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 Rodríguez Martínez**

Tutor: **David Cuesta Frau**

Curso **2015-2016**

Resumen

El objetivo de este proyecto es implementar una red de monitorización medioambiental de bajo coste basada en nodos Arduino. Cada uno de estos nodos incorpora una serie de sensores: temperatura, humedad, monóxido de carbono, luminosidad y sonido, para medir los parámetros de interés, y además está geolocalizado mediante un módulo GPS.

Los datos obtenidos serán transmitidos mediante un módulo Ethernet. La información será gestionada usando MySQL y visualizada en un entorno web basado en el *framework* Laravel. Este entorno mostrará la geolocalización de los nodos en un marco de Google Maps, y los datos en tiempo real mediante gráficas interactivas implementadas utilizando *Chart.js*.

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

Resum

L'objectiu d'aquest projecte es implementar una xarxa de monitorització mediambiental de baix cost basat en nodes Arduino. Cadascú d'aquest node incorpora una sèrie de sensors: temperatura, humitat, monòxid de carbó, lluminositat i so, per a mesurar les dades d'interés, i també està geolocalitzat utilitzant un mòdul GPS.

Les dades obtingudes seran transmeses mitjançant un mòdul Ethernet. La informació està gestionada utilitzant MySQL y visualitzada en un entorn Web basat en el *framework* Laravel. Aquest entorn mostrarà la geolocalització dels nodes en un marc de Google Maps, i les dades en temps real mitjançant gràfiques interactives implementades en *Chart.js*.

Paraules clau: Ciutat Inteligent, Arduino, Hardware, Geolocalització, Sensors, Harware lliure, Electrònica, Servicis Web

Abstract

The purpose of this project is implement a low cost environmental network based in Arduino nodes. Each node adds some sensors: temperature, humidity, carbon monoxide, light and sound, for measure the relevant data, and also is geographic located by a GPS module.

The obtained data will be transmited using an Ethernet module. The information is managed using MySQL and visualised in a webpage based in a framework Laravel. This environment will show the position of nodes in the Google Maps API, and the data in real time using interactive graphics implemented in Chart.js.

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

Índice general

Índice de figuras

Listings

CAPÍTULO 1
Introducción

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

Para aportar un mayor control sobre este problema, se ha decidido en crear una plataforma hardware para poder medir las distintas variables que afectan a este tipo de problemática ambiental de una manera sencilla y barata, de esta forma para que cualquier usuario podrá tener acceso a una información reciente y poder realizar un estudio del entorno. Para ello se configurará un dispositivo hardware, una controladora (llamada nodo) con una serie de sensores para la medición de la contaminación en diferentes puntos geolocalizados. Este sistema, podrá enviar la información a través del protocolo HTTP a un servidor en el que un software recogerá los datos y los almacenará en una base de datos para que con este tipo de aplicación los usuarios puedan tener acceso a esa misma información y realizar un estudio detallado de la misma.

Este proyecto podrá ser útil ya que la aplicación Web 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 podrá servir esos datos en ficheros para aplicarlos a otros programas y realizar análisis.

1.1 Objetivos

El objetivo de este proyecto será implementar la parte hardware compuesta por una microcontroladora en tiempo real para posteriormente comunicarla con un servidor y así para poder trabajar la información recibida y poder servirla a un cliente a través de una aplicación Web, desde la cual el usuario podrá obtener la información en una serie de gráficas o descargarla en ficheros para su posterior estudio.

Los objetivos serán los siguientes:

- Desarrollo de la plataforma Hardware, en este caso se ha 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, ya sean gráficas o ficheros en formato `.txt` o `.csv`.

CAPÍTULO 2
Estudio del estado del arte

En este capitulo se va a realizar un estudio del estado del arte de porqué 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 implementado.

2.1 ¿Por qué Arduino?

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

Sin embargo Arduino dispone de numerosos modelos, por lo que se ha decidido analizar cual de los modelos que se dispone ofrece más garantías para realizar el proyecto. Lo más importante a tener en cuenta es el análisis de consumo de las microcontroladoras, conexiones disponibles y disponibilidad.

En primer lugar se ha 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 interesará el elegir el dispositivo que menor consumo tenga, pero también se ha tenido que mirar la funcionalidad que puede aportar cada controladora. Por ejemplo la opción ideal sera una Arduino Nano cuyo consumo es muy reducido pero por su estructura es incompatible con la shield ethernet.

Cabe destacar que la Raspberry Pi no es una microcontroladora, si no un SBC (*single-board computer*), pero puede aportar una funcionalidad que podría mejorar las prestaciones de este proyecto, en términos de seguridad y comunicación, sin embargo después de las pruebas de consumo, comparando con el Arduino que más consumo energético tiene (Arduino MEGA) consume en mA un 379 % más, ya que necesita mantener un sistema

operativo que aunque sigue siendo un consumo muy bajo para ser un ordenador, es muy elevado para lo que se pretende montar en este proyecto, así que se descartará la Raspberry Pi.

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

El Arduino UNO es una muy buena opción ya que tiene todo lo mencionado anteriormente y es de un consumo reducido aun así, la comunicación serie va por software, y para poder programarlo es necesario implementar una librería especial para poder crear un puerto serie virtual para comunicarse con el GPS, además 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 puede no dar problemas pero, podría dar lugar a valores anómalos en determinadas ocasiones.

El Arduino MEGA es la mejor opción a elegir, sin embargo ya que no se dispone de dicha controladora, se va a elegir el Arduino DUE que, además tiene un consumo más 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

Se han subrayado los valores que son importantes a la hora de seleccionar una controladora, se ha decidido incluir la capacidad de la memoria flash ya que en el estudio del montaje en el Arduino UNO se han tenido problemas con la memoria del programa ya que esta quedaba casi completa utilizando la librerías que implementaban un puerto serie virtual en cualquier pin digital y la gestión del módulo GPS, 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 instalación y el mantenimiento de la aplicación, todo ello al precio actual del año 2016.

En primer lugar se expondrá el coste de 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€

Estos componentes dan un coste de **88 €** por nodo de sensores aproximadamente.

Se podrían utilizar materiales más baratos con el fin de reducir los costes, pero no seria aconsejable ya que, el sistema debe estar activo el mayor tiempo posible y los componentes de coste bajo tienden a fallar con frecuencia. También se pueden modificar los elementos del mismo, nombrados en la parte donde se realiza el estudio del consumo de energía de cada controladora (**Tabla 2.1**) realizando las modificaciones que se crean necesarias para que este sistema pueda funcionar.

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

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

Todo esto da un coste de **642 €** por servidor aproximadamente, pero, como se ha explicado antes, todo esto se puede reducir con material ya disponible, ya que actualmente cualquier usuario puede disponer de un ordenador capaz de albergar el servidor de la aplicación.

Y por último el mantenimiento de lo que seria todo el montaje, ya es complicado hablar de un mantenimiento pues la aplicación 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 daría un resultado muy variable, así que es muy complicado calcular cuanto costaría mantener la aplicación.

2.3 Gestión del proyecto

Por la necesidad de los cambios constantes en el código...

2.3.1. *Bitbucket y SourceTree*

Para realizar este proyecto sus ficheros va a ser gestionados por una aplicación Web llamada *Bitbucket*:

Bitbucket es un servicio de alojamiento basado en Web, para los proyectos que utilizan el sistema de control de revisiones Mercurial y Git. [12]. *Git es un software de control de versiones diseñado, pensado para la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.*[14]

En esta aplicación Web se almacenarán todos los cambios que se realicen en este proyecto ya que ofrece una gestión bastante sencilla de los proyectos y las versiones de cada una, además cuenta con una aplicación llamada *sourceTree* que permite conectar con un usuario de *Bitbucket* y permitirá realizar las subidas de ficheros desde el mismo sistema

operativo sin tener que acceder desde el navegador y a parte, gestionar de una manera sencilla todas las versiones del proyecto.

Esta aplicación se puede descargar de la siguiente URL: https://www.sourcetreeapp.com/

En la **Figura 2.1** se puede observar cada uno de los cambios que se han aplicado para cada modificación del proyecto ya que, sin una gestión como la que ofrece *Bitbucket* y *SourceTree*, se subirían los ficheros manualmente a un servidor como Google Drive o Dropbox sin un control de versiones, y surgirían problemas si por algún motivo se quisiera volver a una versión anterior.

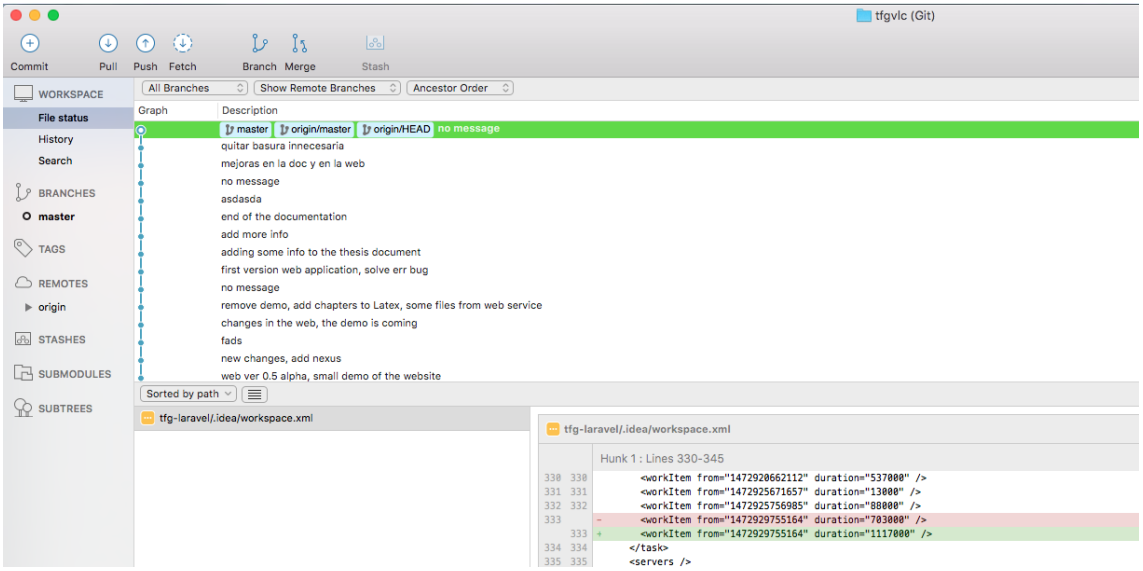


Figura 2.1: Gestion de proyectos con *SourceTree*

CAPÍTULO 3
Descripción del proyecto

Para cumplir con el objetivo principal de este proyecto se va a utilizar un sistema empotrado basado en una plataforma Arduino DUE desarrollada por la compañía *Arduino*, sobre el cual se conectarán una serie de sensores y se implementarán unas librerías. Sobre estas un software permitirá al sistema en el que se esta trabajando, realizar una serie de funciones como geolocalización, medir la temperatura, la humedad ambiental, etc. y posteriormente enviarla a través de la red al servidor. Para enviar la información a la red, el nodo necesitará una *Shield* complementaria que le permitirá conectarse a la red y enviar la información obtenida.

Una vez recogida toda la información obtenida se diseñará una aplicación que escuche las peticiones enviadas por la controladora y las almacene en una base de datos. A su vez esta aplicación Web ofrecerá los datos recogidos al usuario de una manera detallada para su estudio.

Finalmente se le ofrecerá al usuario la posibilidad de generar unos informes que podrá descargarlos para, en un futuro, poder trabajar con esta información en otras aplicaciones i/o plataformas.

De aquí en adelante, se procederá a la explicación de las tecnologías y dispositivos que se utilizarán para diseñar la aplicación.

3.1 Tecnologías utilizadas

3.1.1. Arduino

Arduino DUE es un sistema empotrado basado en el procesador Atmel SAM3X8E ARM Cortex-M3 CPU.

Al programar estos sistemas embebidos es necesario utilizar el IDE de Arduino, para que posibilite una comunicación serie entre la máquina y la controladora. Esto permite que el IDE de desarrollo realice un flash de la memoria interna del Arduino DUE con el fin de implementar en la controladora el código que se ha diseñado. Una vez el programa este cargado en la controladora, se podrá proceder a enviar la información.

Con el fin de programarlo, el IDE de Arduino dispone de una versión simplificada de C++. La estructura del programa esta definida por la configuración inicial (*Setup*) y un bucle infinito (*Loop*). Una vez arrancado el programa en la controladora, el bucle se repetirá indefinidamente realizando siempre la función programada. **Figura 3.1**.

El IDE de Arduino se puede descargar de la siguiente URL: https://www.arduino.cc/en/Main/Software

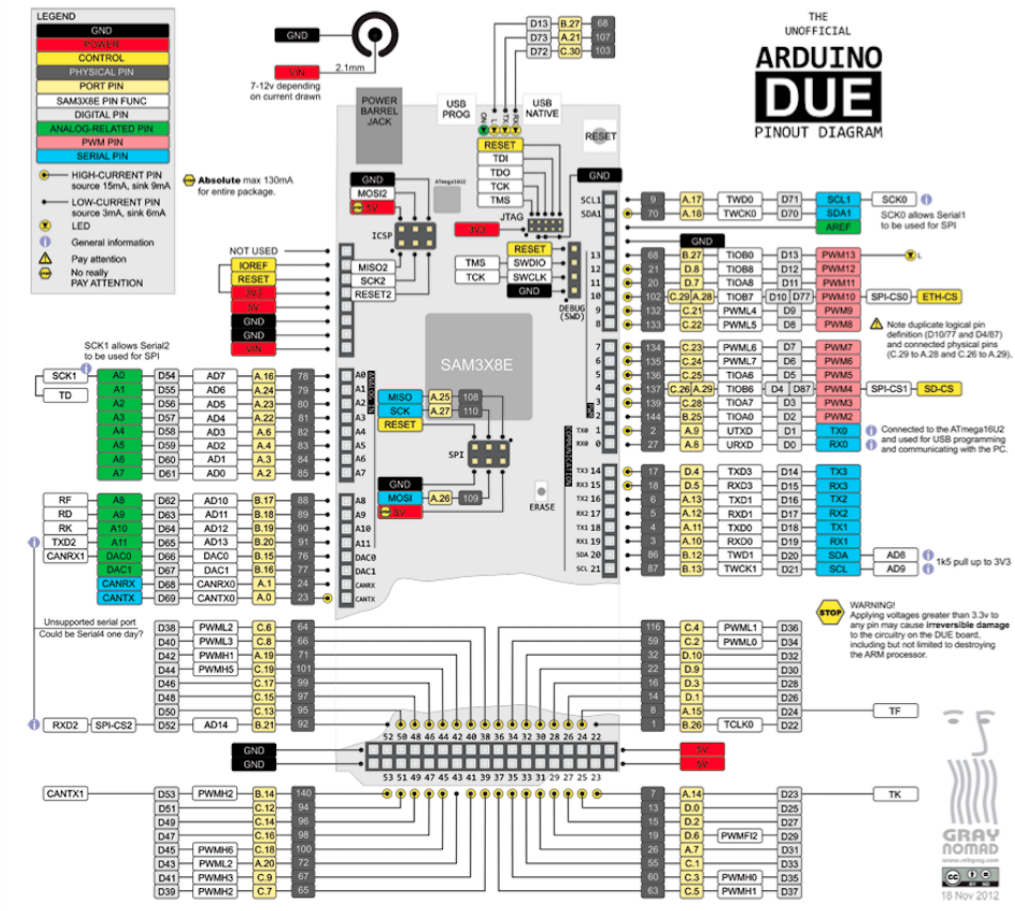


Figura 3.1: Esquema que representa la estructura de conexiones de Arduino

3.1.2. GPS y NMEA Data

Para este proyecto se va a emplear el GPS Neo6mv2. Es un módulo de coste reducido y de bajo consumo, ideal para la propuesta del proyecto pues se busca el ahorro de energía y costes bajos. El módulo GPS enviará información útil obtenida del satélite como la posición, altitud, la fecha y hora, etc. utilizando un formato de cadenas denominado *NMEA Data*. Sin embargo, para evitar tener que analizar las cadenas recibidas en la programación del nodo, se implementará una librería que facilitará la conversión de estas cadenas en un valor más comprensible.

"*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. El receptor GPS Neo6mv2 está incluido en esta especificación. NMEA ofrece datos que incluye 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]

Formato del NMEA Data

El módulo GPS Neo6Mv2 funciona enviando información relativa de la posición a través del puerto serie y recogiendo una serie de frases de números hexadecimales siguiendo el formato de NMEA Data. El GPS envía esta información en intervalos de tiempo de un segundo, por lo que no es necesario configurar el conector TX del Arduino para solicitar esta información, ya que sólo se van a recibir datos de la posición, sin la necesidad de pre-

guntar al módulo GPS por estos. Si se quisiera añadir mejoras como el cambio en la tasa de transferencia, configurar al módulo un arranque o simplemente actualizar el GPS, seria posible configurar el conector TX para enviar al módulo una cadena de hexadecimales específica para cada cambio. Como se especifica en el siguiente código:

```
uint8_t gps_megacfg[98] = { 0xB5, 0x62, 0x06, 0x01, 0x08, 0x00, 0x01, 0x02, 0
    x00, 0x01, 0x00, 0x00, 0x00, 0xB5, 0x62, 0x06, 0x01, 0x08, 0x00, 0xF0
    , 0x00, 0x01, 0x00, 0x01, 0x01, 0x01, 0xB5, 0x62, 0x06, 0x01, 0x08, 0
    x00, 0xF0, 0x01, 0x01, 0x00, 0x01, 0x01, 0x01, 0xB5, 0x62, 0x06, 0x01
    , 0x08, 0x00, 0xF0, 0x02, 0x01, 0x00, 0x01, 0x01, 0x01, 0xB5, 0x62, 0
    x06, 0x01, 0x08, 0x00, 0xF0, 0x03, 0x01, 0x00, 0x01, 0x01, 0x01, 0xB5
    , 0x62, 0x06, 0x01, 0x08, 0x00, 0xF0, 0x04, 0x01, 0x00, 0x01, 0x01, 0
    x01, 0xB5, 0x62, 0x06, 0x01, 0x08, 0x00, 0xF0, 0x05, 0x01, 0x00, 0x01, 0x01
    , 0x01, 0x01};
Serial2.write(gps_megacfg);
```

Listing 3.1: Configuración manual del módulo Neo6Mv2

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

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

Listing 3.2: Cadenas NMEA sin Conexión

Al observarse los datos recogidos, siguiendo la especificación de la NMEA data, se puede entender que se recibe información del GPS pero, si se analiza en concreto las dos frases designadas con la entrada $GPGGA, se aprecia que no se recibe la información de la posición, por lo que hay que posicionar el módulo GPS directamente en contacto con el satélite, para que el sistema establezca conexión y envíe la siguiente cadena:

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

Listing 3.3: Cadenas NMEA con Conexión

Como se puede apreciar en el **Código 3.3**, el módulo GPS devuelve una serie de frases que son ilegibles sin el conocimiento del formato NMEA, pero que utilizando una librería denominada *TinyGPS.h* se obtendrá el valor concreto de la posición sin necesidad de analizar o manipular las cadenas recibidas por el módulo GPS.

3.1.3. Laravel

Basado en *Symphony* Laravel es un *framework Open source* que permite desarrollar aplicaciones y servicios Web sirviéndose de una estructura Modelo-Vista-Controlador (MVC) ya creada en PHP5.

Con este *framework* se va a desarrollar la aplicación Web que servirá para almacenar y mostrar los datos recogidos. Para diseñar ese sistema se utilizarán algunas tecnologías

adicionales que se explicarán posteriormente. Además se va a seleccionar este entorno de desarrollo porque permite mostrar la misma interfaz adaptada a dispositivos móviles como Android y IOS mostrando al usuario la misma interfaz pero adaptada al formato del dispositivo.

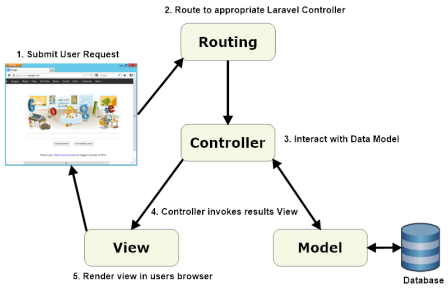


Figura 3.2: Estructura MVC de Laravel 5

El patrón que define a Laravel es el MVC. En este patrón habrá tres ficheros clave: uno que gestione el modelo y pueda manipular la base de datos; otro, el controlador, que gestione la capa de negocio de la aplicación y por último, la vista que interactuará con el cliente. Este sistema permitirá separar la lógica de negocio y los datos de la interacción con el usuario, como se muestra en la **Figura 3.2**.

Un ejemplo de 'hola mundo' en el *framework* Laravel5 seria tal que así:

```
1 //En el fichero app/http/routes.php se introduce la siguiente ruta:
2 Route::get('test', function () {
3   return 'Hola mundo';
4 });
```

Listing 3.4: Ejemplo de 'Hola mundo' en Laravel5

3.1.4. MySQL

MySQL es un SGBD (Sistema de Gestor de Bases de Datos) relacional bajo licencia dual GPL por Oracle. Es el sistema de base datos *Open Source* más utilizado para entornos de desarrollo de aplicaciones Web. Este SGBD servirá para almacenar toda la información recogida por el nodo.

El servicio de MySQL viene por defecto en la aplicación de escritorio MAMP. En esta sólo viene el SGBD por lo que para poder gestionar el servidor habrá que descargar el **MySQL Workbench** que es una aplicación de escritorio para visualizar las tablas y poder trabajar con la base de datos. Se puede obtener de la página oficial: https://www.mysql.com/products/workbench/

3.1.5. Maps, Bootstrap y Chart.js

Bootstrap es una librería CSS para el desarrollo de vistas en aplicaciones Web. Se Utiliza para desarrollar la interfaz de usuario en páginas Web, como los botones, formularios, cabeceras, etc. Esta librería permitirá que la vista de la aplicación sea mucho más elegante que con la interfaz presentada por HTML5 básico.

Para poder utilizar la librería Bootstrap solo es necesario añadir la hoja de estilos CSS en la cabecera *head* y especificar en el atributo *class* a las etiquetas HTML5 la ID asignada para cada elemento donde se desee aplicar el estilo.

Por ejemplo, para la definición de un icono, que se utiliza en la aplicación, se tiene que añadir tal que así:

```
<font color="green"><span class="glyphicon glyphicon-leaf" aria-hidden="true">
    </span></font>
```

Listing 3.5: Ejemplo de implementación en Bootstrap

En la etiqueta mostrada en el código anterior se puede observar en el atributo *class* la llamada al paquete *glyphicon* y en él, seleccionar el atributo a mostrar *glyphicon-leaf* que corresponde con el icono de una hoja.

Chart.js es una librería JavaScript, utilizada para generar gráficos en el la vista de las aplicaciones Web, es capaz de generar gráficos dinámicos. Útil para el proyecto pues se implementarán una serie de gráficos que monitorizarán la información obtenida. Para poder implementar un gráfica es necesario crear un çanvas", que es una etiqueta HTML que creará un marco el cual se podrá programar con javascript:

```
<div id="ejemplo">
  <canvas id="canvasejemplo" width="400" height="400"></canvas>
</div>
<script>
    new Chart(document.getElementById('canvasejemplo').getContext('2d') , {
        type: "line",
        data: {
              labels: ["Ej1", "Ej2", "Ej3", "Ej4", "Ej5", "Ej6", "Ej7"],
              datasets: [{ data: [65, 59, 80, 81, 56, 55, 40],}]
        }
    });
  </script>
```

Listing 3.6: Ejemplo de implementación de una gráfica con chart.js

En la vista se obtendrá una gráfica como esta:

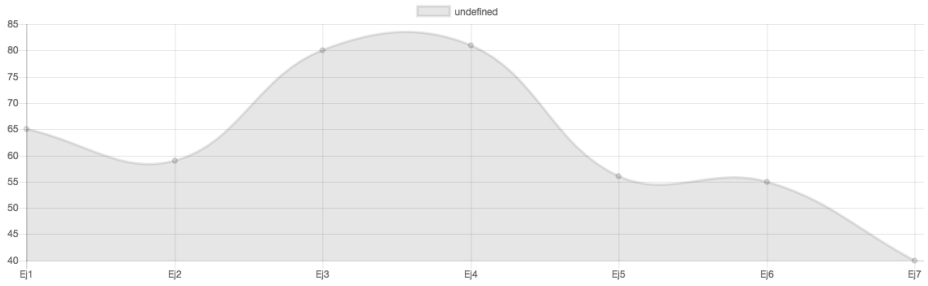


Figura 3.3: Ejemplo de una gráfica con *Chart.js*

Debido a que Laravel5 emplea una capa *middleware* se han presentado problemas a la hora de configurar los atributos de las gráficas.

API de Google Maps: Es una interfaz de programación de aplicaciones desarrollada en javascript, especializada en la realización de mapas y ofrecer al usuario la opción a implementar posiciones, formatos de mapas y otras opciones en el desarrollo de aplicaciones de una forma elegante.

Para una implantación sencilla de la API de Google Maps, se puede emplear el siguiente código:

```
<div id="googleMap" style="width:500px;height:380px;"></div>
<script src="https://maps.googleapis.com/maps/api/js?key=[Clave de
    desarrollador de Google]&callback=initMap" async defer>
var marker;
function initMap() {
  var mapOtions = {
    center: new google.maps.LatLng(38.985852,-0.196266),
    zoom: 14,
    mapTypeId: google.maps.MapTypeId.HYBRID
  };
  marker = new google.maps.Marker({
    position: new google.maps.LatLng(38.985852,-0.196266);,
    map: new google.maps.Map(document.getElementById("googleMap"), mapOtions),
    title: 'ejemplo'
  });
  marker.setMap(new google.maps.Map(document.getElementById("googleMap"),
      mapOtions));
}
</script>
```

Listing 3.7: Ejemplo de implementación de un mapa con la API de Google Maps

A diferencia de *Chart.js* la API de maps trabaja directamente dibujando el mapa en un marco de la página definido por la etiqueta `div`. Al analizar el ejemplo, para que se pueda acceder a la librería se necesita una clave de desarrollador de Google.

CAPÍTULO 4
Montaje del Hardware y del Servicio Web

En este capítulo se procederá a explicar como se ha montado todo este sistema. Además se hará una explicación de sus diferentes piezas y la función detallada que realiza cada una de ellas, tanto en la parte hardware como en la parte software.

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

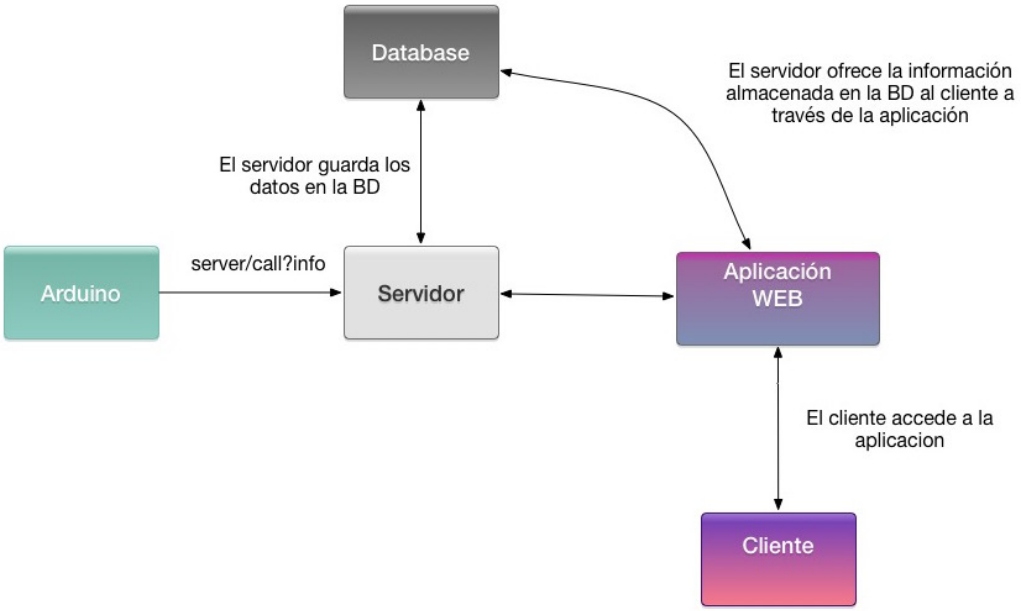


Figura 4.1: Diagrama de montaje del proyecto

En la **Figura** 4.1 se puede observar el diagrama que especifica la función que realizarán las diferentes partes del proyecto, desde el nodo de sensores hasta la información que podrá visualizar el cliente que acceda a la aplicación Web.

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. En esta controladora se realizaran todas las operaciones de la parte hardware del proyecto.

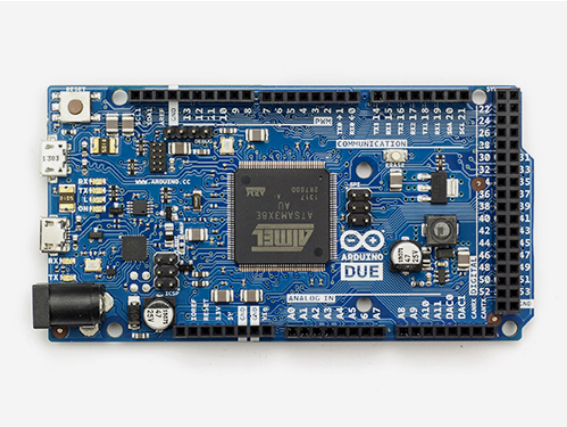


Figura 4.2: Microcontroladora Arduino DUE [2]

Ethernet shield: Es una *shield* que permite a la microcontroladora Arduino DUE conectarse a la red, se conecta por el puerto SPI que dispone Arduino. También incluye un lector de tarjetas por si en un futuro se quisiera realizar una gestión de ficheros. Para anexar la *shield* al Arduino solo hay que conectar los pines del puerto SPI del Arduino DUE en el conector de la *shield*.

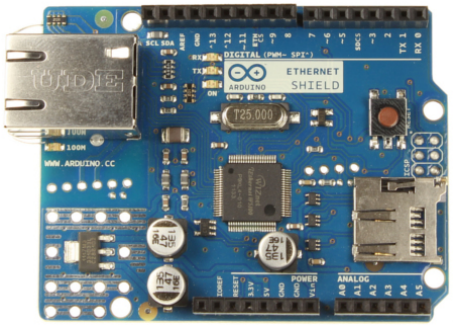


Figura 4.3: Ethernet shield [2]

Módulo GPS NEO6mv2: Es el módulo el cual una vez esté conectado al satélite enviará información de la geolocalización a la controladora.

Figura 4.4: Módulo GPS Neo6mv2

Sensores: Un conjunto de sensores permitirán la obtención de los valores ambientales en el entrono donde esté instalado el nodo.

Estos sensores son:

- **DHT11** Es el sensor de humedad y temperatura de Adafruit recogerá los datos climáticos del ambiente. El sensor está calibrado de serie por lo que en principio los valores que devuelve son precisos y no se necesitan calcular los datos que aporta este sensor.

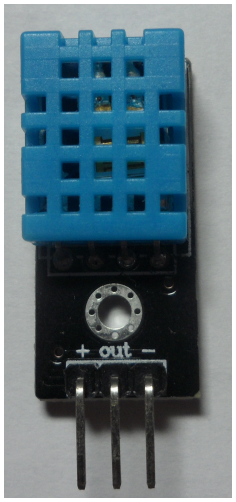
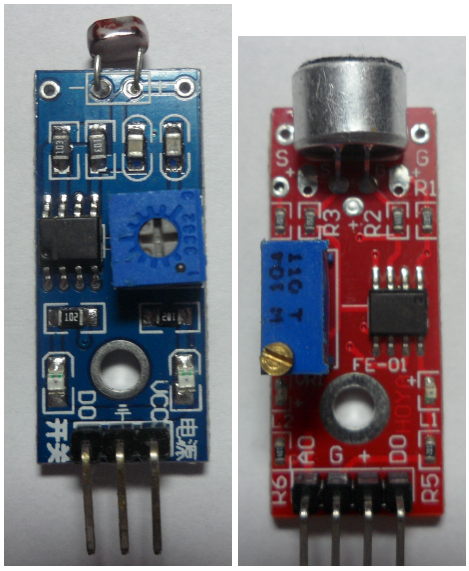


Figura 4.5: Sensor de temperatura y humedad DHT11

- **Sensor de gas CO MQ-7 y el chipset LM393** se va a utilizar para la medición de gases en el ambiente. El sensor ofrece un valor analógico correspondiente a las partes por millón analizadas en el ambiente, pero este valor puede no ser el correcto ya que el sensor necesita ser calibrado y no se dispone de material necesario para realizar dicha calibración.

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

- **Sensor luz y sonido** utiliza el chipset LM393. Medirá la contaminación lumínica y acústica. Estos sensores necesitan ser calibrados y sus valores pueden no ser los correctos, pero siguiendo su especificación, debe devolver decibelios para el ruido y lumen para la contaminación lumínica.



(a) Sensor de luz **(b)** Sensor de Ruido

Figura 4.7: Sensor de luminosidad y ruido

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

Las características de este servidor son:

- **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.
- **S.O:** OSX El Capitan 10.11.6.

Aplicaciones: Aplicaciones utilizadas que han permitido realizar el montaje de todo el sistema.

Entre ellas están:

- **Mamp:** Es una aplicación de escritorio para montar un servidor web Apache y Mysql bajo los puertos 8888 para el web y 8889 para el Mysql.

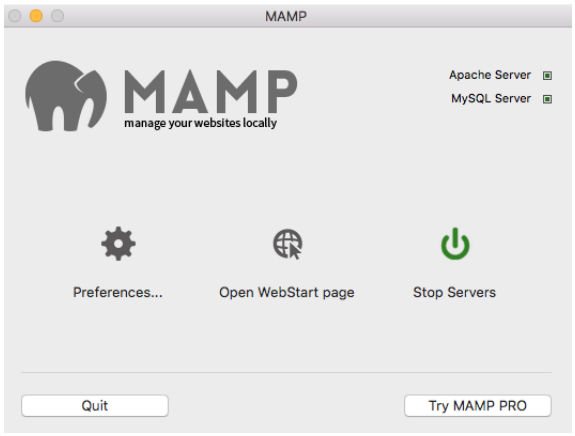


Figura 4.8: Ventana de la aplicación MAMP

- **MySQL Workbench** Laravel 5 es capaz de gestionar la base de datos través de PHP, pero se ha utilizado MySQL Workbench para poder visualizar y gestionar la información almacenada en la base de datos y depurar errores.

- **PhpStorm** Un IDE de desarrollo para programar con PHP así como gestionar proyectos con *Git*.

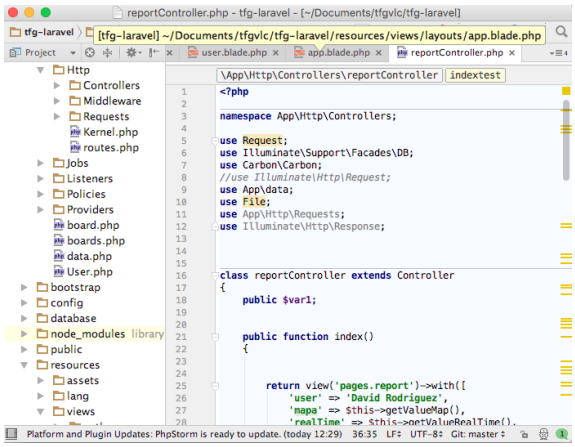


Figura 4.9: Ventana de la aplicación PHPStorm

Cableado: Distintos tipos de cableado para conectarlo todo.

4.2 Montaje del Nodo Arduino DUE

El nodo para esta aplicación será la microcontroladora Arduino DUE con los módulos de sensores que enviarán información al servidor, para ello se realizara el montaje siguiendo el esquema:

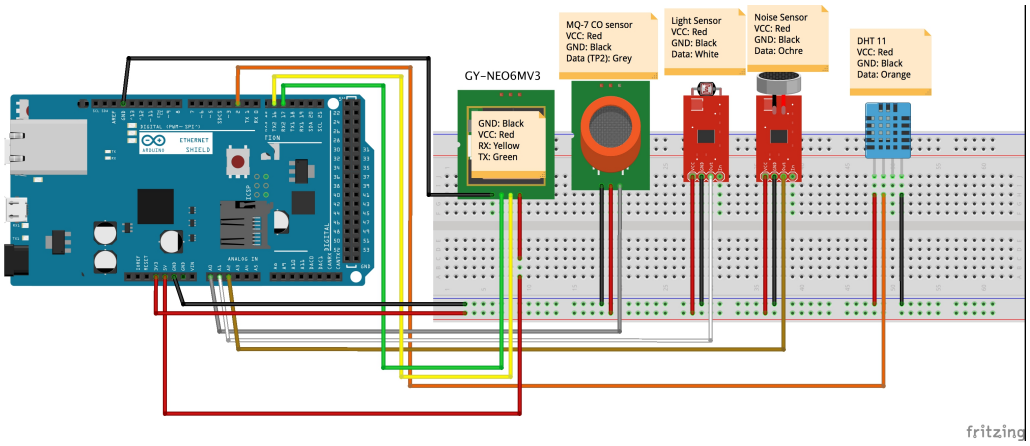


Figura 4.10: Esquema del montaje del Nodo

Como se puede apreciar en la **Figura 4.10** el montaje es sencillo ya que no necesita electrónica adicional como resistencias u otros componentes electrónicos. Para conectar todo el sistema, cada uno de los sensores se conectará VCC (cable rojo) a 3.3V ya que es el voltaje de funcionamiento de cada uno de los sensores, exceptuando el GPS que funcionará a 5V, ya que ha demostrado un mejor funcionamiento a ese voltaje. En GND (cable negro) para el negativo de los módulos. Todos los sensores van conectados a entradas analógicas (de A0 a A2) exceptuando el sensor de temperatura y humedad DHT11 que va a la entrada digital y el módulo GPS que va conectado al Serial 2.

Cabe decir, que al realizar la instalación en un Arduino UNO, al ser una versión más antigua, las lecturas digitales se leerán en un rango de 5V, por lo que hay que diseñar un circuito especial para poder leer el GPS y el sensor DHT11, porque 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 para poder leer correctamente estas entradas. El montaje quedaría de la siguiente manera:

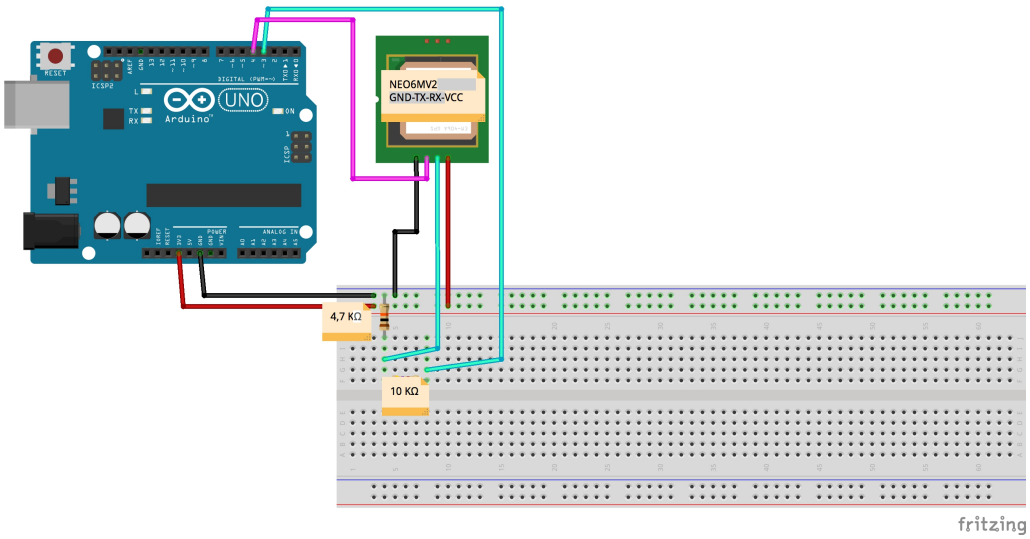


Figura 4.11: Esquema del circuito para el GPS en el Arduino UNO

El circuito de la **Figura 4.11** 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 hay que señalar que los RX y TX de la trasmisión serie van conectados de manera invertida entre el GPS y el Arduino.

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

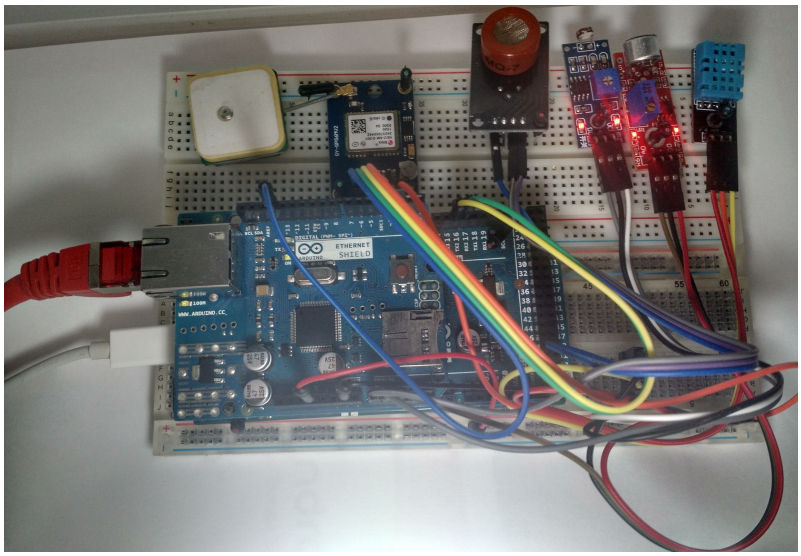


Figura 4.12: Montaje final del nodo de sensores

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

4.3 Programación de la Microcontroladora Arduino DUE

En esta sección se va a explicar cada una de las partes del código que utilizará el Arduino DUE, cada uno de estos fragmentos de código 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 obtener de la siguiente URL: https://github.com/zhelix/tfg-david/blob/master/Arduino/project_1.3.1/project_1.3.1.ino

Con el IDE de programación de Arduino se podrá abrir el siguiente código:

```
/*
*  FILE:      project-v1.3.1.ino
*  AUTHOR:    David Rodriguez Martinez  davidrm146@gmail.com
*  VERSION:   1.3.1
*/

//Necessary Libraries
#include <Ethernet.h>
#include <SPI.h>
#include <TinyGPS.h>
#include "DHT.h"

//Config of the device
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
TinyGPS gps;
#define DHTPIN 2
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
IPAddress ip(192, 168, 1, 102);
EthernetClient client;
String data;

void setup()
{
  Serial.println("EXECUTING setup");
  Serial.begin(9600);
  Serial2.begin(9600);
  if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    Ethernet.begin(mac, ip);
  }
  delay(1000);
}

void loop(){
  Serial.println("EXECUTING LOOP");
  //***************GPS CODE*********************
  bool newData = false;
  unsigned long chars;
  unsigned short sentences, failed;
  for (unsigned long start = millis(); millis() - start < 1000;){
    while (Serial2.available()){
      char c = Serial2.read();
      if (gps.encode(c))newData = true;
    }
  }
  float flat, flon;
  unsigned long age;
  gps.f_get_position(&flat, &flon, &age);
  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.
          satellites());
56    Serial.print(" PREC=");
57    Serial.println(gps.hdop() == TinyGPS::GPS_INVALID_HDOP ? 0 : gps.hdop());
58    data =
59    "board_id=" +  getID()  +
60    "&temp=" +  getTemperature()  +
61    "&hum=" + getHumidity() +
62    "&gas="+ getGas() +
63    "&noise=" + getNoise() +
64    "&luz="+ getLight() +
65    "&poslat=" + String(flat, 6) +
66    "&poslon=" + String(flon, 6);
67    displayData(data);
68    sendDataGet(data);
69    delay(60000);
70  }
71  /*
72  * Debugger option
73  */
74  void displayData(String data) {
75    Serial.print("GET /call?");
76    Serial.println(data);
77    Serial.println("HTTP/1.1");
78    Serial.println("Host: 192.168.1.16");
79    Serial.println("Connection: close");
80    Serial.println();
81  }
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    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
```

```
String getGas() {return String(analogRead(A0));}
String getNoise() {return String(analogRead(A2));}
String getLight() {return String(analogRead(A1));}
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 detalladamente en varias partes con el fin de que el usuario pueda entenderlo:

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

```
#include <Ethernet.h>
#include <SPI.h>
#include <TinyGPS.h>
#include "DHT.h"

//Config of the device
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
TinyGPS gps;
#define DHTPIN 2
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
IPAddress ip(192, 168, 1, 102);
EthernetClient client;
String data;
void setup()
{
  Serial.println("EXECUTING setup");
  Serial.begin(9600);
  Serial2.begin(9600);
  if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    Ethernet.begin(mac, ip);
  }
  delay(1000);
}
```

Listing 4.2: Configuración de la Controladora Arduino Due

Antes de entrar al código se implementarán 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 más sencilla. Las librerías son:

- #include <Ethernet.h>: Permite gestionar la conexión a la red.
- #include <SPI.h>: Facilita al Arduino DUE comunicarse con la Shield de Ethernet.
- #include <TinyGPS.h>La librería traduce el NMEA data en valores legibles sin necesidad de manipular la cadena de NMEA recibida.
- #include 'DHT.h': Esta librería traduce la información recibida por el sensor DHT11, sólo 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 a traves serie para el módulo GPS conectado al Serial2 a 9600 baudios aunque aplicando una serie de cadenas hexadecimales en el arranque del programa escritas en el GPS se puede cambiar esta tasa. También se configurará la conexión serie perteneciente al puerto USB para poder observar la información que

envía cada vez que se repita el bucle del programa, con el fin de poder depurar en caso de algún problema o fallo de los componentes.

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

Listing 4.3: Conexión a la Red

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

Una vez configurado todo el programa ya estará listo para poder leer la información que recojan los módulos de sensores. De esta manera se diseñará el siguiente código para realizar la lectura en la definición del *LOOP* el cual se ejecutará indefinidamente:

```
void loop(){
  Serial.println("EXECUTING LOOP");
//**************GPS CODE*********************
  bool newData = false;
  unsigned long chars;
  unsigned short sentences, failed;
    for (unsigned long start = millis(); millis() - start < 1000;){
      while (Serial2.available()){
        char c = Serial2.read();
        if (gps.encode(c))newData = true;
      }
    }
  float flat, flon;
  unsigned long age;
  gps.f_get_position(&flat, &flon, &age);
  Serial.print("LAT=");
  Serial.print(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat, 6);
  Serial.print(" LON=");
  Serial.print(flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flon, 6);
  Serial.print(" SAT=");
  Serial.print(gps.satellites() == TinyGPS::GPS_INVALID_SATELLITES ? 0 : gps.
      satellites());
  Serial.print(" PREC=");
  Serial.println(gps.hdop() == TinyGPS::GPS_INVALID_HDOP ? 0 : gps.hdop());

//The next variable keeps the information collected by the sensors
  data =
    "board_id=" +  getID()  +
    "&temp=" +  getTemperature()  +
    "&hum=" + getHumidity() +
    "&gas="+ getGas() +
    "&noise=" + getNoise() +
    "&luz="+ getLight() +
    "&poslat=" + String(flat, 6) +
    "&poslon=" + String(flon, 6);

//Debug function, this string works
//data ="board_id=1&temp=23&hum=43&gas=53&noise=54&luz=55&poslat=63&poslon=73";
    displayData(data);
//This function is for send information to the server
    sendDataGet(data);
delay(60000);
}
```

Listing 4.4: Función *LOOP* del código

En el **código 4.4** se realizan tres funciones, la lectura del GPS, el envío de la cadena que contiene las funciones que recogen de datos de los sensores y la pausa que establecerá 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. Utilizando la librería TinyGPS.h traducirá la información directamente sin tener que editar manualmente el NMEA data. De la información obtenida del GPS leyendo el Serial2 del Arduino DUE, interesará **flat** y **flon** que son las coordenadas obtenidas por el GPS.

```
1  float flat, flon;
2  unsigned long age;
3  gps.f_get_position(&flat, &flon, &age);
4  Serial.print("LAT=");
5  Serial.print(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat, 6);
6  Serial.print(" LON=");
7  Serial.print(flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flon, 6);
8  Serial.print(" SAT=");
9  Serial.print(gps.satellites() == TinyGPS::GPS_INVALID_SATELLITES ? 0 : gps.
       satellites());
10 Serial.print(" PREC=");
11 Serial.println(gps.hdop() == TinyGPS::GPS_INVALID_HDOP ? 0 : gps.hdop());
```

Listing 4.6: Traducción y depuración de los datos del GPS

En el **Código 4.6** siguiendo el ejemplo de la librería de *TinyGPS* se puede observar que que la función que recoge la posición es `gps.f\_get\_position(\&flat, \&flon, \&age);` que es la que guardará la información en las variables `flat` y `flon`.

Los `Serial.print` sirven para mostrar la información obtenida ya que son necesarios para depurar cuando no conecta al satélite con el fin de mejorar la conexión. Todo esto se puede observar en en una comunicación serie que ofrece la aplicación de Arduino IDE, como se observa en la **Figura 4.13**.

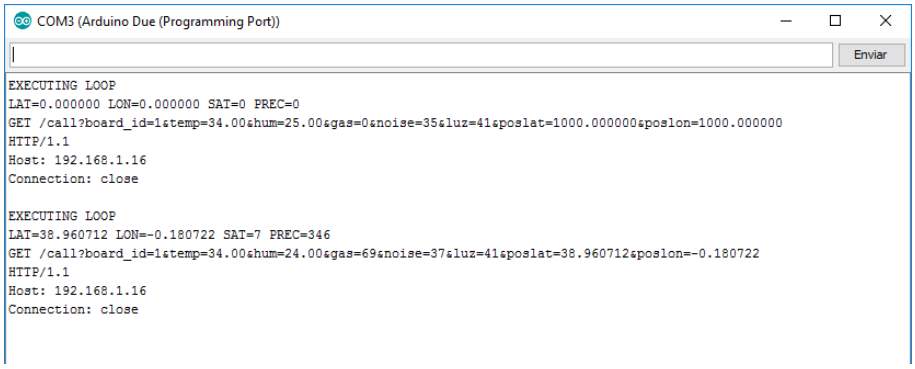


Figura 4.13: Terminal serie de Arduino y envío de la petición GET

Adicionalmente, la información es enviada 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 a enviar 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 funcionará y no se podrá realizar la inserción a la base de datos. Este es un ejemplo de como debería ser definida:

```
    data="board_id={id} +
      &temp={temperatura} +
      &hum={humedad} +
      &gas={medicion del gas} +
      &noise={ruido ambiental} +
      &luz={luminosidad} +
      &poslat={latitud} +
      &poslon={longitud}";
```

Listing 4.7: Formato de la String que se enviará

Y por último una pausa de 1 minuto (60000 milisegundos), que evitará que el Arduino esté enviando información constantemente.

Una vez explicado el bucle se explicarán las funciones que se ejecutan en la variable de datos y que se encargan de obtener la información ambiental de la zona donde esta situado el nodo:

```
String getTemperature() {
  float t = dht.readTemperature();
  float f = dht.readTemperature(true);
  if (isnan(t) || isnan(f)) {
    Serial.println("Failed to read Temperature from DHT. . .");
    return "0";
  }
  return String(t, 2);
}

String getHumidity() {
  float h = dht.readHumidity();
  if (isnan(h)) {
    Serial.println("Failed to read Humidity from DHT. . .");
    return "0";
  }
  return String(h, 2);
}

String getGas() {return String(analogRead(A0));}

String getLight() {return String(analogRead(A1));}

String getNoise() {return String(analogRead(A2));}

String getID() {return "1";}
```

Listing 4.8: Funciones de recogida de datos

En las funciones `getTemperature()` y `getHumidity()` se puede observar que utilizando la librería *dht.h* se realiza la conexión y lectura de la información del sensor sin tener que crear código adicional para medirlo. Sobre las demás funciones son lecturas analógicas de las mismas que darán unos valores normales en su rango de 0 a 1024. Y la última sirve para establecer la ID en la que el nodo es registrado en la base de datos, el número debe coincidir con su controladora como ya se explicará más adelante.

Una vez ya esta explicado el código se explicará la llamada a la web la cual realizará la inserción a la BD, este es su código:

```
void sendDataGet(String data) {
  if (client.connect("192.168.1.16",8888)){
    client.print("GET /call?");
    client.println(data);
    client.println("HTTP/1.1");
    client.println("Host: 192.168.1.16");
    client.println("Connection: close");
  }
  if (client.connected()) { client.stop();}
}
```

Listing 4.9: Función de comunicación con el servidor

Esta función realiza la petición GET al servidor Web que se configurará con la IP y el puerto en el que esté trabajando el servidor. La IP puede ser pública o privada por lo que, Arduino puede trabajar en Internet aunque es mucho más inseguro para la aplicación, pues uno de sus puntos débiles es que Arduino no tiene capacidad para la encriptación y tampoco tiene soporte para peticiones HTTPS. por lo que desde mi punto de vista, una gran fallo 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 será almacenada en la BD.

La dirección del servidor Web quedaría algo como se muestra en la siguiente estructura:

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

Listing 4.10: Formato de la String GET

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

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

Listing 4.11: Controlador 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 esta información se ejecutará el modelo 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 pues 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 4.14: 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. En el proyecto se ha utilizado el servidor MAMP que es una aplicación de escritorio OSX que dispone de estas características.

Para la instalación de la aplicación en el servidor se tiene que descargar de este enlace un paquete que contiene archivos de la aplicación Web:

https://github.com/zhelix/tfg-david/releases/tag/1.0

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

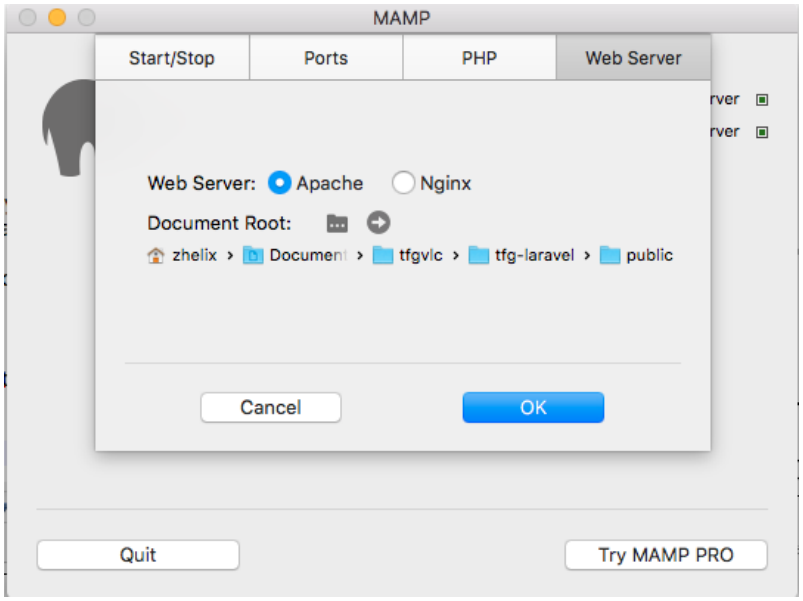


Figura 4.15: Configuración del Servidor MAMP

En la ventana de la **Figura 4.15** 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 dejarán por defecto. El puerto HTTP lo configurará al 8888 y el servidor de Base de Datos al 8889.

Una vez activado se creará 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** que 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 4.12: Configuración de la BD en Laravel

En el **Código 4.12** se puede ver editar la configuración de la base de datos. En este caso, se tiene que poner la configuración del servidor Web MAMP para que pueda conectar, por desgracia la versión del MAMP es *Trial* por lo cual, el usuario por defecto es *root* y la contraseña *root*.

Si toda la conexión está bien realizada cuando se ejecute la migración en el Terminal no dará ningún problema:

```
1  php artisan migrate tfgDatabase
```

Listing 4.13: Migración en OSX

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 tiene que instalar el servicio WEB y el servicio de base de datos para poder alojar la aplicación Web, para ello este ejemplo se va a realizar en un servidor Debian que utiliza un gestor de paquetes llamado apt.

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

Listing 4.14: Instalación del Servicio Apache en Linux

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

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

Para poder acceder como *localhost:8888* a la aplicación hay que realizar unos cambios en el servidor Apache2 y no poner rutas adicionales ya que puede generar problemas de seguridad. Para que esto no pase, en el fichero /etc/apache2/sites-available/000-default.conf se modifica el *DocumentRoot* por esto:

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

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 4.12**.

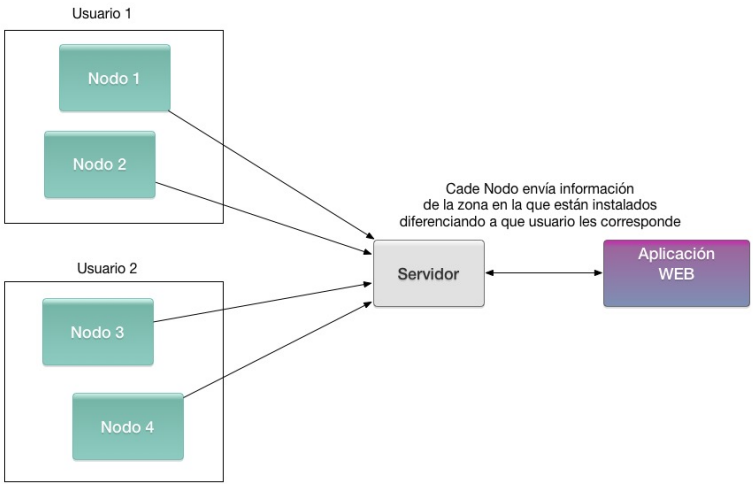
```
1 sudo php artisan migrate tfgDatabase
```

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 explicará como funcionará la aplicación Web diseñada para recoger y mostrar los datos de cada uno de los nodos.



¡

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 gestiona los datos 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 almacena estos datos 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 información en la aplicación 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:

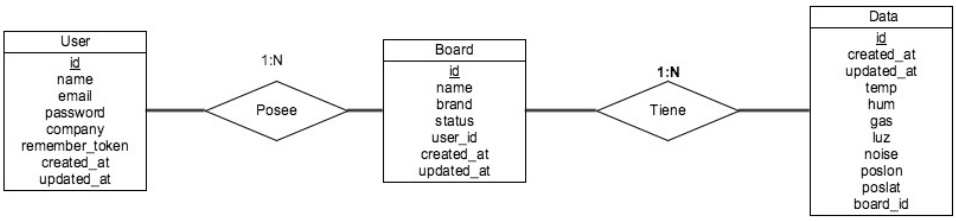


Figura 5.2: Diagrama Entidad-Relación de la base de datos

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.

```
class TfgDatabase extends Migration{
  /**
  * Run the migrations.
  * @return void
  */
  public function up(){
    Schema::create('board', function (Blueprint $table) {
      $table->increments('id');
      $table->string('name');
      $table->string('brand');
      $table->string('status');
      $table->integer('user_id')->unsigned();
      $table->timestamps();
      $table->foreign('user_id')->references('id')->on('users');
    });
    Schema::create('data', function (Blueprint $table) {
      $table->increments('id');
      $table->timestamps();
      $table->string('temp');
      $table->string('hum');
      $table->string('gas');
      $table->string('luz');
      $table->string('noise');
      $table->string('poslon');
      $table->string('poslat');
      $table->integer('board_id')->unsigned();
      $table->foreign('board_id')->references('id')->on('board');
    });
  }
  /**
```

```
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 más 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 4.13**.

5.2 Gestión del *NodeTracker*

Una vez explicado como se almacena la información se procederá a explicar como puede acceder un usuario a la aplicación que 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:

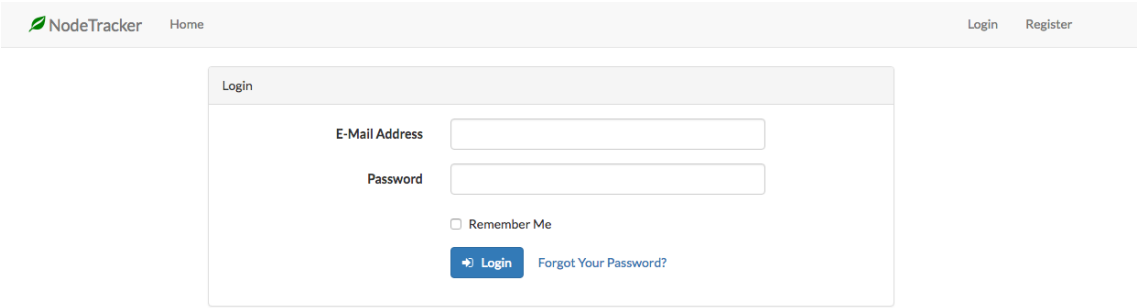


Figura 5.3: Login del *NodeTracker*

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

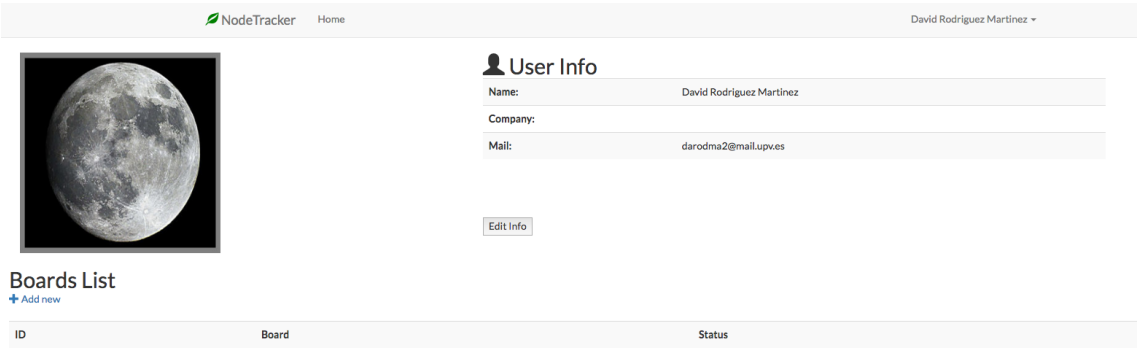


Figura 5.4: Home del *NodeTracker*

Con un pequeño formulario se puede editar la información básica del usuario (**Figura 5.5**). Como se ha diseñado para esta aplicación, se pueden añadir uno o más nodos (**Figura 5.6**), esta funcionalidad devolverá una ID en la tabla para su posterior introducción en la función `getID()` del código del Arduino, con el fin de que guarde la información correcta al usuario correspondiente del nuevo nodo en la base de datos.

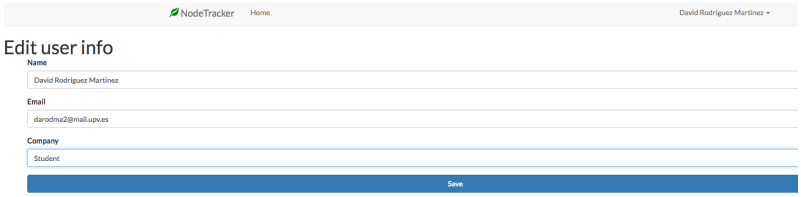


Figura 5.5: Formulario de edición de usuarios del *NodeTracker*.

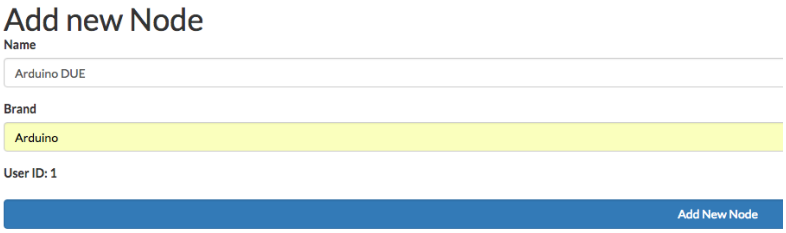


Figura 5.6: Formulario para añadir un nuevo nodo al sistema.

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

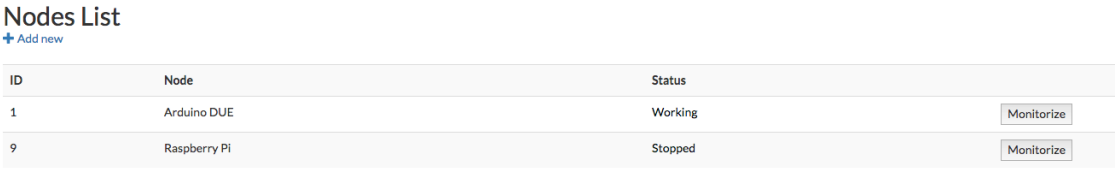


Figura 5.7: Lista de nodos disponibles

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 `getID()` del código en el Arduino. Por ejemplo, si se quisiera enviar información desde una Raspberry Pi cuya ID es 9, habrá que configurar el programa para que la cadena a enviar sea la siguiente:

```
direccion IP:8888/call?board_id=[9]&datos....
```

También esta el botón que redirigirá a la página de los reportes para poder visualizar la información obtenida de cada nodo conectado con la aplicación. Es un panel donde se podrá visualizar la información o generar un reporte con los datos recogidos.

5.3 Visualización de la información

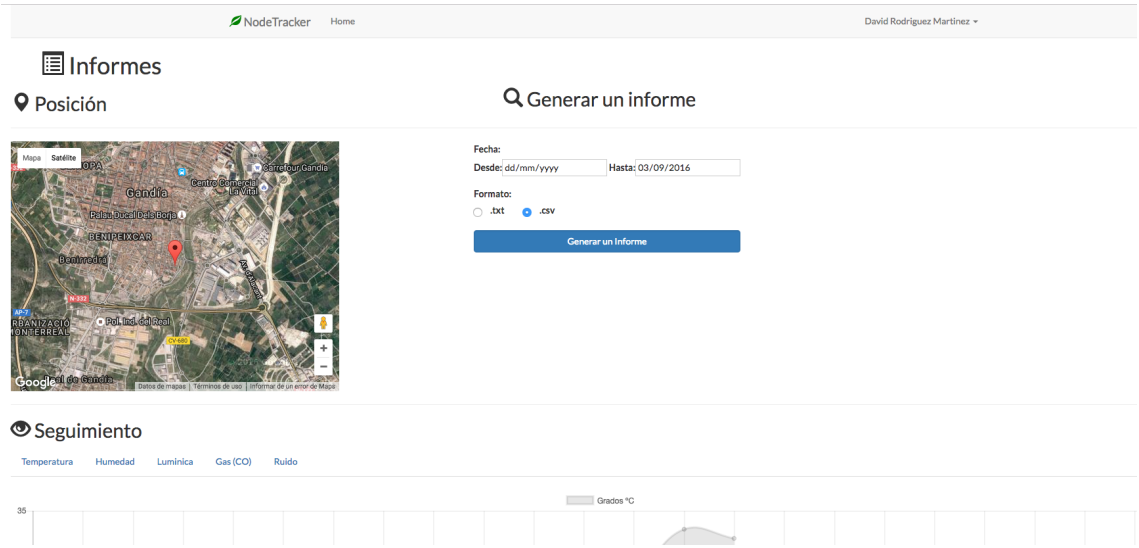


Figura 5.8: Pagina de monitorización de la aplicación

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

5.3.1. Posicionamiento

El Mapa se muestra en la sección de *Posición*. Este mapa se ha podido implementar utilizando la API de Google MAPS. En esta sección de la aplicación se mostrará el mapa es en vista satélite que, al ser una aplicación medioambiental, se necesitará saber si el nodo esta en un núcleo urbano o en territorio natural, y aproximadamente a que altura se puede encontrar, costa o altas montañas. *Para el ejemplo de la* ***Figura 5.9*** *se dispone de las coordenadas -0.180812 y 38.960541 que corresponden al termino municipal de Gandía.*

Figura 5.9: Mapa de la posición del nodo.

Se ha implementado este mapa (*indicando en las opciones del mapa que su configuración de vista sea por satélite*) pues se puede apreciar cuando se trata de un núcleo urbano, terreno montañoso, ríos o mares con el fin de que cuando se recojan los datos el usuario tenga cierta información del área que se esta monitorizando. Analizando la **Figura 5.9** se puede apreciar la montaña en la parte izquierda y el núcleo urbano en la parte derecha.

5.3.2. Generar un informe

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

Figura 5.10: Generación de informes

Esta función generará un informe en formato JSON de la información recogida por el nodo. El resultado puede variar en el formato que se ha seleccionado en cada caso, `.txt` (texto plano) o `.csv` (Hojas de calculo para LibreOfice), y asi para poder utilizar estos ficheros en otras aplicaciones o realizar migraciones. Lo descrito anterior resulta 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 ultimo el la función de monitorización, llamada *seguimiento*, se ha implementado una serie de gráficas, 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. Se presenta una gráfica diferente para cada valor a medir (temperatura, humedad, gases, luz y ruido) que implementado con AJAX realiza el cambio de gráficos sin necesidad de refrescar la página.

Este es el código que permite cambiar de gráfico sin necesidad de recargar toda la página[13]:

```
<script>
    function cambiarPestanna(divc,divc) {
    divc = document.getElementById(divc.id);
    listaPestannas = document.getElementById(divc.id);
    cdivc = document.getElementById('c'+divc.id);
    listacPestannas = document.getElementById('contenido'+divc.id);
    i=0;
```

```
8       while (typeof listacPestannas.getElementsByTagName('div')[i] != 'undefined'
            ){
9           $(document).ready(function(){
10            $(listacPestannas.getElementsByTagName('div')[i]).css('display','none
                  ');
11            $(listaPestannas.getElementsByTagName('li')[i]).css('background','');
12            $(listaPestannas.getElementsByTagName('li')[i]).css('padding-bottom',
                  '');
13          });
14          i+=1;
15        }
16        $(document).ready(function(){
17          $(cdivc).css('display','');
18          $(divc).css('padding-bottom','2px');
19        });
20      }
21  </script>
```

Listing 5.2: Selector de Graficos

Esta función recoge todas las secciones que coinciden con el identificador asignado, una vez se seleccione la pestaña a cambiar recogerá su dirección y recorrerá todos los disponibles y los ocultará para mostrar la gráfica asignada a la pestaña seleccionada.

Estas son dos ejemplos de las gráficas para mostrar la información recogidas por el nodo:

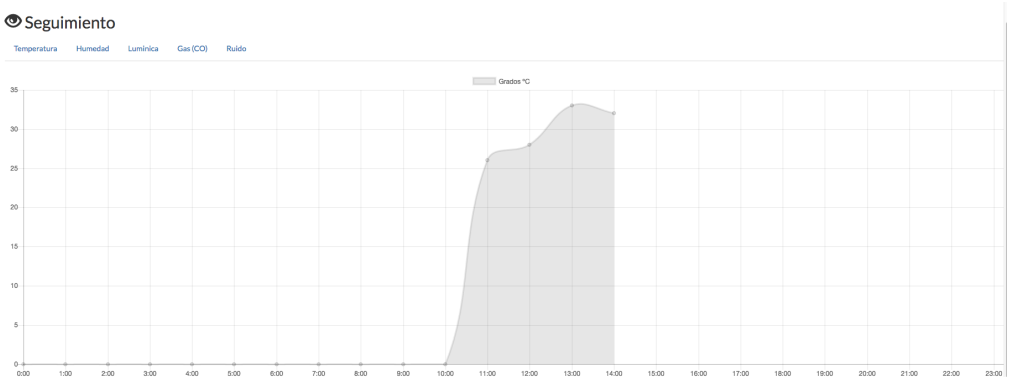


Figura 5.11: Gráfica perteneciente a la temperatura en el seguimiento

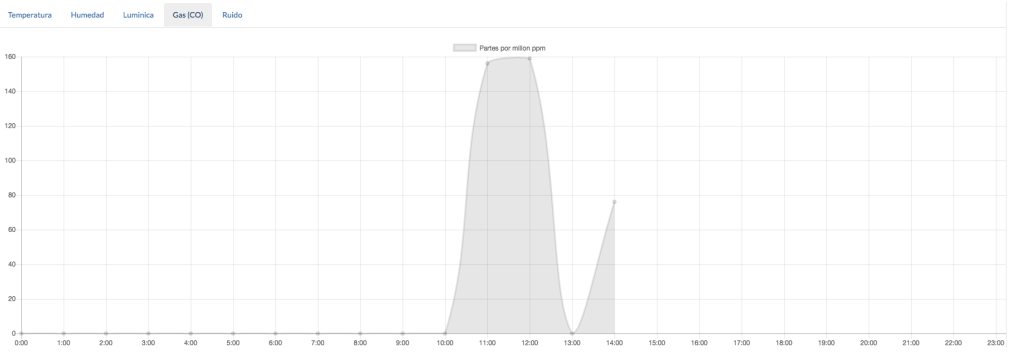


Figura 5.12: Gráfica perteneciente a la contaminación por CO en el seguimiento

(Uno de los principales problemas que se ha tenido a la hora de diseñar esta gráfica ha sido que cuando se reciben los datos, esta no interpretaba bien la hora de cada entrada, así que al dibujar el gráfico empezaba pero se ha solucionado aplicando un 0 a cada hora que el nodo ha estado deshabilitado con el fin de prevenir errores en las lecturas de la aplicación.

También comentar que debido a los numerosos cambios de formato de una versión a otra de *Chart.js* la información es muy confusa pues en cada versión cambia la forma de implementar los atributos de la gráfica y por eso ha resultado muy complicado añadir una magnitud a los valores del eje Y por lo que se ha tenido que implementar con una leyenda que informe al usuario de la magnitud que se esta empleando para medir en esa gráfica.

Otro problema es la selección del color, en este caso queda con color gris aunque se le esta asignando un color y puede ser debido a lo descrito anteriormente y pueda ser una llamada errónea en la librería)

Aquí se presenta un ejemplo de una gráfica en las primeras de la librería de *Chart.js*:

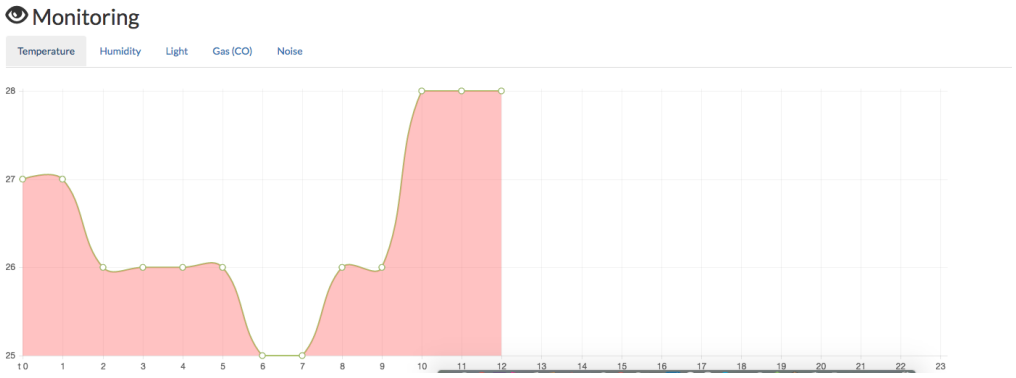


Figura 5.13: Gráfica perteneciente a la Temperatura (*versión antigua Chart.js)*

CAPÍTULO 6
Conclusiones y mejoras

6.1 Seguridad

Después de utilizar Arduino se ha podido observar que es un sistema bastante bueno para el aprendizaje, pero para una aplicación real resulta poco útil, pues suponiendo que en un entorno educacional si se produce algún error se puede corregir en el momento, en un entorno real no puede ir un operario a corregir dichos errores cada ver que ocurran. Además debido a la incapacidad nativa del SAM3X para encirptar la información, (en este caso se pretendía realizar peticiones HTTPS al servidor), el sistema es muy débil a ataques MITM (*Man-in-the-middle*) puesto que la información que envía no ha recibido ningún tipo de encriptación, ni existe algún tipo de bloqueo a peticiones externas que no tengan nada que ver con la aplicación. Después de lo descrito en este párrafo se podría afirmar que la aplicación no puede ser una solución final, ya que se puede romper fácilmente 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, se ha pensado en utilizar controladoras que permitan encriptación o conexión SSH lo cual requeriría un rendimiento mayor y un elevado consumo. Otra solución podría ser que los nodos solo se puedan conectar al servidor en una red interna, como defecto, esta solución limitaría a un área muy reducida, pero si el sistema esta conectado a Internet, el área de alcance es a nivel global pero inseguro, por lo que resolver algunos problemas dan lugar otros nuevos.

6.2 Sensores

Se ha obtenido el aprendizaje en el funcionamiento de la electrónica básica debido a los diversos problemas entre versiones de Arduino y los cambios realizados en los circuitos para poder ser implementados. Además de otros problemas, los sensores no dan siempre un valor fiable. En el proyecto se han tratado como valores anómalos, que son datos no correspondidos con lo que deberían de aportar del ambiente, 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 implementar el circuito en una *protoboard*, se hubiera creado una placa impresa y soldado en esta, los componentes

directamente. Además estos pines se conectarían al Arduino con el fin de suprimir la *protoboard*, cables y componentes de electrónica básica que realizan un montaje caótico del nodo. Esto eliminaría los errores por fallos de conexión o cortocircuitos en la aplicación además de reducir notablemente el tamaño del nodo.

- **Sensores** Los sensores que se han utilizado para el proyecto son de un coste reducido y por tanto, no son para una larga duración. Por ejemplo, hay sensores que durante el montaje de este proyecto se han estropeado. Para solucionar esto se deberían implementar sensores más eficientes de un coste más elevado, por ejemplo reemplazar el DHT11 de unos 2€y este no aporta valores decimales, por su versión más costosa, el DHT33 añadiendo la función de valores decimales, por 17€.

6.3 Boards especializadas

Después de haber finalizado en este proyecto, del que se ha podido obtener de información sobre Arduino, es que no sirve mucho para realizar este tipo de funciones, como se ha descrito anteriormente solo sirve para el ámbito educacional. Con el fin de implementar una mejora, todo este sistema ya existe a nivel comercial, así que buscando en Internet se ha obtenido información sobre una controladora que, ofrezca datos de manera segura y que permita conectarse por una red, tanto cableada como inalámbrica (incluyendo GPRS) y que tenga un consumo muy reducido. Se trata de la controladora FlyPort, que soluciona los problemas que se han descrito anteriormente. Utilizando este modelo se puede crear redes y conectarlas de manera segura al servidor. El problema radica en que al ser una controladora comercial, su precio es muy elevado y aunque la controladora sea barata, requiere de un sistema para poder implementar los sensores y poder programarla que es lo que aumenta su coste.

En la **Figura 6.1** se muestra el *Starter kit* de la controladora en cuestión:

Figura 6.1: FlyPort GPRS y la placa para la controladora.[11]

El coste de este *Starter kit* es de unos 150€ aproximadamente y como se ha hablado en la sección anterior también se va a añadir el coste de sensores de calidad al nodo, teniendo en cuenta que al ser de calidad son de un coste mucho más elevado que los utilizados para este proyecto, el coste por nodo puede subir perfectamente a los 200€ .

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.
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 http://www.ti.com/lit/ds/symlink/lm2903-n.pdf. https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-7.pdf. (Visitado el 15-07-2016)

[6] Información sobre como desarrollar una aplicación WEB utilizando el Framework Laravel 5.2. consultado en https://laravel.com/docs/5.2. (Visitado el 12-03-2016)

[7] Información y descarga de la aplicación para el servidor WEB. https://www.mamp.info/en/. (Visitado el 12-03-2016)

[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)

[11] Informacion sobre el nodo de sensores *FlyPort*. http://wiki.openpicus.com/index.php/Flyport_Tutorials. (Visitado el 03-09-2016)

[12] Informacion y gestión del proyecto. https://bitbucket.org/. (Visitado el 03-09-2016)

[13] Informacion sobre la tecnica de desarrollo AJAX. http://www.uco.es/~lr1maalm/manualdeajax.pdf. (Visitado el 09-07-2016)

[14] Informacion sobre Git para la gestión del Proyecto. https://git-scm.com/book/es/v1/Empezando-Fundamentos-de-Git. (Visitado el 03-09-2016)