontrolador para cambiar el comportamiento o las acciones que pueden suceder a la hora de realizar sus tareas.

El avance de las técnicas de diseño de este tipo de sistemas ha permitido el desarrollo de productos de mejores especificaciones entre las que se encuentran el tamaño, la velocidad de procesado o la robustez, así como el precio. En la década de los 70 se encuentran los primeros microcontroladores de 8 bits [2] fabricados por Motorola e Intel, además de las primera memorias programables, esto hizo que las aplicaciones pudieran tener mayor extensión en cuanto a líneas de código así como número de variables y complejidad de operaciones. En aquella época era común el uso del lenguaje de ensamblador, que se sustituyó por lenguajes de mayor nivel con el paso del tiempo y el desarrollo de las tecnologías.

A lo largo de las décadas de los 80 y 90 se introdujeron nuevas herramientas como un temporizador o interrupciones programables. Empresas como IBM o Microchip comenzaron a desarrollar sus propios modelos introduciendo mejoras como la ampliación de la memoria y la reducción del tamaño.

En la actualidad, el campo de los microcontroladores y sistemas embebidos ha experimentado avances significativos en diversas áreas debido, entre otras razones, a los grandes avances conseguidos en el campo de la electrónica y de la evolución de la informática en los últimos 20 años.

Los microcontroladores y sistemas embebidos modernos han logrado un gran progreso en términos de eficiencia energética. Los fabricantes han desarrollado técnicas avanzadas de administración de energía, como modos de bajo consumo y control de voltaje dinámico, para minimizar el consumo de energía en aplicaciones de batería y prolongar la vida útil de los dispositivos.

Por otro lado, hoy en día, los microcontroladores están diseñados para ofrecer una amplia gama de funcionalidades puesto que además de las capacidades de procesamiento central, incluyen periféricos integrados como convertidores analógico-digitales y digital-analógicos, interfaces de comunicación, controladores de temporización...etc. Esta integración permite a los diseñadores desarrollar sistemas complejos utilizando un solo microcontrolador que controle varios periféricos.

Los microcontroladores modernos están equipados con interfaces de comunicación inalámbrica, como Wi-Fi, Bluetooth y Zigbee, lo que les permite conectarse y comunicarse con otros dispositivos y servicios en la red. Esto abre un amplio abanico de aplicaciones en áreas como automatización del hogar, monitoreo remoto, sistemas de seguridad y más.

Con el desarrollo de la informática ya no es necesario programar el microcontrolador a nivel de ensamblador sino que se han creado entornos de desarrollo integrados, junto con bibliotecas y *middleware* que simplifican el desarrollo de aplicaciones.

Algunos microcontroladores ahora incluyen unidades de procesamiento digital de señales y aceleradores de cálculo de aprendizaje automático integrados para permitir tareas como filtrado, reconocimiento de voz, detección de objetos y más, directamente en el microcontrolador sin requerir un procesamiento externo.

Se deben tener varios criterios a la hora de diseñar el sistema embebido, como la jerarquía de los distintos componentes del sistema como pueden ser los procesadores, las memorias o los sensores y actuadores además del propio microcontrolador. Se debe tener en cuenta la prioridad de actuación de los distintos componentes, que será controlada por el microcontrolador, construyendo así un sistema con una arquitectura del tipo maestro-esclavo entre el microcontrolador y los distintos periféricos a utilizar.

En este TFT se hará uso de un microcontrolador ESP32 de Espressif Systems junto a varios sensores. Las especificaciones de este modelo así como la programación del comportamiento de los periféricos a utilizar serán descritos más adelante, cuando se describa el sistema completo para la aplicación a desarrollar.

En resumen, los sistemas embebidos basados en microcontrolador tienen diversas aplicaciones en todo tipo de ámbitos que van desde electrodomésticos, pasando por la automoción hasta llegar a tecnologías militares y aplicaciones aeroespaciales.

2.2 Comunicación satelital y red Iridium

La comunicación vía satélite es un sistema de transmisión de señales que utiliza satélites artificiales que orbitan alrededor de la Tierra. Esta tecnología fue un gran avance en las comunicaciones a nivel mundial puesto que permite la comunicación desde cualquier punto del planeta sin necesidad de cobertura como ocurre en las comunicaciones móviles modernas, por otro lado también trae sus desventajas que serán comentadas más adelante.

Tras los primeros lanzamientos de satélites artificiales en la carrera espacial a lo largo de la década de los 50, en 1963[3], la organización estadounidense de investigación y desarrollo ARPA lanzó el primer satélite de comunicaciones , que permitía la transmisión en tiempo real de señales de TV, telefonía y datos en Europa y América del Norte. A partir de dicho evento diversas organizaciones espaciales continuaron con sus lanzamientos y puestas en órbita.

A lo largo de las décadas esta tecnología ha ido evolucionando considerablemente, los satélites modernos utilizan frecuencias de radio y microondas para transmitir una amplia gama de señales, incluyendo voz, video, datos e Internet. Además, se han desarrollado sistemas más avanzados que permiten una mayor capacidad de transmisión y una menor latencia, lo que ha mejorado la calidad y la velocidad de las comunicaciones vía satélite.

Se deben tener en cuenta varios factores a la hora de establecer un enlace que comunique por ejemplo una unidad móvil con un satélite, en la Figura 1 se puede ver el esquema básico de un enlace de telecomunicaciones:

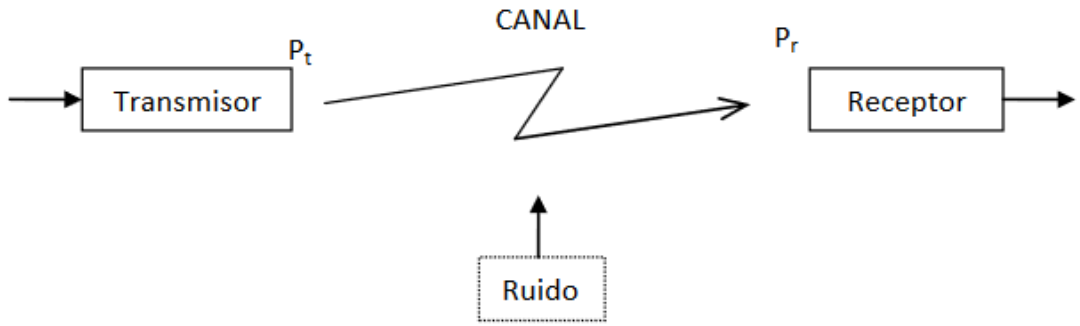


Figura 1. Esquema básico en un enlace de telecomunicación

El transmisor amplifica la señal para ser radiada por la antena, en el canal se tendrá un declive de la potencia y el receptor debe amplificarla sin introducir ruido. Las variables que limitan un enlace vía satélite son la potencia y el ruido, descrito por el teorema de la capacidad de canal de Shannon-Hartley [4] , que es la máxima tasa de información que se puede transmitir en determinado ancho de banda.

$$R \leq C = W\ log_2\left(1 + \frac{S}{N}\right)$$

Ecuación 1. Teorema de Shannon-Hartley

Donde R es la tasa de transmisión en bits/s, C es la capacidad del canal en bits/s, W es el ancho de banda en Hz y S/N es la denominada relación señal a ruido, que es la relación entre la potencia de la señal recibida y la potencia del ruido medido en la recepción.

Se deben tener en cuenta también todas la pérdidas que habrá a lo largo del enlace, como las pérdidas de espacio libre, la absorción atmosférica o la atenuación de la posible lluvia. Además la despolarización también puede ser considerada como una pérdida extra puesto que si se cambia la dirección de los campos que se propagan a través de la onda, la antena receptora no recibe la mayor potencia posible.

Uno de los parámetros importantes a la hora de establecer un enlace satelital es la relación C/N (*carrier/noise*) que relaciona la potencia de la portadora y la potencia de ruido recibida, este parámetro cumple la siguiente relación:

$$\frac{C}{N} = \frac{Eb}{No} + 10\log R - 10\log W \ [dB]$$

Ecuación 2. Ecuación de la relación portadora/ruido

Donde Eb/No es la relación entre la energía recibida por bit con la densidad de ruido, que se puede calcular mediante:

$$\frac{Eb}{No} = \frac{PGtGr}{L_l L_{FSL} L_a k T_s R}$$

Ecuación 3. Relación energía de bit con densidad de ruido

La fórmula anterior se aporta el resultado en escala lineal y en ella se puede encontrar parámetros como la potencia transmitida, las ganancias de transmisión y recepción y por otro lado en el denominador todas las pérdidas como las de espacio libre (FSL), las pérdidas en la línea o las atmosféricas además del factor kTsR que indica el ruido introducido en la recepción. Este último factor está formado por la constante de Boltzmann (k), la temperatura de ruido del sistema (Ts) y la tasa binaria en bits/s (R).

Si se lleva a escala logarítmica (dB) la anterior expresión queda de la siguiente forma:

$$E_b/N_o = P + G_t + G_r - L_l - L_{FSL} - L_a + 228.6 - 10\log T_s - 10\log R$$

Ecuación 4. Eb/No en escala decibélica

Donde se puede definir la PIRE (potencia isotrópica radiada efectiva) como:

$$PIRE = P - L_l + Gt \ [dB]$$

Ecuación 5. Ecuación PIRE

El otro factor importante es la relación G/T, llamado figura de mérito, que es sensibilidad de la estación receptora (un satélite por ejemplo) y puede ser calculada despejando de la siguiente expresión:

$$\frac{C}{No} = PIRE - L_{FSL} - L_a + \frac{Gr}{Ts} + 228.6\ [dB]$$

Ecuación 6. Figura de mérito

Tras realizar los cálculos necesarios y conociendo distintos parámetros se puede conocer la potencia mínima para conseguir comunicarse con un receptor a una distancia determinada. En este TFT se realizará una comunicación entre un emisor móvil y un satélite de la red Iridium por lo que todos estos cálculos serán realizados por el módulo de comunicaciones elegido para esta tarea cada vez que se vaya a utilizar.

La red de satélites Iridium es un sistema de comunicaciones globales por satélite que proporciona servicios de voz y datos en todo el mundo. Fue desarrollada por la empresa Iridium Communications Inc. y se encuentra en funcionamiento desde 1998. La red está compuesta por una constelación de 66 satélites en órbita baja alrededor de la Tierra [5] .

La principal aplicación de la red Iridium es la comunicación en áreas donde la infraestructura terrestre es limitada o inexistente. Los satélites Iridium actúan como enlaces de comunicación entre dispositivos terrestres, marítimos y aéreos, permitiendo la transmisión de voz y datos en tiempo real. Esto es especialmente valioso en situaciones de emergencia, exploración remota, operaciones militares, aviación y marítimas.

El funcionamiento de la red Iridium se basa en un concepto conocido como "conmutación de satélite". Cada satélite en la constelación Iridium está conectado a otros satélites cercanos y a estaciones terrestres, formando una red de comunicación global. Cuando un usuario desea realizar una llamada o enviar datos, su dispositivo se comunica con el satélite más cercano. El satélite recibe la señal y la retransmite a través de la constelación de satélites hasta que alcanza el satélite que se encuentra sobre la estación terrestre de destino. La estación terrestre completa la conexión y permite la comunicación bidireccional entre los dispositivos.

Cada satélite Iridium cubre una amplia área de la superficie terrestre, lo que permite una cobertura global continua. Los satélites se mueven a altas velocidades y están sincronizados para garantizar una distribución uniforme en la órbita baja. Esto significa

que siempre hay múltiples satélites disponibles para mantener una conexión estable y confiable en cualquier lugar del mundo.

Además de la comunicación de voz, la red Iridium también admite la transmisión de datos a través de su servicio de paquetes de datos. Esto permite la transferencia de mensajes de texto, correos electrónicos y acceso a Internet básico en áreas remotas. En este TFT se utilizará un módulo capaz de comunicarse con la red de Iridium para enviar datos y que el satélite los reenvíe a un servidor donde serán procesados, almacenados y mostrados en una web.

2.3 Fundamentos de WordPress y programación web

La creación de páginas web mediante WordPress es una opción popular y versátil que ofrece numerosas ventajas a los usuarios. WordPress [6] es un sistema de gestión de contenido de código abierto que permite a los usuarios crear y administrar sitios web de manera eficiente, incluso sin tener conocimientos de programación avanzados.

Una de las principales ventajas de utilizar WordPress es su facilidad de uso. Ofrece una interfaz intuitiva y amigable, lo que permite a los usuarios crear y gestionar contenido de forma sencilla. Con un amplio conjunto de herramientas y opciones de personalización, WordPress proporciona flexibilidad para adaptar el diseño y la funcionalidad de un sitio web según las necesidades individuales.

WordPress se basa principalmente en PHP, un lenguaje de programación del lado del servidor ampliamente utilizado en el desarrollo web. Además de PHP, también utiliza HTML y otros lenguajes para crear y diseñar la estructura y la apariencia visual de un sitio web. Estos lenguajes permiten a los desarrolladores personalizar y extender la funcionalidad de WordPress mediante la creación de temas y plugins.

En cuanto a la elección de alojamiento web, existen dos opciones comunes: un servidor local o un host externo. En un servidor local, el sitio web se ejecuta en una computadora personal o en una red local, lo que puede ser útil para desarrollo y pruebas. Sin embargo, para que el sitio sea accesible en Internet, es necesario utilizar un host externo o un servidor conectado a internet.

Un ejemplo de host popular es Infinity Host [7]. Se trata un proveedor de servicios de alojamiento web que ofrece diferentes planes y características para satisfacer las

necesidades de los usuarios de WordPress. Proporcionan almacenamiento, ancho de banda y soporte técnico para garantizar un rendimiento óptimo y la disponibilidad del sitio web.

WordPress ofrece una amplia gama de plugins que permiten presentar datos de diversas formas. Por ejemplo, hay plugins para crear galerías de imágenes, formularios de contacto, tablas de precios, y mucho más. Estos plugins facilitan la incorporación de funcionalidades específicas al sitio web sin necesidad de programar desde cero.

En cuanto al almacenamiento de datos, WordPress utiliza una base de datos para almacenar y recuperar información, como contenido de publicaciones, páginas, comentarios, configuraciones de temas y plugins, entre otros. El sistema de gestión de bases de datos utilizado por WordPress es MySQL, que es un sistema de gestión de bases de datos relacional. MySQL utiliza el lenguaje de consulta estructurado (SQL) para administrar y manipular los datos almacenados en la base de datos.

El manejo de datos en WordPress implica la creación, lectura, actualización y eliminación de registros en la base de datos. A través de su API (interfaz de programación de aplicaciones) de base de datos, WordPress proporciona funciones y métodos para interactuar con la base de datos de manera segura y eficiente. Los desarrolladores pueden utilizar consultas SQL personalizadas en combinación con las funciones proporcionadas por WordPress para realizar operaciones de manejo de datos de manera efectiva.

La REST API de WordPress [8] es una interfaz de programación de aplicaciones que permite a los desarrolladores interactuar de forma remota con los datos y funcionalidades de WordPress . Utiliza la arquitectura REST y los métodos HTTP estándar para realizar operaciones en los recursos de WordPress.

Esta API es extremadamente útil y versátil. Proporciona acceso a los datos almacenados en WordPress, como publicaciones, páginas, comentarios, usuarios, taxonomías y metadatos. Los desarrolladores pueden leer, crear, actualizar y eliminar estos datos mediante solicitudes HTTP.

Una de las principales ventajas de la REST API de WordPress es su capacidad de integración con otras aplicaciones y servicios. Permite conectar WordPress con aplicaciones externas y servicios de terceros, lo que facilita la creación de aplicaciones web o móviles personalizadas que se integren con un sitio de WordPress, se trata de una

herramienta fundamental para el desarrollo de temas y *plugins* personalizados. Los desarrolladores pueden aprovechar los puntos finales proporcionados por la API para personalizar y ampliar la funcionalidad de WordPress. Incluso es posible crear nuevos puntos finales para agregar funciones personalizadas.

La REST API también facilita la interacción de WordPress con otros sistemas y servicios, como gestores de correo electrónico y sistemas de comercio electrónico. Esto permite la automatización de tareas y la sincronización de datos entre diferentes plataformas.

Otra ventaja importante de la REST API de WordPress es su capacidad para construir aplicaciones de una sola página. Esto implica cargar el contenido de forma dinámica sin tener que recargar toda la página, lo que mejora la experiencia del usuario y permite crear interfaces interactivas.

En resumen, WordPress es una plataforma poderosa y flexible para la creación de páginas web que ofrece una interfaz intuitiva, facilidad de uso y una amplia variedad de opciones de personalización. Utiliza lenguajes de programación como PHP, HTML, CSS y JavaScript. La elección de alojamiento web puede incluir servicios como Infinity Host, servicios gratuitos pero con limitaciones. Permite presentar datos de diversas formas mediante plugins y almacena los datos en una base de datos MySQL, que se gestiona a través del lenguaje SQL.

En este TFT se utilizará el desarrollo de webs mediante WordPress para crear una página abierta al público a través de la cual, cualquier usuario pueda visualizar los datos recibidos desde el sistema embebido encargado de tomar las distintas medidas.

Capítulo 3 Diseño del sistema

En este capítulo se explicará el diseño del sistema completo. Se realizará una división por bloques atendiendo a las distintas funcionalidades de cada uno y posteriormente se mostrará la elección de los componentes específicos del sistema.

3.1 Descripción general del sistema propuesto

Se definen como componentes cada uno de los elementos que forma un bloque, estos componentes pueden ser sensores o módulos. De esta forma, por ejemplo, en el bloque de medición se encuentran los módulos: sensor de temperatura, anemómetro y GPS. A su vez se distinguirán dos tipos de componentes según su tipo: hardware y software, siendo el primer tipo los componentes físicos y el segundo tipo aquellas herramientas o plataformas que ayudan a la programación y plasmado de los datos.

Los 3 bloques principales que conforman este proyecto: medición, comunicaciones y presentación. A su vez se añaden 2 bloques adicionales como son el de alimentación y el de *testing*, ambos son bloques ya desarrollados que serán de utilidad para alimentar el sistema y comprobar el correcto funcionamiento de este.

Bloque de medición, se define este bloque como todos los componentes que requiere el proyecto para la toma y manejo de los datos. Se encuentran ambos tipos de componentes utilizados para la implementación de este bloque: software y hardware.

El bloque de comunicación, que recogerá los datos del bloque de medición, optimizará su tamaño y los enviará a través de una red satelital que redirigirá los datos a la dirección destino indicada para mostrarlos en el último bloque.

El bloque de presentación se encarga de mostrar de forma correcta y aportando una buena experiencia al usuario, pudiendo interpretar los datos y analizarlos.

Por otro lado, los bloques adicionales: son bloques que están ya desarrollados e implementados para el correcto funcionamiento del sistema, entre los que se encuentra el bloque de alimentación y el bloque de *testing*. En este bloque se encuentran aplicaciones para dispositivos móviles que abren una terminal Bluetooth para recibir los mensajes tal y como se enviarían a la web, así como la recepción de códigos de error que nos ayudarán a depurar nuestra aplicación.

Como se ve en la Figura 2 , el bloque de medición realiza la toma de medidas y su procesado, tras ello pasan al bloque de comunicación, dentro de este bloque será el módulo Rockblock+ que se enviarán los datos a través de la red Iridium. También se pueden ver las conexiones con los bloques adicionales: el de alimentación aporta energía al de medición y comunicación y, por otro lado, el bloque de *testing*, que extrae datos de la comunicación entre los bloques principales para comprobar el correcto funcionamiento de estos.

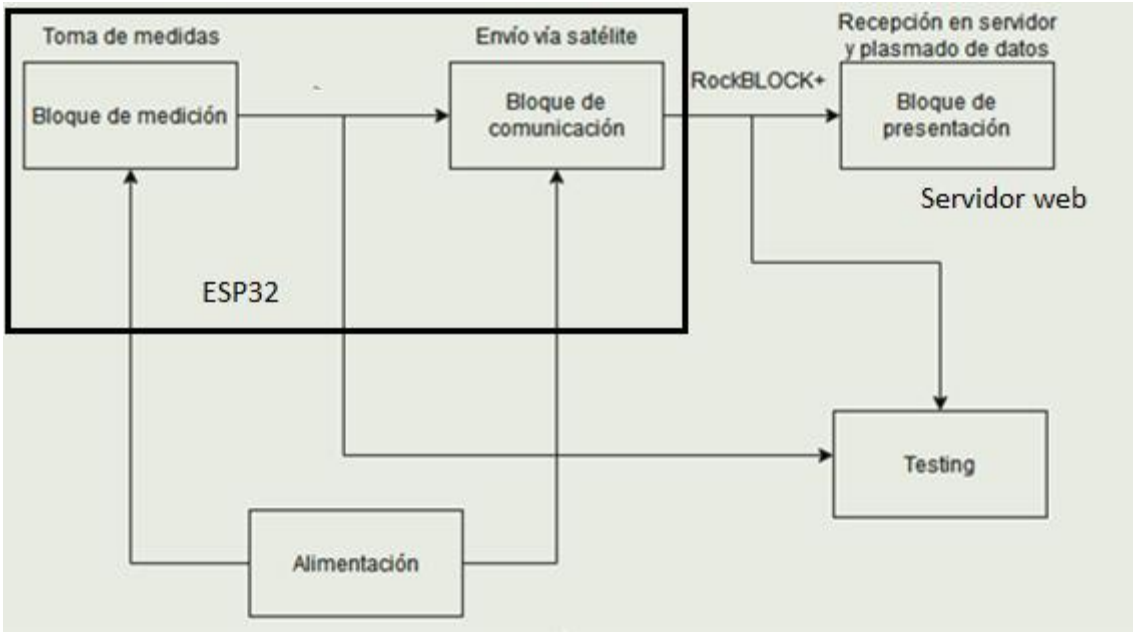


Figura 2. Diagrama de bloques del sistema

El sistema de alimentación es necesario para suministrar la energía necesaria al sistema. Se usará una batería de 12V además de dos placas solares, todo ello conectado a un regulador MPPT, que a su salida dispondrá de un convertidor de 12V a 5V, que irán conectados a los pines de alimentación de la placa.

3.2 Componentes del sistema

Se comenzará con los componentes del bloque de alimentación, entre los que se encuentra un controlador MPPT (*Maximum Power Point Tracking*), que es un dispositivo electrónico que se utiliza en los sistemas de energía solar para optimizar la producción de electricidad y garantizar que los paneles solares funcionen a la máxima potencia.

Los paneles solares tienen un punto de potencia máxima que varía en función de la temperatura, la radiación solar y la carga conectada al sistema. El controlador MPPT monitorea continuamente esta potencia máxima y ajusta el voltaje de carga y la corriente del panel solar para garantizar un funcionamiento óptimo del sistema solar [9].

El bloque de alimentación aportará la energía durante los periodos de tiempo que nos interesen, el sistema no estará en funcionamiento continuamente, sino que se definirá cada cuánto tiempo se quiere que los bloques principales funcionen.

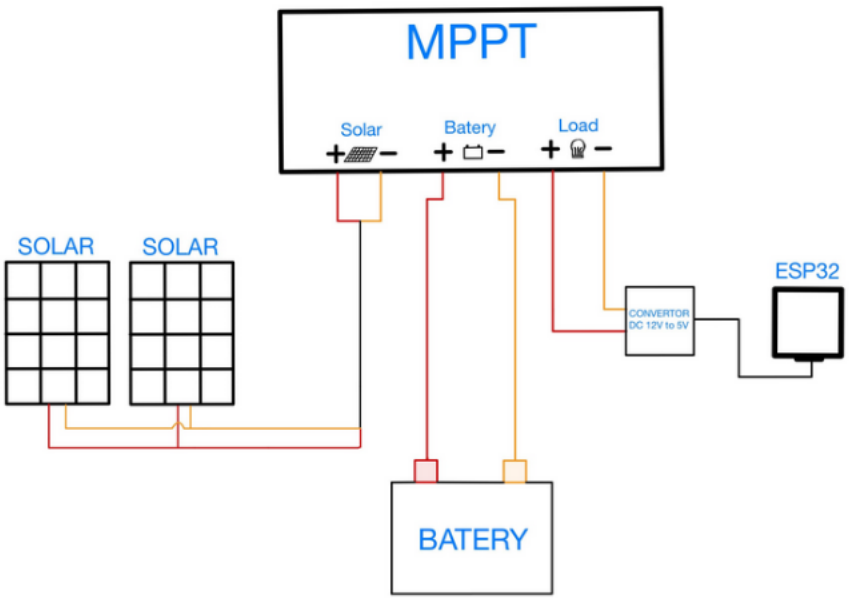


Figura 3. Diagrama del bloque de alimentación

En este capítulo se realizará una tarea de investigación de los componentes y herramientas que se utilizarán durante el proyecto. Se comenzará por los componentes que se utilizarán en el bloque de medición, continuando por los del bloque de comunicaciones y finalmente los de presentación. Se excluirá el microcontrolador de esta lista de componentes puesto que se dedicará un subcapítulo para ello.

3.2.1 Componentes del bloque de medición

Comenzando por los sensores de temperatura, se utilizarán dos DS18B20 [10], [11], termómetros que aportan medidas de resolución de 9 de hasta 12 bits en grados Celsius.

Estos sensores funcionan mediante el protocolo 1-Wire del que se hablará más tarde, pero como se puede intuir por el nombre del protocolo, permite la conexión de varios sensores a través de 1 sola línea de datos por lo que se tendrán los dos sensores conectados a los mismos pines del microcontrolador.

En la siguiente imagen se puede ver la configuración de pines junto a una imagen del sensor:

Figura 4. Esquema e imagen del sensor DS18B20

Las características eléctricas se encuentran en la siguiente tabla:

Tensión de alimentación	3.0V hasta 5.5V
Rango de funcionamiento	-55ºC hasta 125ºC
Error del termómetro	±0.5ºC entre -10ºC y 85ºC

Tabla 1. Características sensor D18B20

El error de la temperatura mostrada si se alcanzan temperaturas más altas de las indicadas en la tabla aumenta hasta los 2ºC para las temperaturas cercanas a las extremas indicadas en el rango de funcionamiento.

El tiempo que tarda este sensor en convertir con una precisión de 12 bits la temperatura es de aproximadamente 750 ms, aunque no se utilizarán esos 12 bits para codificar la temperatura puesto que no nos interesa tal nivel de precisión, con este dato se tienen el tiempo máximo de trabajo de cada uno de nuestros sensores de temperatura.

Algunas alternativas al tipo de sensor utilizado puede ser el sensor LM35, que es un sensor completamente analógico cuyo voltaje de salida es directamente proporcional a la temperatura, pero se debe realizar una conversión analógico-digital para la lectura en el microcontrolador, además de ser menos preciso y requerir de calibración periódica.

En general, para la región de temperaturas de trabajo en las que se trabajará, todos los sensores dan un error de 0.5ºC por lo que interesan más otras variables como la precisión, el tipo de sensor (analógico o digital) e incluso el precio para realizar la elección del mejor sensor para nuestra aplicación.

El siguiente sensor que se va a analizar es el anemómetro FSJT-NPNR [12], se trata de un anemómetro de tres tazas que enviará distintos valores de voltaje al pin correspondiente de la ESP32. En la siguiente tabla se observan las especificaciones de este:

Voltaje de entrada	5V-30V
Rango de medida	0m/s hasta 70m/s
Resolución	0.1m/s
Temperaturas de trabajo	-20ºC hasta 60ºC
Tiempo de respuesta	< 0.5s
Señal de salida	/0V hasta 5V/ /4mA hasta 20mA/

Tabla 2. Características anemómetro

Este sensor cuenta con 4 cables de distintos colores: el negro indicando el polo negativo de alimentación, el marrón indicando el polo positivo, el verde indicando el cable 485-A y el azul para indicar el 485-B, según la conexión de estos dos últimos se configurará el modo de trabajo del sensor.

Tiene dos modos de funcionamiento que condicionan su voltaje de entrada y su comportamiento como el modo pulsado, que es el utilizado en este proyecto; se conectan el cable verde a la salida PNP y el azul al NPN. Para cambiar al modo de funcionamiento continuo se deja el cable negro conectado al polo negativo (tierra) y se pone el marrón en la entrada. Para este modo debe de estar entre 10VDC y 30VDC, en este modo sólo se necesita medir el voltaje del cable azul para obtener el dato.

A continuación, se observa una imagen del sensor a utilizar:

Figura 5. Anemómetro de tres tazas

Las alternativas para medir la velocidad del viento podrían ser el uso de un sensor de presión diferencial como el MPXV7002DP que midiera la diferencia de presión entre dos puntos, dato que sirve para estimar la velocidad del viento u otro tipo de sensores como los sensores de efecto Hall como el A3144, que midiera las variaciones del campo magnético que produce el viento al mover un imán sin embargo esta opción elegida parece la más indicada si se mira tanto la precisión y como la sencillez de implementación en el código e instalación así como su precio.

Otra de las características muy importantes para tener en cuenta en la aplicación es la resistencia al agua puesto que este sensor, independientemente del tipo que fuera, va en la cubierta de la maqueta del barco por lo que se tiene que asegurar la resistencia al agua.

Tras la explicación de los dos tipos de sensores de parámetros físicos a utilizar, falta un último módulo que se puede considerar como sensor, como es el módulo de geolocalización Beitian-BN880. Este módulo proporcionará datos como la altitud, la latitud, la longitud e incluso otros como la distancia hasta cierto punto, así como la velocidad del barco.

El módulo Beitian obtiene los datos de varias redes satelitales como GLONASS, Galileo o BeiDou. Tras el arranque del módulo este comenzará a buscar los satélites más cercanos para poder triangular su posición, este proceso puede llevar más o menos tiempo según la posición y la ubicación en la que se arranque el módulo, una vez fije los satélites a

utilizar, comenzará a procesar los datos y se podrán obtener vía comunicación serie en la ESP32. En la siguiente tabla se ven las especificaciones de este:

Voltaje de alimentación	2.8V-6V
Sensibilidad de detección	-167dBm
Sensibilidad de captura	-148dBm
Tiempo de posicionamiento "Cold Start"	26s
Tiempo de posicionamiento "Warm Start"	25s
Tiempo de posicionamiento "Hot Start"	3s
Precisión de posicionamiento	2m a cielo abierto

Tabla 3. Características Beitian

Otros datos de importancia pueden ser la frecuencia de salida que va desde 1Hz hasta los 10Hz, la precisión de la velocidad, que es de 0.1m/s y el rango de temperaturas de funcionamiento que va desde los -40ºC hasta los 85ºC. Este módulo se comunica a través del protocolo UART.

En la siguiente imagen se puede ver el módulo Beitian-BN880 y a continuación se describirán los pines y las conexiones.

Figura 6. Módulo Beitian-BN880

Tiene un total de 6 pines, de izquierda a derecha son: SDA, GND, TX, RX, VCC y SCL. Para la conexión con la ESP32 no se usarán los pines 1 y 6 puesto que son para utilizar el protocolo de comunicación I2C y en este caso se utilizará UART.

UART es un protocolo asíncrono por lo que no se requiere señal de reloj adicional entre los dispositivos como es el caso de otros protocolos como I2C. En este caso se utilizarán bits de inicio y parada para marcar las transmisiones realizadas, generalmente se utilizan 8 bits de datos y un bit opcional de paridad y 1 o 2 bits de parada. Los módulos tendrán una tasa de trasmisión de 9600 baudios en el caso del Beitian (GPS) y de 19200 baudios en el caso del ROCKBLOCK+ (comunicación satelital).

Este protocolo utiliza una línea para transmitir (TX) y otra para recibir (RX). La transmisión de datos se realiza enviando los bits de la trama de uno en uno. Recibir datos implica leer la trama bit a bit e interpretarlo correctamente de acuerdo con el formato del protocolo UART. Se debe tener en cuenta la conexión de los cables puesto que se deberá cruzar la línea de transmisión del módulo con el pin de recepción de la ESP32 y lo mismo con el caso de la línea de recepción y el pin de transmisión. En ambos casos se utilizará la comunicación que viene configurada por defecto, 8N1, que indica 8 bits de datos, sin bit de paridad y 1 bit de parada.

Gracias a este protocolo se consiguen recibir datos del módulo GPS a través de las librerías correspondientes que se verán más adelante, así como la comunicación con el módulo ROCKBLOCK+ (que se verá más adelante) puesto que necesita los comandos necesarios para realizar ciertas tareas.

En cuanto a alternativas para calcular la posición geográfica se puede valorar la instalación de una antena GNSS, por ejemplo, los chips u-blox NEO, que también se comunican a través de un puerto UART o I2C. En caso de tener la placa funcionando en un lugar con cobertura se podría acceder a la API de Google Maps u otros servicios similares a través del módulo Wifi de la ESP32, pero no se va a trabajar en esas condiciones por lo que se descarta completamente.

Comparando los chips u-blox NEO con el Beitian se ve que los primeros pueden ofrecer una mayor precisión y calidad a cambio de un mayor coste, pero el Beitian ofrece una compatibilidad con múltiples sistemas de sistemas globales de navegación por satélite por

lo que se puede obtener la información de varias redes satelitales, opción que la alternativa no ofrece, además de que no se requiere mucha precisión para la aplicación a desarrollar.

Se utilizarán las librerías necesarias que se comentarán más adelante para el correcto funcionamiento de este para obtener principalmente nuestra latitud y longitud con hasta 6 decimales, ofreciéndonos una precisión más que suficiente.

3.2.2 Componentes del bloque de comunicaciones

En el bloque de comunicaciones, el único componente hardware que se utiliza es el módulo para la comunicación vía satélite para el envío de datos. El modelo concreto utilizado es el RockBlock+, que con el protector hace que este dispositivo sea a prueba de agua siendo perfecto para la aplicación a desarrollar.

Este módulo utiliza el servicio Iridium Short Burst Data de Iridium para enviar los datos desde el RockBlock+ hasta el satélite correspondiente y de ahí a las direcciones que se pueden configurar a través del servicio web que proporciona Iridium. Entre las posibilidades se encuentran direcciones de correo electrónico e incluso número de teléfono, para todas las opciones también se puede seleccionar el formato de los datos de llegada entre los que se encuentran HTTP_POST, HTTP_JSON o EMAIL, entre otros.

Este módulo incluye un conector USB de Tipo A clásico, en cuyo interior se pueden encontrar 6 pines de los cuales se usarán únicamente 4: GND, VCC, TXD y RXD.

Figura 7. Módulo ROCKBLOCK+ y pinout USB

El servicio que utiliza este módulo como se ha comentado previamente es el Iridium Short Burst Data, que es capaz de transmitir hasta 340 bytes y recibir hasta 270 bytes. Posteriormente se verá el uso de las librerías y los comandos utilizados para enviar los mensajes y controlar el flujo saliente del módulo.

Hay varias alternativas al uso de este módulo, pero todos con la misma idea de comunicación vía satélite puesto que se va a trabajar en unas condiciones en las que no se encuentra conectividad a ninguna red y la solución de almacenar la información en un disco duro u otro sistema de almacenamiento no es operativo puesto que se requiere un sistema en tiempo real.

Entre estas alternativas hay otros módulos de comunicación satelital por ejemplo los módulos de Globalstar, como el Globalstar Simplex Trasnsmitter u otras opciones como las de Thuraya o Inmarsat, estos dos últimos son móviles que se comunican vía satélite. Descartando los móviles por comodidad y coste quedan las opciones de Globalstar e Iridium, se elige la última opción debido a la mayor cobertura que aporta la red Iridium y las características que ofrece el módulo elegido a la hora de configurarlo.

3.2.3 Componentes del bloque de presentación

En este bloque no se cuenta con ningún componente hardware puesto que se supone que los datos se encuentran ya en el servidor y la función de este es la presentación de estos en la web. Se utilizará un servidor hospedado con capacidad de tener WordPress instalado y mediante el uso de distintos plugins se mostrarán los datos de forma agradable al usuario para la interpretación y comprensión de estos.

Mediante la REST API se creará una página capaz de recibir datos mediante solicitudes HTTP POST, esta página recibirá las peticiones y guardará los datos en una tabla creada en la base de datos de WordPress. Estos datos se podrán visualiza a través de gráficas y un mapa en otra página de visualización de datos.

3.3 Selección de la ESP32 y sus características

Para el control y el manejo de los bloques de medición y comunicación es necesario el uso de un microcontrolador. En este proyecto se ha optado por el uso de un ESP32, en concreto el modelo WROOM-32D, de ESPRESSIF, que será programado en C++, en el

framework de Arduino puesto que se utilizarán un conjunto de bibliotecas y herramientas que encajan muy bien para el desarrollo de aplicaciones en la placa a utilizar.

Esta placa cuenta con 38 pines de diferentes categorías: 26 pines de propósito general, que se pueden utilizar como entradas o salidas digitales o analógicas, 3 pines de alimentación, 6 pines para comunicación serie UART, SPI, I2C y otros protocolos de comunicación, 3 pines de interfaz de audio I2S que se pueden utilizar como entrada y salida de audio digital y finalmente otros pines de control de la propia placa como los pines de control de reset y control de alimentación.

Entre las posibles ventajas de utilizar este modelo concreto se encuentra el doble núcleo de procesamiento de 32 bits, que permite una mayor capacidad de computación, así como la capacidad de tareas en paralelo.

Por otro lado, también cuenta con un módulo Bluetooth y otro Wifi integrados en la misma, por lo que se podrán enviar los datos a través de estos en caso de tener los puertos serie ocupados con otros periféricos.

Se cuenta también con 448KB de memoria ROM para el arrancado y el funcionamiento de los núcleos, así como 520 KB de memoria SRAM para datos e instrucciones y 8KB de SRAM para la memoria RTC (Real Time Clock) rápida (accedida a través del procesador principal) y RTC lenta (accedida a través del coprocesador), que guardarán datos incluso tras arrancar desde el modo de hibernación de la placa.

En la siguiente imagen se puede ver la ESP32 con toda la descripción del *pinout*:

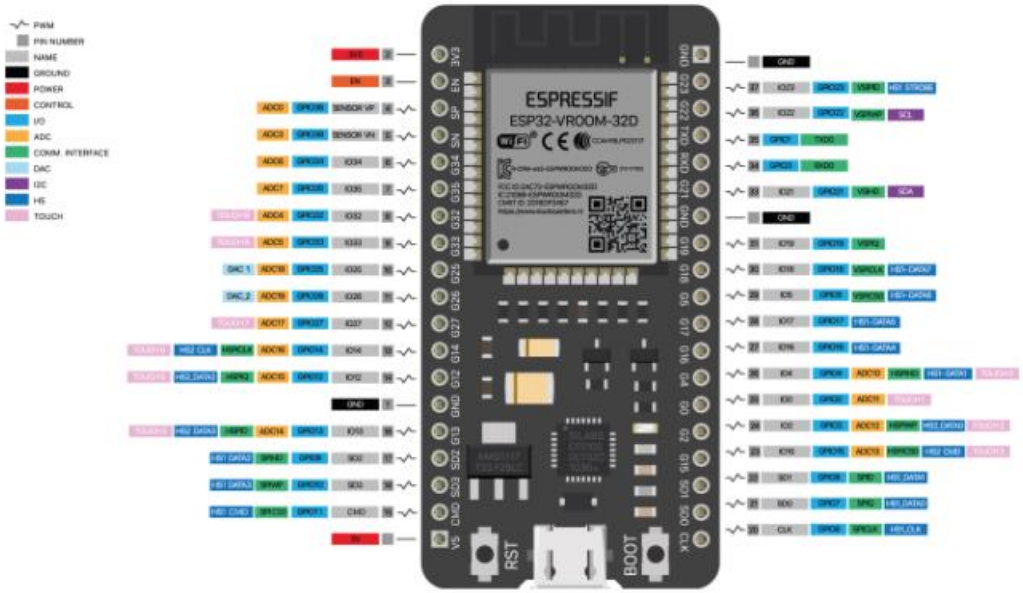


Figura 8. Pinout ESP32

Se pueden ver las características eléctricas y las condiciones de trabajo recomendadas para el microcontrolador en la tabla siguiente:

Oscilador de cristal	40 MHz
Suministro de energía	3.0V a 3.6V (típico 3.3V)
Corriente de trabajo	Promedio de 80mA
Mínima corriente de suministro	500mA
Temperatura de trabajo	-40ºC a 85ºC

Tabla 4. Características ESP32

Otra de las utilidades para tener en cuenta para la selección de esta placa es FreeRTOS, un sistema operativo en tiempo real para sistemas embebidos que se puede utilizar en microcontroladores como la ESP32. Este sistema operativo permite la creación y gestión de tareas con distinta prioridad y función en los distintos núcleos de la placa, así como habilita la gestión de distintos *timers* para cambiar el modo de funcionamiento de la placa, la creación de semáforos y colas, así como otras funcionalidades que no se aplicarán en este proyecto.

En FreeRTOS se definen tareas con una serie de parámetros de entrada como el nombre de la propia tarea, la función a realizar (texto informativo de tipo *String*), el *stack size* que ocupará la tarea, uno o varios posibles parámetros de entrada para nuestra tarea, la prioridad que tendrá en la pila de tareas y finalmente un puntero que actúa como referencia a la tarea.

FreeRTOS ejecuta tareas según la prioridad que se define en cada una, siendo 0 el nivel más bajo de prioridad y pudiendo definir una cifra superior como valor máximo de prioridad. Este sistema operativo permitirá controlar la rutina de ejecución de nuestra aplicación, de hecho, el potencial máximo de esta herramienta permitiría realizar programas que resuelven el problema de acceso a la sección crítica pudiendo programar semáforos y saltar de unas tareas a otras.

Otros modelos similares al utilizado para este proyecto pueden ser las placas de la familia ESP32, como pueden ser el modelos 32U, que añade una antena U.FL o por otro lado el modelo WROVER, que cuenta con memoria RAM adicional para aplicaciones que lo requieran.

En este capítulo se han descrito y justificado el porqué del uso de los componentes elegidos sin entrar en el detalle del funcionamiento interno, tarea realizada en los siguientes capítulos.

Capítulo 4 Bloque de medición

En este capítulo se explicará el funcionamiento del bloque de medición, las conexiones, los protocolos y la rutina que realizará el microcontrolador para este bloque.

4.1 Funcionamiento de los sensores utilizados

El protocolo que utilizan los sensores de temperatura que se presentaron en el capítulo anterior, como se ha mencionado previamente es 1-Wire, que es un protocolo de comunicación de bajo nivel utilizado en sistemas electrónicos para enviar datos a través de un solo cable de datos. Desarrollado por Dallas Semiconductor, se usa para conectar dispositivos electrónicos en redes de sensores y sistemas integrados.

Un protocolo de un solo cable se basa en una arquitectura de bus que permite que varios dispositivos compartan la misma línea de datos. Cada dispositivo tiene un identificador único de 64 bits llamado código ROM que lo distingue de otros dispositivos en el bus. Esto permite conectar varios dispositivos a la misma línea de datos sin necesidad de direcciones adicionales, lo que lo hace adecuado para aplicaciones con espacio y recursos limitados como la que se está desarrollando en este proyecto.

Cuando un dispositivo envía datos, primero baja la línea de datos durante un período de tiempo específico llamado pulso de reinicio. Un pulso de reinicio es una señal especial que inicia la comunicación en un bus de un solo cable. Después del impulso de reinicio, los dispositivos conectados al bus responden con un impulso de presencia para indicar que están listos para comunicarse. A partir de ahí, la comunicación tiene lugar mediante el envío y la recepción de bits individuales.

Se utiliza señalización diferencial para la transmisión de datos. Esto significa que los datos se codifican en la diferencia de potencial entre la línea de datos y la tierra (GND). Un bus en protocolos de este tipo tiene dos estados lógicos: alto (también conocido como estado inactivo) y bajo (estado activo). Cuando la línea de datos está alta, el bus está inactivo y ningún dispositivo está transmitiendo datos.

Para conectar ambos sensores a nuestra ESP32 simplemente se conectan los pines de alimentación y tierra a los pines correctos y posteriormente se elige el pin GPIO 33 para extraer las temperaturas.

Por otro lado, el anemómetro no utiliza ningún protocolo específico, sino que varía el nivel de voltaje del pin al que está conectado para marcar que ha dado una vuelta completa. Se deberá implementar una función que cuente estas vueltas, así como un factor de conversión mediante los datos que nos aporta el fabricante del anemómetro para conocer la velocidad del viento. En la siguiente imagen se puede ver el esquema de conexionado completo:

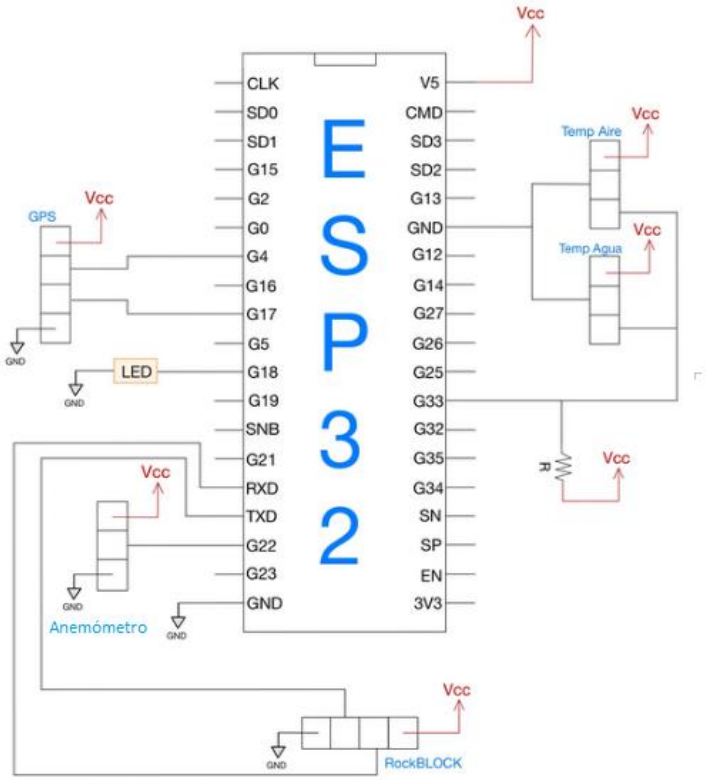


Figura 9. Conexionado de los bloques de medición y comunicación

4.2 Watchdog y modo de hibernación

Como se comentó a la hora de hablar del sistema de alimentación del sistema, no se va a querer utilizar el sistema por completo continuamente, sino que se definirán periodos en los que entrará en hibernación ahorrando energía.

En la ESP32 se encuentran varios modos de hibernación o de dormido de la placa, cada uno con un distinto nivel de profundidad, esto es, con un mayor o menor nivel de apagado de componentes internas de la placa. Los dos principales modos son : *Light Sleep* y *Deep Sleep*.

La principal diferencia radica en los componentes internos que mantiene activa la placa dentro de cada modo. En el modo *Light-Sleep*, los periféricos, la mayor parte de la RAM y las CPU están activados por reloj y su tensión de alimentación se reduce. Al salir del modo *Light-Sleep*, los periféricos digitales, la RAM y las CPU reanudan el funcionamiento y se conservan sus estados internos.

En el modo *Deep-Sleep*, las CPUs, la mayor parte de la RAM y todos los periféricos que se sincronizan desde APB_CLK se apagan. Las únicas partes del chip que permanecen encendidas son: controlador RTC, procesador ULP y las memorias RTC (*fast* y *slow*), que podrán almacenar información hasta con la placa dormida. Para nuestra aplicación nos interesa elegir el modo Deep Sleep.

En cualquier sistema se pueden encontrar errores espontáneos llevando al sistema a estados no deseados por causas fortuitas ajenas al sistema. Para ello se propone la utilización de un *watchdog*, que no es más que un contador que al llegar a cierto valor llevará al sistema a reiniciarse porque se asume que si ha llegado a ese valor el sistema está bloqueado.

Si se acude al manual de la ESP32 se pueden encontrar varios tipos de *watchdog* que se diferencian en cómo se evita el reinicio de la placa: mediante un temporizador o mediante un *trigger* externo (una variación de voltaje en un pin, por ejemplo). En nuestro caso se seleccionará por conveniencia y por la propia naturaleza del sistema el *Task Watchdog Timer*, que como su propio nombre indica, es el que funciona mediante un temporizador. En concreto este watchdog funciona mediante el RTC (Real Time Clock) de la ESP32, que aún en el modo de hibernación más profundo se mantendrá activo.

En el propio manual de la placa se pueden encontrar todas las funciones disponibles para el control del watchdog, cómo inicializarlo, cómo asignarlo a una tarea y cómo reiniciarlo entre otras. En el código de la placa el watchdog controlará la tarea que se encuentre en ejecución y sólo después de realizar la tarea correspondiente será cuando dejará de vigilarse y se asignará a la siguiente.

4.3 Rutina de funcionamiento

Para tener un acceso ordenado a cada uno de los sensores se va a hacer uso de las tareas que define *FreeRTOS*, para cada tipo de sensor se definirá una tarea en la que se tendrá que obtener el dato correspondiente, haciendo uso de los protocolos y funciones

necesarias para cada sensor, sabiendo esto tenemos la siguiente lista de tareas: encendido de un LED, medir temperaturas, medir viento, obtener posición GPS y finalmente aunque en el bloque de comunicación, una tarea para el envío de los datos. Por otro lado, también se debe tener en cuenta la hibernación del sistema y un sistema antibloqueo en caso de fallos inesperados.

Se utilizará la función *vTaskDelay* en vez de la función *delay* puesto que este último realiza la pausa, pero mantiene el microcontrolador ocupado y esto es negativo puesto que puede haber otra tarea en ejecución en paralelo y se estaría ralentizando su funcionamiento. De esta forma se pausa y se libera esa carga del microcontrolador, haciendo que, en caso de tener otra tareas en ejecución, puedan seguir funcionando correctamente.

La idea es definir tareas nuevas al final de las anteriores para ejecutarlas secuencialmente, de esta forma se mantendrá la misma prioridad siempre que se controle que no haya bucles en las tareas y que tras definir la tarea que prosigue se elimine la actual puesto que ya ha realizado su trabajo. Otra solución sería declarar todas las tareas con la misma prioridad desde un inicio y que se ejecuten en un orden aleatorio (o controlado definiendo prioridades) pero de la primera forma expuesta, se controla la tarea en ejecución en cada momento y se asegura el no tener problemas de *timing*.

Se importan las librerías necesarias para poder tener acceso a distintas funciones de utilidad. En el caso de los sensores de temperatura se utiliza la librería de *OneWire.h* para definir los objetos que habrá en el bus además de la librería de *DallasTemperature.h* que permitirá obtener los datos de estos sensores.

A continuación se deben definir todas las librerías junto con los pines y las variables necesarias. Las librerías utilizadas en este bloque son: *OneWire.h*, para conectar ambos sensores de temperatura al mismo pin utilizando el protocolo *1-Wire*, DallasTemperature.h, que aporta las funciones para poder obtener los datos de los sensores de temperatura, *SoftwareSerial.h* que permite crear la comunicación serie (UART) para el GPS y *TinyGPSPlus.h* que da acceso a las funciones necesarias para obtener datos relacionados con la posición GPS así como niveles de calidad de señal y demás información útil de este módulo.

Gracias a estas librerías se define el pin al que se le aplicará el protocolo 1-Wire al que irán conectados los sensores de temperatura. También se define un objeto de tipo *DallasTemperature* al que poder aplicar las funciones de la librería con mismo nombre y por otro lado se define la comunicación serie mediante una variable de tipo *SoftwareSerial* pasando como parámetros de entrada los pines de TX y RX de la misma y un objeto de tipo *TinyGPSPlus* del cual obtener los datos del módulo GPS.

Tras estas declaraciones se comienza con la programación de la función de *setup*. Junto a la función *loop* (no utilizada) equivalen a tareas de prioridad 0 y 1 respectivamente, siendo *loop* definida después de *setup,* por lo que en cada reinicio se ejecutará la tarea de configuración previa a la del bucle. La función loop por defecto se ejecuta en el procesador núcleo 1, donde se ejecuten las demás tareas cuando sean definidas. Por otro lado, en el núcleo 0 se ejecuta el sistema operativo y se realizan otras tareas no utilizadas como el control del Wi-Fi.

En la función de setup se inicializarán las tareas como el *watchdog* así como el RTC que controlará el tiempo que se mantiene la placa dormida. Se inicializarán otros procesos como las comunicaciones seriales necesarias: la del módulo GPS, y la de la comunicación Bluetooth para mostrar los resultados y comprobar el funcionamiento de los sensores (bloque de testing).

Se puede intuir el funcionamiento del sistema: despierta, realiza la primera medida, realiza la segunda…, realiza la medida N, pasa al bloque de comunicación y se pone en hibernación. Se observa que tiene un comportamiento cíclico que se puede modelar mediante una máquina de estados finita como se puede observar en la Figura 10.

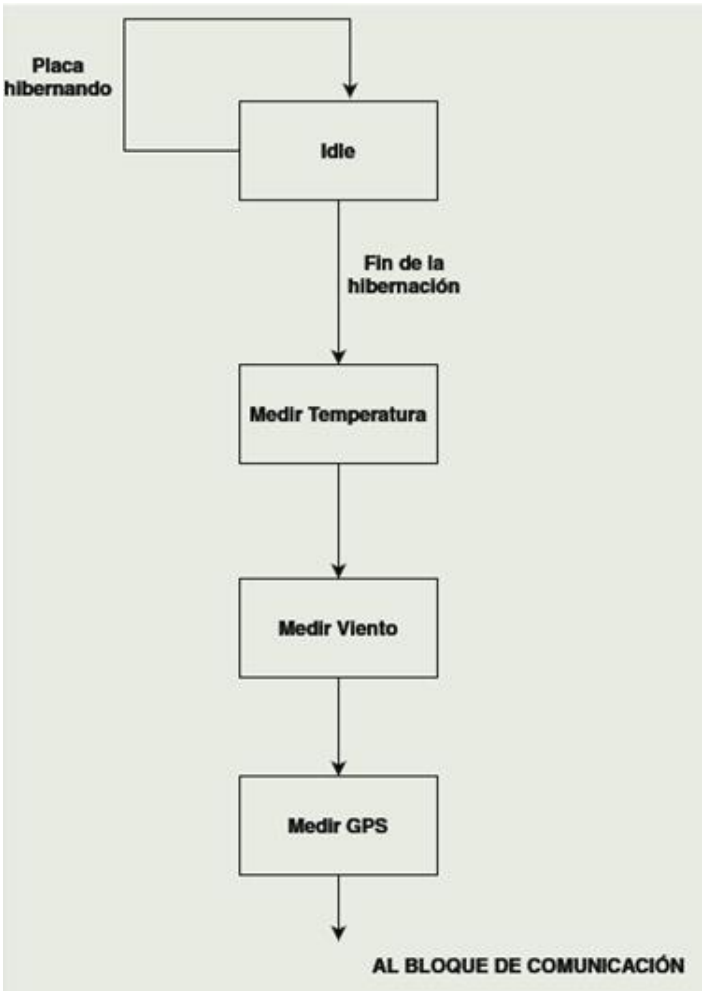


Figura 10. Máquina de estados del bloque de medición

Se sale del estado *Idle* cuando se despierte del modo *Deep Sleep* gracias al RTC, en ese estado se arranca el sistema y se definen las variables, así como se realizan las configuraciones necesarias para el correcto funcionamiento de todo el sistema.

Tras todas estas inicializaciones se define la primera tarea a realizar, que será el parpadeo de un LED, tras el parpadeo se define la tarea de MedirTemp, que se añadirá al *stack* de tareas pendientes y se ejecutará cuando la actual termine. Tras esto, se hará el reinicio del *watchdog* puesto que la tarea ha sido completada, se eliminará el control de la tarea actual y se configurará para controlar la siguiente tarea.

Ya en la tarea simplemente se obtendrá mediante las funciones correspondientes las temperaturas de ambos sensores y las serán optimizadas eliminando un decimal para ocupar 2 bytes menos a la hora de enviar nuestro mensaje puesto que se guardará la parte entera. Una vez en ese punto la tarea ya ha cumplido su función, se resetea el *watchdog*

y tras definir la siguiente tarea se puede asignar el *watchdog* a esta y eliminar la actual de la pila de tareas.

La siguiente tarea será tomar la medida de velocidad del viento, en este caso mediante una ISR (*Interrupt Service Routine*) que se ejecutará siempre que se detecten cambios en el voltaje del pin del anemómetro, cuando esto ocurra simplemente se suma 1 al contador de vueltas producidas, luego sólo se debe calcular la velocidad mediante los parámetros que da el fabricante del anemómetro. Tras esto simplemente se guarda el dato en el mensaje optimizándolo como las temperaturas, eliminando un decimal de precisión del dato. Se realiza el mismo procedimiento que en la tarea anterior con respecto al *watchdog* y a la gestión de tareas y se pasa a ejecutar la siguiente.

Ahora se comenzará con la obtención de la posición GPS mediante el módulo Beitian. En este módulo se debe esperar hasta que encuentre los satélites necesarios para ubicarse de forma correcta por tanto se utilizará un bucle para ejecutar la tarea continuamente, cuando el módulo ya pueda dar la información a la ESP32 se sabrá porque se reciben valores distintos de cero a través de la comunicación serial.

Se decodificará la trama y se guardarán los datos de latitud y longitud y finalmente sólo quedará definir la tarea del envío del mensaje, terminando aquí con el bloque de mediciones.

Se utilizará un testeo de caja negra para comprobar el correcto funcionamiento de este bloque. Mediante el bloque de *Testing* se añadirá una tarea a través de la cual se enviarán todos los datos recogidos en el bloque de mediciones vía Bluetooth. Se pueden testear varias funcionalidades forzando el *watchdog*, así como modificar el tiempo de hibernación de la placa.

Como se puede ver en las siguientes imágenes, se puede ver el funcionamiento de las tareas así como los datos originales y los optimizados tras eliminar los decimales. Se muestra el envío de todos los datos que se enviarán en un mensaje final, la temperatura del agua (segundo dato) tiene un valor de -127 debido a que el sensor se encontraba desconectado.

Figura 11. Envío de datos por Bluetooth

Se puede comprobar que la rutina de ejecución, procesado y envío de los datos es correcta, todo el código desarrollado para este bloque puede ser encontrado en el Anexo I. Código final ESP32 con bloque Testing.

En este capítulo se ha descrito el funcionamiento interno de los componentes del bloque de medición en el sistema en conjunto. Ahora se procederá con la programación del bloque de comunicación para enviar los datos.

Capítulo 5 Bloque de comunicación

Tras explicar las especificaciones de nuestro módulo ROCKBLOCK+ en el capítulo 3, en este capítulo se explicará el funcionamiento del módulo al implementarlo en el sistema.

5.1 Consideraciones iniciales

Para que este módulo esté operativo se debe tener una suscripción activa a los servicios, esta suscripción deberá activarse tras crear una cuenta en la página oficial del producto, esta suscripción tiene un coste de 13 GBP. Además de tener el servicio activo se debe comprender el funcionamiento del sistema de *tokens* que utiliza Rockblock.

Se establece un sistema de *tokens* en el que cada una permite el envío de 50 bytes de datos, esto sumado a la restricción del propio módulo que, como se menciona en el capítulo 3, tiene un límite de 340 bytes de buffer de subida, haría un gasto máximo de 7 tokens para enviar esa cantidad.

Existen paquetes de compra de *tokens* siendo el más pequeño 100 *tokens* por 14.50 GBP. Se debe asegurar que tanto la suscripción esté activa como que la cuenta tenga aún créditos disponibles para enviar para que el módulo funcione correctamente.

Dentro de la misma página de Rockblock existe la opción de configuración de remitentes, donde se deberá especificar el método de envío de los datos y la dirección a la que serán enviados. Existe la opción de crear grupos para, en caso de disponer de varios módulos, establecer distintos destinatarios según convenga.

Entre las opciones de formato de llegada de los datos se encuentran: HTTP_JSON, HTTP_POST, HTTP_THINGSPEAK, EMAIL_ROCKBLOCK y SBD_ROCKBLOCK. Se utilizará la opción de email para comprobar que los mensajes llegan correctamente en una fase previa a la conexión con el siguiente bloque, aunque como se comenta previamente, la opción final será HTTP_POST. Si se acude a las referencias que aporta RockBlock a la hora del envío de mensajes se puede obtener la estructura de llegada de

los datos. Al utilizar el método HTTP_POST, con un contenido del tipo: application/x-www-form-urlencoded.

La solicitud de envío de datos contendrá los siguientes parámetros:

Parameter	Description	Example
`imei`	The unique IMEI of your RockBLOCK	`300234010753370`
`serial`	The serial number of your RockBLOCK	`12345`
`momsn`	The Message Sequence Number set by RockBLOCK when the message was sent from the device to the Iridium Gateway. The value is an integer in the range 0 to 65,535 and is incremented each time a transmit session is successfully completed from the device to the Iridium Gateway. It is a wrap around counter which will increment to 0 after reaching 65535.	`12345`
`transmit_time`	The date & time (always UTC) that the message was transmitted.	`21-10-31 10:41:50`
`iridium_latitude`	The approximate latitude of the RockBLOCK at the time it transmitted.	`52.3867`
`iridium_longitude`	The approximate longitude of the RockBLOCK at the time it transmitted.	`0.2938`
`iridium_cep`	An estimate of the accuracy (in km) of the position information in the previous two fields.	`8`
`data`	Your message, hex-encoded.	`48656c6c6f20576f726c6420526f636b424c4f434b`

Figura 12. Parámetros enviados por el ROCKBLOCK+

De los parámetros mostrados anteriormente sólo se utilizarán *transmit_time* y *data* puesto que el resto no serán de utilidad a la hora de mostrar los datos.

5.2 Módulo ROCKBLOCK+

Para la conexión se utilizarán todos los pines a excepción del 2 y el 6. Se conectan los cables de alimentación y tierra a los pines correspondientes de la ESP32 y a continuación los cables de transmisión y recepción a los pines de recepción y transmisión serial que trae nuestra placa en los pines 1 y 3. La comunicación con este módulo también será a través del protocolo UART.

El mismo módulo tiene un LED rojo incorporado a través del cual se conocerá si este está correctamente alimentado y si esto ocurre sólo falta controlar su comportamiento a través de la ESP32.

El modelo de ROCKBLOCK+ elegido está preparada para aguantar las condiciones climatológicas del mar, tanto temperatura como humedad, además de ser resistente al agua. La colocación de este módulo será sobre la cubierta del barco, fijado a través de un tornillo que la atraviese.

La comunicación con este módulo se realizará a través del envío de comandos y la recepción de respuestas que indicarán el correcto envío o el código de error que corresponda. Se definen dos tipos de mensajes a enviar: MO (*Mobile Originated*) y MT (*Mobile Terminated*), en este caso sólo se utilizarán los MO ya que será la terminal móvil según la arquitectura a utilizar.

Se dispone de una serie de comandos para controlar la comunicación, la cual se establece a 19200 baudios y 8N1 (8 bits de datos, no paridad y 1 bit de stop). El comando que se utilizará principalmente en nuestra comunicación es "AT+SBDWT= Mensaje" , de donde se puede descomponer el "AT" como un comando de petición y "SBDWT" un comando que indica insertar un mensaje en formato ASCII en el buffer de mensajes MO. En caso de querer recibir los códigos de confirmación o error se utilizará "AT+SBDIX" abriendo de esta forma una sesión extendida.

La respuesta que se recibe en caso de una sesión común será "OK" en caso de que todo funcione correctamente. En caso de una sesión extendida (se usará para corregir errores, en la versión final no será utilizada) se recibirán una serie de códigos que siguen la siguiente estructura: "<MO *status*>, <MOMSN>, <MT *status*>, <MTMSN>, <MT *length*>, <MT *queued*>". Cada parámetro de llegada tiene sus valores característicos, por ejemplo, un mensaje que ha llegado correctamente le correspondería el siguiente código "0,0,0,0,0,0". El campo de mayor interés a la hora de corregir errores o ver el estado del sistema es el MO status, donde si se encontrase un código 14 significaría que se está usando un tamaño de segmento inválido o en caso de código 18 se sabría que ha habido una pérdida de conexión. Todos los códigos de cada campo se pueden encontrar en la página de documentos de Rockblock[13].

5.3 Implementación y envío de datos

En este bloque se asume que se han guardado correctamente los datos recogidos del bloque de medición, cada dato en su variable. Si se atiende a los datos que se tienen y a sus tipos se observa que, las variables de temperatura serán valores de 2 cifras con 1

decimal de precisión, el valor del viento tendrá la misma estructura, pero con la posibilidad de tener hasta 3 cifras en casos de viento extremo.

Por otro lado, se ve que la latitud y longitud son valores de 2 cifras y 3 respectivamente teniendo en cuenta también el signo que indica , junto a 6 decimales de precisión. Teniendo en cuenta las optimizaciones, un mensaje de ejemplo para enviar podría ser uno como el siguiente: "238,250,105,2807203,-1544808", de donde la temperatura del ambiente es de 23.8ºC, la del agua 25.0ºC, hay un viento de10.5m/s y la ubicación es 28,07203 N 15,44808 O.

Se puede observar aquí la optimización: se ahorra un byte en cada cifra de la que se elimina la coma que separa los decimales. En la recepción se deberán ajustar de nuevo los valores para su correcta interpretación.

A continuación, se va a realizar un cálculo de cuántos bytes y cuántos *tokens* se utilizarán según cada cuánto se duerma la placa al día. Se va a tomar un mensaje como el siguiente que cumple la condición de tener la longitud máxima (temperatura de tres cifras y signo negativo en ambas componentes GPS): "223,124,1052,-90540234,-150450124".

Este mensaje de ejemplo tiene una longitud de 33 bytes, por tanto, se usa 1 crédito por cada mensaje a enviar y se descarta aquí la opción de empaquetar dos rondas de mediciones en 1 sólo mensaje. Una alternativa para conseguir esto sería no enviar todos los datos cada vez que se despertara la placa, aprovechando esos 17 bytes que faltan hasta los 50 bytes enviando por ejemplo más valores de temperatura y viento aprovechando la capacidad que nos permite el sistema de créditos.

A continuación, se muestra una tabla que relaciona la frecuencia de hibernación de la placa y los bytes (y créditos) utilizados al día.

Frecuencia de hibernación	Medidas al día	Bytes enviados	Tokens/día
2 horas	12	372	8
3 horas	8	248	6
4 horas	6	150	4
6 horas	4	100	3

Tabla 5. Cálculo de consumo de tokens

Se ajustará la frecuencia de hibernación en el código de la ESP32 y esta será la forma en la que se regulará el gasto de créditos, y por tanto el coste económico de este módulo.

Sólo falta configurar la tarea del código de la ESP32 que controle el comportamiento de este módulo. Tras llegar de la tarea anterior con los datos optimizados se comenzará el establecimiento de la conexión con el comando AT, en caso de recibir un OK querrá decir que se puede comenzar la transmisión de los datos enviando el comando AT+SBDWT, en caso contrario (no se recibe OK) el código entrará en un bucle infinito hasta que el watchdog llegue a su tiempo establecido reiniciando la placa. Tras enviar los datos se puede ver el código de recepción y si todo ha ido correcto se espera un OK o en caso de seleccionar la sesión extendida los ceros correspondientes como se menciona previamente.

Tras realizar el envío sea cual sea el código de recepción se procede con el reseteo del watchdog y el paso de la placa al estado de hibernación hasta el siguiente ciclo, con esto termina el bloque de comunicación.

El correcto funcionamiento de este bloque se puede comprobar mediante el uso de la misma aplicación utilizada en el anterior bloque, leyendo los mensajes que envía el Rockblock así como la comprobación de que llegan los mensajes a la dirección de correo electrónico configurada en la web. Todo el código utilizado para la comprobación de funcionamiento de este bloque se puede encontrar en el Anexo I. Código final ESP32 con bloque Testing, tras finalizar la ejecución del bloque de medición.

En la siguiente imagen se puede observar la aplicación del bloque de testing sobre el de comunicación, en la siguiente imagen con la terminal Bluetooth mostrando el envío correcto (con la confirmación OK esperada):

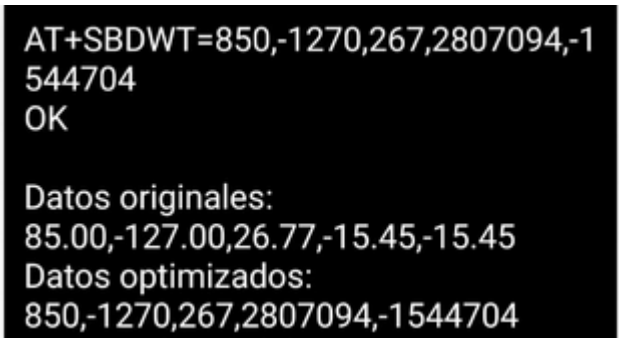


Figura 13. Bloque de testing para comunicación con ROCKBLOCK

Al haberse configurado en esta fase del desarrollo el envío de los mensajes a una dirección de correo electrónico se puede ver en la siguiente imagen la bandeja de entrada junto a un email abierto en el que se muestran los datos recibidos. Se puede comprobar también el gasto de 1 *token* a través de la página de configuración de este módulo.

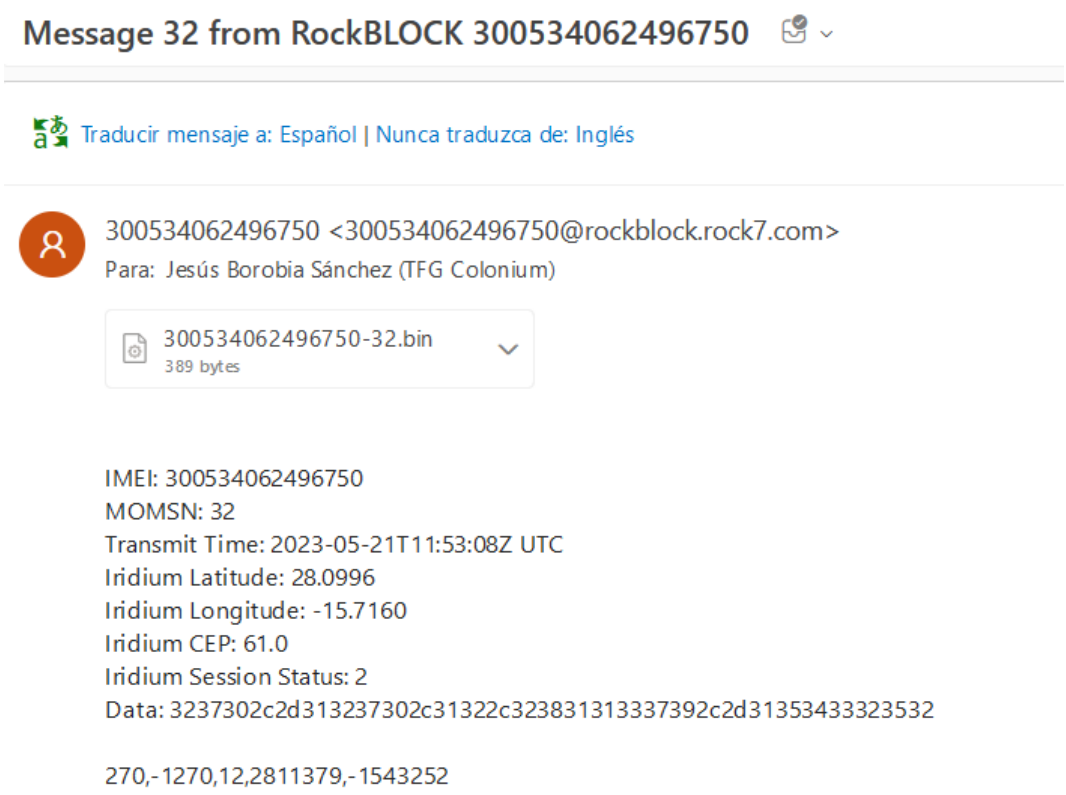


Figura 14. Recepción mensajes ROCKBLOCK a email.

En este capítulo se ha visto un bloque fundamental para el correcto funcionamiento del sistema, este bloque enlaza los sensores con el destino final de los datos estableciendo la conexión necesaria con la red Iridium.

Capítulo 6 Bloque de presentación

6.1 Configuración básica de WordPress

En este capítulo se verá cómo se guardan y cómo se presentan los datos enviados por el bloque anterior. Como se comentó previamente se utilizarán WordPress y sus plugins, además de la base de datos que proporciona WordPress para poder guardar todos los datos que se reciban y poder mostrarlos en diferentes formatos.

Se contemplan varias opciones para implementar la web, desde crear un servidor con todo lo necesario a través de una máquina virtual o utilizar un host que nos aporte lo necesario para crearlo. La opción elegida es la de utilizar el host, en este caso se ha utilizado Infinity Host, que con la opción gratuita permite crear la página web además de manejar la base de datos. Este bloque está diseñado sin tener en cuenta la recepción de datos a tiempo real pero es un diseño lo más cercano posible al sistema final, que sería capaz de escuchar continuamente las peticiones de recepción de datos y plasmado en la web.

En primer lugar, se debe crear la cuenta del host mediante el uso de un correo electrónico y una contraseña. Una vez creada la cuenta, el plan gratuito que ofrece este host permite crear hasta 3 dominios distintos. El dominio elegido ha sido: proyectocolonium.rf.gd y se continua al panel de control para comenzar con las instalaciones necesarias.

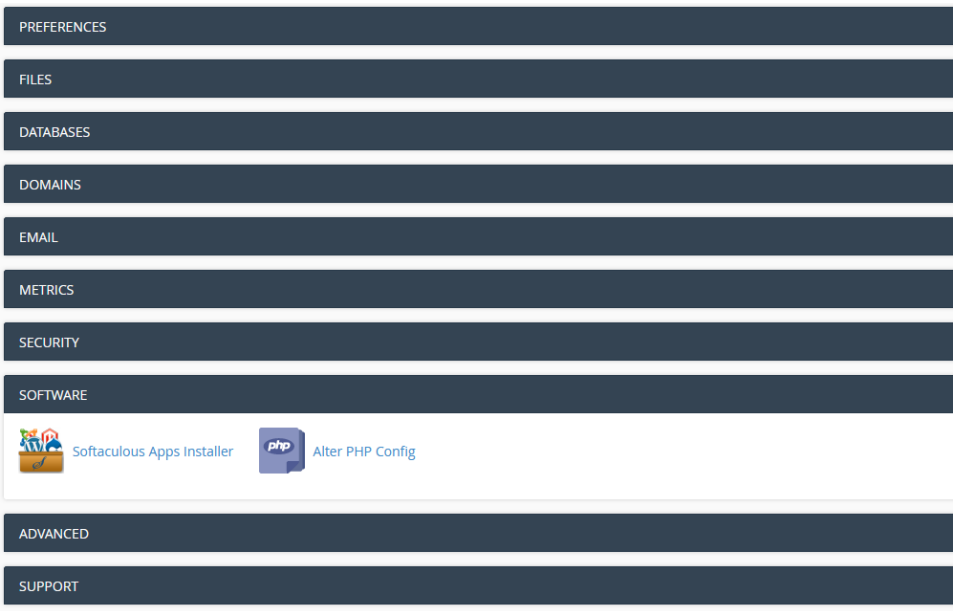


Figura 15. Menú de servicios de Infinity Host

Una vez se accede al panel de control del host se encuentran todas las opciones disponibles para utilizar diversas herramientas como se puede ver en la siguiente imagen:

Para comenzar con la creación de la web se necesita instalar en el servidor WordPress, que se encuentra en la pestaña de software, dentro de la opción del instalador de aplicaciones, una vez dentro se elige la opción de WordPress y se instala.

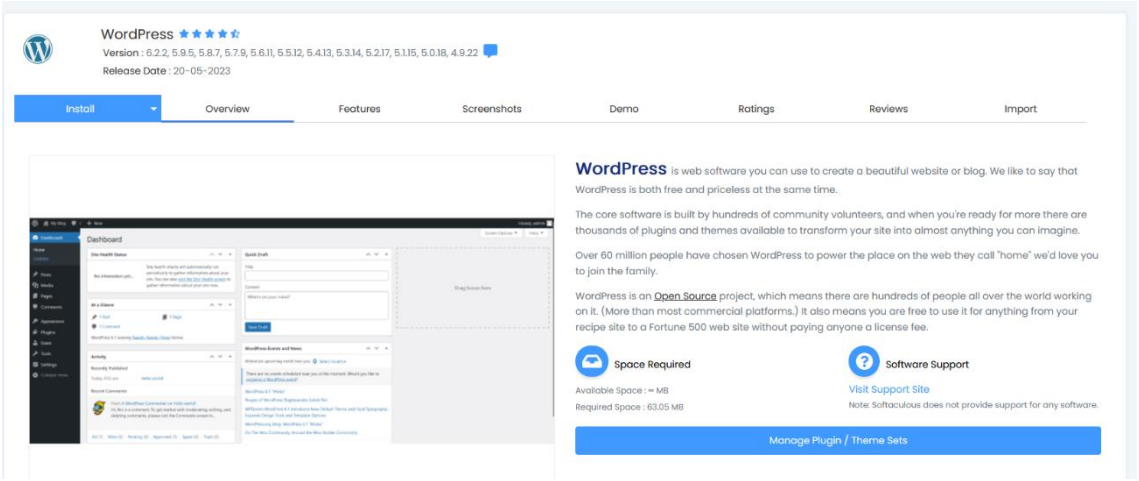


Figura 16. Panel de Instalación de WordPress

En el momento de instalar WordPress se dan varias opciones de personalización comenzando por la elección del nombre de usuario del administrador del WordPress así como la contraseña y el tema de la web de entre varias opciones.

La elección del tema es algo opcional, es una elección meramente estética que estructura la web así como establece disposición de las entradas y páginas de esta. Una vez elegido el tema se procede con la instalación, es un proceso automático y cuando se completa se permite el acceso al gestor de WordPress del host. Desde este gestor se puede acceder a la página como administrador permitiendo así introducir todas las modificaciones que se necesiten para la puesta a punto de la web final. El host también da el usuario y contraseña de acceso a la base de datos de WordPress, base de datos que necesitará modificaciones para el propósito de este proyecto.

Una vez iniciada sesión en WordPress con la cuenta de administrador se llega al panel de WordPress, desde el que ya se puede modificar la web, gestionar los plugins y controlar toda la personalización.

Se puede acceder a la primera versión de la web, que es la que viene incluida en la selección del tema. Se tiene en cuenta que la mayoría de los elementos que se visualizan en la siguiente imagen desaparecerán o serán modificados para el propósito deseado.

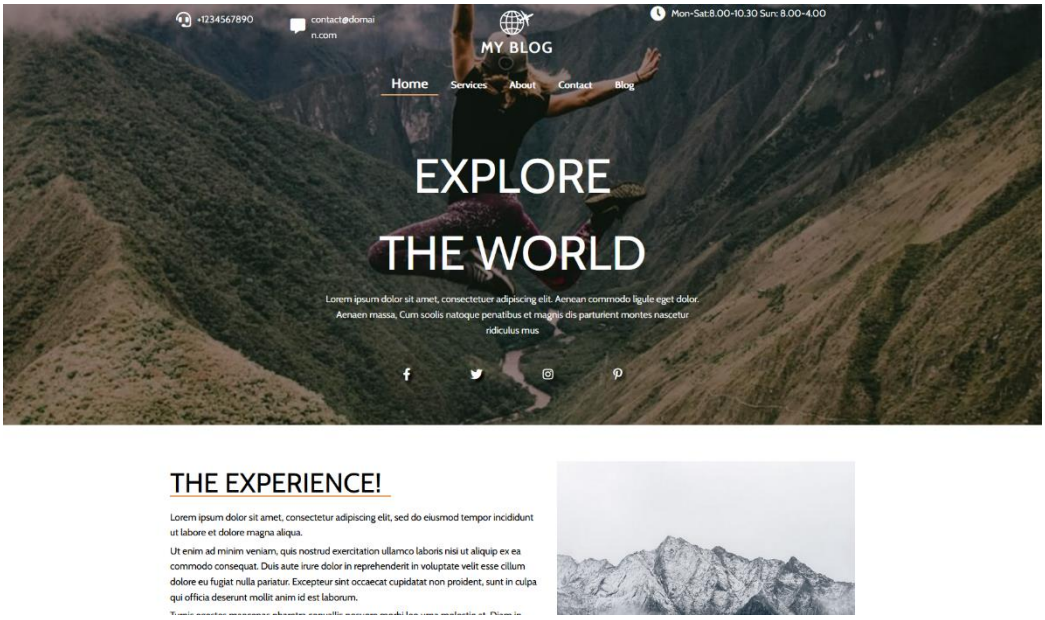


Figura 17. Ejemplo Plantilla WordPress

En WordPress, una página se utiliza para crear contenido estático y atemporal. Las páginas son ideales para información permanente, como la página de inicio, la página de contacto, la página de acerca de, etc. Las páginas se organizan jerárquicamente en una estructura de árbol y pueden tener subpáginas. Se puede personalizar su diseño y agregar contenido multimedia, como imágenes y videos. Las páginas también suelen tener elementos como encabezados, pie de página y menús de navegación. Son fáciles de encontrar en la estructura del sitio y a menudo se muestran en el menú principal de navegación.

En la imagen anterior se observa un post denominado *The Experience!* y varias páginas accesibles como Home (en la que se encuentra el usuario), *Services*, *About*, *Contact* y *Blog*.

6.2 Definición de la tabla en la base de datos

Antes de comenzar con la personalización y la gestión de plugins para representar los datos que se reciben, se debe crear la tabla dentro de la base de datos. Dicha tabla almacenará todos los datos que se reciban a la página que se indique más adelante.

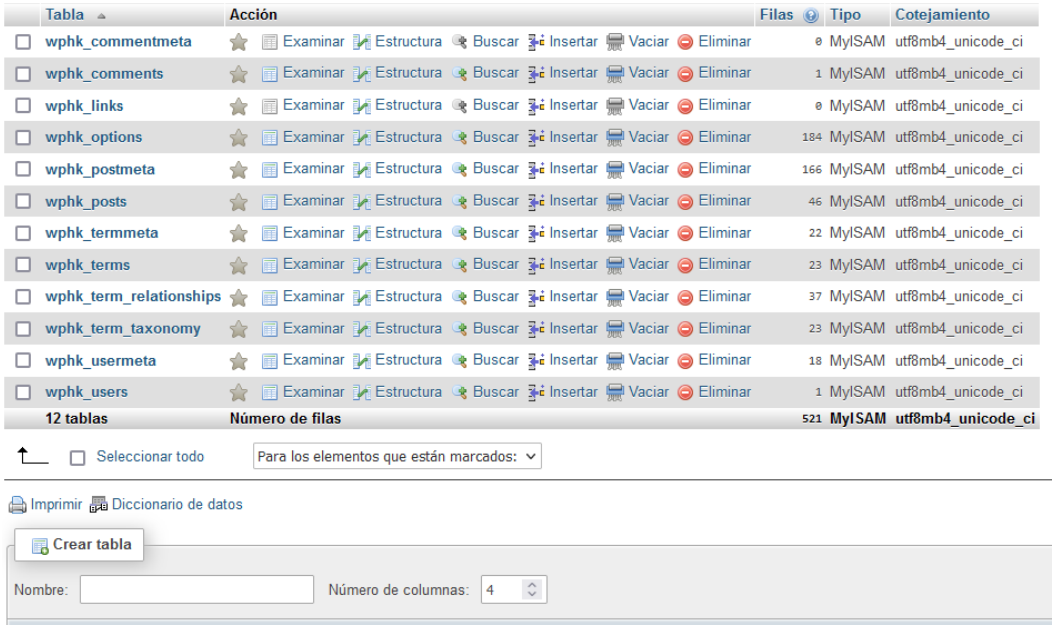


Figura 18. Menú de configuración phpMyAdmin

Para acceder al gestor de la base de datos desde Infinity se debe acceder al panel de control general y buscar la opción de phpMyAdmin, situado en la pestaña de *Databases*. Una vez dentro se ve algo como la siguiente imagen:

Se observa la opción de creación de tablas en la parte inferior de este menú. Se conoce del capítulo anterior la estructura de llegada de los datos vía HTTP_POST y se conoce también la estructura de llegada de los datos: valores separados por comas. Con toda esta información se crea una tabla nueva llamada “mediciones” en la que se agregarán las columnas que se muestran a continuación:

Id	Temp1	Temp2	Viento	Latitud	Longitud	Fecha_hora

Tabla 6. Columnas de la base de datos

El tipo de la variable se especificará a continuación, previo a eso se comenta el contenido de cada una de las columnas y la fuente de donde se obtienen los datos. En primer lugar la columna ID es una columna con un número natural que se irá incrementando. Esta columna es típica en las bases de datos puesto que se trabaja de forma cómoda a la hora de seleccionar, mover o presentar datos.

Las columnas desde la que indica la temperatura ambiente (temp1) hasta la longitud, son los datos obtenidos desde el bloque de mediciones y que van en el parámetro *Data* que envía el bloque de comunicación. Estos datos deberán de ser decodificados y separados antes de ser colocados en sus columnas.

Finalmente la columna “fecha_hora”, en ella irá incluida la fecha y la hora de transmisión del mensaje, este parámetro se obtiene a través del mensaje que manda el bloque de comunicación en el campo *transmit_time*, como se puede ver en el capítulo anterior.

Se rellena la tabla con el formato que tendrán los datos al añadirse a la base de datos, se observa la tabla rellena en la siguiente imagen:

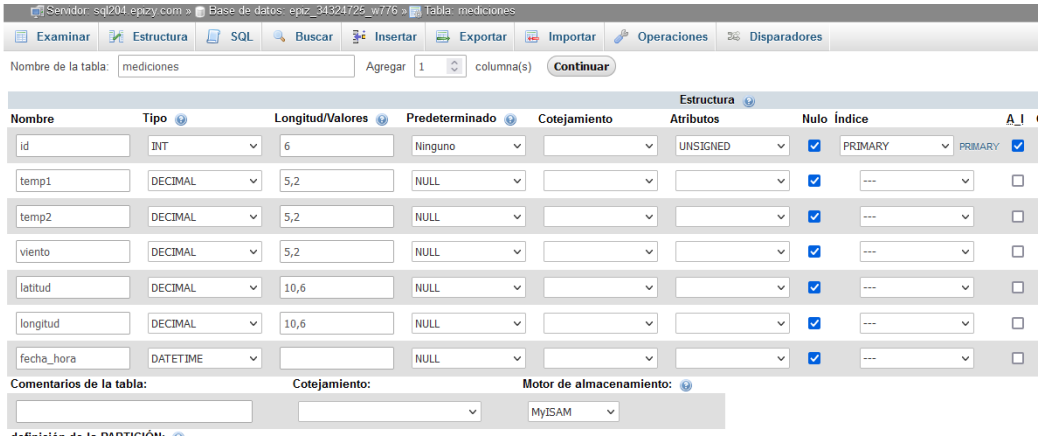


Figura 19. Configuración de variables de la base de datos

Se ha rellenado la tabla con los tipos de dato que se conoce que van a llegar a la base de datos, así como con su longitud en caso necesario. Los dos valores que destacan por tener un tipo o una configuración distinta son la columna ID, a la que se le configura un índice “PRIMARY”. El índice primario proporciona una forma eficiente de buscar y acceder a registros en una tabla. Además de garantizar la unicidad, también se utiliza para ordenar físicamente los datos en la tabla, lo que mejora el rendimiento de las consultas y operaciones de búsqueda.

Al definir una clave primaria, se especifica qué columnas formarán la clave primaria y se establece la restricción de unicidad en esas columnas. Por lo general, se elige una columna que contenga valores únicos y significativos para ser la clave primaria, como un identificador único o un número de serie (en este caso el parámetro ID).

También se ha configurado la opción de autoincrementado para esta columna, de tal forma que no se tengan que añadir los valores de forma manual, sino que se tenga una columna de valores naturales que vayan creciendo conforme se agreguen líneas a nuestra tabla.

Por otro lado, la columna fecha_hora, tiene como tipo DATETIME, que es un tipo de dato dentro de SQL que tiene la siguiente estructura “YYYY-MM-DD HH:MM:SS”, que indica año, mes ,día, hora, minuto y segundo en ese orden.

Como se ve, el resto de las columnas tendrán valores decimales, cada uno con su longitud total y número de decimales indicado en el apartado de Longitud/Valores, que se define como se puede ver en la imagen para cada columna. Todas estas columnas tienen por defecto NULL, que significa que si no se proporciona un valor específico en para esa variable en una nueva fila, se guardará como valor NULL en esa celda.

6.3 API para recepción de datos

Una vez creada la tabla donde se guardarán los datos entrantes se debe configurar la página a la que llegarán los datos vía HTTP_POST por parte de Rockblock. Se va a utilizar la REST API de WordPress y para ello se debe crear un *endpoint* personalizado, que será la página de recepción de datos. Para ello se accede a los archivos del servidor a través del menú correspondiente que ofrece el host. Dentro del gestor de archivos se observa la siguiente estructura de ficheros:

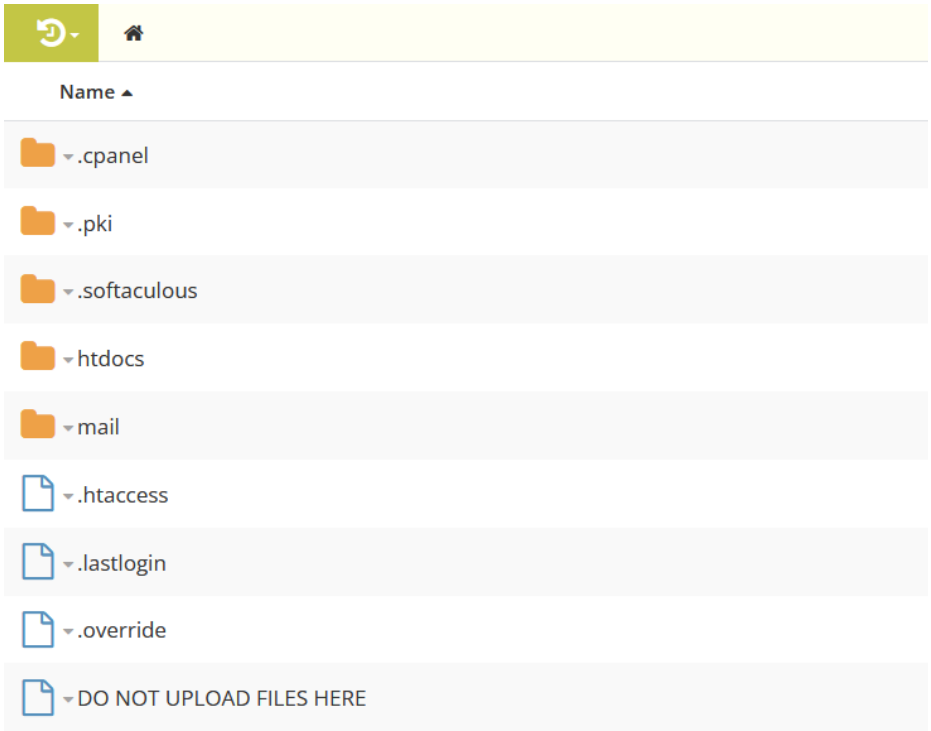


Figura 20. Archivos del host

Se podrá acceder a los archivos de WordPress dentro de la carpeta *htdocs*, dentro de ese directorio se debe acceder a */wp-content/themes/tema_utilizado*, donde en este caso se utilizará el tema llamado *twentytwentyone*. Dentro de la carpeta del tema que se está utilizando se accederá al archivo *functions.php* donde se procederá con la creación de un *endpoint* personalizado gracias a la REST API de WordPress.

En el *endpoint* sólo se aceptarán peticiones POST, cuando se reciba una se guardarán los campos de data y transmit_time en variables. Posteriormente se realizará la conexión a la base de datos utilizando los datos necesarios: nombre del servidor, nombre de usuario, contraseña y nombre de la base de datos. Una vez realizada la conexión SQL se decodifica el campo de data y se separan las variables recibidas por comas, guardando cada una en uno de los campos de la base de datos. Se debe tener en cuenta que hay que deshacer los cambios que se realizaron en cuanto a número de decimales por lo que se dividirá entre 10 las variables de temp1, temp2 y viento y por otro lado se dividirá entre 10000 la latitud y longitud, recuperando así el formato correcto. Se añadirá también la variable de transmit_time en el campo de fecha_hora. En caso de añadirse correctamente se responderá a la petición con un 200 OK, en caso contrario se enviará un código de error.

Se puede observar la creación del endpoint y la función a ejecutar cuando se reciban solicitudes POST en el anexo correspondiente a la recepción de datos HTTP POST.

En el host que se utiliza para el desarrollo de este proyecto no se permite el acceso a páginas web desde API externas ni tampoco el acceso desde cURL u otros clientes por la línea de comandos, y se pueden esperar errores como los que se presentan en la siguiente imagen:

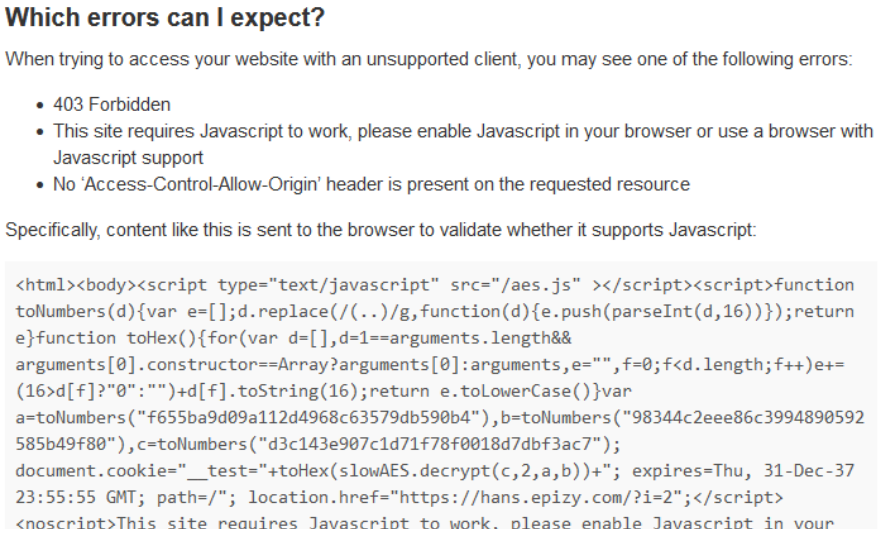


Figura 21. Posibles errores al acceder sin permiso

Se va a crear una cuenta en *Postman*, es una plataforma que permite el envío de, entre otros, solicitudes HTTP. Tras crear la cuenta se accede al espacio de trabajo y se crea un nuevo mensaje, se selecciona el tipo de petición a utilizar (POST) y se añade la URL destino. Tras ello se selecciona en "*Body*" el tipo de datos que se van a enviar, que en este caso es x-www-form-urlencoded y se añaden los nombres de las variables a enviar junto con valores deseados como se observa en la siguiente imagen.

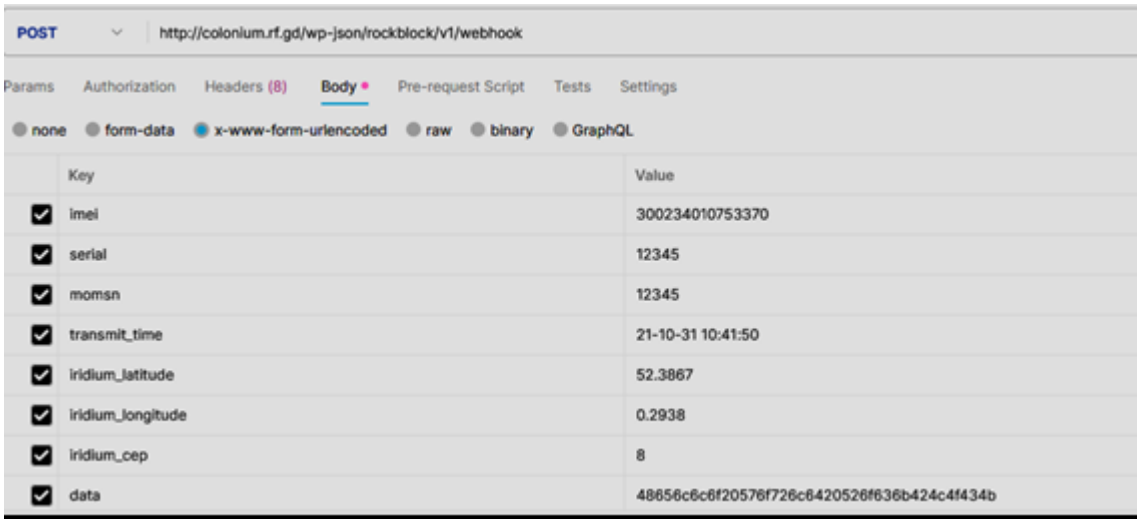


Figura 22. Solicitud HTTP Post en POSTMAN

Si se ejecuta la orden de enviar el mensaje, se observa el error correspondiente con el esperado en caso de acceso a través de clientes externos como indica Infinity Host.

Se concluye que no es posible la recepción de mensajes desde el servicio de ROCKBLOCK con el plan gratuito de Infinity Host por lo que una de las soluciones propuestas es la adquisición de un plan de pago o el cambio a otro host, así como la creación de un servidor propio ya que no se tendrán estas limitaciones.

Para comprobar que el endpoint sí funciona correctamente y por tanto, que se reciben datos, se va a crear una página de WordPress en la que se encuentra un formulario a rellenar con los campos que enviaría ROCKBLOCK. Este formulario enviará los datos a la URL de nuestro endpoint y por otro lado en una página de visualización de datos se podrán ver dispuestos en sus gráficas o mapas. El código del formulario se puede observar en el Anexo II. Formulario para añadir datos de forma manual.

Se crea la página del formulario mediante un código HTML y se indica la URL destino, se especifica también el tipo de petición a utilizar (POST) y se añaden los campos. Por simplicidad se ha creado un formulario con 2 campos únicamente, data y transmit_time.

Se puede comprobar el funcionamiento sin necesidad de una página de visualización de datos al rellenar y enviar el formulario y accediendo a la tabla de la base de datos, ahí se puede observar que los datos se han añadido correctamente. Sólo falta diseñar una página en la que los datos se visualicen de forma cómoda para el usuario.

6.4 Presentación de los datos

Se crea la página de visualización de datos, a la que se tendrá que añadir el código o los bloques de WordPress necesarios para visualizar los datos correctamente. Para ver los datos en gráficas y en caso de las coordenadas en mapas se va a hacer uso de *plugins*.

En primer lugar se debe añadir y activar el *plugin* llamado WP Code, que permite la creación de los llamados *shortcodes*. Estos *shortcodes* se pueden definir como accesos directos a *scripts* que no están ubicados directamente en la página como ocurría con el formulario.

Una vez activo el plugin se accede al menú y se crea un nuevo *script*, este primero permitirá la visualización de una variable en una gráfica. En el eje y estará la temperatura o el viento y en el eje x se encontrarán las distintas fechas y horas de transmisión del mensaje para ese dato.

Para la programación de este *script* se debe comenzar con la conexión a la base de datos SQL, en concreto a la tabla que se creó para recibir los datos, se accede a la base de datos con las credenciales necesarias y se recorren todas las filas de la tabla de mediciones guardando en un vector el dato que se quiera representar, como la temperatura ambiente (temp1) o el viento y por otro lado un vector con las fechas asociadas, que se encuentran en la misma fila, en la columna correspondiente.

Tras cerrar la conexión con la base de datos se crea mediante HTML y mediante el script chart.js una gráfica en la que se pueden seleccionar los ejes, el tipo de gráfica y otro aspectos estéticos como el color de las barras y el grosor del borde. Se configuran los ejes y las etiquetas para estos, así como el tamaño de la gráfica y se concluye el *script* y el HTML. Se puede encontrar el código para la generación de una gráfica en el Anexo IV. Generación de Gráfica. Ejemplo con Temp1.

Este proceso ha de ser repetido para las tres variables que se desean mostrar en un diagrama de barras, como son la temperatura ambiente (temp1), la temperatura del agua (temp2) y el viento.

En la siguiente imagen se pueden ver los resultados de la gráfica de viento junto al mapa con los puntos de tres coordenadas añadidas manualmente.

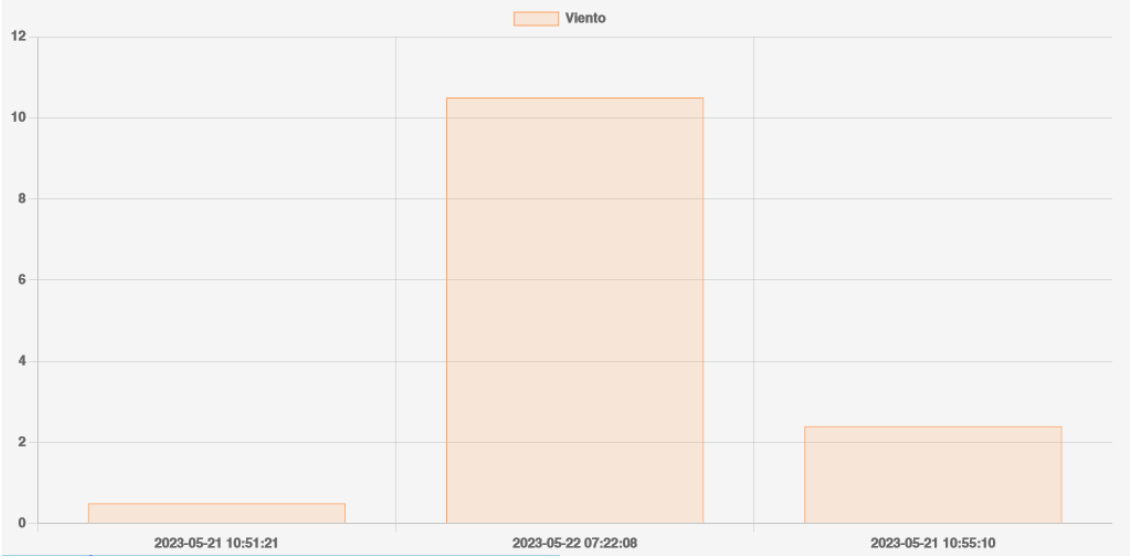


Figura 23. Gráfica con datos añadidos manualmente

Ahora se procederá con la creación de un mapa en el que se representen mediante marcadores las coordenadas formadas por las latitudes y longitudes recibidas para cada mensaje.

Para el mapa se utiliza el *plugin* Leaflet Map, que ha de ser añadido y activado como el resto de plugins. Una vez hecho esto se accede otra vez a WP Code y se crea un nuevo script denominado Mapa. En primer lugar se debe realizar la conexión a la base de datos y obtener los datos necesarios, que serían: latitud, longitud y fecha_hora en cambio se seleccionan todos puesto que se va a añadir un desplegable que muestre el resto de los datos para una fecha y hora concretos.

Para la creación del mapa se van a utilizar *shortcodes* que define el plugin de Leaflet. Estas herramientas permiten en una sóla línea crear un nuevo mapa y configurar opciones como el centrado del mapa así como el control del zoom mediante botones y el tamaño del mapa. Tras la creación del mapa se utiliza otro *shortcode* que añade un marcador en

todas las coordenadas obtenidas de la base de datos, puede verse en el Anexo V. Generación del mapa.

Como se puede observar, los datos de la velocidad del viento se añaden a la gráfica correspondiente, por otro lado en el mapa se añaden las coordenadas obtenidas para todas las medidas. A la derecha del mapa se ha añadido un desplegable programado mediante HTML en el que se pueden seleccionar todas las fechas disponibles de la base de datos y se muestran los datos de las temperaturas y la velocidad del viento para cada fecha y hora.

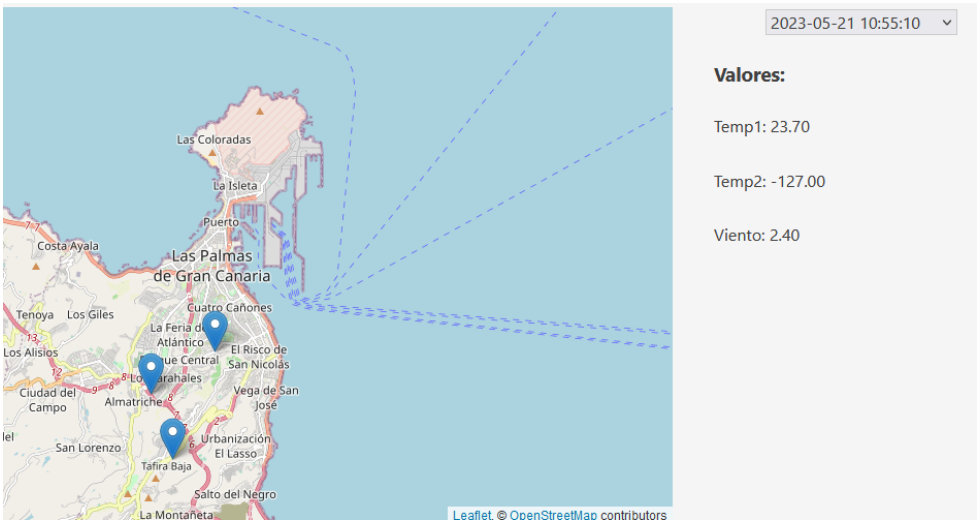


Figura 24. Mapa con coordenadas y desplegable para visualización de datos

La versión final de la web incluye dos páginas: Envío de Datos y Visualización de Datos, de las cuales el usuario final sólo podrá acceder a la segunda. En esta versión, como se comentó previamente no se ha tenido en cuenta la recepción de datos en tiempo real puesto que el uso del host limita estas funcionalidades, para cumplirlas se debería de utilizar un servidor y programar un servicio en segundo plano que escuchara todas las peticiones, en este caso peticiones HTTP_POST y que actualizara la web cada vez que un dato fuera añadido a la base de datos.

Por tanto la versión que se ha desarrollado cuenta con un sistema de simulación de llegada de datos, en la que se encuentra el formulario mostrado previamente, que serían los campos *data* y *transmit_time* que envía Rockblock. Como bloque de *Testing* para esta parte del proyecto se puede enviar el formulario y por el funcionamiento interno, se reciben los datos en el *endpoint* configurado y al acceder a la página correspondiente se visualizan los datos en cada una de las gráficas y en el mapa.

Capítulo 7 Resultados y conclusiones

7.1 Resultados

En la versión final del proyecto se utiliza un servidor que sí acepta las peticiones POST de sitios externos, por lo que sólo hay que copiar los archivos y los códigos utilizados en Infinity Host al WordPress del servidor definitivo cambiando las credenciales necesarias a la hora de conectarse con la base de datos. Se debe tener en cuenta que la página de envío de datos de forma manual no debe ser publicada. Además, en esta versión final se deben de añadir métodos de seguridad para que no cualquier usuario pudiera enviar solicitudes POST al receptor. La solución que se propone es una *whitelist* con las IPs que utiliza el servicio de Rockblock [14] para enviar los datos como se puede ver en el Anexo III. Endpoint y recepción solicitudes HTTP POST.

El servidor donde está instalado WordPress no cuenta con la herramienta de phpMyAdmin ni con el gestor de archivos con el que se trabajaba en Infinity Host, en cambio se pueden utilizar plugins que realizan estas funciones tales como DB-Manager y WP File Manager. Se puede acceder a la web a través de su URL (no definitiva) donde se puede observar tanto el formulario de simulación de llegada de datos a través de HTTP POST y la página de visualización de datos. La URL es: http://18.211.211.219/ .

Se han realizado diversas comprobaciones del correcto funcionamiento del sistema, trasladando el bloque de presentación a un servidor que permite el acceso de datos externos. Se instaló el sistema fuera de la maqueta del barco y se realizaron una serie de excursiones para comprobar que los datos eran emitidos y recibidos correctamente en el destino final. En la Figura 25 se puede ver el mapa con varias ubicaciones desde donde se enviaron datos.

Figura 25. Datos recibidos a la web final

Se pueden observar las distintas fechas de recepción de los datos también en las gráficas correspondientes, como en la siguiente imagen, donde se ve la gráfica de la temperatura del ambiente para las distintas fechas. Se debe tener en cuenta que en esta gráfica también se encuentran los datos añadidos manualmente para comprobar el funcionamiento del *endpoint* previo a la prueba final.

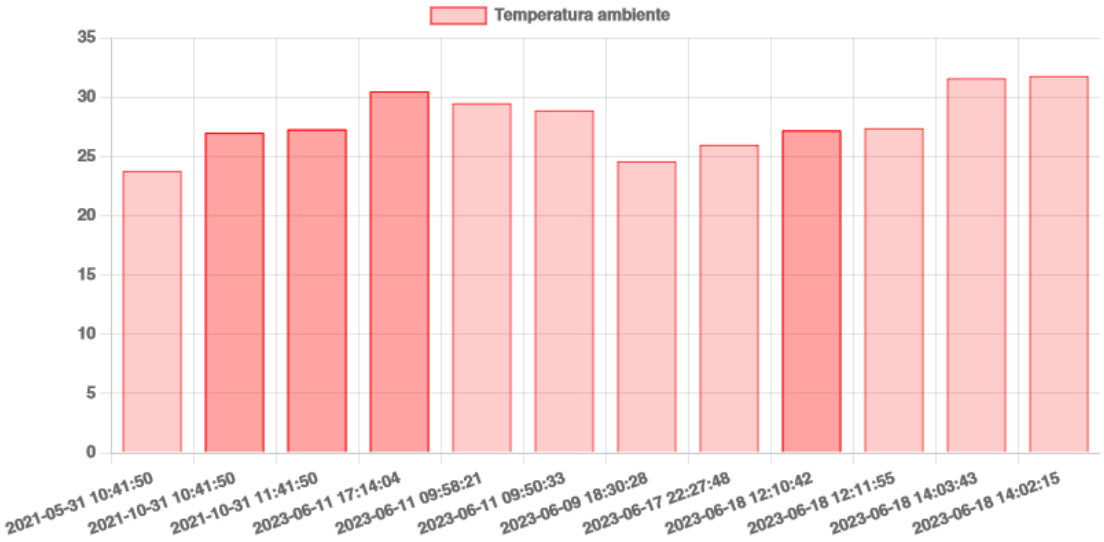


Figura 26. Gráfica con resultados finales

7.2 Conclusiones

A lo largo del proyecto se han desarrollado los bloques necesarios para transmitir los datos recogidos los sensores en el bloque de medición, controlados por la ESP32, optimizados y enviados a través de la red Iridium. Los datos son recibidos en la bandeja de entrada del correo electrónico configurado así como lo será en una versión futura en una página web mediante una solicitud HTTP POST, los datos recibidos se guardan en una tabla de la base de datos de WordPress y son presentados gracias al bloque de presentación en gráficas y en un mapa, pudiendo ver todos los datos asignados a una fecha y hora concreta gracias a un desplegable.

Se puede concluir que se han cumplido los objetivos propuestos, comenzando por O1 ("Diseño e implementación del bloque de toma de medidas en el barco") gracias al diseño, implementación y pruebas a través del bloque de Testing, se comprueba la correcta toma de los datos. Por otro lado O2 ("Envío y recepción de datos vía satélite") a través del diseño e implementación del bloque de comunicación y gracias a las pruebas realizadas se comprueba que se consiguen enviar y recibir los datos recogidos previamente. A su vez, O3 ("Tratamiento de datos y diseño web") también se cumple puesto que se reciben los datos a través de una API y se muestran a través de gráficas y mapas para interpretación del usuario interesado en una web. Finalmente O4 ("Presentación y consideraciones con respecto al proyecto Colonium") también se consigue, no sólo por la combinación e interconexión de los 3 bloques principales involucrados y su correcto funcionamiento en conjunto, sino también por la comprensión de las condiciones de trabajo del sistema y la elección de unos componentes adecuados para el medio.

Bibliografía

[1] O. Suárez, "Diseño e Implementación del Bloque de Comunicaciones del Proyecto Colonium y Estudio de su Impacto sobre el Alumnado".[En línea]. Disponible en: https://accedacris.ulpgc.es/bitstream/10553/111206/2/TFG%20Gabriel%20Ojeda%20Suarez.pdf

[2] "Historia de los Microcontroladores y sus Fabricantes", [En línea]. Disponible en: https://www.vistronica.com/blog/post/historia-de-los-microcontroladores-y-sus-fabricantes.html.

[3] R. Martínez y M. Calvo, "Comunicaciones por Satélite" . [En línea]. Disponible en: https://www.gr.ssr.upm.es/docencia/grado/csat/material/CSAT09-1-Introduccion.pdf

[4] "Teorema de Shannon-Hartley _ AcademiaLab", [En línea]. Disponible en: https://academia-lab.com/enciclopedia/teorema-de-shannon-hartley/

[5] "Iridium Satellite Communications ", [En línea]. Disponible en: https://www.iridium.com/.

[6] " WordPress.org España", [En línea]. Disponible en: https://es.wordpress.org/.

[7] "Free Web Hosting with PHP and MySQL - InfinityFree", [En línea]. Disponible en: https://www.infinityfree.com/

[8] "REST API Handbook | WordPress Developer Resources", [En línea]. Disponible en: https://developer.wordpress.org/rest-api/.

[9] A. Harjai, A. Bhardwaj, y M. Sandhibigraha, "Study of Maximum Power Point Tracking (MPPT) Techniques in a Solar Photovoltaic Array A Project Submitted in Partial Fulfillment of the Requirement for the Degree of Bachelor of Technology in Electrical Engineering". [En línea]. Disponible en: https://core.ac.uk/download/pdf/53187734.pdf

[10] "DS18B20 Datasheet", [En línea]. Disponible en: https://www.analog.com/media/en/technical-documentation/data-sheets/ds18b20.pdf

[11] "Wind Speed Sensor, Three Cups Anemometer - Renke", [En línea]. Disponible en: https://www.renkeer.com/product/polycarbon-wind-speed-sensor/

[12] "Transmit ASCII data", [En línea]. Disponible en: https://docs.rockblock.rock7.com/docs/transmit-ascii-data.

[13] "Integration with your application", [En línea]. Disponible en: https://docs.rockblock.rock7.com/docs/integration-with-application.

[14] ULPGC. "Retribuciones del Personal Investigador, del Personal Técnico y del Personal Técnico de Apoyo Contratado de la ULPGC con Cargo a Proyectos, Convenios y Contratos para el Año 2023". [En línea]. Disponible en: https://www.ulpgc.es/sites/default/files/ArchivosULPGC/transparencia/Personal/2022/Retribuciones/2022_retribuciones_pi_contratado_ulpgc.pdf

ANEXOS

Anexo I. Código final ESP32 con bloque Testing

```
//CODIGO FUNCIONAL PARA LOS SENSORES Y ENVIO IRIDIUM + BLUETOOTH

//EN ESTE CÓDIGO LEE TEMP1,TEMP2 Y VIENTO Y LO ENVÍA ( EJ: 22.37,-
127.00,2.67)
//PAQUETE COMPLETO (TEMP1,TEMP2,VIENTO,LATITUD,LONGITUD)

//Librerías a utilizar
#include <SoftwareSerial.h>  //Comunicaciones Serie (GPS, Bluetooth y
ROCKBLOCK)
#include <OneWire.h>   //Protocolo OneWire para los sensores de
temperatura
#include <BluetoothSerial.h> //Permite crear objetos para comunicacion
Bluetooth
#include <DallasTemperature.h> //Obtencion de datos de sensores de
temperatura
#include <TinyGPSPlus.h> //Obtencion de posicion GPS
#include <esp_task_wdt.h> //Control de watchdog

//Pines a utilizar para los sensores
#define TempPin 33
#define led_pin 18
#define PinW 22

//Parametros para anemometro
#define ANEMOMETER_ADDRESS 0x28
#define WIND_SPEED_20_PULSE_SECOND 1.75
#define ONE_ROTATION_SENSOR 20.0

//Parámetros para dormir la ESP32
#define uS_TO_S_FACTOR 1000000
#define TIME_TO_SLEEP  60

//Pines para comunicacion serie del Beitian-GPS
#define TX_GPS 17
#define RX_GPS 4

//Pines para comunicacion serie del ROCKBLOCK+
#define ROCKBLOCK_RX_PIN 1
#define ROCKBLOCK_TX_PIN 3

//Define el tiempo en segundos para reset en caso de bloqueo
#define WDT_TIMEOUT 1000

//Declaracion de objetos y variables
BluetoothSerial SerialBT;
SoftwareSerial rockBlockSerial(ROCKBLOCK_RX_PIN, ROCKBLOCK_TX_PIN);
OneWire mywire(TempPin);
DallasTemperature sensorTemp(&mywire);
SoftwareSerial BeitianSerial(TX_GPS, RX_GPS);
TinyGPSPlus gps;

double temperatura1;
double temperatura2;
float WindSpeed;
volatile unsigned long Rotations;
String velocidadViento;
int latitud_opt;
int longitud_opt;
String datos;
String datos_opt;
```

```
//Referencias para las tareas
TaskHandle_t parpadeo;
TaskHandle_t temperatura;
TaskHandle_t viento;
TaskHandle_t envio;
TaskHandle_t ubicacion;

//Definicion de las tareas
void parpadear(void *parameter);
void medirTemp(void *parameter);
void medirViento(void *parameter);
void localizacion(void *parameter);
void envio_datos(void *parameter);

//Funcion para contar rotaciones del anemometro
void isr_rotation(){

    Rotations++;

}

void setup(){

  esp_task_wdt_init(WDT_TIMEOUT,true);
  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
  BeitianSerial.begin(9600);
  attachInterrupt(digitalPinToInterrupt(PinW), isr_rotation, CHANGE);
  sei();

  SerialBT.begin("ESP32Jesus");
  SerialBT.println("The device started, now you can pair it with
bluetooth!");

  pinMode(led_pin, OUTPUT);

  xTaskCreate(
    parpadear,    // Funcion a ejecutar
    "LED parpadea",   // Accion de la tarea
    2048,             // Stack size (bytes)
    NULL,             // Parametro de entrada
    0,                // Prioridad de la tarea
    &parpadeo             // Task handle
  );

}


void parpadear(void *parameter){

  digitalWrite(led_pin,HIGH);

  vTaskDelay(2000/portTICK_PERIOD_MS);

  digitalWrite(led_pin,LOW);

  vTaskDelay(2000/portTICK_PERIOD_MS);

  xTaskCreate(
    medirTemp,    // Function that should be called
    "Medir temperatura",   // Name of the task (for debugging)
    2048,             // Stack size (bytes)
```

```
    NULL,             // Parameter to pass
    0,                // Task priority
    &temperatura              // Task handle
  );

  esp_task_wdt_reset();
  esp_task_wdt_delete(parpadeo);
  esp_task_wdt_add(temperatura);

  vTaskDelay(1/portTICK_PERIOD_MS);
  vTaskDelete(NULL);

}

void medirTemp(void *parameter){

  int temp1_opt;
  int temp2_opt;
  SerialBT.println("En task MEDIR TEMP");
  sensorTemp.requestTemperatures(); //Prepara el sensor para la
lectura
  temperatura1=sensorTemp.getTempCByIndex(0);
  temperatura2=sensorTemp.getTempCByIndex(1);

  temp1_opt = (int)(temperatura1*10);
  temp2_opt =(int) (temperatura2*10);

  esp_task_wdt_reset();
  esp_task_wdt_delete(temperatura);


  datos =(String)temperatura1 + "," + (String)temperatura2;
  datos_opt = (String)temp1_opt + "," + (String)temp2_opt;

  xTaskCreate(
    medirViento,
    "Medir viento",
    2048,
    NULL,
    0,
    &viento
  );

  vTaskDelay(1/portTICK_PERIOD_MS);
  vTaskDelete(NULL);


}

void medirViento(void *parameter){

  esp_task_wdt_add(viento);

  int viento_opt;
  SerialBT.println("En task MEDIR VIENTO");
  WindSpeed = WIND_SPEED_20_PULSE_SECOND / ONE_ROTATION_SENSOR *
(float)Rotations;
  velocidadViento = String(WindSpeed);
  viento_opt = (int)(WindSpeed * 10);
  Rotations = 0;
```

```
  datos = datos + "," + velocidadViento;
  datos_opt = datos_opt + "," +(String)viento_opt;

  SerialBT.println(datos);

  esp_task_wdt_reset();
  esp_task_wdt_delete(viento);


  xTaskCreate(
      localizacion,       // Function that should be called
      "Enviar bluetooth", // Name of the task (for debugging)
      2048,               // Stack size (bytes)
      NULL,               // Parameter to pass
      0,                  // Task priority
      &ubicacion          // Task handle
  );

  vTaskDelay(1 / portTICK_PERIOD_MS);
  vTaskDelete(NULL);

}

void localizacion(void *parameter){

  esp_task_wdt_add(ubicacion);

 while(1){

    while (BeitianSerial.available() > 0){
      gps.encode(BeitianSerial.read());
    }

  latitud_opt = (int)(gps.location.lat() * 100000);
  longitud_opt = (int)(gps.location.lng() * 100000);

  if(latitud_opt!=0){

   if(longitud_opt!=0){

    String dato_lat= (String) latitud_opt;
    String dato_lon= (String) longitud_opt;

    SerialBT.println("Latitud: " + dato_lat);
    SerialBT.println("Longitud: " + dato_lon);

    datos = datos + "," + gps.location.lng() + "," +
gps.location.lng();
    datos_opt = datos_opt + "," + latitud_opt + "," + longitud_opt;

    esp_task_wdt_reset();
    esp_task_wdt_delete(ubicacion);

    xTaskCreate(
      envio_datos,        // Function that should be called
      "Enviar bluetooth", // Name of the task (for debugging)
      2048,               // Stack size (bytes)
      NULL,               // Parameter to pass
      0,                  // Task priority
      &envio              // Task handle
```

```
);

    vTaskDelay(1 / portTICK_PERIOD_MS);
    vTaskDelete(NULL);

   }

  }

  esp_task_wdt_reset();

  }

}

//FIN DEL BLOQUE DE MEDICION E INICIO DEL BLOQUE DE COMUNICACIÓN

void envio_datos(void *parameter){

  esp_task_wdt_add(envio);

  SerialBT.println("En task ENVIO DATOS");

  rockBlockSerial.begin(19200);
  SerialBT.println("Iniciando");
  vTaskDelay(5000/portTICK_PERIOD_MS);

  // Inicializar el módulo RockBLOCK+
  rockBlockSerial.println("AT");  // Enviar comando AT para verificar
la conexión
  SerialBT.println("AT");
  vTaskDelay(1000/portTICK_PERIOD_MS);

    if (rockBlockSerial.find("OK")) {
      SerialBT.println("RockBLOCK+ listo");
    } else {
      SerialBT.println("Error al inicializar RockBLOCK+");
      while(1);
    }

  // Enviar mensaje a través de RockBLOCK+
  rockBlockSerial.println("AT+SBDWT="+datos_opt); // Escribir el
mensaje a enviar
  SerialBT.println("AT+SBDWT="+datos_opt);
  vTaskDelay(1000/portTICK_PERIOD_MS);
  rockBlockSerial.println("AT+SBDIX"); // Enviar comando para iniciar
la transmisión
  SerialBT.println("AT+SBDIX");
  vTaskDelay(1000/portTICK_PERIOD_MS);

  // Esperar a recibir la respuesta del módulo RockBLOCK+
  while (!rockBlockSerial.available()){
    vTaskDelay(1000/portTICK_PERIOD_MS);
    }

  // Leer la respuesta del módulo RockBLOCK+
  String response = "";
  while (rockBlockSerial.available()) {
    char c = rockBlockSerial.read();
    response += c;
    }
```

```
  SerialBT.println("Respuesta RockBLOCK+:");
  SerialBT.println(response);

  SerialBT.println("Datos originales: " + datos);
  SerialBT.println("Datos optimizados: " + datos_opt);

  vTaskDelay(3000/portTICK_PERIOD_MS);

  esp_task_wdt_reset();
  esp_task_wdt_delete(envio);

  SerialBT.println("A dormir");
  esp_deep_sleep_start();

}

void loop(){

}
```

Anexo II. Formulario para añadir datos de forma manual.

```
<html>
<head>
    <title>Envio POST</title>
</head>
<body>
    <h1>Envio POST</h1>
    <form method="post" action="http://proyectocolonium.rf.gd/wp-
json/api/login">
        <label for="dato">Dato:</label>
        <input type="text" name="data" id="data"><br><br>
        <label for="fecha_hora">Fecha y Hora:</label>
        <input type="text" name="transmit_time"
id="transmit_time"><br><br>
        <input type="submit" value="Enviar">
    </form>
</body>
</html>
```

Anexo III. Endpoint y recepción solicitudes HTTP POST

```
add_action('rest_api_init', function () {
    register_rest_route('api', 'login', array(
        'methods'  => 'POST',
        'callback' => 'receiver',
    ));
});

function receiver(WP_REST_Request $request) {

    $allowed_ips = array(
        '127.0.0.1', // Local IP address
        '109.74.196.135', //RockBlock IPs
        '212.71.235.32'
    );

    $remote_ip = $_SERVER['REMOTE_ADDR'];

    //Check if the remote IP is in the whitelist
    $ip_in_whitelist = false;
    foreach ($allowed_ips as $allowed_ip) {
        if (ip_in_range($remote_ip, $allowed_ip)) {
            $ip_in_whitelist = true;
            break;
        }
    }

    if (!$ip_in_whitelist) {
        $response = array(
            'error' => 'Access denied. Your IP address (' . $remote_ip
. ') is not allowed.',
        );
        return new WP_REST_Response($response, 403);
    }
    // Obtener los datos del cuerpo de la solicitud
    $data = $request->get_param('data');
    $transmit_time = $request->get_param('transmit_time');

    $decodedData = hex2bin($data);
    // Separate the values between commas
    $values = explode(',', $decodedData);

    // Extract the variables
    $temp1 = floatval($values[0]) / 10;
    $temp2 = floatval($values[1]) / 10;
    $viento = floatval($values[2]) / 10;
    $latitud = floatval($values[3]) / 100000;
    $longitud = floatval($values[4]) / 100000;

    // Datos de conexión a la base de datos
    $servername = 'sql204.byetcluster.com';
    $username = '34324725_1';
    $password = '[90Qp7SU3-';
    $dbname = 'epiz_34324725_w776';

    // Crear la conexión a la base de datos
    $conn = new mysqli($servername, $username, $password, $dbname);

    // Verificar si se estableció la conexión correctamente
    if ($conn->connect_error) {
```

```
// Error al conectar a la base de datos
        $response = array(
            'error' => 'Error al conectar a la base de datos',
        );
        return new WP_REST_Response($response, 500);
    }

    // Nombre de la tabla en la que deseas insertar los datos
    $table_name = 'mediciones';

    // Crear la consulta SQL para insertar los datos en la tabla
    $sql = "INSERT INTO $table_name (temp1, temp2, viento, latitud,
longitud, fecha_hora)
            VALUES ('$temp1', '$temp2', '$viento', '$latitud',
'$longitud', '$transmit_time')";

    // Ejecutar la consulta SQL
    if ($conn->query($sql) === TRUE) {
        // Datos insertados correctamente en la base de datos
        $response = array(
            'message' => 'Datos recibidos y guardados correctamente',
            'data' => array(
                'data' => $data,
                'transmit_time' => $transmit_time,
            ),
        );
        return new WP_REST_Response($response, 200);
    } else {
        // Error al insertar los datos en la base de datos
        $response = array(
            'error' => 'Error al insertar los datos en la base de
datos',
        );
        return new WP_REST_Response($response, 500);
    }

    // Cerrar la conexión a la base de datos
    $conn->close();
}

function ip_in_range($ip, $range) {
    list($subnet, $bits) = explode('/', $range);
    $subnet = ip2long($subnet);
    $ip = ip2long($ip);
    $mask = -1 << (32 - $bits);
    $subnet &= $mask;
    return ($ip & $mask) == $subnet;
}
```

Anexo IV. Generación de Gráfica. Ejemplo con Temp1

```php
<?php

// Conexión a la base de datos de MySQL
$servername = 'sql204.byetcluster.com';
$username = '34324725_1';
$password = '[90Qp7SU3-';
$dbname = 'epiz_34324725_w776';

$conn = mysqli_connect($servername, $username, $password, $dbname);
if (!$conn) {
    die("Conexión fallida: " . mysqli_connect_error());
}

// Consulta a la base de datos para obtener los datos de la tabla de
mediciones
$sql = "SELECT temp1, fecha_hora FROM mediciones";
$result = mysqli_query($conn, $sql);

// Crear un array con los datos de temp1 y fecha_hora
$data = array();
while($row = mysqli_fetch_assoc($result)) {
    $temp1 = $row['temp1'];
    $fecha_hora = $row['fecha_hora'];
    $data[] = array('x' => $fecha_hora, 'y' => $temp1);
}

// Cerrar la conexión a la base de datos
mysqli_close($conn);

// Imprimir el script para generar la gráfica
?>

<!DOCTYPE html>
<html>
<head>
    <title>Gráfico1</title>
    <style>
        .container {
            display: flex;
            flex-direction: row;
                  justify-content: center;
            margin: 0 auto;
                  width:1000px;

        }

        .filter {
            flex: 1;
            margin-right: 20px;
        }

        .chart {
            flex: 2;
        }

        #myChart1 {
            width: 800px;
```

```
width: 800px;
            height: 400px;
        }
    </style>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
    <div class="container">
        <div class="chart">
            <canvas id="myChart1"></canvas>
        </div>
    </div>

    <script>
        var ctx = document.getElementById('myChart1').getContext('2d');
        var chart = new Chart(ctx, {
            type: 'bar',
            data: {
                datasets: [{
                    label: 'Temp1',
                    data: <?php echo json_encode($data); ?>,
                    backgroundColor: 'rgba(255, 0, 0, 0.2)',
                    borderColor: 'rgba(255, 0, 0, 1)',
                    borderWidth: 1
                }]
            },
            options: {
                scales: {
                    xAxes: [{
                        type: 'time',
                        time: {
                    displayFormats: {
                        day: 'DD/MM/YYYY HH:mm'
                    },
                    tooltipFormat: 'DD/MM/YYYY HH:mm'
                },
                        distribution: 'linear'
                    }],
                    yAxes: [{
                        ticks: {
                            beginAtZero: true
                        }
                    }]
                },
                responsive: true,
                maintainAspectRatio: true,
                width: 800,
                height: 400
            }
        });

    </script>
</body>
</html>
```

Anexo V. Generación del mapa.

```
<?php

global $wpdb;
$results = $wpdb->get_results("SELECT latitud, longitud, fecha_hora,
temp1, temp2, viento FROM mediciones ORDER BY fecha_hora ASC");

// Incluye la librería de Leaflet
wp_enqueue_script('leaflet');

// Inicializa el mapa
echo do_shortcode('[leaflet-map lat=28.12888 lng=-15.4465
height="500px" zoomcontrol=1 zoomwheel="yes"]');

// Crea un array para almacenar las ubicaciones
$locations = array();

// Agrega un pin por cada medición y almacena la ubicación en el array
foreach ($results as $row) {
    $latitud = $row->latitud;
    $longitud = $row->longitud;
    $fechaHora = $row->fecha_hora;
    $temp1 = $row->temp1;
    $temp2 = $row->temp2;
    $viento = $row->viento;

    $fechaHora = date('Y-m-d H:i:s', strtotime($fechaHora));

    echo do_shortcode('[leaflet-marker lat=' . $latitud . ' lng=' .
$longitud . ' visible]
        <span class="tooltip">' . $fechaHora . '</span>
    [/leaflet-marker]');

}
?>
<!DOCTYPE html>
<html>
<head>
  <title>Desplegable</title>
  <style>
    #desplegable {
      position: relative;
      top: -500px; /* Ajusta la posición vertical relativa */
      left: 250px; /* Ajusta la posición horizontal relativa */
    }
    #valores {
      position: relative;
      top: -500px; /* Ajusta la posición vertical relativa */
      right: 300px; /* Ajusta la posición horizontal relativa */
    }
  </style>
  <script>
    function mostrarValores() {
      var desplegable = document.getElementById("desplegable");
      var fechaHoraSeleccionada = desplegable.value;

      // Obtener los valores de temp1, temp2 y viento asociados a la
fecha y hora seleccionada
      var valores = <?php echo json_encode($results); ?>;
      var temp1Seleccionada = "";
      var temp2Seleccionada = "";
      var vientoSeleccionado = "";
```

```
for (var i = 0; i < valores.length; i++) {
        if (valores[i].fecha_hora === fechaHoraSeleccionada) {
          temp1Seleccionada = valores[i].temp1;
          temp2Seleccionada = valores[i].temp2;
          vientoSeleccionado = valores[i].viento;
          break;
        }
      }

      // Mostrar los valores en elementos HTML
      document.getElementById("temp1").innerHTML = temp1Seleccionada;
      document.getElementById("temp2").innerHTML = temp2Seleccionada;
      document.getElementById("viento").innerHTML =
vientoSeleccionado;
    }
  </script>
</head>
<body>
  <select id="desplegable" onchange="mostrarValores()">
    <option value="">Selecciona una opción</option>
    <?php
    // Mostrar los valores en el desplegable
    foreach ($results as $row) {
        $fechaHora = $row->fecha_hora;
        $fechaHora = date('Y-m-d H:i:s', strtotime($fechaHora));
        echo "<option value='$fechaHora'>$fechaHora</option>";
    }
    ?>
  </select>

  <div id="valores">
    <h3>Valores:</h3>
    <p>Temp1: <span id="temp1"></span></p>
    <p>Temp2: <span id="temp2"></span></p>
    <p>Viento: <span id="viento"></span></p>
  </div>
</body>
</html>
```

PRESUPUESTO

P.1 .Introducción

En esta parte del documento se calculará el coste económico asociado al desarrollo de este TFG. Se puede desglosar el coste total en los costes de:

- Recursos hardware.
- Recursos software.
- Recursos humanos.
- Redacción del documento.

El periodo de desarrollo de este TFG ha sido de 4 meses, periodo muy inferior a los 3 años que se estipulan para el coste de amortización. Por dicha razón se calculan los costes sobre la base de los derivados de los primeros 4 meses.

P.2 .Recursos hardware

La siguiente tabla muestra los recursos hardware con su vida útil aproximada, posteriormente en la Tabla 8 se muestra el coste de todo el equipo hardware en el momento de adquisición, junto con la amortización anual y el coste final de cada producto.

Recurso hardware	Vida útil (años)
MSI GF63	5
ESP32-WROOM-32D	7
Protoboard	10
Sensor DS18B20	10
Anemómetro de tres tazas	7
Beitian BN880	6
Rockblock+	6

Tabla 7. Recursos hardware y vida útil

Objeto	**Costo de adquisición**	**Amortización**
MSI GF63	€1,000	€166
ESP32-WROOM-32D	€8	€0.80
Protoboard	€12	€1.50
Sensor DS18B20	€1,08	€0.10
Anemómetro de tres tazas	€38,54	€3,85
Beitian BN880	€13,88	€2,31
Rockblock+	€370	€61,66

Tabla 8. Coste de recursos hardware

El coste de los recursos hardware asciende a DOS CIENTOS TREINTA Y SEIS EUROS CON VEINTIDÓS CÉNTIMOS.

P.3 .Recursos software

La siguiente tabla, recoge los recursos software al igual que su vida útil y seguidamente se encuentra la Tabla 10, que muestra el coste de los equipos software en el momento de la adquisición, su amortización anual y el coste final de cada dispositivo

Recurso software	Vida útil (años)
SO Windows 11 pro	10
Anualidad Office 365	Continua
VS Code	Continua
Infinity Host	Continua

Tabla 9. Recursos software y vida útil.

Recurso	Costo de adquisición	Amortización
SO Windows 11 (licencia Pro)	€259	€25.9
Office 365 (suscripción anual)	€59	€59
VS Code	€0	N/A (sin costo anual)
Infinity Host (plan gratuito)	€0	N/A (sin costo anual)

Tabla 10. Coste de recursos software

El coste final de los recursos software asciende a OCHENTA Y CUATRO EUROS CON NUEVE CÉNTIMOS.

P.4. Recursos humanos

Según “RETRIBUCIONES DEL PERSONAL INVESTIGADOR, DEL PERSONAL TÉCNICO Y DEL PERSONAL TÉCNICO DE APOYO CONTRATADO DE LA ULPGC CON CARGO A PROYECTOS, CONVENIOS Y CONTRATO” del BOULPGC PARA EL AÑO 2023” del BOULPGC a 3 de febrero de 2023 [15], el sueldo base de un técnico con formación a nivel de grado a 20 horas semanales es de €745,15, para 4 meses de trabajo suma un coste final de DOS MIL NOVECIENTOS OCHENTA EUROS CON SESENTA CÉNTIMOS.

P.5. Redacción del documento

Según las horas asignadas a la redacción del documento en el anteproyecto y siguiendo el coste por hora del ingeniero técnico según el BOULPGC a 3 de febrero de 2023 [15], los costes de redacción se recogen en la

Redacción			
Personal	Coste por hora	Horas	Total
Ingeniero Técnico	€9,30	40	€ 372,20

Tabla 11. Costes de redacción del documento

Por tanto, el coste total de la redacción del documento asciende a TRESCIENTOS SETENTA Y DOS EUROS CON VEINTE CÉNTIMOS.

P.6. Aplicación de impuestos y coste final

Presupuesto Final	
Partidas	Totales
Recursos Hardware	€236,22
Recursos Software	€84,9
Recursos Humanos	€2970,80
Total parcial	€3291,92
Redacción del documento	€372,20
IGIC (7%)*	€256,48
Total aplicando IGIC	€ 3.548,40

Tabla 12. Coste total tras impuestos

*Todos los costes materiales anteriormente calculados no incluían el IGIC.

El coste total asciende a TRES MIL QUINIENTOS CUARENTA Y OCHO EUROS CON CUARENTA CÉNTIMOS.