Este Trabajo de Fin de Grado trata sobre la recogida y muestra de datos provenientes de sensores ubicados en un prototipo de exoesqueleto (ExoBOOT).

Estas páginas recogen todo el desarrollo orientado al producto tanto, desde la identificación de los requisitos, hasta el diseño de la aplicación web encargada de la visualización de los datos, como del firmware encargado de gestionar ExoBOOT.

Ginés Navarrete Campos, nació en Sabiote (Jaén) en 1994. Comenzó a trabajar con el equipo de GranaSAT en 2019, lo cual le ha permitido colaborar en otros proyectos, como el proyecto educativo CanSAT, perteneciente a ESERO, en el que colabora el Paque de las Ciencias de Granada (España). Con este desafiante Trabajo Fin de Grado mejora su educación en otras especialidades y finaliza su Título de Grado en Ingeniería Informática en la Universidad de Granada (España).

Andrés María Roldán Aranda, es la persona que acamémicamente dirige el presente proyecto, y el tutor del estudiante. Es profesor en el Departamento de Electrónica y Tecnología de Computadores

Copia para el alumno / Copy for the student

TRABAJO FIN DE GRADO

Sistema de monitorización de señales basado en ESP32

Ginés Navarrete Campos

INGENIERÍA INFORMÁTICA

2020/21

UNIVERSIDAD DE GRANADA

Grado en
Ingeniería Informática

Trabajo Fin de Grado

Sistema de monitorización de señales basado en ESP32

Ginés Navarrete Campos

2020/2021

Tutor: Andrés María Roldán Aranda

GRADUADO EN
INGENIERÍA INFORMÁTICA

Tesis del Grado

"Sistema de monitorización de señales basado en ESP32"

CURSO ACADÉMICO: 2020/2021

Ginés Navarrete Campos

GRADUADO EN INGENIERÍA INFORMÁTICA

"Sistema de monitorización de señales basado en ESP32"

AUTOR:

Ginés Navarrete Campos

SUPERVISADO POR:

Andrés María Roldán Aranda

DEPARTAMENTO:

Electrónica y Tecnología de los Computadores

D. Andrés María Roldán Aranda, Profesor del departamento de Electrónica y Tecnología de los Computadores de la Universidad de Granada, como director del Trabajo Fin de Grado de D. Ginés Navarrete Campos,

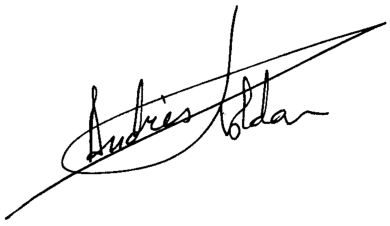
Informa:

Que el presente trabajo, titulado:

"Sistema de monitorización de señales basado en ESP32"

ha sido realizado y redactado por el mencionado alumno bajo mi dirección, y con esta fecha autorizo a su presentación.

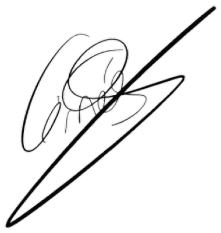
Granada, a 17 de noviembre de 2020



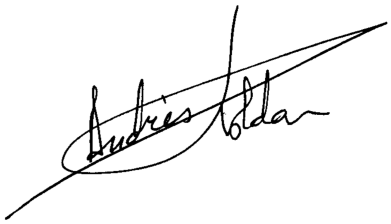
Fdo. Andrés María Roldán Aranda

Los abajo firmantes autorizan a que la presente copia de Trabajo Fin de Grado se ubique en la Biblioteca del Centro y/o departamento para ser libremente consultada por las personas que lo deseen.

Granada, a 17 de noviembre de 2020



Fdo. Ginés Navarrete Campos



Fdo. Andrés María Roldán Aranda

Sistema de monitorización de señales basado en ESP32

Ginés Navarrete Campos

PALABRAS CLAVE:

Arduino, ExoBOOT,firmware, GranaSAT, FreeRTOS, hardware, software, NodeJS, ExpressJS, websocket, PUG, ESP32 .

RESUMEN:

Este proyecto parte de la investigación del Ministerio de Defensa y la Universidad de Granada [9], incluyendo de esta forma a GranaSAT, tratándose de desarrollar un sistema que facilite los desplazamientos de los soldados y reduzca los niveles de cansancio en largas caminatas. Para el correcto desarrollo de esto entran en juego los diferentes sensores de los que dispone. Por ello el principal objetivo de este proyecto es integrar en una aplicación web, un sistema capaz de recibir información de dichos sensores para que quien lo requiera pueda visualizarlos. De este modo, el objetivo será implementar un firmware capaz de leer de los sensores, teniendo en cuenta las limitaciones de la placa ESP32, así como desarrollar esa aplicación web en la que se visualizarán de forma gráfica.

De este modo, para el firmware, se abandona la programación en alto nivel para tener total control del hardware, debido a sus limitados recursos.

Es por esto que se pretende emular un encargo profesional real, siguiendo una metodología orientada al producto. Las diversas decisión se tomarán a partir de los requisitos del cliente, GranaSAT. Partiendo de estos se hará un análisis para ofrecer las mejores soluciones posibles en función a los recursos establecidos.

Finalmente, se llevará a cabo el diseño, realizando las implementaciones necesarias para la correcta visualización de los datos y comunicación tanto de firmware, como de software. Terminando con las de conclusiones y posibles mejoras al respecto.

Se ha tratado de enfocar el proyecto de forma multidisciplinar, tratando de adquirir nuevas técnicas y aptitudes, a parte de las utilizadas durante la titulación. Por último, se culmina con la obtención de un sistema, que cumple con los objetivos y requisitos de este proyecto y con el cual se cierra la etapa universitaria de Grado en Ingeniería Informática.

ESP32-based signal monitoring system

Ginés Navarrete Campos

KEYWORDS:

Arduino, ExoBOOT,firmware, GranaSAT, FreeRTOS, hardware, software, NodeJS, ExpressJS, websocket, PUG, ESP32 ..

ABSTRACT:

This project is based on the investigation of the Ministry of Defense and the University of Granada [9], thus including GranaSAT, trying to develop a system that facilitates the movement of soldiers and reduces levels from tiredness on long walks. For the correct development of this, the different sensors available to it come into play. Therefore, the main objective of this project is to integrate in a web application, a system capable of receiving information from said sensors so that whoever requires it can view them. In this way, the objective will be to implement a firmware capable of reading the sensors, taking into account the limitations of the ESP32 board, as well as developing that web application in which it will be displayed graphically.

Thus, for firmware, high-level programming is abandoned to have full control of hardware, due to its limited resources.

This is why it aims to emulate a real professional commission, following a product-oriented methodology. The various decisions will be made based on the customer's requirements, GranaSAT. Based on these, an analysis will be made to offer the best possible solutions based on the established resources.

Finally, a design will be carried out, making the necessary implementations for the correct visualization of the data and communication of both firmware and software. Ending with the conclusions and possible improvements in this regard.

An attempt has been made to focus the project in a multidisciplinary way, trying to acquire new techniques and skills, some of those used during the degree. Finally, it culminates in obtaining a system that meets the objectives and requirements of this project and with which the university stage of the Degree in Computer Engineering is closed.

"Llegar a la meta no es vencer,
lo importante es el camino y en él caer,
levantarse, insistir y aprender."

Agradecimientos:

En primer lugar agradecer a mi madre todo lo que ha hecho siempre para poder brindarme la oportunidad de conseguir llegar hasta aquí y a mi hermana por estar siempre ahí. También agradecerles todo el apoyo que me han dado durante este tiempo, porque sin vosotras a mi lado esto no hubiese sido posible. Sois el mayor pilar de mi vida y mi motivación diaria para mejorar.

Como no, agradecer a Elena G. S. todo su esfuerzo y apoyo por ayudarme en todo lo que necesitaba, sin rechistar en todo este tiempo.

Creo también necesario agradecer a mis compañeros y compañeras, esos que comprenden tanto como yo lo difícil y frustrante que puede llegar a ser este grado, porque si no lo vives en primera persona no puedes ni imaginar lo que puede llegar a suponer.

Finalmente, me gustaría agradecer a mi tutor Andrés Roldán, por el trato, la dedicación y todas las oportunidades que me ha brindado, que han sido muchas, tanto a nivel docente como profesional. Al igual que a los estudiantes que han pasado por el laboratorio, por ilustrarme con sus ideas.

Lo más importante está por venir.

Acknowledgments:

First of all, I would like to thank my mother for everything she has always done to give me the opportunity to get here and to my sister for always being there. Also thank you for all the support you have given me during this time, because without you by my side this would not have been possible. You are the greatest pillar of my life and my daily motivation to improve.

Of course, thank Elena G. S. for all her effort and support for helping me in everything I needed, without questioning all this time.

I also think it necessary to thank my classmates, those who understand as much as I do how difficult and frustrating this grade can be, because if you don't experience it in the first person, you can't even imagine what it might mean.

Finally, I would like to thank my tutor Andrés Roldán, for the treatment, dedication and all the opportunities she has given me, which have been many, both at a teaching and professional level. As well as the students who have passed through the laboratory, for enlightening me with their ideas.

The most important is yet to come.

ÍNDICE

LISTA DE FIGURAS

ÍNDICE CÓDIGOS

LISTA DE TABLAS

GLOSARIO

API Una interfaz de programación de aplicaciones o API (del inglés Application Programming Interface) es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Son usadas generalmente en las bibliotecas (también denominadas comúnmente *librerías*).

Arduino Compañía open source de hardware y software, así como un proyecto y comunidad internacional que diseña y manufactura placas de desarrollo de hardware para construir dispositivos digitales y dispositivos interactivos que puedan sensar y controlar objetos del mundo real.

bootable Modo de arranque de sistemas informáticos.

buffer Espacio en memoria destinado a almacenar información.

C Lenguaje de programación.

CSS Hojas de estilo usadas en el diseño gráfico con el objetivo de obtener un documento estructurado..

ESP32 Microcontrolador de bajo coste y consumo energético con tecnología WiFi y Bluetooth, diseñado por Esspresif Systems.

ESP8266 Microchip de bajo coste diseñado por Esspresif Systems y predecesor del ESP32.

Esspresif Systems Multinacional pública que desarrolla semiconductores.

ExoBOOT Nombre del prototio de exoesqueleto del proyecto de investigación del Ministerio de Defensa y la Universidad de Granada.

ExpressJS Infraestructura de aplicaciones web en NodeJS..

firmware Es la lógica a más bajo nivel para controlar el hardware.

footprint Utilizado para conectar los componentes a una paca de circuito impreso..

framework Estructura conceptual con módulos de software concretos, para la realización y organización de proyectos..

FreeRTOS API de código abierto que pone a nuestra disposición una serie de herramientas para construir un planificador y de esta forma tener la posibilidad de frabricar nuestro propio RTOS [4].

GPIO Pin genérico en un chip, cuyo comportamiento se puede programar..

GranaSAT GranaSAT grupo de la Universidad de Granada. Coordinado por el profesor Andrés María Roldán Aranda, GranaSAT es un proyecto multidisciplinar con estudiantes de diferentes grados, donde pueden adquirir y ampliar los conocimientos necesarios para enfrentar un proyecto aeroespacial real.

hardware Conjunto de elementos físicos o materiales que constituyen una computadora o un sistema informático.

IMC Sensor de proximidad inductivos..

IMU Unidad de medición inercial, incluye sensor de acelerómetro, giroscopo, magnetómetro y temperatura..

JSON Notación de objeto de JavaScript. Formato utilizado para el intercambio de datos..

Linux Sistema Operativo de código abierto, es de tipo UNIX, multiplataforma, multiusuario y multitarea.

MacOs Sistema operativo de tipo UNIX diseñado por Apple.

NodeJS Entorno de ejecución multiplataforma, para el código del servidor y basado en JavaScript.

NodeMCU Placa de desarrollo de IoT.

OTA Actualizaciones de software de forma inalámbrica..

PUG Motor de plantillas para realización de páginas web implementado con JavaScript y para NodeJS..

software Conjunto de programas y rutinas que permiten a la computadora u otro dispositivo realizar determinadas tareas.

SRAM Memoria RAM estática, mantiene los datos mientras siga conectada..

TMB Tasa metabólica basal, es el gasto energético diario. La energía que necesita el cuerpo para funcionar durante el día.

TSMC Multinacional de semiconductores..

websocket Protocolo de comunicación bajo TCP..

Windows Sistema operativo realizado por Microsoft™.

ACRÓNIMOS

ACID Atomicity, Consistency, Isolation and Durability: Atomicidad, Consistencia, Aislamiento y Durabilidad.

FIFO First In First Out. Primero en Entrar, Primero en Salir.

RTOS Real Time Operative System. Sistema Operativo en Tiempo Real.

UGR Universidad de Granada.

CAPÍTULO

1

INTRODUCCIÓN

1.1 Contexto

El presente proyecto se ha realizado en GranaSAT, un grupo multidisciplinario de la Universidad de Granada (UGR) creado para estudiantes de diferentes ramas y grados, desde Informática o Telecomunicaciones, hasta Física o Electrónica.

Entre otros proyectos que están a cargo de este grupo, se encuentra el que se va a desarrollar en estas páginas, como parte del Trabajo Fin de Grado, titulado: *"Sistema de monitorización de señales basado en ESP32"*. Este parte de una iniciativa de de desarrollo del Ministerio de Defensa y consta de una bota militar a la que se le adhiere una estructura fabricada con PLA, material que se utiliza en la impresión 3D, método de fabricación de dicha estructura. Esta soportará un muelle creado con una banda elástica, además de diferentes sensores para monitorizar dicho muelle, así como la posición del pie.

Es por ello, que el proyecto al que pertenece este trabajo está dividido en dos partes, la parte hardware, que consiste en la unión de la bota con los diferentes componentes y la otra parte compuesta por software y firmware, que es la que se tratará aquí, gestionando el hardware, así como la aplicación de escritorio que hace posible la recepción y visualización de los datos referentes a los sensores para realizar un correcto estudio y poder desarollar un dispositivo capaz de ayudar a evitar ese cansancio.

Andrés María Roldán Aranda es el dirigente de GranaSAT, así como el coordinador del presente proyecto.

1

Figura 1.1 – *Logo de GranaSAT.*

Las herramientas necesarias para el presente proyecto se encuentran en el edificio iMUDS, junto al Parque Tecnológico de la Salud, en Granada (España). El proyecto, sobretodo el montaje de ExoBOOT, ha sido principalmente desarrollado en este laboratorio. Desarrollado en una última instancia desde casa, con la ayuda del servidor propio de GranaSAT y el gitlab privado ubicado en él para compartir los avances con Andrés Roldán, además de una comunicación constante vía Telegram.

1.2 Motivación

La principal motivación por la que ha sido abordado este tema es que se trata de algo totalmente diferente a lo enseñado en el Grado de Ingeniería Informática, ya que en dicho grado el gran peso del estudio, a nivel de programación recae en un alto nivel, sin tener en cuenta los recursos de los que se dispone.

En este caso, dado que ExoBOOT será un sistema en tiempo real que dispondrá de un microcontrolador con recursos muy limitados en referencia a la memoria y al procesador, por lo que es necesaio realizar una programación teniendo en cuenta todo lo posible tanto las variables usadas como las funciones y cómo se ejecuta.

Por ello, este trabajo me ayudará en gran medida a sobrepasar ese límite como futuro ingeniero y me servirá para ampliar mis conocimientos.

Es un motivo de orgullo pensar que este proyecto servirá de base para estudios futuros y que forma parte de un proyecto tan importante

1.3 Objetivos del proyecto

Los principales objetivos son:

- Establecer los principales requisitos que debe tener el proyecto para su correcto desarrollo.

- Realizar un profundo análisis de lo que ofrece el mercado, así como de las diferentes herramientas y frameworks en la que existe la libertad de decidir.
- Diseñar y desarrollar tanto el software, como el firmware. Aprendiendo a programar sobre los limitados recursos del microcontrolador ESP32, asi como la correcta comunicación entre estos y porder realizar el envío y lectura de datos reportados por los sensores.
- Comprobar los resultados obtenidos, verificando que se cumple la funcionalidad y que todo lo hace de forma adecuada.
- Demostrar el conocimiento adquirido a lo largo del Grado, no solo en la especialidad cursada, si no desde un punto de vista general.

1.4 Estructura del proyecto

El proyecto se estructura en 6 capítulos y un apéndice formado por 2 capítulos más. Todo está organizado de un modo lógico y cronológico. Empezando por la definición de los requisitos del sistema por parte del usuario, siguiendo por un análisis con las diferentes opciones a elegir, continuando por el diseño y desarrollo que llevan a completar con éxito los objetivos y finalizando por la evaluación y conclusiones del proyecto.

- **Capítulo uno**: Capítulo actual, donde se realiza una introducción del proyecto, así como las motivaciones, objetivos y planteamiento del Proyecto.
- **Capítulo dos**: En este capítulo se lleva a cabo el estudio de los requisitos, donde se establen aquellos que se deben cumplir y estar presentes en el resultado final.
- **Capítulo tres**: Aquí se procede a analizar el sistema, las opciones que ofrece el mercado, la existencia de proyectos parecidos y las decisiones justificadas en torno al proyecto completo. Incluyendo pruebas de rendimiento entre los posibles microcontroladores NodeMCU.
- **Capítulo cuatro**: Se destina al diseño llevado a cabo para el sistema. Empezando por superficial análisis del desrrollo hardware, para entender las bases y proceder al correcto desarrollo del firmware, documentado su diseño en este capítulo, al igual que el software.
- **Capítulo cinco**: capítulo destinado a las evaluaciones del sistema, realizadas para verificar que el sistema cumple con los objetivos y requisitos establecidos.
- **Capítulo seis**: Capítulo final, donde se concluye el proyecto con una reflexión y algunas futuras mejoras para ExoBOOT.

1

CAPÍTULO

2

REQUISITOS DEL SISTEMA

En este proyecto, como en cualquier proyecto software, se han definido unos requisitos mínimos por parte del cliente, los cuales se deben cumplir. Esos requisitos, funcionales y no funcionales, se especifican a continuación.

2.1 Requisitos Funcionales

Estos requisitos, los cuales definen funcionalidades del proyecto se dividen en:

- **RF1**. Requisitos del usuario:
  1. Elegir tiempo a medir
- **RF2**. Requisitos del cliente, que en este caso hace referencia al dispositivo hardware:
  1. Conexión WiFi
  2. Esperar a que empiece la medición.
  3. Leer datos
  4. Enviar datos
  5. Conexión websocket cliente
- **RF3**. Requisitos del servidor:
  1. Recopilar datos recibidos
  2. Mostrar datos

3. Conexión websocket servidor

4. Almacenamiento en base de datos

En las siguientes tablas, desde la tabla 2.1 hasta la 2.10, se encuentran más detallados los requisitos enumerados anteriormente:

Tabla 2.1 – *Elegir tiempo a medir*

Número de requisito	RF1
Nombre de requisito	Elegir tiempo a medir
Tipo	☐ Requisito ☒ Restricción
Prioridad del requisito	☐ Alta/Esencial ☒ Media/Deseado ☐ Baja/ Opcional
Descripción del requisito: El usuario que usa la aplicación puede elegir el tiempo que durará la realización de la medida, los datos se recogerán y se mostrarán en las gráficas.	

Tabla 2.2 – *Conexión WiFi*

Número de requisito	RF2.1
Nombre de requisito	Conexión WiFi
Tipo	☒ Requisito ☐ Restricción
Prioridad del requisito	☒ Alta/Esencial ☐ Media/Deseado ☐ Baja/ Opcional
Descripción del requisito: El dispositivo se debe conectar a una red WiFi para poder conectarse a la aplicación de ordenador, así como para realizar la configuración oportuna y enviar la información que recogen los sensores.	

Tabla 2.3 – *Esperar a que empiece la medición*

Número de requisito	RF2.2
Nombre de requisito	Esperar a que empiece la medición
Tipo	☒ Requisito ☐ Restricción
Prioridad del requisito	☐ Alta/Esencial ☒ Media/Deseado ☐ Baja/ Opcional
Descripción del requisito: EXOBOOT debe esperar a que el sistema esté funcionando y el usuario haya introducido la configuración necesaria para proceder a realizar la medida. Cuando el usuario acepta la configuración recibirá una notificación de OK para comenzar la medida y proceder con el envío ordenado de datos.	

Tabla 2.4 – *Leer Datos*

Número de requisito	RF2.3
Nombre de requisito	Leer datos
Tipo	☒ Requisito ☐ Restricción
Prioridad del requisito	☒ Alta/Esencial ☐ Media/Deseado ☐ Baja/ Opcional
Descripción del requisito: Se debe poder leer los datos de los diferentes sensores y enviar todos los que el usuario haya solicitado y configurado en el inicio de la sesión. Posteriormente se procesarán para su envío. Estos datos llevan una marca temporal para su correcta representación e identificación.	

Tabla 2.5 – *Enviar Datos*

Número de requisito	RF2.4
Nombre de requisito	Enviar datos
Tipo	☒ Requisito ☐ Restricción
Prioridad del requisito	☒ Alta/Esencial ☐ Media/Deseado ☐ Baja/ Opcional
Descripción del requisito: Una vez leídos y procesados los datos se enviarán mediante websocket y TCP. Estos datos se envían y si en algún momento se pierde la conexión se deben almacenar en un buffer para no perder ningún dato. Mientras que este buffer de pérdidas esté lleno no se enviarán otros datos. Esta conexión es directa con el servidor.	

Tabla 2.6 – *Conexión websocket cliente*

Número de requisito	RF2.5
Nombre de requisito	Conexión websocket cliente
Tipo	☒ Requisito ☐ Restricción
Prioridad del requisito	☒ Alta/Esencial ☐ Media/Deseado ☐ Baja/ Opcional
Descripción del requisito: Exoboot y la aplicación de escritorio se comunicarán mediante websocket y el protocolo TCP para el intercambio de información. Ambos deben estar en la misma red para su correcto funcionamiento.	

Tabla 2.7 – *Recopilar datos recibidos*

Número de requisito	RF3.1
Nombre de requisito	Recopilar datos recibidos
Tipo	☒Requisito ☐ Restricción
Prioridad del requisito	☒Alta/Esencial ☐Media/Deseado ☐Baja/ Opcional
Descripción del requisito: El servidor se encarga de recopilar los datos recibidos. Los datos pueden ser de varios clientes a la vez, por lo que debe identificar al cliente que lo envía. Cada cliente se corresponde con un par de Exoboots	

Tabla 2.8 – *Mostrar Datos recibidos*

Número de requisito	RF3.2
Nombre de requisito	Mostrar datos recibidos
Tipo	☒Requisito ☐ Restricción
Prioridad del requisito	☒Alta/Esencial ☐Media/Deseado ☐Baja/ Opcional
Descripción del requisito: La aplicación mostrará, en una página web, las gráficas con los diferentes valores que ExoBOOT haya leído de sus sensores. Dichos datos deben de ir identificados por la bota que los envía y por una marca temporal que indica el instante en el que se ha leído y enviado hacia el servidor.	

Tabla 2.9 – *Conexión websocket servidor*

Número de requisito	RF3.3
Nombre de requisito	Conexión websocket servidor
Tipo	☒Requisito ☐ Restricción
Prioridad del requisito	☒Alta/Esencial ☐Media/Deseado ☐Baja/ Opcional
Descripción del requisito: La aplicación de escritorio ejecutará un backend, el cual crea un websocket que empieza a enviar la información a los clientes que se han conectado. En este caso ese cliente es la página web en la que se realiza la configuración de Exoboot y dónde se mostrarán las diferentes gráficas. Los datos se reciben en JSON, esto ayuda a procesar más rápido el dato.	

Tabla 2.10 – *Almacenamiento en la base de datos*

Número de requisito	RF3.4
Nombre de requisito	Almacenamiento en base de datos
Tipo	☒ Requisito ☐ Restricción
Prioridad del requisito	☐ Alta/Esencial ☒ Media/Deseado ☐ Baja/ Opcional
Descripción del requisito: Los datos recibidos por parte de los clientes se deben almacenar en una base de datos para un posterior estudio y poder cunsultarlos en cualquier momento, así como, para el caso que sea necesario, se pueda realizar una representación gráfica de dichos datos. Esta representación podrá ser estática o animada.	

2.2 Requisitos No Funcionales

Algunos de los requisitos no funcionales, es decir, los que imponen restricciones de diseño o implementación al sistema se pueden ver estructurados en el siguiente esquema.

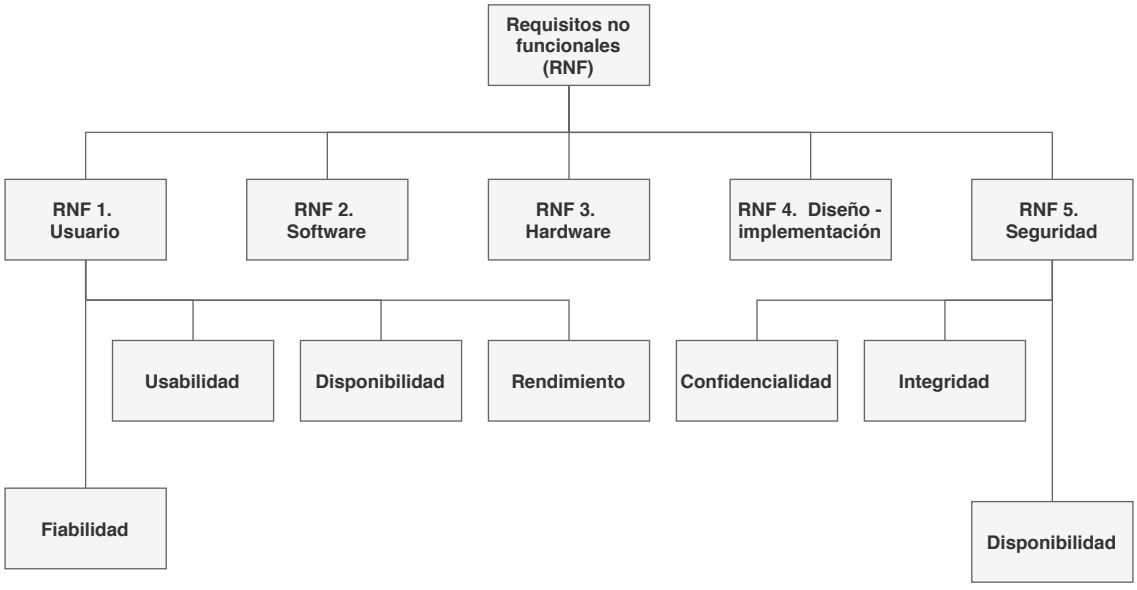


Figura 2.1 – *Esquema de los requisitos no funcionales del sistema*

Como se puede apreciar en el esquema 2.1 el sistema tiene 5 requisitos no funcionales principales y dentro del RNF 1 (tabla 2.11) y el RNF 5 (tabla 2.15) se deben cumplir otros aspectos. Estos requisitos, mostrados en la figura anterior, se encuentran descritos más detalladamente desde la tabla 2.11 hasta la tabla 2.15. Dichas tablas se encuentran a

continuación.

Tabla 2.11 – *Usuario y sistema*

Número de requisito	RNF 1
Nombre de requisito	Usuario y sistema
Tipo	☒ Requisito ☐ Restricción
Prioridad del requisito	☐ Crítico ☒ No crítico

Descripción del requisito:
Este requisito se aplica tanto al usuario como al sistema y se deben cumplir unos aspectos básicos relacionados con:

- **Fiabilidad.** Exoboot cuenta con un buffer para evitar pérdidas de información. Este buffer entra en funcionamiento cuando se pierde la conexión con el receptor. Hasta que este buffer no esté vacío no envía los datos leídos. Este buffer es de tipo FIFO, es decir, el primero en entrar es el primero en salir, actúa como una cola. En esta situación ya no sería en tiempo real, pero posteriormente y una vez terminada la medida se puede reproducir la medida en diferido si así se desea.

- **Usabilidad.** La aplicación se ejecutará a través de un programa portable, por lo que no es necesario instalarlo en el sistema. Este ejecutable mostrará una ventana con información de la web a la que hay que acceder para ver los datos y configurar la medida que se realizará en ExoBOOT.

  Los diferentes usuarios serán el cliente y quién quiere ver los datos en la web. Los datos pueden ser de medidas anteriores, que estarán almacenadas en el sistema o pueden ser archivos que se han tomado con otro ordenador. Estos archivos son la base de datos dónde se han almacenado las diferentes medidas realizadas en el espacio temporal indicado.

- **Disponibilidad.** Una vez ejecutada la aplicación, aparte del gestor de la aplicación, se podrá acceder y consultar las diferentes medidas realizadas. También se podrá ver otra información relacionada con las medias.

- **Rendimiento.** Tiene que ser capaz de dar una respuesta rápida y mostrar el contenido lo más rápido posible, es decir, debe ser un sistema en tiempo real en la medida de lo posible.

Tabla 2.12 – *Software*

Número de requisito	RNF 2
Nombre de requisito	Software
Tipo	☒ Requisito ☐ Restricción
Prioridad del requisito	☐ Crítico ☒ No crítico
Descripción del requisito: El principal requisito software será el de disponer de un ordenador con una conexión a internet, que disponga de un navegador web y el ejecutable de la aplicación. Este ejecutable ya contendrá todo lo necesario para poder ejecutar el sistema correctamente. Debe ser todo lo multiplataforma que se pueda, esto quiere decir que debe ser compatible con Windows, Linux y MacOs.	

Tabla 2.13 – *Hardware*

Número de requisito	RNF 3
Nombre de requisito	Hardware
Tipo	☒ Requisito ☐ Restricción
Prioridad del requisito	☐ Crítico ☒ No crítico
Descripción del requisito: Uno de los requisitos indispensables es la posesión del dispositivo hardware, es decir, tener ExoBOOT debidamente configurado para que pueda obtener las medias correctamente y sea posible la visualización de estas. Si no se posee el dispositivo y existen medidas realizadas, será posible verlas en formato diferido.	

Tabla 2.14 – *Diseño e implementación*

Número de requisito	RNF 4
Nombre de requisito	Diseño e implementación
Tipo	☒ Requisito ☐ Restricción
Prioridad del requisito	☐ Crítico ☒ No crítico
Descripción del requisito: El lenguaje de programación elegido para la aplicación es NodeJS y C para el firmware del ESP32, el cual monta ExoBOOT. Existe libertad para elegir la forma de desarrollar las otras partes del proyecto que sean necesarias. Entre ellas la elección de usar bases de datos relacionales o no relacionales o el framework encargado de la parte visual de la web. En esta parte visual se deben poder ver la gráficas de forma fluida y con una interfaz sencilla para la fácil y rápida compresión de los datos por parte de los usuarios.	

Tabla 2.15 – *Seguridad*

Número de requisito	RNF 5
Nombre de requisito	Seguridad
Tipo	☒ Requisito ☐ Restricción
Prioridad del requisito	☐ Crítico ☒ No crítico

Descripción del requisito:
Las restricciones de seguridad deben cumplir:

- **Confidencialidad** Los datos están almacenados confidencialmente y con seguridad. El envío de los datos se realiza en la misma infraestructura por lo que es muy improbable que personal no autorizado acceda a la información.
- **Disponibilidad** El sistema no es necesario que esté disponible constantemente, solo es necesario que lo esté cuando se ejecute la aplicación y se proceda a las medidas o a la visualización en diferido.

CAPÍTULO

3

ANÁLISIS DEL SISTEMA

Una vez establecidos los diferentes requisitos, detallados en el capítulo 2, es momento de analizar el sistema más detalladamente, estableciendo las herramientas y proceso de desarrollo para la correcta realización de éste, así como un pequeño estudio y comparativas con lo que ofrece el mercado.

A lo largo de este capítulo se tendrán en cuenta las limitaciones existentes del NodeMCU para el correcto desarrollo del firmware. Todas las elecciones que conforman el proyecto estarán justificadas en la sección 3.4, junto con un estudio si corresponde.

3.1 Comparativas y estudio de mercado

Según la investigación sobre el mercado que se ha llevado a cabo a través de internet, acerca de la existencia de un exoesqueleto en forma de bota, se ha localizado que la primera vez que se habla de un ExoBOOT y de la que hay un estudio realizado que se remonta a 2015. El articulo en cuestión es *"Reducing the energy cost of human walking using an unpowered exoskeleton"* [7]. En este estudio se desarrolla una versión completamente mecánica, partiendo de una base de fibra de carbono, siendo diseñada a medida para cada participante.

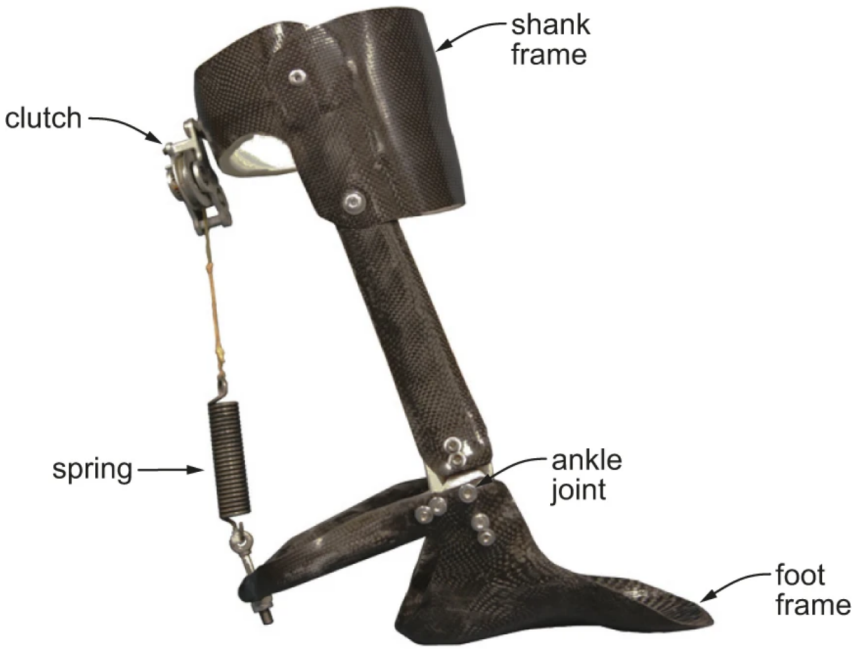


Figura 3.1 – *Diseño del frame del exoesqueleto [7].*

El objetivo de este dispositivo es reducir la tasa metabólica (TMB) del usuario descargando la fuerza muscular que se produce en el gesto de caminar. Todo esto usando un dispositivo que no necesita ningún tipo de energía para realizar su función de reducir la TMB.

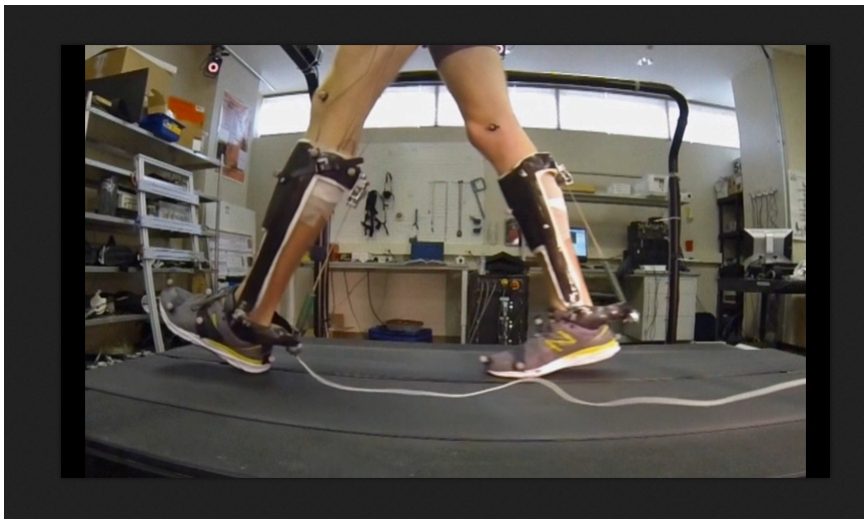


Figura 3.2 – *Frame del video en movimiento [7].*

Otro estudio [6] que se llevo a cabo para una conferencia en Holanda en 2018, partía de

la misma base que el anterior, es decir, reducir en lo posible la tasa metabólica en el gesto de caminar. En este caso el prototipo cuenta con un sistema hinchable, que ayuda al tobillo cuando se flexiona.

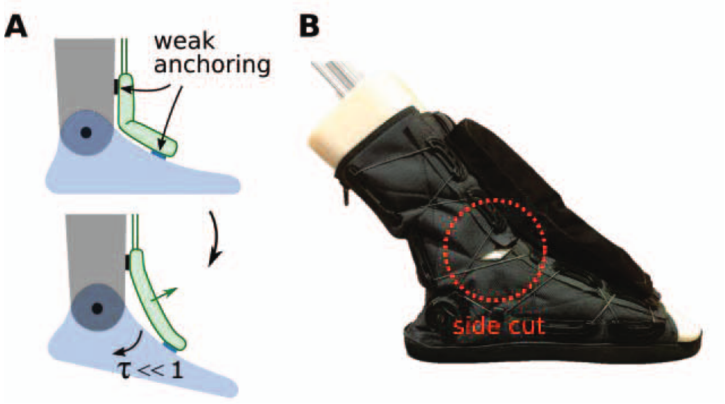


Figura 3.3 – *Esquema funcionamiento del prototipo [6].*

La obtención de datos en este caso, se llevaron a cabo a través de un microcontrolador que incluía una placa Arduino, la cual ayuda a gestionar el fujo de aire dependiendo de las lecturas de los diferentes sensores.

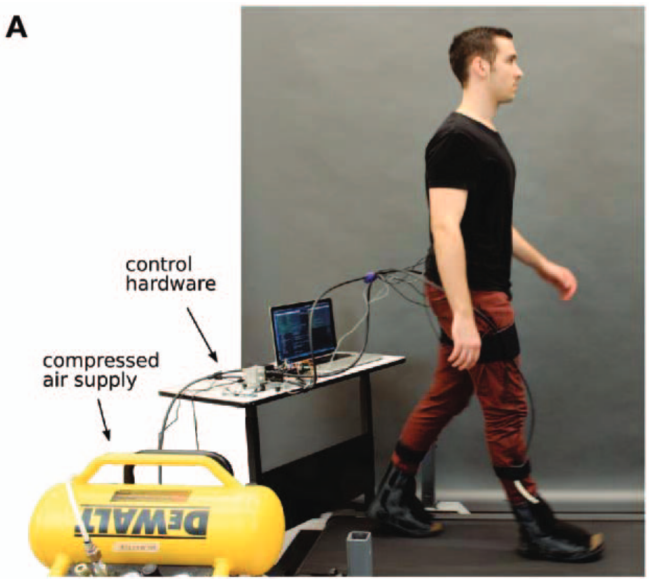


Figura 3.4 – *Sistema completo para el funcionamiento [6].*

Las opciones anteriormente citadas, son proyectos desarrollados para su utilización sin peso extra. El proyecto planteado por GranaSAT, en cambio, se ha desarrollado con la idea de poder reducir el coste metabólico y el cansancio en las articulaciones inferiores cuando el

individuo transporta una elevada carga.

Una de las importantes diferencias con las otras versiones citadas, es que resulta más económico a la hora de reproducir el estudio, ya que se usa la tecnologia de impresión 3D, dicho diseño se puede ver en el capítulo 4.

Otra diferencia visible se puede ver en la forma de llevar a cabo los estuidos, así como sus respectivas medidas. Se utlizan equipos conectados directamente al ordenador, como se puede ver en las figuras 3.2 y en la 3.4. En ambos casos estos sistemas de obtención de datos resultan incómodos para los pacientes, ya que limita los movimientos. Es por ello que en el sistema desarrollado por GranaSAT se buscaba lo máximo posible que esto se realizase de forma inalámbrica y en tiempo real para el correcto estudio. Así se facilita la obtención de los datos, dando más libertad al sujeto que porta las botas. De esta forma se pueden realizar en un terreno más realista y no sobre una cinta de correr.

3

3.2 NodeMCU: ESP32 y arquitectura

Para la recogida y envío de datos se utiliza el ESP32[1], sucesor del ESP8266. Es un único chip que combina tecnología Wi-Fi y Bluetooth de 2.4 GHz, desarrollado por Esspresif Systems y diseñado con tecnología TSMC de ultra baja potencia de 40 nanómetros de tecnología. Está diseñado para logar la mejor potencia y rendimiento de radiofrecuencia , mostrando robustez, versatilidad y fiabilidad en una amplia variedad de aplicaciones y escenarios de potencia.

De este NodeMCU existen diferentes versiones, iguales entre ellas y con la única salvedad de que tienen diferente footprint. En nuestra EXOBOOT se usa el ESP32 WROVER.

Figura 3.5 – *Controlador ExoBOOT.*

[1] Antes de nada se debe poder programar el ESP32. Este primer inicio se encuentra documentado en el anexo A

Algunas de sus características, entre otras, es que cuenta con 512 KB de SRAM o un procesador de 40 nm que puede operar hasta los 240 Mhz.

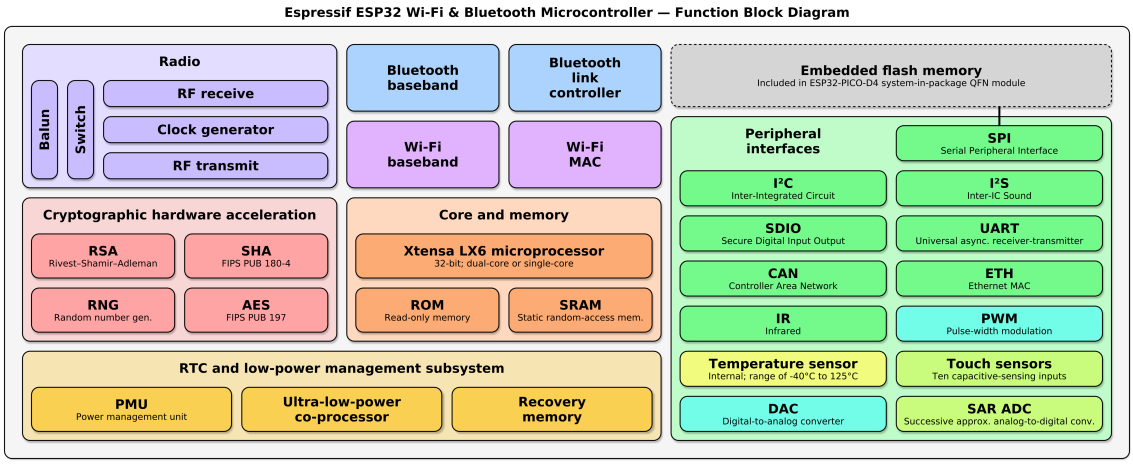


Figura 3.6 – *Esquema de funcionamiento del ESP32 [24].*

El firmware de ExoBOOT, encargado de gestionar el chip, se estudiará para encontrar la forma más eficiente de realizar la lectura y envío de los datos. Como se puede comprobar en la justificación 3.4.1, se desarrollará bajo sus dos núcleos, para así aprovechar el máximo potencial que puede aportar.

3.3 Data Budget

En las diferentes pruebas realizadas, actualmente, se hace un promedio de 710 paquetes por minuto, ya que debe leer información de sensores a través del GPIO. Debido a esto, se tarda 85 milisegundos en realizar una lectura.

```
20:30:23.751 -> EMPIEZA MEDIDA--
20:31:23.804 ->  ---------- FIN MEDIDA ------
20:31:23.804 ->  Paquetes en 1 minuto: 713 paq/min
20:31:23.839 -> Total tiempo envío: 60066.00 ms
20:31:23.839 -> Media de tiempo de lectura por los paquetes enviados (total_tiempo / paq): 84.24 ms
20:31:23.839 -> ********************************
20:31:23.839 -> Paquetes PERDIDOS: 0
```

Figura 3.7 – *Máximo número de paquetes enviados en 1 minuto.*

Como se puede apreciar en la captura anterior 3.7 se envían un total de 713 paquetes en 1 minuto, siendo el tiempo promedio para leer cada dato de 84.24 milisegundos. Estos datos corresponden a la lectura y envío de los sensores de fuerza y el potenciómetro, dado que en el desarrollo hardware se produjo un problema y no se puede obtener información de los otros sensores, en concreto de los que se leen por la IMU. En el caso de estar todos los

sensores conectados, puede alterar el tiempo de lectura en cada paquete.

3.4 Justificación de herramientas

Muchas de las decisiones, dada la libertad a la hora de desarrollar, han sido tomadas en torno al aspecto teórico y práctico. Dichas decisiones son las relacionadas con:

- La forma de envío de los datos. Ya sea usando REST API o sockets.
- En lo relacionado con la base de datos, usar SQL o NoSQL
- El hardware a utilizar entre el ESP32 y el ESP8266.

Para cada decisión, se ha realizado un profundo estudio y han sido tomadas de forma conjunta con el cliente, GranaSAT. Para comprender mejor dicha comparación de lo que ofrece el mercado, se exponen a continuación, en cada sección del capítulo, los diferentes elementos que conforman el proyecto y de los que ha sido necesario elegir una opción.

3.4.1 Hardware - NODEMCU

Teniendo en cuenta los diferentes microprocesadores que se pueden usar para este proyecto, se han elegido los NodeMCU por su tamaño y por las prestaciones que ofrecen. Partiendo de esta base se pasó a hacer una comparativa entre los NodeMCU conocidos como ESP32 y su predecesor el ESP8266. Finalmente se optó por el sucesor, dado que es una actualización y mantiene el reducido coste en relación a la potencia que ofrece.

En la siguiente tabla se muestran algunas características que los diferencian:

Característica	**ESP8266**	**ESP32**
Procesador	Tensilica LX106	Tensilica Xtensa X36
Nº bits	32 bits	
Nº nucleos	Single core	Single core o Dual core
Velocidad	80Mhz (hasta 160 Mhz)	160 MHz (hasta 240 MHz)
SRAM	160 kB	512 kB
Alimentación	3.0 a 3.6V	2.2 a 3.6V
GPIO (utilizables)	17	36
Elección	✗	✓

Tabla 3.1 – *Tabla comparativas NodeMCU [10].*

Como se puede apreciar en la anterior tabla, el ESP32 mejora en gran medida al ESP8266, como mayor velocidad de procesado, en concreo hasta 2 veces más, con $3,2$ veces más de SRAM o el poder hacer uso del dual core, el cuál mientras lee un dato puede estar enviando

orto que está almacenado en el buffer. Una menor alimentación, la cual permite funcionar el dispositivo con un bajo coste y así alargar el uso con batería. Otra mejora importante es el disponer de hasta 19 GPIOs utilizables más, lo cúal en un futuro permite el incorporar más sensores, si así se desea.

Para determinar la elección correcta, se realizó un test a nivel computacional, el cual mide la velocidad de lectura en los pines GPIO, así como un cálculo que consiste en realizar la potencia a 2 de cada elemento entre el 0 y el 100.000. Estos test se ejecutan en la sección loop, por lo que se realiza infinitamente, arrojando los datos que se muestran a continuación en la figura 3.8.

```
20:04:59.818 -> loop 1000000 gpio: 1912 ms
20:05:01.729 -> double 100000 pow: 9999800001.00 257 ms
20:05:02.008 -> ESP8266 ---- total 2.17 s
20:05:02.008 ->
20:05:02.982 -> loop 1000000 gpio: 1913 ms
20:05:04.898 -> double 100000 pow: 9999800001.00 257 ms
20:05:05.177 -> ESP8266 ---- total 2.17 s
20:05:05.177 ->
20:05:06.185 -> loop 1000000 gpio: 1912 ms
20:05:08.070 -> double 100000 pow: 9999800001.00 257 ms
20:05:08.350 -> ESP8266 ---- total 2.17 s
20:05:08.350 ->
20:05:09.356 -> loop 1000000 gpio: 1912 ms
20:05:11.265 -> double 100000 pow: 9999800001.00 257 ms
20:05:11.510 -> ESP8266 ---- total 2.17 s
20:05:11.510 ->
20:05:12.520 -> loop 1000000 gpio: 1913 ms
20:05:14.430 -> double 100000 pow: 9999800001.00 257 ms
20:05:14.675 -> ESP8266 ---- total 2.17 s
20:05:14.675 ->
20:05:15.683 -> loop 1000000 gpio: 1913 ms
20:05:17.598 -> double 100000 pow: 9999800001.00 257 ms
20:05:17.841 -> ESP8266 ---- total 2.17 s
```

(a) *ESP8266*

```
20:30:25.306 -> loop 1000000 gpio: 252 ms
20:30:25.548 -> double 100000 pow: 9999800001.00 59 ms
20:30:25.618 -> ESP32 ---- total 0.31 s
20:30:25.618 ->
20:30:26.614 -> loop 1000000 gpio: 252 ms
20:30:26.856 -> double 100000 pow: 9999800001.00 59 ms
20:30:26.926 -> ESP32 ---- total 0.31 s
20:30:26.926 ->
20:30:27.930 -> loop 1000000 gpio: 252 ms
20:30:28.175 -> double 100000 pow: 9999800001.00 59 ms
20:30:28.244 -> ESP32 ---- total 0.31 s
20:30:28.244 ->
20:30:29.254 -> loop 1000000 gpio: 252 ms
20:30:29.496 -> double 100000 pow: 9999800001.00 59 ms
20:30:29.566 -> ESP32 ---- total 0.31 s
20:30:29.566 ->
20:30:30.543 -> loop 1000000 gpio: 252 ms
20:30:30.788 -> double 100000 pow: 9999800001.00 59 ms
20:30:30.858 -> ESP32 ---- total 0.31 s
20:30:30.858 ->
20:30:31.863 -> loop 1000000 gpio: 252 ms
20:30:32.104 -> double 100000 pow: 9999800001.00 59 ms
20:30:32.174 -> ESP32 ---- total 0.31 s
```

(b) *ESP32*

Figura 3.8 – *Resultado ejecución test.*

Como se observa en la anterior figura, los tiempos que tarda en leer el ESP32 de sus respectivos GPIO es 7,6 veces más rápido que en el ESP8266 y 4,4 más rápido en realizar el cálculo de la potencia. En rangos generales el ESP32 completa estás tareas 7 veces más rápido que en el ESP8266, por lo que debido al proyecto que se va a desarrollar en el cúal es necesario la lectura de GPIOs y realizar cálculos, el chip que mejor se adapta es el ESP32, siendo este el elegido.

Puesto que en el NodeMCU elegido se puede usar hasta dos cores, se procede a realizar un test parecido al anterior para determinar si hay mejora de velocidad al usar un core o dos. Los resultados de la ejecución del test se pueden comprobar en la siguiente captura.

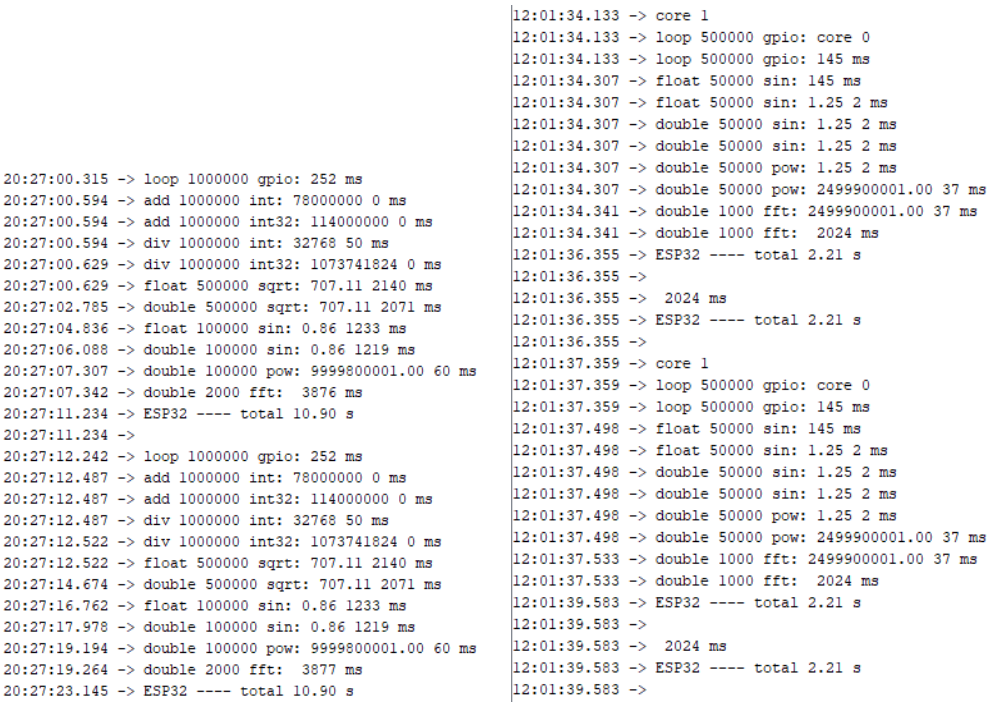


Figura 3.9 – *Resultado ejecución test en los núcleos.*

Como se puede comprobar los tiempos totales de completar las tareas son parecidos, con la diferentcia de que en doble núcluo la tareas realizan en paralelo, cada núcleo hace una mitad, por lo que el tiempo total de la ejecución se ve altamnete reducidos, más concretamente en 8 segundos.

Por esto la eleccion del desarrollo del firmware se basará en dual core, dado que de esta forma es posible repartir las tareas y así a la vez que se leen datos, en esos 85 ms que tarda en leer los sensores del GPIO, el otro core puede enviar dichos datos al servidor para su representación y almacenamiento.

3.4.2 Software - Métodos de envío-lectura y programación

La aplicación web, o para ordenador, ha sido desarrollada en NodeJS [18], lenguaje basado en JavaScript. La ejecución es multiplataforma, pudiendo ser usada tanto en Linux, en Windows o en MacOs. Otro aspecto de NodeJS es que se usa en el lado del servidor y su fucnionamiento se basa en la orientación a eventos, permitiendo desarrollar aplicaciones escalables.

En las siguientes subsecciones se comparará aspectos necesarios de la aplicación, como la forma de envío o la base de datos. Estas comparaciones y elecciones están tomadas entorno al apartado teórico.

3.4.2.1 Socket.IO, POST o GET

Para el envío de datos entre el servidor y el cliente existen diversas posibilidades para usar y realizar el envío de datos. Entre esas, están los métodos REST [15] de HTTP, en concreto el método POST y GET, pero este último se usa para listar, leer datos y obtener datos con ayuda de la API. Otra opción es usar el protocolo [13] websocket, opción que finalmente se ha elegido.

En este caso la decisión es más clara, ya que que usar el método POST implica hacer continuas llamadas a una API para enviar los datos. Esto puede ocasionar sobrecarga, debido a que se realizarían 710 peticiones POST en 1 minuto. En este caso ya no se podría desempeñar la tarea en tiempo real, condición indispensable del cliente. Usando POST primero se debe almacenar el dato y luego consultarlo, por lo que ralentizaría la aplicación en tiempo real.

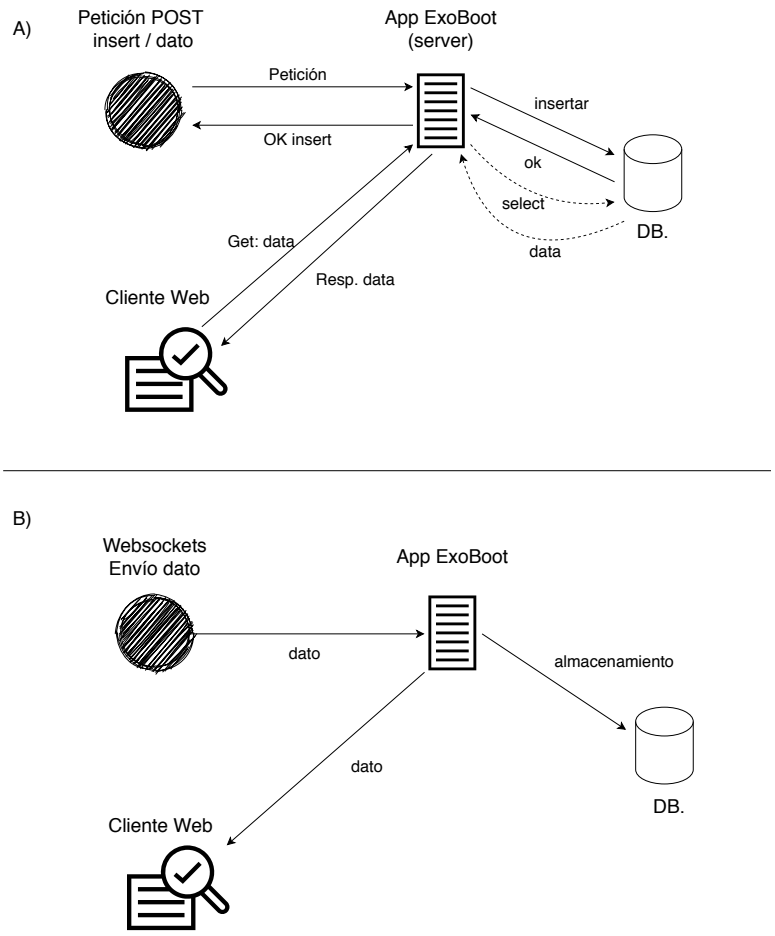


Figura 3.10 – *A) Petición POST. B) Websockets.*

El funcionamiento de usar POST se apreciar en la simulación del esquema de la 3.10 para

realizar el envío de datos de ExoBOOT al servidor se deben hacer peticiones al servidor para insertar el dato, insertarlo en la base de datos y posteriormente esperar la petición GET o la consulta de la base de datos para mostrar los datos leídos por el NodeMCU incorporado en la bota. En cambio usando websocket, como se ve en la mitad inferior del esquema, no es necesario realizar peticiones, ya que cuando el servidor y ExoBOOT están conectados, los datos se reenvían al cliente web y a la base de datos. De esta forma se consigue que la aplicación se muestre en tiempo real, tal y como solicitaba el cliente.

3.4.2.2 SQL o NoSQL

En lo referente al almacenamiento de los datos existe la tecnologías de SQL y NoSQL, siendo sus principales diferencias las se muesstran en la siguiente tabla:

	Base de datos SQL	**Base de datos NoSQL**
Almacenamiento	Estructurado: Tablas con filas y columnas	No estructurado con varios modos: Documentos, Wide-column y gráficos
Escalabilidad	Vertical	Horizontal
Estructura	Garantiza ACID, Centralizada	Distrubuida
USOS	Tener estructura y garantizar la atomiciada	No tener una estructura, como en la recogida masiva de datos
Elección	✗	✓

Tabla 3.2 – *Comparativa SQL y NoSQL [14].*

Como se puede comprobar en la tabla anterior, cada tecnología ofrece sus ventajas y desventajas en torno al proyecto que se quiera desarrollar, por lo que es necesario en este punto elegir bien y realizar un profundo estudio.

En este caso, puesto que no es necesario que los datos esten estructurados, ni sean modificados una vez almacenados en la base de datos, se procederá al desarrollo del sistema usando NoSQL. Dado que se pueden usar documentos JSON como base para recopilar la información, facilita la manipulación de los datos y almacenamiento de estos. Esto es debido a que los datos llegan, por parte del firmware, en formato JSON y con una simple instrucción como `db1.insert(dato_recivido)` permite la insercción de los datos más rápidamente en el documento.

Dentro de la tecnología NoSQL existen diversas herramientas para el manejo de dichos datos. Para este caso una buena elección es usar base de datos embebidas, es decir, una base de datos que ocupe poco y almacene sus datos en un archivo de texto sin formato. Una de esas librerías es neDB [5], basada en MongoDB, que para listar o filtar los datos usa simples instrucciones como `find` o la anterior citada, `insert`.

3.4.3 Firmware

En el lado del firmware es una de las partes más importantes en las que se deben tomar decisiones, desde la estructura de los datos hasta las variables necesarias, debido a que disponemos de una memoria muy limitada.

Una de dichas decisiones fue la del método de envío, comentada en la sección 3.4.2.1 de este capítulo. Otras opciones que surjieron fue la de usar *array* o JSON para la lectura y envío de los datos.

El *array* en cuestión debería tener una longitud de 14 y ser un `long int`, porque el valor de la fuerza es el máximo que se puede representar con este tipo de dato, el cual necesita 24 bits para ser representado y un long int tiene 32 bits.

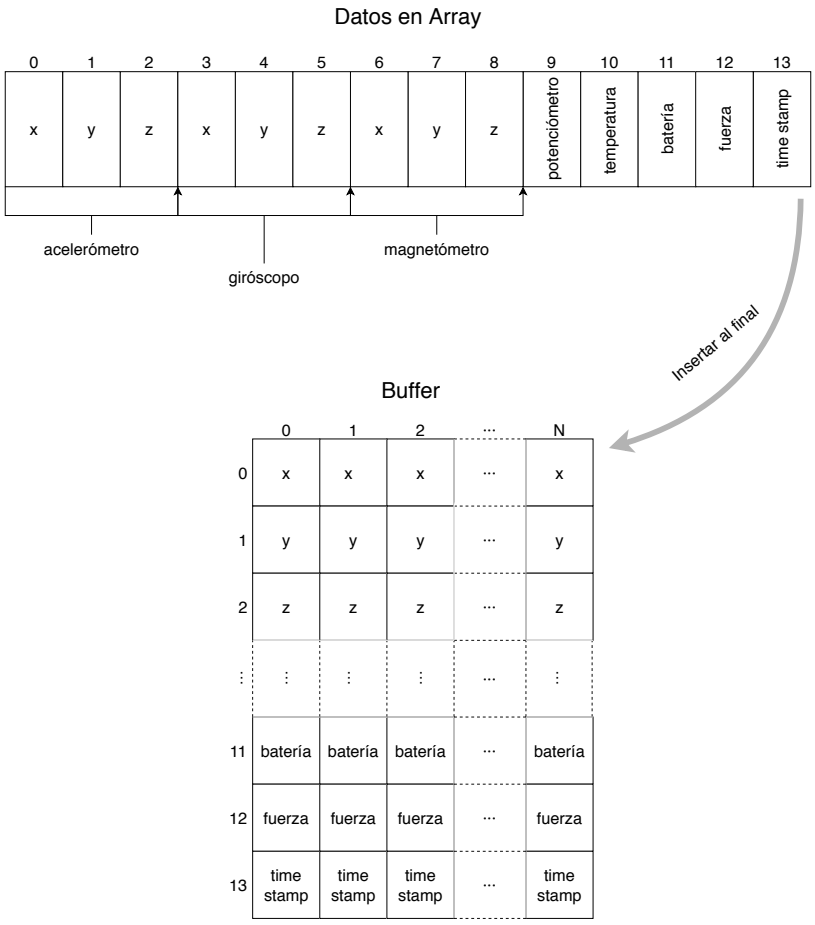


Figura 3.11 – *Array y buffer.*

Dicho almacenamietno debería seguir la estructura visual de como se muestra en la imagen anterior 3.11. De ser así serían necesarios $14 * 32\ bits = 448\ bits$, que en *Bytes* serían

448 $bits \frac{1\ Bytes}{8\ bits} = 56\ Bytes$ para guardad los datos de una única medida. Si de media se recogen unos 710 $\frac{paq}{min}$, es decir aproximadamente 12 $\frac{paq}{s}$, se necesitaría una memoria total de $56\ Bytes * 12\ \frac{paq}{s} = 672\ \frac{Bytes*paq}{s}$. Dadas las característica del ESP32, esto es un desperdicio de memoria, así como que hace que el sistema funcione más lento por los constantes accesos a la memoria para consultar las diferentes posiciones del *array*.

Por el contrario usar JSON, reduce esa cantidad a 32 *Bytes* cuando se lee y almacena el dato, aunque este dato depende del valor leído. Para el correcto funcionamiento y procesamiento con el servidor, se debe pasar a *string* para que quede registrado correctamente y sea más sencillo enviarlo.

El valor máximo de dicho *string* es de 166 *Bytes*, algo que es difícil que se cumpla porque eso indicaría que todos los sensores han leído el mayor dato posible. Haciendo cálculos lo que en un segundo se necesita $166 * 12 = 1991\ Bytes$ más del doble del array, pero por el contrario no siempre será este tamaño y no es necesario estar haciendo accesos todo el rato a posiciones de memoria para almacenar y consultar el dato.

Un dato que envía desde el sistema de la bota tiene esta estructura:

```
{
  "a":{
    "x":1,"y":2,"z":3
  },
  "g":{
    "x":1,"y":2,"z":3
    },
  "m":{
    "x":1,"y":2,"z":3
    },
  "p":40,
  "t":1,
  "b":2,
  "f":10,
  "tm":61696
}
```

Código 3.1 – *JSON que envía el ESP32.*

Donde se redujo el nombre identificativo, ya que eso ocasionaba que la cadena de *string* fuese más larga y por lo consiguiente que ocupase más en memoria. Cada elemento se correscponde con los sensores que miden, como lo son el acelerómetro (a), giróscopo (g), potenciómentro (p), magnetómetro (m), la temperatura (t), la bateria (b), la fuerza (f) y el instante de tiempo (tm) que se ha tomado la medida.

Una vez se ha transformado a tipo *string*, pasa por un buffer antes de ser enviado. Este buffer también actua en el caso en el que se desconecte del servidor o se pierda la conexión, así dicho dato no se pierde. Por parte del servidor, ese dato es insertado en la base de datos al mismo tiempo que es enviado al cliente web para mostrarlo a través de las correspondientes gráficas.

A la estructura comentada anteriormente, antes de ser insertado en el documento, se le añade la eqitueta `"izquierda"` o `"derecha"` dependiendo de que bota provenga el dato, así como el identificador único para ese dato, que lo inserta aútomaticamente la herramienta que gestiona los datos, en este caso neDB. A continuación se puede ver una muestra de como esos datos se almacenan:

```
1 {"a":{"x":0,"y":1,"z":2},"g":{"x":0,"y":1,"z":2},"m":{"x":0,"y":1,"z":2},"p":23,"t":0,"b":0,"
  f":45,"tm":15546,"bota":"izquierda","_id":"03S5laZwzt7iJR0H"}
```

Código 3.2 – *JSON almacenado en la base de datos.*

3.5 Gantt

La planificación de este proyecto se puede contemplar en la gráfica de Gantt dónde el trabajo se ha dividido en varias tareas.

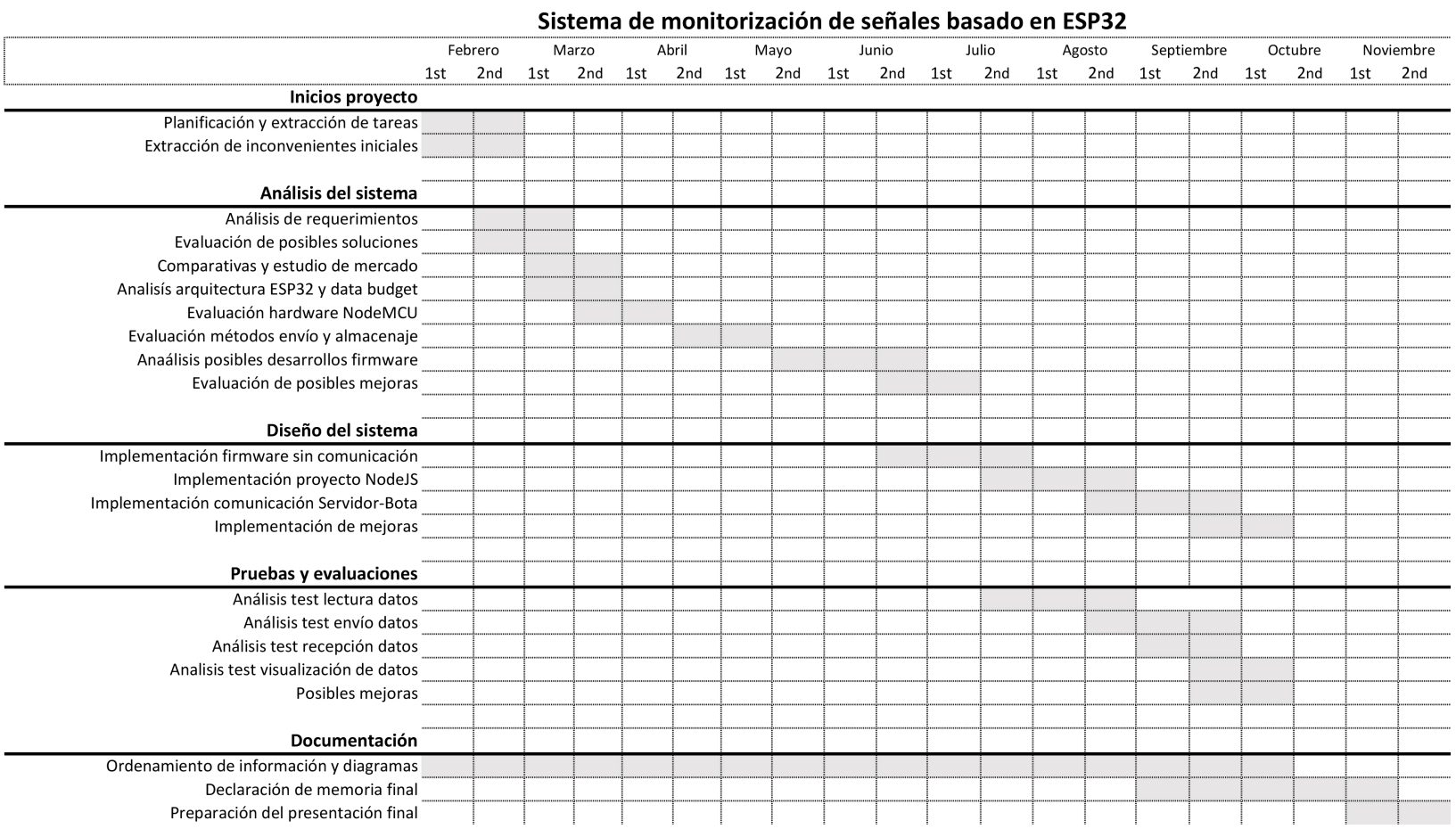


Figura 3.12 – *Diagrama de Gantt.*

3.6 Presupuesto

3.6.1 Hardware

La parte Hardware de ExoBOOT está formada por diferentes elementos, desde el chip ESP32 hasta la estructura impresa en 3D, así como los diferentes sensores. Todo este coste de una sola bota está reflejado en la tabla siguiente, por lo que el total del hardware asciende a 530 €.

Hardware	Coste (€)
Botas	Cedidas (0) €
ESP32	5,00 €
HX711 + Galga	250,00 €
IMC 20948	5,00 €
Potenciómetro	2,00 €
Diseño soportes 3D	BiblioMaker (0) €
Diseño Placa	(3,00 €)
Subtotal	**265,00 €**

Tabla 3.3 – *Presupuesto Hardware.*

3.6.2 Software

Todo lo que concierne a este trabajo es relacionado con está parte, donde gran parte de desarrollo se ha centrado tanto en el servidor, como en el firmware controlador de ExoBOOT. Los diferentes programas y recursos usados son en su mayoría de uso libre, pudiendo desarrollar sin coste alguno. De igual modo, dichas herramientas y frameworks aparecen detallados en la tabla.

Software	Coste (€)
NodeJS	0 €
Arduino IDE	0 €
Miktex	0 €
SumatraPDF	0 €
Visual Estudio Code	0 €
Inkscape	0 €
Microsoft Excel 2019	Prueba (0 €)
Subtotal	**0 €**

Tabla 3.4 – *Presupuesto Software.*

3.6.3 Humano

En el apartado de los recursos humanos, este proyecto a necesitado la mano de 3 personas, dos ingenieros junior, (10 €/h), uno para parte hardware y otro para software y un ingeniero senior, (50 €/h). El gasto humano total ha sido de 12.000 €.

Posicion	Horas	Coste (€)
Juniors	800	8.000,00 €
Senior	80	4.000,00 €
	Subtotal	**12.000 €**

Tabla 3.5 – *Presupuesto Humano.*

3.6.4 Total

Finalmente el importe total del proyecto, junto con un I.V.A. del 21% y los gastos indirectos del laboratorio, asciende a 15.203,65 €

Gasto total		Coste (€)
Hardware		265,00 €
Software		0 €
Humano		12.000,00 €
Gastos indirectos		300,00 €
	Total	*12.565,00 €*
	I.V.A. (21%)	*2.638,65 €*
	Total	**15.203,65 €**

Tabla 3.6 – *Presupuesto total.*

CAPÍTULO

4

DISEÑO DEL SISTEMA

Una vez sentadas las bases de los requisitos necesarios y de las diferentes herramientas y frameworks usados, es el momento de explicar más detalladamente todo el proyecto partiendo desde el hardware de ExoBOOT, información proporcionada por GranaSAT, y continuando por el objetivo del proyecto, es decir, la parte software y el firmware correspondiente al hardware. Aquí se explicarán todos los aspectos importantes relacionados con el código, tanto los métodos úsados como los implementados.

Para la elaboración del proyecto se ha seguido el diagrama 4.1 donde está la parte hardware que se conecta al servidor a través de websockets, el servidor gestiona todos los eventos procedentes de ExoBOOT, así como permitir las conexiones y la base de datos. Por último está la parte visual del software, la cual se conecta mediante websockets para permitir que la aplicación funcione en tiempo real. Todo esto se encuentra más desarrollado en las secciones siguientes.

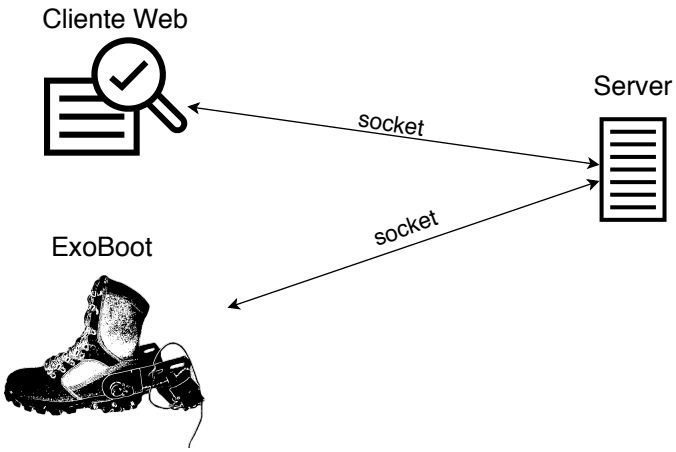


Figura 4.1 – *Sistema ExoBOOT.*

4.1 Hardware

El hardware en sí está diseñado para medir la fuerza que está sufriendo la banda elástica de la bota, conocer el ángulo de flexión en el que se genera y con un sensor IMU[1] que incluye un giróscopo 3D, acelerómetro 3D, magnetómetro 3D y sensor de temperatura, para medir la orientación, aceleración e intensidad magnética respectivamente.

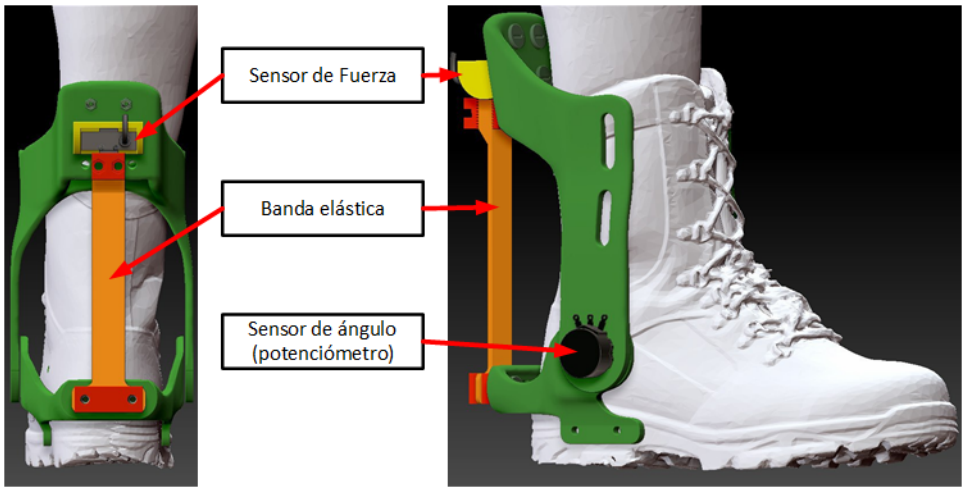


Figura 4.2 – *Hardware componente de ExoBOOT.*

Lo que se observa en la figura 4.2 son las distintas partes que forman el dispositivo hardware. Donde el sensor de fuerza soporta la banda elástica, el cual medirá la fuerza (N) que se ejerce en dicha banda. Con ayuda del potenciómetro se puede obtener las medidas

[1]IMU: este por un error en el proceso de producción no es posible obtener las medidas correspondientes

correspondientes al ángulo en que se produce esa fuerza.

El sistema en conjunto se encarga de monitorizar la fuerza que está soportando la banda elástica para saber la ayuda que está recibiendo el sujeto que lleva puesto ExoBOOT. Este sistema consta del microcontrolador ESP32 con capacidad de conectividad WiFi y Bluetooth, entre otras características descritas en el capítulo 3, sección 3.4.1. Esta conectividad que aporta el NodeMCU ayuda a la transmisión de los datos de forma inalámbrica, para así no necesitar conexiones por cables que entorpezcan las mediciones.

4.2 Estados

El firmware de ExoBOOT se ha programado siguiendo los siguientes estados, los cuales debe cumplir para el correcto funcionamiento. Este diagrama se puede ver a continuación.

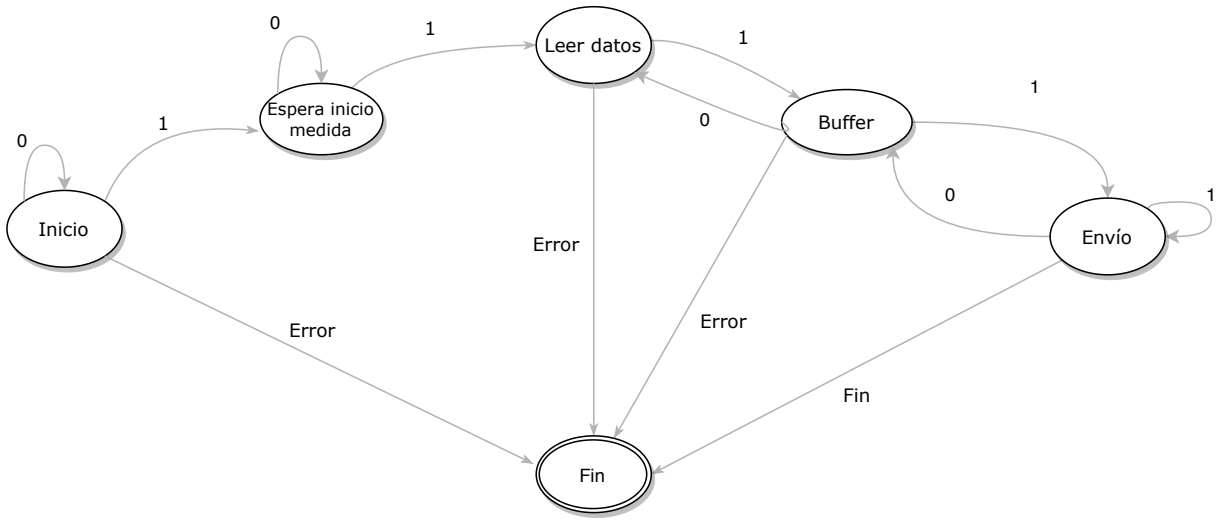


Figura 4.3 – *Estados del firmware de ExoBOOT.*

- **Inicio**: El estado inicial, es el comienzo de la ejecución, en la que se intenta establecer la conexión con la red wifi y el servidor websockets. Esto se intenta un número determinado de veces, correspondiente al estado 0. Si finalmente no se consigue conectar se produce un error y termina, pero por el contrario si la conexión se ha realizado con éxito (1) pasa al siguiente estado.
- **Esperar inicio medida**: Este estado es en el que espera que desde el servidor llegue la orden de empezar la medida y el tiempo que estará en ejecución. Cuando esa información es recibida (1), pasa al siguiente estado.
- **Leer datos**: Se realiza la lectura de los diferentes sensores y se envía al buffer. Este

estado se termina si la cola se encuentra llena y se pierden una gran cantidad de datos.

- **Buffer**: Se pueden dar 3 casos, uno de los casos es cuando se ha desconectado del servidor, por lo que vuelve al estado anterior para leer más datos y seguir almacenados (0). El segundo caso (1) se da cuando la conexión con el servidor websockets o el WiFi sigue siendo correcta. El tercer y último estado (error) se da cuando se han producido demasiados intentos de conexión y no se consigue restablecer, por lo que se produce el fin de la ejecución.
- **Fin**: Estado final, en el que se finaliza la ejecución.

4.3 Software

El software es la parte del proyecto que se manejará los eventos necesarios para el correcto funcionamiento y desarrollo de ExoBOOT. En las siguientes secciones se documentará lo que ha llevado al correcto funcionamiento y las herramientas necesarias.

4.3.1 Proyecto NodeJS

La aplicación de escritorio se ha desarrollado para que sea un ejecutable y se muestre la información en forma de webapp pudiéndose ver a través del navegador. Se ha seguido una estructura de NodeJS+ ExpressJS [19] y como gestor de plantillas HTML con PUG [22]. Sigue una filosofía de MVC (modelo vista controlador) [25]. Este paradigma se basa en:

- **Model** - Es quién se encarga de la lógica de negocio. Con todo lo que tiene que ver con la gestión de los datos, así como de las transiciones. Esto es el *backend* de la aplicación.
- **View** - Presenta los datos de forma visual, representa sobre todo al *frontend*.
- **Controller** - El controlador se encarga de la gestión de los datos mediante eventos y puede estar representado tanto al *backend*, como al *frontend*.

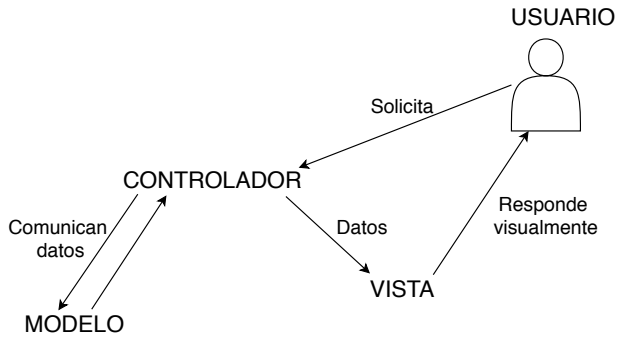


Figura 4.4 – *Funcionamiento modelo vista controlador.*

4.3.2 Estructura

La base del proyecto se puede crear rápidamente y a través de `express-generator` [20], proporcionado por ExpressJS [19], previa instalación con ayuda del gestor de paquetes de NodeJS llamado NPM [17]:

```
$> npm install express-generator -g
```

Tras la instalación, se puede usar para generar el proyecto. Consta de la opción `-view` para elegir el motor de vistas y el nombre de la aplicación. En concreto para este proyecto se ha utilizado:

```
$> express -view=pug Exoboot_app
```

Obteniendo a la salida de dicha ejecución una estructuración de proyecto, con ficheros básicos del proyecto, como los *package.json*, y unos comandos para empezar a ejecutar la aplicación:

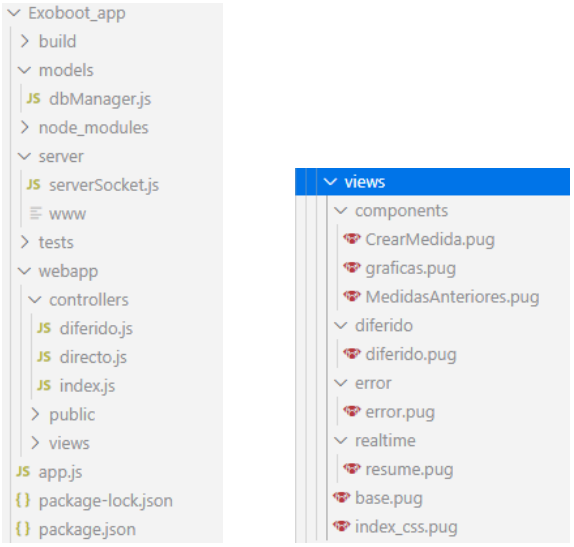
```
create : Exoboot_app/
create : Exoboot_app/public/
create : Exoboot_app/public/javascripts/
create : Exoboot_app/public/images/
create : Exoboot_app/public/stylesheets/
create : Exoboot_app/public/stylesheets/style.css
create : Exoboot_app/routes/
create : Exoboot_app/routes/index.js
create : Exoboot_app/routes/users.js
create : Exoboot_app/views/
create : Exoboot_app/views/error.pug
create : Exoboot_app/views/index.pug
create : Exoboot_app/views/layout.pug
create : Exoboot_app/app.js
create : Exoboot_app/package.json
create : Exoboot_app/bin/
create : Exoboot_app/bin/www

change directory:
  $ cd Exoboot_app

install dependencies:
  $ npm install

run the app:
  $ DEBUG=exoboot-app:* npm start
```

Siguiendo unas buenas prácticas de NodeJS [12] y ExpressJS [19][3], es bueno modificar dicho directorio y separar el contenido para poder realizar un mantenimiento adecuado y actualizar los componentes de la aplicación. Tras esta modificación la estructura de este proyecto finalmente es:



(a) *Estructura app* **(b)** *Contenido views*

Figura 4.5 – *Estructura y archivos aplicación escritorio*

Donde se ha separado mejor la configuración del servidor, la aplicación y la webapp. Otros directorios como node_modules son generados automáticamente cuando se realiza `npm install` y se instalan todas las dependencias necesarias.

Estos son los 3 principales directorios que forman la app:

- **server** –> Para toda la configuración del servidor, donde se crea y configura. En esta ubicación también está lo relacionado con el servidor del websocket, donde tanto las botas, como el cliente web se conectarán para el intercambio de datos.
- **models** –> Este directorio se corresponde con la configuración necesaria, como la conexión con la base de datos y su gestión. En este caso está declarada en la base del proyecto puesto que tanto le servidor como la web consultan esto. Como los datos están almacenados en ficheros, cuando se van a consultar es cuando se pasa a la base de datos.
- **webapp** –> Todo lo relacionado con la aplicación web, como sus controladores, las vistas y los ficheros públicos.

Los demás ficheros o carpetas son los relacionados con los paquetes de NodeJS o la configuración de la aplicación.

- **app.js** - Este archivo se genera automáticamente y es donde va la configuración de los controladores para la gestión de las rutas correctas para la web.
- **package.json y package-lock.json** - Configuración del proyecto, como las dependencias, nombre, autor o versión.
- **node_modules** - Directorio con los módulos de NodeJS necesarios para el proyecto.

4.3.3 Backend

El *backend* es donde se desarrolla todo para que la aplicación funcione de forma correcta. Dentro de esta parte se incluye la creación del servidor, el servidor websocket, el modelo de datos y los controladores del *frontend*.

4.3.3.1 Servidor principal

La configuración de dicho servidor está en `/server/www` y es uno de los ficheros que se generan solos al crear el proyecto con el comando de ExpressJS [19].

En primer lugar se crea el servidor asociando las diferentes rutas y elementos declarados en `app.js`, así como el puerto donde escuchará y la gestión de los eventos necesarios.

```
var server = http.createServer(app);
...
server.listen(port);
server.on('error', onError);
server.on('listening', onListening);
```

Código 4.1 – *Creación servidor.*

En dicha configuración del servidor también se declará el servidor de socket,

```
var serverSocket = require('./serverSocket')
var io = new serverSocket(server);
io.events('crearsala', 'botaIZQ', 'botaDCH', 'medicion');
```

Código 4.2 – *Creación servidor websocket.*

usando la clase llamada `serverSocket`, que ha sido creada para identificar mejor dicho servidor. Esto se abordará en la siguiente subsección.

4.3.3.2 Servidor socket

Los websocket se han creado con ayuda de la librería de Socket.IO [23], que se encuentra bien integrada con NodeJS. Para la creación del servidor de websocket se ha creado la clase `serverSocket`. Con esta clase se crea el objeto partiendo del servidor que se pasa como parámetro. También se modifica el pingTimeout a 25 segundos. Esto se hace para que sea ese tiempo máximo que esta conectado al websocket sin interactuar.

```
1   constructor(server){
2     this.io = new socketio(server,{
3       pingInterval: 1000,
4       pingTimeout: 25000
5   });
6     }
```

Código 4.3 – *Constructor* `serverSocket.js`.

Una vez ha sido creado y se aceptan conexiones a través del evento `connection`, ya se puede trabajar con los diferentes clientes y comunicarse para el intercambio de información. Para el desempeño de la comunicación se ha creado la función `events(evento1, evento2, evento3, evento4)`, en la que se recogen todas las posibles comunicaciones y eventos que tendrá la aplicación.

De esta forma se gestiona la conexión de los clientes y los respectivos eventos:

```
1   this.io.on('connection', function (socket) {
2     console.log("Conectado cliente");
3
4     socket.on(evento1, function (mensaje) {
5         console.log(mensaje);
6     });
```

Código 4.4 – *Gestión conexión de clientes.*

4

Cada cliente se conecta a sus respectiva sala, para que el servidor no envíe iformación innecesaria a otros clientes, y así no se ralentice el funcionamiento. En este caso las botas, cada una con un chip ESP32, se conectan y acceden a una sala llamada ESP.

```
1   socket.on(crearsala, function (room) {
2     socket.join("ESP");
3     socket.in("resume").emit("bota", "ONLINE");
4     console.log("Entrando en la sala: " + nombresala +" " + socket.id);
5 });
```

Código 4.5 – *Gestión conexión ESP32.*

En la linea 3 se realiza un envío hacía el cliente web para indicar que la bota se ha conectado y mostrar su estado.

Estos clientes serán, la web en la que se ve la medición en tiempo real y las botas[2]. Para la correcta transmisión de información los clientes deben estar correctamente conectados y dentro de su sala.

El cliente socket de la web se conecta mediante la instrucción

```
1   var socket = io.connect();
```

Código 4.6 – *Conexión del cliente web al servidor socket.*

Una vez conectado con éxito y con la medida comenzada, el cliente web se queda a la espera de los datos para actualizar y modificar la gráfica correctamente.

[2]Para ExoBOOT se explicará dicho procedimiento en la sección 4.4.

4.3.3.2.1 Eventos

Los diferentes eventos a los que se hace uso son:

- **connection** –> Un cliente se conecta al servidor websocket
- **crearsala** –> Se crea la sala donde entra la web y ESP32
- **b_izq y b_dch** –> En este evento se recibe el dato de la bota izquierda o derecha. Cada bota manda un evento diferente para identificar correctamente cada dato, pero ambas se insertan en la misma base de datos.
- **medicion** –> Con este evento se empieza la medición, así como lo relacionado con la base de datos y se prepara todo para la inserción de los datos.
- **end** –> cuando la bota termina el envío de datos se envía este evento para detener la medida y proceder al procesamiento de la base de datos. Este procesamiento se realiza en referencia al instante de tiempo para en un futuro mostrarla a través de diferido de forma correcta, ya que el chip ESP32 solo devuelve los milisegundos desde que lleva encendido.
- **disconnect** –> Evento usado para controlar que las botas se encuentren conectadas al websocket y así poder realizar correctamente las mediciones y las conexiones.

Por otra parte, en los cliente también se utilizan eventos para que el servidor se comunique con ellos, estos son;

- **start** –> Cuando comienza la medición y se pulsa el botón, el servidor envía esta información para que se empiece la medida, lo envía junto con el dato del tiempo que durará la medida.
- **grafica** –> Este evento se usa para que el servidor reenvíe el dato que recibe del sistema ExoBOOT a la vez que lo inserta en la base de datos correspondiente.

4.3.3.3 Gestión de los datos

Los datos recibidos son guardados en el directorio llamado `datos` y en un fichero `nombre.db`, el cual se le asocia el nombre que se establece en la configuración. Esto se genera automaticamente. Ese fichero donde se almacenan las medidas sigue una estructura de JSON y se usa NoSQL como se explicó en la sección 3.4.2.2 del capítulo anterior. Estás medidas, una vez finalizado el tiempo establecido se normalizan, para así poder mostrar en diferido la información correcta.

```
{"a":{"x":0,"y":1,"z":2},"g":{"x":0,"y":1,"z":2},
"m":{"x":0,"y":1,"z":2},"p":30,"t":0,"b":0,"f":4,
"tm":1604436556780,"bota":"izquierda","_id":"wiZkclFkLlI66cZU"}
```

Este es un ejemplo del dato almacenado, el tiempo se encuentra en milisegundos desde el 01-01-1970 y los datos se encuentran tal cual son recogidos. Esto se comentó en la sección 3.4.3 del capítulo anterior, en la cual se especificaba a que hacía referencia cada etiquta.

Toda está gestión de datos se lleva a cabo a través de la clase `dbManager`, declarada en `models/dbManager.js`. Con esta clase se pretende tener todo lo relacionado con la base de datos en un mismo fichero y así poder realizar modificaciones más fácil.

Cuando se crea el objeto de la base de datos se tiene que pasar el fichero donde se encuentra la base de datos y si es el archivo donde se guarda la información de las medidas, llamado `conf.db`. Aquí los datos siguen la estructura:
`{"date":1603928250456,"name":"Prueba","time":"1","_id":"tWgTVqQLskRF6UhT"}`
con el *date* en la fecha que se creó en milisegundos, el *name* el nombre de la medida y *time* el tiempo que duró.

Una vez creado dicho objeto correspondiente, con su correspondiente base de datos, se cargan los datos que contiene.

```
1  constructor(db, conf=null){
2        if(conf){
3            this.database = new Nedb({ filename: './config/' + db })
4            this.database.loadDatabase();
5        }
6        else{
7          this.database = new Nedb({ filename: './datos/' + db })
8          this.database.loadDatabase();
9        }
```

Código 4.7 – *Constructor* `dbManager`.

Para la gestión de esos datos NeDB tiene métodos, como insertar (*insert*), listar (*find*) o actualizar (*update*) que devuelven el objeto correspondiente en caso de que se realice con éxito o un error en caso contrario. En la clase creada realizó estas funciones mediante el uso de *Promise* [16] de NodeJS. Estas *Promise* devuelven los datos si las operaciones se realizan correctamente y en caso contrario devuelve error.

```
1  insert(data){
2    return new Promise(function(resolve, reject){
3        this.database.insert(data, function(err){
4            if (err)
5                reject(err);
6             resolve();
7        });
8    }.bind(this))
9  }
10
11 updateTimestamp(doc,new_time){
12
13     return new Promise(function(resolve, reject){
14         // tm es timestamp
15        this.database.update({ tm: doc.tm }, { $set: { tm: new_time } }, {}, function (err,
           numReplaced) {
16            if (err)
17               reject(err)
18            else
19                resolve();
20
21        })
22   }.bind(this));
23 }
24
25 get( bota){
26     var find = {}
27     if(bota){
28
```

```
29          find = { bota: bota }
30      }
31      // tm es timestamp
32      return new Promise (function(resolve, reject){
33          this.database.find(find).sort({ tm: 1 }).exec(function (err, docs) {
34              if(err)
35                  reject(err)
36
37              resolve(docs);
38          });
39
40
41      }.bind(this));
42
43
44  }
```

Código 4.8 – *Métodos de* `dbManager` *para listar, insertar y actualizar.*

Cuando la medida ha finalizado esta clase también se encarga de la correcta normalización de los datos partiendo de una hora inicial, que es el instante en el que se insertó el primer dato. Esa normalización se lleva a cabo con:

```
1   async normalize(start_hour) {
2           var items = 0;
3           var millis_inicial = 0;
4           var new_time = 0;
5           var docs = []
6
7           try {
8               // tm es timestamp
9                   for(let b of ["izquierda", "derecha"]){
10                  docs = await this.get(b)
11                  items= docs.length
12                  if(items)
13                      millis_inicial = docs[0].tm;
14                  for (var i = 0; i < items; i++) {
15
16                      new_time = (docs[i].tm - millis_inicial) + start_hour;
17                      if (new_time > start_hour*1.5 ){
18                          //PAra no modificar datos dobles.
19
20                      }
21                      else{
22                          await this.updateTimestamp(docs[i], new_time);
23                      }
24                  }
25              }
26              //Se vuelve a cargar la base de datos para actualizar las entradas.
27              this.database.loadDatabase();
28
29          }
30          catch (err) {
31              console.log("Normalize ERROR" + err.message);
32          }
33
34
35      }
```

Código 4.9 – *Método para normalizar hora en* `dbManager`.

donde se separan los datos de cada bota para así normalizarlos por separado y que los tiempos no se mezclen.

4.3.3.4 Controller

Por la parte del controlador se gestiona lo referente a los datos y a su renderizado en la parte de las vistas. Estos controladores están asociados a las rutas que se pueden consultar en la aplicación y están ubicados en `webapp/controllers`. Este proyecto consta de 3 rutas básicas:

- **index/** - Esta es la ruta principal del proyecto, en ella se leen todas las bases de datos que se encuentran en el directorio `datos/` para dar la posibiliadd de elegir una de ellas.

```
1     router.get('/', async (req, res,next) =>{
2       var db = new DataStore1("config.db", true)
3       var files = await db.get()
4       var files_nodisponibles = db.list()
5       res.render('index_css', { files, files_nodisponibles});
6   });
```

Código 4.10 – *Controlador ruta principal.*

- **directo/** - Este controlador es el que gestiona que se guarde la información de la medida en la base de datos con la información de todas las medidas. Otro aspecto del que se encarga es la de enviar a la vista la información de la medida a realizar.

```
1   router.get('/:id/:time', function (req, res, next) {
2     res.render('realtime/resume', { 'name' : req.params.id, 'tiempo': req.params.time
       });
3 });
4
5 router.post('/save', function (req, res, next){
6     var data = req.body;
7
8     let date = Date.now();
9    DataStore.insert({'date': date,'name' :data.name, 'time':data.time})
10
11    res.redirect('/directo/'+data.name+'/'+data.time)
12 })
```

Código 4.11 – *Controlador ruta `directo/`.*

- **diferido/** - Esta es la ruta a la que se accede cuando se consulta una medida que ya se realizó. Para poder consultar la medida se debe pasar el identificador de la base de datos existente y si no existe devuelve un error. Una vez seleccionada la base de datos correcta, el controlador se encarga de crear el objeto asociado a dicha base de datos, cargar los datos en diferentes arrays para así mostrarlos en sus respectivas gráficas y por último esos datos se renderizan en la vista.

```
1     router.get('/:id', async (req, res, next) => {
2
3       var file_db =  req.params.id +".db"
4       if (file_db.match(/\.db$/)) { // se hace un filtrado para que coja la base de
         datos correcvta
5           var datos = new DataStore(file_db);
6           var docs1 = await datos.get("izquierda")
7           var docs2 = await datos.get("derecha")
8
9           res.render('diferido/diferido', { info: file_db, docs1: docs1, docs2: docs2
             });
10
11      }
12      else{
13          res.render('error/error');
14      }
15  })
```

Código 4.12 – *Controlador ruta `diferido/`.*

Todas estas rutas llevan asociadas una plantilla para mostrar la información que procesan. La creación y desarrollo de la parte visual del proyecto se encuentra en la siguiente sección.

4.3.4 Frontend

El *frontend* es la parte visual, en forma de web, donde ver los datos e información que se está recibiendo y generando por parte de ExoBOOT. Esta parte se ha generado con una plantilla CSS de acceso gratuito [8] y el motor de plantillas PUG [22], como se comentó en el capítulo anterior.

En primer lugar, nada más ejecutar la aplicación y en el momento que se abre el navegador web automáticamente, aparece la página de inicio.

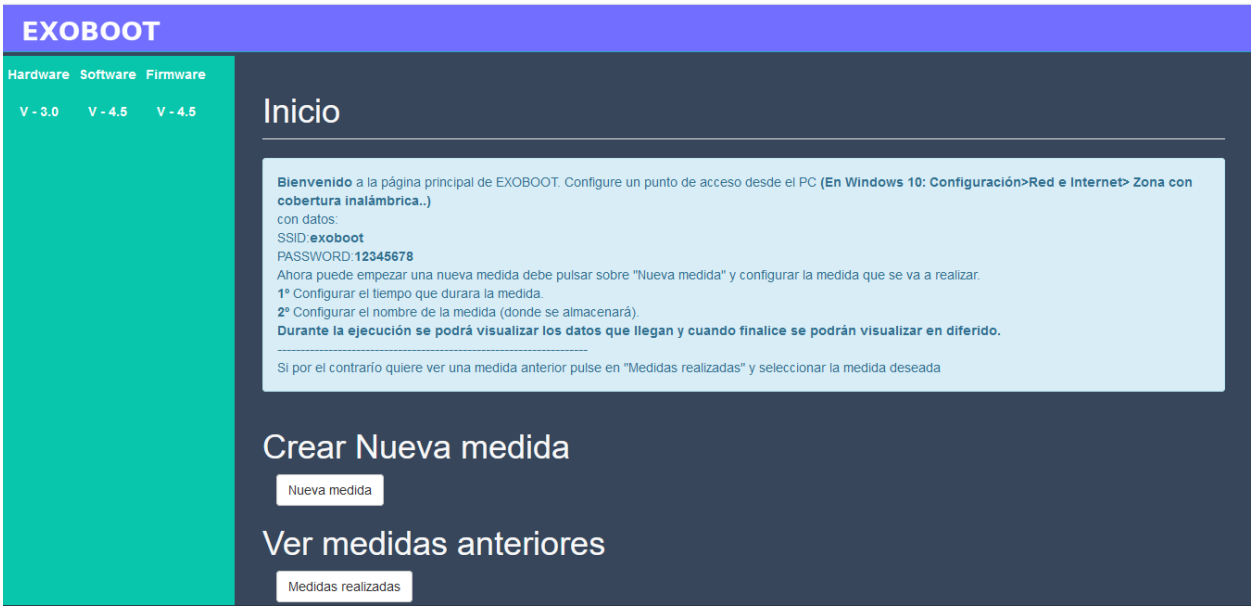


Figura 4.6 – *Pantalla inicio frontend.*

En la que se puede observar un párrafo con información de como crear el punto de acceso en Windows para la conexión de las botas, así como crear las diferentes medidas y como visualizarlas en diferido. Esta pantalla también ofrece los diferentes botones asociados a esa creación o vista en diferido. En el caso de crear una nueva medida aparecerá un formulario para introducir el nombre y tiempo que durará dicha medida, figura 4.7. De igual modo si se elige ver las medidas anteriores, aparecerá una tabla detallada de cada medida y la posibilidad de acceder a sus datos 4.8.

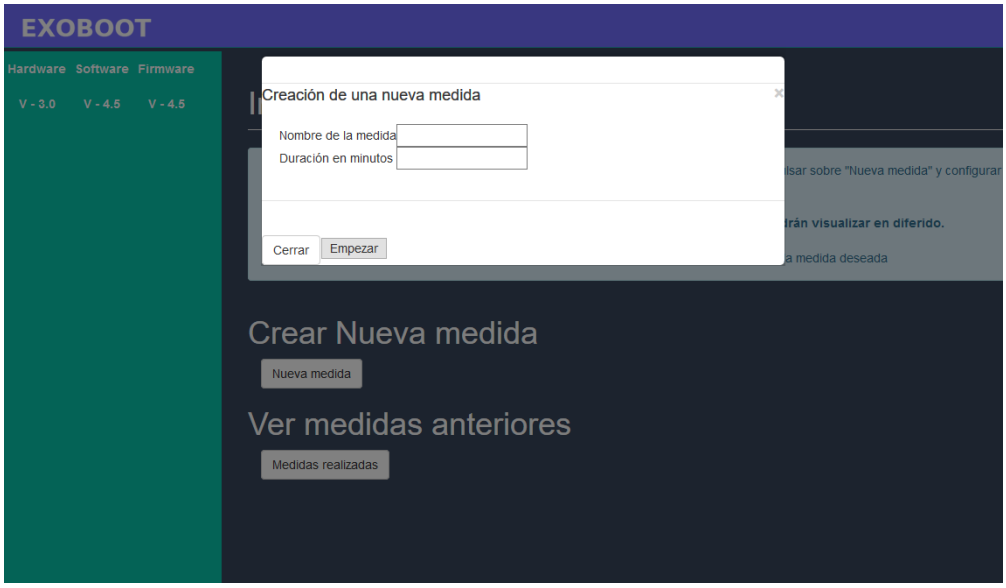


Figura 4.7 – *Configuración tomar medida.*

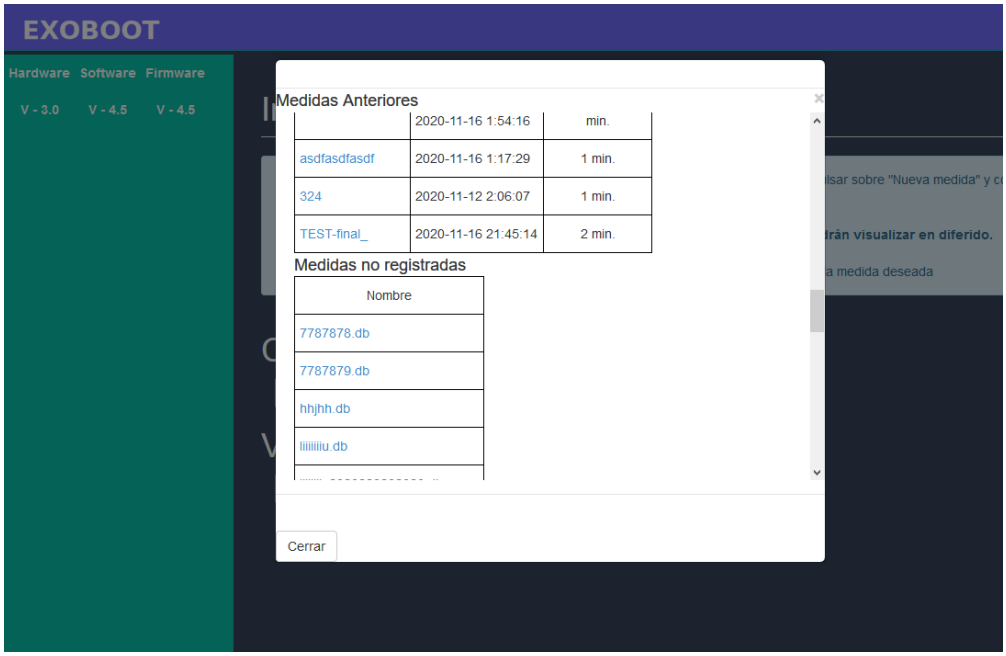


Figura 4.8 – *Medidas anteriores.*

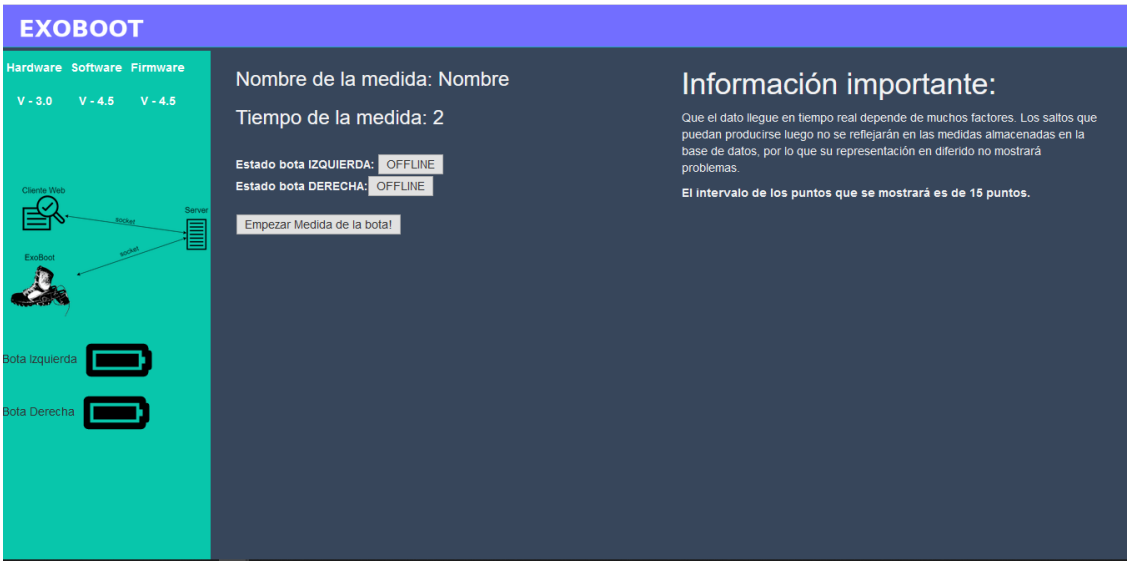


Figura 4.9 – *Configuración frontend.*

Una vez configurada la medida que se va a realizar (figura 4.9), aparecerá información relevante sobre el funcionamiento del sistema, al igual que el nombre de la medida y el tiempo que tardará en ejecutarse, se puede empezar la medida, siempre y cuando las botas se encuentren conectadas. Para empezar dicha media hay que hacer click en "empezar medida" y empezará a recibir datos el sistema mostrándose dichas medidas en las diferentes gráficas que se visualizan en está pantalla (figura 4.10).

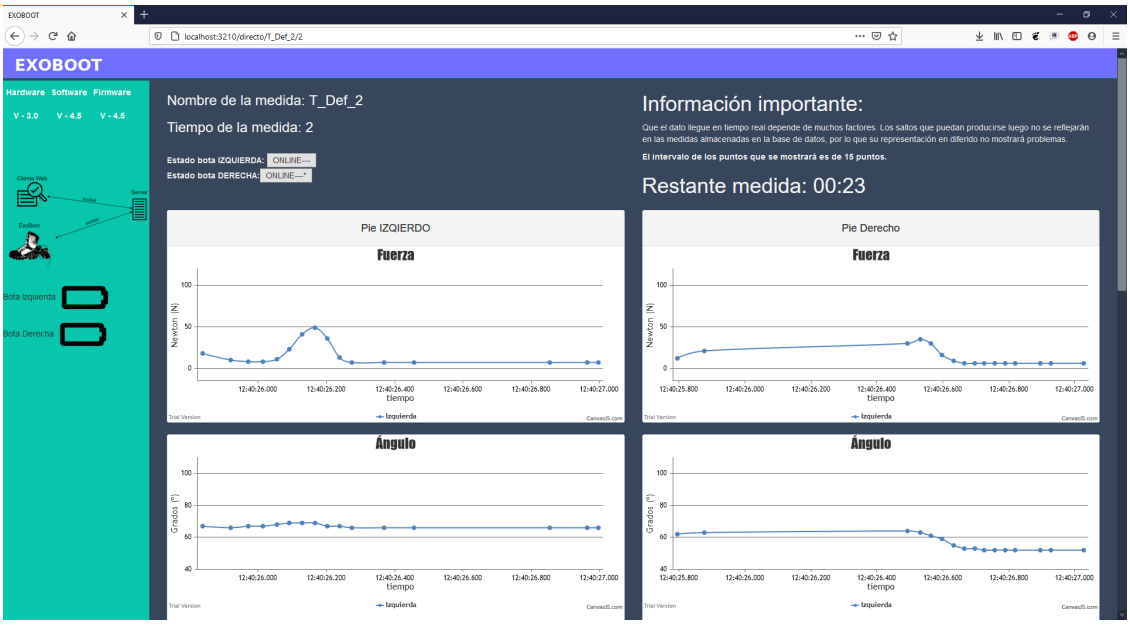


Figura 4.10 – *Gráficas de una medida en tiempo real.*

Una vez finalizada la medida mostrará información de que se está normalizando la base de datos (figura 4.11) y cuando finalice aparecerá un elance para redireccionarse a la ruta de diferido (figura 4.12).

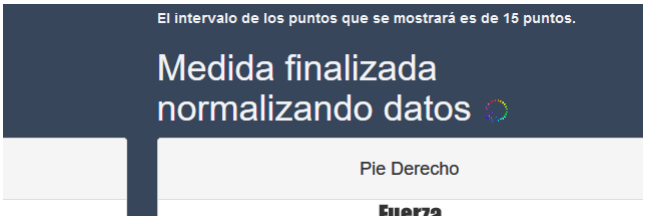


Figura 4.11 – *Procesando normalizado.*

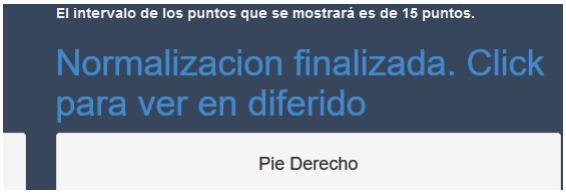


Figura 4.12 – *Normalizado completado y redirección.*

La última pantalla del *frontend* es la que concierne a los datos diferidos, figura 4.13

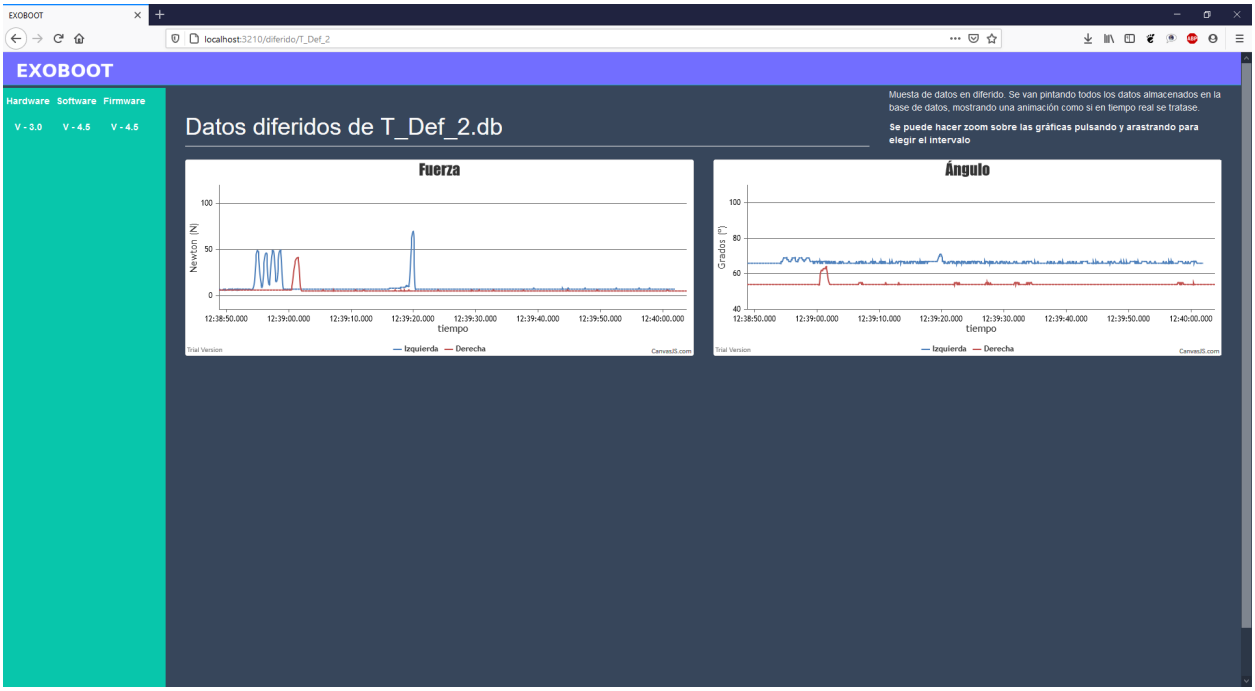


Figura 4.13 – *Medidas en diferido.*

Como se observa tanto en la captura 4.10 y 4.13 solo se visualizan la fuerza y el ángulo. Todas las demás gráficas están declaradas y preparadas para cuando se de el siguiente paso de solucionar los problremas hardware;

Para que esto funcione debe ir acompañado de un fichero *JavaScript* que se encarga de su control. En otras palabras, es el controlador explicado en la sección 4.3.3.4.

Tanto la vista en diferido, como en directo, incluye código en *JavaScript* para gestionar esos datos que sirve el controlador. Por ejemplo, en la vista en directo se hace uso de la librería *Socket.io* [23], al igual que en el firmware y en el *backend*. Con el uso de esta librería se pretende que la comunicación y la visualización de datos sea todo lo posible en tiempo real.

En el caso del diferido, ese *JavaScript* dentro de la vista, se encarga de hacer posible la visualizacion de los datos en las gráficas, permitiendo ver esas medidas como si estuvieran en tiempo real.

4.3.5 Generar ejecutable

Para generar el ejecutable se utiliza *PKG* [26], una herramienta de NodeJS que encapsula todo el proyecto en un ejececutable, tanto para Windows, MacOs o Linux.

Una vez completado el desarrollo y previamente instalado el encapsulador, estas *builds* se generan ejecutando: `pkg .package.json -out-path build/ -debug`. Se le pasa una ruta en la que se generará y el archivo de configuración del proyecto, es decir el `package.json`.

4.4 Implementación firmware

Esta implementación se ha desarrollado siguiendo la siguiente estructura:

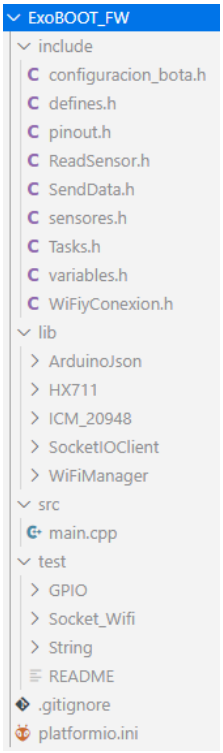


Figura 4.14 – *Estructura firmware.*

Esta estructura se crea al generar un nuevo proyecto en Platform.io [21], (un potente complemento de VisualStudio code). En dicha estructura los elementos son los siguientes:

- **include** -> En este directorio van los archivos de cabecera y declaración de macros. Aquí se encuentran las funciones y elementes necesarios para el proyecto, como son las lecturas de sensores, el envío a través del websocket o las conexiones al servidor. En las respectivas secciones se enecuentran más detalladas dichas funciones.
- **lib** -> Aquí se incluyen todas las librerias privadas u obtenidas, como las necesarias para la lectura de sensores, para la conexión al servidor e internet o las necesarias para tratar JSON en Arduino. Algunas librerías han sido modificadas para añadir una funcionalidad extra o una mejora en el desarrollo.
- **src** -> Aquí se encuentra el programa principal, el cual realiza la configuración del ESP32 y carga todo lo necesario.
- **test** -> Por último, el directorio donde se encuentran los test, en este caso los test se corresponden con la conexión GPIO, para comprobar que todos los sensores están disponibles y la conexión al servidor *socket*. Dicha ejecución se realiza con ayuda de *Platform.io*.
- **platform.ini** -> donde está la configuración del proyecto en *Platform.io* [21].

4.4.1 Test

Estos test realizados se han hecho con unity.h [11], pequeño y funcional. Esto ha ayudado a detectar errores antes del proceso de desarrollo del firmware.

```
1
2   #include <Arduino.h>
3   #include <unity.h>
4   #include <HX711.h>
5   #include "pinout.h"
6   #include "sensores.h"
7
8   void test_DOUT_PIN(void)
9   {
10      TEST_ASSERT(25 == DATAOUT);
11  }
12
13  void test_CLK_PIN(void)
14  {
15      TEST_ASSERT(26 == CLK);
16  }
17
18  void test_POT_PIN(void)
19  {
20      TEST_ASSERT(33 == POTPIN);
21  }
22
23  void test_OUT_PIN(void)
24  {
25      TEST_ASSERT(2 == OUTPUT_PIN);
26  }
27
28  void test_NO_read_ICM(void){
29      TEST_ASSERT(ICM_conexion(10));     // Así falla el test. No detecta IMC.
30  //      TEST_ASSERT(!ICM_conexion(10)); // de está forma el test es correcto.
31
32  }
33
34  void test_read_HX711(void){
35      HX711 scale;
36      scale.begin(DATAOUT, CLK);
37      TEST_ASSERT_GREATER_OR_EQUAL(0,  scale.read());
38  }
39
40
41  void test_read_POT(void){
42      TEST_ASSERT_GREATER_THAN(0, analogRead(POTPIN));
43  }
44
45  void setup()
46  {
47      delay(2000); // service delay
48      UNITY_BEGIN();
49
50
51      RUN_TEST(test_DOUT_PIN);
52      RUN_TEST(test_CLK_PIN);
53      RUN_TEST(test_POT_PIN);
54      RUN_TEST(test_OUT_PIN);
55
56      RUN_TEST(test_NO_read_ICM);
57      RUN_TEST(test_read_POT);
58      RUN_TEST(test_read_HX711);
59      UNITY_END(); // stop unit testing
60  }
61
62  void loop(){
63
64  }
```

Código 4.13 – *Test pines GPIO.*

En este test se comprueba que los pines sean los correctamente declarados y documentados del sistema ExoBOOT, que son los encargados de leer de los sensores. Tras ejecutar dichos test se obtiene el resultado que se muestra en la siguiente imagen.

```
test\GPIO\test_GPIO.cpp:73:test_DOUT_PIN       [PASSED]
test\GPIO\test_GPIO.cpp:74:test_CLK_PIN [PASSED]
test\GPIO\test_GPIO.cpp:75:test_POT_PIN [PASSED]
test\GPIO\test_GPIO.cpp:76:test_OUT_PIN [PASSED]
test\GPIO\test_GPIO.cpp:28:test_read_ICM:FAIL: Not connected    [FAILED]
test\GPIO\test_GPIO.cpp:79:test_read_POT       [PASSED]
test\GPIO\test_GPIO.cpp:80:test_read_HX711     [PASSED]
-----------------------
7 Tests 1 Failures 0 Ignored
[FAILED] Took 33.15 seconds
```

Figura 4.15 – *Unitest para GPIO.*

Como se observa, el test relacionado con el IMC, es decir con la IMU, falla. Esto es debiedo a que, como se ha comentado anteriormente, en un fallo en el proceso de desarrollo hardware no se conectó dicho sistema.

El otro test es el realacionado con la conexión al servidor websocket, dónde se comprueba como se ha de conectar al servidor websocket y recibir información. Este test pasa sin problemas.

```
1
2   #include <Arduino.h>
3   #include <unity.h>
4
5   #include "ReadSensor.h"
6   #include "SendData.h"
7
8   SocketIOClient socketclient_test;
9   bool conexionWifi(int intento)
10  {
11      WiFi.begin(ssid, password);
12      while (WiFi.status() != WL_CONNECTED && intento != 0)
13      {
14          delay(85);
15          intento--;
16      }
17
18      return WiFi.isConnected();
19
20  }
21
22  void test_WIFI_conn(void)
23  {
24      TEST_ASSERT_TRUE(conexionWifi(10));
25  }
26
27
28  void test_socket_conn(void)
29  {
30      conexionWifi(10);
31      TEST_ASSERT_TRUE(socketclient_test.connect(const_cast<char *>(WiFi.gatewayIP().toString
          ().c_str()), port));
32  }
33
34  void test_send_recive_socket(void)
35  {
36      socketclient_test.send("crearsala", "nombresala", "ESP_test");
37      socketclient_test.send("test", "prueba", "HOLA!");
38      socketclient_test.monitor();
39
40      if(RID == "test"){
41          TEST_ASSERT_EQUAL("OK", Rcontent.c_str());
42      }
43
44
45  }
46
47  void setup()
48  {
49      delay(2000); // service delay
50      UNITY_BEGIN();
51
52     // RUN_TEST(test_WIFI_conn);
53      RUN_TEST(test_socket_conn);
54      RUN_TEST(test_send_recive_socket);
55
```

```
56        UNITY_END(); // stop unit testing
57    }
58
59    void loop()
60    {
61    }
```

Código 4.14 – *Test conexión socket.*

4.4.2 Estados Firmware

El firmware de ExoBOOT se ha programado siguiendo una serie de estados y pasos para garantizar el correcto funcionamiento de este, así como aprovechar al máximo la potencia del NodeMCU. El sistema de programación, como se puede apreciar en la figura 4.16 sigue la siguiente jerarquía:

Como inicio, o sí se reinica la bota hay que comprobar sí se encuentra dentro del alcance del punto de acceso creado adecuadamente (4.4.2 lineas 1 a 4). Si esta operación tiene exito se realiza un intento de conexión al servidor socket, que se encuentra en la puerta de enlace de la conexión WiFi (4.4.2 linea 5). Entra en el estado de intento de conexión hasta que es posible la conexión, tanto del punto de acceso como del servidor socket.

Una vez establecida la conexión al websocket, primero entra en la sala correspondiente y después se realiza un bucle infinito, esperando la confirmación de inicio de medida. Esta iteracción es infinita hasta que recibe el dato del tiempo, en minutos, que estará en funcionamiento y en el que se realizara la medidas (4.4.2 lineas 11 a 22). La parte del código que se muestra a continuación, se realiza en el `main.cpp` del proyecto, dentro de la sección *setup*.

```
1   while (WiFi.status() != WL_CONNECTED)
2   {
3     delay(500);
4     Serial.print(".");
5   }
6   while (!socketclient.connect(const_cast<char *>(WiFi.gatewayIP().toString().c_str()), port)
     )
7   {  }
8
9   socketclient.send("crearsala", "nombresala", "ESP");
10
11  bool start = false;
12  int tiempoInt = 0;
13  while (!start)
14  {
15    socketclient.send("keepalive", "sigo", "conectado");
16    socketclient.monitor();
17    if (RID == "start")
18    {
19      start = true;
20      tiempoInt = Rcontent.toInt();
21      time_to_stop = tiempoInt * 60000 + millis();
22      Serial.print("EMPIEZA MEDIDA--aaaa");
23    }
24  }
```

Código 4.15 – *Conexión WiFI, socket y entrar en sala.*

Una vez completada esta rutina y recibido el tiempo de medida se procede a la creación de las tareas para que un *core* realice la lectura de los sensores y el otro el envío de estos datos.

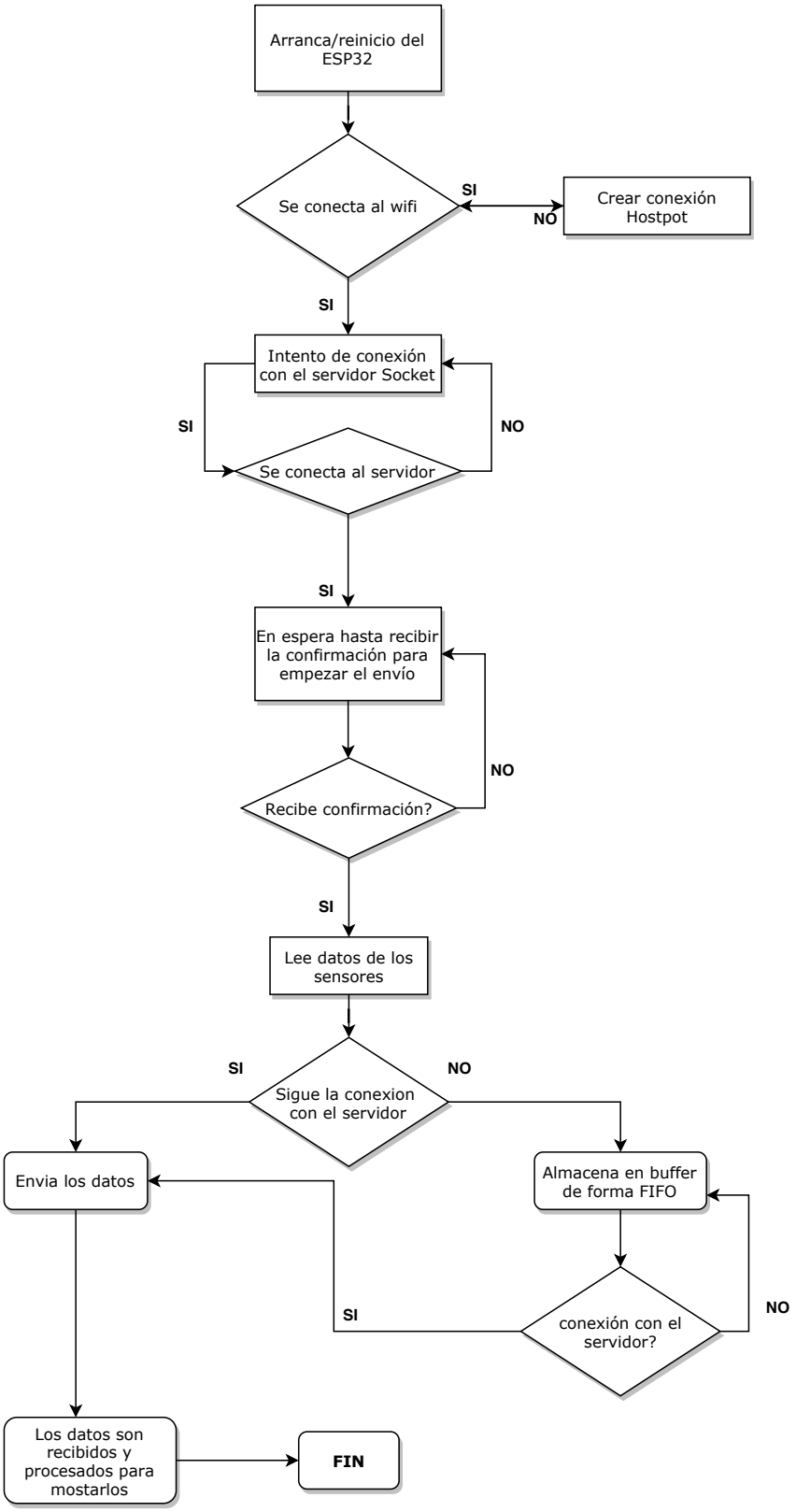


Figura 4.16 – *Flowchart del funcionamiento del firmware de ExoBOOT.*

4.4.3 Creación y comunicación de tareas

Aprovechando el doble núcleo del chip ESP32, se han creado dos tareas, una para la lectura de sensores y otra para el envío de los datos registrados. Para comunicar estos datos a través de las tareas se usan colas y en concreto la librería de FreeRTOS *QueueHandle_t*. Esta librería ya gestiona internamente la exlusión mutua y la sección crítica.

4.4.3.1 Creación de tareas

Para la creación de las tareas primero se deben declarar:

```
1 TaskHandle_t Task1 = NULL,
2              Task2 = NULL;
```

Código 4.16 – *Creación tareas.*

y posteriormente asociar lo que ejececutarán dichas tareas. *TaskHandle_t* tiene el método para crearlas en las que se le puede asociar el *core* donde se ejecutará. En el siguiente fragmento de código se puede apreciar como se crean dichas tareas. El método *xTaskCreatePinnedToCore()* recibe:

- Lo que ejececutará la tarea.
- Un nombre para esa tarea.
- El tamaño que tendrá el buffer de la tarea.
- Para pasar un parámetro a la tarea si se desea, en este caso es NULL.
- La prioridad de la tarea. Ambas tareas tendrán la misma prioridad.
- La TaskHandle_t que controlará la ejecución.
- El núcleo del procesador donde se ejecutará.

```
1 xTaskCreatePinnedToCore(tareaLectura,"tareaLectura", 10000,
2         NULL, 1,&Task1, 1);
3 delay(1);   // Este delay es necesario para empezar la tarea 1.
4
5 xTaskCreatePinnedToCore(tareaEnvio, "tareaEnvio", 10000,
6         NULL,1,&Task2,0);
7 }
```

Código 4.17 – *Asignación tareas a cores.*

Tanto *tareaLectura* como *tareaEnvío* son dos funciones *void*, que cada una ejecuta un bucle infinito para leer los sensores y otro para enviar el dato respectivamente. Para realizar una comunicación de estas se usa *QueueHandle_t*, con las funciones *xQueueSend* y *xQueueReceive*.

4.4.3.2 Comunicación tareas

En primer lugar hay que declarar dicha cola con un tamaño, asociado a la cadena que almacenará y un tamaño específico. La elección del tamaño de la cola de 200 es debido a que ofrece la posibilidad de recuperar frente a perdidas de conexión y no se llene. El tamañao de 166 es el tamaño máximo que tendrá el *string* porviniente de la serialización del JSON.

```
1| QueueHandle_t xQueue = xQueueCreate(200, 166);
```

Código 4.18 – *Creación cola*

Cuando un dato se ha leído y transformado a *string* (doc_send) se añade al fondo de la cola con *xQueueSend* y con los parámetros de la cola donde se añade, el dato leído y el tiempo máximo que se bloquea si la cola está llena. Se indica el valor 0, puesto que primero debe leer el dato antes de enviar.

```
1| if (xQueueSend(xQueue, (void *) doc_send, (TickType_t)0) != pdPASS)
2| {
3|   Serial.println("ERROR EN ALMACENAR ********************** LLENOO");
4| }
```

Código 4.19 – *Almacenar en cola.*

En el lado del envío se recibe ese dato y se envía al servidor websocket. *xQueueReceive* tiene los mismos párametros que *xQueueSend*, salvo que en este caso el tiempo máximo se asocia al bloqueo cuando está llena. El tiempo máximo es 85 milisegundos, ya que tarda ese tiempo en leer un dato.

```
1|   xQueueReceive(xQueue, (void *)doc_recive, (TickType_t)(85 / portTICK_PERIOD_MS));
```

Código 4.20 – *Extraer de cola.*

4.4.4 Lectura y envío

En primer lugar para la correcta lectura de los datos es necesario calibrar los diferentes sensores en ambas botas. En este caso y como se comentó anteriormente, solo se dispone del sensor de fuerza y un potenciómetro, con el que se obtiene el ángulo en el que se encuentra el talón de Aquiles.

La calibración del potenciómentro se realiza siguiendo los pasos especificados a continuación.

1 Se deben coger dos puntos, el ángulo en el que está y el voltaje de dicho valor.
Esos valores se toman en dos puntos diferentes, en la figura se muestra la referencia tomada para el ángulo y el primer punto medido. Al ser realizado en ambas botas, es necesario coger 4 puntos por bota.
En la figura 4.17 se muestra el método de obtención de dichos ángulos. Para llevar a cabo estas medidas la referencia ha sido toma siendo para el eje $y = 90$ º (posición

natural en estado de reposo) y $x = 0$ º (posición del dispositivo sin sujección). Por lo que en la imagen mostrada, el dispositivo tiene un ángulo de unos 10 º aproximadamente.

2 Realizar la regresión lineal de los datos citados anteriormente para obtener una expresión que devuelva el valor del angulo adecuado. Para calcular esta regresión se parte de la ecuación de la recta $y = mx + n$, para obtener el sistema de ecuaciones a resolver y obtener una ecuación a la cual poder pasarle el valor del voltaje para hallar el ángulo. Dicha ecuación es $\Theta = AV + B$ y ese valor A y B se obtiene a partir del sistema:

$$\begin{cases} \Theta_1 = AV_1 + B \\ \Theta_2 = AV_2 + B \end{cases} \tag{4.4.1}$$

Donde tanto Θ_1, V_1, Θ_2 y V_2 son los puntos conocidos y elegidos en el paso 1. Para resolver este sistema se ha seguido el método de sustiución, despejando en primer lugar la B de la segunda ecuación:

$$B = \Theta_2 - AV_2 \tag{4.4.2}$$

Figura 4.17 – *Referencia tomada para el cálculo de la regresión lineal. 0º Correspondiente a lo más cercano al talón y 180º lo más cercano a la punta.*

y sustituyendola en la primera ecuación:

$$\Theta_1 = AV_1 + (\Theta_2 - AV_2) \tag{4.4.3}$$

se despeja A, obteniendo su valor:

$$A = \frac{\Theta_1 - \Theta_2}{V_1 - V_2} \tag{4.4.4}$$

y por último se sustituye este A en la anterior B despejada 4.4.2:

$$B = \Theta_2 - \frac{\Theta_1 - \Theta_2}{V_1 - V_2} * V_2 \tag{4.4.5}$$

Trás esto ya se puede sustituir en la ecuación de la recta los valores A y B, para así encontrar los diferentes ángulos en referencia al voltaje del potenciómentro.

$$\Theta = \frac{\Theta_1 - \Theta_2}{V_1 - V_2} * V + \Theta_2 - \frac{\Theta_1 - \Theta_2}{V_1 - V_2} * V_2 \tag{4.4.6}$$

De esta forma la ecuación 4.4.6, es una ecuación general, que tras sustituir los valores de cada bota referentes a Θ_1, V_1, Θ_2 y V_2, obtendremos su ecuación de regresión lineal para la obtención del ángulo bien calibrado.

3 Insertar esa ecuación en el código. Una vez planteadas las ecuaciones para el cálculo de la regresión lineal es el momento de introducirlas en el firmware para una correcta lectura de los valores.

```
1       float fact_calibracion_pot_a_V = 3.3 / 4095.0;
2       float A = (ANGULO_THETA_1 - ANGULO_THETA_2) / (ANGULO_V_1 - ANGULO_V_2);
3       float B = ANGULO_THETA_2 - A * ANGULO_V_2;
4       int obtenerAnguloCalibrado(float lecturaPin)
5       {
6           return A * ( lecturaPin * fact_calibracion_pot_a_V ) + B;
7       }
```

Código 4.21 – *Ecuación en firmware*

Tanto para el factor de calibración, como A y B, se hace el cálculo basado en el elemento en *float*, aunque el angulo sea un entero. Esto se hace para poder ajustar al máximo el valor leído. Esas variables han sido declaradas fuera, para que solo se calculen una vez y resulte más eficiente la ejecución del código, puesto que la división es un cálculo muy costoso y más aún en un dispositivo con capacidades tan limitadas.

Por otro lado, la calibración del sensor HX711, se puede hacer con ayuda de su librería y aplicando una simple división. Para esto:

1 . Ejecutar el programa con el valor de la escala por defecto, es decir con valor 1 y con la tara, tambien por defecto que en este caso tiene un valor de 10. Este valor indica las veces que realiza la lectura para proceder a la tara. Para establecer la escala y la tara se puede ver como se hace en el código 4.4.4, lineas 12 y 14.

2 . Una vez establecido y con el programa en ejecución, se cuelga un peso conocido y se leen varios datos con ese peso encima. Para realizar está medida se usa el método *get_value* (4.4.4 linea 25), el cual hace una media del N que se establezca. En este caso n=20.

3 . La escala ya se puede calcular a partir de esa medida, pero en este caso he obtado por escoger 10 de esas medidas y sacar una única media, para tener una aproximación más exacta.

	Valor leído
1	77193
2	77210
3	76973
4	77192
5	77147
6	77155
7	77108
8	77132
9	77265
10	77125
Media	**77150,00**
Escala	**5048,09**

Tabla 4.1 – *Media valor HX711 para calibrar*

Una vez obtenida esa media se puede obtener el valor de la escala realizando

$$escala = \frac{Mediavalorleído}{Peso} \tag{4.4.7}$$

En este caso, como se busca que la medida sea en *Newton*, se ha realizado la escala a dividiendo por el peso, en *Newtons*, como se uso para calibrar

$$escala = \frac{77150}{1,6 * 9,86} = 5048,09 \tag{4.4.8}$$

4 . Insertar esa calibración en el código.

```
1  #include "HX711.h";
2  #define SERIAL_PORT Serial
3
4  #define F_PIN 25
5  #define CLK 26
6  #define OUTPUT_PIN 2
7  #define POTPIN 33
8
9  void setup() {
10   pinMode(OUTPUT_PIN, OUTPUT);
11   scale.begin(F_PIN, CLK);
12   scale.set_scale();  // Comentar cuando se haya calibrado
```

```
13    // scale.set_scale(5048,09);  // valor ajustado descomentar una vez calibrado
14    scale.tare();
15    SERIAL_PORT.begin(115200);
16    while(!SERIAL_PORT){};
17
18  }
19
20  void loop() {
21    SERIAL_PORT.print("\t\t\t\tFUERZA: ");
22    SERIAL_PORT.println(scale.read());
23
24     SERIAL_PORT.print("\t\t\t\tFUERZa MEDIA: \t\t\t");
25     SERIAL_PORT.println(scale.get_value(20));
26
27     //SERIAL_PORT.print(scale.get_units());  // metodo para medir el valor en newtons, usando
         la escala
28  }
```

Código 4.22 – *Calibración HX711.*

Una vez que todos los sensores están correctamente calibrados se procede al desarrollo completo de la lectura y envío. En primer lugar la parte de almacenar el dato tras la lectura se sigue el formato JSON. La librería usada para la correcta representación del JSON en arduino es `ArduinoJSON`. Ésta ofrece una herramienta [1] para calcular la memoria necesiria a reservar para representar el dato JSON, ya que esto es indispensable para un uso eficaz. Otros métodos disponibles son los usados para convertir de JSON a *string* y viceversa.

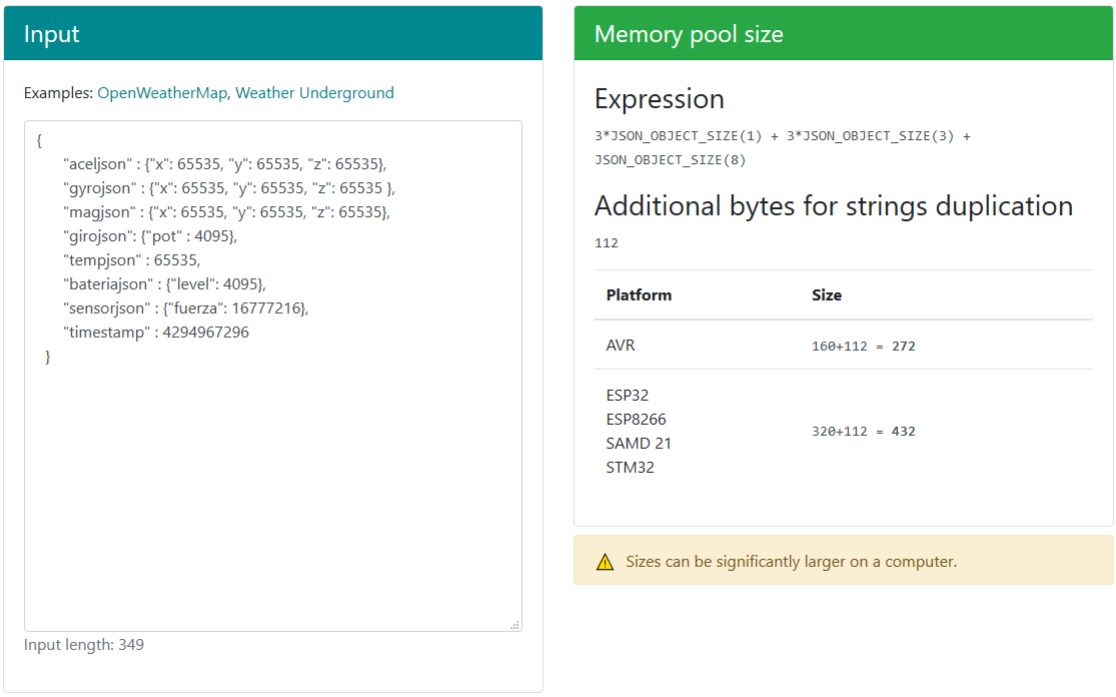


Figura 4.18 – *Captura herramienta ArduinoJSON [1].*

Para la correcta obtención del la memoria que se necesita a reservar y formar el JSON hay que rellenar el texto de la izquierda con el JSON que utilizaremos, así como valor máximo

que será representado. Tal como se muestra en la figura 4.18.

El JSON en cuestión, al igual que la representación de la variable es:

```
  const size_t capacity = 3 * JSON_OBJECT_SIZE(3) + JSON_OBJECT_SIZE(8);
/* para la nueva version si tantas letras /*
    Hay que especificar la capacidad del json
    Para saber el tamaño: https://arduinojson.org/v6/assistant/
    El JSON con la estructuras de datos quedaría:

        JSON: {
          "aceljson" : {"x": 65535, "y": 65535, "z": 65535},
          "gyrojson" : {"x": 65535, "y": 65535, "z": 65535 },
          "magjson" : {"x": 65535, "y": 65535, "z": 65535},
          "girojson": {"pot" : 4095}, SIN SIGNO 12 BITS
          "tempjson" : 65535,
          "bateriajson" : {"level": 4095},
          "sensorjson" : {"fuerza": 16777216}, SIGNO 24
          "timestamp" : 4294967296
      }

    Esos valores son asignados ya que 65535 es el mayor número que se puede representar en
    binario con 16 bit, al igual 4095 con 12 y 16777216 con 24.
    La capacida se ha obtenido con el asistente de arduinojson y en la seccion de serializing
     .*/
DynamicJsonDocument doc(capacity);

JsonObject a = doc.createNestedObject("a");
// a["x"] = 65535;
// a["y"] = 65535;
// a["z"] = 65535;

JsonObject g = doc.createNestedObject("g");
// g["x"] = 65535;
// g["y"] = 65535;
// g["z"] = 65535;

JsonObject m = doc.createNestedObject("m");
// m["x"] = 65535;
// m["y"] = 65535;
// m["z"] = 65535;
// doc["p"] = 4095;
// doc["t"] = 65535;
// doc["b"] = 4095;
// doc["f"] = 16777216;
// doc["tm"] = 4294967296;
```

Código 4.23 – *JSON creado con la herramienta de ArduinoJSON [1].*

En este caso la lectura se realiza mediante *pulling*, dado que como se ha comentado anteriormente, solo se leen los sensores de fuerza y ángulo, por lo que leyéndolo cada 85 milisegundos es suficiente.

Los datos leídos, como se ve en el siguiente código, se almacenan en el documento JSON. Al no poder obtener datos de los otros sensores se pasa un valor *dummy* para ver una representación en la gráfica.

```
  void lecturaDatos()
  {
    t = millis();
    a["x"] = dummy;
    a["y"] = dummy + 1;
    a["z"] = dummy + 2;

    g["x"] = dummy;
    g["y"] = dummy + 1;
    g["z"] = dummy + 2;

    doc["p"] = obtenerAnguloCalibrado(analogRead(POTPIN));

    m["x"] = dummy;
    m["y"] = dummy + 1;
    m["z"] = dummy + 2;

    doc["t"] = dummy;
    doc["b"] = dummy;

```

```
21      doc["f"] = int(scale.read() * fact_calibracion_fuerza);
22
23      doc["tm"] = t;
24      ...
```

Código 4.24 – *Lectura de datos y asignación al JSON.*

Cuando todos los datos han sido almacenados en el JSON se serializan para transformarlos a *string* y poder almacenarlos en la cola. Se realiza esta conversión con el objetivo de necesitar el menor espacio posible, así como una mejor comunicación entre las tareas.

```
1       ...
2     serializeJson(doc, doc_send);
3       ...
4   if (xQueueSend(xQueue, (void *) doc_send, (TickType_t)0) != pdPASS)
5   {
6     Serial.println("ERROR EN ALMACENAR ********************** LLENOO");
7   }
```

Código 4.25 – *Envío JSON a la cola.*

Estos datos se van almacenando mientras que la otra tarea realiza el envío de dichos datos al servidor.


```
1   void envioDatos()
2   {
3     if (socketclient.connected()){
4
5       xStatus = xQueueReceive(xQueue, (void *)doc_recive, (TickType_t)(85 /
          portTICK_PERIOD_MS)); // asi perdidos 580
6
7       if (xStatus == pdPASS)
8         socketclient.sendJSON(EVENTO_ENVIO, doc_recive);
9
10      else
11        Serial.println("ERROR EN LA COLAA ---------------------- VACIO");
12
13    }
14    else{
15        if(WiFi.isConnected())
16          while(!socketclient.connect(const_cast<char *>(WiFi.gatewayIP().toString().c_str())
              , port)){
17            Serial.print("socket reconect");
18          }
19        else
20        {
21          WiFi.begin(ssid, password);
22          while (WiFi.status() != WL_CONNECTED)
23          {
24            delay(85);
25            Serial.print(".");
26          }
27        }
28
29        }
30
31
32  }
```

Código 4.26 – *Sacar JSON de la cola y enviar a servidor.*

En este código lo que se realiza es la extracción de la cola y envío del dato a través del websocket al evento correspondiente de cada bota. Este evento se usa para identificar el dato almacenado, así como para pintarlo en las correspondientes gráficas y mostrar el dato con el método especificado en la sección 4.3.3.2.1.

Si se queda sin conexión intenta la reconexión al servidor *socket* y al punto de acceso. Cuando se reconecta empieza a enviar esos datos leídos en el momento en que se estaba

intentando la reconexión. Sí por el contrario, el problema es de la bota o no se conigue volver a conectar después de perder 200 paquetes, esta se apaga y se debe volver a empezar la medida.

```
  42["start",{"time":"1"}]  42["start",{"time":"1"}]
Message size = 24
Received message = 42["start",{"time":"1"}]
EMPIEZA MEDIDA--    [E][WiFiClient.cpp:392] write(): fail on fd 55, errno: 113, "Software caused connection abort"
.....................................................................................HTTP/1.1 200 OK
Content-Type: text/plain; charset=UTF-8
Content-Length: 102
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: Arduino
Set-Cookie: io=tYR3klKH0nHqKRQMAAAE; Path=/; HttpOnly
Date: Mon, 09 Nov 2020 23:37:22 GMT
Connection: keep-alive

95:0{"sid":"tYR3klKH0nHqKRQMAAAE","upgrades":["websocket"],"pingInterval":1000,"pingTimeout":5000}2:40Connecting via Websocket
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: zHuM6HyFD5q5IqKrO0Iso99I8Us=
Sec-WebSocket-Extensions: permessage-deflate


 ---------- FIN MEDIDA ------
 Paquetes en 1 minuto: 712 paq/min
Total tiempo envío: 60068.00 ms
Media de tiempo de lectura por los paquetes enviados (total_tiempo / paq): 84.37 ms
********************************
Paquetes PERDIDOS: 0
Paquetes LEidos 712 paq
********************************
Paquetes PERDIDOS: 0
```

Figura 4.19 – *Salida por pantalla de perdida de conexión.*

Como se ve en la salida por pantalla, 4.19 del chip ESP32, subrayado en color rojo, indica que se ha perdido la conexión al punto WiFi, indicando con puntos suspensivos, el intento de reconexión. Una vez esto tiene éxito continúa con dicho envío, el cual termina cuando la cola está vacía. Como se puede apreciar en dicha figura, se leen 712 paquetes y son 0 los paquetes perdidos.

4

CAPÍTULO

5

TEST Y EVALUACIÓN

En este último capítulo se verificará si los resultados obtenidos son los esperados en los diferentes requisitos establecidos en el capítulo 2.

Esta evaluación consistirá en verificar varios aspectos del proyecto. Dichos aspectos estarán divididos en las diferentes secciones que se encuentran a continuación.

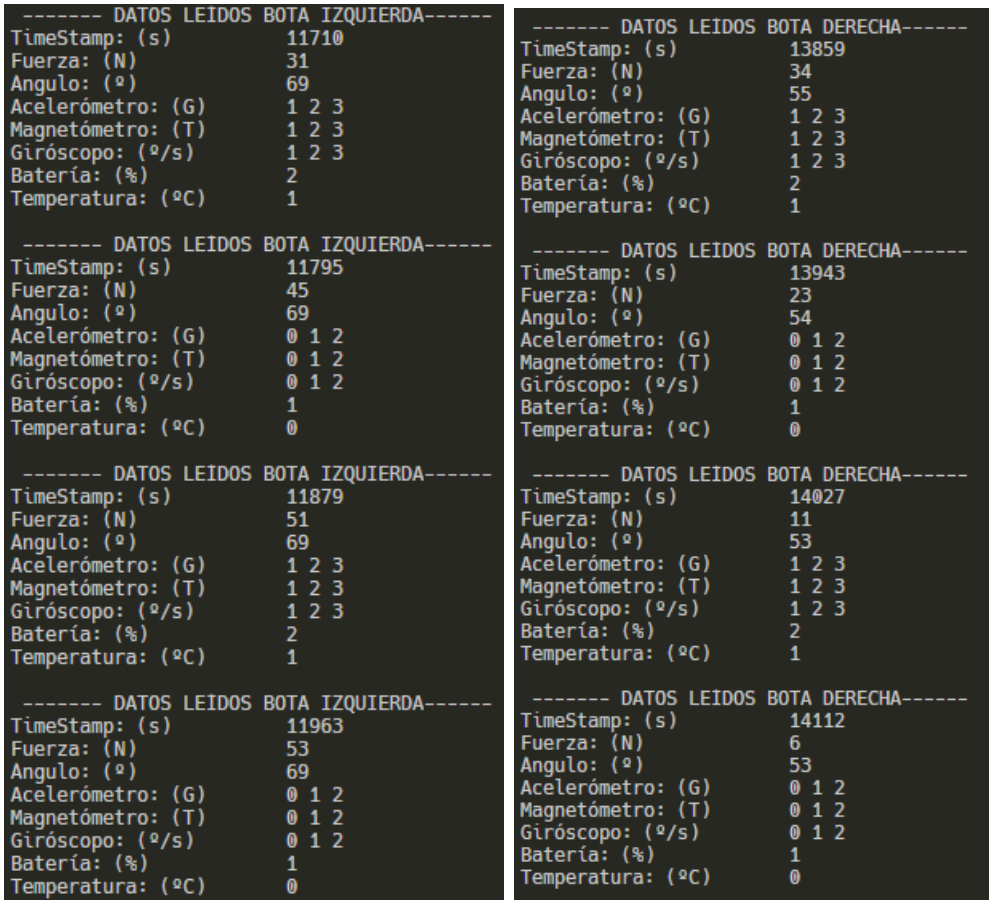
Una vez en este punto los requisitos no funcionales que se cumplen de por si,

- RNF 2 (Software 2.12), puesto que es un requisito indispensable sin el cual no se podría ejecutar nada.
- RNF 3 (Hardware 2.13), al igual que el anterior requisito, si no se dispone de las botas no se puede medir nada.
- RNF 4 (Diseño e implementación 2.14). Como se ha documentado en el capítulo 4, la parte software se encuentra desarrollada en NodeJS, así como el firmware en C.
- RNF 5 (Segurida 2.15). El sistema está disponible cuando el usuario quiere que esté.

5.1 Análisis de lectura de sensores en ExoBOOT

En esta primera sección se evaluará si el sistema ExoBOOT cumple con los requisitos referentes a la lectura y obtención de datos por parte de los sensores.

Dicha evaluación consistirá en ejecutar el firmware principal, pero con la opción de debuguear para mostrar la información por pantalla.



(a) *Bota izquierda* **(b)** *Bota derecha*

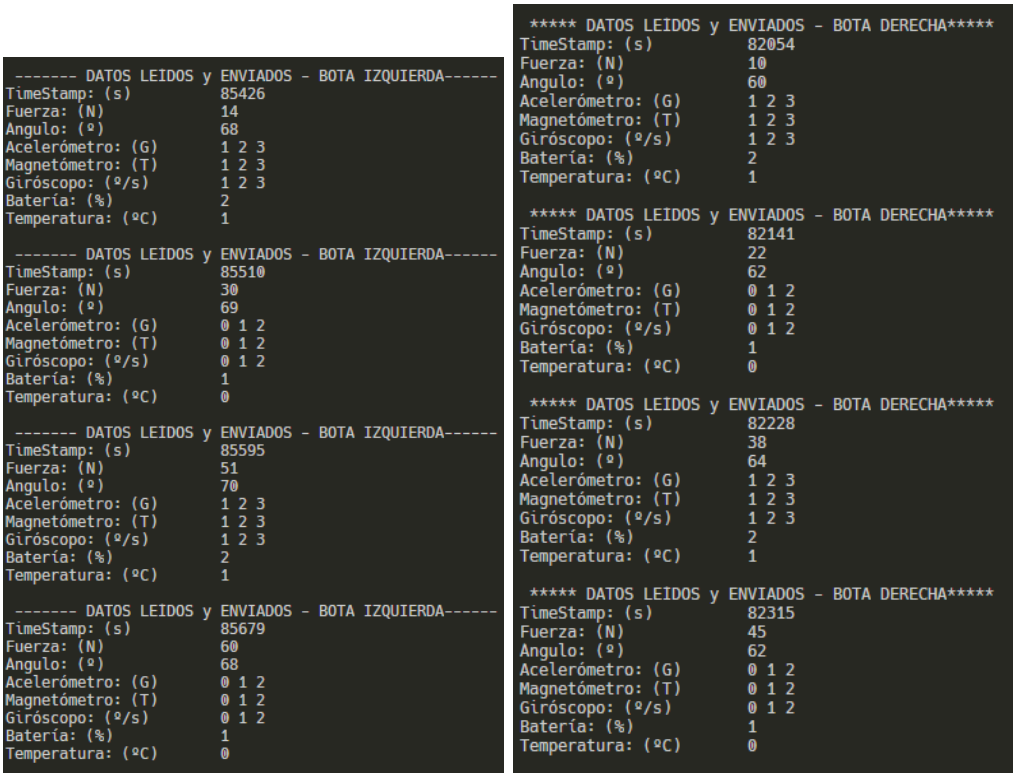
Figura 5.1 – *Resultado ejecución test lectura sensores.*

Como se puede comprobar en la figura 5.1 la lectura de los sensores, que se encuentran disponibles a nivel hardware, se realiza correctamente, devolviendo el dato normalizado y calibrado. Por otro lado nos encontramos los datos pertenecientes a los sensores no conectados, unos datos *dummy*, establecidos por defecto.

El hecho de que se lean correctamente los sensores hace que el requisito funcinal 2.3 (Leer datos tabla, tabla 2.4) se cumpla.

5.2 Análisis envío y recepción de datos

Al igual que en la sección anterior, esta evaluación continúa con la opción de debugueo, mostrando información por pantalla tanto en ExoBOOT como en el servidor.



(a) *Bota izquierda* **(b)** *Bota derecha*

Figura 5.2 – *Resultado ejecución test lectura y envío. ExoBOOT*

En estas capturas, al igual que en la sección anterior, se pueden ver las salidas de las lecturas que ExoBOOT realiza de los sensores correspondientes a cada una de las botas. A su vez estos datos son enviados al servidor.

```
--- DATO RECIBIDO BOTA IZQUIERDA ----

TimeStamp: (s)     85510
Fuerza: (N)        30
Angulo: (º)        69
Acelerómetro: (G)  0 1 2
Magnetómetro: (T)  0 1 2
Giróscopo: (º/s)   0 1 2
Batería: (%)       0
Temperatura: (ºC)  0
--- DATO RECIBIDO BOTA IZQUIERDA ----

TimeStamp: (s)     85595
Fuerza: (N)        51
Angulo: (º)        70
Acelerómetro: (G)  1 2 3
Magnetómetro: (T)  1 2 3
Giróscopo: (º/s)   1 2 3
Batería: (%)       1
Temperatura: (ºC)  1
**** DATO RECIBIDO BOTA DERECHA ******

TimeStamp: (s)     82228
Fuerza: (N)        38
Angulo: (º)        64
Acelerómetro: (G)  1 2 3
Magnetómetro: (T)  1 2 3
Giróscopo: (º/s)   1 2 3
Batería: (%)       1
Temperatura: (ºC)  1
--- DATO RECIBIDO BOTA IZQUIERDA ----

TimeStamp: (s)     85679
Fuerza: (N)        60
Angulo: (º)        68
Acelerómetro: (G)  0 1 2
Magnetómetro: (T)  0 1 2
Giróscopo: (º/s)   0 1 2
Batería: (%)       0
Temperatura: (ºC)  0
```

Figura 5.3 – *Resultado ejecución test envío en servidor.*

Como se ve en la 5.3 los datos llegan de ambas botas a la vez, superponiendose. Al solo disponer de una salida. Si se comparan ambas figuras se puede comprobar que los datos se corresponden. Esto hace que se cumpla correctamente con el requisito 2.4 (Enviar datos, tabla 2.5)

5.3 Almacenamiento de datos

Una vez verificado tanto que se leen los datos como que se reciben se ha de verificar que esos datos se pueden guardar, cumpliendo así los requisitos funcionales 3.1 (Recopilar datos recibidos, tabla 2.7) y 3.4 (Almacenamiento en base de datos, tabla 2.10). Esto se puede verificar en la siguiente captura, dónde se ven los datos almacenados en la base de datos con formato JSON.

Figura 5.4 – *Base de datos, medida de evaluación.*

Puesto que el requisito citado anteriormente se cumple, implica a su vez que otros requisitos se han resuelto con éxito. Dichos requisitos implicados son:

- 2.1 (Conexión WiFi, tabla 2.2)
- 2.5 (Conexión websocket cliente, tabla 2.6)
- 3.3 (Conexión websocket servidor, tabla 2.9)

5.4 Análisis de la muestra de datos

El siguiente aspecto a evaluar si cumple el proyecto es que esos datos recibidos sean mostrados mientras se reciben, así como su posible visualización en formato diferido.

Antes de empezar la medición el sistema ExoBOOT espera que se confirme el inicio de la medida. Esto se puede comprobar en la siguiente captura, ya que cuando recibe que se debe empezar la medida, muestra por pantalla el mensaje con dicha información. Así se verifica el requisito funcional 1 (Elegir tiempo a medir 2.1)

Como se observa en las figuras 5.5 y 5.6, se hace posible la visualización de los datos a través de las gráficas, por lo que esto hace que se cumpla el requisito funcional 3.2 (Mostrar los datos recibidos 2.8).

El llegar a este punto hace que se verifique también el RNF 1 (Usuario y sistema 2.11). Los aspectos de dicho requisito son superados, puesto que, como se documento en 4.4.4, el firmware está preparado para posibles incidencias de comunicación, enviando de este modo los datos leídos durante la desconexión del servidor websocket. El siguiente aspecto es verificado mediante las evaluaciones de las anteriores secciones, debido a que se ha ejecutado dicho binario. El hacer que los datos se puedan visualizar en diferido verifican el aspecto de la disponibilidad, de este requisito, así como el rendimiento que los datos se envían de la forma más rápida para obtener unas lecturas viables.

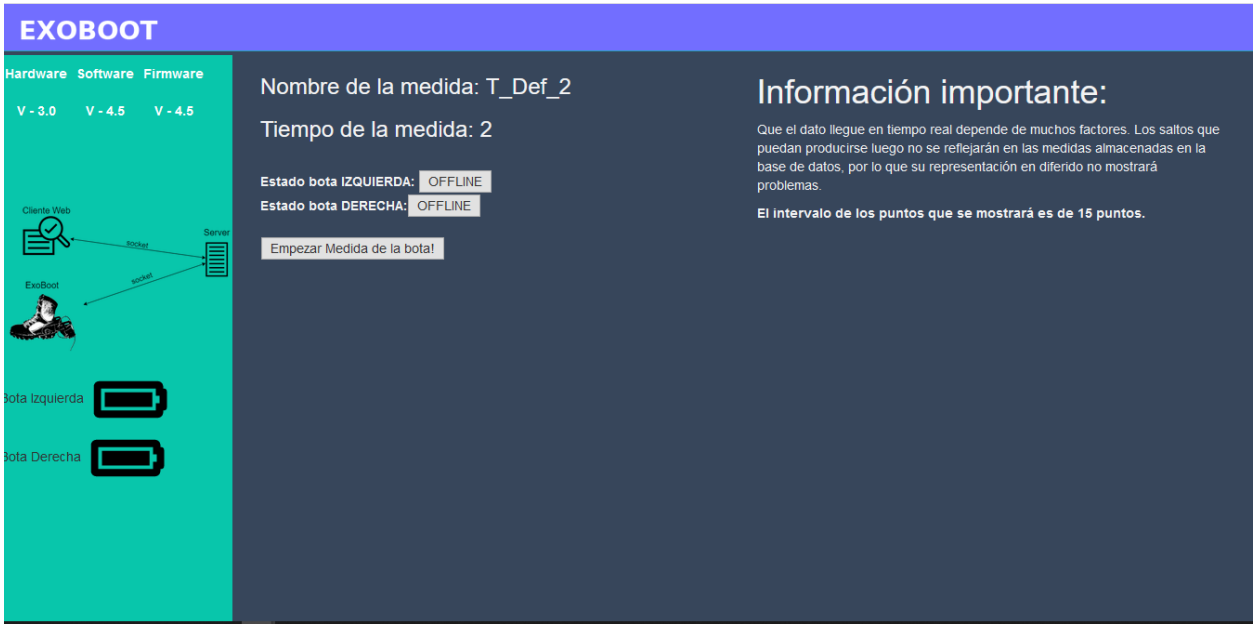


Figura 5.5 – *Configuración medida.*

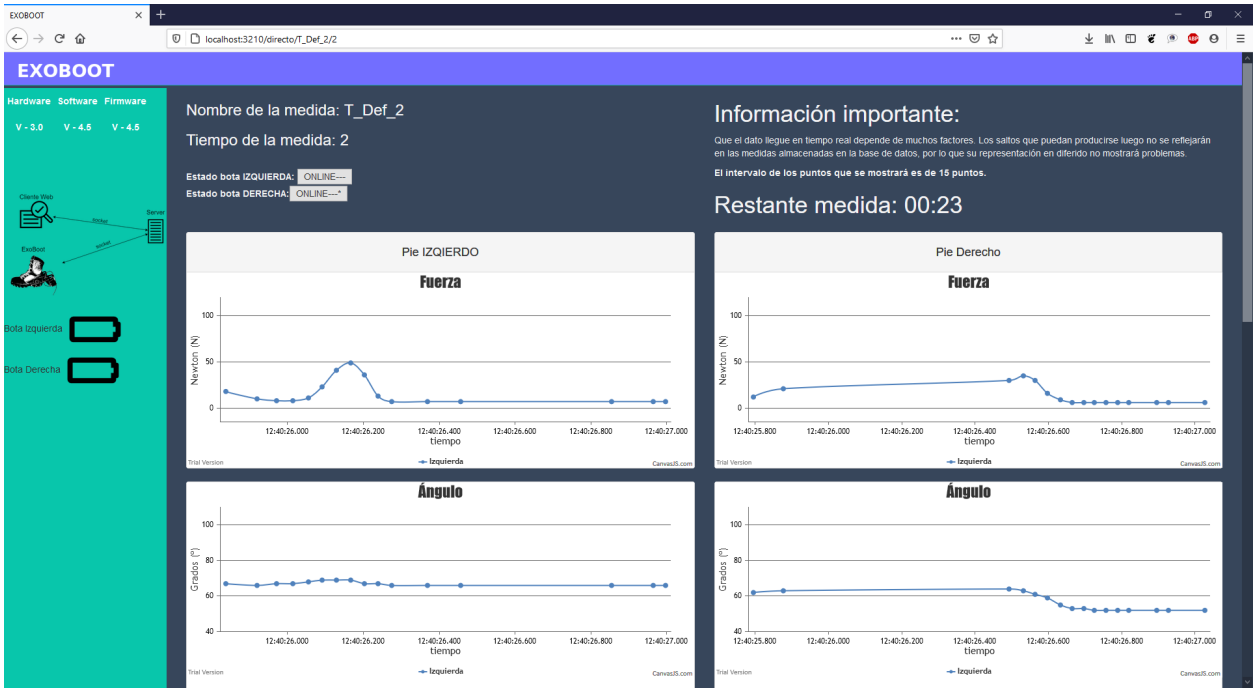


Figura 5.6 – *Gráficas de medida en tiempo real.*

CAPÍTULO

6

CONCLUSIONES Y MEJORAS DE FUTURO

Se muestra, en conclusión, la recopilación de diferentes aspectos del trabajo que se ha llevado a cabo en colaboración con GranaSAT, desde los requisitos iniciales para empezar a trabajar, pasando por el análisis de estos, hasta un diseño completo de la solución requerida. Dando solución al acuerdo de GranaSAT con el Ministerio de Defensa, se ha realizado un trabajo completo de programación, totalmente orientado al producto final, teniendo en cuenta todas sus necesidades y especificaciones, para lo cual se ha realizado un profundo estudio de requisitos y posibilidades, determinando cuales eran alcanzables.

Dicha filosofía de trabajo centrada en el producto final que se desea realizar partiendo de cero, ofrece la posibilidad de realizar este Trabajo Fin de Grado, de tal forma que se aproxime más a un trabajo real, que podría realizarse para cualquier empresa, más que como un mero trabajo académico, lo cual ha supuesto un enorme desafío a nivel personal, así como un orgullo poder ver todo ese trabajo aquí reunido.

El resultado final mostrado a lo largo de este documento, no ha sido para nada sencillo, ya que tras él quedan cientos de horas de pruebas erróneas, solventadas a base de estudio e investigación, gracias a lo cual se han adquirido otros conocimientos, que no hubiesen sido posibles sin la realización de enfrentarse a un reto de esta magnitud.

Uno de los principales retos que se plantearon vino de la mano del chip ESP32, ya que se trata de un microcontrolador que ofrece unos recursos muy limitados. Debido a que nunca había realizado anteriormente una programación de este tipo de placa a lo largo del Grado de

Ingeniería Informática, ha supuesto un gran aprendizaje de nuevos modos de programación, para conseguir elaborar este sistema en tiempo real.

Parte de dichos conocimientos han sido adquiridos gracias al mundo de la electrónica, así como del Arduino, pasando por nuevas técnicas de desarrollo para otros lenguajes como NodeJS.

Que se de por finalizado aquí este proyecto no significa ni mucho menos que se de por finalizado definitivamente, sino que se seguirá trabajando para su mejora, ya que ningún sistema es perfecto, más aún si tenemos en cuenta los avances tecnológicos que se dan cada día en la sociedad, por lo que siempre se puede evolucionar un sistema que se acabe de generar, partiendo tanto de aspectos hardware como de software.

Por ello se está pensando en realizar las siguientes mejoras futuras:

- Posibilidad de actualización del firmware vía OTA, lo cual requiere la modificación hardware que permita entrar en modo de bootable.
- Corrección de errores hardware, para que permita la lectura de todos los sensores disponibles.
- Crear un servicio web, permitiendo la descarga de la app y el registro de todas las medidas sin necesidad de hacer uso del paquete portable.
- Mejoras estéticas relacionadas con el *frontend*.
- Mejoras tanto de envío, como de recepción de paquetes.
- Posibilitar el uso de una tarjeta SD, como respaldo de datos y así poder realizar medidas a mayor distancia.
- Visualización desde otros dispositivos.

Es por ello que se pretende seguir trabajando en este proyecto, para un mayor desarrollo a nivel personal, ya que como se ha citado anteriormente, no se pretende que quede como un mero trámite académico, sino que se trata de un gran logro a nivel personal.

APÉNDICE

A

PRIMEROS PASOS ESP32

A.1 Introducción

En el siguiente apartado se explica de forma detallada y grafica el material necesario, así como los pasos a seguir para poder reproducir el proyecto, así como para instalar y acomodar el entorno de desarrollo.

Los pasos se pueden seguir tanto para Windows como Linux.

A.2 Primeros pasos

Que el ordenador detecte el ESP32 simplemente hay que conectarlo por USB y lo reconocerá, si es Windows 10 istalará solo lo necesario para el controlador, al igual ocurre con linux, y más concretamente Ubunutu 18.

Para empezar, lo principal es descargar el IDE [2] de Arduino e instalarlo, así como instalar las librerías del ESP32 para poder desarrollar.

Una vez instalado se debe configurar para añadir dicha tarjeta al editor. Archivo > Preferencias y añadir `https://dl.espressif.com/dl/package_esp32_index.json` en Gestor de URLs Adicionales de Tarjetas. Después pulsar OK.

1

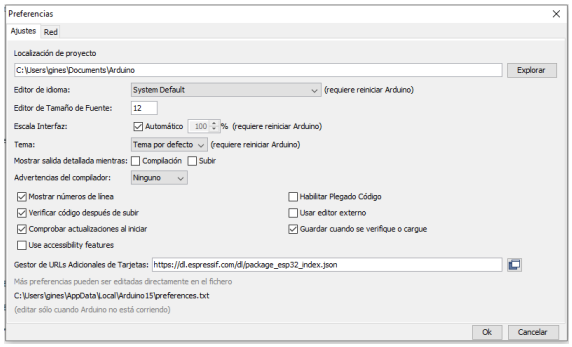


Figura A.1 – *Añadir tarjeta ESP32*

Posteriormente se ha de instalar, ir a Herramientas > Placa > Gestor de Tarjetas , buscar “ esp32 ” y pulsar en instalar

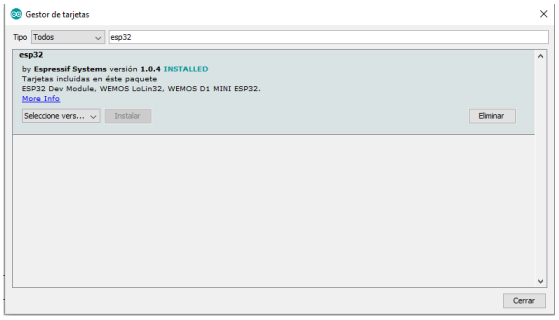


Figura A.2 – *Instalar librerías necesarias para tarjeta ESP32*

Una vez completado todo ya se puede probar la instalación. Se debe seleccionar la placa, eso se hace en Herramientas > Placa y se busca el modelo, en este caso la DOIT ESP32 DEVKIT V1. Después seleccionar el puerto que corresponda en Herramientas > Puerto > PUERTO y subir el código.

El siguiente ejemplo enciende el led azul del ESP32.

```
1 void setup() {
2 pinMode(2, OUTPUT);
3 }
4 void loop() {
5 digitalWrite(2, HIGH);
6 delay(1000);
7 digitalWrite(2, LOW);
8 delay(1000);
9 }
```

Código A.1 – *Ejemplo encender y apagar luz placa.*

REFERENCIAS

[1] Arduino. Arduinojson assistant. https://arduinojson.org/v6/assistant/.

[2] Arduino. Descarga ide de arduino. https://www.arduino.cc/en/Main/Software.

[3] Au-Yeung, J. Best practices for structuring express apps, may 2020. https://medium.com/swlh/best-practices-for-structuring-express-apps-1b3f0b7c9be5.

[4] Barry, R. *Using The FreeRTOS Real Time Kernel*. A practical Guide, 2009.

[5] Chatriot, L. The javascript database. https://github.com/louischatriot/nedb.

[6] Chung, J., Heimgartner, R., O'Neill, C., Phipps, N., and Walsh, C. Exoboot, a soft inflatable robotic boot to assist ankle during walking: Design, characterization and preliminary tests. In *7th IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics (BIOROB)* (2018).

[7] Collins, Steven H.and Wiggin, M. B., and Sawicki, G. S. Reducing the energy cost of human walking using an unpowered exoskeleton. *Nature 522*, 7555 (Jun 2015), 212–215.

[8] CSS, F. https://www.free-css.com/free-css-templates.

[9] de defensa, M. Prototipo de exoesqueleto de bota para aliviar la carga de los soldados, 2019-07-19. https://www.defensa.gob.es/comun/slider/2019/07/190719-exoesqueleto-bota-et.html.

[10] de Luis LLamas, P. Comparativa esp8266 frente a esp32, los soc de espressif para iot. https://www.luisllamas.es/comparativa-esp8266-esp32/.

[11] Espressif. Esp32 unity.h. https://github.com/espressif/arduino-esp32/blob/master/tools/sdk/include/json/tests/unity/src/unity.h.

[12] Goldberg, Y. Node.js best practices, 2020. https://github.com/goldbergyoni/nodebestpractices.

[13] I. Fette, A. M. The websocket protocol, Dec. 2011. https://tools.ietf.org/html/rfc6455.

[14] INC., M. Nosql vs sql databases. https://www.mongodb.com/nosql-explained/nosql-vs-sql.

[15] Mozilla. Métodos de petición http. https://developer.mozilla.org/es/docs/Web/HTTP/Methods.

[16] Mozilla. *Promise*. https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Promise.

[17] NPM. About npm. https://docs.npmjs.com/about-npm.

[18] OpenJSFundation. About nodejs. https://nodejs.org/en/about/.

[19] OpenJSFundation. Expressjs. https://expressjs.com/.

[20] OpenJSFundation. Generador de aplicaciones express. https://expressjs.com/es/starter/generator.html.

[21] PlatformIO. https://platformio.org/.

[22] Pug. https://github.com/pugjs/pug.

[23] Socket.IO. https://socket.io/.

[24] Wikipedia. Estándar de envío de datos ccsds (consultative committee for space data systems), Oct. 2017. https://es.wikipedia.org/wiki/ESP32#/media/Archivo:Espressif_ESP32_Chip_Function_Block_Diagram.svg.

[25] Wikipedia. Modelo–vista–controlador, nov 2020. https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador.

[26] Zeit. pkg. https://www.npmjs.com/package/pkg.