



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Politècnica Superior d'Alcoi
Universitat Politècnica de València

Implementación de sensores geolocalizados y una aplicación para la obtención de datos en un área metropolitana

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: David Rodriguez Martinez

Tutor: David Cuesta Frau

Curso 2015-2016

Resumen

El motivo de este proyecto es utilizar una plataforma de "hardware libre" llamada Arduino con el fin de implementar uno o varios nodos de sensores el cual enviará a través de la red Información sobre un área específica, y podremos monitorizar la información gracias a una aplicación web.

Mi intención es que gracias a lo explicado en el párrafo anterior se pueda implementar una utilidad hardware y una Web para utilizar en una Smart City

Palabras clave: Ciudad Inteligente, Arduino, Hardware, Geolocalizacion, Sensores, Hardware Libre, Electronica, Servicios Web

Resum

El motiu d'aquest projecte es utilitzar una plataforma de "hardware lliure" anomenada arduino amb l'intenció d'implementar u o varios nodes de sensors, el qual enviara utilitzant una xarxa, informació sobre una àrea específica la qual podrém monitoritzar gracies a una aplicació web .

La meua intenció es que gracies a el que he explicat en el parraf anterior es puga implementar una utilitat de gestió per utilitzar en una Smart City

Paraules clau: Ciutat Inteligent, Arduino, Hardware, Geolocalizació, Sensors, Harware Lliure, Electrònica, Servicis Web

Abstract

The purpose of this project is use a special Open-source hardware platform called Arduino, with the intention of implement one or more nodes of sensors, which will send on the network, information about a specific area that we can monitoring, using a web service.

My intention is that thanks to I've explained in the last paragraph, I will can implement a hardware and web application to track that information to use in the future in the Smart city.

Key words: Arduino, Smart City, IoT, Remote Sensor, Open-source hardware

Índice general

Índice general	v
Índice de figuras	vii
Índice de tablas	vii
<hr/>	
Listings	viii
1 Introducción	1
1.1 Objetivos	1
2 Estudio del Arte	3
2.1 ¿Porque Arduino?	3
2.2 Presupuesto	4
3 Descripción del proyecto	7
3.1 Tecnologías utilizadas	7
3.1.1 Arduino	7
3.1.2 GPS y Nmea Data	8
3.1.3 Nmea Data	8
3.1.4 Formato del NMEA Data	8
3.1.5 Laravel	9
3.1.6 MySQL	9
3.1.7 Bootstrap	9
3.1.8 Chart.js	9
4 Montaje del Hardware y el Servicio Web	11
4.1 Material	12
4.2 Montaje del Nodo Arduino DUE	14
4.3 Programación de la Microcontroladora Arduino DUE	16
4.4 Instalación del Servicio WEB	22
4.4.1 Instalación en MacOSX	22
4.4.2 Instalación en Linux	24
5 Explicación de la aplicación	25
5.1 Gestión de la información	26
5.2 Gestión del <i>NodeTracker</i>	27
5.3 Visualización de la información	29
5.3.1 Posicionamiento	29
5.3.2 Obtener un informe	29
5.3.3 Seguimiento de la información	30
6 Conclusiones y mejoras	31
6.1 Seguridad	31
6.2 Sensores	31
Bibliografía	33

Índice de figuras

3.1 Logo Arduino	7
3.2 Logo de Laravel 5	9
4.1 Diagrama de montaje del proyecto	11
4.2 Microcontroladora Arduino DUE [2]	12
4.3 Ethernet shield [2]	12
4.4 Sensor de Temperatura y Humedad DHT11	12
4.5 Sensor de Temperatura y Humedad DHT11	13
4.6 Sensor de Mesura de Gas CO MQ-7	13
4.7 Sensor de Luminosidad y Ruido	14
4.8 Esquema del montaje del Nodo	15
4.9 Esquema del circuito para el GPS en el Arduino UNO	15
4.10 Montaje final del nodo de sensores	16
4.11 String JSON generada por la petición GET de Arduino	22
4.12 Configuración del Servidor MAMP	23
5.1 Esquema del funcionamiento de la aplicación	25
5.2 Diagrama Entidad-Relación de la base de datos	26
5.3 Login del NodeTracker	27
5.4 Home del NodeTracker	27
5.5 Formulario de edición de usuarios del <i>NodeTracker</i>	28
5.6 Formulario para añadir un nuevo nodo al sistema.	28
5.7 Lista de nodos disponibles	28
5.8 Pagina de monitorización de la aplicación	29
5.9 Generación de Informes	29
5.10 Grafica perteneciente a la temperatura en el seguimiento	30

Índice de tablas

2.1 Consumo de los diferentes dispositivos	3
--	---

Listings

3.1	Cadena Hexadecima del NMEA sin Conexión	8
3.2	Ejemplo para una ID diferente	8
4.1	Codigo de la Controladora Arduino Due	16
4.2	Configuración de la Controladora Arduino Due	18
4.3	Conexion a la Red	19
4.4	Función <i>LOOP</i> del código	19
4.5	Lectura en crudo del GPS	20
4.6	Formato de la String que se enviará	20
4.7	Funciones de recogida de datos	21
4.8	Función de comunicación con el servidor	21
4.9	Formato de la String GET	22
4.10	Controller que almacena la String en la BD	22
4.11	Configuracion de la BD en Laravel	23
4.12	Migracion	23
4.13	Instalación del Servicio Apache en Linux	24
4.14	Instalación del Servicio Apache en Linux	24
4.15	Asignación de directorio	24
4.16	Migración en Debian	24
5.1	Migración de la base de datos	26
5.2	Ejemplo para una ID diferente	28

CAPÍTULO 1

Introducción

En la actualidad cada día aumentan el consumo de productos, la necesidad de demanda, la producción en las fábricas, el aumento del tráfico,... todo esto ha generado en el mundo el gran problema de la contaminación.

Con este problema he decidido en crear una plataforma para poder medir este problema de una manera sencilla y barata para que cualquier usuario pueda tener acceso a esta información ideando como una especie de plataforma para la medición de contaminación en puntos específicos y este mandar la información vía protocolo HTTP a un servidor en el que cada usuario puede acceder cualquiera para compartir sus datos.

Este proyecto puede ser útil ya que la aplicación web realizada podrá servir para el uso de gestión de una *SMART City*, o simplemente para recoger información y realizar estudios de contaminación de ciertos lugares ya que la aplicación podría servir esos datos en ficheros para aplicarlos a otros programas y realizar análisis, todo muy útil en el sistema de minería de datos.

1.1 Objetivos

El objetivo de este proyecto es trabajar una parte de informática electrónica y comunicar esta parte con un servidor para poder trabajar esta información y poder servirla a un cliente a través de una aplicación web la cual el usuario podrá obtener y visualizar esa información.

Los objetivos son los siguientes:

- Desarrollo de la plataforma Hardware, en este caso he decidido utilizar la Plataforma Arduino.
- Configuración de la plataforma para que pueda enviar la información al servidor vía HTTP.
- Configurar el servidor para que ofrezca un servicio web y a la vez reciba la información del Arduino.
- Configurar la base de datos para que almacene la información recibida por la plataforma Hardware.
- Crear la aplicación web que sirva esta información detallada.
- Esta aplicación ademas ofrecerá estas funciones:
 - Podrá servir información en formato .txt o .csv para poder trabajar con ellos.

CAPÍTULO 2

Estudio del Arte

En este capitulo se hablará de porque se han elegido las tecnologías que se van a utilizar en el proyecto así como mostrar algunas de las opciones adicionales que se podrían haber utilizado.

2.1 ¿Porque Arduino?

Arduino es una plataforma Hardware Open Source lo que implica que todo el sistema esta disponible a los usuarios y esto ayudara al proceso de montaje ya que una gran parte de personas (la comunidad de Arduino) trabajan de manera desinteresada en el desarrollo de librerías y aplicaciones para esta plataforma.

Sin embargo Arduino dispone de numerosas versiones por lo que dentro de las mas famosas que hay he decidido analizar cual de los que hay me ofrece mas garantías a la hora de realizar el proyecto por lo que en principio me decantare por el consumo de las microcontroladora, puertos y conexiones disponibles y disponibilidad.

En primer lugar he realizado un estudio del consumo de las placas disponibles para poder realizar este proyecto,

Board	Consumo(W/h)	Consumo(mA/h)
Arduino UNO	0.23	46
Arduino DUE	0.375	75
Arduino MEGA	0.465	93
Arduino Nano	0.075	15
Raspberry Pi	1.77	353

Tabla 2.1: Consumo de los diferentes dispositivos

En este caso nos interesaría el elegir el dispositivo que menor consumo tenga pero también he tenido que mirar que funcionalidad pueden aportar cada placa, cabe destacar que la Raspberry Pi no es una microcontroladora, si no un ordenador de placa reducida, pero ofrece una funcionalidad que podría mejorar las prestaciones de este proyecto, hablando de seguridad y comunicación, pero sin embargo después de las pruebas de consumo comparando con el Arduino que mas consume, en este caso el MEGA, consume en mA un 379 % mas ya que necesita mantener un sistema operativo que aunque sigue siendo un consumo muy bajo para ser un ordenador, es muy elevado para este proyecto así que en este punto descartamos la Raspberry Pi.

Otro punto a tener en cuenta son las capacidades de conexión que tienen las microcontroladoras, ya que para poder conectarlo todo voy a necesitar 3 entradas analógicas, una entrada digital, 2 entradas para el rx y tx del puerto serie y compatibilidad con la Shield ethernet de Arduino.

El Arduino UNO es una muy buena opción ya que tiene todo esto y es de un consumo reducido aun así, la comunicación serial va por software ya que hay que implementar una librería especial para poder realizar un puerto serie para comunicarse con el GPS, ademas habría que diseñar un circuito por los problemas en los voltajes de las lecturas digitales ya que algunos componentes devuelven 3.3V y Arduino los detecta a 5V, que bien que no daría problemas pero podría dar lugar a valores anómalos en determinadas ocasiones.

El Arduino MEGA es la mejor opción disponible sin embargo ajustando con el material disponible, he de elegir el Arduino DUE ya que tiene un consumo mas reducido.

Estas son las características del Arduino DUE:

- **Microcontrolador:** AT91SAM3X8E
- **Velocidad de reloj:** 84 MHz
- **Tensión de trabajo:** 3.3V
- **Pines de entradas digitales:** 54
- **Pines de entradas analógicas:** 12
- **Memoria Flash:** 512 KB

He subrayado los valores que son importantes ya que en el estudio del montaje en el Arduino UNO tenia problemas con la memoria del programa ya que la memoria Flash quedaba casi completa utilizando la librería que implementaba un serial en cualquier pin digital, imposibilitando la capacidad de poder implementar mejoras en la aplicación.

2.2 Presupuesto

En esta parte del proyecto se va a realizar un estudio de lo que puede llegar a costar la creación, el montaje y el mantenimiento de la aplicación, todo ello al precio actual del año 2016.

Em primer lugar se expondrá el coste de que costaría aproximadamente cada nodo de la aplicación:

- **Arduino DUE:** 36€
- **Shield de Ethernet:** 26€
- **Neo6Mv2:** 13,50€
- **Sensor de Gas MQ-7:** 8€
- **Sensor de Temperatura y Humedad DHT11:** 3€
- **Sensor de Sonido:** 0,50€
- **Sensor de Luz:** 1€

Todo esto nos da un coste de **88 €** por nodo de sensores aproximadamente.

Se podrían utilizar materiales mas baratos con el fin de reducir los costes aun mas pero no seria aconsejable pues del proyecto interesa que este activo el mayor tiempo posible. También se pueden modificar los elementos del mismo, nombrados en la parte donde se realizaba el estudio del consumo de energía de cada controladora (**Tabla 2.1**) realizando las modificaciones necesarias para que este pueda funcionar.

También se ha incluido el coste de lo que costaría añadir el servidor de la aplicación, en este caso he puesto componentes genéricos:

- **Servidor:** 600€
- **Router:** 30€
- **Cableado:** 12€

Todo esto nos da un coste de **642 €** por servidor aproximadamente, pero, como se ha explicado antes, todo esto se puede reducir con material ya disponible, porque, "quien no tiene un ordenador hoy día?".

Y por ultimo el mantenimiento de lo que seria todo el montaje, ya es difícil hablar de un mantenimiento pues la aplicación pues aumentaría con cada nodo añadido, la ampliación del servidor en caso de haber demasiadas peticiones, el modo de alimentar los nodos que podría ser por microUSB, POE o baterías, diferentes Controladoras, todo esto nos daría un resultado muy variable, así que es muy complicado cuanto costaría mantener la aplicación.

CAPÍTULO 3

Descripción del proyecto

Para este proyecto se utilizará una microcontroladora Arduino DUE desatollada por **Arduino**, en esta se implementarán una serie de sensores y librerías que permitirán realizar una serie de funciones. Para enviar la información a la red, dicha controladora necesitará una Shield que le permita conectarse a la red también desatollada por **Arduino**.

Para recoger toda esta información se diseñará una aplicación que reciba las peticiones y podrá almacenarla en una base de datos, a su vez esta aplicación web también ofrecerá estos datos al usuario de una manera detallada para su estudio.

Finalmente se le ofrece al usuario generar unos reportes en los que este podrá descargarlos para, en un futuro, poder trabajar con esa información en otros programas o plataformas.

Esta es la explicación de las tecnologías y dispositivos que voy a utilizar:

3.1 Tecnologías utilizadas

3.1.1. Arduino

Arduino DUE es una placa Microcontroladora basada en el Atmel SAM3X8E ARM Cortex-M3 CPU. Es el primer Arduino basado en un Núcleo microcontrolador ARM de 32 bits ya que su versión anterior (Arduino UNO), trabaja directamente sobre un microcontrolador.



Figura 3.1: Logo Arduino

Para programar estas controladoras es necesario utilizar el IDE de Arduino, realizando una comunicación serie entre la máquina y la controladora podremos realizar un flash de la memoria con el fin de incorporar a la memoria interna del Arduino DUE el programa que hemos diseñado para realizar las funciones. Para programarlo se utiliza una versión simplificada de C++ la cual realiza una configuración inicial (*Setup*) y un bucle infinito (*Loop*), una vez arrancado el programa en la controladora este se repetirá indefinidamente realizando siempre la función programada.

El IDE de Arduino podemos descargarlo de aquí:

<https://www.arduino.cc/en/Main/Software>

3.1.2. GPS y Nmea Data

Para el proyecto vamos a utilizar el GPS Neo6mv2, este GPS es bastante útil ya que es barato y de bajo consumo, ideal para la propuesta del proyecto ya que lo que se interesa es el ahorro de energía.

Este GPS nos podrá enviar información útil del satélite como la posición, la altitud, la fecha y la hora todo utilizando un formato de cadena Hexadecimal llamada NMEA DATA

3.1.3. Nmea Data

"El National Marine Electronics Association (NMEA) ha desarrollado una especificación que define la interfaz entre varias piezas de equipos electrónicos. La norma permite la electrónica de la marina enviar información a los ordenadores ya otros equipos marinos."^[3]

Los receptores GPS están incluidos en esta especificación. Muchos de los programas de Posicionamiento están comprendidos bajo el formato NMEA. Estos datos incluyen PVT (Posición, Velocidad, Tiempo) generada por el receptor GPS. La idea del NMEA es enviar información llamada frase la cual es totalmente independiente de otras frases.

3.1.4. Formato del NMEA Data

El módulo Neo6Mv2 funciona enviando al serie cadenas de números hexadecimales en formato de NMEA data. El GPS envía información al serie cada intervalo de tiempo por lo que no sería necesario el puerto Tx del Arduino ya que solo se va a recibir información, no obstante, lo he configurado con el fin de poder configurar lo desde el Arduino.

Este es el resultado que envía el GPS por el Serial2 configurado en Arduino:

```

1 $GPRMC,,V,,,,,,,,,N*53
2 $GPVTG,,N*30
3 $GPGGA,,,0,00,99.99,,,,,*48
4 $GPGSA,A,1,,,,,,,,,,99.99,99.99,99.99*30
5 $GPGSV,1,1,01,,06,,18*77
6 $GPGLL,,,192128.00,V,N*4B

```

Listing 3.1: Cadena Hexadecima del NMEA sin Conexión

Si analizamos los datos, siguiendo la especificación de la NMEA data se observa que recibe información del GPS pero analizando en concreto las dos cadenas \$GPGGA se puede apreciar que no se recibe la información de la posición por lo que hay que posicionar el módulo GPS directamente en contacto con el satélite, una vez así este establecerá conexión y enviará esta cadena:

```

1 $GPRMC,193326.00,V,,,,,,310816,,,N*7C
2 $GPVTG,,N*30
3 $GPGGA,193326.00,,,,0,00,99.99,,,,,*6A
4 $GPGSA,A,1,,9
5 $GPGSV,3,1,09,01,17,140,,02,09,315,,03,41,072,34,06,47,310,19*77
6 $GPGSV,3,2,09,11,00,151,,17,32,233,,19,37,259,,22,23,085,34*3
7 ,1,09,01,17,140,,02,09,315,,03,41,072,34,06,47,310,19*77
8 $GPGSV,3,2,09,11,00,151,,17,32,233,,19,37,259,,22,23,085,35*79

```

Listing 3.2: Ejemplo para una ID diferente

Como se puede apreciar en el **Código 4.5** devuelve una serie de cadenas que son ilegibles sin el conocimiento del formato pero gracias a una librería llamada *TinyGPS* se podrá devolver el valor de la posición sin necesidad de analizar la cadena.

3.1.5. Laravel

Basado en *Symphony* Laravel es un *framework Opensource* que permite desarrollar aplicaciones y servicios web sirviéndose de una estructura Modelo-Vista-Controlador (MVC) ya creada en PHP 5.

Con este entorno desarrollare la aplicación web que nos servirá para almacenar los datos y usara algunas tecnologías para ello, ademas he escogido pues este framework de desarrollo permite mostrar la misma interfaz adaptada a Android mostrando, mas o menos, la misma interfaz.



Figura 3.2: Logo de Laravel 5

3.1.6. MySQL

MySQL es un SGBD (Sistema de Gestión de Bases de Datos) relacional bajo licencia dual GPL por Oracle está considerada como la base datos open source más popular para entornos de desarrollo de aplicaciones Web. Este SGBD servirá para almacenar toda la información y viene por defecto en la aplicación de Servidor WEB MAMP.

Por defecto solo viene el servidor por lo que habrá que descargar el MySQL Workbench para trabajar con la base de datos. En esta url:

<https://www.mysql.com/products/workbench/>

3.1.7. Bootstrap

Bootstrap es una librería CSS para el desarrollo de vistas para aplicaciones web. Utilizado para desarrollar la interfaz de usuario en páginas web, como los botones, formularios, cabeceras...

3.1.8. Chart.js

Chart.js es una librería JavaScript, utilizada para generar gráficos en la vista de las aplicaciones web, es capaz de generar gráficos dinámicos, útil para el proyecto pues se implementara una serie de gráficos que monitorizaran la información obtenida.

CAPÍTULO 4

Montaje del Hardware y el Servicio Web

En este capítulo se va a explicar como se ha montado todo este sistema, así como una explicación de sus diferentes piezas y la función que realiza cada una de ellas tanto en la parte hardware como en la parte software. También se realizará un estudio del coste de todos los materiales, así como el mantenimiento y consumo de estos.

Esta es la estructura que se seguirá para el montaje de la aplicación:

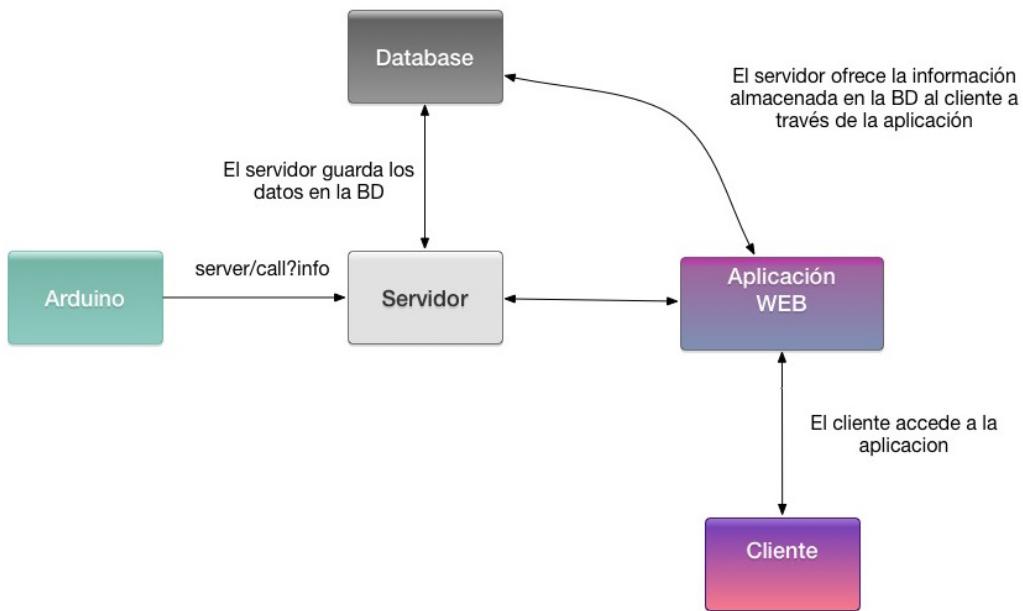


Figura 4.1: Diagrama de montaje del proyecto

En la **Figura 4.1** se puede observar el diagrama de montaje del proyecto en que función realizan las diferentes partes, desde el módulo hasta la información que podrá visualizar el cliente.

4.1 Material

Para este proyecto se ha utilizado el siguiente material:

Arduino Due: Es la Controladora que se encarga de recibir la información y procesarla para enviarla al servidor.

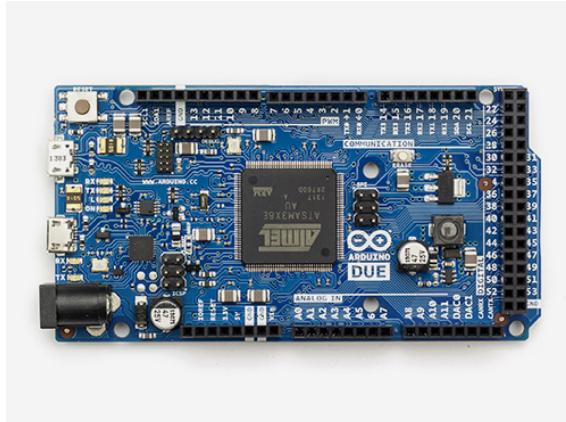


Figura 4.2: Microcontroladora Arduino DUE [2]

Ethernet shield: Es una Shield que permite a la micro controladora conectarse a la red.

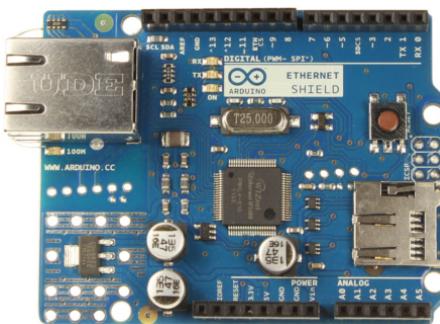


Figura 4.3: Ethernet shield [2]

Modulo GPS NEO6mv2: Es un módulo el cual contiene un GPS que una vez este conectado al satélite enviara información de la geolocalización.



Figura 4.4: Sensor de Temperatura y Humedad DHT11

Sensores: Un conjunto de sensores que permitirá la obtención de los valores ambientales del entorno donde este instalado el nodo.

Estos sensores son:

- **DHT11** Es el sensor de Humedad y temperatura de Adafruit recogerá estos datos climáticos del ambiente.



Figura 4.5: Sensor de Temperatura y Humedad DHT11

- **Sensor de gas CO MQ-7 y el chipset LM393** el que se utilizara para la medición de gases en el ambiente.

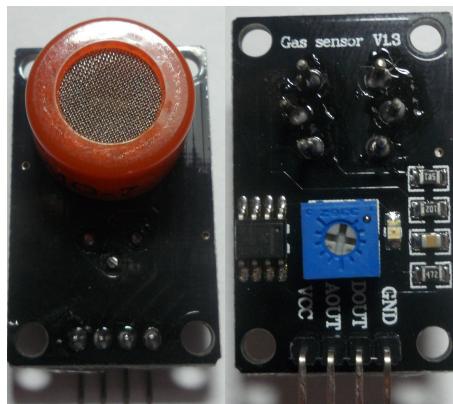


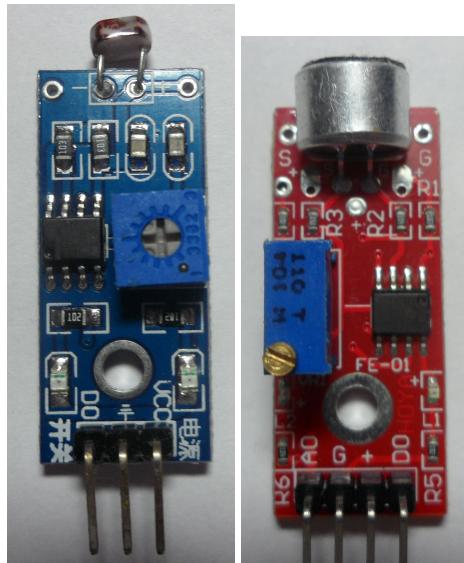
Figura 4.6: Sensor de Mesura de Gas CO MQ-7

- **Sensor luz y sonido** el cual también utiliza el chipset LM393 medirá la contaminación lumínica y acústica.

Servidor: El servidor que se va a utilizar para realizar las función de conexión entre el nodo y la la base de datos y entre el cliente y la conexión con el cliente a través de la aplicación web.

Las características de este servidor serán:

- **CPU:** Intel i5 2500K a 3.3 GHz de funcionamiento.
- **Memoria:** 8 GB 1600 MHz DDR3.
- **Almacenamiento:** 2 TB de Almacenamiento, el que permitirá tener una base de datos bastante amplia.



(a) Sensor de Luminosidad (b) Sensor de Ruido

Figura 4.7: Sensor de Luminosidad y Ruido

- S.O: OSX El Capitan 10.11.6.

Aplicaciones: Aplicaciones que me han permitido realizar el montaje de todo el sistema. Entre ellas están:

- **Mamp:** Es una aplicación para montar un servidor web Apache y Mysql bajo los puertos 8888 para el web y 8889 para el Mysql.
- **MySQL Workbench** aunque es cierto que Laravel 5 gestiona la base de datos través de PHP se ha utilizado esta aplicación para poder gestionar la información almacenada en la base de datos.
- **PhpStorm** Un IDE de desarrollo para programar con PHP así como gestionar proyectos con Git.

Cableado: Distintos tipos de cableado para conectarlo todo.

4.2 Montaje del Nodo Arduino DUE

El nodo para esta aplicación sera la microcontroladora con los módulos de sensores que enviaran información al servidor, para ello se realizara el montaje siguiendo el siguiente esquema:

En la **Figura 4.8** se puede apreciar el esquema de las conexiones que he seguido para el montaje del nodo para que el código de la aplicación para Arduino recoja los valores correctamente.

Como se puede apreciar en el esquema el montaje es sencillo ya que no necesita electrónica adicional como resistencias u otros componentes electrónicos. Para conectar esto cada uno de los sensores, siguiendo el esquema, se conectara VCC (**Rojo**) a 3.3V ya que es el funcionamiento de cada uno de los sensores exceptuando el GPS que ira conectado

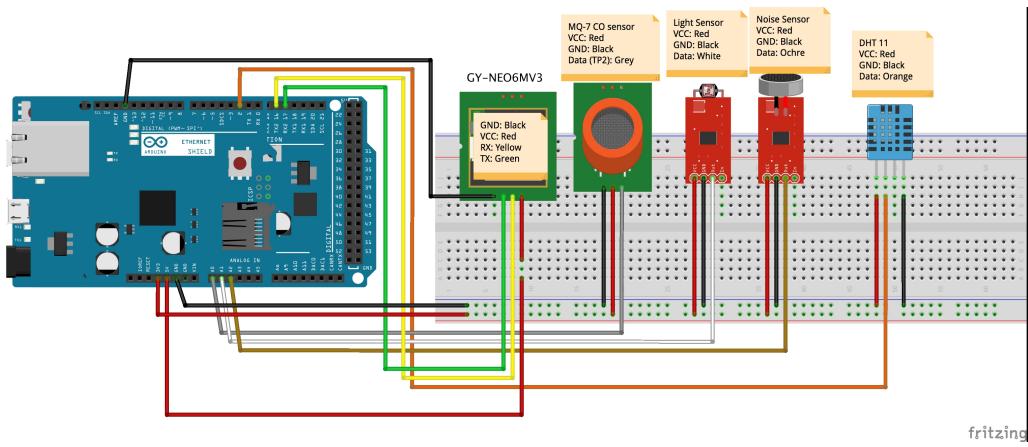


Figura 4.8: Esquema del montaje del Nodo

a 5V ya que ha demostrado un mejor funcionamiento a ese voltaje y GND (Negro) para el negativo de los módulos, todos los sensores exceptuando el sensor de temperatura y humedad DHT11 que va a la entrada digital y el GPS que va conectado al Serial 2, van conectados a entradas analógicas (de A0 a A2).

Tener en cuenta que si se realiza en un Arduino UNO al ser una versión mas antigua las lecturas digitales se leen en un rango de 5V por lo que hay que diseñar un circuito especial para poder leer el GPS y el sensor DHT11 ya que estos devuelven una señal digital de 3.3V, para solucionar esto se tendrían que poner resistencias en el circuito con el fin de diseñar un sistema que aumente la señal a los 5V que necesita el Arduino UNO, que en este caso quedaría tal que así:

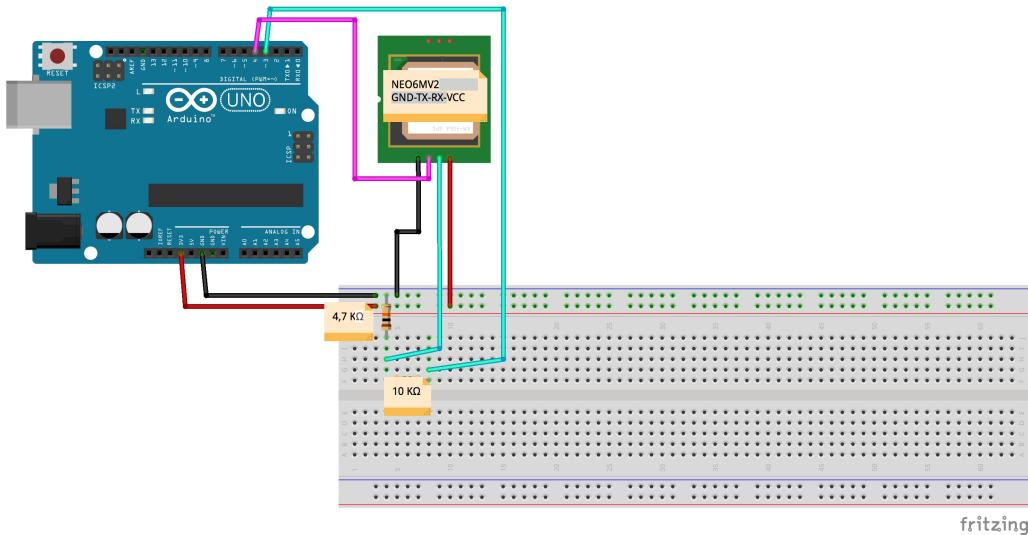


Figura 4.9: Esquema del circuito para el GPS en el Arduino UNO

El circuito de la Figura 4.9 es un ejemplo de instalación para un módulo en Arduino UNO cuya respuesta en digital sea de 3.3V y sirve para otros módulos.

También comentar que los RX y TX de la transmisión serie van conectados de manera invertida entre el GPS y el Arduino.

Este sería el montaje final de la electrónica de la controladora con todo el cableado y funcionando enviando la información al servidor.

Cada uno de estos sensores tendrá su función en el código de manera modular, así que es más sencillo entender el funcionamiento del mismo.

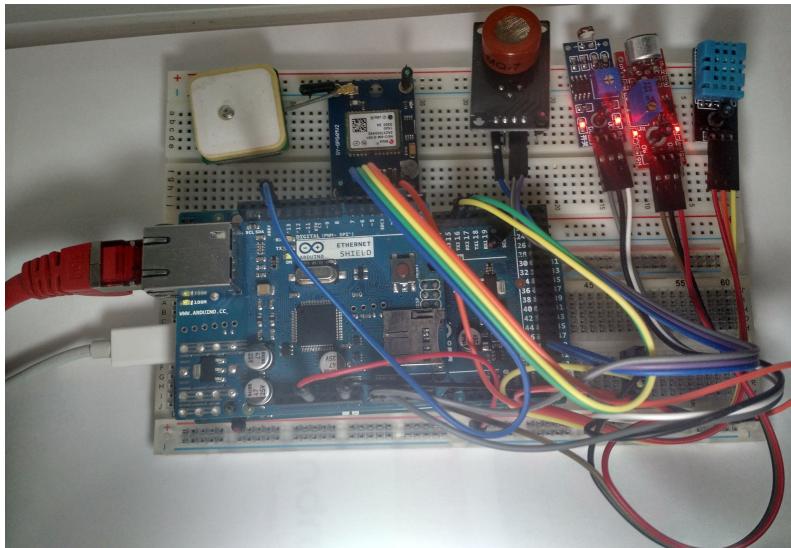


Figura 4.10: Montaje final del nodo de sensores

4.3 Programación de la Microcontroladora Arduino DUE

En este apartado se va a explicar cada una de las partes del código que utilizará el Arduino DUE, dicha parte realizará una función en la controladora por lo que el conjunto de todo el código realizará la función propuesta para este proyecto.

El código se puede descargar de la siguiente URL: https://www.dropbox.com/s/qzzff1y6fqgluh8/project_1.3.1.ino?dl=0

Con el IDE de programación de Arduino podemos abrirlo el cual mostrara el siguiente código:

```

1 /*
2 * FILE:      project-v1.3.1.ino
3 * AUTHOR:    David Rodriguez Martinez davidrm146@gmail.com
4 * VERSION:   1.3.1
5 */
6
7 //Necessary Libraries
8 #include <Ethernet.h>
9 #include <SPI.h>
10 #include <TinyGPS.h>
11 #include "DHT.h"
12
13 //Config of the device
14 byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
15 TinyGPS gps;
16 #define DHTPIN 2
17 #define DHTTYPE DHT11
18 DHT dht(DHTPIN, DHTTYPE);
19 IPAddress ip(192, 168, 1, 102);
20 EthernetClient client;
21 String data;
22
23 void setup()
24 {
25     Serial.println("EXECUTING setup");
26     Serial.begin(9600);
27     Serial2.begin(9600);
28     if (Ethernet.begin(mac) == 0) {

```

```
29 |     Serial.println("Failed to configure Ethernet using DHCP");
30 |     Ethernet.begin(mac, ip);
31 | }
32 | delay(1000);
33 |
34 |
35 void loop(){
36     Serial.println("EXECUTING LOOP");
37 //*****GPS CODE*****
38     bool newData = false;
39     unsigned long chars;
40     unsigned short sentences, failed;
41     for (unsigned long start = millis(); millis() - start < 1000;){
42         while (Serial2.available()){
43             char c = Serial2.read();
44             if (gps.encode(c))newData = true;
45         }
46     }
47     float flat, flon;
48     unsigned long age;
49     gps.f_get_position(&flat, &flon, &age);
50     Serial.print("LAT=");
51     Serial.print(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat, 6);
52     Serial.print(" LON=");
53     Serial.print(flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flon, 6);
54     Serial.print(" SAT=");
55     Serial.print(gps.satellites() == TinyGPS::GPS_INVALID_SATELLITES ? 0 : gps.
56                 satellites());
57     Serial.print(" PREC=");
58     Serial.println(gps.hdop() == TinyGPS::GPS_INVALID_HDOP ? 0 : gps.hdop());
59     data =
60     "board_id=" + getID() +
61     "&temp=" + getTemperature() +
62     "&hum=" + getHumidity() +
63     "&gas=" + getGas() +
64     "&noise=" + getNoise() +
65     "&luz=" + getLight() +
66     "&poslat=" + String(flat, 6) +
67     "&poslon=" + String(flon, 6);
68     displayData(data);
69     sendDataGet(data);
70     delay(60000);
71 }
72 /* 
73 * Debugger option
74 */
75 void displayData(String data) {
76     Serial.print("GET /call?");
77     Serial.println(data);
78     Serial.println("HTTP/1.1");
79     Serial.println("Host: 192.168.1.16");
80     Serial.println("Connection: close");
81     Serial.println();
82 }
83 void sendDataGet(String data) {
84     if (client.connect("192.168.1.16",8888)){
85         client.print("GET /call?");
86         client.println(data);
87         client.println("HTTP/1.1");
88         client.println("Host: 192.168.1.16");
89         client.println("Connection: close");
90         client.println();
91     }
92 }
```

```

92 if (client.connected()) {client.stop();}
93 }
94
95 String getTemperature() {
96     float t = dht.readTemperature();
97     float f = dht.readTemperature(true);
98     if (isnan(t) || isnan(f)) {
99         Serial.println("Failed to read Temperature from DHT. . .");
100    return "0";
101 }
102 return String(t, 2);
103 }
104
105 String getHumidity() {
106     float h = dht.readHumidity();
107     if (isnan(h)) {
108         Serial.println("Failed to read Humidity from DHT. . .");
109    return "err";
110 }
111 return String(h, 2);
112 }
113
114 String getGas() {return String(analogRead(A0));}
115 String getNoise() {return String(analogRead(A2));}
116 String getLight() {return String(analogRead(A1));}
117 String getID() {return "1";}

```

Listing 4.1: Código de la Controladora Arduino Due

Una vez puesto todo el código se ha dividido en diferentes secciones para poder explicarlo todo en varias partes con el fin de que el usuario pueda entenderlo:

Esta es la parte perteneciente a la **configuración del Arduino DUE**:

```

1 #include <Ethernet.h>
2 #include <SPI.h>
3 #include <TinyGPS.h>
4 #include "DHT.h"
5
6 //Config of the device
7 byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
8 TinyGPS gps;
9 #define DHTPIN 2
10 #define DHTTYPE DHT11
11 DHT dht(DHTPIN, DHTTYPE);
12 IPAddress ip(192, 168, 1, 102);
13 EthernetClient client;
14 String data;
15 void setup()
16 {
17     Serial.println("EXECUTING setup");
18     Serial.begin(9600);
19     Serial2.begin(9600);
20     if (Ethernet.begin(mac) == 0) {
21         Serial.println("Failed to configure Ethernet using DHCP");
22         Ethernet.begin(mac, ip);
23     }
24     delay(1000);
25 }

```

Listing 4.2: Configuración de la Controladora Arduino Due

Antes de entrar al código se implementaran las librerías para el funcionamiento de los módulos, en este caso lo que hacen es traducir la información para que pueda ser manipulada de manera mas fácil. las librerías son:

- `#include <Ethernet.h>`: Permite gestionar la conexión a la red.
- `#include <SPI.h>`: Permite al Arduino DUE comunicarse con la Shield de Ethernet.
- `#include <TinyGPS.h>`: Esta librería traduce el NMEA data en valores legibles sin necesidad de manipular la cadena de hexadecimales recibida.
- `#include 'DHT.h'`: Esta librería traduce la información recibida por el sensor DHT11, solo hace falta configurar que pin es el que recibirá la información y que tipo de sensor se va a utilizar ya que la librería configura automáticamente la comunicación con el sensor.

Esta es la configuración de Arduino encapsulada en la definición de SETUP en esta parte del código se definirá la comunicación vía serie para el módulo GPS conectado al serial 2 a 9600 baudios aunque aplicando una serie de cadenas hexadecimales en el arranque del programa escritas en el GPS vía serie. También se configurara la conexión serie que le pertenece al puerto USB para poder observar la información que envía cada vez que se repita el bucle del programa con el fin de poder depurarlo en caso de algún problema o fallo de los componentes.

```

1 if (Ethernet.begin(mac) == 0) {
2     Serial.println("Failed to configure Ethernet using DHCP");
3     Ethernet.begin(mac, ip);
4 }
```

Listing 4.3: Conexion a la Red

Esta parte es la conexión a la red, en este caso por DHCP que solo hace falta asignarle la MAC al dispositivo o si falla asignar manualmente una dirección IP que corresponderá a la de la red interna.

Una vez configurado todo el programa ya estará listo para leer la información que recojan los módulos de sensores asi que se dieñara el siguiente codigo para realizar dicha lectura en la definición del LOOP el cual se ejecutara indefinidamente:

```

1 void loop() {
2     Serial.println("EXECUTING LOOP");
3 //*****GPS CODE*****
4     bool newData = false;
5     unsigned long chars;
6     unsigned short sentences, failed;
7     for (unsigned long start = millis(); millis() - start < 1000;){
8         while (Serial2.available()){
9             char c = Serial2.read();
10            if (gps.encode(c))newData = true;
11        }
12    }
13    float flat, flon;
14    unsigned long age;
15    gps.f_get_position(&flat, &flon, &age);
16    Serial.print("LAT=");
17    Serial.print(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat, 6);
18    Serial.print(" LON=");
19    Serial.print(flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flon, 6);
```

```

20 Serial.print(" SAT=");
21 Serial.print(gps.satellites() == TinyGPS::GPS_INVALID_SATELLITES ? 0 : gps.
22     satellites());
23 Serial.print(" PREC=");
24 Serial.println(gps.hdop() == TinyGPS::GPS_INVALID_HDOP ? 0 : gps.hdop());
25 //The next variable keeps the information collected by the sensors
26 data =
27     "board_id=" + getID() +
28     "&temp=" + getTemperature() +
29     "&hum=" + getHumidity() +
30     "&gas=" + getGas() +
31     "&noise=" + getNoise() +
32     "&luz=" + getLight() +
33     "&poslat=" + String(flat, 6) +
34     "&poslon=" + String(flon, 6);
35
36 //Debug function, this string works
37 //data ="board_id=1&temp=23&hum=43&gas=53&noise=54&luz=55&poslat=63&poslon=73";
38 displayData(data);
39 //This function is for send information to the server
40 sendDataGet(data);
41 delay(60000);
42 }

```

Listing 4.4: Función *LOOP* del código

En el **código 4.4** se realizan 3 funciones, la lectura del GPS, el envío de la cadena que contiene las funciones que recogen de datos de los sensores y el delay que establecer un periodo de envío de información con el fin de evitar saturar la red.

```

1 for (unsigned long start = millis(); millis() - start < 1000;){
2     while (Serial2.available()){
3         char c = Serial2.read();
4         if (gps.encode(c)) newData = true;
5     }
6 }

```

Listing 4.5: Lectura en crudo del GPS

Esto recogerá la información del gps en formato de NMEA data pero gracias a la librería TinyGPS.h que nos traduce directamente sin tener que editar manualmente el NMEA data, la información obtenida del GPS leyendo el Serial2 del Arduino DUE devolviendo nos información del GPS, en este caso nos interesa **flat** y **flon** que son las coordenadas que recoge el GPS.

Los Serial.print están para mostrar la información obtenida ya que este no conecta al instante al satélite como ya se vera en el funcionamiento del vídeo mas adelante.

También se recoge la información para enviar al servidor en la variable **data** en la que se llaman las funciones que recogerán la información, es importante que el formato de la cadena sea como esta especificado en los comentarios, siguiendo el nombre de las variables ya que si no están bien nombrados la función del servidor que se encarga de recoger la información no funcionara y no se podrá realizar la inserción a la base de datos. Este es un ejemplo de como debería ser definida:

```

1 data="board_id={id} +
2     &temp={temperatura} +
3     &hum={humedad} +
4     &gas={medicion del gas} +
5     &noise={ruido ambiental} +

```

```

6  &luz={luminosidad} +
7  &poslat={latitud} +
8  &poslon={longitud}";

```

Listing 4.6: Formato de la String que se enviará

Y por último una delay o pausa que evitara que el Arduino esté enviando información constantemente de 1 minuto (60000 milisegundos)

Una vez explicado el bucle se explicaran las funciones que se ejecutan en la variable de datos ya que son las que recogen la información ambiental de la zona donde esta situado:

```

1 String getTemperature() {
2     float t = dht.readTemperature();
3     float f = dht.readTemperature(true);
4     if (isnan(t) || isnan(f)) {
5         Serial.println("Failed to read Temperature from DHT. . .");
6         return "0";
7     }
8     return String(t, 2);
9 }
10
11 String getHumidity() {
12     float h = dht.readHumidity();
13     if (isnan(h)) {
14         Serial.println("Failed to read Humidity from DHT. . .");
15         return "0";
16     }
17     return String(h, 2);
18 }
19
20 String getGas() {return String(analogRead(A0));}
21
22 String getLight() {return String(analogRead(A1));}
23
24 String getNoise() {return String(analogRead(A2));}
25
26 String getID() {return "1";}


```

Listing 4.7: Funciones de recogida de datos

En las funciones *getTemperature* y *getHumidity* se puede apreciar que gracias a la librería dht.h se realiza la conexión y lectura de la información del sensor sin tener que crear código adicional para controlar este sensor. Sobre las demás funciones son lecturas analógicas de las mismas que nos dan unos valores normales en su rango de 0 a 1024. Y la última sirve para establecer la id en la que esta registrada en la base de datos, este número debe coincidir con su controladora como ya se explicará mas adelante.

Una vez ya esta explicado el código se explicara la llamada a la web la cual realizara la inserción a la bd, este es su código:

```

1 void sendDataGet(String data) {
2     if (client.connect("192.168.1.16", 8888)) {
3         client.print("GET /call?");
4         client.println(data);
5         client.println("HTTP/1.1");
6         client.println("Host: 192.168.1.16");
7         client.println("Connection: close");
8     }
9     if (client.connected()) { client.stop(); }
10 }


```

Listing 4.8: Función de comunicación con el servidor

Esta función realiza la petición GET al servidor web el cual se configurara con la ip y el puerto en el que esté trabajando el servidor, puede ser pública o privada por lo que el Arduino puede trabajar en Internet aunque es mucho mas inseguro para la aplicación pues uno de sus puntos débiles es que el Arduino no tiene capacidad para la encriptación y tampoco tiene soporte para peticiones HTTPS por lo que una gran falla de este proyecto son los ataques *man in the middle*, la llamada corresponde a una petición a la web `call?información....en` la cual se obtendrá la información en forma de JSON y sera almacenada den la BD.

La llamada o dirección del servidor web quedaría algo tal que así:

```
1  direccionIP:8888 / call?board_id={id}&temp={temperatura}&hum={humedad}&gas={  
    medicion del gas}&noise={ruido ambiental}&luz={luminosidad}&poslat={  
    latitud}&poslon={longitud}
```

Listing 4.9: Formato de la String GET

Este es el código de la página web PHP que recogerá los datos en formato JSON y los guardara en la base de datos, todo ello realizado en Laravel 5.

```
1  public function setData(){  
2      $getData = Request::all();  
3      $data :: create($getData);  
4      if (! $this -> active) { $this -> setWorking(Request :: get('board_id'));}  
5      return $getData;  
6  }
```

Listing 4.10: Controller que almacena la String en la BD

La función `$getData = Request::all();` recoge toda la información que reciba en un GET o POST y lo introduce en la variable en formato JSON luego se ejecutaría el controlador de data la función `data::create($getData)`; de ahí la importancia que las variables introducidas en la variable data de Arduino estén nombradas como se ha especificado ya que en la BD se han de nombrar igual para que las entradas puedan ser introducidas correctamente.

Este es el resultado del JSON cuando se recibe la petición GET en la función:

```
{"board_id": "1", "temp": "26", "hum": "31", "gas": "93", "noise": "35", "luz": "35", "poslat": "38.960045", "poslon": "-0.181004"}
```

Figura 4.11: String JSON generada por la petición GET de Arduino

Una vez configurado esto ya se podría dejar la microcontroladora trabajando para la recolección de datos.

4.4 Instalación del Servicio WEB

4.4.1. Instalación en MacOSX

Para la instalación en este caso se requiere de un Servidor WEB con la funcionalidad de PHP5 y un SGBD (en este caso MySQL) añadido, en este proyecto se ha utilizado el servidor MAMP que es una aplicación de OSX que dispone de estas características el cual se va a utilizar para almacenar la información.

Para la instalación de la aplicación en el servidor se ha de descargar de este enlace el cual contiene un paquete de archivos que contienen la aplicación:

<https://www.dropbox.com/sh/nf8qp6jtqgrqb9d/AACmTXwt8dyTvCTKeraueG9oa?dl=0>

Una vez descomprimido se le aplican los permisos lectura y escritura al usuario y lectura a otros (`sudo chmod 744 tfg-laravel`) desde la terminal y se configura el directorio del servidor WEB:

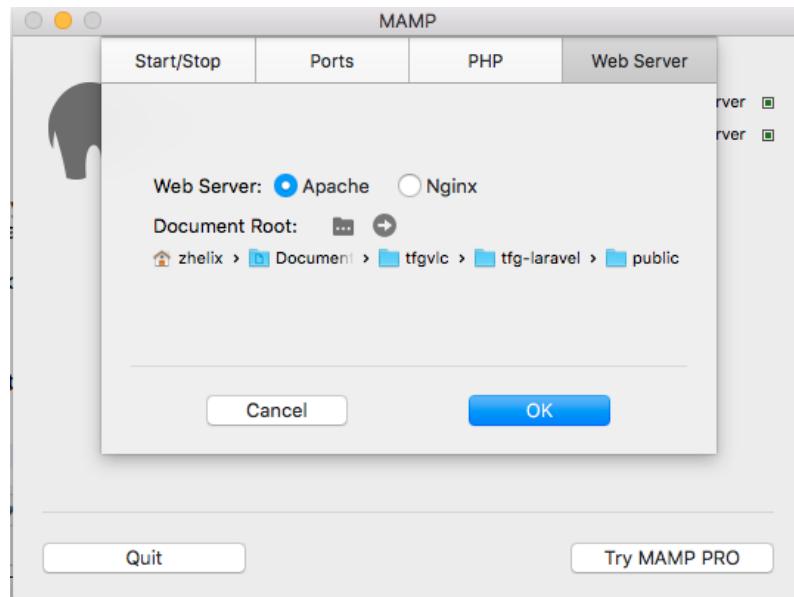


Figura 4.12: Configuración del Servidor MAMP

En la ventana de la **Figura 4.12** se muestra la configuración básica del directorio de la aplicación, que en este caso ha sido alojado en la carpeta de usuario, los demás parámetros se dejaran por defecto como es el puerto HTTP que lo configurara al 8888 y el servidor de Base de Datos que sera el 8889.

Una vez activado creamos la base de datos con la aplicación de Laravel llamada *artisan*. Para poder crear la base de datos dentro del directorio del NodeTracker hay que configurar el fichero `.env` el cual contiene la información de la base de datos para poder conectarse a ella:

```

1 DB_CONNECTION=mysql
2 DB_HOST=127.0.0.1
3 DB_PORT=8889
4 DB_DATABASE=[schema]
5 DB_USERNAME=[ user ]
6 DB_PASSWORD=[ pass ]
```

Listing 4.11: Configuracion de la BD en Laravel

En el **Código 4.11** se puede ver editar la configuración de la base de datos, en este caso se ha de poner la configuración del servidor WEB MAMP para que pueda conectar, por desgracia la versión del MAMP es *Trial* por lo que el usuario por defecto es *root* y la contraseña *root*.

Si toda la conexión esta bien realizada cuando ejecutemos la migración no dará ningún problema:

```
1 php artisan migrate tfgDatabase
```

Listing 4.12: Migracion

Una vez hecho esto ya estaría totalmente funcional el NodeTracker en nuestro sistema operativo MacOSX.

4.4.2. Instalación en Linux

Para poder instalarlo en un servidor Linux primero se ha de instalar el servicio WEB y el servicio de base de datos para poder alojar la aplicación WEB, para ellos este ejemplo se va a realizar en un servidor Debian el cual utiliza un gestor de paquetes llamado apt.

```
1 sudo apt-get install apache2 mysql-server php5 php-pear php5-mysql
```

Listing 4.13: Instalación del Servicio Apache en Linux

Una vez estos servicios estén funcionando se tiene que mover el directorio con los ficheros de la aplicación WEB al directorio del Apache2.

```
1 sudo mv ./tfg-laravel /var/www
```

Listing 4.14: Instalación del Servicio Apache en Linux

Pero para que se pueda acceder como *localhost:8888* a la aplicación hay que realizar unos cambios en el servidor Apache2 para que no se tengan que poner rutas adicionales, lo cual puede generar problemas de seguridad, para ello en el fichero */etc/apache2/sites-available/000-default.conf* se modifica el *DocumentRoot* por esto:

```
1 DocumentRoot /var/www/tfg-laravel
```

Listing 4.15: Asignación de directorio

Una vez esté hecha la configuración y responda *localhost:8888* hay que realizar la migración siempre teniendo en cuenta que la conexión a la base de datos este bien configurada en el fichero *.env* del *NodeTracker* la cual se puede ver en el **Código 4.11**.

```
1 sudo php artisan migrate tfgDatabase
```

Listing 4.16: Migración en Debian

Una vez hecho esto ya estaría totalmente funcional el *NodeTracker* en nuestro sistema operativo Linux.

CAPÍTULO 5

Explicación de la aplicación

En este capítulo se explicara como funciona la aplicación WEB diseñada para recoger y mostrar los datos de cada uno de los sensores.

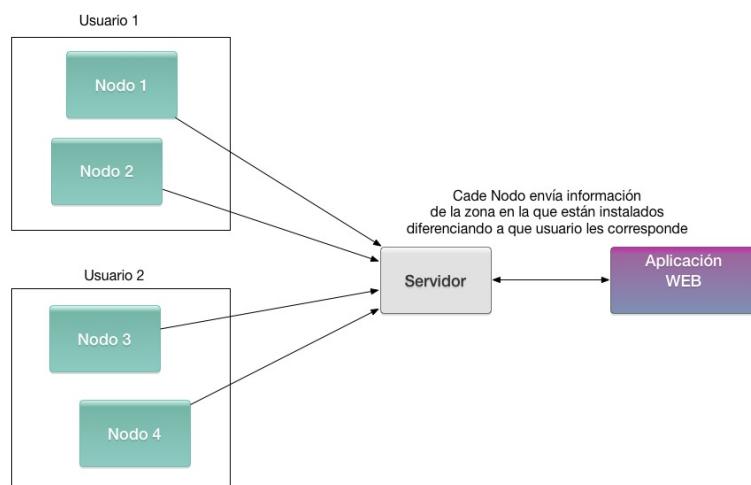


Figura 5.1: Esquema del funcionamiento de la aplicación

En la **Figura 5.1** se muestra el diagrama de la aplicación en el servidor en el que se dispone de una aplicación que recoge la información y la gestiona para que cada dispositivo o nodo se le pueda asignar a un usuario, este puede tener uno o muchos nodos y estos están recogiendo información independiente unos de otros y la almacena en la base de datos de manera organizada. En la aplicación pueden haber muchos usuarios y cada uno puede almacenar la información de cada uno de sus nodos.

5.1 Gestión de la información

La gestión de la información es sencilla, se trata de una base de datos simple en la que se almacenará la información de los nodos, la estructura de esta base de datos esta creada automáticamente por Laravel la cual hay que especificarle el siguiente diagrama E-R:

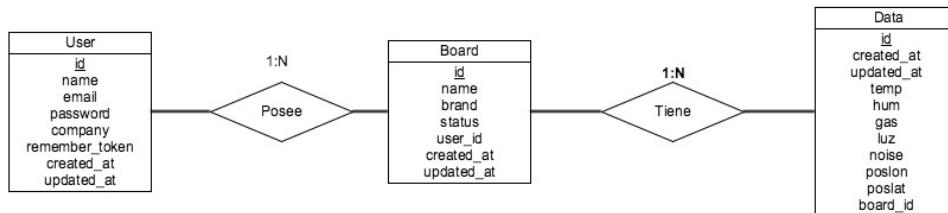


Figura 5.2: Diagrama Entidad-Relación de la base de datos

Esta base de datos ha sido creada automáticamente por *Artisan* de Laravel, la gestión automática crea la tabla de *usuarios* y algunos campos por defecto como *created_at* y *updated_at* sin embargo el resto de los campos hay que añadirlos manualmente así como especificar los tipos de relación entre los campos, este es el código del fichero alojado en *laravel-tfg/database/migrations/tfgDatabase.php* que especifica estos campos en la BD.

```

1 class TfgDatabase extends Migration{
2     /**
3      * Run the migrations.
4      * @return void
5      */
6     public function up(){
7         Schema::create('board', function (Blueprint $table) {
8             $table->increments('id');
9             $table->string('name');
10            $table->string('brand');
11            $table->string('status');
12            $table->integer('user_id')->unsigned();
13            $table->timestamps();
14            $table->foreign('user_id')->references('id')->on('users');
15        });
16        Schema::create('data', function (Blueprint $table) {
17            $table->increments('id');
18            $table->timestamps();
19            $table->string('temp');
20            $table->string('hum');
21            $table->string('gas');
22            $table->string('luz');
23            $table->string('noise');
24            $table->string('poslon');
25            $table->string('poslat');
26            $table->integer('board_id')->unsigned();
27            $table->foreign('board_id')->references('id')->on('board');
28        });
29    }
30    /**
31     * Reverse the migrations
  
```

```

32 * @return void
33 */
34 public function down(){
35     Schema::drop('data');
36     Schema::drop('board');
37 }
```

Listing 5.1: Migración de la base de datos

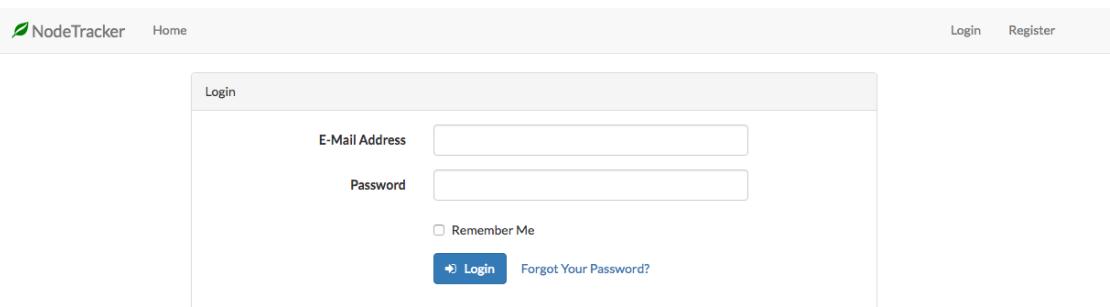
Si se quisieran añadir mas sensores a la aplicación se tiene que insertar una nueva entrada en la tabla data y luego aplicar la migración como se ha visto en el [Código 4.12](#).

5.2 Gestión del *NodeTracker*

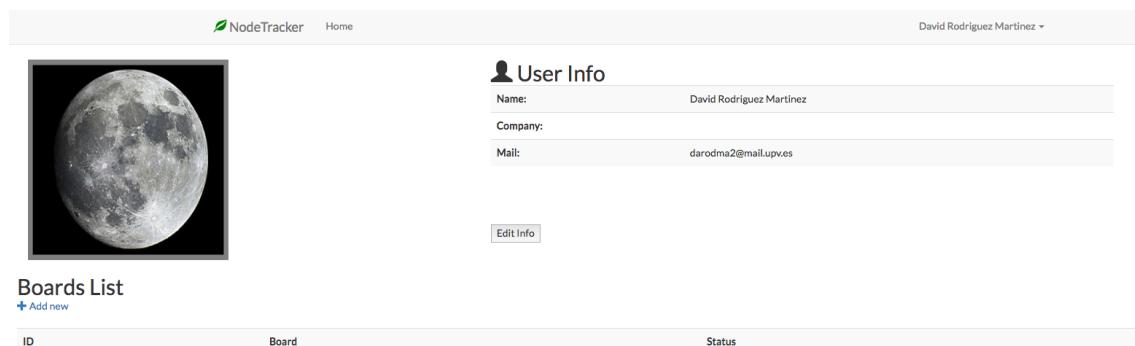
Una vez explicado como se almacena la información se procederá como puede acceder un usuario a la aplicación la cual se ha llamado *NodeTracker*, Para ello hay que o bien instalarla y configurar los nodos a la dirección del servidor o bien acceder al servidor ya instalado en la siguiente URL:

<http://84.123.151.120:8888/>

Una vez aparecerá el Login del Nodetracker, en el cual se podrá crear un usuario o entrar con uno ya disponible:

**Figura 5.3:** Login del NodeTracker

Una vez Logueado en la aplicación se muestra lo que se ha denominado como el Home, un panel para la gestión de los diferentes nodos la cual incorporara la funcionalidad de editar el usuario o añadir nuevos nodos.

**Figura 5.4:** Home del NodeTracker

Con un pequeño formulario se puede editar la información básica del usuario (**Figura 5.5**) y ademas como se había diseñado para esta aplicación se pueden añadir uno o mas nodos (**Figura 5.6**), esta funcionalidad devolverá una ID en la tabla la cual se tendrá que modificar en el código del Arduino con el fin de que guarde la información correcta correspondiente al nuevo nodo en la base de datos.

Figura 5.5: Formulario de edición de usuarios del *NodeTracker*.

Figura 5.6: Formulario para añadir un nuevo nodo al sistema.

Una vez creado para poder asignarlo al Arduino devolverá una ID para poder asignársela al nodo.

Nodes List		
+ Add new		
ID	Node	Status
1	Arduino DUE	Working
9	Raspberry Pi	Stopped

Figura 5.7: Lista de nodos disponibles

En la **Figura 5.7** se puede ver la ID el nombre asignado y el estado del nodo que por defecto viene parado, para poder enlazar los nodos con la aplicación hay que visualizar la ID i añadirla en la función del código que recoge la ID en el Arduino, pero básicamente hay que cambiar en la String a enviar al servidor, por ejemplo si se quisiera enviar información desde una Raspberry Pi la cual la ID es 9, habrá que configurar la String tal que así:

```
1 192.168.1.16:8888 / call?board_id= 8....
```

Listing 5.2: Ejemplo para una ID diferente

También esta el botón que redirigirá a la pagina de los reportes para poder visualizar la información obtenida que es la parte fuerte de la aplicación, es un panel donde se podrá visualizar la información o generar un reporte con la información recogida.

5.3 Visualización de la información

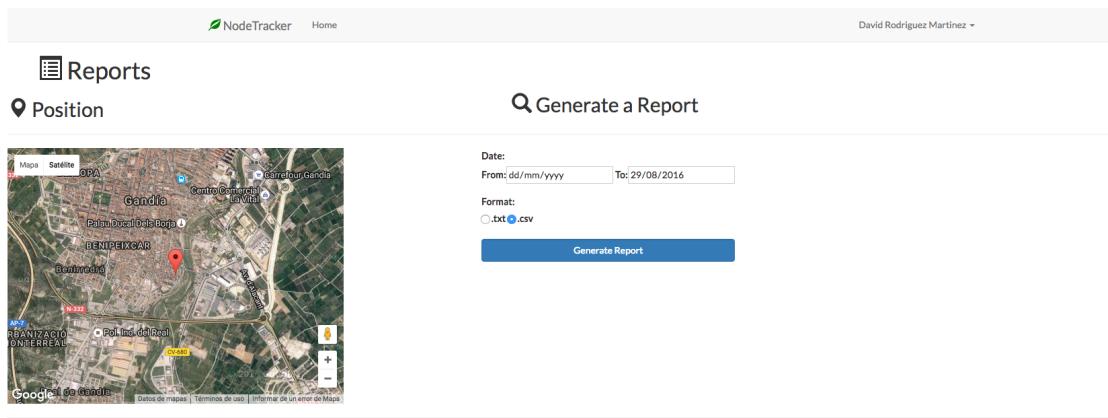


Figura 5.8: Pagina de monitorización de la aplicación

Esta es la página en la que se podrá visualizar la información, consta de 3 secciones, un mapa para mostrar la ubicación de la última entrada a la base de datos, otra sección que se encargara de generar los reportes para poder trabajar con ellos, y una gráfica que intentara mostrar un progreso de la información recogida a lo largo del día.

5.3.1. Posicionamiento

Para ello esta el Mapa en la sección de *Position* que gracias a la API de Google MAPS se ha podido implementar esta funcionalidad, en este caso el mapa es en vista satélite ya que al ser una aplicación medioambiental, se necesitará saber si el nodo está en un núcleo urbano o en territorio natural, y aproximadamente a qué altura se puede encontrar, costa o altas montañas. *Para el ejemplo de la Figura 5.10 se dispone de las coordenadas -0.180812 y 38.960541 que corresponden al término municipal de Gandía.*

5.3.2. Obtener un informe

Se ha implementado una función para generar informes, llamada *Reports* la cual generará un informe definido por un rango de fechas.

Date:
From: dd/mm/yyyy To: 30/08/2016

Format:
 .txt .csv

Generate Report

Figura 5.9: Generación de Informes

Esta función generara un informe en formato JSON de la información recogida cada minuto por el nodo unido a la aplicación, el resultado puede variar en el formato escogido en este caso .txt (texto plano) o .csv (Hojas de cálculo para LibreOffice) para poder trabajarlas en otras aplicaciones o realizar migraciones de estas, muy útil para la minería de datos que, utilizando programas en R o Python se puede trabajar este formato JSON.

5.3.3. Seguimiento de la información

Por último la función de monitorización, llamada *monitoring* se ha implementado una gráfica utilizando la API de Chart.js que permitirá realizar un seguimiento los valores recogidos en cada hora de ese mismo día realizando un promedio de los valores obtenidos en esa hora del día, ya que implementar cada minuto sería visualmente cargado y podría desestabilizar la gráfica si se presentan valores anómalos. Además presenta una gráfica diferente para cada valor a medir (temperatura, humedad, gases, luz y ruido) que implementado con AJAX realiza el cambio de gráfica sin necesidad de refrescar la página.

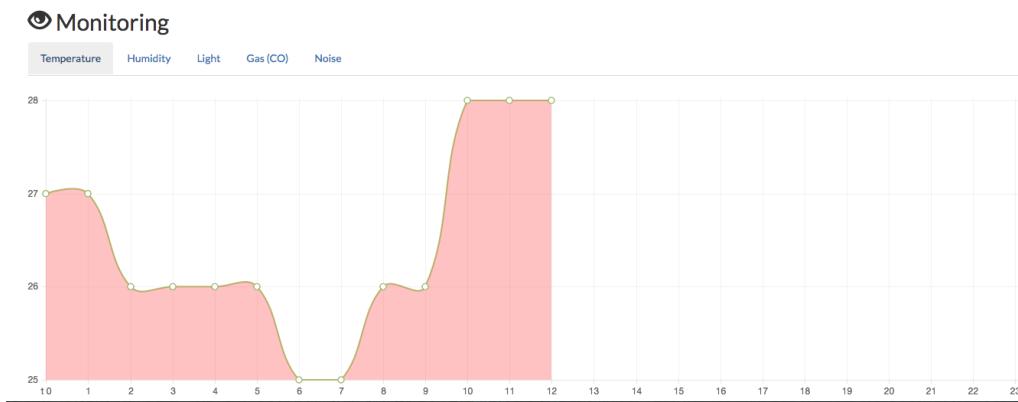


Figura 5.10: Gráfica perteneciente a la temperatura en el seguimiento

CAPÍTULO 6

Conclusiones y mejoras

6.1 Seguridad

Después de Utilizar Arduino he podido apreciar que es bastante bueno para el aprendizaje pero para una aplicación real deja mucho que desear ya que en un entorno educacional si se produce algún error se puede corregir en el momento y estos no serán provocados, ademas debido a la incapacidad nativa del SAM3X para encirptar la informacion, en este caso pretendía realizar peticiones HTTPS al servidor, el sistema es muy debil a ataques MitM (*Man-in-the-middle*) pues la informacion que envía no esta encirptada de ningún modo ni existe algún tipo de bloqueo a peticiones externas que no tengan nada que ver con la aplicación por lo que esto queda muy lejos de ser una aplicación final pues se puede romper por este hueco de seguridad.

Un ataque man-in-the-middle es un ataque en el que se adquiere la capacidad de leer, insertar y modificar a voluntad, los mensajes entre dos partes sin que ninguna de ellas conozca que el enlace entre ellos ha sido violado. El atacante debe ser capaz de observar e interceptar mensajes entre las dos víctimas.[10].

Para solucionar esto, he pensado en utilizar controladoras que permitan encrptación o conexión SSH lo cual requeriría un rendimiento mayor y un mayor consumo, o también, la aplicación solo pueda estar disponible para una red interna, lo que la limitaría a un área muy reducida ya que si esta conectada a Internet, el área de alcance es a nivel global pero insegura, por lo que resolviendo algunos problemas surgen otros nuevos.

6.2 Sensores

He tenido aprendizaje en el funcionamiento de la electrónica básica ha sido gracias a los diversos problemas entre versiones de Arduino y los cambios realizados en los circuitos para poder implementarlos.

Ademas de otros problemas, los sensores no dan siempre un valor fiable, en el proyecto los he tratado como valores anómalos, los cuales son valores que no corresponden con lo que deberían de dar ya sea por error de lectura o fallo de los componentes debido a su montaje en cableado.

Para poder solucionar esto habría que tener en cuenta 2 casos:

- **Circuito:** Todo esto podría mejorar si en vez de implementa el circuito en una *proto-board* se habría creado un circuito impreso y soldado los componentes directamente a la placa y estos pines conectarlos al Arduino con el fin de suprimir la *protoboard*,

cables y componentes de electrónica básica que permiten funcionar correctamente los sensores. Esto suprimiría los errores de fallos de conexión o cortocircuitos en la aplicación ademas de reducir notablemente el tamaño del nodo.

- **Sensores** Los sensores que he utilizado para el proyecto son de un coste reducido, por ejemplo, hay sensores que durante el montaje de este proyecto se han quemado, no son para una larga duración, debido a su coste y tampoco son muy exactos, para solucionar esto se deberian implementar sensores mas eficientes de un coste mas elevado, por ejemplo reemplazar el DHT11 de unos 2€ por su version mas cara el DHT33 por 17€.

Bibliografía

- [1] Jose Manuel Ruiz Gutierrez Arduino + Ethernet Shield *Funcionamiento de Arduino*, Versión 1, Enero, 2013. (Consultado el 27-12-2015)
- [2] Información sobre Arduino DUE, imágenes, conexiones y funcionamiento.
<https://www.arduino.cc/en/Main/ArduinoBoardDue>.
(Visitado el 10-01-2016)
- [3] Información, decodificación, programación y DataSheet del dispositivo de Localización NEO6mv2 utilizado en el proyecto [https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_\(GPS.G6-HW-09005\).pdf](https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_(GPS.G6-HW-09005).pdf).
<http://www.gpsinformation.org/dale/nmea.htm>.
<http://arduiniana.org/libraries/tinygps/>
(Visitado el 15-01-2016)
- [4] Información del funcionamiento y librerías del sensor DHT11 <http://playground.arduino.cc/Main/DHT11Lib>.
<https://cdn-learn.adafruit.com/downloads/pdf/dht.pdf>.
(Visitado el 20-03-2016)
- [5] Información del funcionamiento y DataSheet del sensor LM393 y MQ-7 Sensor CO consultado en <http://www.ti.com/lit/ds/symlink/lm2903-n.pdf>. <https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-7.pdf>.
- [6] Información sobre como desarrollar una aplicación WEB utilizando el Framework Laravel 5.2 consultado en <https://laravel.com/docs/5.2>.
- [7] Información y descarga de la aplicación para el servidor web obtenido en <https://www.mamp.info/en/>.
- [8] Documentación de la librería JavaScript Chart.js <http://www.chartjs.org/docs/>.
(Visitado el 12-04-2016)
- [9] Documentación de la API de Google Maps. <https://developers.google.com/maps/>. (Visitado el 15-05-2016)
- [10] Definición de *Man In the Middle* <https://www.icann.org/news/blog/que-es-un-ataque-de-intermediarios>. (Visitado el 20-08-2016)

