



UNIVERSIDADE DA CORUÑA

Facultad de Informática

Trabajo de Fin de Grado
Grado en Ingeniería Informática

Mención en Ingeniería de Computadores

Sistema autónomo de monitorización meteorológica con emisión de informes periódicos

Alumno: Estanislao Ramón Pérez Nartallo

Director: José Antonio García Naya

A Coruña, septiembre de 2015

A mi familia y amigos, gracias por vuestro apoyo y confianza.

Agradecimientos

A mi madre y a mi hermana que me han apoyado durante todos estos años, sin vosotras no lo habría conseguido.

A mi padre, cuya pasión por la informática es el causante de que hoy estuviese escribiendo estas palabras.

Gracias a Jose, mi director de proyecto, por su continua dedicación y apoyo a lo largo de todo este trabajo.

A Yaiza, por todo su cariño y apoyo a lo largo de todo este tiempo.

Por último, también quiero agradecer todo el cariño de mis amigos, mi segunda familia, y a mis compañeros de la Universidad que se han convertido en grandes amistades, en especial a Avelino.

Gracias a todos.

Resumen

En este Trabajo de Fin de Grado se ha construido una estación meteorológica que recoge datos de sus sensores y los envía a un servidor mediante conectividad WiFi.

Para el desarrollo del trabajo se ha empleado una metodología incremental basada en iteraciones:

1. Estudio sobre el estado del arte.
2. Análisis de requisitos.
3. Diseño e implementación del servicio REST.
4. Diseño e implementación de la estación meteorológica.
5. Estudio del consumo energético de la estación meteorológica.
6. Desarrollo del cliente Web.

En cuanto al hardware nos hemos decantado por emplear soluciones de hardware abierto como es Arduino, Adafruit CC3000 y una Raspberry Pi. La estación meteorológica está constituida por una placa Arduino Uno, una Shield WiFi Adafruit CC3000 y un par de sensores: DHT22 y BMP180. La Raspberry Pi dispone de una distribución GNU/Linux que ejecuta el software del servidor REST y el cliente Web.

El software del sistema, el servidor REST y el cliente Web, están desarrollados en Javascript, empleando el entorno de ejecución multiplataforma Node.js, lo que nos otorga independencia de la plataforma y del sistema operativo. La programación de Arduino se ha realizado empleando el entorno de desarrollo propio de Arduino mediante el lenguaje de programación Processing, basado en C++.

El servidor REST ha sido diseñado de forma que su API sea versionable y se adapte a diferentes versiones de los clientes que la empleen. Como sistema de almacenamiento se ha optado por MongoDB. El sistema emplea unas políticas de acceso y seguridad basada en grupos de usuarios: administradores, usuarios y estaciones meteorológicas; limitando de esta manera las funciones que pueden realizar.

Por último, el cliente Web permite manejar todas las funciones del sistema empleando la API REST desarrollada. Se pueden consultar los datos de la estación meteorológica de forma gráfica y configurar sus parámetros de transmisión.

Palabras clave

Arduino, Raspberry Pi, WiFi, REST, MongoDB, estación meteorológica, Javascript, Node.js.

Índice general

1. Introducción.....	1
1.1. Motivación y Contexto	1
1.2. Objetivo del Trabajo de Fin de Grado	2
1.3. Planificación del proyecto	3
1.4. Acerca del software de este trabajo	6
1.5. Estructura de la memoria	6
2. Estudio del estado del arte	9
2.1. Introducción	9
2.2. Tecnologías de comunicación inalámbricas	9
2.2.1. GPRS	9
2.2.2. WiFi.....	11
2.2.3. ZigBee	13
2.2.4. 6LoWPAN.....	14
2.2.5. Bluetooth LE	15
2.3. Dispositivos basados en microcontroladores	15
2.3.1. Raspberry Pi, Odroid, CubieBoard y derivados	16
2.3.2. Arduino.....	17
2.3.3. Zigduino	18
2.3.4. Waspmote.....	19
2.3.5. Teensy	20
2.3.6. Freescale Boards	21
2.4. Conclusión	21
3. Análisis de requisitos.....	23
3.1. Servicio REST	23
3.2. Estación meteorológica.....	23
3.2.1. Casos de uso.....	24
3.2.2. Selección de hardware	26
3.3. Cliente web	28
3.3.1. Autenticación	28

3.3.2. Casos de uso.....	28
4. Diseño del servicio REST	35
4.1. Introducción	35
4.2. Base de datos	36
4.3. API del servicio	38
5. Implementación y prueba del servicio.....	39
5.1. Dependencias	39
5.2. Pruebas.....	39
5.3. Funcionamiento	40
6. Diseño e integración de la estación meteorológica	43
6.1. Testeo de los componentes	43
6.1.1. Sensor BMP180	43
6.1.2. Sensor DHT22	46
6.2. Diseño de la estación meteorológica	50
6.3. Programación e integración	53
7. Estudio del consumo energético de la estación meteorológica	55
7.1. Descripción del estudio.....	55
7.2. Consumo energético	56
7.2.1. Consumo energético en modo de bajo consumo.....	56
7.2.2. Consumo energético de la estación meteorológica en funcionamiento	57
7.2.3. Consumo energético total	58
7.3. Posibles mejoras para reducir el consumo	58
8. Desarrollo de la aplicación web.....	61
8.1. Tecnologías empleadas	61
8.2. Autenticación	61
8.3. Funciones de la aplicación web	62
9. Conclusiones y trabajo futuro.....	71
9.1. Conclusiones	71
9.2. Trabajo futuro	72
A. API del servicio REST	73
B. Instalación del software en una Raspberry Pi	87

Bibliografia.....	91
-------------------	----

Índice de tablas

Tabla 1: Planificación temporal del proyecto.....	3
Tabla 2: Coste del hardware necesario.	3
Tabla 3: Velocidades de descarga y de subida de los sistemas GPRS.	10
Tabla 3: Consumo energético del módulo SM5100B.	11
Tabla 4. Consumo energético de la Shield Wizfi210	12
Tabla 5: Representación de 6LoWPAN en el modelo OSI.	14
Tabla 6: Consumo energético de Waspote.	19
Tabla 7: Comparativa de precios entre Arduino y Waspote.....	20
Tabla 8: Caso de uso 01.....	25
Tabla 9: Caso de uso 02.....	26
Tabla 10: Hardware seleccionado.....	26
Tabla 11: Caso de uso 03.....	29
Tabla 12: Caso de uso 04.....	30
Tabla 13: Caso de uso 05.....	30
Tabla 14: Caso de uso 06.....	30
Tabla 15: Caso de uso 07.....	31
Tabla 16: Caso de uso 08.....	31
Tabla 17: Caso de uso 09.....	32
Tabla 18: Caso de uso 10.....	32
Tabla 19: Caso de uso 11.....	32
Tabla 20: Caso de uso 12.....	33
Tabla 21: Caso de uso 13.....	33
Tabla 22: Caso de uso 14.....	33
Tabla 23: Grupos de usuarios.	36
Tabla 24: Esquema de la colección Group.	36
Tabla 25: Esquema de la colección User.....	36
Tabla 26: Esquema de la colección Person.	37
Tabla 27: Esquema de la colección Station.	37
Tabla 28: Esquema de la colección Sample.	37
Tabla 29: Consumo energético en modo de bajo consumo.	56
Tabla 30: Consumo energético en funcionamiento.	57
Tabla 31: Función GET /v1/login.....	73
Tabla 32: Función GET /v1/groups.....	74
Tabla 33: Función GET /v1/groups/:groupName.....	75
Tabla 34: Función GET /v1/groups/:groupName/users.....	75
Tabla 35: Función GET /v1/stations.....	76
Tabla 36: Función GET /v1/stations/:MAC.	77
Tabla 37: Función GET /v1/stations/:MAC/data.....	78

Tabla 38: Función GET /v1/stations/:MAC/data/:id.	79
Tabla 39: Función DELETE /v1/stations/:MAC/data/:id.	79
Tabla 40: Función GET /v1/users.	80
Tabla 41: Función POST /v1/users.	81
Tabla 42: Función PUT /v1/users/:login.	82
Tabla 43: Función DELETE /v1/users/:login.	82
Tabla 44: Función POST /v1/stations.	83
Tabla 45: Función DELETE /v1/stations/:MAC.	84
Tabla 46: Función PUT /v1/stations/:MAC.	85
Tabla 47: Función POST /v1/stations/:MAC/data.	85
Tabla 48: Contenido del fichero rest.service.	88
Tabla 49: Contenido del fichero web.service.	89

Índice de Figuras

Figura 1: Esquema del sistema.	2
Figura 2: Planificación inicial del proyecto.	4
Figura 3: Seguimiento del proyecto.	5
Figura 5: Raspberry Pi B+ vs Odroid C1.	16
Figura 6: Arduino UNO R3.	17
Figura 7: Zigduino.	18
Figura 8: Waspote	19
Figura 9: Teensy.	20
Figura 10: Freescale KL25Z.	21
Figura 11: Diagrama de flujo del funcionamiento de la estación.	24
Figura 12: Casos de uso de la estación.	25
Figura 13: ESP8266.	27
Figura 14: Adafruit CC3000 instalado en un Arduino Uno.	27
Figura 15: Casos de uso correspondientes a los roles de usuario normal y de administrador.	29
Figura 16: Ejecución de los tests.	40
Figura 17: Ejemplo de consulta de la API empleando Postman.	41
Figura 18: Sistema de logs empleando Scribe-JS.	42
Figura 19: BMP180 conectado a un Arduino Uno.	43
Figura 20: Montaje físico del sensor BMP180.	44
Figura 21: Código de prueba para el sensor BMP180.	45
Figura 22: Salida serial del sensor BMP180.	46
Figura 23: DHT22 conectado a un Arduino Uno.	47
Figura 24: Montaje físico del sensor DHT22.	48
Figura 25: Código de prueba para el sensor DHT22.	49
Figura 26: Salida serial del sensor DHT22.	50
Figura 27: Conexiones de la Shield CC3000.	51
Figura 28: Sensores y led soldados a la Shield CC3000.	52
Figura 29: Estación meteorológica.	53
Figura 30: Logs de la estación meteorológica enviando muestras.	54
Figura 31: Agilent N6705B.	55
Figura 32: Medición del consumo energético de la estación meteorológica.	56
Figura 33: Consumo energético en modo de bajo consumo.	57
Figura 34: Consumo energético de la estación en funcionamiento.	58
Figura 35: Hardware de la estación meteorológica mejorado.	59
Figura 36: Panel de login de la aplicación web.	62
Figura 37: Estaciones meteorológicas registradas en el sistema.	63
Figura 38: Panel para dar de alta una nueva estación.	63

Figura 39: Representación gráfica de los valores de la presión atmosférica.....	64
Figura 40: Representación gráfica de la variación de la temperatura.....	65
Figura 41: Visualización en texto plano de todas las muestras de la estación.	65
Figura 42: Parámetros de configuración de la estación.	66
Figura 43: Visualización de los usuarios del sistema.	67
Figura 44: Añadir un usuario al sistema.	67
Figura 45: Visualización de los datos de un usuario.	68
Figura 46: Grupos del sistema.	69
Figura 47: Usuarios del grupo User.....	69

Capítulo 1

Introducción

1.1. Motivación y Contexto

Las estaciones meteorológicas son dispositivos autónomos que se emplean para adquirir información ambiental a través de sus sensores (temperatura, humedad, presión atmosférica, etc.). Típicamente, la información recogida es enviada a un servidor para su almacenamiento y posterior procesamiento. También pueden ser presentados los datos en una pantalla en la propia estación.

La característica más destacada de las estaciones meteorológicas es la autonomía, que les permite funcionar de manera independiente, sin intervención humana, a lo largo del tiempo mientras poseen energía para hacerlo y su comportamiento no es erróneo.

Las estaciones meteorológicas son empleadas por organismos, empresas, individuos y máquinas con distintos fines. Algunos usos típicos son:

- Los gobiernos de multitud de países los emplean en las carreteras para recoger información ambiental e informar a los conductores de posibles riesgos.
- En el mundo empresarial se pueden observar este tipo de dispositivos en forma de termostato para regular la temperatura de una sala, CPD, industria, etc...
- Empresas que trabajan en el propio sector o que las emplean directamente para procesar datos ambientales y realizar investigaciones.
- Para uso privado, en el hogar.
- Cada vez es más común incluir este tipo de sensores en otros dispositivos, como por ejemplo, los automóviles. En la actualidad, muchos vehículos son capaces de encender automáticamente las luces de cruce, activar los limpiaparabrisas, encender las luces de niebla, detectar colisiones, etc... que no serían posibles de no ser por los sensores correspondientes.

En el mercado actual existen estaciones meteorológicas con diferentes rangos de precios y de características. La motivación de este proyecto es la de crear una estación meteorológica que se adecúe a las necesidades del cliente optimizando energía y costes.

1.2. Objetivo del Trabajo de Fin de Grado

El objetivo de este Trabajo de Fin de Grado es la creación de un prototipo de una estación meteorológica basada en hardware abierto que recoge información ambiental y la envía de forma inalámbrica a un servidor. Los usuarios podrán interactuar con el sistema mediante un cliente web. El sistema se divide en tres partes: la estación meteorológica, el servicio REST y el cliente web.

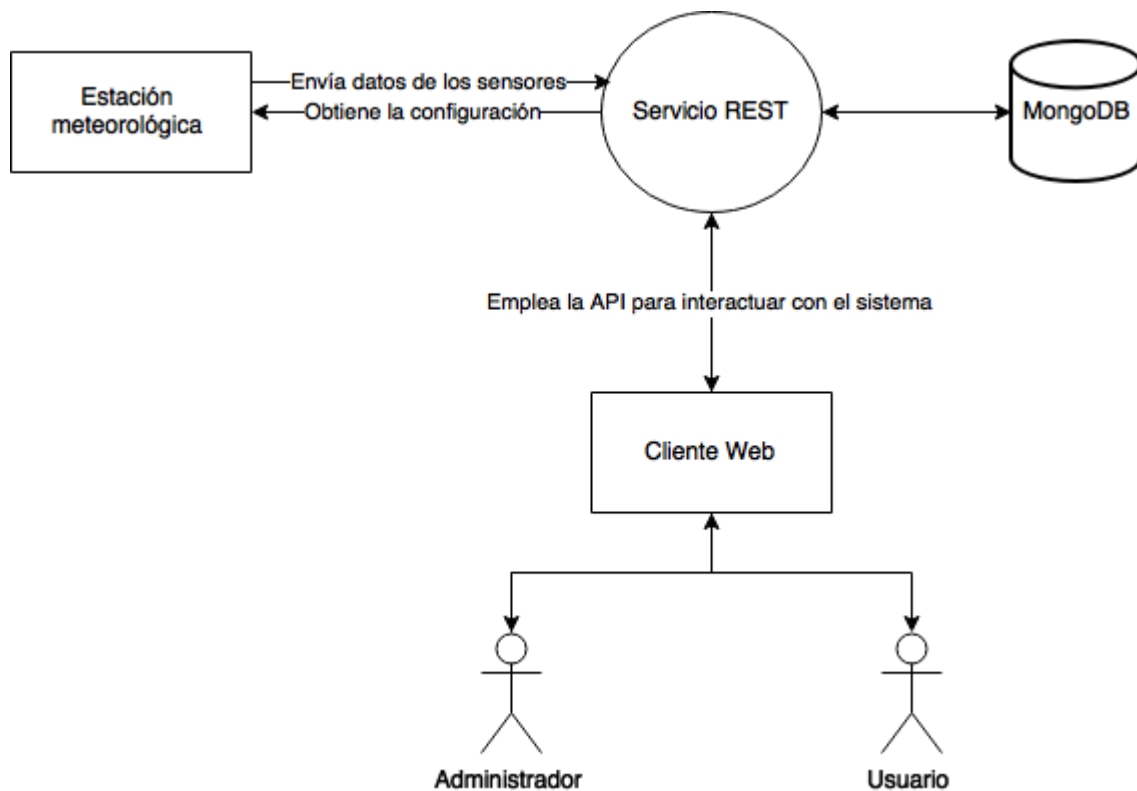


Figura 1: Esquema del sistema.

Para el desarrollo del Trabajo de Fin de Grado se empleará una metodología incremental basada en iteraciones:

1. Estudio del estado del arte.
2. Análisis de requisitos.
3. Diseño e implementación del servicio REST.
4. Diseño e programación de la estación meteorológica.
5. Estudio del consumo energético de la estación meteorológica.
6. Desarrollo de la aplicación Web.

1.3. Planificación del proyecto

En este apartado presentaremos la planificación inicial que se realizó inicialmente (ver Figura 2), el seguimiento del trabajo (ver Figura 3) y el presupuesto.

Realizando el seguimiento del trabajo se observaron retrasos en algunas tareas, provocando un retraso en el proyecto de 20 días.

	Trabajo	Comienzo	Fin
Previsto	194 horas	17/02/2015	01/07/2015
Real	275 horas	17/02/2015	27/08/2015

Tabla 1: Planificación temporal del proyecto.

Los costes del proyecto son todos debidos al hardware, en la tabla 2 se muestran los precios de todo el hardware necesario para la realización de este trabajo. El coste total del hardware utilizado fue de 78,16 €.

Elemento hardware	Precio
Adafruit CC3000 WiFi Shield	39,95 €
SMA to uFL/u.FL/IPX/IPEX RF Adapter Cable	3,95 €
Antena 2.4 GHz de 2.2 dBi	9,90 €
Arduino UNO R3 ATmega328 Compatible	3 €
Sensor de temperatura y humedad DHT22	4,36 €
Sensor de presión atmosférica BMP180	10,90 €
Resistencias	0,20 €
LEDs	0,90 €
Estañó para soldadura	5 €
Total	78,16 €

Tabla 2: Coste del hardware necesario.

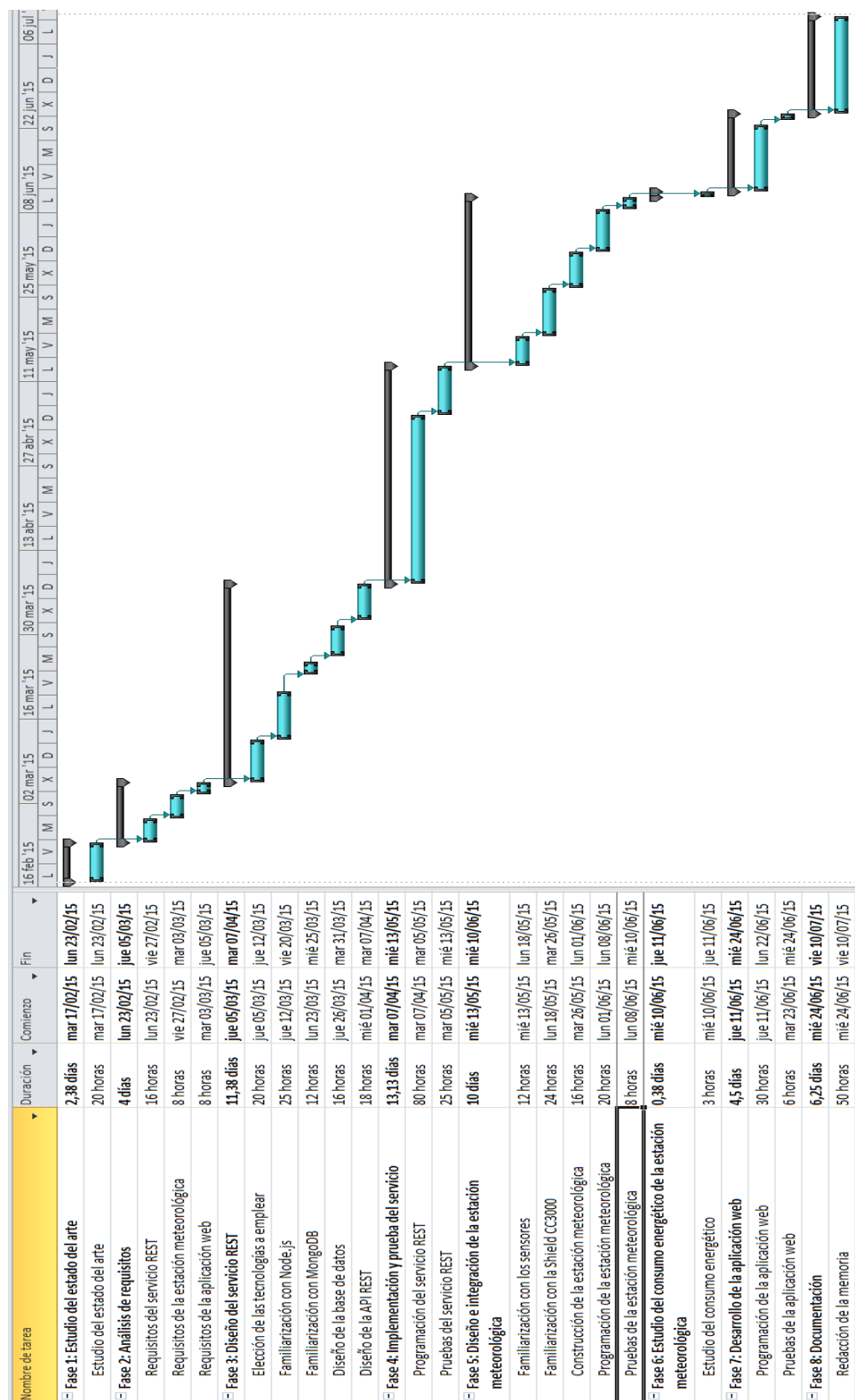


Figura 2: Planificación inicial del proyecto.

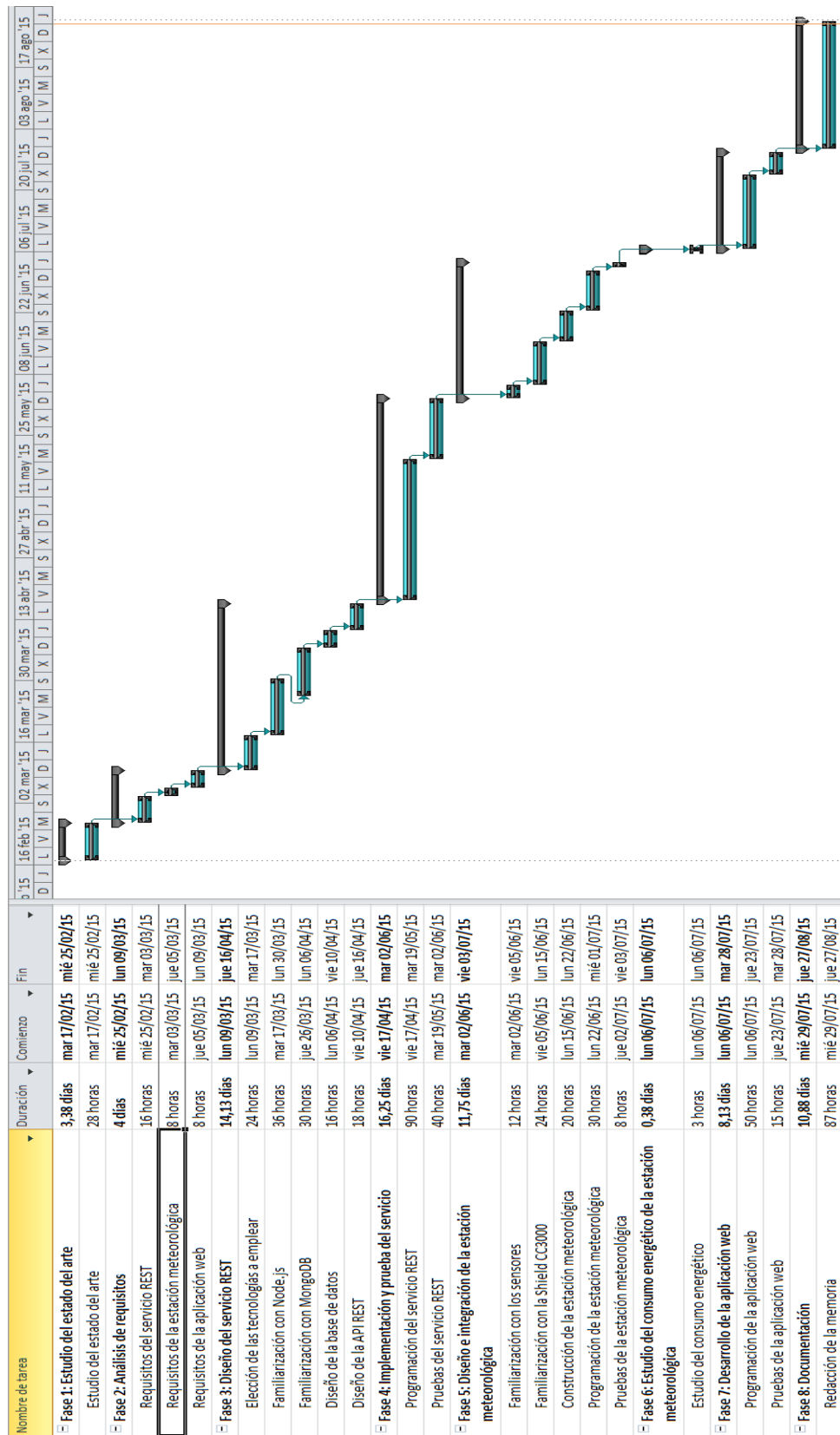


Figura 3: Seguimiento del proyecto.

1.4. Acerca del software de este trabajo

El software desarrollado en este trabajo de fin de grado está a la disposición del público en el repositorio de código del autor [0]. El código fuente está disponible bajo la licencia GPLv3.

1.5. Estructura de la memoria

En este apartado se muestran los capítulos y apéndices en los que se encuentra dividida la memoria, junto una breve explicación del contenido de cada uno.

1. **Introducción:** En este primer capítulo se comenta el contexto de la situación actual, los objetivos que se quieren alcanzar y la estructura de esta memoria.
2. **Estudio del estado del arte:** En este capítulo se realiza una investigación sobre las tecnologías actuales de comunicación inalámbrica y las placas basadas en microcontroladores. También, se realiza una valoración de las características de cada una y finalmente se escoge la tecnología de comunicación inalámbrica y el microcontrolador a emplear en el proyecto.
3. **Análisis de requisitos:** En este capítulo se detallan los requisitos del proyecto, funcionamiento de la estación y los casos de uso de cada parte.
4. **Diseño del servicio REST:** En este capítulo se describe la arquitectura del servicio así como las tecnologías empleadas para su desarrollo. También se habla del diseño de la base de datos y sus ventajas frente a otras soluciones.
5. **Implementación y prueba del servicio:** En este capítulo se especifican las dependencias del servicio REST y se habla sobre las pruebas y el funcionamiento del servicio.
6. **Diseño e integración de la estación meteorológica:** En este capítulo se realizan los testeos de los sensores, el diseño de la estación meteorológica, su construcción y programación.

7. **Estudio del consumo energético de la estación meteorológica:** En este capítulo se realiza un análisis del consumo energético empleando un analizador de potencia Agilent N6705B.
 8. **Desarrollo de la aplicación web:** En este capítulo se detallan las tecnologías empleadas para el desarrollo de la web y se describe su funcionamiento.
 9. **Conclusiones y trabajo futuro:** En este capítulo se habla de las principales conclusiones que se han obtenido fruto del trabajo y de posibles vías de desarrollo futuras.
- A. **API del servicio REST:** En este apéndice se detalla en profundidad la API del servicio REST.
 - B. **Instalación del software en una Raspberry Pi:** En este apéndice se describe el proceso de instalación del software desarrollado en una placa basada en microcontrolador como es la Raspberry Pi. El software se instalará como un *daemon* del sistema que se ejecutará en orden según sus dependencias con otros servicios.

Capítulo 2

Estudio del estado del arte

2.1. Introducción

En este capítulo se realizará un estudio sobre el estado de las tecnologías actuales en cuanto a tecnologías de comunicación inalámbricas y placas basadas en microcontroladores. La elección de la tecnología a emplear se realizará en función de nuestras necesidades y los requisitos del sistema, analizando sus ventajas y desventajas.

2.2. Tecnologías de comunicación inalámbricas

Actualmente existe en el mercado un conjunto de tecnologías inalámbricas destinadas a diferentes usos y entornos. Las prioridades que se tendrán en cuenta a la hora de seleccionar una tecnología son:

1. Consumo energético.
2. Coste.
3. Conocimientos sobre la tecnología y dificultad de despliegue.
4. Alcance.
5. Seguridad.

2.2.1. GPRS

General Packet Radio Service (GPRS) [1] es una extensión del Sistema Global para Comunicaciones Móviles (Global System for Mobile Communications o GSM) para la transmisión de datos mediante conmutación de paquetes.

La tecnología GPRS permite acceso a servicios como el Protocolo de Aplicaciones Inalámbrico (WAP), el servicio de mensajes cortos (SMS) y multimedia (MMS), acceso a Internet y correo electrónico.

Habitualmente, las transferencias de datos empleando GPRS se cobran por megabytes de transferencia, mientras que el pago de la comunicación tradicional mediante conmutación de circuitos se cobra por tiempo de conexión, independientemente de si el usuario está utilizando el canal o no.

Dependiendo de la tecnología empleada, la velocidad de transferencia varía sensiblemente. La siguiente tabla muestra las velocidades de bajada y de subida de datos para cada tipo de tecnología.

Tecnología	Descarga (kbit/s)	Subida (kbit/s)
CSD	9.6	9.6
HSCSD	28.8	14.4
HSCSD	43.2	14.4
GPRS	80.0	20.0 (Clase 8, 10 y CS-4)
GPRS	60.0	40.0 (Clase 10 y CS-4)
EGPRS (EDGE)	236.8	59.2 (Clase 8, 10 y MCS-9)
EGPRS (EDGE)	177.6	118.4 (Clase 10 y MCS-9)

Tabla 3: Velocidades de descarga y de subida de los sistemas GPRS.

En cuanto a la usabilidad de GRPS, dispone de una amplia cobertura desde casi cualquier localización, por lo que la distancia no supondría un problema para nuestro sistema. Por otro lado, esta tecnología presenta una alta latencia en las comunicaciones, típicamente sobre 600 – 700 ms llegando a alcanzar 1 s en algunas situaciones. La latencia es un factor importante a tener en cuenta, pero para este sistema no supondría un gran inconveniente debido a la baja frecuencia de las transmisiones y el reducido tamaño de los paquetes. También hay que tener en cuenta el coste asociado de la tarjeta SIM y el consumo de datos del proveedor de la red.

Los módulos GSM/GPRS comerciales tienen diferentes consumos energéticos según el modo en el que se encuentre funcionando. Los consumos energéticos también pueden variar según la antena que se esté empleando, ya que si la antena no tiene una ganancia suficiente, el módulo GSM/GPRS intenta aumentar la potencia de transmisión provocando un mayor consumo energético.

En la siguiente tabla se muestran los consumos típicos del módulo GSM/GPRS SM5100B [2]:

Power consumption GSM/GPRS module SM5100B	
Off mode	< 100 uA
Sleep mode	< 2 mA
Idle mode	< 7 mA (average)
Communications mode	350 mA (average)
Communications mode	2 A (Typical peak during Tx slot, GSM)

Tabla 4: Consumo energético del módulo SM5100B.

Existen otras alternativas como Arduino GSM Shield, SIM900 o CC3000, que tienen consumos similares. En cuanto al coste, los precios parten de los 50€ por módulo.

2.2.2. WiFi

WiFi (Wi-Fi, Wireless Fidelity) es una tecnología inalámbrica de red de área local que permite que un dispositivo pueda comunicarse con otros empleando la banda de frecuencias ISM (del inglés, Industrial, Scientific and Medical) de 2.4 GHz y 5 GHz. Esta tecnología está definida de acuerdo a la familia de estándares IEEE 802.11 [3] y es compatible con todos los servicios de las redes locales (LAN) definidas en el estándar IEEE 802.3 (Ethernet) [4].

Existen distintas implementaciones del estándar IEEE 802.11, algunas de ellas:

- IEEE 802.11b, IEEE 802.11g e IEEE 802.11n poseen una aceptación internacional debido a que la banda de 2.4 GHz está disponible casi universalmente.
- Actualmente, el nuevo estándar IEEE 802.11ac está siendo implantado en la mayoría de los nuevos dispositivos.

Un problema común a tener en cuenta cuando se despliegan redes WiFi es la saturación del espectro radioeléctrico, debido a la masificación de usuarios, esta

situación afecta especialmente a las conexiones de larga distancia (más de 100 metros). En realidad, el estándar WiFi está diseñado para conectar ordenadores a la red a distancias reducidas, cualquier uso de mayor alcance está expuesto a un excesivo riesgo de interferencias.

También se ha de tener en cuenta la seguridad de las redes WiFi [5], ya que la información se transmite por un medio compartido, como es el aire, por lo que cualquier otro dispositivo puede estar capturando los datos transmitidos. Para evitar esta situación surgen una serie de medidas o estándares de cifrado para asegurar la confidencialidad de estas redes:

- WEP (Wired Equivalente Privacy).
- WPA (Wi-Fi Protected Access) y WPA2.
- IPSec y autenticación IEEE 802.1X.
- Filtrado por MAC.
- Ocultación del punto de acceso.

Una Shield interesante es Wizfi210 [6], que está basada en comunicaciones serie y es compatible con Arduino y clones. Soporta redes WiFi 802.11n, alcanza hasta 11 Mbps y es compatible con todos los tipos de cifrado (e.g. WEP y WPA/WPA2-PSK). También dispone de un conector coaxial para una antena externa (lo que lleva asociado una mejora de la cobertura y, por lo tanto, del consumo energético) y soporta los protocolos de red más habituales, entre ellos, HTTP y HTTPS.

Los módulos WiFi comerciales para placas basadas en microcontroladores tienen diferentes consumos energéticos, dependiendo del modo en el que se encuentren funcionando. Para el caso de la Shield Wizfi210 dichos consumos se muestran en la siguiente tabla.

Consumo energético de la Shield Wizfi210	
Standby	34.0 μ A
Receive	125.0 mA
Transmit	135.0 mA

Tabla 5. Consumo energético de la Shield Wizfi210

2.2.3. ZigBee

ZigBee es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radiodifusión digital de bajo consumo, basada en el estándar IEEE 802.15.4 [7] de redes inalámbricas de área personal *Wireless Personal Area Network*, WPAN). Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías.

Esta tecnología define tres tipos de dispositivos [8]:

- Coordinador ZigBee. El tipo de dispositivo más completo, debe existir al menos uno por red. Su principal función es la de controlar la red y los caminos que deben seguir los dispositivos para conectarse entre ellos.
- Router ZigBee. Interconecta dispositivos separados en la topología de la red, además de ofrecer un nivel de aplicación para la ejecución de código de usuario.
- Dispositivo final. Posee la funcionalidad necesaria para comunicarse con su nodo padre (el coordinador o un router), pero no puede transmitir información destinada a otros dispositivos. De esta forma, este tipo de nodo puede estar dormido la mayor parte del tiempo, aumentando la vida media de sus baterías.

ZigBee permite tres topologías de red:

- Topología en estrella: el coordinador se sitúa en el centro.
- Topología en árbol: el coordinador será la raíz del árbol.
- Topología en malla: al menos uno de los nodos tendrá más de dos conexiones.

Las redes ZigBee han sido diseñadas para conservar la potencia en los nodos “esclavos”. De esta forma se consigue un bajo consumo de energía.

La estrategia consiste en que, durante mucho tiempo, un dispositivo “esclavo” está en modo “dormido”, de tal forma que solo se “despierta” por una fracción de segundo para confirmar que está “vivo” en la red de dispositivos de la que forma parte.

El módulo XBee [9] tiene una antena de 2 mW y permite alcanzar distancias de hasta 100 metros. También hay que mencionar su bajo consumo, de tan solo 50 mA, y su bajo coste, unos 22 €. Por otro lado, sería necesario poseer un nodo coordinador en el sistema y que opere como Gateway para traducir los mensajes de la red ZigBee a peticiones HTTP.

2.2.4. 6LoWPAN

6LoWPAN (IPv6 over Low Power Wireless Personal Area Networks) [10] es un estándar que posibilita el uso de IPv6 sobre redes basadas en el estándar IEE 802.15.4. Hace posible que dispositivos como los nodos de una red inalámbrica puedan comunicarse directamente con otros dispositivos IP.

Junto con este estándar surge otro protocolo conocido como CoAP (Constrained Application Protocol) [11] que permite que los dispositivos se comuniquen con servicios REST, utilizando UDP como capa de transporte. CoAP presenta una API idéntica a REST-HTTP, es decir, el conjunto de operaciones CRUD (Create, Read, Update y Delete), pero el transporte se realiza sobre el protocolo UDP, no orientado a conexión y no fiable.

CoAP permite dos tipos de mensajes: confirmados y no confirmados. Los mensajes confirmados suponen una aceptación por parte del receptor (ACK), de tal manera que, si se pierden, se retransmiten un número limitado de veces. Los mensajes no confirmados se pueden emplear para aquellos casos en que sea preferible perder información periódica de algunos sensores antes que saturar la red con retransmisiones, algo que no es posible con el modelo REST-HTTP.

Este conjunto de protocolos y tecnologías se situarían de la siguiente manera en el modelo OSI:

IEEE 802.15.4	Para el nivel radio (capa física y de enlace, niveles 1 y 2 OSI)
6LoWPAN	Para el nivel de red (capa de red, nivel 3 OSI)
UDP	Para el nivel de transporte (nivel 4 OSI)
CoAP	Ofrece una API REST y capacidades de reenvío para mensajes confirmados.

Tabla 6: Representación de 6LoWPAN en el modelo OSI.

6LoWPAN/CoAP presenta todas las ventajas de ZigBee e incluye la comunicación con servidores REST de forma transparente. Por otro lado, es una tecnología experimental y con poca documentación. Además es necesario un Gateway 6LoWPAN en la red del sistema.

Existe una plataforma conocida como Wasmote Mote Runner 6LoWPAN Development Platform [12] pensada para este tipo de aplicaciones. Por desgracia, los kits Wasmote son bastante caros.

2.2.5. Bluetooth LE

Bluetooth LE (Bluetooth Low Energy) [13] es una tecnología de radio (inalámbrica) que ofrece comunicación entre dispositivos móviles o computadores y otros dispositivos más pequeños. Está diseñada específicamente para funcionar consumiendo poca energía.

Bluetooth LE opera en el mismo rango de espectro (entre 2.400 GHz y 2.4835 GHz) que la tecnología clásica de Bluetooth, pero utiliza un conjunto diferente de canales. El ancho de banda es de 1 Mbit/s y el consumo máximo de potencia transmitiendo es de 10 mW.

En cuanto a la distancia de transmisión, su máxima teórica es de más de 100 metros. Dispone de cifrado AES de 128 bits para las comunicaciones y tiene una latencia muy baja en comparación con el Bluetooth clásico: 6 ms de latencia para pasar de un estado desconectado a conectado y 3 ms de tiempo mínimo total para enviar datos. También tiene un consumo pico menor de 15 mA, lo que lo hace idóneo para todo tipo de aplicaciones. Se han realizado estudios sobre su impacto en las baterías y un beacon conectado a una batería de 1000 mAh podría funcionar durante 1-2 años.

Los módulos Bluetooth LE suelen tener reducidas dimensiones y un bajo precio, unos 20 €, como es el caso de Bluefruit LE [14]. Es fácilmente programable y cuenta con las ventajas propias de Bluetooth LE. Sin embargo, tendríamos un sistema similar al de ZigBee y 6LoWPAN, se necesitaría un receptor Bluetooth que convierta los mensajes a peticiones REST.

2.3. Dispositivos basados en microcontroladores

A continuación se analizan una serie de dispositivos basados en microcontroladores candidatos a formar parte de la estación meteorológica del proyecto. Las características que han de poseer son: eficiencia energética, bajo coste y soporte.

2.3.1. Raspberry Pi, Odroid, CubieBoard y derivados

La Raspberry Pi [15], Odroid [16] y CubieBoard [17] son placas basadas en microcontroladores de propósito general, es decir, pueden ser utilizadas para controlar un pequeño robot o ser un computador personal de bajos recursos.

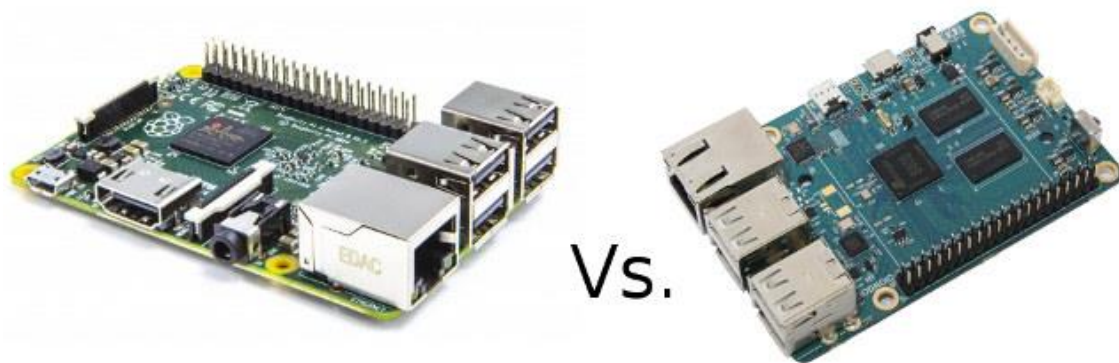


Figura 4: Raspberry Pi B+ vs Odroid C1.

La Raspberry Pi es la placa más popular entre este tipo de dispositivos y posee un bajo precio (entre 22 y 33 €) junto con un gran número de periféricos compatibles y una gran comunidad que mantiene el software.

Los dispositivos Odroid suelen tener una mayor potencia de cómputo a cambio de incrementar el precio final. Los periféricos de la Raspberry Pi suelen ser compatibles con los módulos Odroid e incluso tiene periféricos propios. En cuanto a la comunidad, no tiene tanta presencia como la de la Raspberry Pi.

Finalmente, tenemos una gran variedad de dispositivos similares como pueden ser: CubieBoard, Banana Pi, BeagleBone, PandaBoard, etc.

Esta serie de dispositivos nos otorgan una gran versatilidad en cuanto a potencia de cómputo, software, periféricos, etc... pero su punto débil es el consumo energético. Por ejemplo, la Raspberry Pi A con un precio de 23 € tiene un consumo medio de unos 400 mAh.

2.3.2. Arduino

Arduino [18] es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo. Existe una gran variedad de modelos, que se adaptan a las necesidades según el tamaño, potencia de cómputo, número de periféricos, consumo energético, etc...

- Arduino Uno.
- Arduino Due.
- Arduino Yún.
- Arduino Nano.
- Arduino Mega.



Figura 5: Arduino UNO R3.

También dispone de una serie de placas de expansión o Shields que permiten ampliar las capacidades del Arduino.

- Arduino GSM Shield.
- Arduino Ethernet Shield.
- Arduino WiFi Shield.
- Arduino Motor Shield.
- Arduino Proto Shield.
- Arduino USB Host Shield.
- Arduino Wireless SD Shield.
- Arduino Wireless Proto Shield.

El entorno de programación de Arduino permite utilizar diversas librerías que facilitan las tareas de desarrollo. Como lenguaje de programación se emplea Processing, un lenguaje de programación de alto nivel similar a C++.

En cuanto al consumo energético, los Arduino más básicos, suelen consumir unos 40 mA y pueden emplear modos de bajo consumo para reducir esa cifra. El precio oficial de estos dispositivos oscila entre los 15 y los 90 €, aunque existen modelos clónicos, a partir de 2 €.

2.3.3. Zigduino

Zigduino [19] es una plataforma hardware basada en microcontroladores compatibles con Arduino que integra un sistema de radiofrecuencia basado en el estándar IEEE 802.15.4. Este puede ser configurado para cualquier protocolo del estándar 802.15.4, incluyendo ZigBee y 6LoWPAN.

Dispone de una antena instalada en la superficie del dispositivo, permitiendo la instalación de Shields compatibles con Arduino.

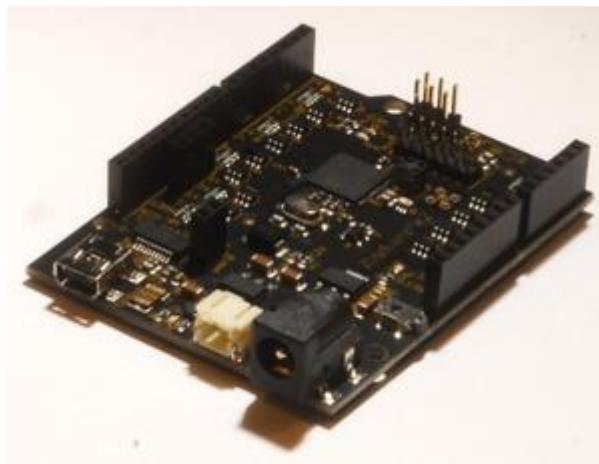


Figura 6: Zigduino.

Este dispositivo puede ser una gran alternativa a Arduino si nos decantamos por ZigBee o 6LoWPAN en cuanto a tecnología de comunicación inalámbrica ya que viene integrada en el dispositivo, permitiendo reducir costes y facilitando la programación. Su precio es de unos 70 € y posee un consumo energético similar al de Arduino.

2.3.4. Wasmote

La plataforma Wasmote [20] es una arquitectura hardware especialmente diseñada para aplicaciones de ultra bajo consumo. Posee switches digitales que permiten apagar cualquier interfaz de los sensores para ahorrar energía de forma eficiente.



Figura 7: Wasmote

Consumo energético	
ON	15 mA
Sleep	55 μ A
Deep Sleep	55 μ A
Hibernate	0.07 μ A

Tabla 7: Consumo energético de Wasmote.

En cuanto a la programación de Wasmote, utiliza el mismo IDE (compilador y librerías) que Arduino, y es compatible a nivel binario. A mayores, dispone de un modo de actualización OTAP (Over the Air Programming) del programa almacenado en la memoria flash de forma inalámbrica empleando redes ZigBee u otro protocolo del estándar IEEE 802.15.4.

La plataforma Waspnote es software libre pero posee un precio algo mayor que Arduino ya que está destinada a un ambiente más profesional. La siguiente tabla muestra una comparativa de precios.

	Arduino UNO	Arduino Mega 2560	Waspnote
Board	22,00 €	41,00 €	155,00 €
Arduino Xbee 802.15.4 + 2dBi antenna	45,00 €	45,00 €	
Triple axis accelerometer	7,75 €	7,75 €	
On Board Programmable LED + ON/OFF Switch	1,00 €	1,00 €	
RTC DS3234 + Button Battery	16,00 €	16,00 €	
uSD Adaptor	20,00 €	20,00 €	
Solar Panel Socket	47,00 €	47,00 €	30,00 €
6600mAh Battery			
Total	158,75 €	177,75 €	185,00 €

Tabla 8: Comparativa de precios entre Arduino y Waspnote.

2.3.5. Teensy

Teensy [21] es un sistema de desarrollo basado en microcontroladores de pequeñas dimensiones que permite implementar todo tipo de proyectos.



Figura 8: Teensy.

La carga del programa se realiza mediante el cable USB y permite cargar programas escritos en C. Su precio oscila entre los 10 y los 20 €, con unas capacidades de cómputo similares a las de Arduino. Su consumo energético es similar al de Arduino.

Una desventaja es la falta de documentación sobre esta plataforma hardware y la limitada comunidad de usuarios que posee.

2.3.6. Freescale Boards

Las placas de desarrollo Freescale [22] incorporan procesadores ARM Cortex M0+ de bajo consumo y están destinados al aprendizaje y familiarización con la arquitectura. La programación y depuración se realiza a través de un IDE llamado Keil, que permite desarrollar sistemas en C y ensamblador ARM.

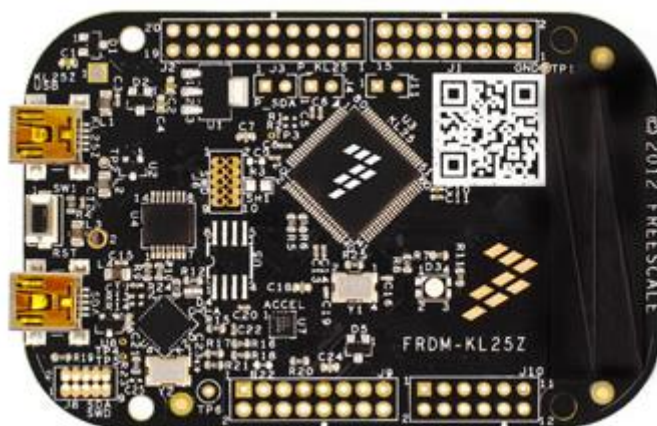


Figura 9: Freescale KL25Z.

Posee un consumo de 20 mA, pero este puede ser reducido a unos pocos mA desactivando características de la placa [23] que no se van a utilizar. En cuanto al precio, los modelos básicos parten de los 15 €.

2.4. Conclusión

En cuanto a las tecnologías inalámbricas:

- GPRS dispone de unas características muy interesantes como son el alcance y el acceso directo a Internet. Pero el requisito fundamental para nuestro sistema no lo cumple, que es el consumo energético. Con un consumo medio de 350 mA durante las transmisiones y picos de hasta 2 A lo descarta

totalmente como tecnología de comunicación inalámbrica para el prototipo del proyecto.

- 6LoWPAN es una tecnología prometedora y que requeriría un trabajo de investigación y un profundo conocimiento del estándar IEEE 802.15.4. Además, sería necesario un Gateway 6LoWPAN que gestione las comunicaciones entre las estaciones meteorológicas y el servidor REST.
- Bluetooth LE posee características deseables como bajo coste y bajo consumo energético pero presenta dos problemas: programación algo compleja y requiere de un nodo receptor que trabaje como Gateway.
- ZigBee posee un consumo energético mucho más eficiente que WiFi y lo hace idóneo para el prototipo. No obstante, los módulos ZigBee también requieren de un nodo receptor que funcione como Gateway y, a día de hoy, no es una tecnología inalámbrica tan extendida como lo son Bluetooth LE o WiFi. Debido a esto, y como se va a realizar un prototipo, no la versión final de un posible producto, se empleará WiFi como tecnología de comunicación inalámbrica.

Debido a la elección de WiFi como tecnología de comunicación inalámbrica, los dispositivos Zigduino quedan descartados debido a que incorporan conectividad ZigBee. En cuanto a la Raspberry Pi, Odroid, CubieBoard y derivados no son idóneos para el sistema debido a su alto consumo energético.

Las placas basadas en Teensy poseen poca documentación y comunidad de usuarios, por lo que supone un riesgo para el proyecto.

Las placas Freescale, como la KL25Z, pueden ser una buena elección ya que tienen un bajo coste, en cambio el IDE Keil es de pago y tiene una serie de limitaciones para el desarrollo del proyecto. Ante esto y que la programación es en ensamblador ARM y C, el desarrollo resultará más complejo y tedioso que en otras plataformas, por lo que se descarta como opción.

Finalmente, Arduino y Waspote son dos opciones totalmente válidas. Quizás Waspote proporcione mejores resultados, pero su precio y su reducida comunidad de usuarios son un problema. Por este motivo se escoge Arduino como plataforma de desarrollo y WiFi como sistema de comunicación inalámbrico.

Capítulo 3

Análisis de requisitos

3.1. Servicio REST

El servicio REST será el modelo del sistema e implementará todas las funciones y políticas de seguridad necesarias para el correcto funcionamiento del sistema. Las peticiones REST tendrán autenticación básica [24] empleando la cabecera propia del protocolo HTTP. Los usuarios del sistema emplearán su login y contraseña, mientras que las estaciones meteorológicas utilizarán su MAC (o identificador único) y como contraseña la MAC al revés.

Otro requisito que se impondrá desde un principio es que soporte diferentes versiones del sistema, es decir, en la URL de las peticiones de la API se establecerá el número de la versión. De esta manera, clientes antiguos podrán seguir interactuando con el sistema.

3.2. Estación meteorológica

La autonomía de la estación meteorológica es su requisito fundamental, pudiendo funcionar de forma autónoma sin intervención humana. Para lograr ese objetivo, la estación será programada para estar siempre en modo de bajo consumo (ver Figura 11), excepto cuando tenga que realizar la adquisición y envío de los valores de las variables meteorológicas.

La estación meteorológica tendrá varios parámetros configurables: localización (nombre de la ubicación o coordenadas), el nombre y la contraseña de la red WiFi a la que conectarse, dirección IP y puerto del servidor REST dentro de la red WiFi y el intervalo de espera entre transmisiones (tiempo en segundos en modo de bajo consumo).

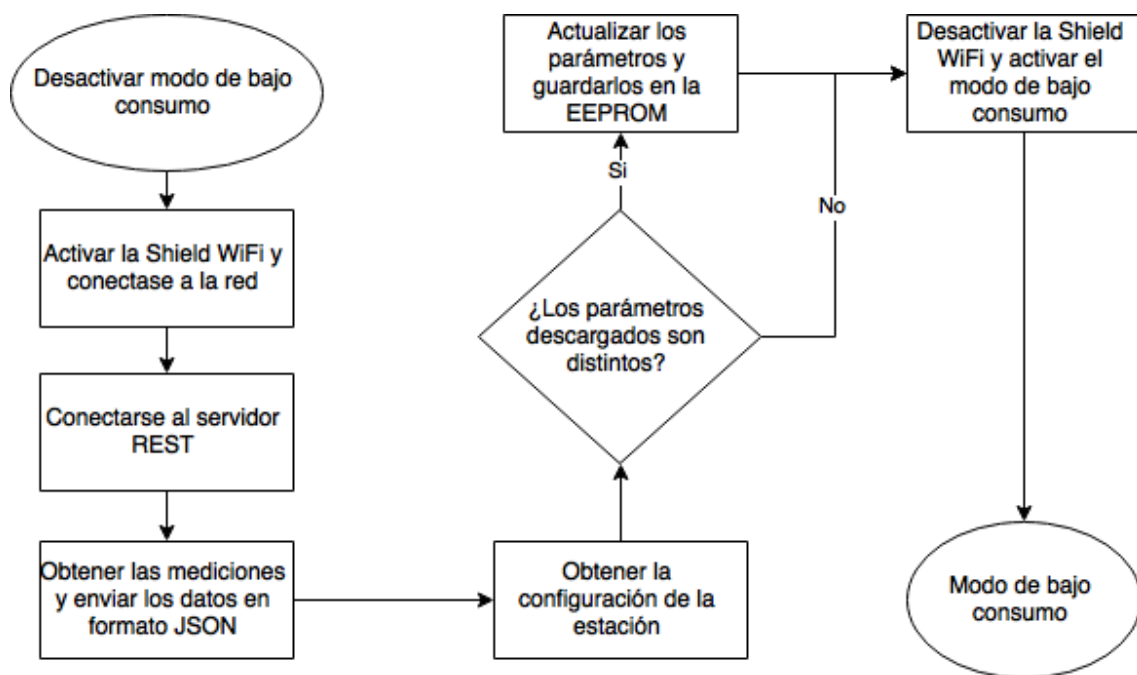


Figura 10: Diagrama de flujo del funcionamiento de la estación.

Otras características de la estación meteorológica a tener en cuenta son:

- Bajo coste.
- Exactitud y precisión en las mediciones.
- Comunicaciones inalámbricas seguras.

Las comunicaciones van a disponer de autenticación, pero cuando se habla de seguridad se refiere al cifrado de los datos. Lo ideal sería emplear soluciones estándar cuyo funcionamiento esté garantizado como puede ser HTTPS, pero no todos los dispositivos basados en microcontrolador cuentan con la potencia de cómputo necesaria para este tipo de cifrado.

3.2.1. Casos de uso

A continuación se describen los casos de uso de la estación meteorológica (ver Fig. 9) que se modelan como funciones de la API del servicio REST:

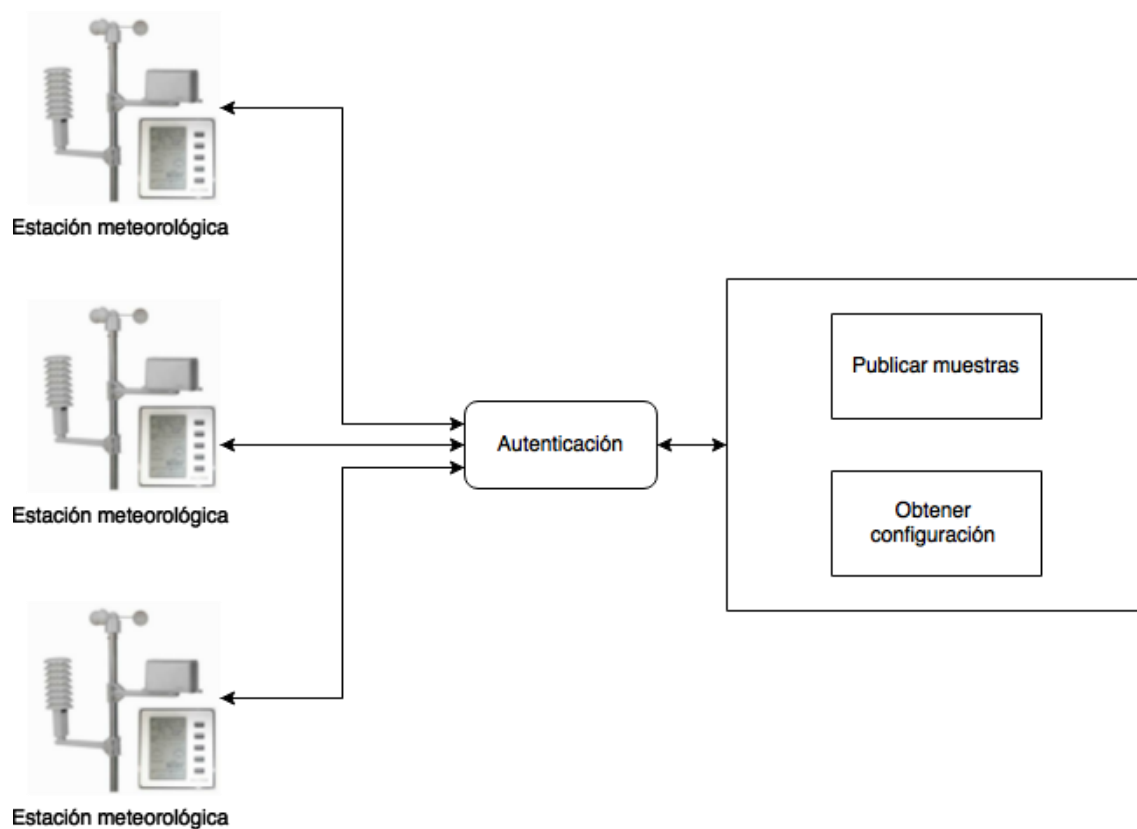


Figura 11: Casos de uso de la estación.

CU: 01	Publicar muestras.	
Descripción:	Se envían los datos recogidos por los sensores al servidor REST.	
Precondición:	Conectividad con el servidor REST.	
Secuencia normal:	Paso	Acción
	1	Obtener los datos de los sensores y codificarlos en formato JSON.
	2	Enviar la petición REST con las muestras.
	3	El servidor REST procesa la petición y responde con el código de estado correspondiente.
Excepciones:	Paso	Descripción
	2	Fallo en la conexión.
	3	Petición errónea.

Tabla 9: Caso de uso 01.

CU: 02	Obtener configuración.	
Descripción:	Se solicita al servidor REST la configuración para la estación meteorológica.	
Precondición:	Conectividad con el servidor REST.	

Secuencia normal:	Paso	Acción
	1	La estación meteorológica envía una petición REST para solicitar su configuración actual.
	2	El servidor REST responde con la configuración de la estación en formato JSON.
	3	La estación meteorológica procesa la configuración y actualiza sus valores si es necesario.
Excepciones:	Paso	Descripción
	1	Fallo en la conexión.
	2	Petición errónea.
	3	La configuración JSON tiene un formato inválido.

Tabla 10: Caso de uso 02.

3.2.2. Selección de hardware

En este apartado se procederá a detallar el hardware seleccionado para construir la estación meteorológica, reflejando el precio de venta al público de cada elemento tal y como se muestra en la Tabla 10:

Elemento	Precio
Adafruit CC3000 WiFi Shield	39,95 €
SMA to uFL/u.FL/IPX/IPEX RF Adapter Cable	3,95 €
Antena 2.4 GHz de 2.2 dBi	9,90 €
Arduino UNO R3 ATmega328 Compatible	3 €
Sensor de temperatura y humedad DHT22	4,36 €
Sensor de presión atmosférica BMP180	10,90 €
Coste total	72,06 €

Tabla 11: Hardware seleccionado.

Como Shield WiFi nos hemos decantado por la solución de Adafruit, ya que fabrica hardware de gran calidad, tiene un buen soporte software con tutoriales y dispone de una buena comunidad.

Una gran alternativa habría sido emplear un ESP8266 [25] (ver Fig. 10). Este pequeño módulo WiFi cuenta con una gran potencia de cómputo ya que dispone de una CPU de 32 bits que puede ser programada directamente, sin necesidad de depender de un Arduino para controlarla. Además, tiene un consumo reducido y un coste de unos 7 € por unidad. Por otro lado, es un producto que cuenta con un software inmaduro y errático, pero con un futuro prometedor.

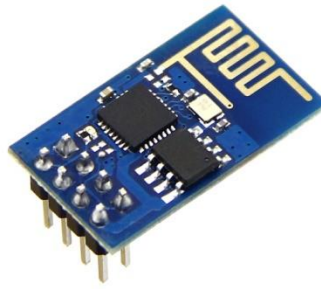


Figura 12: ESP8266.

Debido al estado del software del ESP8266, nos decantamos por emplear la placa Adafruit CC3000, debido a su facilidad de uso y el soporte con el que cuenta.

La Shield Adafruit CC3000 [26] (ver Fig. 11) que seleccionamos no cuenta con antena integrada, por lo que necesita un adaptador y una antena.

Como cerebro de la estación se empleará un Arduino UNO R3 compatible. Se ha decidido emplear el Arduino UNO para instalar la Shield Wifi sobre él, ya que sus pines lo permiten.

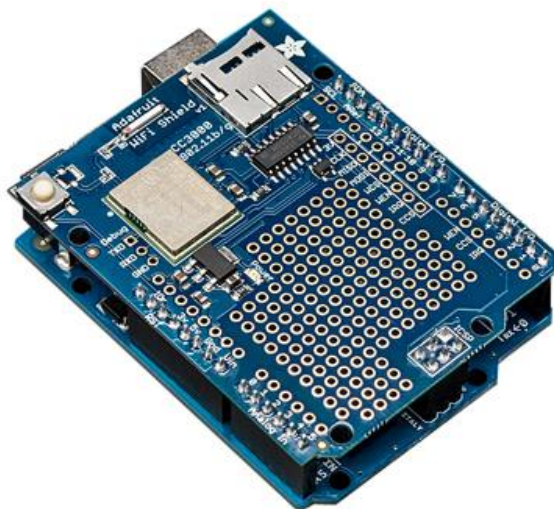


Figura 13: Adafruit CC3000 instalado en un Arduino Uno.

Para medir la temperatura y humedad se empleará un sensor DHT22 [27] de gran precisión, con un margen de error de 0.5°C y 2-5% en cuanto al porcentaje de humedad. El sensor BMP180 [28] nos permitirá obtener la presión atmosférica con un margen de error de 0.03hPa.

3.3. Cliente web

El cliente web proporciona la interfaz de usuario con la que interactuar con el servicio REST y la estación meteorológica. Estará desarrollado empleando los estándares web actuales y su interfaz se deberá visualizar correctamente en los navegadores modernos.

3.3.1. Autenticación

La autenticación es una prioridad con la que se trabajará desde un primer momento. No es posible realizar ningún caso de uso sin estar autenticado correctamente.

En el sistema se distinguen tres grupos de usuarios: usuarios normales, usuarios con permisos de administrador y estaciones meteorológicas. El motivo de realizar esta distinción es porque hay tareas que no pueden ser realizadas por cualquier usuario, por ejemplo, dar de baja una estación meteorológica del sistema.

No es una prioridad añadir una compleja lógica de grupos de usuarios con distintos roles, por lo que tan sólo existirán esos tres, pero se implementará de tal forma que en un futuro puedan existir más.

3.3.2. Casos de uso

A continuación se realizará un análisis de los casos de uso que se requieren por parte del cliente web (ver Fig. 12). Se inicia el análisis partiendo de las necesidades reales de un usuario y se continuará con el desglose para obtener los requisitos más técnicos.

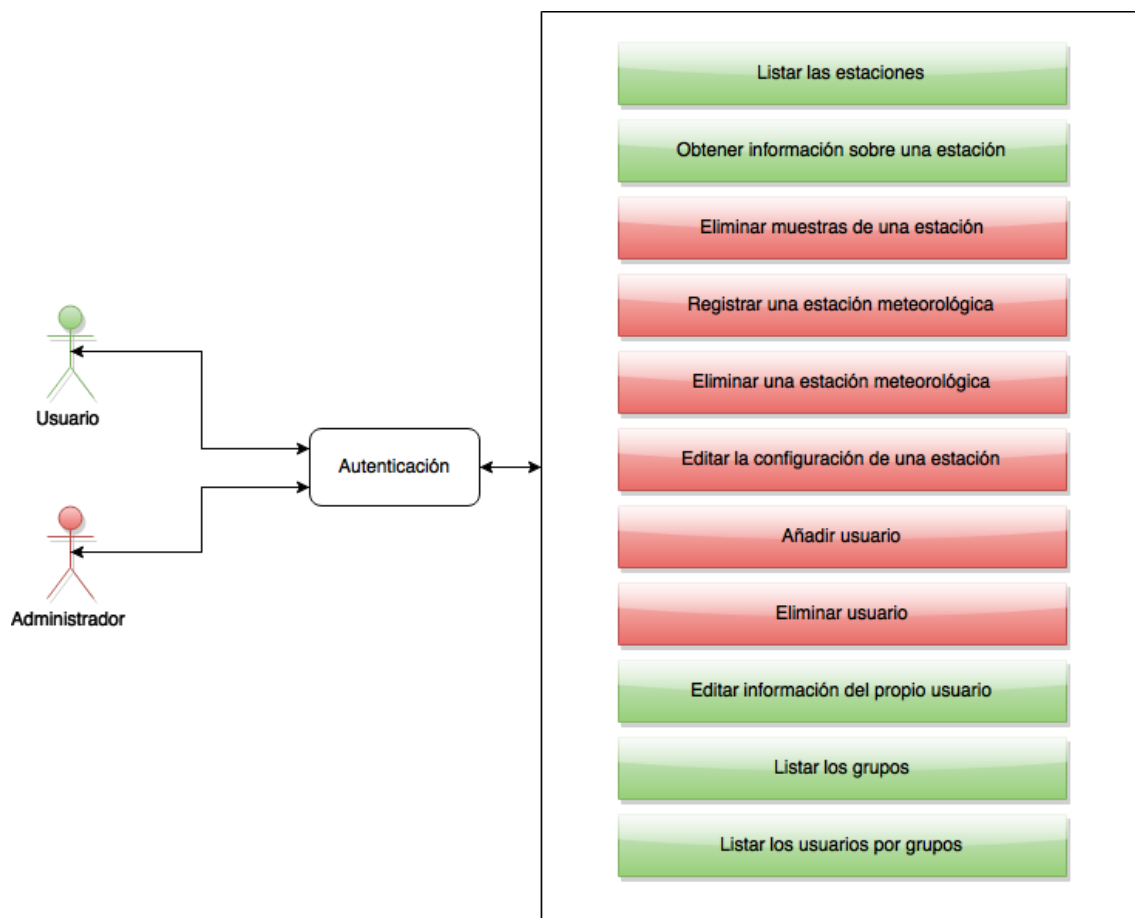


Figura 14: Casos de uso correspondientes a los roles de usuario normal y de administrador.

CU: 03	Autenticación.	
Descripción:	Comprueba que un usuario está dado de alta en el sistema y crea una sesión.	
Precondición:		
Secuencia normal:	Paso	Acción
	1	El usuario ingresa su nombre de usuario y contraseña.
	2	El sistema autentica al usuario.
	3	El usuario accede al sistema y puede realizar determinadas acciones según el grupo al que pertenezca.
Excepciones:	Paso	Descripción
	2	Usuario y/o contraseña incorrectos.

Tabla 12: Caso de uso 03.

CU: 04	Listar las estaciones.
Descripción:	Obtener una lista con todas las estaciones meteorológicas

	registradas en el sistema.	
Precondición:	El usuario debe estar autenticado.	
Secuencia normal:	Paso	Acción
	1	El sistema devuelve una lista con las estaciones meteorológicas registradas.
Excepciones:	Paso	Descripción
	1	No hay estaciones registradas en el sistema.
Comentarios:	La información que se obtiene incluye algunos datos como pueden ser: nombre de la estación, localización, fecha de registro, etc...	

Tabla 13: Caso de uso 04.

CU: 05	Obtener información sobre una estación.	
Descripción:	Obtiene información sobre una estación meteorológica, permitiendo visualizar el histórico de todas las muestras recogidas por la estación.	
Precondición:	El usuario debe estar autenticado.	
Secuencia normal:	Paso	Acción
	1	El usuario especifica la estación meteorológica que quiere visualizar.
	2	El sistema obtiene los datos y se los devuelve al usuario.
Excepciones:	Paso	Descripción
	1	La estación meteorológica no está registrada en el sistema.

Tabla 14: Caso de uso 05.

CU: 06	Eliminar muestras de una estación.	
Descripción:	Elimina una determinada muestra recogida por una estación meteorológica.	
Precondición:	El usuario debe estar autenticado y tener permisos de administrador.	
Secuencia normal:	Paso	Acción
	1	El usuario especifica la estación meteorológica.
	2	El usuario selecciona la muestra a borrar.
	3	El sistema elimina la muestra del sistema.
Excepciones:	Paso	Descripción
	1	La estación meteorológica especificada no existe.
	2	La muestra especificada no existe.

Tabla 15: Caso de uso 06.

CU: 07	Registrar una estación meteorológica.	
Descripción:	Registra en el sistema una estación meteorológica.	
Precondición:	El usuario debe estar autenticado y tener permisos de administrador.	
Secuencia normal:	Paso	Acción
	1	Se completa un formulario con los datos de la estación y se emplea la MAC como identificador único de la estación.
	2	El sistema registra la estación meteorológica permitiendo almacenar las muestras que envía.
Excepciones:	Paso	Descripción
	1	La estación ya está registrada.

Tabla 16: Caso de uso 07.

CU: 08	Eliminar una estación meteorológica.	
Descripción:	Da de baja una estación meteorológica del sistema y elimina todos sus datos almacenados.	
Precondición:	El usuario debe estar autenticado y tener permisos de administrador.	
Secuencia normal:	Paso	Acción
	1	Se selecciona la estación meteorológica que se desea dar de baja.
	2	El sistema la da de baja.
Excepciones:	Paso	Descripción
	1	La estación meteorológica seleccionada no existe.

Tabla 17: Caso de uso 08.

CU: 09	Editar la configuración de una estación.	
Descripción:	Permite modificar los parámetros de configuración de una estación meteorológica.	
Precondición:	El usuario debe estar autenticado y tener permisos de administrador.	
Secuencia normal:	Paso	Acción
	1	Se selecciona la estación meteorológica que se desea modificar.
	2	Se realizan los cambios en los parámetros de configuración y se confirman.
	3	El sistema guarda los parámetros de configuración.
Excepciones:	Paso	Descripción
	1	La estación meteorológica seleccionada no existe.
	2	El formato de los parámetros de configuración es incorrecto.

	3	Los parámetros superan los umbrales de configuración.
Comentarios:	La estación meteorológica obtendrá los nuevos parámetros de configuración la próxima vez que envíe datos al sistema.	

Tabla 18: Caso de uso 09.

CU: 10	Añadir usuario.	
Descripción:	Añadir un nuevo usuario al sistema.	
Precondición:	El usuario debe estar autenticado y tener permisos de administrador.	
Secuencia normal:	Paso	Acción
	1	Se introducen los datos de acceso del usuario: nombre de usuario, contraseña, grupo, etc...
	2	Se confirman los datos.
	3	El sistema añade el nuevo usuario al sistema.
Excepciones:	Paso	Descripción
	2	El formato de los parámetros es incorrecto.
	3	Ya hay un usuario dado de alta con ese nombre.

Tabla 19: Caso de uso 10.

CU: 11	Eliminar un usuario.	
Descripción:	Elimina un usuario del sistema.	
Precondición:	El usuario debe estar autenticado y tener permisos de administrador.	
Secuencia normal:	Paso	Acción
	1	Se selecciona el usuario que se desea eliminar.
	2	El sistema da de baja el usuario.
Excepciones:	Paso	Descripción
	2	El usuario seleccionado no existe.

Tabla 20: Caso de uso 11

CU: 12	Editar información del propio usuario.	
Descripción:	Permite a cualquier usuario modificar la información de su perfil.	
Precondición:	El usuario debe estar autenticado en el sistema.	
Secuencia normal:	Paso	Acción
	1	Se editan los datos que sean modificables.
	2	Se confirman los cambios.
	3	El sistema actualiza la información del perfil de usuario.
Excepciones:	Paso	Descripción
	2	El formato de los datos no es correcto.
	3	Los valores de los datos son conflictivos o superan los

		umbrales del sistema.
--	--	-----------------------

Tabla 21: Caso de uso 12.

CU: 13	Listar los grupos.	
Descripción:	Obtiene una lista con todos los grupos de usuario del sistema.	
Precondición:	El usuario debe estar autenticado en el sistema.	
Secuencia normal:	Paso	Acción
	1	Obtiene una lista con los grupos de usuario.
Excepciones:	Paso	Descripción

Tabla 22: Caso de uso 13.

CU: 14	Listar los usuarios por grupos.	
Descripción:	Obtiene una lista con todos los miembros de un grupo determinado.	
Precondición:	El usuario debe estar autenticado en el sistema.	
Secuencia normal:	Paso	Acción
	1	Se selecciona el grupo de usuarios.
	2	El sistema devuelve una lista con todos los usuarios miembros de ese grupo.
Excepciones:	Paso	Descripción
	1	El grupo seleccionado no existe.

Tabla 23: Caso de uso 14.

Capítulo 4

Diseño del servicio REST

4.1. Introducción

Para la programación del servicio REST se escogió Node.js [29] [30], un entorno en tiempo de ejecución multiplataforma, pensado para la capa del servidor y está basado en el lenguaje ECMAScript. Entre sus características destacan: programación asíncrona no bloqueante, I/O de datos orientados a eventos, alta escalabilidad, código abierto, comunidad de desarrollo activa, documentación de calidad, librerías de terceros, etc. Todas estas características, y que es un entorno de programación que permite desarrollar aplicaciones rápidamente, lo determinaron como la herramienta idónea para esta tarea.

En cuanto al sistema de almacenamiento de los datos se empleará MongoDB [31]. MongoDB es un sistema de base de datos NoSQL [32] orientado a documentos. En lugar de almacenar los datos en tablas como en las bases de datos relacionales, MongoDB almacena estructuras de datos en documentos tipo JSON con un esquema dinámico (MongoDB llama a este formato BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

Una característica importante de MongoDB es que no exige que los documentos posean una estructura fija, como sucede con los campos de una tabla en sistemas relacionales. Esta característica nos será de utilidad para permitir el registro de estaciones meteorológicas con diferentes sensores y para el almacenamiento de sus datos.

Como formato para el intercambio de los datos se utilizará JSON, ya que es nativo para MongoDB, Node.js y para el lenguaje javascript de los navegadores web.

La API del sistema se desarrollará a partir de las necesidades del usuario, es decir, los casos de uso del cliente web y los casos de uso de la estación meteorológica.

4.2. Base de datos

En este capítulo se describe el diseño propuesto para la base de datos en MongoDB. La nomenclatura de los elementos en MongoDB cambia significativamente. Así, una base de datos se denomina esquema y una tabla es una colección de documentos (filas).

El sistema dispone de un sistema de grupos de usuarios que funcionan como roles. Podría existir un rol para registrar nuevos usuarios en el sistema con un nivel de privilegio distinto y asignarse a un usuario normal sin privilegios. A pesar de que el sistema está diseñado y programado para ello, tan sólo existen tres grupos de usuarios (ver Tabla 23):

Nombre del grupo	Nivel de privilegio
Admin	0
User	1
Station	2

Tabla 24: Grupos de usuarios.

El esquema de la colección de grupos es el que se muestra en la Tabla 24:

Group		
Nombre del campo	Tipo de datos	Características
groupName	String	trim, unique, index, required
privilegeLevel	Number	min: 0, required

Tabla 25: Esquema de la colección Group.

Las estaciones meteorológicas se van a modelar como un usuario del sistema perteneciente al grupo de las estaciones meteorológicas y con otro nivel de privilegio. De este modo, la función *login* será la misma para usuarios y estaciones. Para ello, será necesario crear una colección padre que englobe a los usuarios y a las estaciones (ver Tabla 25):

User			
Nombre del campo	Tipo de datos	Características	
_type	String	Required	
Login	String	unique, index, trim, required	
password	String	Required	
groups	Array		
	ref: 'Group'	Schema.ObjectId	required
Data	String	required	

Tabla 26: Esquema de la colección User.

La colección User contiene la estructura básica del documento y surgen dos tipos distintos a partir de ella (ver Tablas 26 y 27):

Person			
Nombre del campo	Tipo de datos	Características	
_type	String	required, default: 'Person'	
Login	String	unique, index, trim, required	
Password	String	required	
Groups	Array		
	ref: 'Group'	Schema.ObjectId	required
Data			
	name	String	trim, required
	secondName	String	trim, required
	dateOfBirth	Date	required

Tabla 27: Esquema de la colección Person.

Station			
Nombre del campo	Tipo de datos	Características	
_type	String	required, default: ‘Station	
Login	String	unique, index, trim, required	
password	String	required	
Groups	Array		
	ref: ‘Group’	Schema.ObjectId	required
Data			
	location	String	trim, required
	dateOfRegistration	Date	default:now, required
	wifi	String	trim, required
	wifiPassword	String	trim, required
	Ip	String	trim, required
	Port	Number	required
	Interval	Number	required

Tabla 28: Esquema de la colección Station.

Finalmente, falta por definir cómo se modelarán los datos (muestras) de las estaciones meteorológicas (ver Tabla 28).

Sample		
Nombre del campo	Tipo de datos	Características
MAC	String	trim, required
sampleDate	Date	default: now, required
samples	[]	

Tabla 29: Esquema de la colección Sample.

La MAC del módulo WiFi servirá como identificador de la estación meteorológica. En caso de que haya diferentes tecnologías de comunicación inalámbrica no habrá problemas ya que otras tecnologías como ZigBee o Bluetooth también disponen de una MAC.

Las estaciones meteorológicas enviarán los datos en formato JSON y se insertarán directamente en la base de datos sin realizar ningún tipo de cambio en la estructura de los datos recibidos. Los campos fijos que van a tener los datos de las estaciones meteorológicas son: MAC (identificador del dispositivo) y la fecha de la muestra. El resto de los valores pueden ser variables de una estación a otra y no hay ningún problema ya que MongoDB está diseñada para trabajar con colecciones de documentos con campos variables.

4.3. API del servicio

La API del servicio REST (ver Apéndice A) se ha diseñado para que sea versionable y requieran autenticación todas sus funciones. El código fuente del servicio REST puede ser consultado en el repositorio de código del autor.

Capítulo 5

Implementación y prueba del servicio

5.1. Dependencias

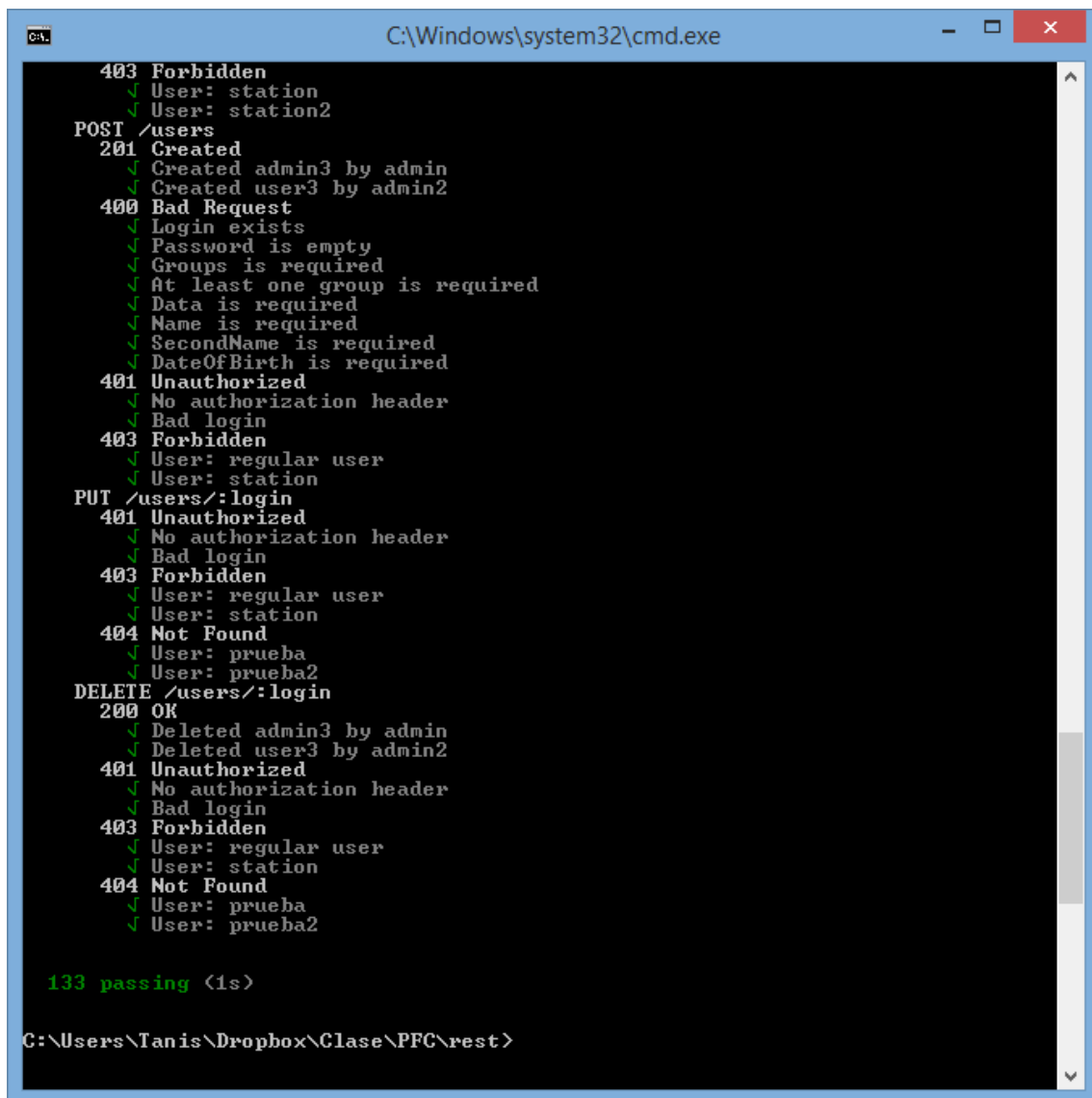
Para el desarrollo del proyecto debemos tener instalado Node.js y se utilizará el programa npm como gestor de paquetes. Estas son las dependencias del proyecto:

- Body-parser. Es un middleware que se encarga de procesar los cuerpos de todas las peticiones HTTP y permite convertirlas a un objeto JSON automáticamente.
- Debug. El paquete estándar encargado de las tareas de depuración.
- Express. Es un framework empleado para desarrollar el backend de aplicaciones web. En este caso lo emplearemos para el servidor REST.
- Mongoose. Es el software encargado de realizar las comunicaciones con MongoDB.
- Scribe-JS. Es un middleware para un sistema avanzado de logs que emplearemos para llevar un registro de todas las peticiones REST y un control de los usuarios.
- HTTP-auth. Un paquete que permite trabajar fácilmente con la cabecera Authorization de las peticiones HTTP.
- Mocha. Es un sistema de automatización de tests empleando la herramienta npm.
- Supertest. Un paquete que permite realizar tests para servicios REST.

Todas estas dependencias estarán reflejadas en el fichero package.json y se pueden instalar automáticamente con el comando `npm install`, dentro de dicho directorio.

5.2. Pruebas

Como se mencionó anteriormente, las pruebas se realizan empleando el paquete Mocha y Supertest. Para ello, arrancamos el servicio REST en modo testing con el comando `npm start test` y ejecutamos los tests con `npm test` (ver Figura 16).



```
C:\Windows\system32\cmd.exe

403 Forbidden
  ✓ User: station
  ✓ User: station2
POST /users
201 Created
  ✓ Created admin3 by admin
  ✓ Created user3 by admin2
400 Bad Request
  ✓ Login exists
  ✓ Password is empty
  ✓ Groups is required
  ✓ At least one group is required
  ✓ Data is required
  ✓ Name is required
  ✓ SecondName is required
  ✓ DateOfBirth is required
401 Unauthorized
  ✓ No authorization header
  ✓ Bad login
403 Forbidden
  ✓ User: regular user
  ✓ User: station
PUT /users/:login
401 Unauthorized
  ✓ No authorization header
  ✓ Bad login
403 Forbidden
  ✓ User: regular user
  ✓ User: station
404 Not Found
  ✓ User: prueba
  ✓ User: prueba2
DELETE /users/:login
200 OK
  ✓ Deleted admin3 by admin
  ✓ Deleted user3 by admin2
401 Unauthorized
  ✓ No authorization header
  ✓ Bad login
403 Forbidden
  ✓ User: regular user
  ✓ User: station
404 Not Found
  ✓ User: prueba
  ✓ User: prueba2

133 passing (1s)

C:\Users\tanis\Dropbox\Clase\PFC\rest>
```

Figura 15: Ejecución de los tests.

5.3. Funcionamiento

Se puede comprobar el correcto funcionamiento del servicio REST realizando peticiones con cualquier cliente HTTP, como por ejemplo, Postman (ver Figura 17).

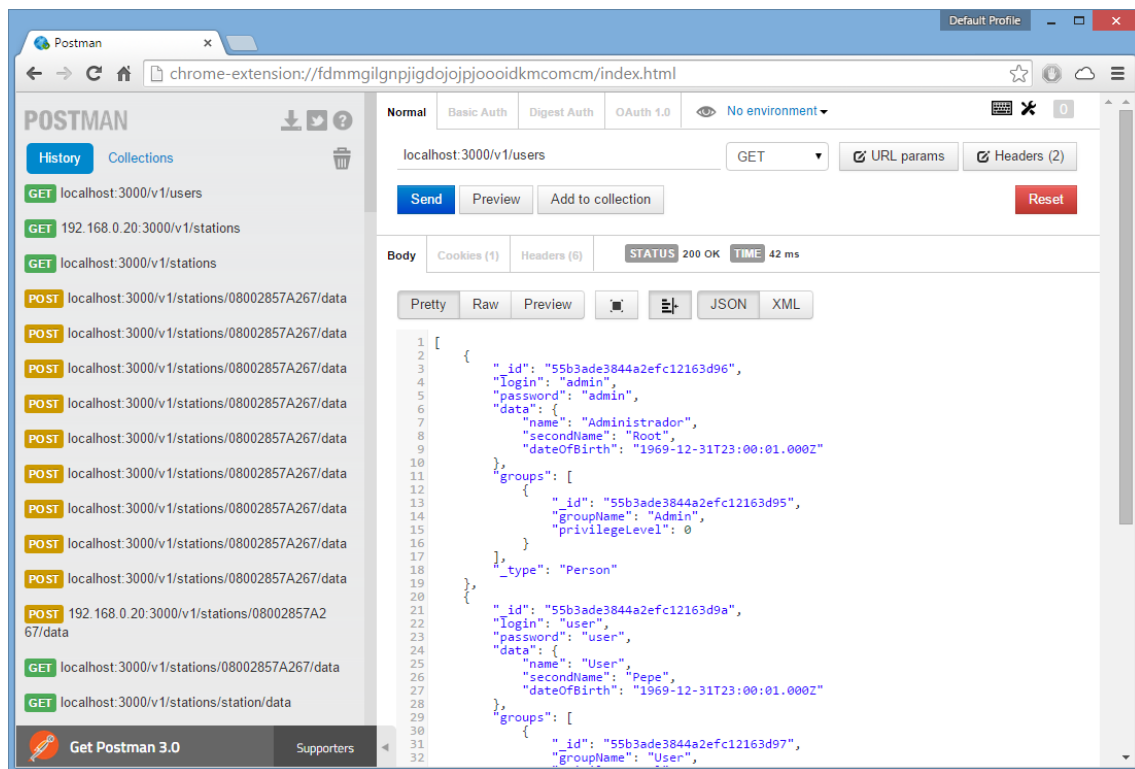


Figura 16: Ejemplo de consulta de la API empleando Postman.

El servicio dispone de un sistema de logs avanzados que se almacenan en ficheros JSON por fechas. Se pueden consultar a través de la propia interfaz web que provee la librería Scribe-JS, para ello accedemos a la URL localhost:3000/logs (ver Figura 18) y hay que autenticarse con la cuenta de algún usuario administrador.

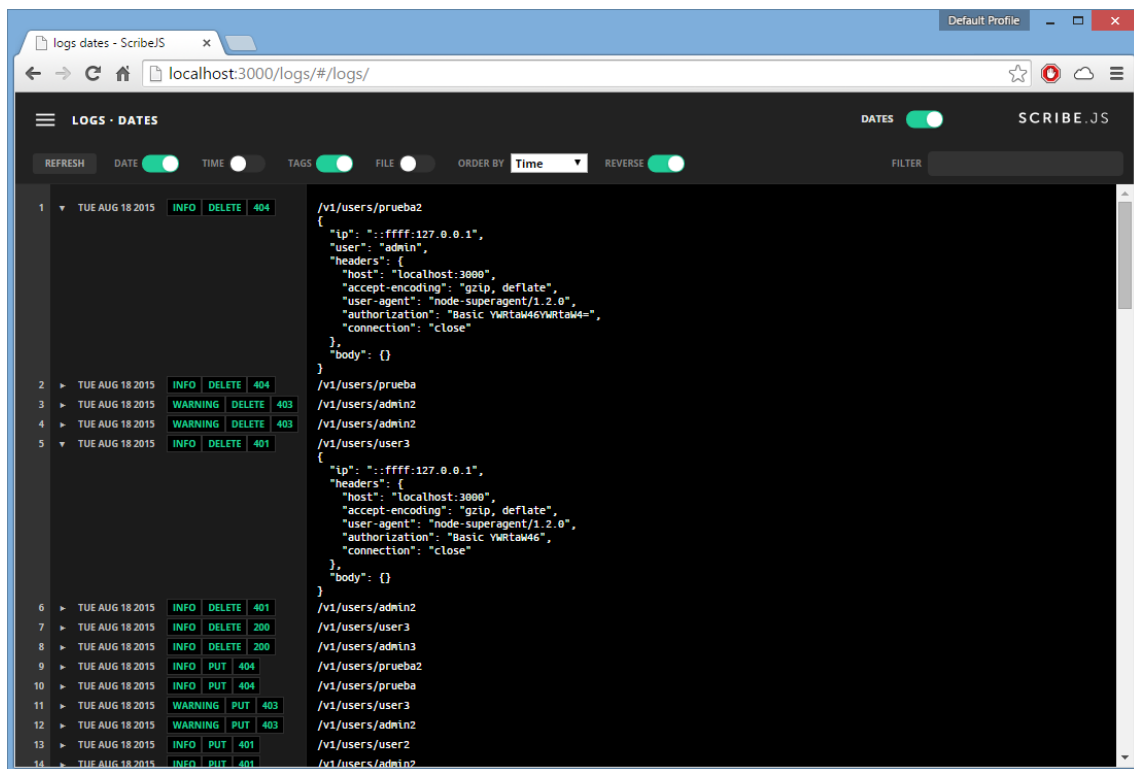


Figura 17: Sistema de logs empleando Scribe-JS.

Capítulo 6

Diseño e integración de la estación meteorológica

6.1. Testeo de los componentes

Primero necesitamos comprobar el correcto funcionamiento de los dos sensores: el sensor BMP180 y el DHT22. Para ello emplearemos el Arduino Uno y haremos las conexiones básicas con un pequeño programa que recupera los valores de sus mediciones.

6.1.1. Sensor BMP180

Este sensor nos permite medir valores de temperatura, presión atmosférica y altitud. Los valores de temperatura no son muy precisos por lo que se empleará el DHT22 para esta tarea, en cuanto a la altitud, no la consideramos de importancia ya que la estación se va a encontrar a una altura fija.

El circuito es muy sencillo (ver Figura 19), tan sólo necesita 4 conexiones: VCC, GND, SDA y SCL.

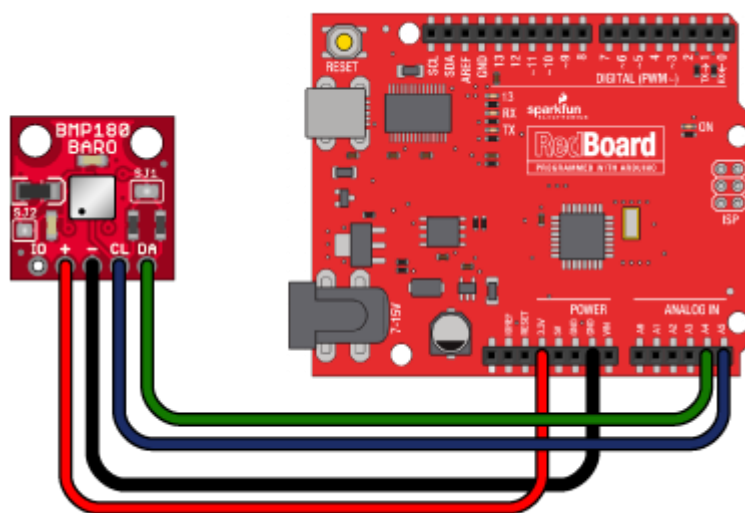


Figura 18: BMP180 conectado a un Arduino Uno.

A continuación podemos ver el montaje físico realizado empleando una protoboard (ver Figura 20).



Figura 19: Montaje físico del sensor BMP180.

Para obtener los valores del sensor BMP180 emplearemos la librería de Adafruit “Adafruit_BMP085“, que es software libre (ver Figura 21).

```

1  #include <Wire.h>
2  #include <Adafruit_BMP085.h>
3
4  Adafruit_BMP085 bmp;
5
6  void setup() {
7    Serial.begin(9600);
8    if (!bmp.begin()) {
9      Serial.println("Could not find a valid BMP085 sensor, check wiring!");
10     while (1) {}
11   }
12 }
13
14 void loop() {
15   Serial.print("Temperature = ");
16   Serial.print(bmp.readTemperature());
17   Serial.println(" *C");
18
19   Serial.print("Pressure = ");
20   Serial.print(bmp.readPressure());
21   Serial.println(" Pa");
22
23   // Calculate altitude assuming 'standard' barometric
24   // pressure of 1013.25 millibar = 101325 Pascal
25   Serial.print("Altitude = ");
26   Serial.print(bmp.readAltitude());
27   Serial.println(" meters");
28
29   Serial.print("Pressure at sealevel (calculated) = ");
30   Serial.print(bmp.readSealevelPressure());
31   Serial.println(" Pa");
32
33   // you can get a more precise measurement of altitude
34   // if you know the current sea level pressure which will
35   // vary with weather and such. If it is 1015 millibars
36   // that is equal to 101500 Pascals.
37   Serial.print("Real altitude = ");
38   Serial.print(bmp.readAltitude(101500));
39   Serial.println(" meters");
40
41   Serial.println();
42   delay(2000);
43 }
44

```

Figura 20: Código de prueba para el sensor BMP180.

Compilamos y cargamos el programa empleando el IDE de Arduino y obtendremos las lecturas a través del puerto serie (ver Figura 22).

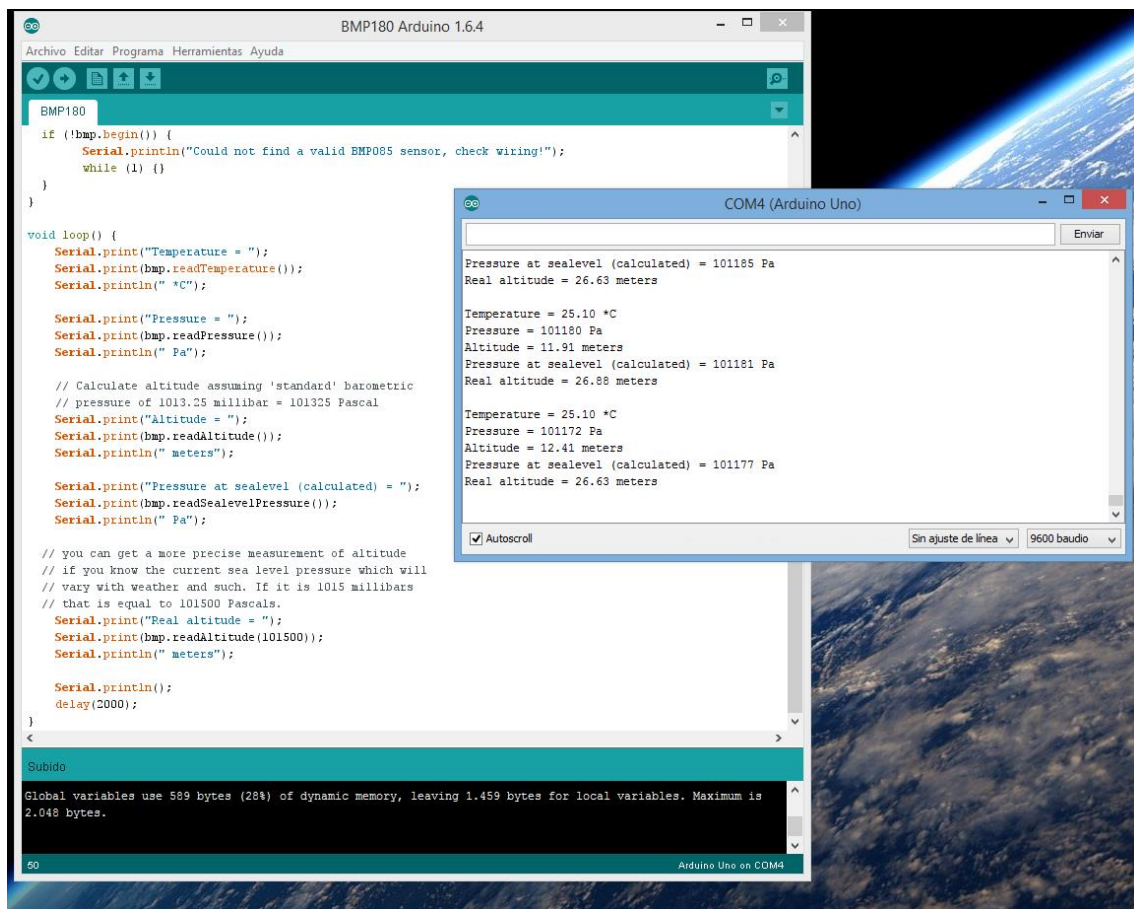


Figura 21: Salida serial del sensor BMP180.

6.1.2. Sensor DHT22

El DHT22 es un sensor digital de gran precisión que permite medir la temperatura ambiente y el porcentaje de humedad.

El circuito para probar su funcionamiento (ver Figura 23), requiere tres conexiones (de izquierda a derecha): VCC (3 a 5V), salida de datos con una resistencia de 10K y la toma de tierra en el último pin.

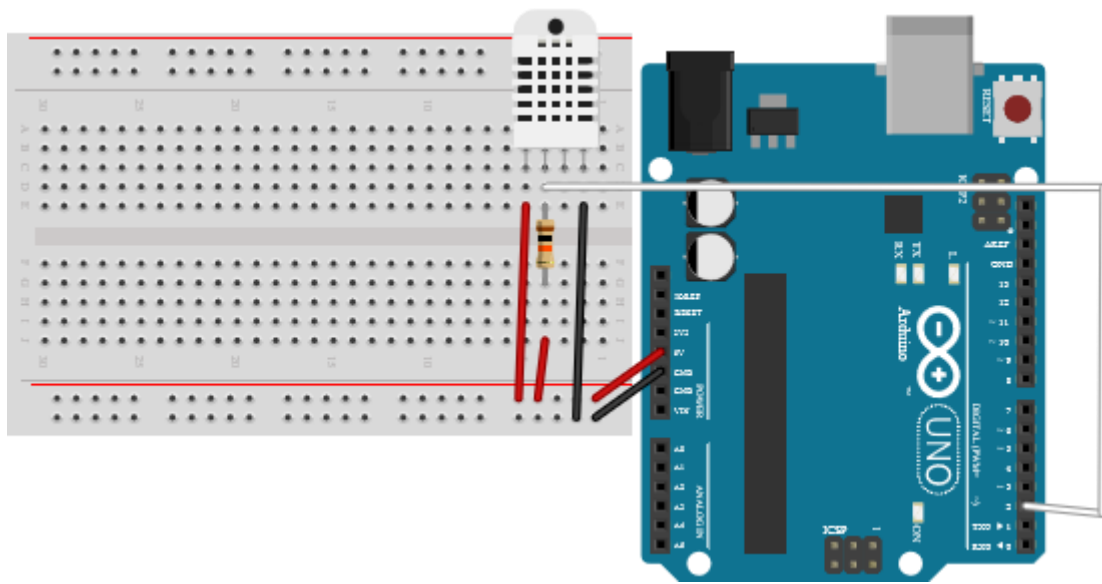


Figura 22: DHT22 conectado a un Arduino Uno.

A continuación podemos ver el montaje físico realizado empleando una protoboard (ver Figura 24).

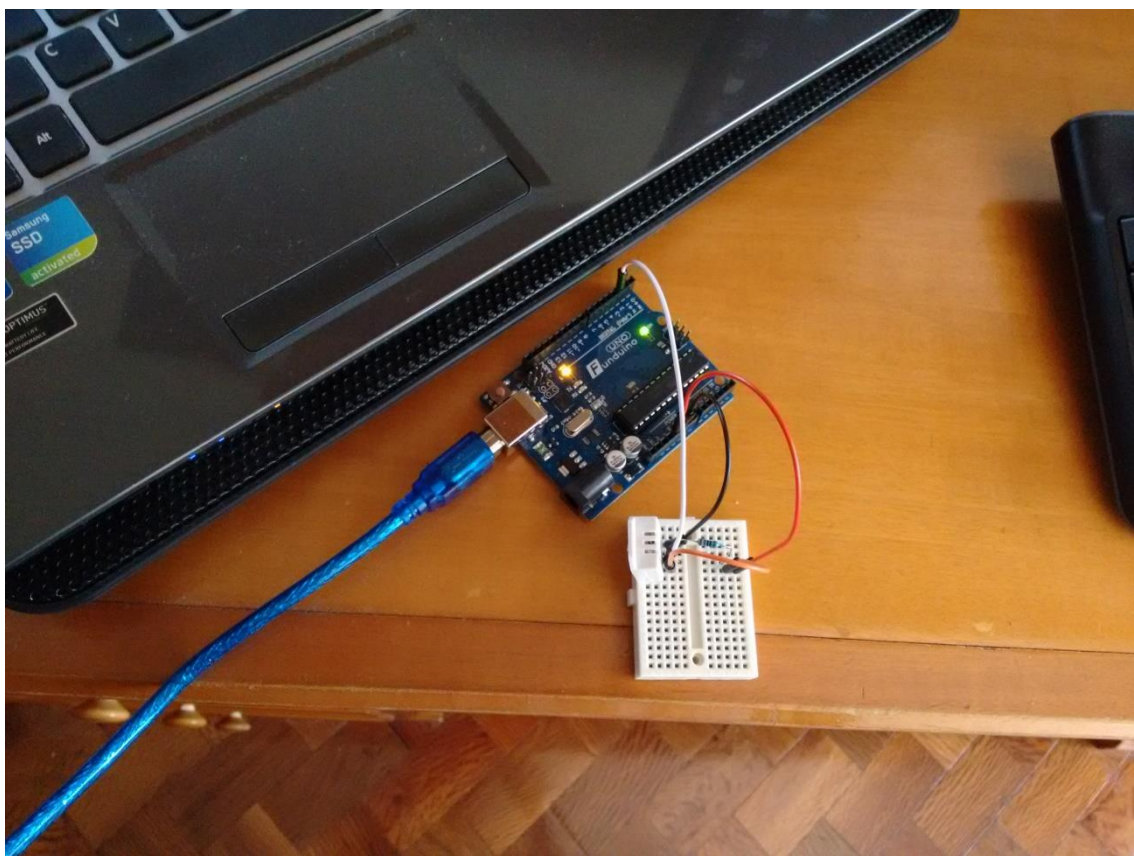


Figura 23: Montaje físico del sensor DHT22.

Para obtener los valores de temperatura y humedad del sensor DHT22 emplearemos la librería DHT.h (ver Figura 25).


```

1  #include <DHT.h>
2
3  #define DHTPIN 2      // what pin we're connected to
4  #define DHTTYPE DHT22 // DHT 22 (AM2302)
5
6  DHT dht(DHTPIN, DHTTYPE);
7
8
9  void setup() {
10     Serial.begin(9600);
11     Serial.println("DHTxx test!");
12
13     dht.begin();
14 }
15
16 void loop() {
17     // Wait a few seconds between measurements.
18     delay(2000);
19
20     // Reading temperature or humidity takes about 250 milliseconds!
21     // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
22     float h = dht.readHumidity();
23     // Read temperature as Celsius
24     float t = dht.readTemperature();
25     // Read temperature as Fahrenheit
26     float f = dht.readTemperature(true);
27
28     // Check if any reads failed and exit early (to try again).
29     if (isnan(h) || isnan(t) || isnan(f)) {
30         Serial.println("Failed to read from DHT sensor!");
31         return;
32     }
33
34     // Compute heat index
35     // Must send in temp in Fahrenheit!
36     float hi = dht.computeHeatIndex(f, h);
37
38     Serial.print("Humidity: ");
39     Serial.print(h);
40     Serial.print(" %\t");
41     Serial.print("Temperature: ");
42     Serial.print(t);
43     Serial.print(" *C ");
44     Serial.print(f);
45     Serial.print(" *F\t");
46     Serial.print("Heat index: ");
47     Serial.print(hi);
48     Serial.println(" *F");
49 }
50

```

Figura 24: Código de prueba para el sensor DHT22.

Compilamos y cargamos el programa empleando el IDE de Arduino y obtendremos las lecturas a través del puerto serie (ver Figura 26).

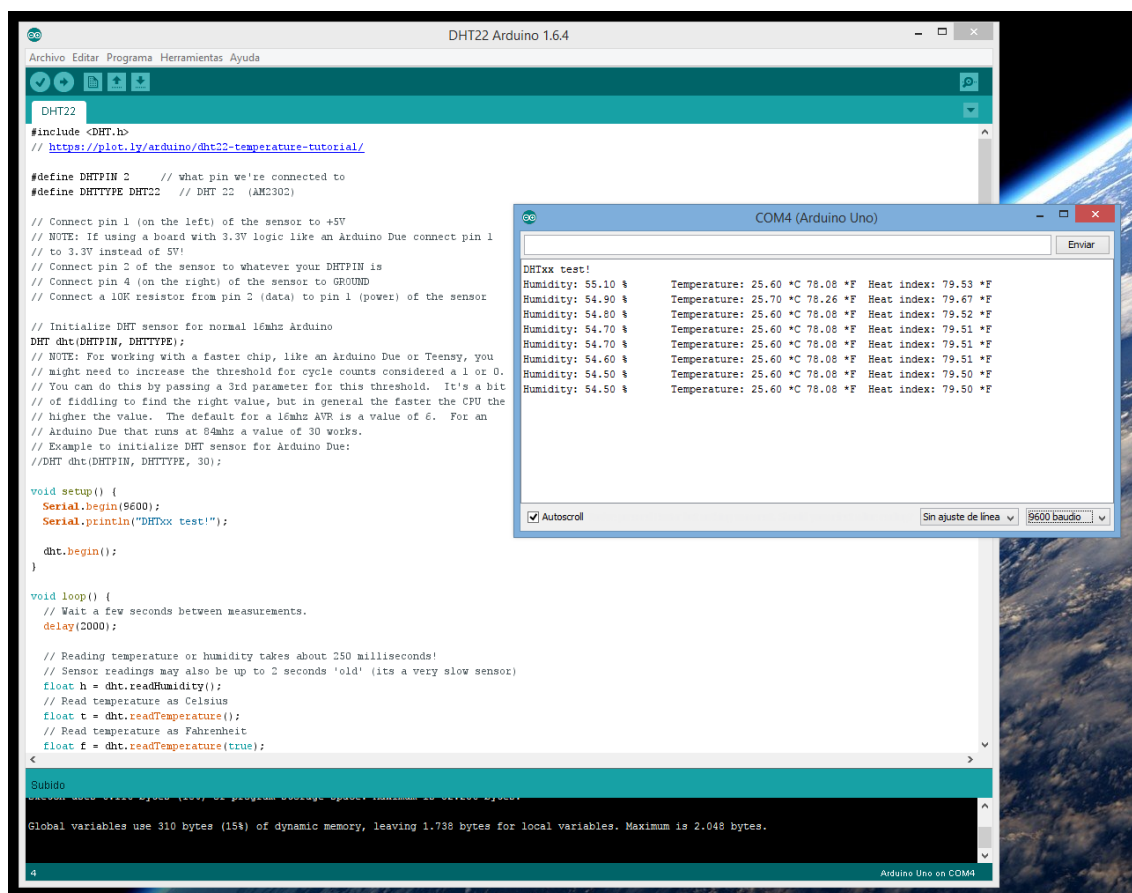


Figura 25: Salida serial del sensor DHT22.

6.2. Diseño de la estación meteorológica

Para evitar el uso de una protoboard y las malas conexiones de los cables debido a movimientos, decidimos soldar los cables y los sensores encima de la Shield CC3000 acoplada al Arduino Uno (ver Figura 14). Además soldaremos un led para indicar la actividad de la estación: encendiéndose en caso de actividad, parpadeando en caso de error y apagado el resto del tiempo. En un montaje final se suprimiría este led debido a su elevado consumo energético, pero en cambio resultará muy útil para el desarrollo y prueba de la estación meteorológica.

Los cables se soldaron por debajo de la Shield CC3000 (ver Figura 27) por motivos de estética, dejando solamente visibles los dos sensores y el led encima de la Shield CC3000 (ver Figura 28).

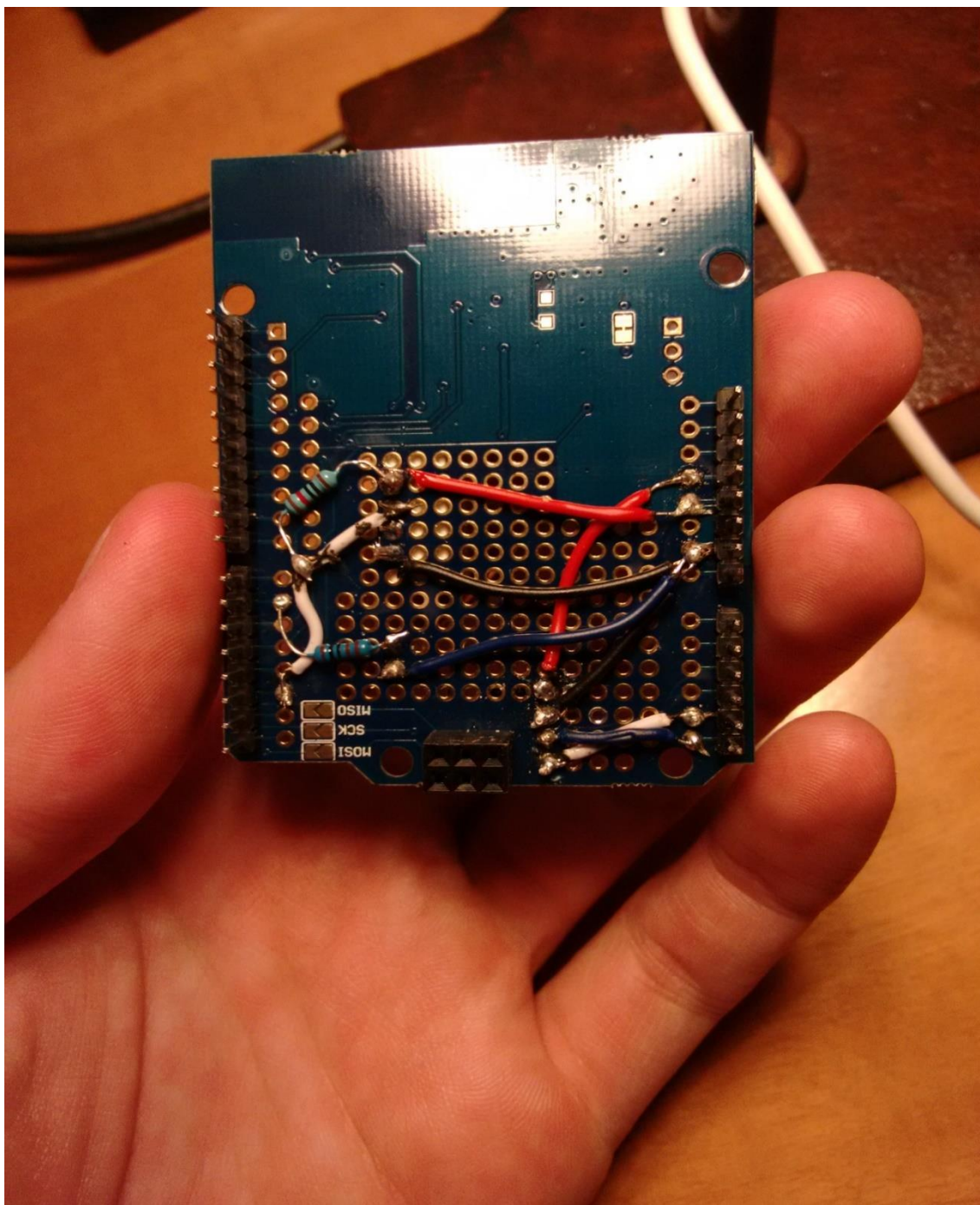


Figura 26: Conexiones de la Shield CC3000.

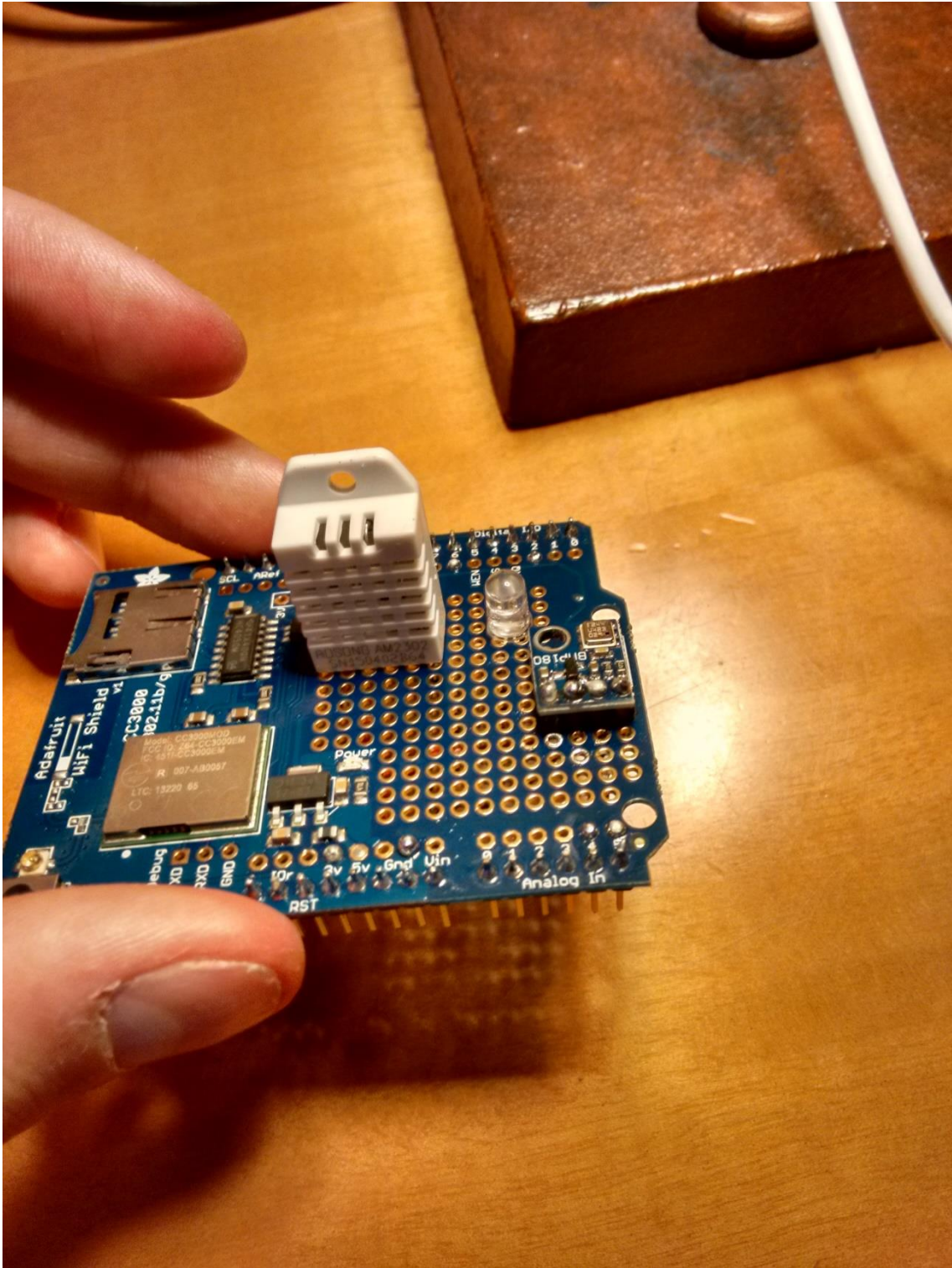


Figura 27: Sensores y led soldados a la Shield CC3000.

Finalmente este es el aspecto de nuestra estación meteorológica (ver Figura 29):

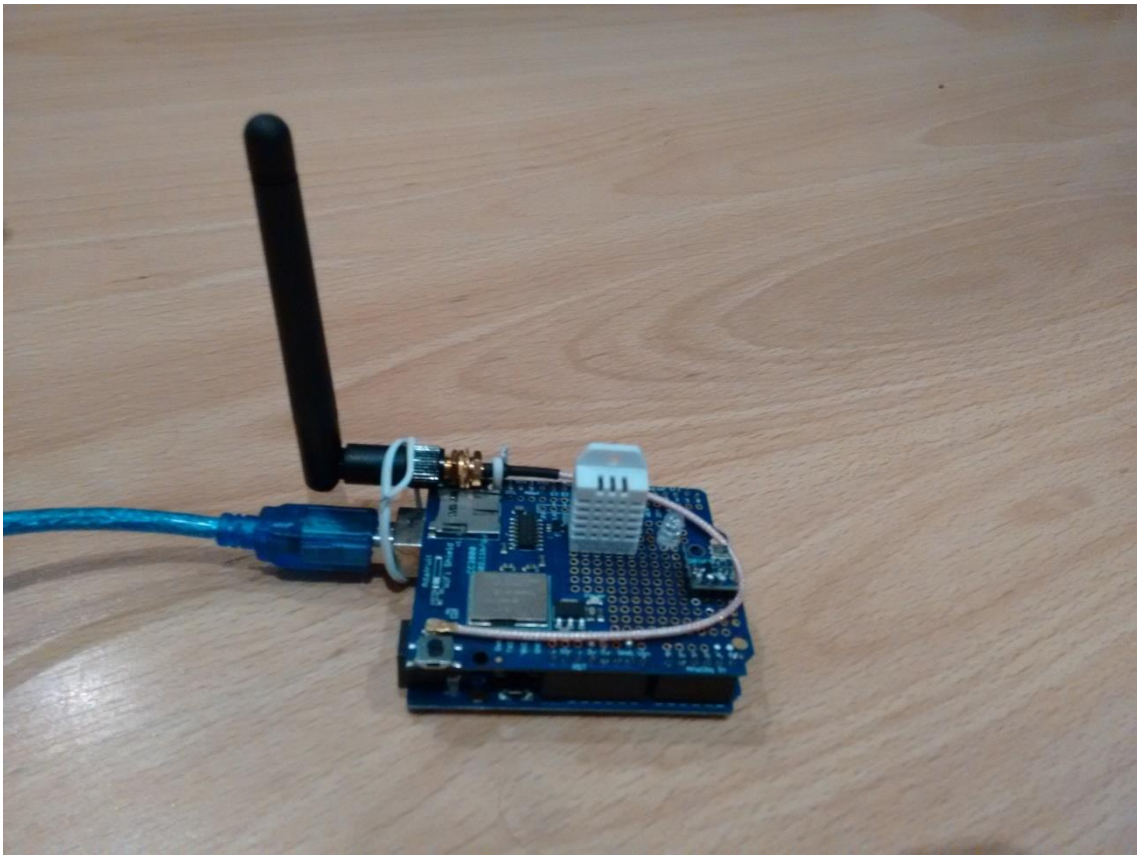


Figura 28: Estación meteorológica.

6.3. Programación e integración

En la programación de la estación meteorológica se emplean diversas librerías de terceros bajo licencias de software libre:

- Adafruit_CC3000. Esta librería es empleada para la comunicación del Arduino con la Shield CC3000.
- Adafruit_BMP085. Utilizada para obtener las mediciones del sensor BMP180.
- DHT sensor library. Utilizada para obtener las mediciones del sensor DHT22.
- Base64. Empleada en la codificación base64 para la cabecera Authorization de las peticiones HTTP.
- JSMM. Esta librería es utilizada para procesar cadenas JSON al obtener la configuración del servidor.

La estación meteorológica sigue el algoritmo la Sección 3.2 (ver Figura 11) y emplea las siguientes funciones del API (ver Apéndice A):

- POST /v1/stations/:MAC/data. Esta petición POST almacena los valores de los sensores de la estación meteorológica.
- GET /v1/stations/:MAC/. Obtiene la configuración de la estación meteorológica.

Empleando el sistema de logs podemos comprobar que la estación meteorológica funciona correctamente, pudiendo consultar fecha, dirección IP, MAC de la estación (parámetro user) y los valores de sus sensores (ver Figura 30).

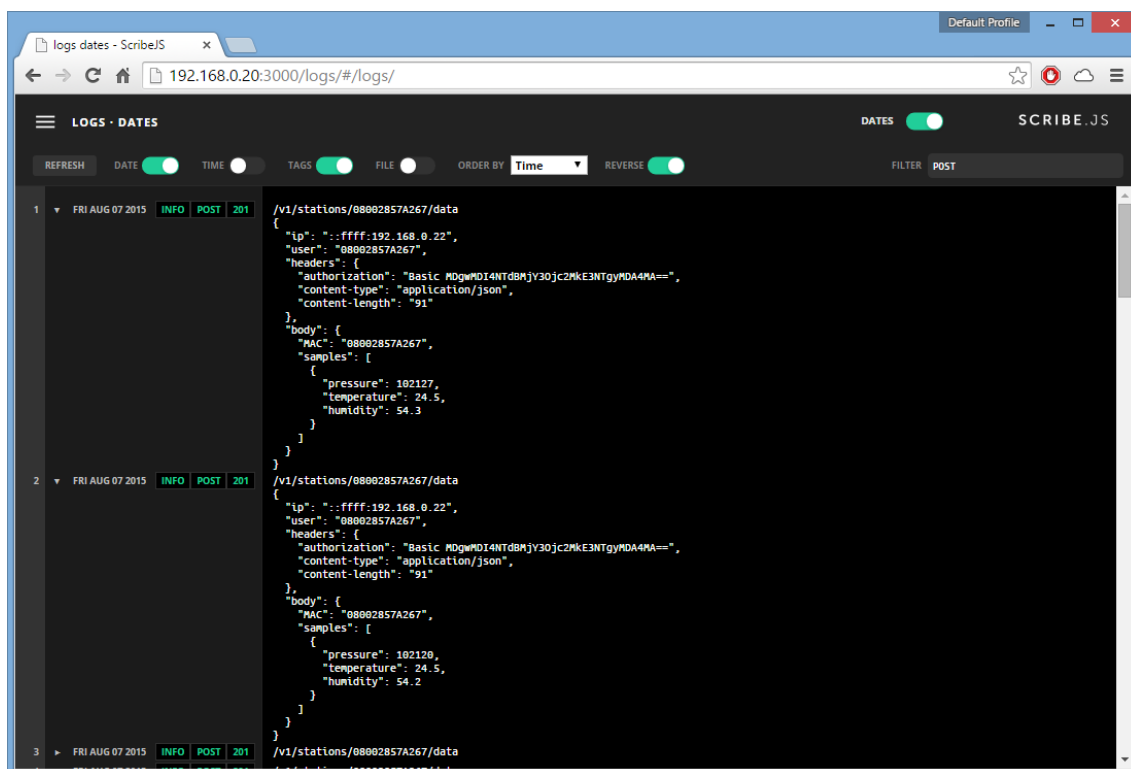


Figura 29: Logs de la estación meteorológica enviando muestras.

Capítulo 7

Estudio del consumo energético de la estación meteorológica

7.1. Descripción del estudio

En este capítulo se procede a la medición del consumo energético de la estación meteorológica para determinar la duración de su funcionamiento, alimentada por una batería. Para ello se empleará un analizador de potencia Agilent N6705B (ver Figura 31) disponible gracias a la colaboración del CITIC [33].



Figura 30: Agilent N6705B.

El analizador de potencia lleva asociado un software, Agilent 14585A Control and Analysis Software, que es el encargado de recibir las mediciones, a través de la red, y representarlas gráficamente (ver Figura 32).

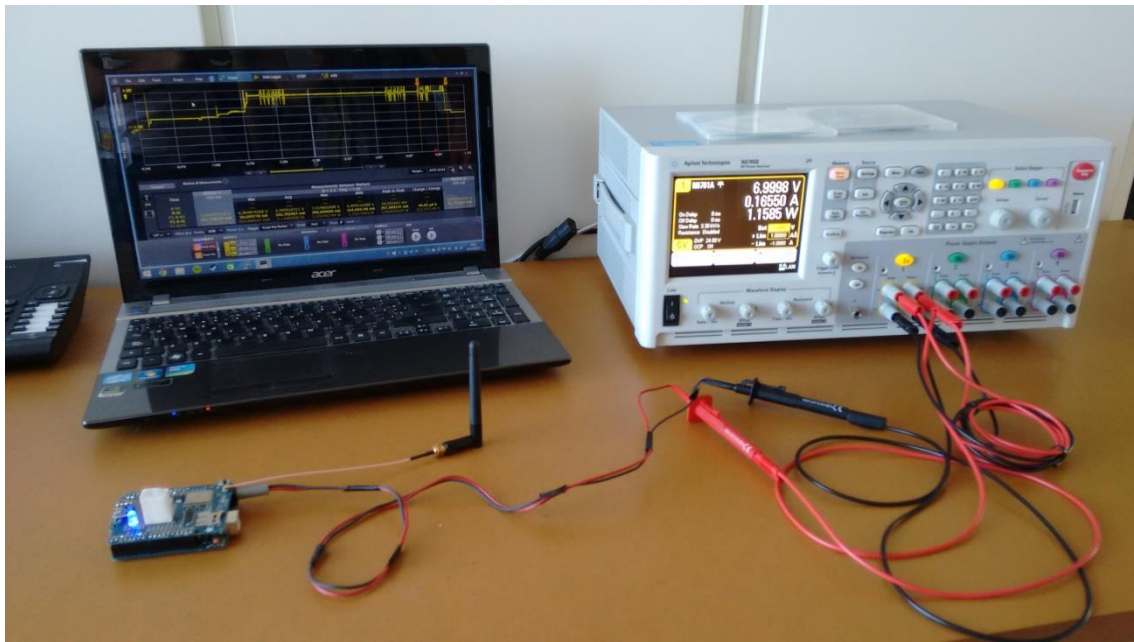


Figura 31: Medición del consumo energético de la estación meteorológica.

7.2. Consumo energético

A la hora de realizar las mediciones hay que tener en cuenta el consumo energético en modo de bajo consumo y en modo de adquisición y envío de muestras (este modo se denomina “en funcionamiento” en contraposición al modo de bajo consumo). Para ello se realizará un análisis del consumo mínimo, máximo y medio en cada uno de los estados de la estación meteorológica. Finalmente, se hará un cálculo aproximado del consumo total.

7.2.1. Consumo energético en modo de bajo consumo

La Tabla 29 muestra los consumos de la estación meteorológica (ver Figura 33) en modo de bajo consumo:

Mínimo	Máximo	Media
50.61119 mA	54.568663 mA	51.982614 mA

Tabla 30: Consumo energético en modo de bajo consumo.

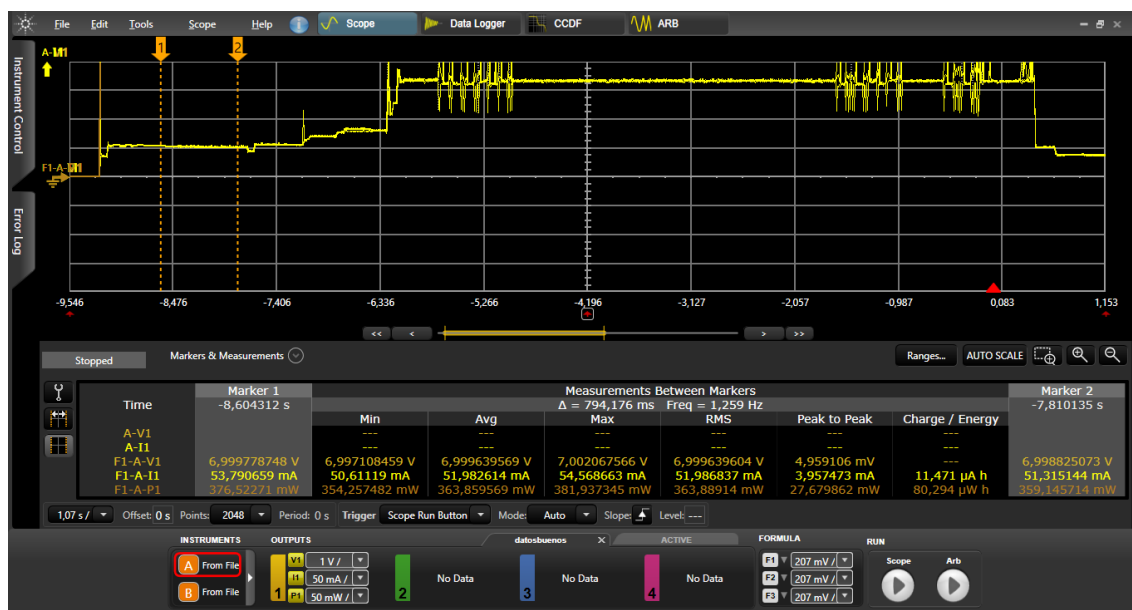


Figura 32: Consumo energético en modo de bajo consumo.

Si la estación meteorológica estuviera alimentada por una batería de 5V y 2000 mAh, se mantendría funcionando durante unas 38 horas y media. Este es un cálculo aproximado ya que la duración de la batería depende de muchos factores que se están pasando por alto. Por otra parte, podría utilizarse por ejemplo un panel solar para cargar dicha batería y obtener una estación completamente autónoma.

7.2.2. Consumo energético de la estación meteorológica en funcionamiento

Estos son los consumos de la estación meteorológica (ver Figura 34) cuando se encuentra en funcionamiento:

Mínimo	Máximo	Media
50.910782 mA	308.609009 mA	155.212728 mA

Tabla 31: Consumo energético en funcionamiento.

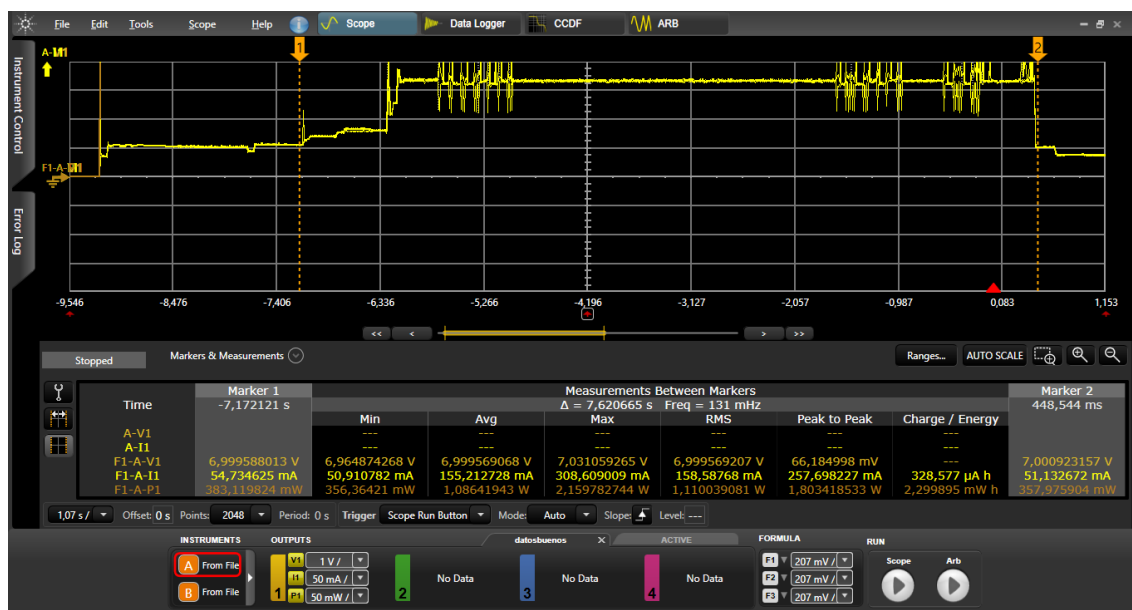


Figura 33: Consumo energético de la estación en funcionamiento.

El consumo energético de la estación meteorológica se dispara durante las comunicaciones WiFi. No obstante, este consumo sólo se produce un par de veces en una hora (depende del intervalo que se haya programado) y durante un período muy corto, de unos 8 segundos de media.

7.2.3. Consumo energético total

Con la estación meteorológica programada para enviar los valores medidos por sus sensores cada 24 minutos, aproximadamente, tendríamos un consumo medio de unos 53,1124 mAh. Este consumo permitiría a la estación meteorológica funcionar de forma continua durante unas 37 horas y media, aproximadamente.

7.3. Posibles mejoras para reducir el consumo

Estos son algunos puntos de mejora en la estación para optimizar su consumo energético [34]:

- LEDs. El Arduino Uno y la Shield CC3000 tienen pequeños LEDs que están constantemente consumiendo energía. Cada LED puede consumir unos 8-10 mA.
- Reguladores de voltaje. Los reguladores de voltaje de la placa Arduino y de la Shield CC3000 no son muy eficientes.
- Las resistencias Pull-Up. Como ocurre con los LEDs, estas resistencias son otra fuente de constante consumo energético.
- El led que hemos soldado encima de la Shield CC3000. Para tareas de depuración y prototipado es una gran ayuda, pero en un producto final es totalmente innecesario.
- Emplear el microcontrolador ATmega328P sin la placa Arduino.
- Utilizar una versión de la Shield CC3000 con unas características recortadas.

Finalmente, según Adafruit (el fabricante de la Shield CC3000), podríamos conseguir un consumo medio de 0.9 mA en modo de bajo consumo empleando sus diseños (ver Figura 35) y recomendaciones.

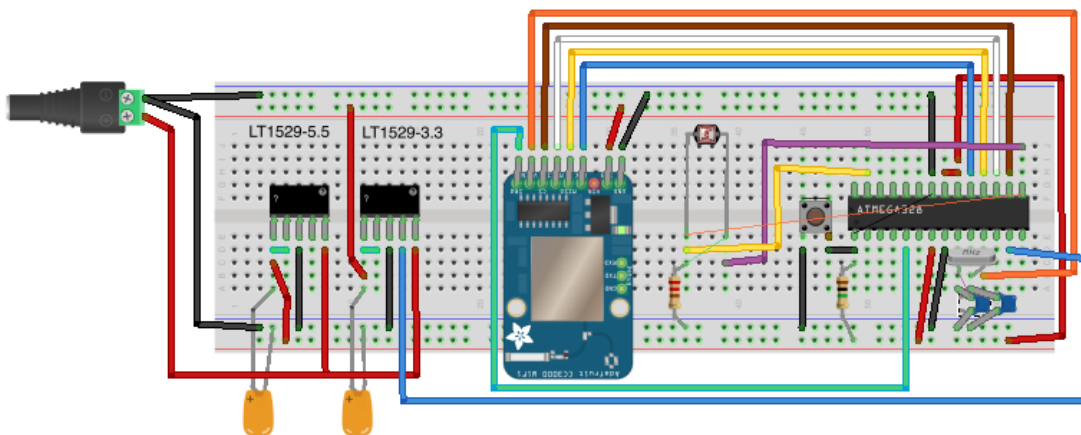


Figura 34: Hardware de la estación meteorológica mejorado.

Empleando otro hardware, la placa ESP8266 sería una gran alternativa una vez que su soporte software sea mejorado.

Capítulo 8

Desarrollo de la aplicación web

8.1. Tecnologías empleadas

Para la programación de la aplicación web se ha empleado Node.js como sistema backend, gestionando las rutas y las comunicaciones con el servidor REST. Se ha decidido separar la aplicación web del servicio REST para permitir que la API y la aplicación web se ejecuten en procesos diferentes.

Como en el servicio REST se ha empleado el framework Express y su sistema de plantillas basadas en el lenguaje JADE [35]. Para la maquetación de la página se han empleado HTML5, CSS3, bootstrap [36], Chart.js [37] y jQuery [38] para las comunicaciones con el servidor y las validaciones.

8.2. Autenticación

El servicio REST requiere autenticación para realizar todas las funciones, es decir, un usuario y una contraseña válidas. El funcionamiento de la aplicación web se basará en una autenticación obligatoria al entrar (ver Figura 36), Node.js y Express se encargarán de mantener la sesión sin la utilización de cookies.

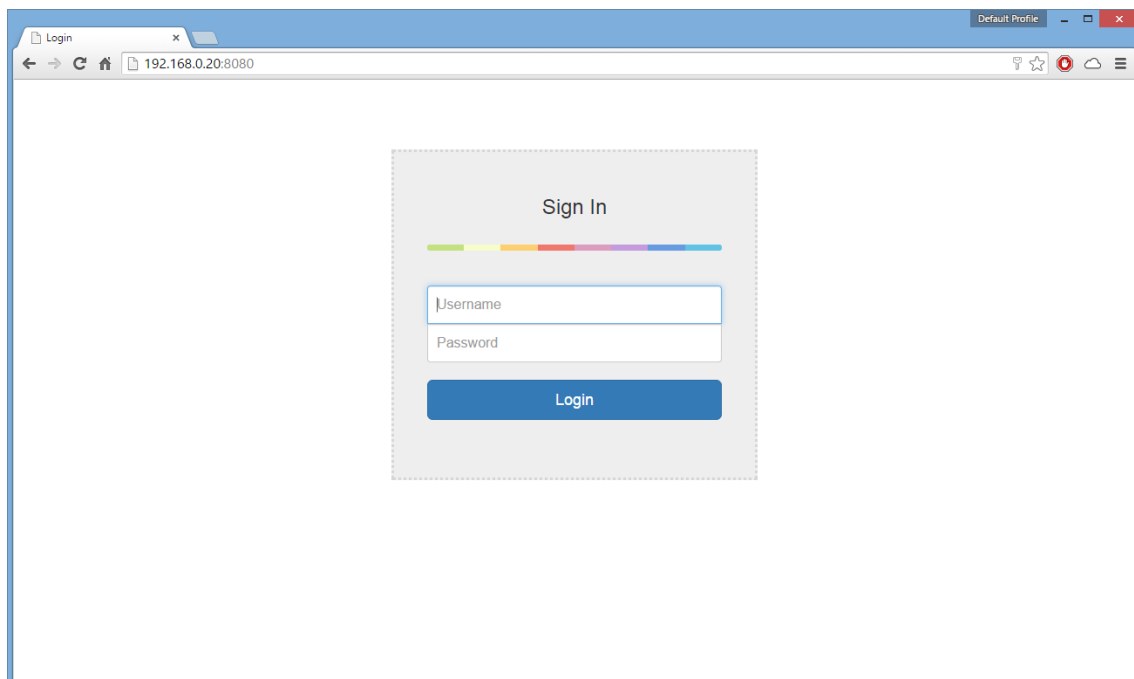


Figura 35: Panel de login de la aplicación web.

8.3. Funciones de la aplicación web

La aplicación web es un cliente que permite realizar todas las funciones programadas en el servidor REST mediante una interfaz visual.

Una vez autenticado en la página, el servidor nos redirecciona a /stations donde podemos consultar las estaciones meteorológicas registradas en el sistema (ver Figura 37) y registrar nuevas si poseemos los privilegios adecuados (ver Figura 38).

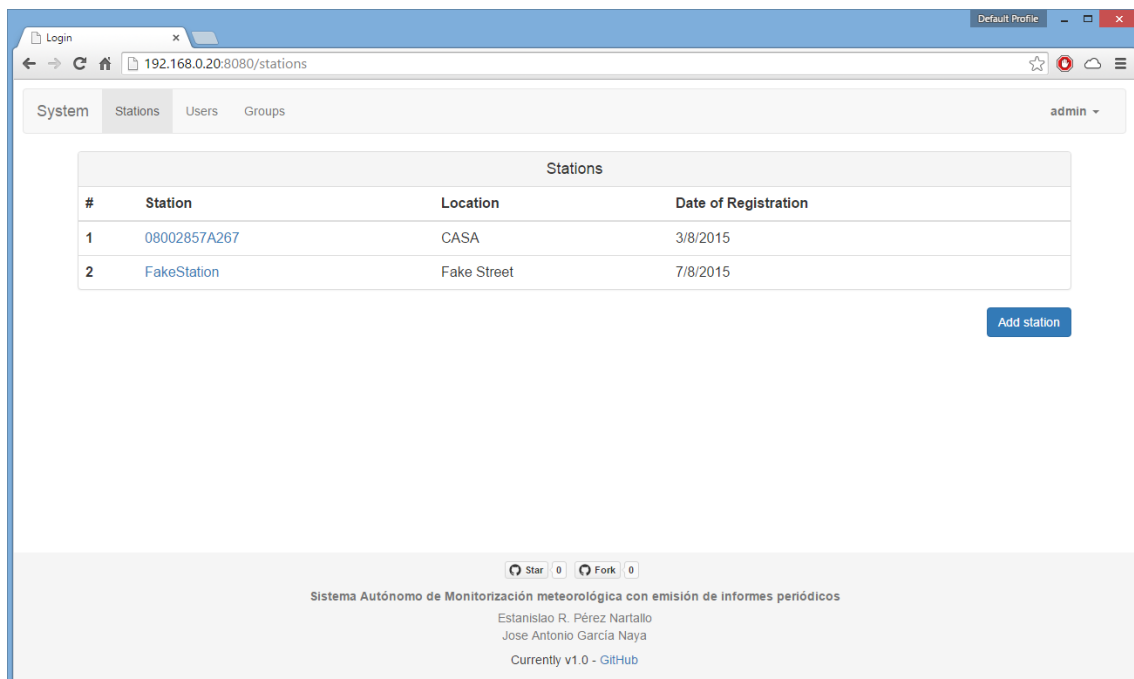


Figura 36: Estaciones meteorológicas registradas en el sistema.

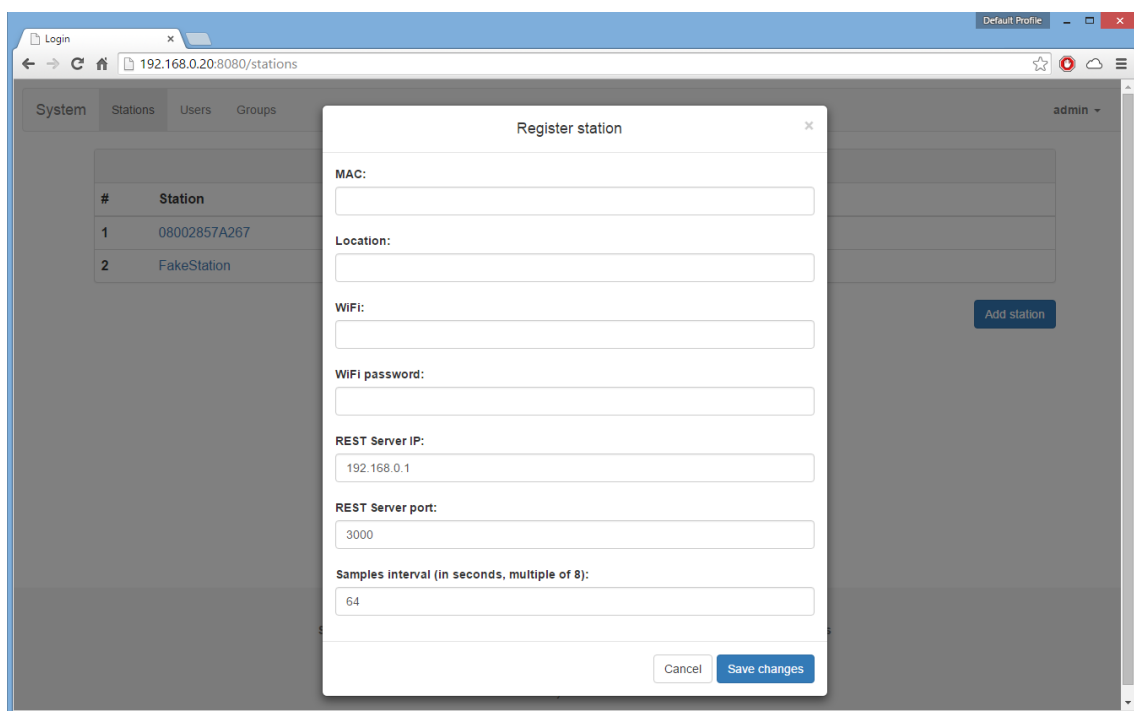


Figura 37: Panel para dar de alta una nueva estación.

Cada estación meteorológica permite consultar los valores de sus sensores que ha enviado al servicio REST. Se permiten dos tipos de vista: visualizar los valores gráficamente (ver Figuras 39 y 40) o en texto plano (ver Figura 41).

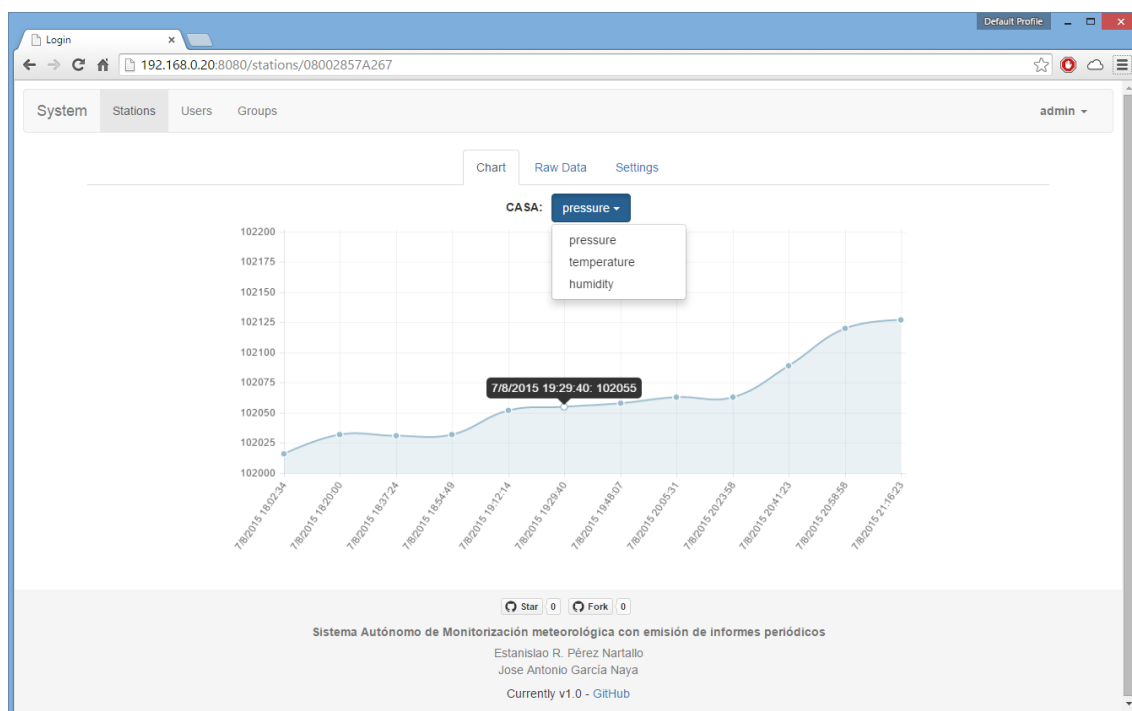


Figura 38: Representación gráfica de los valores de la presión atmosférica.

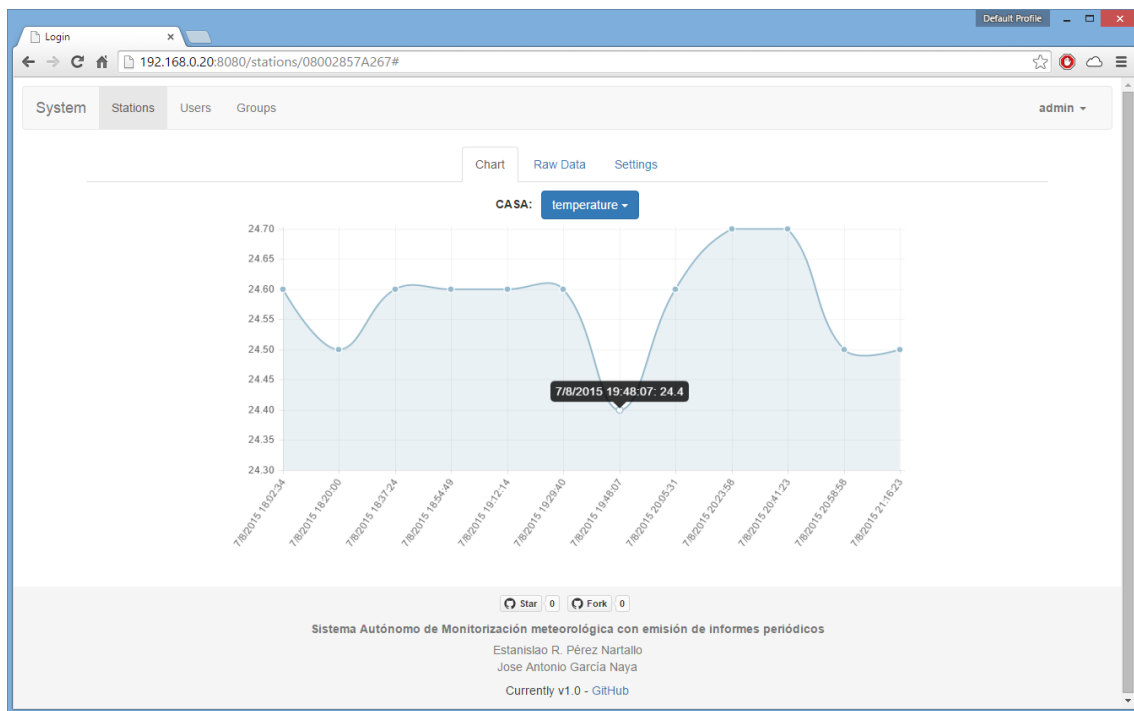


Figura 39: Representación gráfica de la variación de la temperatura.

Figure 40 is a screenshot of a web application displaying a table of meteorological samples. The table has columns for "#", "Sample Date", "pressure", "temperature", and "humidity". It contains 12 rows of data. The interface includes tabs for "Chart", "Raw Data", and "Settings", and a navigation bar with "System", "Stations", "Users", and "Groups". A "Delete samples" button is visible at the bottom right.

#	Sample Date	pressure	temperature	humidity
1	7/8/2015 18:02:34	102016	24.6	52.8
2	7/8/2015 18:20:00	102032	24.5	52.9
3	7/8/2015 18:37:24	102031	24.6	53.4
4	7/8/2015 18:54:49	102032	24.6	53.5
5	7/8/2015 19:12:14	102052	24.6	53.8
6	7/8/2015 19:29:40	102055	24.6	54.1
7	7/8/2015 19:48:07	102058	24.4	54.5
8	7/8/2015 20:05:31	102063	24.6	54.2
9	7/8/2015 20:23:58	102063	24.7	54.1
10	7/8/2015 20:41:23	102089	24.7	53.8
11	7/8/2015 20:58:58	102120	24.5	54.2
12	7/8/2015 21:16:23	102127	24.5	54.3

Figura 40: Visualización en texto plano de todas las muestras de la estación.

La vista en texto plano permite eliminar las muestras deseadas siempre que tengamos los permisos necesarios para ello. Esta visualización y la representación gráfica se han desarrollado sin tener en cuenta el tipo de unidad a representar, es decir, funcionaría perfectamente con cualquier estación meteorológica e independientemente de los sensores que tenga (respetando el formato que acepta la API v1).

Finalmente, la pestaña “Settings” nos permite cambiar los parámetros de configuración de la estación meteorológica (ver Figura 42), así como darla de baja del sistema.

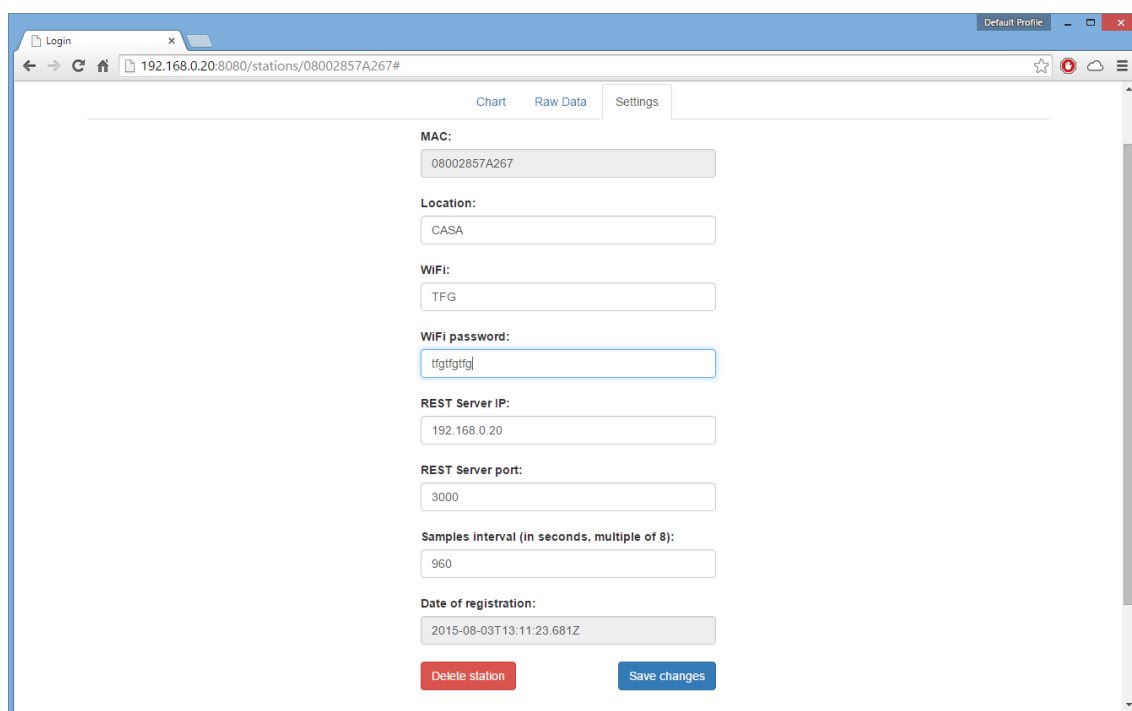
The image shows a web browser window with the address bar displaying '192.168.0.20:8080/stations/08002857A267#'. The page has three tabs: 'Chart', 'Raw Data', and 'Settings', with 'Settings' being the active tab. The settings form includes the following fields: 'MAC' (08002857A267), 'Location' (CASA), 'WiFi' (TFG), 'WiFi password' (tfgtfgtfgt), 'REST Server IP' (192.168.0.20), 'REST Server port' (3000), 'Samples interval (in seconds, multiple of 8)' (960), and 'Date of registration' (2015-08-03T13:11:23.681Z). At the bottom of the form are two buttons: 'Delete station' (red) and 'Save changes' (blue).

Figura 41: Parámetros de configuración de la estación.

Las funciones de editar los valores de configuración y eliminar la estación requieren privilegios de administrador. Toda esta lógica de los permisos está recogida en el servicio REST.

En la sección “Users” (ver Figura 43) podemos consultar los usuarios registrados en el sistema, registrar nuevos usuarios y eliminarlos del sistema.

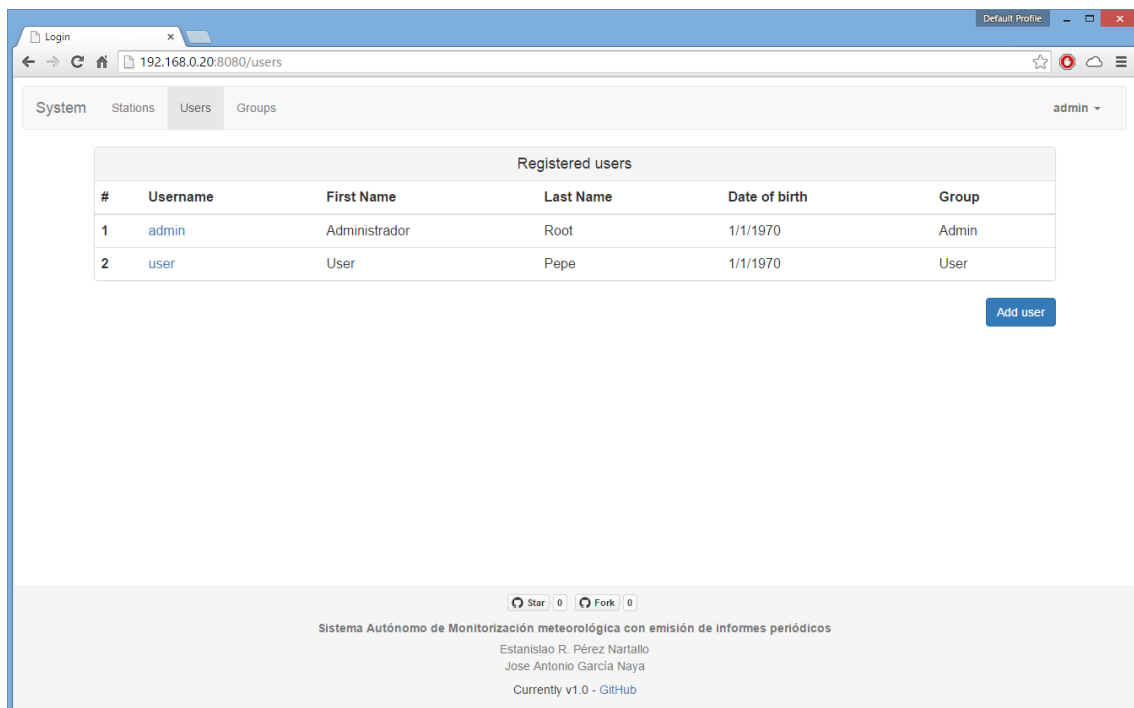


Figura 42: Visualización de los usuarios del sistema.

El botón “Add user” (ver Figura 44) despliega un panel flotante que nos permite añadir nuevos usuarios.

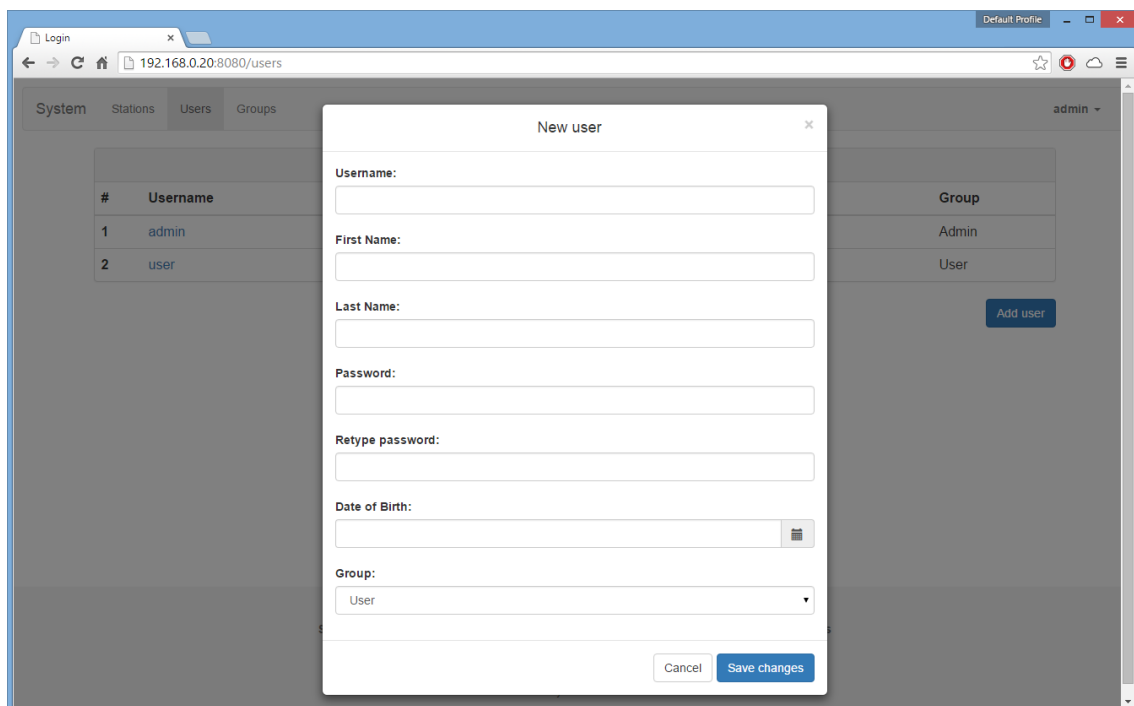


Figura 43: Añadir un usuario al sistema.

Este mismo panel (ver Figura 45) es aprovechado para consultar los datos de los usuarios al hacer click sobre ellos y darlos de baja del sistema.

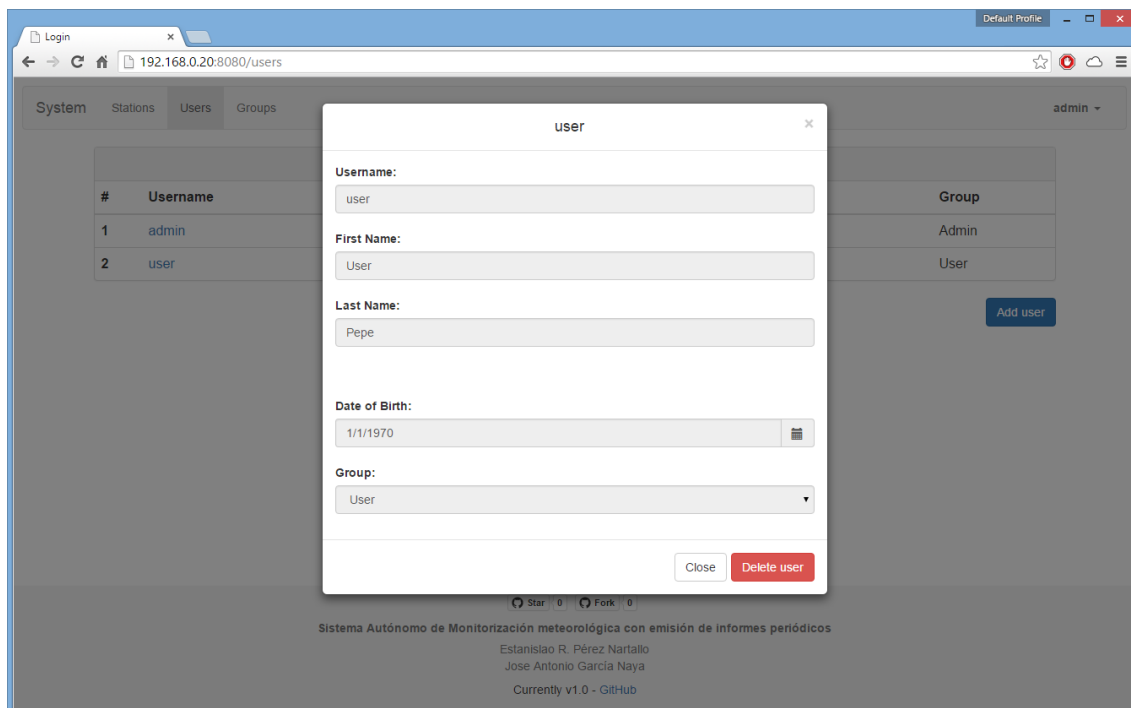


Figura 44: Visualización de los datos de un usuario.

En la sección “Groups” se pueden consultar los grupos de usuarios del sistema (ver Figura 46) y consultar los usuarios que tiene cada grupo (ver Figura 47).

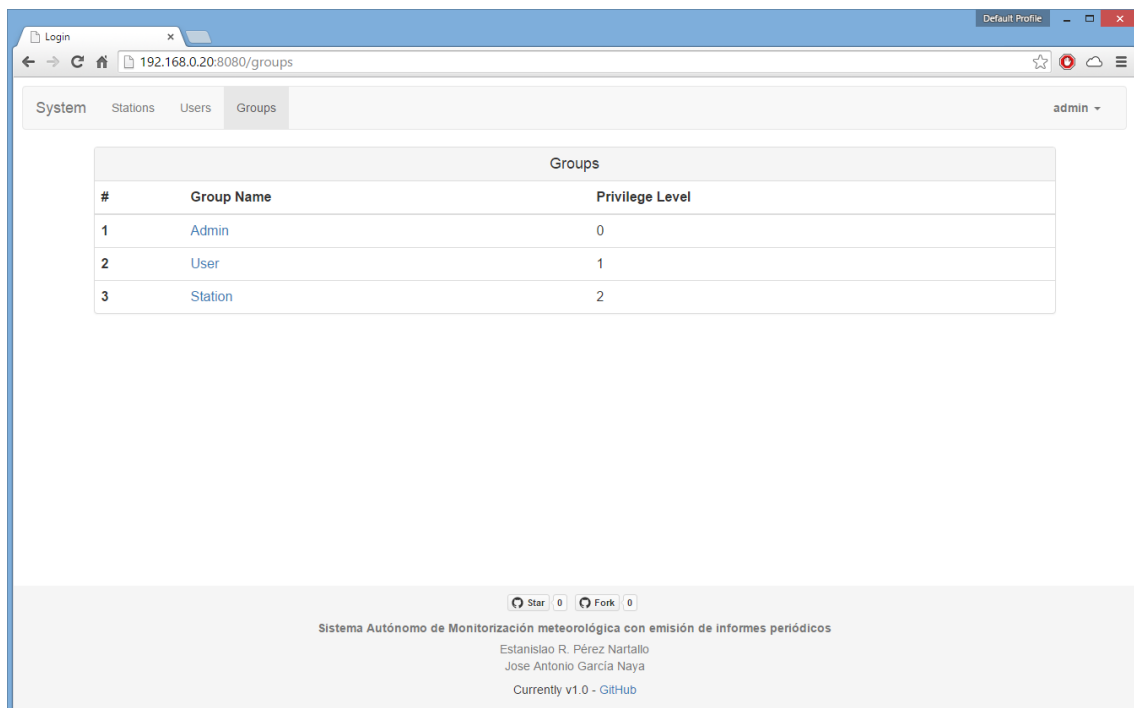


Figura 45: Grupos del sistema.

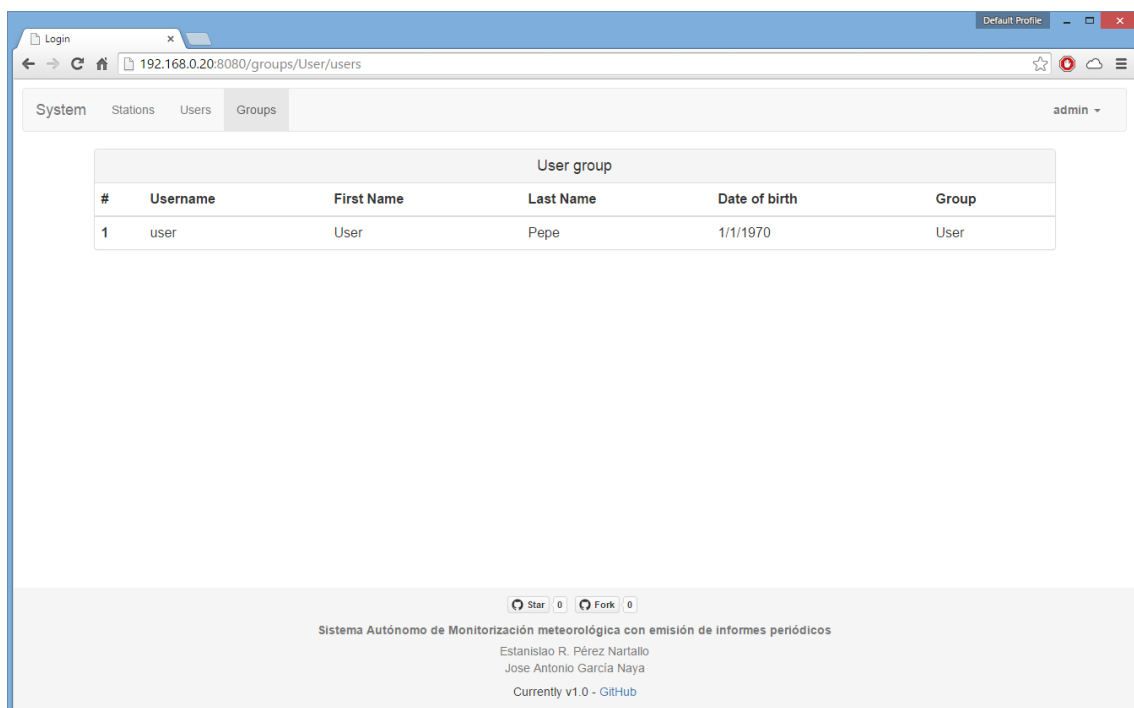


Figura 46: Usuarios del grupo User.

Capítulo 9

Conclusiones y trabajo futuro

9.1. Conclusiones

En este proyecto hemos conseguido alcanzar los principales objetivos marcados:

- Realizamos un estudio sobre el estado del arte en cuanto a tecnologías de comunicación inalámbricas y placas basadas en microcontroladores.
- Diseñamos un servicio REST empleando tecnologías modernas como son Node.js y MongoDB.
- Diseñamos y construimos una estación meteorológica empleando soluciones de hardware libre como Arduino.
- Desarrollamos un cliente web para probar el correcto funcionamiento de la API REST y visualizar de forma gráfica las medidas de la estación.
- Se realizó un estudio sobre el consumo energético de la estación.

Los principales objetivos del proyecto se han podido realizar en el tiempo estimado de forma satisfactoria. Estas son las conclusiones que hemos sacado a partir de los objetivos alcanzados:

- Se podrían haber empleado otras tecnologías de comunicación inalámbrica como Bluetooth LE o ZigBee. Su consumo energético es menor, y en el caso del primero, es muy barato.
- Para el servicio REST se podrían haber empleado otras soluciones como puede ser Java, empleando el framework Spring e Hibernate, pero consideramos que Node.js y MongoDB se adaptaban perfectamente a los requisitos del proyecto. Finalmente, el rendimiento del sistema es muy satisfactorio, ya que es capaz de funcionar de forma fluida en dispositivos de bajos recursos como puede ser una Raspberry Pi B+.
- Existen mejores diseños y otras soluciones hardware para las comunicaciones y para reducir el consumo energético. Este sería el objetivo a mejorar en futuros prototipos.
- La experiencia del cliente web en dispositivos móviles no es muy buena, ya que no se realizó con ese objetivo. El cliente web debería detectar si se trata de un dispositivo móvil y adaptar la vista de la web.

9.2. Trabajo futuro

En un futuro, se debería evaluar otras tecnologías de comunicación inalámbricas para realizar otros prototipos. También habría que optimizar el consumo energético de la estación meteorológica con otro diseño hardware más eficiente. Otra vía de trabajo futuro sería el uso de un panel fotovoltaico acompañado de una batería, de esta forma la estación meteorológica sería totalmente autónoma.

En cuanto al servicio REST, su diseño basado en versiones le permite añadir nuevas funcionalidades sin romper la compatibilidad con dispositivos y clientes antiguos. Se podrían añadir nuevas funciones dependiendo de los requisitos que surjan.

Sería interesante el desarrollo de más aplicaciones que empleen la API del servicio, como por ejemplo, Android o iOS.

En el cliente web, tendríamos como prioridad desarrollar una versión adaptada a dispositivos móviles, debido a su amplio uso. También habría que mejorar ciertos aspectos estéticos de la web.

A. API del servicio REST

En este apéndice se procede a documentar la API versión 1 del servicio REST, con todas sus funciones, descripción de las operaciones y códigos de estado.

GET /v1/login		
Realiza una petición de autenticación en el sistema empleando la cabecera Authorizaion con autenticación básica y cifrado base64.		
Requisitos	Nivel de privilegio	Todos
Petición	Cabecera	Contenido
	Authorization	Basic: Usuario:Contraseña
Respuesta	Código de estado	Cuerpo/descripción
	200 OK	Devuelve el documento del usuario en formato JSON, a excepción de la contraseña. Ejemplo: <pre>{ "login": "user", "data": { "name": "User", "secondName": "Pepe", "dateOfBirth": "1969-12-31T23:00:01.000Z" }, "groups": [{ "_id": "55bf687bb461d1dd046c6337", "groupName": "User", "privilegeLevel": 1 }] }</pre>
	401 Unauthorized	La autenticación ha fallado o no se ha especificado.
	500 Internal Server Error	Se produjo un error realizando la búsqueda en la base de datos.

Tabla 32: Función GET /v1/login.

GET /v1/groups		
Obtiene una lista con los grupos registrados en el sistema y su nivel de privilegio.		
Requisitos	Nivel de privilegio	0, 1
Petición	Cabecera	Contenido
	Authorization	Basic: Usuario:Contraseña

Respuesta	Código de estado	Cuerpo/descripción
	200 OK	Ejemplo: [{ "_id": "55bf6875b461d1dd046c6335", "groupName": "Admin", "privilegeLevel": 0 }, { "_id": "55bf687bb461d1dd046c6337", "groupName": "User", "privilegeLevel": 1 }, { "_id": "55bf687bb461d1dd046c6338", "groupName": "Station", "privilegeLevel": 2 }]
	401 Unauthorized	La autenticación ha fallado o no se ha especificado.
	403 Forbidden	La autenticación es válida, pero el usuario no tiene los permisos suficientes para realizar la acción.
	500 Internal Server Error	Se produjo un error realizando la búsqueda en la base de datos.

Tabla 33: Función GET /v1/groups.

GET /v1/groups/:groupName		
Obtiene información sobre un grupo.		
Requisitos	Nivel de privilegio	0, 1
Petición	Cabecera	Contenido
	Authorization	Basic: Usuario:Contraseña
Respuesta	Código de estado	Cuerpo/descripción
	200 OK	Ejemplo: { "_id": "55bf6875b461d1dd046c6335", "groupName": "Admin", "privilegeLevel": 0 }
	401 Unauthorized	La autenticación ha fallado o no se ha especificado.
	403 Forbidden	La autenticación es válida, pero el usuario no tiene los permisos suficientes para realizar la

		acción.
	404 Not Found	El grupo no existe.
	500 Internal Server Error	Se produjo un error realizando la búsqueda en la base de datos.

Tabla 34: Función GET /v1/groups/:groupName.

GET /v1/groups/:groupName/users		
Obtiene los usuarios de un grupo.		
Requisitos	Nivel de privilegio	0, 1
Petición	Cabecera	Contenido
	Authorization	Basic: Usuario:Contraseña
Respuesta	Código de estado	Cuerpo/descripción
	200 OK	Ejemplo: <pre>[{ "login": "user", "data": { "name": "User", "secondName": "Pepe", "dateOfBirth": "1969-12-31T23:00:01.000Z" }, "groups": [{ "_id": "55bf687bb461d1dd046c6337", "groupName": "User", "privilegeLevel": 1 }] }, {...}, ...]</pre>
	401 Unauthorized	La autenticación ha fallado o no se ha especificado.
	403 Forbidden	La autenticación es válida, pero el usuario no tiene los permisos suficientes para realizar la acción.
	404 Not Found	El grupo no existe.
	500 Internal Server Error	Se produjo un error realizando la búsqueda en la base de datos.

Tabla 35: Función GET /v1/groups/:groupName/users.

GET /v1/stations		
Obtiene una lista con las estaciones meteorológicas registradas en el sistema.		
Requisitos	Nivel de privilegio	0, 1
Petición	Cabecera	Contenido
	Authorization	Basic: Usuario:Contraseña
Respuesta	Código de estado	Cuerpo/descripción
	200 OK	Ejemplo: <pre>[{ "login": "08002857A267", "data": { "location": "CASA", "wifi": "TFG", "wifiPassword": "tfgtfgtfg", "ip": "192.168.0.20", "port": 3000, "interval": 960, "dateOfRegistration": "2015-08-03T13:11:23.681Z" }, "groups": [{ "_id": "55bf687bb461d1dd046c6338", "groupName": "Station", "privilegeLevel": 2 }] }, {...}, ...]</pre>
	401 Unauthorized	La autenticación ha fallado o no se ha especificado.
	403 Forbidden	La autenticación es válida, pero el usuario no tiene los permisos suficientes para realizar la acción.

Tabla 36: Función GET /v1/stations.

GET /v1/stations/:MAC		
Obtiene información sobre la estación meteorológica especificada.		
Requisitos	Nivel de privilegio	0, 1, 2

Petición	Cabecera	Contenido
	Authorization	Basic: Usuario:Contraseña
Respuesta	Código de estado	Cuerpo/descripción
	200 OK	Ejemplo: <pre>{ "login": "08002857A267", "data": { "location": "CASA", "wifi": "TFG", "wifiPassword": "tfgtfgtfg", "ip": "192.168.0.20", "port": 3000, "interval": 960, "dateOfRegistration": "2015-08-03T13:11:23.681Z" }, "groups": [{ "_id": "55bf687bb461d1dd046c6338", "groupName": "Station", "privilegeLevel": 2 }] }</pre>
	401 Unauthorized	La autenticación ha fallado o no se ha especificado.
	403 Forbidden	La autenticación es válida, pero el usuario no tiene los permisos suficientes para realizar la acción.
	404 Not Found	La estación meteorológica no existe.
	500 Internal Server Error	Se produjo un error realizando la búsqueda en la base de datos.

Tabla 37: Función GET /v1/stations/:MAC.

GET /v1/stations/:MAC/data		
Obtiene las muestras enviadas por la estación meteorológica especificada.		
Requisitos	Nivel de privilegio	0, 1
Petición	Cabecera	Contenido
	Authorization	Basic: Usuario:Contraseña
Respuesta	Código de estado	Cuerpo/descripción
	200 OK	Ejemplo: <pre>[{</pre>

		<pre> "_id": "55c4d69a92abd23a046177f7", "MAC": "08002857A267", "samples": [{ "pressure": 102016, "temperature": 24.6, "humidity": 52.8 }], "sampleDate": "2015-08-07T16:02:34.969Z" }, { "_id": "55c4dab092abd23a046177f8", "MAC": "08002857A267", "samples": [{ "pressure": 102032, "temperature": 24.5, "humidity": 52.9 }], "sampleDate": "2015-08-07T16:20:00.040Z" }, {..}] </pre>
	401 Unauthorized	La autenticación ha fallado o no se ha especificado.
	403 Forbidden	La autenticación es válida, pero el usuario no tiene los permisos suficientes para realizar la acción.
	404 Not Found	La estación meteorológica no existe.
	500 Internal Server Error	Se produjo un error realizando la búsqueda en la base de datos.

Tabla 38: Función GET /v1/stations/:MAC/data.

GET /v1/stations/:MAC/data/:id		
Obtiene una muestras específica de una estación meteorológica.		
Requisitos	Nivel de privilegio	0, 1
Petición	Cabecera	Contenido
	Authorization	Basic: Usuario:Contraseña
Respuesta	Código de estado	Cuerpo/descripción
	200 OK	Ejemplo:

		<pre>{ "_id": "55c4d69a92abd23a046177f7", "MAC": "08002857A267", "samples": [{ "pressure": 102016, "temperature": 24.6, "humidity": 52.8 }], "sampleDate": "2015-08-07T16:02:34.969Z" }</pre>
	401 Unauthorized	La autenticación ha fallado o no se ha especificado.
	403 Forbidden	La autenticación es válida, pero el usuario no tiene los permisos suficientes para realizar la acción.
	404 Not Found	La estación meteorológica o la muestra no existen.

Tabla 39: Función GET /v1/stations/:MAC/data/:id.

DELETE /v1/stations/:MAC/data/:id		
Elimina una muestras específica de una estación meteorológica.		
Requisitos	Nivel de privilegio	0
Petición	Cabecera	Contenido
	Authorization	Basic: Usuario:Contraseña
Respuesta	Código de estado	Cuerpo/descripción
	200 OK	Muestra eliminada.
	401 Unauthorized	La autenticación ha fallado o no se ha especificado.
	403 Forbidden	La autenticación es válida, pero el usuario no tiene los permisos suficientes para realizar la acción.
	404 Not Found	La estación meteorológica o la muestra no existen.

Tabla 40: Función DELETE /v1/stations/:MAC/data/:id.

GET /v1/users		
Obtiene una lista con los usuarios registrados en el sistema.		
Requisitos	Nivel de privilegio	0, 1
Petición	Cabecera	Contenido

	Authorization	Basic: Usuario:Contraseña
Respuesta	Código de estado	Cuerpo/descripción
	200 OK	Ejemplo: <pre>[{ "_id": "55bf687bb461d1dd046c6336", "login": "admin", "password": "admin", "data": { "name": "Administrador", "secondName": "Root", "dateOfBirth": "1969-12-31T23:00:01.000Z" }, "groups": [{ "_id": "55bf6875b461d1dd046c6335", "groupName": "Admin", "privilegeLevel": 0 }], "_type": "Person" }, { ... },]</pre>
	401 Unauthorized	La autenticación ha fallado o no se ha especificado.
	403 Forbidden	La autenticación es válida, pero el usuario no tiene los permisos suficientes para realizar la acción.
	500 Internal Server Error	Se produjo un error realizando la búsqueda en la base de datos.

Tabla 41: Función GET /v1/users.

POST /v1/users		
Añade un usuario al sistema.		
Requisitos	Nivel de privilegio	0
	Body	Ejemplo: <pre>{ "login" : "usuario", "password" : "password", "groups" : [{ "_id" :</pre>

		<pre>"55bf687bb461d1dd046c6337" }, "data" : { "name" : "Nombre", "secondName" : "Apellidos", "dateOfBirth" : "1969-12-31T23:00:01.000Z" } }</pre>
Petición	Cabecera	Contenido
	Authorization	Basic: Usuario:Contraseña
Respuesta	Código de estado	Cuerpo/descripción
	201 Created	Usuario creado correctamente.
	400 Bad Request	Formato incorrecto.
	401 Unauthorized	La autenticación ha fallado o no se ha especificado.
	403 Forbidden	La autenticación es válida, pero el usuario no tiene los permisos suficientes para realizar la acción.

Tabla 42: Función POST /v1/users.

PUT /v1/users/:login		
Actualiza la información personal del propio usuario. No son obligatorios todos los campos.		
Requisitos	Nivel de privilegio	0, 1
	Body	Ejemplo: <pre>{ "password" : "new password", "data" : { "name" : "Nombre", "secondName" : "Apellidos", "dateOfBirth" : "1969-12-31T23:00:01.000Z" } }</pre>
Petición	Cabecera	Contenido
	Authorization	Basic: Usuario:Contraseña
Respuesta	Código de estado	Cuerpo/descripción
	200 OK	Devuelve el documento del usuario en formato JSON, a excepción de la contraseña. Ejemplo: <pre>{ "login": "user", "data": {</pre>

		<pre> "name": "User", "secondName": "Pepe", "dateOfBirth": "1969-12-31T23:00:01.000Z", }, "groups": [{ "_id": "55bf687bb461d1dd046c6337", "groupName": "User", "privilegeLevel": 1 }] </pre>
	400 Bad Request	Formato incorrecto.
	401 Unauthorized	La autenticación ha fallado o no se ha especificado.
	403 Forbidden	La autenticación es válida, pero el usuario no tiene los permisos suficientes para realizar la acción.
	404 Not Found	El usuario especificado no existe.

Tabla 43: Función PUT /v1/users/:login.

DELETE /v1/users/:login		
Elimina un usuario del sistema.		
Requisitos	Nivel de privilegio	0
Petición	Cabecera	Contenido
	Authorization	Basic: Usuario:Contraseña
Respuesta	Código de estado	Cuerpo/descripción
	200 OK	Usuario eliminado.
	401 Unauthorized	La autenticación ha fallado o no se ha especificado.
	403 Forbidden	La autenticación es válida, pero el usuario no tiene los permisos suficientes para realizar la acción.
	404 Not Found	El usuario especificado no existe.
	500 Internal Server Error	Se produjo un error añadiendo el usuario a la base de datos.

Tabla 44: Función DELETE /v1/users/:login.

POST /v1/stations
Registra una estación en el sistema.

Requisitos	Nivel de privilegio	0
Petición	Cabecera	Contenido
	Authorization	Basic: Usuario:Contraseña
	Body	Ejemplo: <pre>{ "login": "MAC", "password": "MAC al revés", "data": { "location": "CASA", "wifi": "miWiFi", "wifiPassword": "ContraseñaWiFi", "ip": "192.168.0.1", "port": 3000, "interval": 640 }, "groups": [{ "_id": "55bf687bb461d1dd046c6338", }] }</pre>
Respuesta	Código de estado	Cuerpo/descripción
	201 Created	Estación registrada correctamente.
	400 Bad Request	Formato incorrecto.
	401 Unauthorized	La autenticación ha fallado o no se ha especificado.
	403 Forbidden	La autenticación es válida, pero el usuario no tiene los permisos suficientes para realizar la acción.

Tabla 45: Función POST /v1/stations.

DELETE /v1/stations/:MAC		
Elimina una estación meteorológica del sistema.		
Requisitos	Nivel de privilegio	0
Petición	Cabecera	Contenido
	Authorization	Basic: Usuario:Contraseña
Respuesta	Código de estado	Cuerpo/descripción
	200 OK	Se ha eliminado la estación del sistema.
	401 Unauthorized	La autenticación ha fallado o no se ha especificado.
	403 Forbidden	La autenticación es válida, pero el usuario no tiene los permisos suficientes para realizar la acción.
	404 Not Found	La estación meteorológica especificada no existe.

	500 Internal Server Error	Se produjo un error eliminando la estación meteorológica de la base de datos.
--	---------------------------	---

Tabla 46: Función DELETE /v1/stations/:MAC.

PUT /v1/stations/:MAC		
Modifica los parámetros de configuración de la estación meteorológica. No son obligatorios todos los campos.		
Requisitos	Nivel de privilegio	0
Petición	Cabecera	Contenido
	Authorization	Basic: Usuario:Contraseña
	Body	Ejemplo: <pre>{ "data": { "location": "CASA", "wifi": "miWiFi", "wifiPassword": "ContraseñaWiFi", "ip": "192.168.0.1", "port": 3000, "interval": 640 } }</pre>
Respuesta	Código de estado	Cuerpo/descripción
	200 OK	Devuelve el documento de la estación meteorológica actualizado. Ejemplo: <pre>{ "login": "08002857A267", "data": { "location": "CASA", "wifi": "TFG", "wifiPassword": "tfgtfgtfg", "ip": "192.168.0.20", "port": 3000, "interval": 960, "dateOfRegistration": "2015-08-03T13:11:23.681Z" }, "groups": [{ "_id": "55bf687bb461d1dd046c6338", "groupName": "Station", "privilegeLevel": 2 }] }</pre>

		}
	400 Bad Request	Formato incorrecto.
	401 Unauthorized	La autenticación ha fallado o no se ha especificado.
	403 Forbidden	La autenticación es válida, pero el usuario no tiene los permisos suficientes para realizar la acción.
	404 Not Found	La estación meteorológica no existe.

Tabla 47: Función PUT /v1/stations/:MAC.

POST /v1/stations/:MAC/data		
Añade muestras de los sensores de la estación meteorológica.		
Requisitos	Nivel de privilegio	2
Petición	Cabecera	Contenido
	Authorization	Basic: Usuario:Contraseña
	Body	Ejemplo: <pre>{ "MAC": "MAC", "data": { "temperature": "28", "humidity": "55", "preassure": "10222", ... } }</pre>
Respuesta	Código de estado	Cuerpo/descripción
	201 Created	La muestra se ha añadido satisfactoriamente.
	400 Bad Request	Formato incorrecto.
	401 Unauthorized	La autenticación ha fallado o no se ha especificado.
	403 Forbidden	La autenticación es válida, pero el usuario no tiene los permisos suficientes para realizar la acción.
	404 Not Found	La estación meteorológica no existe.

Tabla 48: Función POST /v1/stations/:MAC/data.

B. Instalación del software en una Raspberry Pi

Se ha escogido una Raspberry Pi B+ para la instalación del software, el servicio REST y aplicación web, ya que cuenta con la potencia necesaria para el correcto funcionamiento del software. Además, con un consumo de unos 4W, la Raspberry Pi es perfecta para estar en continuo funcionamiento y recibir las muestras de la estación meteorológica.

Como sistema operativo se ha seleccionado Arch Linux ARM [39], debido a la experiencia del autor con esta distribución. Pero el proceso de instalación sería muy similar en otras distribuciones que empleen *systemd* como sistema de arranque.

En la propia web de Arch Linux ARM vienen detallados los pasos para su instalación, que consiste básicamente en crear las particiones del sistema en la tarjeta MicroSD, descargar la imagen del sistema y descomprimirla.

Una vez instalado el sistema, deberemos instalar las siguientes dependencias:

```
sudo pacman -Syu
sudo pacman -S mongodb nodejs npm forever
```

A continuación, copiamos el software a la MicroSD de la Raspberry Pi, que consiste en dos carpetas: rest y web. Las copiamos a la ruta /usr/bin con los permisos adecuados. Dentro de cada una de ellas deberemos instalar las dependencias propias de cada proyecto, para ello empleamos el comando npm:

```
npm install
```

El proceso de instalación puede tardar un rato debido a la cantidad de dependencias. Una vez finalizado el proceso de instalación, debemos crear la base de datos en MongoDB, para ello, basta con ejecutar el script:

```
# node /usr/bin/rest/install.js
```

Este script crea la base de datos, los grupos de usuarios y dos usuarios en el sistema:

- El usuario admin, con contraseña admin.
- El usuario user con contraseña user, una cuenta con permisos limitados.

Finalmente podemos comprobar el funcionamiento del servicio REST del siguiente modo:

```
/usr/bin/rest# npm start
```

El servicio REST y la aplicación web están programados para funcionar de forma interactiva, es decir, no se ejecutan en segundo plano. Para que se ejecuten al arrancar el sistema como *daemons* emplearemos la herramienta forever. Este programa se encarga de ejecutar en segundo plano aplicaciones desarrolladas en Nodejs y de capturar sus logs para redirigirlos a ficheros de texto plano. También se encarga de detectar errores y reiniciar los servicios si es necesario.

Primero deberemos crear la carpeta donde se almacenarán los logs:

```
# mkdir /var/log/tfg
```

A continuación debemos crear los servicios para systemd, para ello crearemos dos ficheros en /etc/systemd/system/.

```
[Unit]
Description=REST Service
After=network.target mongod.service

[Service]
Type=forking
WorkingDirectory=/usr/bin/rest
ExecStart=/usr/bin/forever -o /var/log/tfg/rest.log -e /var/log/tfg/rest-error.log start
./bin/www
ExecStop=/usr/bin/forever stop app.js
Restart=always
StandardOutput=syslog
StandardError=syslog
SyslogIdentifier=rest-service
Environment=NODE_ENV=production

[Install]
WantedBy=multi-user.target
```

Tabla 49: Contenido del fichero rest.service.

```
[Unit]
Description=Web Client
After=network.target mongod.service rest.service
```



```

[Service]
Type=forking
WorkingDirectory=/usr/bin/web
ExecStart=/usr/bin/forever -o /var/log/tfg/web.log -e /var/log/tfg/web-error.log start
./bin/www
ExecStop=/usr/bin/forever stop app.js
Restart=always
StandardOutput=syslog
StandardError=syslog
SyslogIdentifier=web-client
Environment=NODE_ENV=production

[Install]
WantedBy=multi-user.target

```

Tabla 50: Contenido del fichero web.service.

En estos ficheros de configuración se especifican las rutas de los programas, parámetros de arranque y parada, dependencias [40], etc... En el servicio REST se especifica que se debe de ejecutar el servicio después de los servicios de red y de MongoDB [41], ya que si se ejecuta antes se produciría un error. Ocurre la misma situación en el servicio web, pero además, necesita que se ejecute antes el servicio REST.

Finalmente, activamos para que se carguen en el arranque y ejecutamos los servicios:

```

# systemctl enable rest.service
# systemctl enable web.service
# systemctl start rest.service
# systemctl start web.service

```


Bibliografía

- [0] Github. “tanisperez/tfg” [Online]. Disponible. <https://github.com/tanisperez/tfg>
- [1] Wikipedia. “GPRS” [Online]. Disponible.
https://es.wikipedia.org/wiki/Servicio_general_de_paquetes_v%C3%ADa_radio
- [2] Sparkfun. “GSM/GPRS Module SM5100B” [Online]. Disponible.
<https://www.sparkfun.com/products/9533>
- [3] IEEE. “IEEE Std 802.11” [Online]. Disponible.
<http://standards.ieee.org/getieee802/download/802.11-2012.pdf>
- [4] IEEE. “IEEE 802.3 Ethernet” [Online]. Disponible.
<https://standards.ieee.org/about/get/802/802.3.html>
- [5] Wikipedia. “Wireless Security” [Online]. Disponible.
https://en.wikipedia.org/wiki/Wireless_security
- [6] Wiznet. “WizFi210” [Online]. Disponible.
<http://www.wiznet.co.kr/product-item/wizfi210/>
- [7] IEEE. “IEEE 802.15.4 Wireless Personal Area Networks (PANs)” [Online].
<https://standards.ieee.org/about/get/802/802.15.html>
- [8] Wikipedia. “ZigBee” [Online]. Disponible
<https://en.wikipedia.org/wiki/ZigBee>
- [9] Sparkfun. “XBee 2mW Wire Antenna” [Online]. Disponible.
<https://www.sparkfun.com/products/10414>
- [10] Blogthinkbig, Telefónica. “IPv6: El motor de la Web de las cosas” [Online].
<http://blogthinkbig.com/ipv6-motor-internet-de-las-cosas-iot/>
- [11] Wikipedia. “Constrained Application Protocol” [Online]. Disponible.
https://en.wikipedia.org/wiki/Constrained_Application_Protocol
- [12] Libelium. “Wasp mote Mote Runner” [Online]. Disponible.
<http://www.libelium.com/es/products/waspmote-mote-runner-6lowpan>
- [13] Developer Bluetooth. “Bluetooth Low Energy Technology” [Online]. Disponible.
<https://developer.bluetooth.org/TechnologyOverview/Pages/BLE.aspx>
- [14] Adafruit. “Bluefruit LE – Bluetooth Low Energy” [Online]. Disponible.
<http://www.adafruit.com/products/1697>

- [15] Raspberrypi. “Raspberry Pi” [Online]. Disponible.
<https://www.raspberrypi.org/>
- [16] Hardkernel. “ODROID | Hardkernel” [Online]. Disponible.
<http://www.hardkernel.com/main/main.php>
- [17] Cubieboard. “Cubieboard | A series of open source hardware” [Online].
<http://cubieboard.org/>
- [18] Arduino. “Arduino” [Online]. Disponible.
<https://www.arduino.cc/>
- [19] Logos-electro. “Zigduino” [Online]. Disponible.
<http://www.logos-electro.com/zigduino/>
- [20] Libelium. “Waspnote” [Online]. Disponible.
<http://www.libelium.com/es/products/waspnote/>
- [21] PJRC, “Teensy USB Development Board” [Online]. Disponible.
<https://www.pjrc.com/teensy/>
- [22] Freescale. “Freescale Freedom Development Platform” [Online]. Disponible.
http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=FRDM-KL25Z
- [23]. Mcuoneclipse. “Tutorial: Using the FRDM-KL25Z as Low Power Board” [Online]. Disponible.
<http://mcuoneclipse.com/2013/10/20/tutorial-using-the-frdm-kl25z-as-low-power-board/>
- [24] Wikipedia. “Basic access authentication” [Online]. Disponible.
https://en.wikipedia.org/wiki/Basic_access_authentication
- [25] Sparkfun. “WiFi Module – ESP8266” [Online]. Disponible.
<https://www.sparkfun.com/products/13678>
- [26] Adafruit. “Adafruit CC3000 WiFi Shield with uFL Connector for Ext Antenna” [Online]. Disponible. <https://www.adafruit.com/products/1534>
- [27] Adafruit. “DHT22 temperature-humidity sensor” [Online]. Disponible.
<https://www.adafruit.com/products/385>
- [28] Adafruit. “BMP180 Barometric Pressure/Temperature/Altitude Sensor” [Online]. Disponible. <https://www.adafruit.com/products/1603>
- [29] Wikipedia. “Node.js” [Online]. Disponible. <https://en.wikipedia.org/wiki/Node.js>

- [30] George Ornbo, “Node.js”. Anaya Multimedia, Edición (15 de enero de 2013).
- [31] Wikipedia. “MongoDB” [Online]. Disponible. <https://en.wikipedia.org/wiki/MongoDB>
- [32] MongoDB. “NoSQL Databases Explained” [Online]. Disponible. <https://www.mongodb.com/nosql-explained>
- [33] CITIC. “Centro de Investigación de Tecnologías TIC” [Online]. Disponible. <http://citic-research.org/>
- [34] Adafruit. “Low Power WiFi Datalogger with upgraded hardware” [Online]. Disponible. <https://learn.adafruit.com/low-power-wifi-datalogging/upgraded-hardware>
- [35] Expressjs. “Using templete engines with Express” [Online]. Disponible. <http://expressjs.com/guide/using-template-engines.html>
- [36] GetBootstrap. “Bootstrap” [Online]. Disponible. <http://getbootstrap.com/>
- [37] Chart.js. “Chart.js – Open Source HTML5 charts for your website” [Online]. Disponible. <http://www.chartjs.org/>
- [38] jQuery. “jQuery” [Online]. Disponible. <https://jquery.com/>
- [39] Arch Linux ARM. “Raspberry Pi Arch Linux ARM” [Online]. Disponible. <http://archlinuxarm.org/platforms/armv6/raspberry-pi>
- [40] Arch Linux Wiki. “Systemd – Handling dependencies” [Online]. Disponible. https://wiki.archlinux.org/index.php/Systemd#Handling_dependencies
- [41] StackOverflow. “Systemd start service after specific service” [Online]. Disponible. <http://stackoverflow.com/questions/21830670/systemd-start-service-after-specific-service>