

TRABAJO DE FIN DE GRADO EN INGENIERÍA TELEMÁTICA



ARDUINO SIM DATA ACQUISITION SYSTEM

RESUMEN

[Arduino Sim Data Acquisition System](#) es un proyecto que cubre toda la parte de adquisición de datos de sensores desde el circuito físico, a la transmisión de los datos, su almacenaje, y su visualización en tiempo real desde un navegador Web.

Para su desarrollo se han utilizado muchos recursos de ingeniería de diferentes asignaturas recibidas en el grado. Entre ellas [Circuitos electrónicos](#), [Sistemas electrónicos digitales II](#) y [Programación](#) para la parte del circuito y del microcontrolador que conforman la estación.

Para la parte del servidor han sido utilizados diferentes recursos de las asignaturas de [Desarrollo de aplicaciones web](#), [Bases de datos](#), [Ampliación de informática](#) e [Ingeniería del Software](#).

Adicionalmente se han utilizado recursos con conocimientos adquiridos de forma autónoma para aplicar tecnologías y metodologías novedosas.

El proyecto lo conforman lenguajes como [C](#), [PHP](#), [Javascript](#), [Sql](#), [Json](#), [Html](#) y [Css](#); librerías como [Jquery](#), [Ajax](#), [Bootstrap](#), [Lodash](#), [Leaflet](#) y [Adminlte](#); Herramientas de desarrollo y diseño como [Visual Paradigm](#), [Arduino IDE](#), [Netbeans](#), [Blender](#), [KiCad](#); Chips como [Atmel](#) y [SimCom](#).

ABSTRACT

[Arduino Sim Data Acquisition System](#) is a project which covers all the data acquisition part of sensors starting from the physical circuit followed by the data transmission, its storage and visualization in real time from a Web Browser.

For its development there were used a lot of Engineering resources from different subjects assisted during the Grade. Among them [Electronic circuits](#), [Digital Electronic Systems II](#) and [Programming](#) for the circuit and microcontroller part that build the station.

For the server part there were used different resources from subjects like [Web Application Development](#), [Databases](#), [Computing Extension](#) and [Software Engineering](#).

Additionally there were used resources from an autodidact source for applying the newest technologies and methods.

The project is built by languages like [C](#), [PHP](#), [Javascript](#), [Sql](#), [Json](#), [Html](#) y [Css](#); libraries like [Jquery](#), [Ajax](#), [Bootstrap](#), [Lodash](#), [Leaflet](#) and [Adminlte](#); Development and design tools like [Visual Paradigm](#), [Arduino IDE](#), [Netbeans](#), [Blender](#), [KiCad](#); Chips like [Atmel](#) y [SimCom](#).

ÍNDICE

1.	INTRODUCCIÓN	6
1.1.	MOTIVACIÓN	7
1.2.	OBJETIVOS	7
1.3.	ESTRUCTURA DEL DOCUMENTO	8
1.4.	GLOSARIO DE TÉRMINOS	9
1.5.	REQUISITOS	9
1.5.1.	REQUISITOS FUNCIONALES	9
1.5.2.	REQUISITOS NO FUNCIONALES	9
1.5.3.	REQUISITOS OPCIONALES	10
1.6.	APLICACIONES PRÁCTICAS	10
2.	PLANIFICACIÓN	11
3.	ESTADO DEL ARTE	12
4.	ALTERNATIVAS DE DISEÑO	14
4.1.	ESTACIÓN	14
4.1.1.	DISPOSITIVO CONTROLADOR	14
4.1.1.1.	MICROCONTROLADOR	14
4.1.1.2.	RASPBERRYPI/MINI ORDENADOR	15
4.1.1.3.	SMARTPHONE	15
4.1.2.	TRANSMISIÓN DE LA COMUNICACIÓN	15
4.1.2.1.	COMUNICACIÓN MÓVIL	16
4.1.2.2.	COMUNICACIÓN SATELITAL	16
4.2.	SERVIDOR	18
4.2.1.	BASE DE DATOS	18
4.2.1.1.	MARIADB	19
4.2.1.2.	SQLITE	19
4.2.1.3.	JSON	19
4.2.1.4.	POSTGRESQL	20
4.2.2.	CÓDIGO WEB	20
4.2.3.	SOFTWARE/ENTORNO DE DESARROLLO	20
4.2.4.	SERVIDOR WEB	22
4.2.5.	SISTEMAS DE GESTIÓN DE CONTENIDO	23
4.2.6.	FRAMEWORKS PHP	24
5.	ELECCIÓN DE DISEÑO FINAL	26
5.1.	ESTACIÓN	26
5.1.1.	TRANSMISIÓN DE LA INFORMACIÓN	26
5.1.2.	DISPOSITIVOS	28
5.1.3.	COSTE TOTAL	29
5.1.4.	CONSUMO ENERGÉTICO	30

5.1.5.	ESQUEMA DE LOS DISPOSITIVOS DE LA ESTACIÓN.....	31
5.1.6.	DISPOSITIVOS EN DETALLE	32
5.1.7.	MODULARIDAD DEL DISEÑO	39
5.1.8.	SOFTWARE/ENTORNO DE DESARROLLO	41
5.1.9.	LIBRERÍAS USADAS	41
5.2.	SERVIDOR.....	43
5.2.1.	BASE DE DATOS	43
5.2.2.	SERVIDOR WEB	44
5.2.3.	SOFTWARE/ENTORNO DE DESARROLLO	44
5.2.4.	LIBRERÍAS Y FRAMEWORKS	44
5.2.4.1.	PHP	44
5.2.4.2.	CSS.....	45
5.2.4.3.	JAVASCRIPT	45
6.	CÓDIGO	46
6.1.	ESTACIÓN	46
6.1.1.	ESTRUCTURA DEL CÓDIGO	46
6.1.2.	VARIABLES DE CONFIGURACIÓN	48
6.1.3.	FLUJO DE EJECUCIÓN	49
6.2.	SERVIDOR.....	50
6.2.1.	INTERFAZ	50
6.2.2.	PRÁCTICAS USADAS.....	52
6.2.3.	ESTRUCTURA DEL CÓDIGO	55
6.2.4.	VARIABLES DE CONFIGURACIÓN	56
6.2.5.	FLUJO DE EJECUCIÓN	57
7.	CIRCUITO	59
7.1.	DISEÑO	59
7.1.1.	ESQUEMA DEL CIRCUITO	59
7.1.2.	PCB DEL CIRCUITO	60
7.2.	ENSAMBLADO Y MONTAJE	62
8.	PRUEBAS	66
9.	MEJORAS	71
10.	BIBLIOGRAFÍA.....	73

1. INTRODUCCIÓN

Cada día que pasa el humano tiene más y más sed de información. Información que posteriormente utiliza para analizar procesos y sacar conclusiones, descubrimientos, fallos, predicciones, o estadísticas.

En 2012 fue confirmada una partícula de la que se teorizaba desde los años 60: el [bosón de Higgs](#). Partícula que se encarga de dar sentido a la masa de los cuerpos. Para ello hubo que recabar datos de millones de millones de colisiones de partículas, ya que el higgs podría aparecer en una de cada 10 millones de colisiones. Es un ejemplo claro de cómo recabando muchos datos realizamos descubrimientos a partir de los resultados.

En el año 1999 la [NASA](#) lanzó el satélite [TERRA](#) al espacio. En el 2002 se envió el satélite [AQUA](#) al espacio, y se consiguió cruzar la información de los satélites [TERRA](#) y [AQUA](#), ya que juntos crean una cobertura global de la tierra. A pesar de que sus finalidades principales no son las mismas, aunque comparten otras, un equipo de la [NASA \(JPL\)](#) propuso utilizar los datos de radiación térmica de los satélites para predecir la entrada en erupción de los volcanes.

Con dos pequeños ejemplos podemos observar la gran importancia de recabar información para su posterior análisis.

Paralelamente, cada día que pasa la tecnología baja de precio e introduce nuevas funciones que nos permiten crear nuevos aparatos que mejoran nuestra vida diaria ya sea ahorrándonos mucho tiempo en tareas o directamente solventándolas.

Y esa es la idea con la que se aborda el proyecto: sacando mucho de poco; fabricando algo con recursos limitados; creando un producto con múltiples prestaciones sin gastar mucho en él.

1.1. MOTIVACIÓN

Este proyecto inicialmente iba a ser una propuesta de proyecto concreta solicitada por una tercera empresa con fines marítimos.

La idea finalmente no salió adelante, pero mi interés por desarrollar un proyecto que permita la recepción de comunicación remota sin Wifi y a bajo coste lo mantuve.

Gracias al trabajo de fin de grado he podido desarrollar este tipo de proyecto.

Se ha cambiado la tecnología usada y el propósito final del proyecto para transformarlo en un sistema altamente modular que permita una gran variedad de aplicaciones prácticas.

1.2. OBJETIVOS

El propósito del proyecto es crear un dispositivo lo suficientemente flexible y una página web en el lado del servidor lo suficientemente abierta para que cualquiera con unos conocimientos básicos pueda transmitir informaciones de ubicación o sensores conectados desde el dispositivo al servidor y visualizarlo de manera sencilla y muy visual.

Para ello se desarrollarán una serie de tareas:

- Creación de una web dinámica valorando las necesidades del proyecto procurando buscar la mayor sencillez y optimización posible.
- Creación una base de datos escalable y que sea accedida de manera flexible
- Programación de un dispositivo que controle módulos de comunicación a través del puerto de serie, lea sensores, y cree colas de datos para prevenir errores de transmisión.
- Diseño de un sistema de autonomía que gestione una batería, un panel solar y un microcontrolador.
- Diseño del modo de cableado, soldado y estructuras necesarias para ensamblar el proyecto

1.3. ESTRUCTURA DEL DOCUMENTO

Para facilitar la lectura del documento, a continuación se detalla un resumen de su estructura:

- En este primer capítulo se resumen los detalles del proyecto, las terminologías que lo componen, sus requisitos y sus posibles aplicaciones prácticas.
- En el capítulo 2 se detalla la planificación seguida para el desarrollo del proyecto.
- En el tercer capítulo se compara este proyecto con otros similares
- En el cuarto capítulo se detalla de forma extendida cada posible alternativa de diseño tanto para el dispositivo estación, como para el servidor web y base de datos comparando tecnologías, hardware, librerías, y entornos de desarrollo.
- En el quinto capítulo se detallan exhaustivamente todas las tecnologías, librerías, hardware y entornos utilizados para el desarrollo final del proyecto.
- En el capítulo sexto se desarrollan los diagramas de planificación del proyecto, que engloban los diagramas de casos de uso, el formato extendido, los diagramas de clases y los diagramas de secuencia general del sistema, tanto para la estación como para el servidor.
- En el séptimo capítulo se define la estructura del código fuente, las variables de configuración para el usuario final y el flujo de ejecución del programa, tanto para la estación como para el servidor.
- En el octavo capítulo se describe el diseño realizado del circuito y su realización en formato PCB de forma digital, así como el ensamblado final del mismo.
- En el capítulo número 9 se detallan las pruebas realizadas.
- El capítulo número 10 trata sobre las posibles mejoras futuras del proyecto.
- El capítulo 11 contiene la bibliografía y links de información referenciada.
- Por último el anexo contiene los diagramas de secuencia, postergados a esta sección por su gran tamaño y poca carga de información esencial.

1.4. GLOSARIO DE TÉRMINOS

A partir de ahora se hablará de todo el sistema como "**ASDAS**" (Arduino Sim Data Acquisition System) para agrupar en un solo término la definición de todo el sistema.

Hablaremos de la "**estación**" como el dispositivo que se encargará de enviar los datos hacia el servidor. Y de "**servidor**" como el propio servidor que recibirá, almacenará y mostrará esos datos.

1.5. REQUISITOS

1.5.1. REQUISITOS FUNCIONALES

- Se requiere la construcción de una estación para la transmisión de información cada cierto tiempo.
- La estación no tiene conexión directa a internet por WiFi o cable, es necesaria su conexión a través de otro sistema inalámbrico de comunicaciones.
- La información irá a un servidor que recogerá los datos y los almacenará en un sistema de base de datos.
- La información que envía se compone de:
Identificador, fecha, hora, latitud, longitud, y salida de un sensor o sistema externo.
- En caso de error de transmisión, la información se almacenará en una cola y se enviará cuando le sea posible.

1.5.2. REQUISITOS NO FUNCIONALES

- La interfaz del cliente mostrará gráficamente la información por días y meses.
- La interfaz del cliente cargará la información a medida que se le pide, y preferiblemente por peticiones Ajax, para evitar la sobrecarga del servidor y disminuir el tiempo de espera en el cliente.

1.5.3. REQUISITOS OPCIONALES

- Comprobar que la petición que realiza cada estación es legítima con un campo de contraseña que tendrá almacenada tanto el servidor como la estación.
- Evitar ataques de inyección [SQL](#) y [XSS](#) para las peticiones que recibe el servidor.
- La información se enviará encriptada si el sistema lo permite.
- Se creará opcionalmente una interfaz adicional para el cliente cuyo objetivo es configurar los parámetros de visualización de los datos.

1.6. APLICACIONES PRÁCTICAS

El diseño que se ha realizado hará posible su uso para múltiples aplicaciones finales, desde su uso para recibir la humedad o nivel de agua de la tierra en un lugar concreto sin ningún tipo de mantenimiento para controlar el riego o el cauce de un canal, o una pequeña estación meteorológica, un sensor remoto que controle cierto tipo de gas, el análisis del ruido en el espectro radioeléctrico de una ciudad, un sistema localizador de almacenaje o vehículo...

APLICACIONES PRÁCTICAS	
CON GPS	SIN GPS
Estudio en movimiento de gas/ruido/luminosidad/velocidad/consumo con una sola estación	Estudio zonal de gas/ruido/luminosidad (en un lugar estático)
Tracking de mercancías en camiones o barcos	Aviso de nivel de agua en canales y acequias
Tracking antirrobo de todo tipo de vehículos	Control de riego
	Control de humedad
	Control de gases

Tabla 1.1: Posibles aplicaciones prácticas del proyecto

2. PLANIFICACIÓN

Para la planificación del proyecto se ha diseñado un diagrama Gantt con el cronograma de las tareas realizadas.

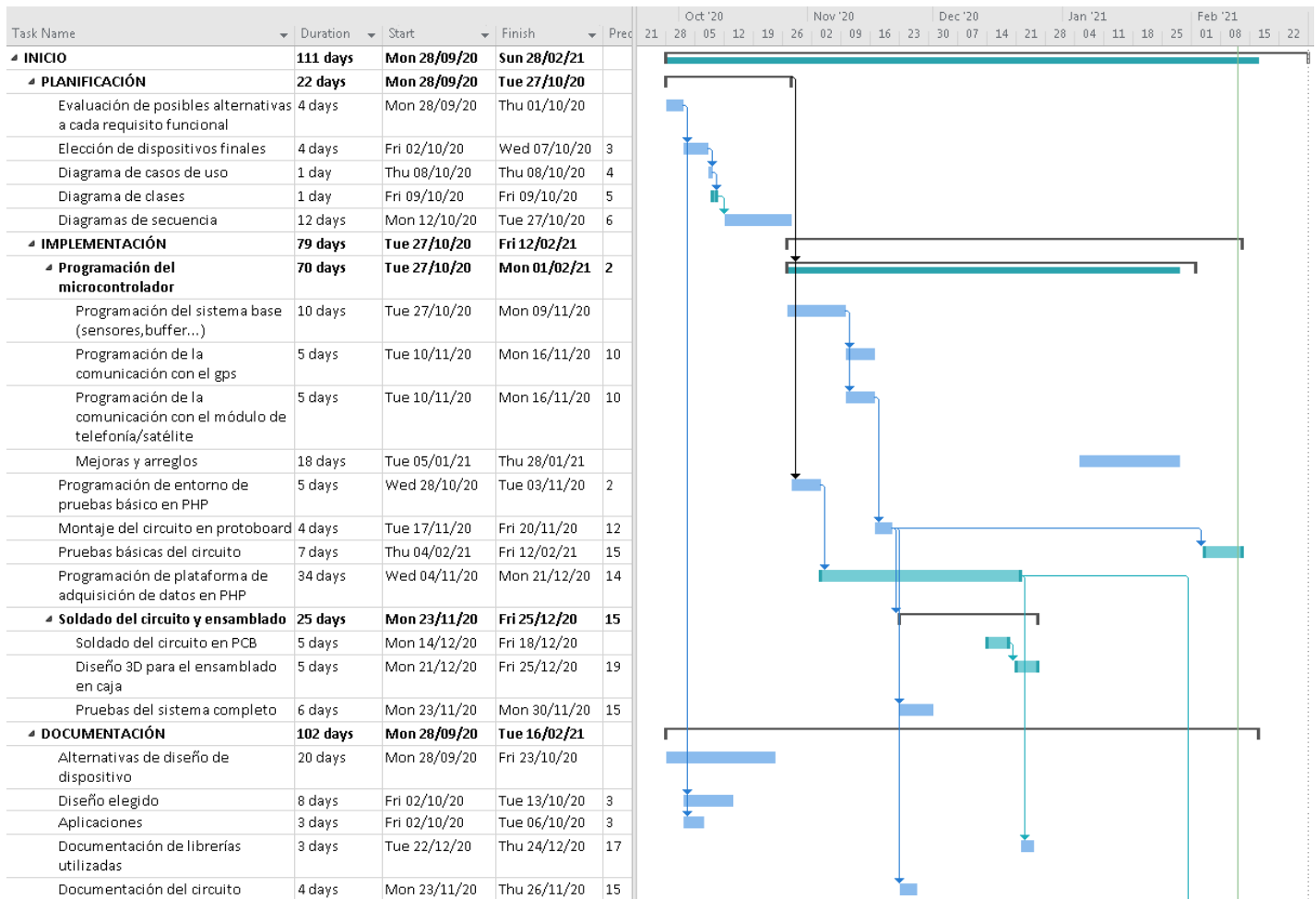


Figura 2.1: Diagrama de Gantt del proyecto

Se ha seguido una fase de evaluación de posibles alternativas a los requisitos funcionales, después se han realizado los esquemas y diagramas de casos de uso, clases y secuencias del sistema, y por último se han implementado tanto el código de la estación como el de la web del servidor siguiendo la metodología de control de versiones con [Git](https://github.com/laresistenciadelbit/asdas_station/commits/main) en [GitHub](https://github.com/laresistenciadelbit/asdas_station/commits/main).

https://github.com/laresistenciadelbit/asdas_station/commits/main

<https://github.com/laresistenciadelbit/asdas/commits/main>

3. ESTADO DEL ARTE

Tal como hemos visto en el apartado de aplicaciones prácticas, el proyecto puede ser aplicado para fines muy diferentes.

Ejemplos muy concretos de productos que transmiten y almacenan datos sin posibilidad de internet por cable o WiFi tenemos a [FLYGHT](#), la primera compañía canadiense de vuelos que implantó la transmisión de datos de las cajas negras de sus aviones por vía telefónica + satélite. O a [VESSELFINDER](#), que es un servicio que se dedica a ubicar en tiempo real buques marinos por vía satélite + radio terrestre. También tenemos por ejemplo a un sistema de riego inteligente ofrecido por la empresa [NUNSYS](#), que sirve para regar y analiza los datos del riego de forma automatizada, los datos de meteorología, consumo realizado, registro de anomalías del sistema, alertas programadas..

Más allá de los ejemplos de productos concretos, en este apartado trataré de detallar proyectos y publicaciones científicas que compartan al menos parcialmente la misma flexibilidad con éste, ya que el proyecto es lo suficientemente abierto como para que no hayan alternativas que cubran exactamente los mismos puntos.

Proyectos con cierta similitud son:

➤ [Real-Time 2G/3G/LTE Arduino GPS Tracker+IoT Dashboard^{\[1\]}](#)

Se trata de un proyecto en el que a partir de un Arduino y un módulo tipo Shield (encajado encima) que integra un sensor de temperatura y un chip de comunicación 4G y GPS, enviará información de la ubicación y del sensor de temperatura a un servidor cuya interfaz web mostrará Widgets en forma de gráficas y un mapa.

Éste proyecto es similar en cuanto a funcionalidad, pero no está pensado para un sistema de varios sensores. Además está diseñado para un módulo SIM integrado en una placa pensada para un tipo de Arduino de un tamaño grande (UNO, Mega, o Leonardo) y solo el módulo de comunicación tiene un coste de 65 dólares.

➤ [Sending data via SIM800L GRPS to Thingspeak^{\[24\]}](#)

Este proyecto detalla el código y pasos a seguir para crear un dispositivo que envíe los datos de un sensor de presión atmosférica a la plataforma ThingSpeak por medio de una API.

La plataforma ThingSpeak pertenece a MathWorks (los creadores de Matlab) y ofrecen un servicio gratuito para recibir datos y mostrarlos en un gráfico con un eje de tiempos con un límite de 4 canales por usuario y 3 millones de peticiones al API.

Es una buena alternativa si no nos importa una solución de terceros para almacenar y visualizar los datos, y no necesitamos escalar el servidor para dotarle de más funcionalidades.

Así mismo la solución del dispositivo no está lo suficientemente desarrollada como para ser usada de manera flexible con diversos sensores de forma nativa, y no dispone de GPS.

Publicaciones científicas sobre proyectos similares:

- Development of Internet of Things (IoT) based Data Acquisition system using Arduino Uno^[2]

Se trata de un paper del *International Research Journal of Engineering and Technology* que nos detalla información sobre un proyecto basado en Arduino que soporta pantalla LCD, comunicación por Wifi con el módulo ESP8266 y comunicación GSM con el módulo SIM900A. Adicionalmente de mostrar los datos en la pantalla LCD también los envía al servidor de thingspeak para la posterior visualización de los datos.

Al ser un paper no se puede valorar si ese proyecto ofrece flexibilidad a todo tipo de sensores de manera nativa, o solo muestra información sobre temperatura y humedad como refleja en el contenido.

- Internet of Things (IoT) based Data Acquisition system using Raspberry Pi^[3]

Se trata de un paper del *International Journal of Computer Science and Engineering* que nos enseña las posibilidades como sistema de adquisición de datos de un Raspberry Pi junto con un módulo de GSM, un módulo RTC, un adaptador Wifi y un convertidor ADC.

4. ALTERNATIVAS DE DISEÑO

Se han valorado diferentes alternativas dispares.

Desde la valoración del dispositivo que gestionará todo el proceso de almacenaje, lectura y envío de datos hasta la forma de comunicación que se va a utilizar.

4.1. ESTACIÓN

Para obtener los datos del sensor, almacenarlos, y enviarlos escogeremos el dispositivo controlador lo más optimizado para esta tarea que podamos encontrar.

4.1.1. DISPOSITIVO CONTROLADOR

Entre los dispositivos valorados, se encuentra un [microcontrolador](#), un [Raspberry Pi](#) y un [Smartphone](#).

4.1.1.1. MICROCONTROLADOR

Funcionamiento:

- Llevaría un programa hecho en c que se comunicaría por comandos [AT](#) con el módulo [GSM](#)

Pros:

- Tiene muchas librerías preparadas para todo tipo de sensores
- Tiene un consumo muy bajo de energía
- Es más barato
- Tiene librerías que facilitan el proceso de comunicación con módulos Sim

Contras:

- No tiene encriptación [GPG](#) nativa
- No tiene capacidad de almacenar muchos datos (requerirá módulo [SD](#))

4.1.1.2. RASPBERRYPI/MINI ORDENADOR

Funcionamiento:

- Llevaría un programa hecho en C++ ó Python que se comunicaría por comandos AT con el módulo GSM

Pros:

- Puede usar encriptación GPG de manera nativa

Contras:

- Opción más cara si le sumamos el módulo GPS/GSM
- Opción con alto consumo de potencia

4.1.1.3. SMARTPHONE

Funcionamiento:

- Llevaría una App hecha en java que se autoejecutaría al arrancar

Pros:

- Tiene internet, Gps, cámara y giroscopio/acelerómetro integrados

Contras:

- No está preparado para un entorno industrial
- No está preparado para añadirle más sensores

4.1.2. TRANSMISIÓN DE LA COMUNICACIÓN

Para la transmisión de información requeriremos de una tecnología que transmita información digital por el aire y la pueda enviar a un servidor alojado en la nube.

Descartando opciones de radio con rango limitado, nos quedan como alternativas viables la red móvil de datos y la comunicación satelital.

4.1.2.1.

COMUNICACIÓN MÓVIL

Desde 1992 contamos con redes móviles que permiten la transmisión de datos digital a partir de la tecnología **GSM**^[4] (Global System for Mobile Communications), también identificada como **2G**.

En ese año aparecieron los primeros teléfonos que soportaban esta tecnología y las redes europeas de **GSM** iniciaron su actividad en la banda de los 900Hz.

Es una tecnología muy extendida desde hace muchos años, los que la convierte en objetivo potencial del proyecto debido a su abaratamiento y cobertura.

Las alternativas **3G** y **4G** también son factibles, pero el coste de los módulos que lo soportan es mayor y la cobertura de estas redes en Europa es menor que la ofrecida por la red **2G**.

4.1.2.2.

COMUNICACIÓN SATELITAL

Como alternativa viable también podemos hacer uso de la comunicación satelital.

Esto es, que con ayuda de un módem o módulo podamos comunicarnos con un satélite para mandar y recibir datos.

Es una opción descartada por el alto coste que supone de tarifas, pero que nos brinda conexión con cobertura en cualquier lugar de la tierra.

Como opciones de comunicación satelital viables tenemos dos:

➤ IRIDIUM

Iridium, es un grupo de 66 satélites destinados a ofrecer transmisión de datos y voz desde cualquier parte del mundo.



Utiliza un protocolo de envío de mensajes cortos ([SBD](#)^[5]) entre el módulo y el servidor con el que se comunican los satélites.

Figura 4.1: Esquema de red satelital Iridium

Sus tarifas funcionan por créditos que equivalen a caracteres de mensaje enviado (bytes en su protocolo [SBD](#)).

Como dispositivos compatibles con las perspectivas del proyecto encontramos:

[Iridium9602](#)^[6], [Rockblock MK2](#)^[7], [Rockblock+](#) y [Rockblock 9603](#).

Las diferencias entre ellos son, en precios entre 100 y 300€, y en características principales el hecho de tener antena integrada, diferencia en el tamaño del Pcb, o tener el Pcb protegido dentro de una carcasa resistente al agua/golpes.



Figura 4.2: Módulo Iridium

El protocolo [SBD](#) de Iridium está limitado a la comunicación de mensajes simples entre el dispositivo y el servidor de Iridium, pero Iridium nos provee de apis^[8] para enviar mensajes por peticiones [POST](#) al dispositivo o recibir los mensajes del dispositivo, redireccionándolos del servidor a una dirección web de la que seamos propietarios, recibiendo allí las peticiones [POST](#).

Es posible el uso de esta tecnología tanto con dispositivos microcontroladores como con pequeños ordenadores como [Raspberry Pi](#).

➤ BGAN

[Broadband Global Area Network](#)^[9], es una red satelital que ofrece servicios de internet y llamadas.

En esta opción ya no nos tenemos que adaptar a un protocolo como el [SBD](#), sino que es un módem que actúa como Router en los dispositivos a los que lo conectemos ya sea por Wifi o por cable en base al dispositivo utilizado.

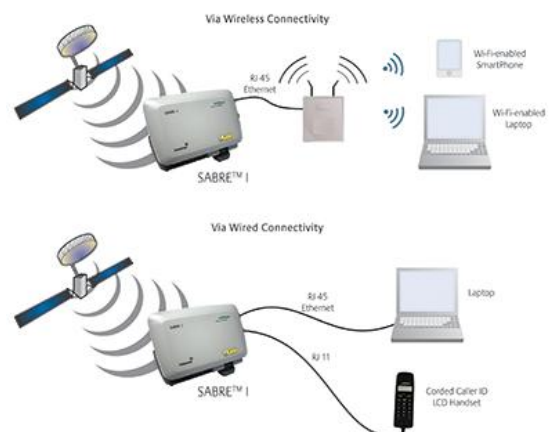


Figura 4.3: Esquema de red satelital BGAN

Como dispositivos de esta tecnología encontramos:
Hugues 9502, Explorer 510 BGAN, Hugues 92011.
Con precios rondando entre los 1300 y 2300€.

Con esta tecnología estamos forzados a usar un pequeño ordenador, o a usar un microcontrolador con funciones de red (Wifi/Ethernet).



Figura 4.4: Router BGAN

4.2. SERVIDOR

Para recibir los datos del sensor, almacenarlos y mostrarlos al usuario valoraremos las diferentes alternativas.

4.2.1. BASE DE DATOS

Para almacenar los datos en el servidor hay que decidir qué estrategia usar.

Las posibilidades son muy diversas; desde almacenar la información en texto plano, a almacenarla de manera no relacional en [JSON](#) o introducirla en una base de datos relacional de tipo [SQL](#) como [Oracle](#) o [Mysql](#).

Valoraremos opciones de base de datos relacional como [MariaDB](#), [Sqlite](#) obviando otras como [Oracle](#) y [Microsoft Sql Server](#) por no ser de código abierto y por estar limitadas a ciertos sistemas.

Adicionalmente valoraremos opciones de base de datos no relacional como [PostgreSQL](#) o [Json](#) en texto plano.

4.2.1.1. MARIADB

MariaDB es una base de datos relacional de código abierto ampliamente usada.

Su nombre original era **Mysql**, pero **Oracle** la compró y la convirtió en una alternativa de pago. Entonces el cofundador y buena parte de su equipo original continuaron desarrollando un Fork desde la última versión de código abierto de **Mysql** llamándola **MariaDB**.

Algunas de las características de **Mysql/MariaDB** más destacadas son la posibilidad de recuperaciones "roll-back" que permiten volver a un estado anterior de la base de datos, una gran eficiencia de memoria, y dispone de clientes muy potentes y con diversas herramientas para su administración.

4.2.1.2. SQLITE

Se trata de una base de datos de tipo relacional que no requiere ningún tipo de aplicación servidor para funcionar, tan solo necesita un motor que la interprete; pues es un simple fichero binario que puede ser leído por diferentes apis para los distintos lenguajes de programación.

Sus desventajas son la imposibilidad de escritura simultánea y la carencia de clientes elaborados para este tipo de base de datos.

4.2.1.3. JSON

Es una de las formas más simples de almacenar información.

Antiguamente se almacenaba en ficheros **CSV** donde los datos estaban separados por comas, y **JSON** desbancó ese formato debido al potencial que supone un formato que permite almacenar objetos de datos sin tener una estructura relacional.

A efectos prácticos no es una buena opción usar como base de datos un fichero **JSON** ya que se pierde mucha versatilidad, es inviable la escritura múltiple, y ocupa más espacio que un sistema que comprima los datos. Por eso su uso está más destinado a la transmisión de información o a ser almacenado en un sistema de base de datos junto con otros elementos **JSON**.

En cualquier caso sigue siendo una opción viable para un sistema muy simple en el que se almacenen los datos en un fichero **JSON** cada día, pero muy peligrosa si quisiéramos escalar el sistema.

4.2.1.4. POSTGRESQL

Es una base de datos no relacional que funciona como aplicación servidor y almacena datos de tipo [JSON](#).

4.2.2. CÓDIGO WEB

El lenguaje más utilizado para crear código web es PHP con gran diferencia.

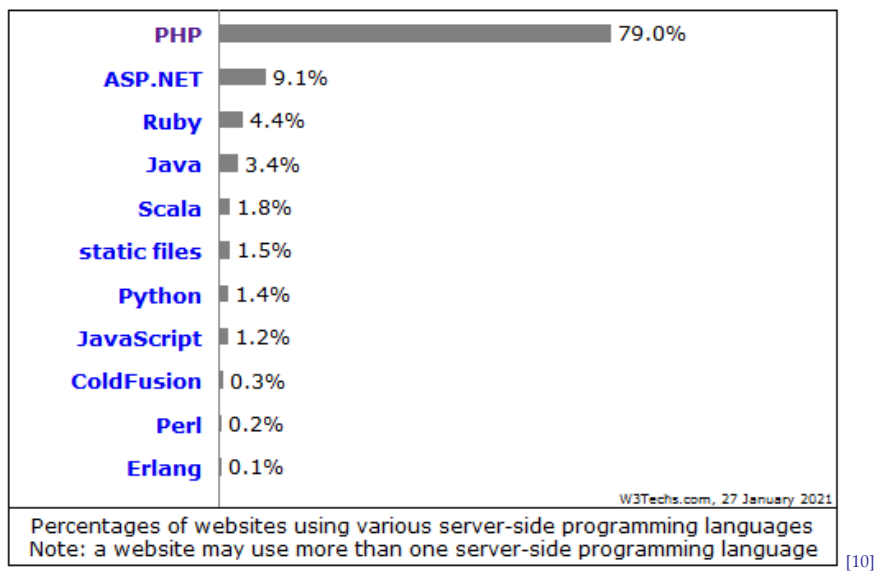


Figura 4.5: Porcentaje de uso de lenguajes de programación web

A pesar de que tenemos otras alternativas, no entraremos en detalle con ellas, ya que PHP nos brinda un motor muy completo y versátil para implementar páginas web dinámicas.

4.2.3. SOFTWARE/ENTORNO DE DESARROLLO

Para programar el código de la web en PHP tenemos múltiples herramientas de desarrollo.

Entre las más populares se encuentran [Eclipse](#), [Netbeans](#), [Notepad++](#), [IntelliJ](#).

Procederé a analizarlas brevemente:

➤ **Eclipse**

Antiguamente no soportaba PHP, ya que nació para java y ofreció soporte a c++. Actualmente cuenta no solo con soporte a PHP, sino además con debugger, refactorización de código, plantillas de código, extensiones que dan soporte a frameworks de terceros y muchas funcionalidades.

➤ **Netbeans**

Actualmente de Apache, el IDE [Netbeans](#) pensado para Java, lleva dando soporte a PHP prácticamente desde sus inicios.

Al igual que [Eclipse](#), ofrece refactorización de código, extensiones de soporte a frameworks de terceros, debugger...

➤ **Notepad++**

No es un entorno de desarrollo como tal, pero se puede utilizar o incluso convertir en algo muy similar gracias a algunos plugins como [npp_exec](#) que permiten ejecutar una consola internamente en la aplicación en la que podemos preconfigurar diversos compiladores, o [3P^{\[1\]}](#) que contiene toda clase de herramientas para sentirnos como en cualquier otro IDE (autocompletar código, ver funciones del código, ver árbol de directorios...)

Además de base ofrece muchas funcionalidades que no poseen otros IDE's, como el reemplazo de texto en múltiples ficheros a la vez y con expresiones regulares, modificar la codificación del fichero o el tipo de retorno de línea, comparar ficheros línea a línea, creación de macros...

➤ **IntelliJ**

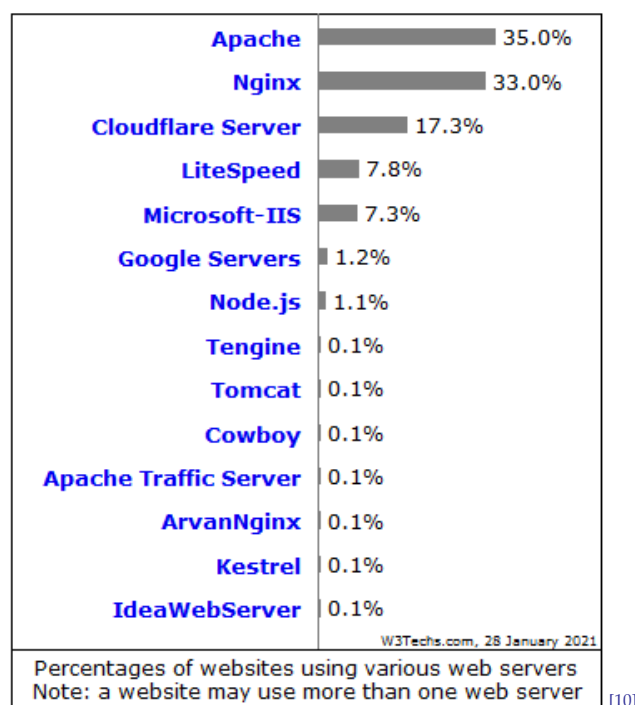
Nació como IDE para java y evolucionó para adaptarse a todo tipo de lenguajes. Es un entorno aparentemente muy simple pero con un potencial enorme. Dispone de características como autocompletado de código, búsquedas exhaustivas, conexión directa a base de datos y otras funciones de las que otros IDE's no disponen.

4.2.4. SERVIDOR WEB

La mayoría de servidores web dan soporte a PHP, y entre los más usados se encuentran [Apache](#), [Nginx](#) y [Microsoft ISS](#).

Otros han sido descartados como optativa por ser servicios en la nube o no dar soporte a PHP como [Node.js](#) que es un sistema hecho para programar en Javascript netamente.

Esta es la gráfica de los más usados actualmente:



[10]

Figura 4.6: Porcentaje de uso de servidores web

Entre las variantes decisivas como [Apache](#), [ISS](#) y [Nginx](#) cabe destacar que [ISS](#) es solo compatible con Microsoft Windows y tampoco representa grandes ventajas sobre los otros salvo mayor soporte a ASPX y .NET.

[Nginx](#) a efectos prácticos es más veloz que [Apache](#), y [Apache](#) tiene muchas extensiones y gran diversidad de configuraciones. Así que ambas son buenas elecciones.

4.2.5. SISTEMAS DE GESTIÓN DE CONTENIDO

Se ha valorado el uso de CMS's^[12] para la creación de la web debido a que éstos engloban toda la estructura y código necesario para crear con poco esfuerzo una web completa y funcional.

Entre ellos:



Figura 4.7: Varios CMS's

➤ **Wordpress**

Este es con diferencia el más usado. Principalmente por ser un simple blog con una comunidad enorme que fabrica cientos de miles de plugins y consiguen que puedas pasar de tener un simple blog a una web compleja de venta de productos con niveles de usuarios y un sin fin de posibilidades sin necesidad de programar nada.

Su desventaja es que no es modular de base y por defecto tiene mucho contenido integrado.

Tampoco dispone de routing, broadcasting u otros elementos que puede brindar un Framework moderno.

Y la seguridad de sus plugins es la confianza que el usuario final tenga de ellos.

➤ **Drupal**

Se trata de un CMS más simple y liviano que **Wordpress**, con una comunidad más pequeña, pero puede funcionar bien para hacer proyectos robustos.

➤ **OktoberCMS**

Este CMS está basado en el framework **Laravel**, es bastante simple y liviano, y ofrece diversos plugins y plantillas, así como funciones nativas de blog, autenticación, foro, páginas...

➤ **Lavalite**

Otro CMS basado en **Laravel** que integra funciones básicas de administración como usuarios, roles, administrador de ficheros y plantillas.

4.2.6. FRAMEWORKS PHP

Se ha valorado el uso de un framework PHP para seguir una estructura estandarizada en el proyecto, y para beneficiarse de las características adicionales que llevan.

Primero mostraré una tabla comparativa entre los frameworks más populares y usados, y después una comparativa en usar un framework o crear una estructura personal basada en el modelo vista controlador.

	CODEIGNITER	LARAVEL	SYNFONY	CAKEPHP
ORM (base de datos relacional abstraída)	No, pero fácilmente integrable (Doctrine, DataMapper,...)	Si	Si	Si
REQUISITOS DE USO		comando artisan	comando symfony	
FUNCIONALIDADES ADICIONALES		-Fácil integración con AWS (Amazon Web Service) -Autenticación de usuarios -Broadcasting		
WEB OFICIAL DE PLUGINS ADICIONALES	No, pero en su repositorio oficial hay varios plugins	No, pero hay webs que ofrecen varios plugins gratis	knpbundles.com	plugins.cakephp.org
PLANTILLAS	No, pero se puede implementar Twig o Blade	Blade	Twig	Sistema propio de plantillas

Tabla 4.1: Comparativa entre diferentes frameworks PHP

* Adicionalmente todos estos frameworks se basan en una estructura de modelo vista controlador, e incluyen clases que facilitan el manejo de sesiones, validación de datos (evitando inyecciones xss o sql), sistema de rutas para forzar como válidas solo ciertas rutas de acceso, cacheo de datos y manejo de errores, entre otras.

[*]Estructura web personal basada en MVC
[PRO]
-Simple y sencillo de usar, ya que su funcionalidad está limitada a nuestras necesidades
-No precarga librerías innecesarias ni requiere precompilación de ningún tipo
-Tiene una estructura personal de modelo-vista-controlador completamente moldeable
-Tiene mucha facilidad y flexibilidad para integrar cualquier tipo de librería o contenido
-No requiere de ningún tipo de herramienta o prerequisite para funcionar salvo un servidor web con PHP.
[CONTRA]
-Tienes que tratar los datos para evitar inyecciones de código
-Tienes que tratar los documentos y las rutas para evitar que se acceda a ficheros o directorios no deseados.
-Si requieres de características como hooks, broadcasts o excepciones tendrás que crear y adaptar un sistema propio de 0.

Tabla 4.2: Pros y contras del uso de una estructura personal PHP

[*]Framework basado en MVC
[PRO]
-Te obliga a utilizar prácticas muy restrictivas por seguridad (plantillas, routing...)
-Te brinda librerías integradas para el manejo de bases de datos o acceso de usuarios
-Te asegura que los datos recibidos no produzcan inyecciones sql o xss al tratar los inputs desde la plantilla.
[CONTRA]
-Te obliga a utilizar sintaxis muy diferente en el código que incluso a veces es sustituida con otros sistemas mejorados (como es el caso por ejemplo de Perfex usando un sistema de hooks propio a parte del integrado en el framework Codeigniter)
-Te brinda librerías o sistemas de plantillas convierten a zonas de tu código en pseudocódigo, acabando de esta manera con una mezcla de código PHP puro en algunos scripts y pseudocódigo en otros, que además limitan la funcionalidad.
-Te obliga a crear clases extendidas de otras clases y subclases en todos los aspectos (excepciones, base de datos, hookers, validación...) y después es necesario precompilarlo en muchos casos (como laravel)
-Es muy escalable pero solo para la versión que hayas elegido, pues en futuras versiones el código suele cambiar tanto que se vuelve una tarea extremadamente complicada la de migrar a una nueva versión de framework (esto pasa prácticamente con la totalidad de frameworks conocidos)
-Suele requerir muchos prerequisites para instalar y herramientas para ponerla en funcionamiento.

Tabla 4.3: Pros y contras del uso de un framework PHP

5. ELECCIÓN DE DISEÑO FINAL

5.1. ESTACIÓN

5.1.1. TRANSMISIÓN DE LA INFORMACIÓN

➤ Red de transmisión de datos:

Como elección final, se ha optado por la tecnología móvil **2G** debido al rango de cobertura ofrecida^[13]. Mientras que **3G** o **4G** en ciertos puntos no tenemos acceso a esas redes, la red **2G** está más extendida que el resto.



Figura 5.1: Mapa de cobertura 2G en España

Adicionalmente la red **2G** utiliza frecuencias ligeramente inferiores, lo que también ayuda a que sus ondas penetren más a través de paredes y superficies.

Frecuencias y bandas usadas en España

Cobertura	Bandas y Frecuencias
2G	B3 (1800), B8 (900)
3G	B1 (2100), B8 (900)
4G	B3 (1800), B7 (2600), B20 (800)
5G	B78 (3500)

Tabla 5.1: Bandas y frecuencias de cobertura en España

➤ **Compañía de comunicación:**

Se ha elegido la red **Simyo** para la creación del prototipo por tener una tarifa de datos de poca capacidad, que es suficiente para pequeñas transmisiones de datos (en nuestro caso una transmisión de un mensaje por minuto).

Su coste final es de 1€ al mes por 100Mbytes de datos.

Como alternativa para un diseño final sería buena opción la de tomar una compañía que ofrezca una conexión global para todo el planeta como:

- Thingsmobile: Tiene una tarifa flexible de 12€ por 100MB
- Ince: Tiene una tarifa de 10€ por 500MB
- Easym2m: Tiene una tarifa de 1€ por MB
- Hologram.io: Tiene una tarifa de 1.5 \$ al mes + 0.4 \$ por MB
- Olivia: Tiene una tarifa de 2.75\$ al mes + 0.05\$ por MB

➤ **Modo de transmisión de los datos:**

El protocolo de comunicaciones que usan los módulos de transmisión móvil funcionan a través de comunicación en serie por **comandos AT**^[14].

Se enviarán un conjunto de comandos AT de manera coordinada para configurar la conexión, conectarse a la red móvil, configurar la petición POST HTTP y enviarla al servidor destino.

5.1.2. DISPOSITIVOS

-Arduino pro micro 3.3v

[link] <https://www.aliexpress.com/item/32827452509.html>

[Función] Controlar sensores y módulo de comunicación

[Consumo] 1mA idle, 5mA en uso

[Precio] 4€



Figura 5.2: Arduino Pro Micro

-Módulo de comunicaciones sim800l

[link] <https://www.aliexpress.com/item/32831787649.html>

[Función] Nos provee de conexión 2G y GPS

[Consumo] 0.7mA dormido, 350mA de media, 2000mA de pico

[Precio] 8€



Figura 5.3: BK-SIM808

-Antena SIM + antena GPS

[link] <https://www.aliexpress.com/item/32811652115.html>

[función] Dar señal al módulo sim800

[precio] 2€



Figura 5.4: Antena SIM y GPS

-Sensor

[Función] gas/temperatura/humedad/movimiento...

[Consumo]

[Precio] 1-5€

-Lector tarjeta micro SD

[link] <https://www.aliexpress.com/item/32802684245.html>

[Función] Almacena los datos del sensor y posición GPS si
no le llega señal 2G.

[Consumo]

[Precio] ~1€



Figura 5.5: Módulo micro SD

-Batería de 3.7v

[link] <https://www.aliexpress.com/item/33017241715.html>

[especific.] 3.7v 1000mA ; 5x3.4cm

[Precio] ~3€

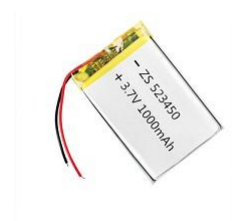


Figura 5.6: Batería de litio

-Step up+charger circuit

[link] <https://www.aliexpress.com/item/32998743006.html>

[Función] se encargará de cargar la batería y de alimentar el resto del circuito

[especific.] 4.5-8v in, 4.3-27v 2A out (ajustable) ; 3.3x2.3cm

[Precio] 1.4€

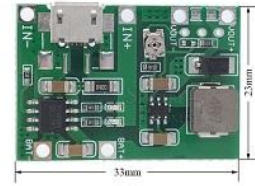


Figura 5.7: Circuito cargador

-Placa solar de 6v

[link] <https://es.aliexpress.com/item/32877897718.html>

[Función] cargará la batería a través del circuito cargador

[especific.] 6v 100mA ; 6x9cm

[Precio] 1.84€



Figura 5.8: Panel solar

-Caja a prueba de agua para el circuito

[link] <https://www.aliexpress.com/item/33013726416.html>

[especific.] (tamaño interno) 84x89 ó 109x46 , IP65

[Precio] 8.6€

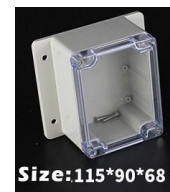


Figura 5.9: Caja IP65

-Placas Pcb de prototipo

[link] <https://www.aliexpress.com/item/10Pcs-high-quality-new-Prototype-Paper-Copper-PCB-Universal-Experiment-Matrix-Circuit-Board-5x7cm/32902801591.html>

[función] soldar las piezas y cables sobre las placas

[precio] ~1€



Figura 5.10: Placas para prototipo

-Cableado

[link] <https://www.aliexpress.com/item/33027953616.html>

[función] interconectar todos los componentes

[precio] ~3€



Figura 5.11: Cables para el circuito

5.1.3. COSTE TOTAL

Presupuesto final: 29€

*El coste es real y orientativo ya que es un coste de piezas de prototipo.

5.1.4. CONSUMO ENERGÉTICO

➤ CONSUMO TOTAL

Total (para 5v): ~375mA en uso , ~4mA en stand by.

Asumimos un consumo de uso de 5segundos por minuto, e idle de los 55s restantes:

8.3% de tiempo en uso, 91.7% de tiempo en idle:

$375 \times 0.083 = 31.125\text{mA}$ en uso + $4 \times 0.917 = 3.67\text{mA}$ en idle = 34.8 mA de consumo, que hace un total de $5 \times 0.0348\text{A} = 174\text{mW/h}$ de consumo estimado.

➤ BATERÍA

Para mantenerlo con una autonomía de 12 horas por la noche, se necesitaría una batería de $12 \times 34.8 = 417.6\text{mA} \times 5\text{v} = 2088\text{mW} / 3.7\text{v} = 564\text{mA}$ en batería de 3.7V

Dado que nunca debemos descargar la batería por completo y que la batería no tiene una descarga ideal, asumimos como mínimo el doble de la capacidad de batería (1000 miliamperios).

➤ PANEL SOLAR

Usaremos un panel que cargue el doble de lo que consume como mínimo:

$34.8\text{mA} \times 2 = \sim 70\text{mA}$ <- usaremos pues un panel de 6v y $\geq 0.4\text{W}$

*lo usaremos de 6v ya que el circuito cargador soporta entrada de 4.5 a 8v y sólo en condiciones favorables conseguiremos que el panel devuelva 6v.

5.1.5. ESQUEMA DE LOS DISPOSITIVOS DE LA ESTACIÓN

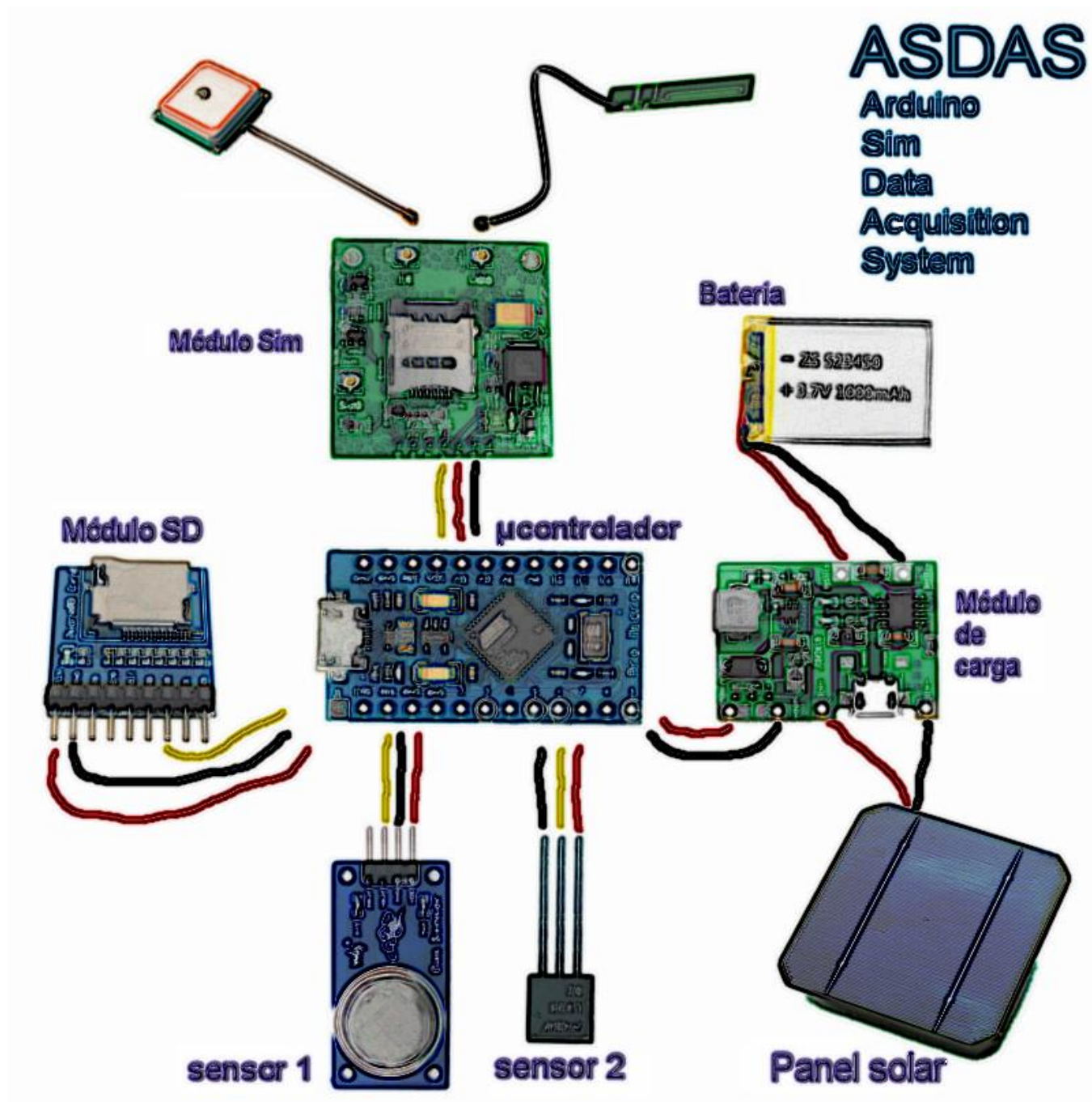


Figura 5.12: Esquema de los dispositivos de la estación

5.1.6. DISPOSITIVOS EN DETALLE

➤ Arduino pro micro 3.3v

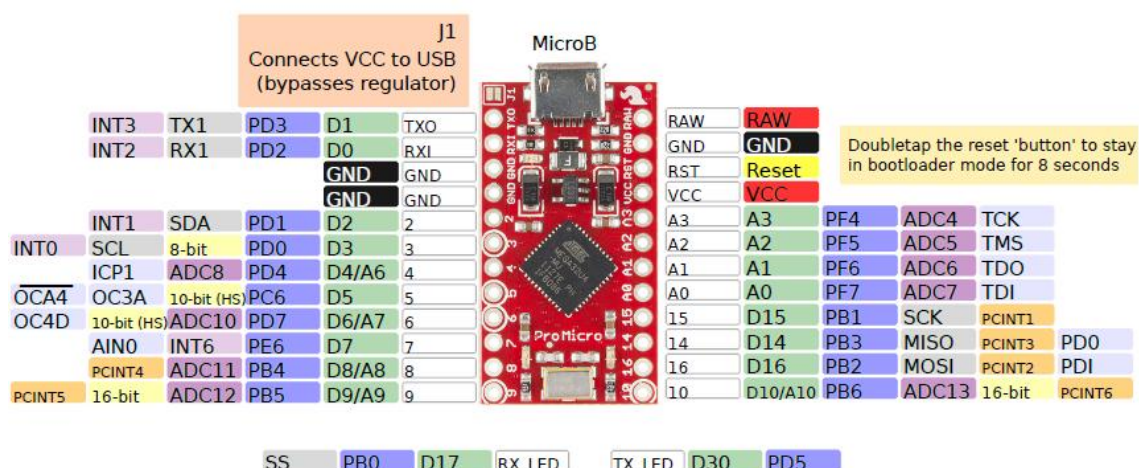
Se ha elegido este microcontrolador debido a su tamaño, a su bajo coste, y a su facilidad para compilar el código ya que no requiere un conversor ttl o un programador externo.

Tiene 18 pines digitales de entrada o salida que serán suficientes para nuestro objetivo.

De los cuales 9 pueden usarse como entrada analógica y 5 como salida pwm.

Soporta comunicaciones uart,i2c y spi que usaremos para conectar los diferentes módulos del proyecto.

Se eligió en su versión de 3.3v frente a la de 5v para evitar conversiones de voltaje, ya que la comunicación del módulo Sim se realiza a través de Uart y requiere 3.3 voltios por sus pines para muchos de los módulos de la familia sim800, así como la alimentación del módulo micro SD.



Power RAW: 4V-16V VCC: 3.3V at 500mA USB HID enabled VID: 0x1B4F PID: 0x9203 (bootloader); 0x9204 (sketch)	ATmega32U4 Built in USB 2.0 Absolute maximum VCC: 6V Maximum current for chip: 200mA Maximum current per pin: 40mA Recommended current per pin: 20mA 8-bit Atmel AVR Flash Program Memory: 32kB EEPROM: 1kB Internal SRAM 2.5kB ADC: 10-bit PWM: 8bit High Speed PWM with programmable resolution from 2-11 bits	LEDs Power: Red RX: Yellow TX: Green Serial Use Serial for the USB connection Use Serial1 for the hardware serial connection
--	---	--

Figura 5.13: Definición de pines y características del Arduino Pro Micro

Otros microcontroladores similares:



nano



micro



pro mini

Nano, micro, pro mini:

Otros micros que podrían haberse usado por similar tamaño, precio y soporte de librerías habrían sido los microcontroladores arduino nano, micro y pro mini.

Sus diferencias no son muy sustanciales (diferente cantidad de entradas/salidas, salidas pwm, forma de programarlo, botón de reset incluido, tamaño ligeramente distinto).

Figura 5.14: Alternativas de microcontrolador

Arduino MKR GSM 1400:



Es una alternativa que trae en la propia placa del microcontrolador el módulo de comunicación móvil integrado.

Su elevado coste y su falta de GPS lo descarta como alternativa.

Figura 5.15: Alternativa de microcontrolador con módulo de comunicación integrado

➤ Módulo de comunicaciones BK-SIM808

Se ha elegido este componente por su bajo coste y por cumplir los requerimientos para el proyecto: mandar peticiones POST a través de un módulo Sim y recoger datos de posicionamiento [GPS^{\[15\]}](#). Se alimentará directamente del conversor conectado a la batería y no a través del microcontrolador ya que tiene picos de consumo de hasta 2 amperios.



Figura 5.16: Módulo BK-SIM808

Se le ha añadido una antena más larga que la que traía por defecto para poder ensamblarla de manera que quede en la parte superior del ensamblaje.



Figura 5.17: Antena para módulo SIM



Tiene 7 pines en total, de los cuales 3 son de alimentación (5-10v y 2 de tierra), uno de control de encendido (key pin), uno de control de comunicaciones (DTR) y 2 de transmisión de las comunicaciones por comunicación en serie (TX y RX)

Figura 5.18: Parte trasera del BK-SIM808

Su chip principal (SIM808) de la empresa SimCom tiene las siguientes características:



General features <ul style="list-style-type: none">•Quad-band 850/900/1800/1900MHz•GPRS multi-slot class 12/10•GPRS mobile station class B•Compliant to GSM phase 2/2+<ul style="list-style-type: none">– Class 4 (2 W @ 850/900MHz)– Class 1 (1 W @ 1800/1900MHz)•Dimensions: 24*24*2.6mm•Weight: 3.3g•Control via AT commands (3GPP TS 27.007,27.005 and SIMCOM enhanced AT Commands)•Supply voltage range 3.4 ~ 4.4V•Low power consumption•Operation temperature:-40°C ~85°C	Software features <ul style="list-style-type: none">•0710 MUX protocol•Embedded TCP/UDP protocol•FTP/HTTP•MMS•POP3/SMTP•DTMF•Jamming Detection•Audio Record•SSL•Bluetooth 3.0 (optional)•TTS CN(optional)•Embedded AT (optional) Specification for GPS <ul style="list-style-type: none">•Receiver type<ul style="list-style-type: none">-22 tracking /66 acquisition -channel•GPS L1 C/A code•Sensitivity<ul style="list-style-type: none">-Tracking: -165 dBm-Cold starts : -148 dBm•Time-To-First-Fix<ul style="list-style-type: none">-Cold starts: 32s (typ.)-Hot starts: <1s•Warm starts: 3s•Accuracy<ul style="list-style-type: none">-Horizontal position : <2.5m CEP	Interfaces <ul style="list-style-type: none">•68 SMT pads including•Analog audio interface•PCM interface(optional)•SPI interface (optional)•RTC backup•Serial interface•USB interface•Interface to external SIM 3V/1.8V•Keypad interface•GPIO•ADC•GSM Antenna pad•Bluetooth Antenna pad•GPS Antenna pad Certifications <ul style="list-style-type: none">•CE•FCC
Specifications for GPRS Data <ul style="list-style-type: none">•GPRS class 12: max. 85.6 kbps (downlink/uplink)•PBCC support•Coding schemes CS 1, 2, 3, 4•PPP-stack•USSD Specifications for SMS via GSM/GPRS <ul style="list-style-type: none">•Point to point MO and MT•SMS cell broadcast•Text and PDU mode		

Figura 5.19: Características del chip interno del BK-SIM808 (SIM808)

De las cuales se hará especialmente uso de peticiones POST HTTP mediante el módulo Sim a través de la conexión GPRS y de la adquisición de datos del GPS.

Este tipo de módulos se comunica por comandos AT a través de una interfaz de serie, y son los que configuran e inician la Sim y la red de datos y realizan el proceso de envío de datos.

➤ Alternativas de módulo SIM:

Adicionalmente, y como alternativa de diseño, ya que el proyecto intenta ser lo más flexible y modular en lo posible, se ha adaptado al proyecto el uso de un dispositivo SIM sin GPS por si el proyecto no requiere esa funcionalidad.

Se trata del módulo SIM800L EVB.

Se ha elegido como alternativa de diseño ya que su alimentación es de 5 voltios como en el caso del módulo BK-SIM808.

Su precio es muy asequible y ronda los 3€



Figura 5.20: Módulo SIM800L EVB

Mencionar también, que como alternativa a otro módulo con GPS integrado, en vez del BK-SIM808, se puede usar de la misma manera el BK-SIM868 ya que sus pines son 100% compatibles con los del BK-SIM808.

La principal diferencia con respecto al BK-SIM808 es que soporta dos ranuras de SIM, y que soporta el uso del protocolo Bluetooth.



Figura 5.21: Módulo BK-SIM868

Otros módulos SIM:



Existen varias alternativas del chip SIM808 creadas por otros fabricantes que tienen un precio más elevado y ocupan más espacio; razón por la cual no fueron elegidos; pero nos brindan de mayor soporte a librerías, tienen componentes que lo hacen más estable, tienen salidas y entradas de audio, entrada de batería...

Sus precios rondan los 16€.

Figura 5.22: Módulos SIM alternativos

SIM800(x),SIM900:



por precio, simpleza, tamaño y soporte de librerías, es el más ideal, pero no tiene GPS incorporado.

Algunas variantes de [SIM800^{\[16\]}](#) tienen radio FM o Bluetooth incorporado. Su precio es de 2€.

Figura 5.23: Módulo SIM800L

SIM5360,SIM7100,SIM7600E:



Estas variantes incorporan desde 3G a 4G y llevan GPS integrado, pero son más caras, costando alrededor de 26, 45 y 35€ respectivamente.

Figura 5.24: Módulo SIM5360

-Lector tarjeta micro SD

Usaremos un lector genérico compatible con SPI para comunicarnos con él desde Arduino.

-Batería de 3.7v

Usaré una batería de litio genérica (3.7v) con una capacidad de 1 Amperio, ya que con el cálculo previo que realizamos, necesitaríamos una de 564mA, pero a eso hay que añadir que no haya pérdidas, que consiga cargarse completamente, y el consumo que también se hace mientras se carga. Por ello tomo una batería con el doble de capacidad de lo calculado.

-Step up+charger circuit

Se ha elegido este componente ya que integra un circuito cargador de baterías de litio y un circuito conversor de voltaje de salida regulable, lo que nos ahorra tiempo y trabajo de tener que diseñarlo nosotros mismos.

En su entrada estará conectado el panel solar (5.3) y en su salida regulada (5v) para el microcontrolador y módulo Sim.

-Placa solar de 6V nominales

Se ha elegido este voltaje debido a que entra en el rango de entrada que soporta el conversor para cargar la batería (4.5 a 8 voltios).

Hay que tener en cuenta que aun siendo de 6V hay que restarle 0.7v de un diodo rectificador que irá en serie para evitar corrientes en sentido inverso.




Se ha elegido en el tamaño necesario para que entrase en la carcasa de ensamblaje para la que hemos diseñado el proyecto.

-Caja a prueba de agua para el circuito

Se ha elegido este ensamblaje por su bajo coste y su tamaño ideal para el proyecto.

Es IP65 y como prototipo es suficiente, ya que protege del polvo, golpes, y agua sin demasiada presión.

Para un diseño final sería muy recomendable que cumpliese el estándar IP67, ya que protege mucho mejor contra el agua, incluyendo inmersiones temporales a poca profundidad.

0	No protection	No protection ("x")	0	No protection ("x")
1	Protected against contact with large areas of the body (e.g. back of hand)	Protected against solid objects up to 50mm ³	1	Protection against vertically falling drops of water (e.g. condensation)
2	Protected against contact with fingers	Protected against solid objects up to 12mm diameter	2	Protection against direct sprays of water up to 15 degrees from vertical
3	Protected against tools and wires over 2.5mm in diameter	Protected against solid objects up to 2.5mm diameter	3	Protection against direct sprays of water up to 60 degrees from vertical
4	Protected against tools and wires over 1mm in diameter	Protected against solid objects up to 1mm diameter	4	Protection against water sprayed from all directions, limited ingress permitted
5	Protected against tools and wires over 1mm in diameter	Protected against dust: limited ingress (i.e. no harmful deposit)	5	Protected against low-pressure jets of water from all directions, limited ingress permitted
6	Protected against tools and wires over 1mm in diameter	Totally protected against dust	6	Protected against strong jets of water from all directions, limited ingress permitted (e.g. a ship's deck)
Rating example:			7	Protected against the effect of "temporary" immersion between 15cm and 1m
			8	Protected against long period of immersion under pressure
			9	Protected against high pressure, high temperature jets of water from multiple directions*

IP

6

5

* IP69K is defined in DIN 40050-9

Tabla 5.1: Tipos de estándares de protección ante golpes y agua

Otros componentes

Para el diseño del circuito se han requerido otros elementos como placas para integrar los componentes necesarios, y cables para comunicar los diferentes módulos

5.1.7. MODULARIDAD DEL DISEÑO

Al ser un sistema con tantas aplicaciones prácticas y dispares, ha de ser muy modular.

Para ello, está pensado que el dispositivo cuente con o sin batería, GPS, sensores adicionales o varias estaciones.

De esta manera tenemos un dispositivo que tendrá una diversidad funcional muy grande.

Y no solo tendrá una alta modularidad en cuanto al uso de ubicación o autonomía, sino que también usaremos otros sistemas terceros como sensores.

Es decir, para crear un buen grado de modularidad, en vez de dificultar la lógica del código de la estación al utilizar sensores que requieran de librerías o cálculos adicionales, los leeremos a partir de un sistema adicional.

A parte de explotar esta ventaja, esto se hace también porque nuestro microcontrolador no tiene tanta memoria como para soportar librerías adicionales requeridas por otros sensores complejos. Así que es una buena excusa para convertir nuestro diseño en modular para casos complejos.

Para esta tarea crearemos canales de comunicación i2c para comunicarse entre la estación y otros sistemas adicionales, o si no requieren una resolución alta de datos usaremos directamente la lectura analógica (de 0-1023; 10bits de resolución).

Componentes modulares

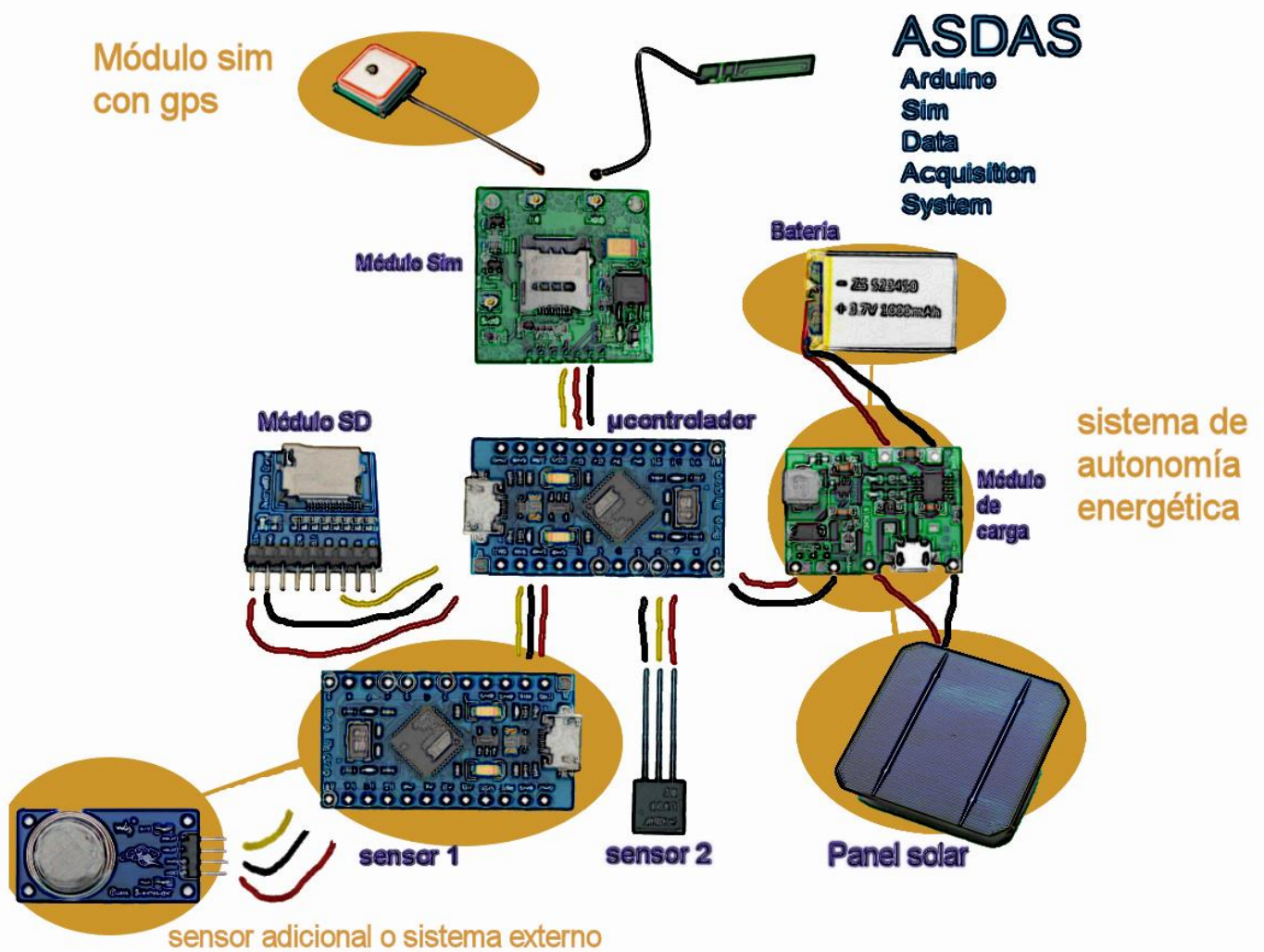


Figura 5.25: Esquema del diseño con los elementos modulares

5.1.8. SOFTWARE/ENTORNO DE DESARROLLO

Para la programación de la estación se ha usado [Arduino IDE](#) junto con [Notepad++](#) ya que vamos a usar un microcontrolador Arduino (modelo Pro Micro) y [Arduino IDE](#) integra librerías y herramientas de compilación.

Para la comunicación del módulo de GSM se podría implementar una clase de cero que hiciera todas las transferencias Uart por comandos, pero se ha hecho uso de la librería oficial de Sim808 para Arduino ya que facilita mucho todo el proceso de sincronización con la red celular, tiene bien estructurados e implementados los posibles códigos de error y tiempos de espera entre comandos y otras funciones adicionales que nos son útiles.

5.1.9. LIBRERÍAS USADAS

La primera fase del trabajo consistió en unir las librerías oficiales de los módulos [Sim808](#) y [Sim800L](#) con la idea principal del proyecto de realizar las tareas con una estructura modular tal que podamos cambiar uno por otro en base al uso que vayamos a hacer del proyecto.

Posteriormente se encontró una librería que ya unía las capacidades de diferentes módulos Sim en una ([TinyGSM](#)).

Es decir, soporte de peticiones GET y POST, y GPS para todos los módulos de la familia SIM8XX, SIM9XX y muchos otros, siempre y cuando lo soporten.

Finalmente se descartó debido a que usaba otras librerías adicionales para la creación de las peticiones web que aumentaban de forma desorbitada el tamaño final de compilación. Así que como alternativa se tomó la librería oficial de [Sim808](#)^[17] y se modificó a conveniencia.

-> **SIM808:**

Las características de la librería son:

- Control preciso sobre la gestión de energía del módulo
- Envío de sms's
- Envío de peticiones POST y GET por HTTP
- Recepción de posiciones GPS
- Lectura del estado del dispositivo (batería,gps,red...)

Esta librería se basa en la librería **FONA** que está preparada para controlar módulos concretos que integran varios chips SIM800.

Para hacer funcionar correctamente la librería se han debido de realizar algunos pequeños cambios.

Esto se debe a que la librería está preparada para módulos de otros fabricantes que usan otros pines del chip y actúan de forma diferente.

En el caso del módulo elegido (BK-808), hubo que modificar lo siguiente:

- Desactivar la espera al mensaje de respuesta de "RDY":

```
while (waitResponse(TO_F(TOKEN_RDY)) != 0);
```

- Inicializar la clase solo con los pines de reset y powerkey:

```
SIM808(SIM_RST, SIM_PWR);
```

- Modificar el baudrate del módulo Sim a 9600 baudios.

Adicionalmente se le han añadido dos funcionalidades:

getBattStat() : Nos devuelve el nivel de batería que tiene el módulo (calculado en base al voltaje que le llega)

getInternalClock() : Obtiene la fecha y hora del reloj interno del módulo SIM, que éste pide a la estación móvil GSM.

-> SDFAT (greiman):

Nos permite la gestión de lectura/escritura del módulo de tarjetas micro SD.

Esta no es la librería oficial, ya que la librería oficial ocupaba demasiado espacio y no era viable.

5.2. SERVIDOR

5.2.1. BASE DE DATOS

Tras dudarlo mucho, se ha optado por el uso elegible entre base de datos [MariaDB](#) o base de datos [Sqlite](#).

La razón principal de mantener ambas es que según lo grande y escalable que queramos que sea nuestro proyecto podamos cambiar de una a otra sin problemas.

[MariaDB^{\[18\]}](#) nos brinda una base de datos muy escalable, amplia, con posibilidad de escritura múltiple y facilidad de administración gracias a los clientes que dispone.

En cambio [Sqlite^{\[19\]}](#) no tiene esas ventajas, pero nos permite usar un sistema de base de datos sin necesidad de tener ningún servidor funcionando. Además es muy ágil.

Por defecto está configurado para usar [Sqlite](#) pero se puede cambiar sin ningún tipo de problema a [Mysql](#) con solo modificar la configuración del fichero inicial de la web (index.php).

Aunque se haya descartado el uso de ficheros [JSON](#) como gestión de datos en el servidor, se va a hacer uso de este formato para las peticiones web del registro de datos de las estaciones hacia el servidor.

5.2.2. SERVIDOR WEB

Finalmente se ha elegido el servidor web [Apache](#) como alternativa ya que es un estándar como servidor web e incluye [PHP](#) y además cuenta con muchos módulos integrados que nos facilitan la conexión con bases de datos o la restricción de acceso a ciertas urls.

Es necesario activar la extensión [Sqlite3](#) en el fichero php.ini para poder manejar bases de datos [Sqlite](#) desde [PHP](#).

5.2.3. SOFTWARE/ENTORNO DE DESARROLLO

Para la implementación de la web se ha usado tanto [Netbeans](#) como [Notepad++](#) como herramientas de trabajo.

[Netbeans](#) principalmente por tener un debugger incorporado, y [Notepad++](#) debido a la cantidad de recursos y herramientas que brinda al usuario.

5.2.4. LIBRERÍAS Y FRAMEWORKS

5.2.4.1. PHP

Se ha optado por crear un propio sistema de plantillas basado en el modelo vista controlador, ya que esto nos brinda una flexibilidad y rapidez de implementación que no nos ofrece otro framework.

Más adelante analizaré el diseño de la estructura de la web en detalle.

5.2.4.2. CSS

Se ha usado el framework css [Bootstrap](#) con la plantilla [Adminlte](#) que nos brinda de una plantilla de diseño muy personalizable, ya que tiene cientos de clases css que podemos aplicar en muchos contenedores de HTML y nos los transforma en una versión mucho más visual de esos elementos.

5.2.4.3. JAVASCRIPT

El framework Bootstrap junto con la plantilla utilizada (Adminlte) incluyen varias librerías para la mejora visual de la interfaz y para el control de ciertos elementos.

Entre ellas usaremos:

- [ChartJS](#): Para formatear gráficas
- [Datepicker](#): Para mostrar un calendario y elegir fechas
- [Bootstrap](#): Para la interacción del usuario
- [Jquery](#): Para el manejo de elementos de Javascript

Otras librerías adicionales añadidas:

- [Moment.js](#): Permite un manejo mucho más elaborado de las fechas
- [Lodash](#): Nos permite filtrar o agrupar datos en Javascript
- [Leaflet](#): Nos permite cargar y manipular mapas de Openstreetmap

6. CÓDIGO

En este bloque se hablará sobre cómo ha sido diseñado, estructurado y realizado el código en la parte de la estación y del servidor.

6.1. ESTACIÓN

La parte del cliente la conforma el circuito del microcontrolador, el módulo Sim de comunicación, el módulo micro SD de almacén de datos y los sensores adicionales conectados.

El microcontrolador será el cerebro de todo el sistema, y se encargará de dirigir y controlar el resto de componentes.

6.1.1. ESTRUCTURA DEL CÓDIGO

El código se define en 3 ficheros principales y el resto forma parte de las librerías. De las cuales, la librería que controla la comunicación con el módulo Sim ha sido modificada para admitir más funciones adicionales.

-> **ASDAS.ino**

Contiene la lógica del programa, cuenta con las funciones originales `setup()` y `loop()`.

`setup()` se encarga de inicializar los pines y otras configuraciones y `loop()` es la función principal del programa.

Adicionalmente se ha añadido la función `build_status_array()` para facilitar el trabajo al construir vectores.

-> **ModuleManager.cpp**

Contiene la clase que se encarga de administrar el módulo Sim, formatear los datos y enviarlos al servidor, así como almacenar los datos en la tarjeta SD si no pudieron enviarse al servidor.

-> **ModuleManager.h**

Contiene las cabeceras de la clase ModuleManager.

Algunas peculiaridades y características del código:

- Todas las variables de configuración son constantes globales ya que algunas obligatoriamente tenían que serlo ya que las clases no son dinámicas en Arduino, y otras para facilitar la configuración del usuario final.
- Hacemos uso del recurso `ifdef` en varias ocasiones para que el precompilador filtre las partes de código innecesarias según la configuración de variables.
- Reutilizamos el buffer de parámetros para mandar los datos al servidor y para recibirlos ya que la memoria de la que disponemos es muy limitada.
- Usamos siempre vectores de caracteres en vez de variables tipo `String` para evitar el uso de la librería `String` ya que no disponemos de mucha memoria en el chip.
- Por cada envío de lectura de los sensores se envía una petición adicional con los datos del Gps.
Inicialmente no era así, ya que se enviaba todo en la misma petición y ahorra tiempo y trabajo.
El problema de esto es que si la estación cuenta con más de un sensor, en cada envío de datos de cada sensor estaría enviando también los datos del Gps, y nos encontraríamos con una enorme redundancia de datos.
Así que se optó por enviar adicionalmente tramas con el "estado" de la estación. Donde llamamos estado no solo al Gps, sino a más condiciones opcionales asociadas a la lectura de los sensores en un instante.
Es decir, poder enviar a parte de los sensores el nivel de batería, la ubicación, los satélites Gps's detectados o cualquier propiedad que no tenga que ver con la lectura de sensores.

6.1.2. VARIABLES DE CONFIGURACIÓN

ID: Palabra que identifique al módulo (nombre de estación)

PASSWD: Contraseña que compartirá con las otras estaciones

(De esta manera evitamos que puedan generar peticiones falsas si no cuentan con la contraseña)

USE_GPS: Definimos si usará Gps o no (puede ser booleana)

SEND_STATUS: Definimos cuántas estadísticas se enviarán (debe ser numérica)

DEBUG_MODE: Envía más salida de datos por el puerto de serie (booleana)

GPRS_APN: Nombre del APN de conexión a la red móvil

GPRS_USER: Usuario del APN (ocasionalmente ninguno)

GPRS_PASS: Contraseña del APN (ocasionalmente ninguno)

PIN_RX: Pin RX de comunicación con el módulo SIM

PIN_TX: Pin TX de comunicación con el módulo SIM

PIN_PWKWY: Pin de energía del módulo Sim (algunos módulos lo usan)

PIN_RST: Pin de reset del módulo Sim (algunos módulos lo usan)

SENSOR_NUMBER: Número de sensores que tenemos conectados

sensor_pin[SENSOR_NUMBER]: Pines de los sensores conectados

sensor_name[][30]: Nombre identificativo de los sensores

PIN_OUTPUT: Definimos un pin de salida si queremos que se active tras un valor suficientemente alto del sensor principal.

SENSOR1_THRESHOLD: Valor de sensor máximo para que active el pin de salida.

OUTPUT_TIMEOUT: Definimos un tiempo en milisegundos tras el cual desactivará el pin de salida anteriormente activado.

SERVER_ADDRESS: Dirección de servidor a la que enviará las peticiones

STATUS_CHAR_LIMIT: Define un límite de caracteres para el envío de estado

WORKING_INTERVAL: Definimos cada cuantos segundos leerá y mandará la información de los sensores

SIM_TIMEOUT_THRESHOLD: Definimos cuántas veces leerá e intentará enviar la información antes de resetear el microcontrolador por incapacidad de conexión para que reinicie todos los módulos

6.1.3. FLUJO DE EJECUCIÓN

A pesar de quedar mayormente definido el flujo en los diagramas de secuencia voy a desarrollar de manera resumida la ejecución del programa a nivel de código.

Inicialmente se definen todas las constantes globales y la clase que administra todo el proceso de creación de las peticiones web, envío de datos, almacenaje en tarjeta micro SD... (`ModuleManager`), así como las funciones del fichero principal (`setup`, `loop` y `build_status_array`).

Después ejecuta la función `setup()` inicializando los pines de entrada y salida.

Entonces entra en el `loop()` donde inicializa la clase `ModuleManager`, y ésta al inicializarse (en su constructor) inicializa los módulos SD y Sim, y de éste último realiza varios intentos de conexión a la red móvil y después a la red de datos GPRS (ambas de manera recursiva).

Aquí también se inicializará el Gps si lo hemos indicado en la constante booleana global.

Si el módulo Sim se conectó correctamente a la red de datos móvil, entonces entra en un bucle del que no saldrá hasta que se hayan producido varios intentos erróneos de conexión.

Dentro de ese bucle se tomarán los valores de los sensores `ASDAS.get_sensor_val(pin)`, se obtendrá la hora actual obtenida de la red móvil `ASDAS.get_time()` y la ubicación del Gps si la hemos configurado `ASDAS.get_gps()`.

Enviaré los datos que tenga en la cola de la tarjeta SD al servidor (`ASDAS.send_last_sd_line()`) en caso de que se produjese un fallo de conexión previo.

Enviaré los datos recogidos de los sensores al servidor `ASDAS.send_data_to_server()` y en caso de error los almacenará en la tarjeta SD `ASDAS.save_data_in_sd()`.

Prepararé los datos de estado de la estación creando un array de nombres de estado y su valor con ayuda de la función `build_status_array()`, y enviaré los datos de estado al servidor `ASDAS.send_additional_data_to_server()` y en caso de fallar los almacenará en la tarjeta SD `ASDAS.save_additional_data_in_sd()`.

Finalmente esperará `WORKING_INTERVAL` segundos (el tiempo asignado de espera entre registro de lectura de los sensores).

Antes de finalizar verificará que haya conexión con la red de datos móvil, y si supera un umbral de veces sin conexión, reseteará el dispositivo para reiniciar las conexiones partiendo de cero.

6.2. SERVIDOR

En el lado del servidor tenemos una web en PHP personalizada basada en la arquitectura `MVC`^[20] (modelo vista controlador) con tildes de `MVVM` (modelo vista vista-modelo) ya que una vez cargada la vista, la lógica del modelo se procesa en la vista mediante Javascript.

6.2.1. INTERFAZ

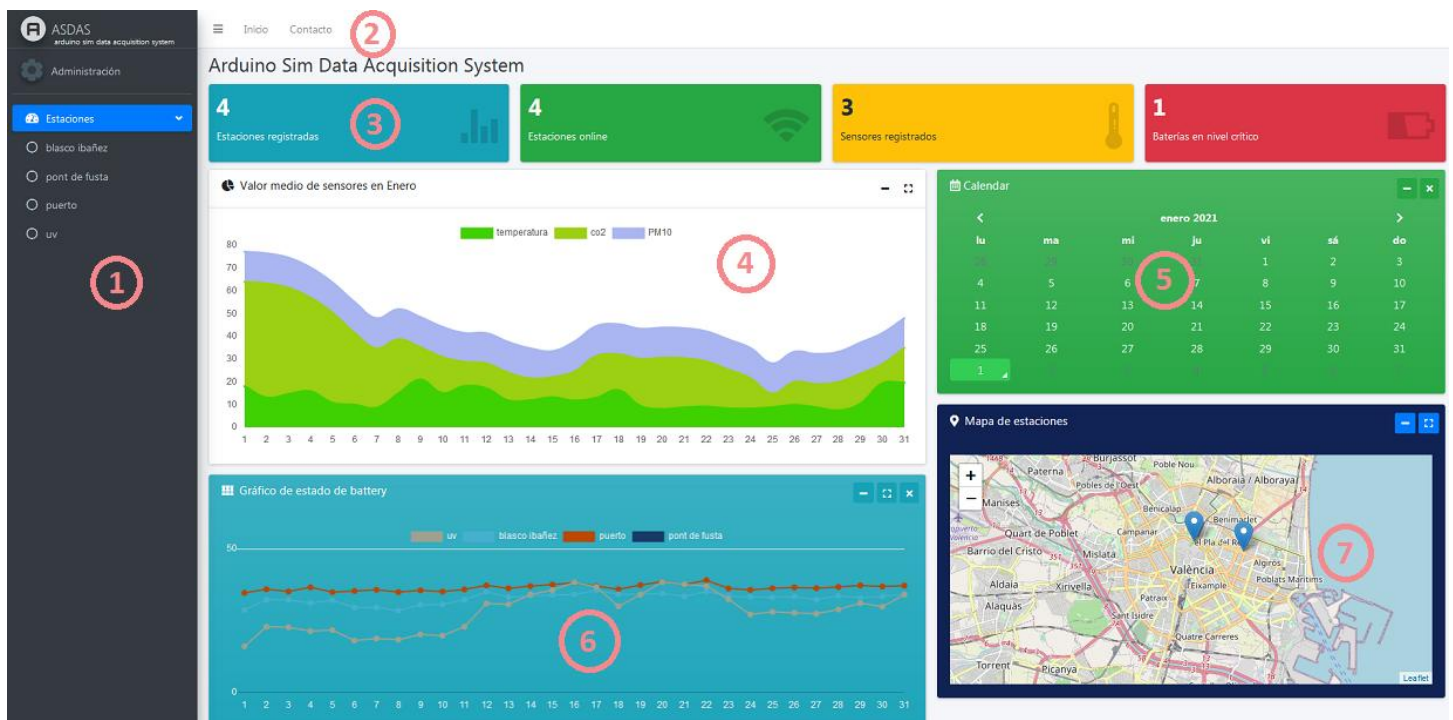


Figura 7.1: Interfaz de la web en el navegador

- (1) Barra lateral de acceso al área de configuración y a las distintas estaciones
- (2) Barra superior de acceso a la página principal y a la de contacto
- (3) Cuatro cajas que muestran estadísticas
- (4) Gráfica que muestra los sensores de la o las estaciones
- (5) Calendario en el que seleccionamos el día o mes para mostrar los datos
- (6) Gráfica que muestra los estados de las estaciones
- (7) Mapa que muestra la ubicación de cada estación



- Gracias al framework CSS [Bootstrap](#) conseguimos una interfaz muy responsiva en cualquier tipo de dispositivo.
- Gracias a la plantilla [Adminlte](#) conseguimos elementos muy visuales y dinámicos como cajas con efectos, gráficas con elementos selectivos, contenedores arrastrables y maximizables, menús dinámicos y una estructura de diseño muy completa.
- Gracias a la librería de Javascript [Leaflet](#) podemos cargar mapas con indicadores y rutas
- Gracias a la librería [Moment](#) de Javascript junto con la librería [Jquery](#) usando Ajax podemos cargar los datos en el navegador y procesarlos y filtrarlos desde él tras haber cargado la plantilla principal de la página.

Figura 7.2: Interfaz de la web en un smartphone

Web de demostración con datos simulados:

<https://demo.diab.website>

Web de datos reales de ASDAS (Arduino Sim Data Acquisition System):

<https://asdas.diab.website>

6.2.2. PRÁCTICAS USADAS

Para el diseño del código y su ejecución se han tenido en cuenta varias estrategias:

-Uso de una estructura basada en modelo-vista-controlador:

El navegador pide un recurso que le llega al controlador y solicita al modelo la información requerida previamente validada.

Por último se le devuelve la vista al navegador con los datos ya obtenidos del modelo (base de datos)

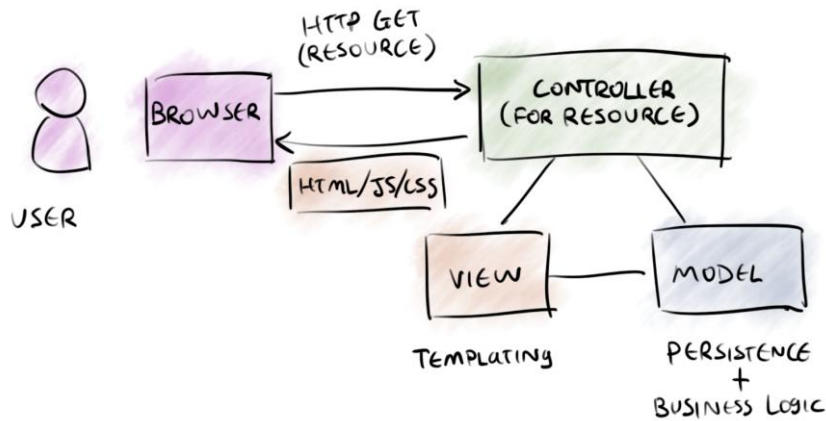


Figura 7.3: Esquema de la arquitectura MVC^[21]

De esta manera separamos funcionalidades y abstraemos clases, como en el caso del modelo, que independientemente de usar [SQLite](#) o [MariaDB](#) no tenemos que hacer cambios en el funcionamiento de la aplicación para pasar de uno a otro.

-Petición de datos mediante ajax:

Con esto reducimos la lógica de las peticiones a recibir por un lado la vista, y por otro lado los datos sin tener que pedir y recibir de nuevo la vista cuando requiramos nuevos datos.

-Filtrado y procesado de datos por Javascript:

Se trata de una buena práctica para reducir la carga del servidor, ya que éste simplemente se encargará de darte un bloque de datos que el navegador a través de **Javascript** se encargará de procesar.

Las partes de la web que funcionan así se podrían categorizar como parte de la arquitectura **MVVM** (modelo vista vista-modelo) ya que toda la lógica del modelo es procesada en la vista mediante **Javascript**.

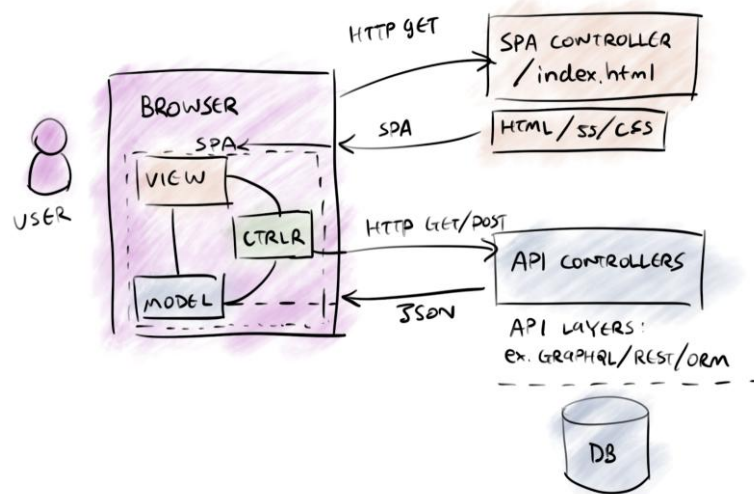


Figura 7.4: Esquema de la arquitectura MVVM^[22]

-Ejecución de CSS al inicio, y de Javascript al final de todo el HTML:

Con esto conseguimos una carga de la web fluida, ya que al cargar al comienzo las hojas de estilo **CSS** y posteriormente la estructura de la vista, conseguiremos tener cargado de manera secuencial y sin parones el contenido de la web. Y al final del todo se ejecutarán los scripts de tipo **Javascript**. De otra manera, si se cargasen al inicio, la web se quedaría en blanco hasta que la carga finalizase.

-Restricción de acceso a Urls:

Al no hacer uso de un Framework, no hay un sistema de rutas que te obligue a tener que crear cada petición que reciba el servidor.

Esto es inseguro, ya que se podría acceder a ciertos documentos de la web que pudieran ofrecer información sensible a un atacante.

Para subsanar estos problemas usamos tres prácticas:

➤ Verificar que la página tenía cargada una constante

Es decir, definiendo una constante en la página principal (index.php), el resto de páginas verificará que esa constante exista.

```
if (!defined('FROM_INDEX')) die();
```

➤ Crear ficheros .htaccess con reglas de apache

Apache brinda un sistema de reglas para denegar el acceso a ciertas rutas.

Lo usaremos para bloquear el acceso directo al fichero de base de datos Sqlite que se almacena en el servidor.

➤ Crear ficheros index.php vacíos

Como método de prevención si el servidor estuviera mal configurado o se desconfigurase, para evitar el listado de directorio creamos en cada carpeta un fichero index.php vacío.

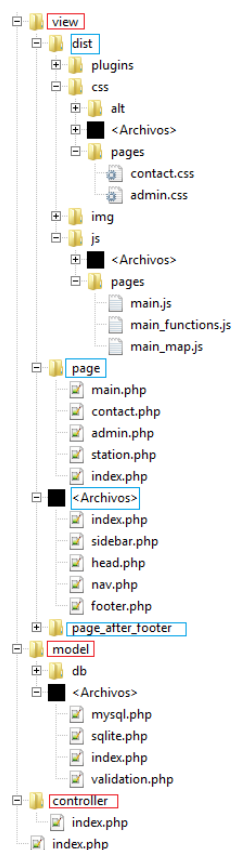
De esta manera en vez de listar los directorios mostraría un fichero vacío.

-Validación de datos:

Para los datos numéricos se verifica que cumplan un rango concreto, y para las cadenas se les escapa el carácter comilla para evitar ataques de inyección SQL.

6.2.3. ESTRUCTURA DEL CÓDIGO

Como se indicó anteriormente, la estructura del código de la web iba a estar basada en MVC (modelo-vista-controlador), ya que de esta manera conseguimos diferenciar en capas las funcionalidades de la aplicación, y al mismo tiempo conseguimos abstraer ciertas capas como la de la base de datos.



Las carpetas principales son: controller, model y view.

-El index.php llama primero al controller y éste llama al model.

-El controller (index.php dentro de controller) se encarga de llamar al model y de recibir las peticiones ya sea del sensor o del usuario.

-El model es llamado desde el controller, y éste a su vez llama a mysql o a sqlite según que base de datos hayamos configurado.

-El view lo llama el index después de gestionar las peticiones y la base de datos. Lo que hace el view es mostrar la web si es necesario (hay peticiones que no lo requieren) Y no solo mostrar la web, sino procesar después los datos en Javascript (todo lo que es php lo procesa el servidor, pero javascript lo procesa el ordenador del usuario).

En view está dist, page, los propios ficheros de la carpeta view, y page_after_footer

-dist contiene:

>plugins del framework que ayudan a hacer las gráficas, procesar datos, manejar fechas, etc. Y otros que he añadido como el del mapa.

>css, todo el contenido css que ya trae el framework mas el mio añadido para cada página dentro de la carpeta pages

>img, carpeta de imágenes para la web

>js todo el contenido de scripts de Javascript, tanto del framework, como los míos, que los he añadido en "pages" para cada página.

Éstos procesarán los datos para generar las gráficas, estadísticas y mapas.

-page contiene el contenido de cada página en modo plantilla, donde algunas de las variables fueron procesadas por php y otras serán procesadas por javascript.

-<Archivos> son los archivos de esa carpeta view, que son la plantilla de lo que es la cabecera de la web, las barras superior e izquierda, el pie de página y el contenido principal.

-page_after_footer es una carpeta donde se encuentran scripts php que llaman a la ejecución de scripts javascript, y es esencial que estén al final para que la web cargue antes que los scripts y éstos no demoren su carga.

Figura 7.5: Estructura de la web explicada

6.2.4. VARIABLES DE CONFIGURACIÓN

- Variables de configuración en la página principal (index.php)

DEBUG: Define si estará en modo de depuración (para ver mayor salida de errores)

DEBUG_REQUESTS_FILE: Define el nombre de un fichero de depuración que almacenará todas las peticiones web recibidas.

DATABASE_TYPE: Define el tipo de base de datos que vamos a usar (*mysql o sqlite*), también está la opción de *sqlite-demo* que hace uso de una base de datos Sqlite con datos simulados.

db_conf: Será el nombre de la base de datos (en caso de haber elegido Sqlite) o un vector con la configuración de la base de datos (en caso de ser Mysql)

Ejemplos:

```
$db_conf="base_de_datos_sqlite.db"; //sqlite
```

```
$db_conf= ["host","usuario","contraseña","nombreDB"]; //mysql
```

- Variables de configuración en la base de datos

PASS: Define una contraseña para el área de administración

FM: Define un muestreo de datos por defecto, es decir, por cada cuantos minutos mostrará un dato en la gráfica

ONLINE_THRESHOLD_MINUTES: Define el tiempo para considerar una estación online. Por encima de este valor la estación aparecerá desconectada.

PRIMARY_SENSOR: Define un sensor principal para visualizar sus datos en las estadísticas

PRIMARY_STATUS: Define un estado principal para visualizar sus datos en las estadísticas

6.2.5. FLUJO DE EJECUCIÓN

En el apartado anterior explicamos un poco el flujo de ejecución de la web analizando jerárquicamente los directorios de la misma, pero ahora analizaremos de inicio a fin el flujo secuencial completo de una petición web.

-> **index.php** Inicializa variables, ejecuta `controller/index.php` y después devuelve la vista en `view/index.php`

-> **controller/index.php** Carga la base de datos `model/index.php` , recoge las peticiones web, y las analiza. Si tiene que interaccionar con el modelo (base de datos), llama a las funciones de `model/index.php`, una vez lo ha hecho devuelve la vista o el mensaje de respuesta al cliente (según sea una petición con o sin vista).

-> **model/index.php** Carga la validación de datos `model/validation.php`, y la clase específica para la base de datos configurada (`model/sqlite.php` ó `model/mysql.php`) , y ejecuta las consultas necesarias en la clase de la base de datos previamente validando los datos en la clase de validación.

-> **model/validation.php** Valida los datos recibidos

-> **model/sqlite.php** Ejecuta sentencias compatibles con esta base de datos.

-> **view/index.php** Ésta es la página principal de la vista, contiene la estructura básica, y al final los scripts de librerías javascript que cargará. Funciona como plantilla, ya que dentro de ésta llamará a `view/head.php` , `view/nav.php` , `view/sidebar.php`, `view/page/'.$current_view.'.php` , `view/footer.php` y `view/page_after_footer/af_'.$current_view.'.php`

-> **view/head.php** Carga todo lo referente al <head> del html (ficheros css, etiquetas meta...) y el fichero `view/dist/css/pages/'.$current_view.'.css`

-> `view/dist/css/pages/'.$current_view.'.css` fichero css dinámico que es diferente según la vista solicitada.

-> **view/nav.php** Carga la barra superior de enlaces

-> **view/sidebar.php** Carga la barra horizontal de enlaces

-> **view/page/'.\$current_view.'.php** Carga la vista de la página solicitada

-> **view/footer.php** Carga el pie de página

-> **view/page_after_footer/af_'.\$current_view.'.php** Ejecuta scripts tipo javascript referentes a la vista solicitada tras cargar el footer (es decir cuando se da por cargada toda la vista)

Con esto podríamos dar por finalizado el flujo de la web, pero aquí sólo finaliza el flujo PHP.

Posteriormente y tras cargar la vista, Javascript ejecuta varias funciones, y entre ellas, si es la vista principal (página principal y páginas de los sensores), realizará peticiones Ajax al servidor para solicitar los datos de los sensores que previamente procesa y filtra.

De todo esto se encargan los ficheros `main.js` y `main_functions.js`
Almacenados en la carpeta `view\dist\js\pages`

Estos scripts imprimirán los datos en la web de manera ordenada.

No se va a extender la explicación sobre el contenido de estos scripts ya que no es la funcionalidad final del proyecto, así que se tomarán como un "sistema externo" que nos filtra e imprime la información en el navegador.

7. CIRCUITO

7.1. DISEÑO

Se ha diseñado el circuito mediante [Kicad](#), ya que es una alternativa gratuita que nos permite diseñar tanto el circuito a nivel esquemático como el diseño de la PCB (Printed Circuit Board).

7.1.1. ESQUEMA DEL CIRCUITO

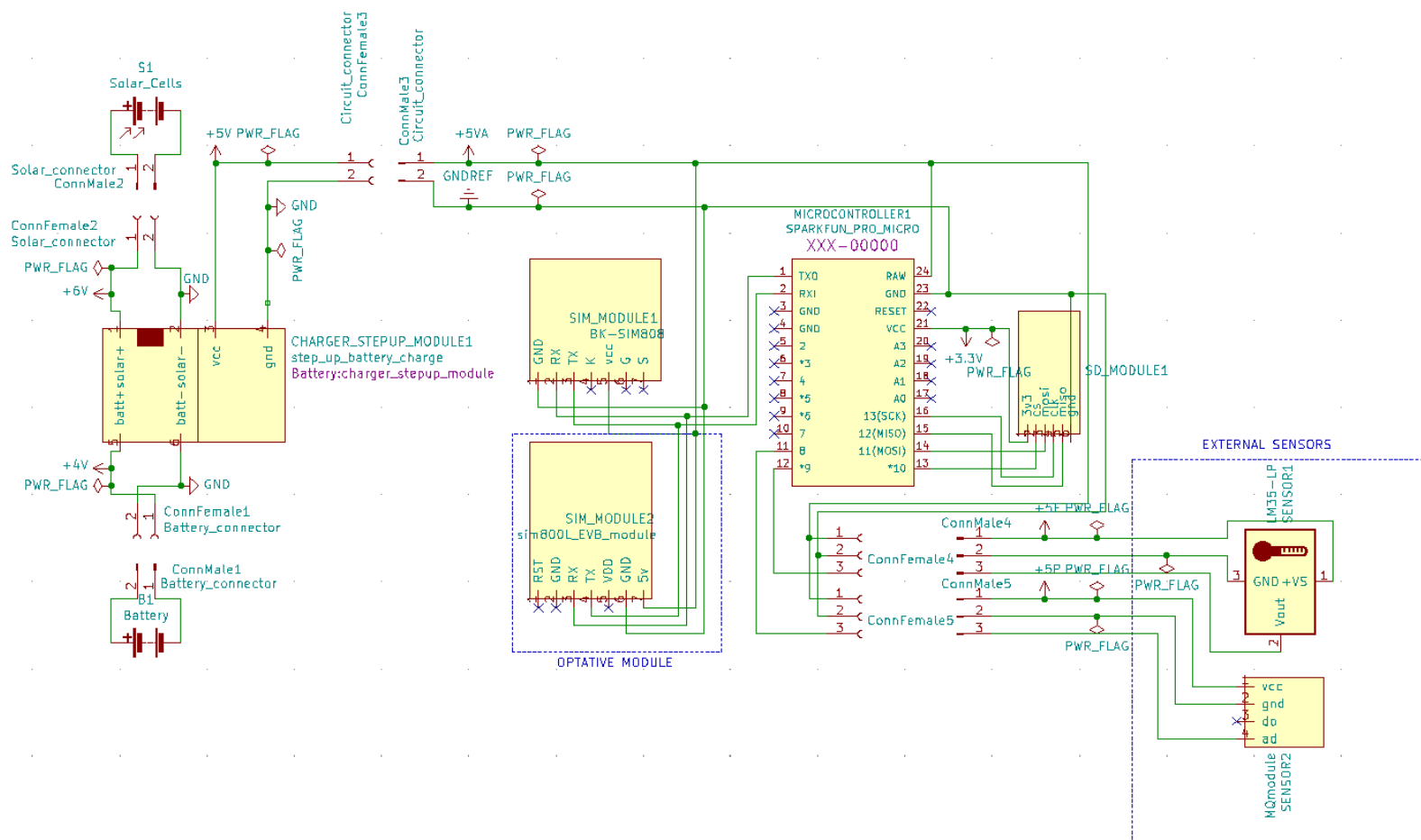


Figura 8.1: Diseño del esquema del circuito de la estación con Kicad.

Para el diseño del circuito se han incluido conectores para dividir el diseño en partes.

Por un lado va conectada la batería, el panel solar al módulo de carga de tensión y por el otro lado el resto de circuito también va conectado con otro conector a esa placa.

El resto del circuito se compone de los módulos (SIM, SD), el microcontrolador, y los sensores que van conectados también a través de un conector de 3 cables.

Los voltajes que distribuye el circuito son de 6 Voltios que genera el panel solar y deriva al módulo de carga, el cual se encarga de cargar la batería y al mismo tiempo proporcionar 5 Voltios al resto del circuito, y adicionalmente el microcontrolador ofrece 3.3 Voltios al módulo SD.

7.1.2. PCB DEL CIRCUITO

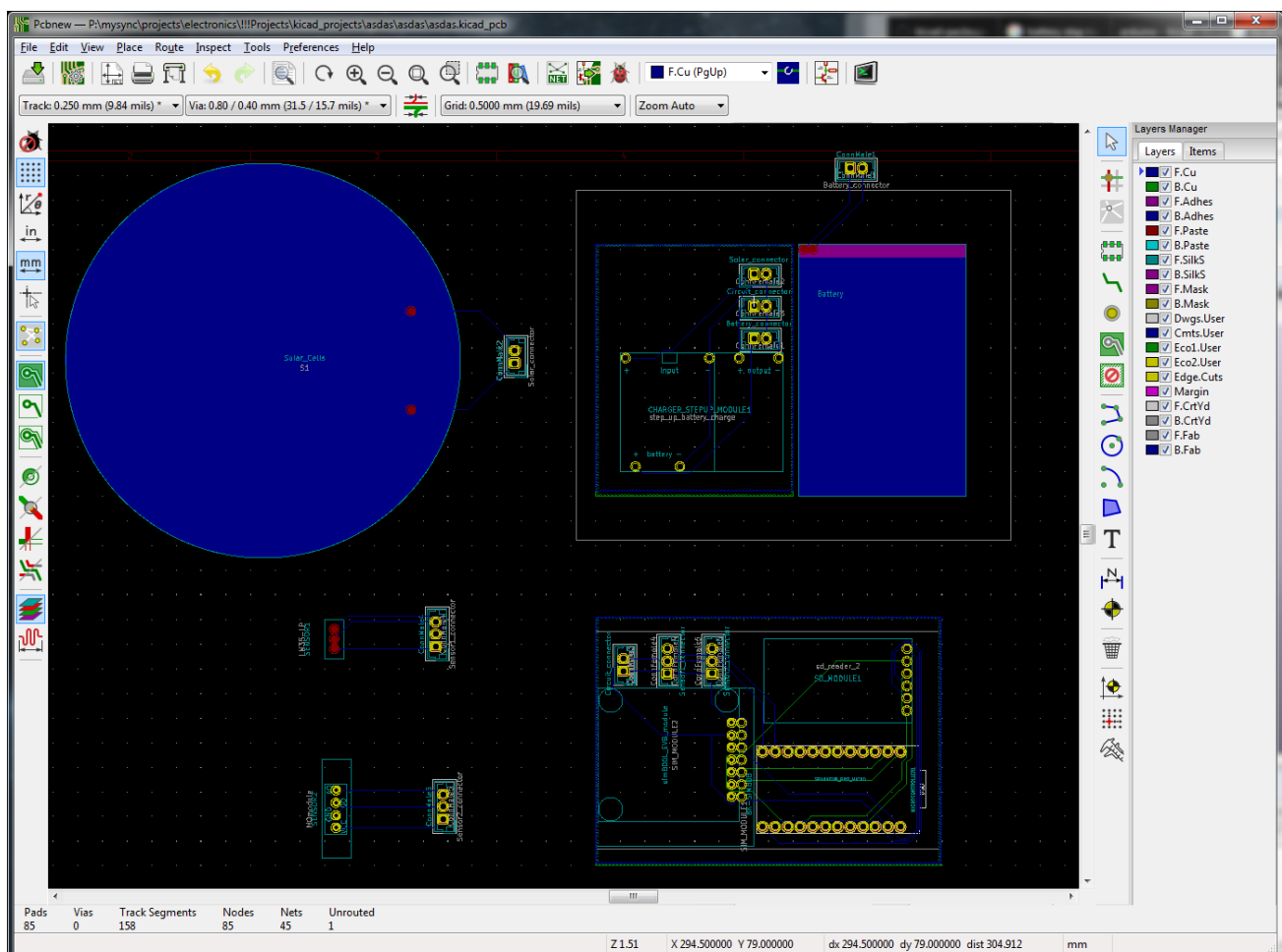
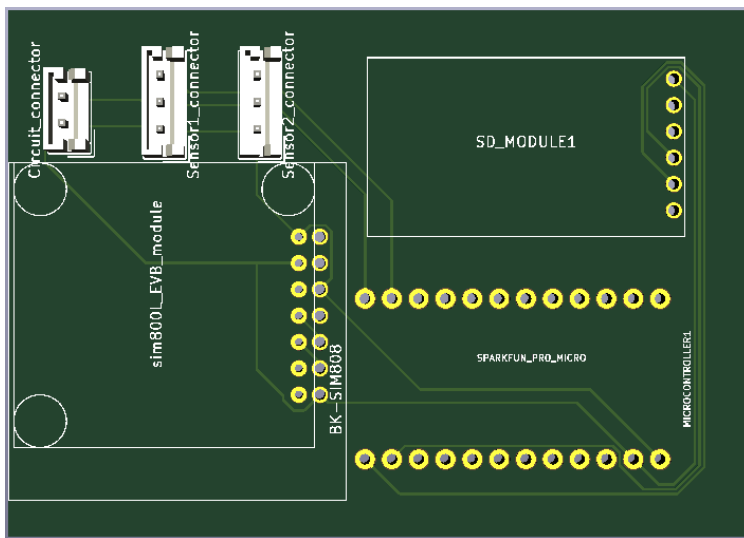


Figura 8.2: Diseño del PCB del circuito en Kicad

Se ha realizado el diseño del **PCB** para el prototipo, pensado para la placa de test de 7x5cm comprada, así que no es un diseño optimizado como tal, sino un diseño pensado para este prototipo con las piezas previstas.

Para un diseño más optimizado las medidas no estarían limitadas a un mínimo de 7x5cm sino que a parte de ser mucho más pequeño, varios de los módulos se diseñarían por componentes en el **PCB** en vez de usar el módulo completo (como es el caso del microcontrolador, del que diseñaríamos el **PCB** únicamente con su chip interno **ATmega32U4** y los componentes necesarios).



Esta es la placa de prototipo en la que iría conectada el microcontrolador y los módulos SIM y lector de tarjetas micro SD.

Adicionalmente tiene un conector para conectarse a la placa de alimentación, y dos conectores para conectarle dos sensores.

Figura 8.3: Diseño del PCB(placa de microcontrolador y módulos)

Esta es la placa de alimentación.
En ella va el módulo de carga que se encarga de recibir y distribuir la energía en el circuito.

Tiene 3 conectores para conectar el panel solar, la batería y el PCB del circuito.

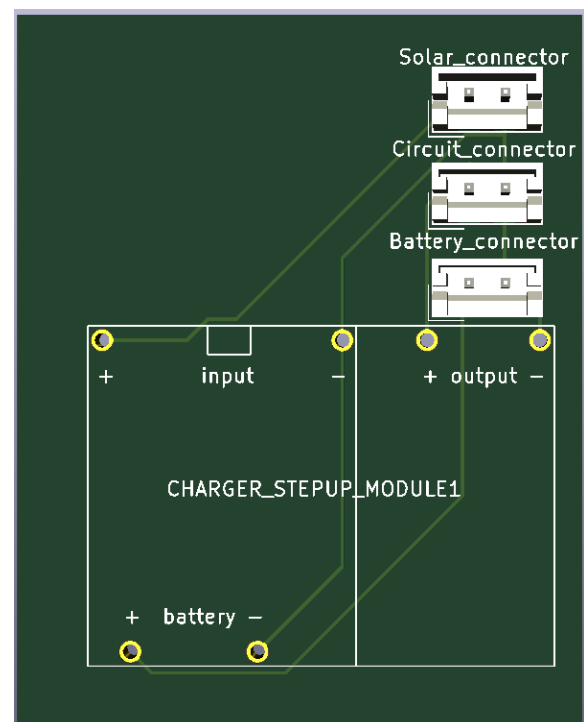


Figura 8.4: Diseño del PCB (placa del circuito de control de energía)

7.2. ENSAMBLADO Y MONTAJE

Para montar el circuito se han soldado los módulos, conectores y cables a las placas de agujeros de 70x50mm de la lista de materiales adquiridos, y esto se ha hecho siguiendo el procedimiento del PCB elaborado en el apartado anterior.

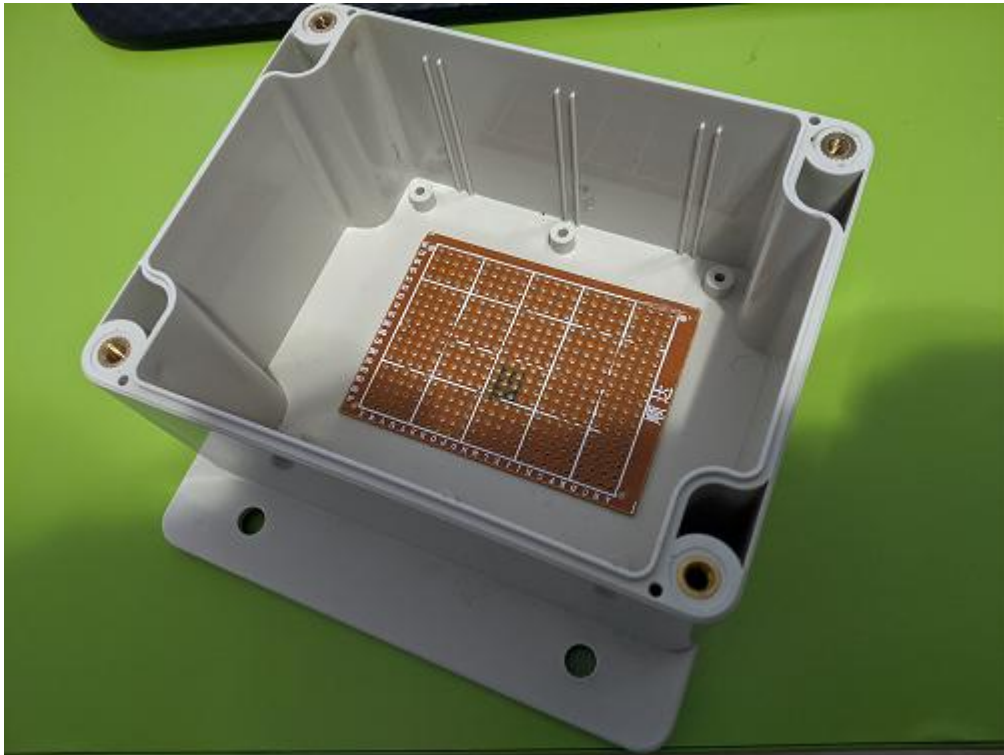


Figura 8.5: Caja de montaje y placa de soldado

Para montar la placa ya soldada con todos los componentes en la caja IP65 de la lista de materiales, se ha diseñado en Blender e impreso en 3D un sistema que permitiese adaptar la placa de 70x50mm a la caja con un interior de 88x83mm.

Se ha ideado una especie de base con 6 torres, aprovechando que el fondo de la caja cuenta con 6 agujeros salientes pensados para ser atornillados.

Adicionalmente, a esa superficie se le ha añadido unas "guías" para que pase por ahí el PCB, un agujero debajo para que quepan los cables soldados por debajo, y una abertura a un lado para poder insertar todo el circuito una vez soldado.

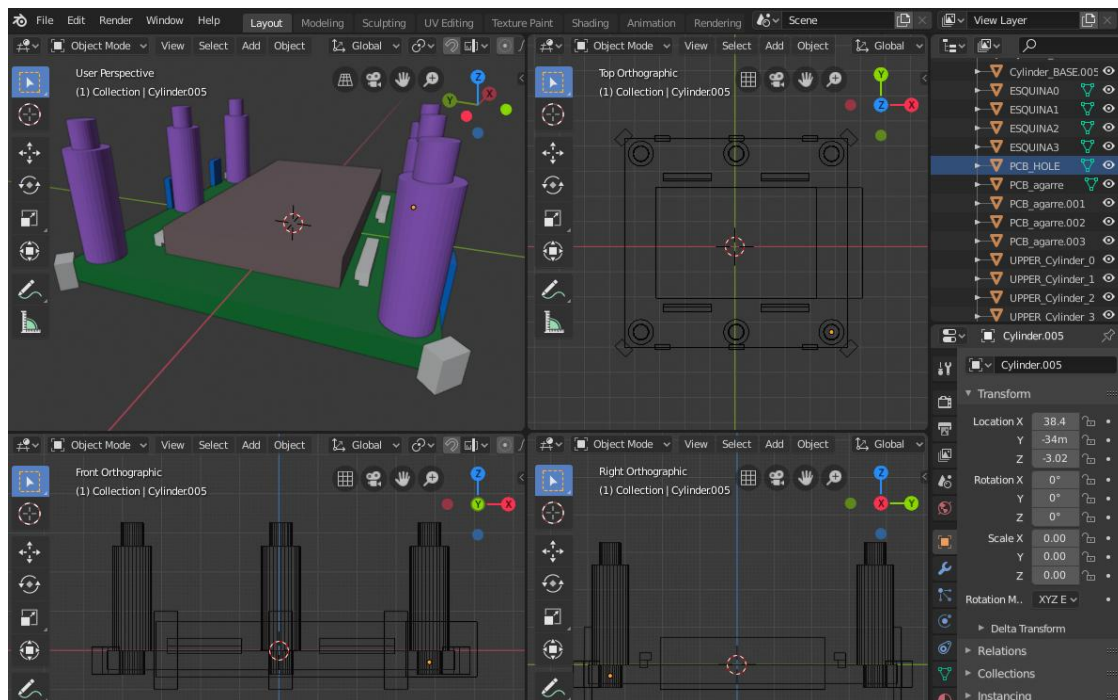


Figura 8.6: Diseño en blender de la estructura a base de figuras simples

El proceso ha sido realizado a partir de figuras simples unidas o restadas a otras con la herramienta de booleanos.

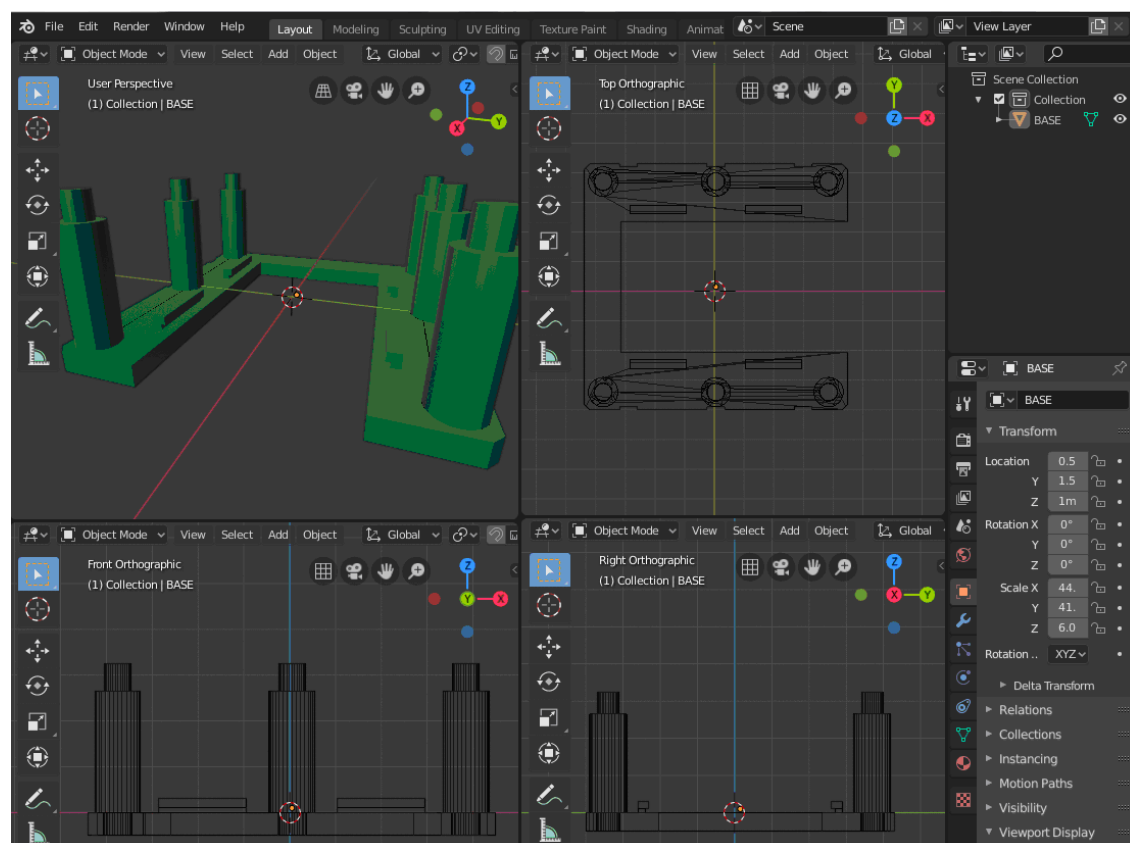


Figura 8.7: Diseño en blender de la estructura inferior (zona del microcontrolador y módulos)

En esta base irá el PCB del microcontrolador y módulos, y a ella va encajada encima a modo "Lego" la siguiente PCB en otra base utilizando el mismo tipo de diseño:

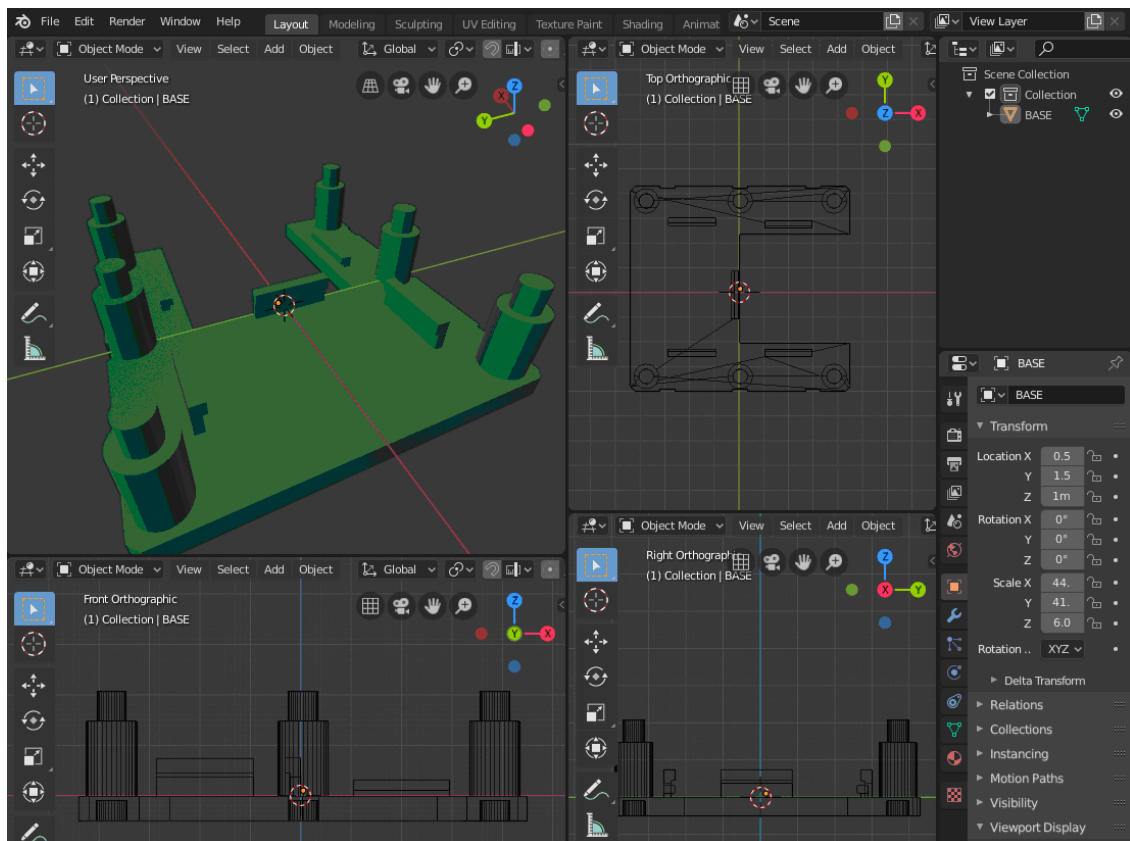


Figura 8.8: Diseño de la estructura intermedia (zona de la batería y módulo de control de energía)

La diferencia de esta otra es que tiene media plataforma reservada para el PCB y la otra media para encajar la batería.

Y por último, encima de esta va otra plataforma más en la que se encaja la placa solar:

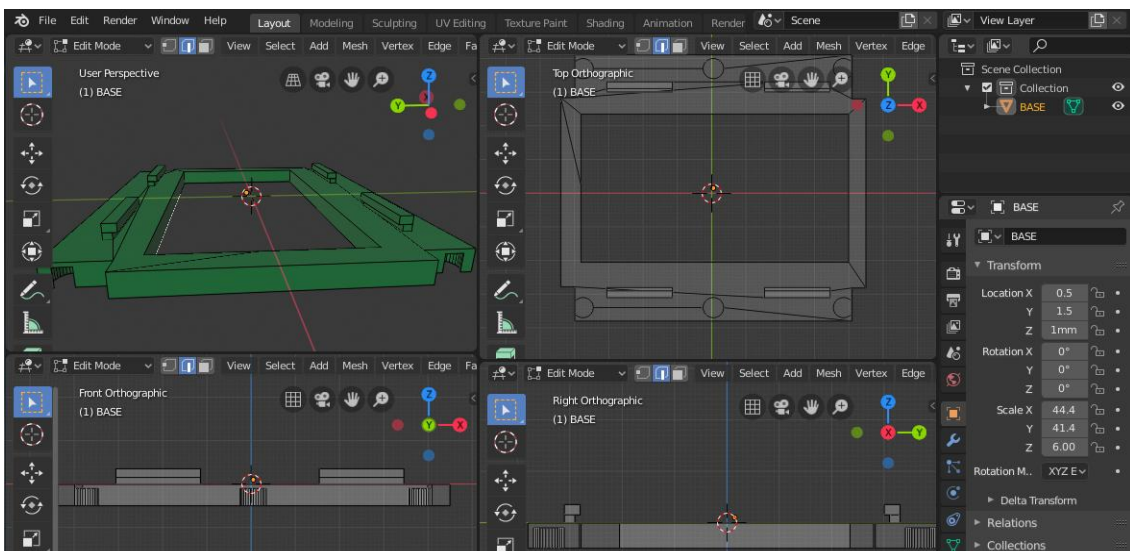


Figura 8.9: Diseño de la estructura superior (zona del panel solar)

Una vez terminado el diseño e impreso en 3D este es el resultado:

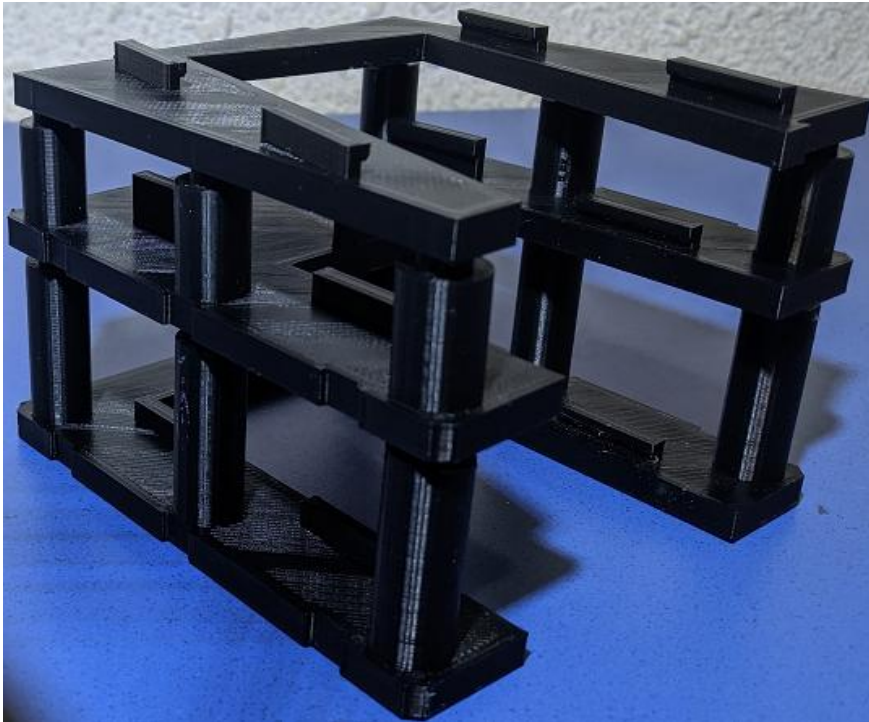


Figura 8.10: Diseño final de la estructura

Y una vez terminado el proceso de soldar los cables y conectores a las placas de pruebas y tras haberlos ensamblado en el diseño impreso este es el resultado:

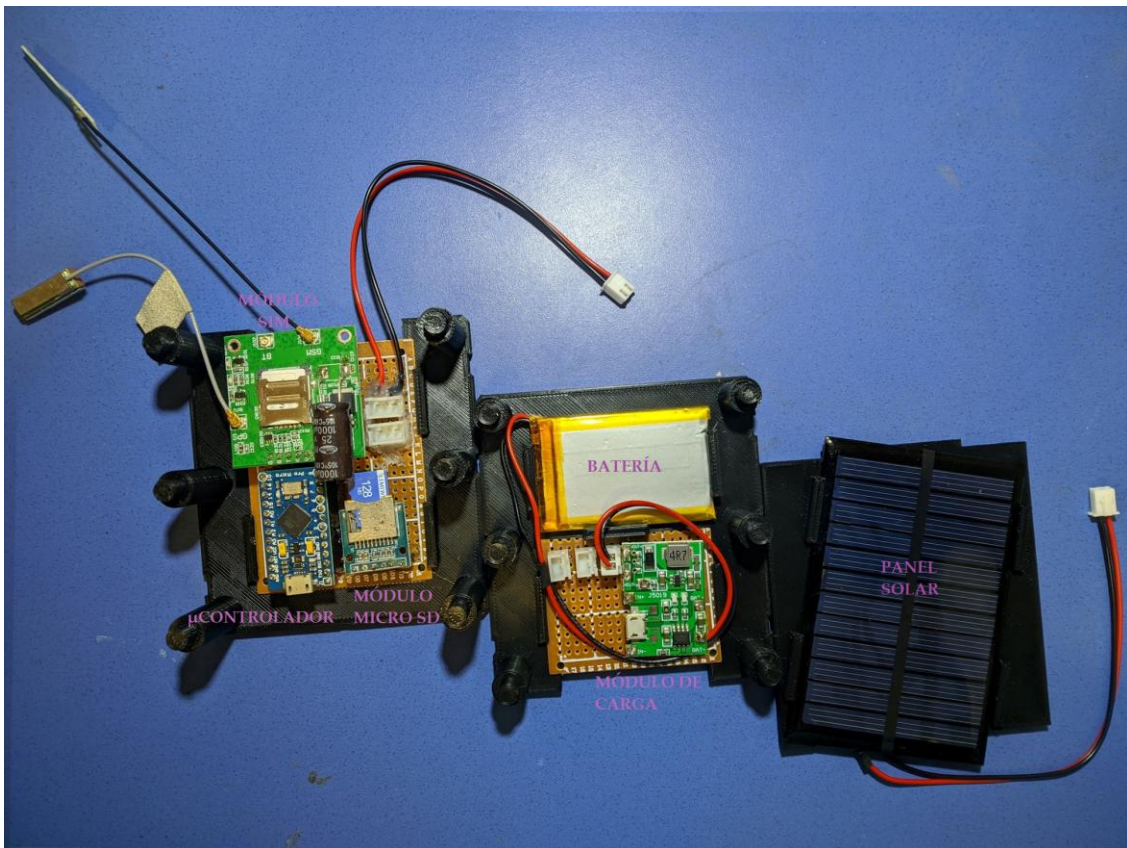


Figura 8.11: Diseño del montaje del circuito en la estructura

8. PRUEBAS

Inicialmente se testearon todos los componentes principales y las librerías que controlan los módulos en una [protoboard](#).

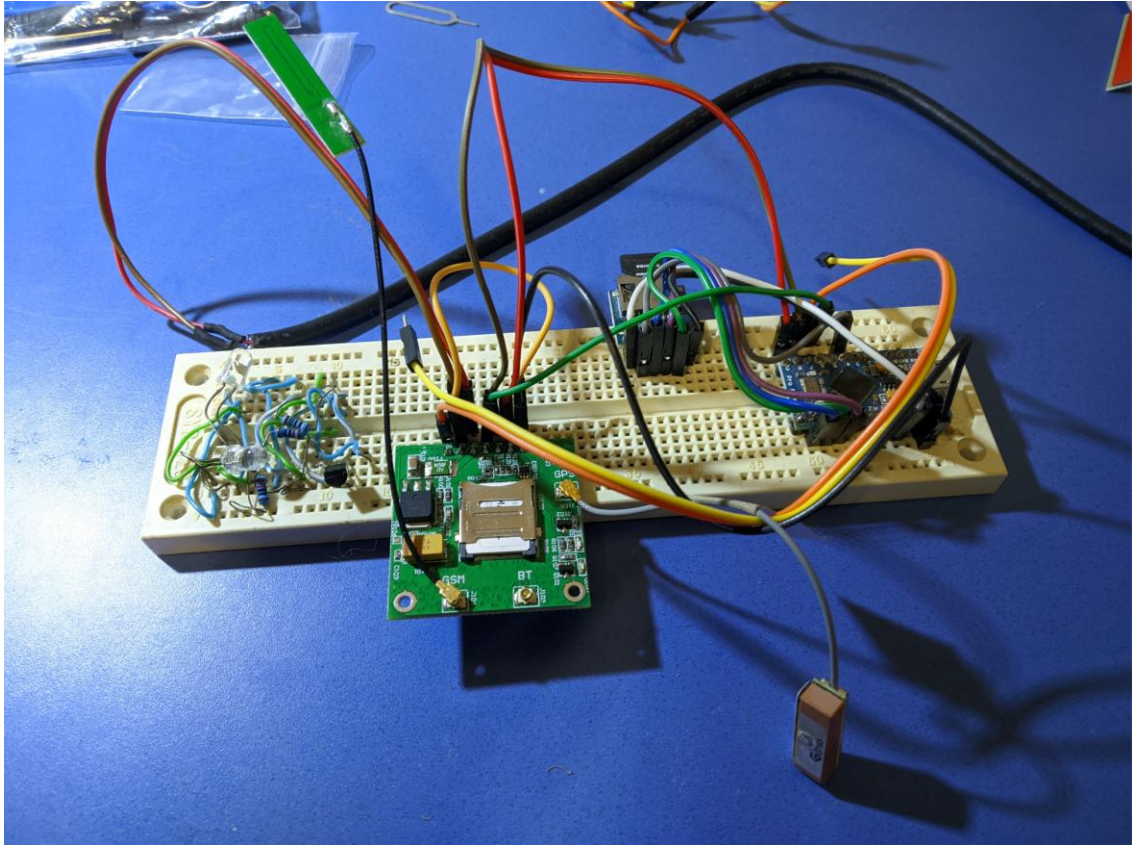


Figura 9.1: Pruebas del circuito en placa Protoboard

Para la web del servidor se creó una simulación de datos de estaciones ficticias con datos de sensores y ubicaciones configurable fácilmente si activamos la opción de tipo de base de datos "[sqlite-demo](#)" en el [index.php](#).

Para generar la simulación se creó un script PHP:

"[model/db/EXTERNAL_sql_data_generator.php](#)" que generaba los datos con cierta aleatoriedad siguiendo un patrón y entre unas fechas indicadas.

Si la web está funcionando este modo ("[sqlite-demo](#)") adicionalmente, cada día moverá los datos simulados hacia el día actual para poder continuar testando siempre sobre el día actual.

Así se logró testear el funcionamiento general de la web (agrupación de datos, generación de gráficas, etc).

Previamente a ensamblar finalmente todo el diseño con PCBs en la caja de protección IP65, se testeó el dispositivo funcionando únicamente con batería.

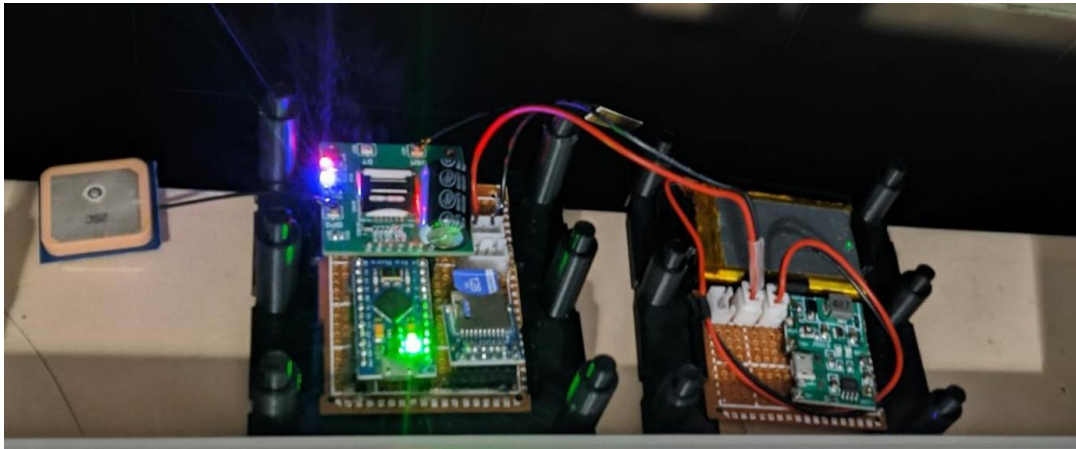


Figura 9.2 Pruebas del circuito con batería y estructura final

Finalmente se testeó el dispositivo ensamblado recabando datos de sensores de temperatura y monóxido de carbono junto con los datos de la ubicación con el GPS.



Figura 9.3 Pruebas del dispositivo ensamblado

En la figura 9.4 podemos observar los datos recogidos por los sensores de temperatura y monóxido de carbono cada 5 minutos entre las 17:20 y las 23:45 el día 17 de Marzo.

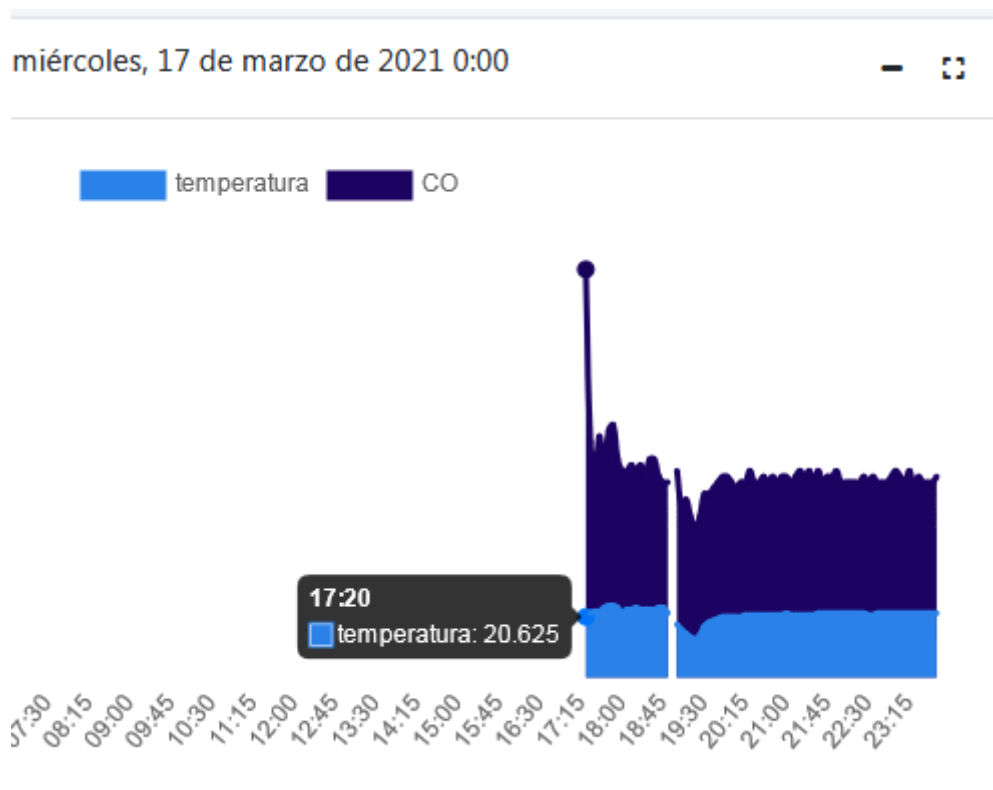


Figura 9.4 Datos recogidos de los sensores

De ella, podemos sacar varias conclusiones:

1. El sensor de monóxido tiene un pico ascendente inicial, cosa que entra dentro de lo normal, ya que tarda unos minutos en estabilizar su valor hasta que se calienta el gas.
2. El sensor de temperatura tiene un pico descendente sobre las 19:20, coincidiendo con el momento en el que la estación fue sacada de la mochila, asumiendo un descenso de la temperatura durante el periodo de alrededor de 10 minutos que fue sacado al aire libre.
3. A las 18:55 hay una pérdida de información en el envío de los valores de esa medición.
4. La batería ha durado hasta agotarse durante 6 horas y 25 minutos.

Sobre el anterior punto número 3, se ha analizado el sistema de almacenamiento de peticiones en el área de administración de la web de [ASDAS](#) creado para facilitar el análisis de las peticiones recibidas, y entre ese intervalo de tiempos obtenemos los datos recogidos en la siguiente figura:



Figura 9.5 Log de peticiones web de la estación

Como podemos observar, en el intervalo entre las 18:53 y las 19:03 se han recibido peticiones pero mal formadas o vacías.

Esto significa que ha habido un fallo de transmisión que puede ser causado por múltiples circunstancias.

Entre ellas podría ser causado por una falla temporal en la señal del módulo SIM, o en el suministro de energía requerido en ese instante por el módulo SIM.

Habría que tenerlo en cuenta para mejorar el sistema de verificación de peticiones recibidas correctamente.

Como conclusión esto significaría que en esta prueba de 77 muestras obtenidas (una muestra cada 5 minutos durante 385 minutos) una ha fallado, es decir, una tasa de error del 1.29% para este día.

En otros días a veces también se ha observado este fenómeno perdiendo también una única muestra.

En la figura 9.6 podemos observar los datos recogidos de ubicación en el mapa referentes a un trayecto entre la zona de la playa y el centro de Valencia.

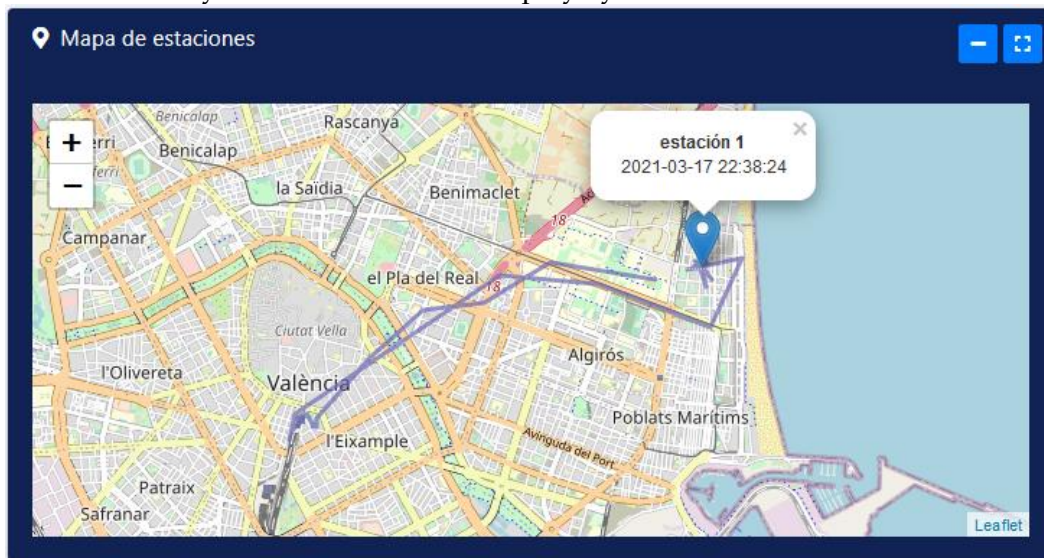


Figura 9.6 Datos recogidos de ubicación GPS en el centro de Valencia

Y en la figura 9.7 podemos observar un trayecto entre la playa y el puerto.

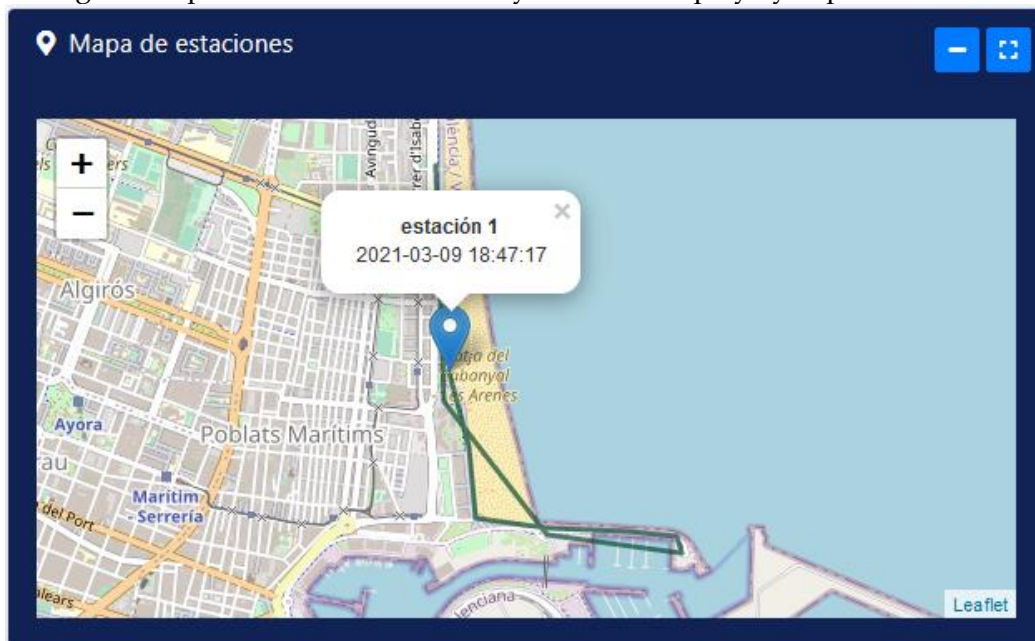


Figura 9.7 Datos recogidos de ubicación GPS en el puerto

Como se puede observar en ambas figuras, la ubicación recogida es bastante fiel a la realidad. Evidentemente se podría mejorar cuantas más muestras tomemos en el menor tiempo posible, ya que al tomar una muestra de ubicación cada 5 minutos unirá los puntos entre cada uno de esos 5 minutos y partes del trayecto no se verán reflejadas de manera fiel.

Además, ambos trayectos fueron realizados en patines, es por eso que hay tramos que no reflejan con exactitud el trayecto dado que en 5 minutos se ha recorrido mucha distancia.

9. MEJORAS

➤ Encriptación de la transmisión:

Sería importante de cara a la seguridad de la información transmitida por la red 2G de datos móvil el encriptarla al transferirla.

Primeramente no se hace uso de HTTPS porque los módulos SIM para 2G o bien no lo soportan, o lo soportan de manera muy inestable, y de soportarlo no soportan protocolos superiores a TLS 1.0.

La solución pasaría por implantar un sistema de claves con un algoritmo AES, lo cual aumentaría el tamaño de nuestro programa, y con gran seguridad, tal y como está diseñado no cabría en la memoria flash de nuestro dispositivo, por lo que habría que buscar otro dispositivo con mayor capacidad para esta tarea.

➤ Optimización de recursos energéticos:

Para conseguir que la estación sea sostenible energéticamente hablando, sería necesaria la optimización del uso de recursos del microcontrolador.

Para ello habría que poner en modo ahorro y ajustar los valores para que se ponga en modo de bajo consumo durante el tiempo que no va a recoger datos de los sensores ni de ubicación.

Esto requiere una valoración a fondo y depende mucho de la frecuencia de trabajo que le queramos asignar a la estación, ya que en iniciar la conexión tanto de GPS como de GSM toma tiempo y consume picos altos de energía, por lo tanto, se requiere valorar la rentabilidad energética en base al tiempo de uso en activo.

➤ Lectura real de la batería restante:

A pesar de obtener por comando AT del módulo SIM la batería restante, ésta siempre va a ser del 100% salvo que tenga fallos de suministro. Esto se debe a que esa funcionalidad está pensada para alimentar el chip SIM808 con la batería, y no su módulo. En este caso no tiene un remedio directo, ya que el módulo requiere obligatoriamente un voltaje superior al que ofrece una batería común, y por tanto requiere una conversión de voltaje que nosotros regulamos a 5 voltios con el módulo de "Step Up".

Como solución se puede medir con dos resistencias conectadas entre un pin de entrada del microcontrolador y otro de la batería, para medir el voltaje restante real que le queda a la batería.

- Análisis exhaustivo sobre la autonomía con panel solar :

No se han hecho pruebas suficientes para saber si en condiciones no ideales el panel solar puede abordar con la autonomía suficiente como para aguantar encendido con funcionalidades de diferentes sensores y gps activos.

- Elaboración de un PCB real:

El diseño de PCB realizado, tal y como se indicó, es para prototipo y también sirve de guía para soldar los módulos del circuito sin PCB.

Por tanto ,para un diseño real, habría que sustituir ciertos componentes para convertirlo en un PCB optimizado.

-Microcontrolador:

Se cambiaría la placa del microcontrolador por únicamente su chip interno (Atmega32U4). Esto ahorraría costes y espacio en el PCB.

-Regulador de voltaje:

Se dejaría de usar un módulo de Step-up sustituyéndolo por componentes electrónicos que estarían integrados en el PCB.

-Lector de tarjetas micro SD:

Se sustituiría por los componentes que irían directamente integrado en el PCB.

-Módulo Sim:

Se sustituiría por su chip interno con los componentes del módulo integrados en el PCB. Este cambio no saldría muy rentable, dado el trabajo que requiere diseñar un módulo para este tipo de chip.

10. BIBLIOGRAFÍA

- [1]Proyecto de adquisición de datos con arduino,« Real-Time 2G/3G/LTE Arduino GPS Tracker + IoT Dashboard © CC BY-NC », <https://create.arduino.cc/projecthub/botletics/real-time-2g-3g-lte-arduino-gps-tracker-iot-dashboard-01d471>
- [2]Paper sobre desarrollo de sistema de adquisición de datos con Arduino,« Development of Internet of Things (IoT) based Data Acquisition system using Arduino Uno », <https://www.irjet.net/archives/V8/i1/IRJET-V8I1152.pdf>
- [3]Paper sobre desarrollo de sistema de adquisición de datos con Raspberry Pi,« Internet of Things (IoT) based Data Acquisition system using Raspberry Pi », <http://www.internationaljournalssrg.org/IJCSE/2016/Volume3-Issue11/IJCSE-V3I11P108.pdf>
- [4]Red celular y Gsm,« Estándar GSM », <https://es.ccm.net/contents/681-estandar-gsm-sistema-global-de-comunicaciones-moviles>
- [5]SDB,« Iridium Short Burst Data », https://www.datawell.nl/Portals/0/Documents/Brochures/datawell_brochure_iridium_sbd_b-40-03.pdf
- [6]Módulo de comunicación Iridium 9602,« Iridium 9602 SBD Transceiver Developer's Guide », http://g-layer.com.au/wp-content/uploads/IRDM_9602DeveloperGuideV4_DEVGUIDE_Sep2012.pdf
- [7]Módulo de comunicación satelital RockBLOCK MK2,« RockBLOCK MK2 », <https://www.rock7.com/products/rockblock-iridium-9602-satellite-modem>
- [8]SDB HTTP API,« Iridium SDB HTTP POST API », <https://www.rock7.com/downloads/RockBLOCK-Web-Services-User-Guide.pdf>
- [9]BGAN,« What is BGAN? », https://www.groundcontrol.com/What_Is_BGAN.htm
- [10]Estadísticas de uso de tecnologías, lenguajes y servidores, « World Wide Web Technology Surveys », <https://w3techs.com/>
- [11]3P,« 3P : Progress Programmers Pal », <https://jcaillon.github.io/3P/>
- [12]Comparación de diferentes CMS's, « WordPress vs. Joomla vs. Drupal », <https://websitesetup.org/cms-comparison-wordpress-vs-joomla-drupal/>
- [13]Señal 2G en España,« Mapa cobertura móvil », <https://www.movistar.es/particulares/coberturas/movil/>
- [14]Manual de comandos AT internos de la familia de chips Sim800, « SIM800Series_AT CommandManual_V1.10 », https://simcom.ee/documents/SIM868/SIM800%20Series_AT%20Command%20Manual_V1.10.pdf

- [15]Manual de comandos gps y especificaciones del chip Sim808, « GSM/GPRS+GPS Module SIM808 », <http://www.ds-tech.com.tw/IOT/SIM808-2.html>
- [16]Especificaciones del módulo Sim800l, « SIM800L GSM / GRPS module », <https://nettigo.eu/products/sim800l-gsm-grps-module>
- [17]Librería sim808,« blemasle/arduino-sim808 », <https://github.com/blemasle/arduino-sim808>
- [18]Características Mariadb/Mysql,« MySQL Features », <https://www.javatpoint.com/mysql-features>
- [19]Características Sqlite,« Distinctive Features Of SQLite », <https://sqlite.org/different.html>
- [20]Arquitecturas MVC y MVVM, « MVC vs MVVM: Key Differences with Examples », <https://www.guru99.com/mvc-vs-mvvm.html>
- [21]Arquitecturas MVC y MVVM, « From MVC to Modern Web Frameworks », <https://hackernoon.com/from-mvc-to-modern-web-frameworks-8067ec9dee65>
- [22]Arquitecturas MVC y MVVM, « Software Design Patterns », <https://patterns.florian-rappl.de/Category/Presentation%20patterns>
- [23]Visual paradigm con MVC, « How to Model MVC Framework with UML Sequence Diagram? », <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/how-to-model-mvc-with-uml-sequence-diagram/>
- [24] Proyecto de envío de datos de sensores por GPRS a la nube, « Sending Data via SIM800L GPRS to ThingSpeak », <https://www.teachmemicro.com/send-data-sim800-gprs-thingspeak/>