Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

# Internet de las cosas: Desarrollo de un servidor Domótico

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor**: Soriano Saiz, Saúl

**Tutor**: Albaladejo Meroño, José

Curso 2014-2015

# Resumen

El objetivo del TFG es el desarrollo e instalación de un servidor domótico de bajo coste en una red doméstica. Centrado en la creación de un sistema comunicación entre las plataformas Arduino y Raspberry a través de red Eth/wifi.

Este servidor ofrece al usuario un control domótico a través de una aplicación web. La comunicación entre el servidor y los nodos Arduino usa un protocolo de descubrimiento basado en UDP, con un proceso continuo de lectura de datos que serán almacenados en la BBDD. La página web utilizara las tecnologías AJAX y PHP para leer los datos almacenados en el servidor y mostrarlos en tiempo real.

Configuraremos una VPN (Virtual Private Network), la cual nos permitirá acceder a nuestra red local desde cualquier punto.

Además añadiremos más servicios al servidor: la plataforma Owncloud nos permitirá tener nuestro propio sistema de almacenamiento de datos en línea, por otro lado el servidor de impresión nos permitirá imprimir desde cualquier punto de la red.

**Palabras clave:** IoT, Plataforma Web, PHP, JAVA, HTML, CSS, JavaScript/jQuery, Mysql, Owncloud, Arduino, Raspberry.

# Abstract

The goal of this Project End of Degree is the development and installation of a low cost domotic server in a domestic network. Focused on the creation of a communication system between Arduino and Raspberry platforms through Eth/Wifi net.

This server offers the user a domotic control through a web application. The communication between the server and the Arduino nodes uses a discovery protocol based on UDP, with a continuous process of data reading that will be stored on the database. The webpage will use AJAX and PHP technologies to read the data stored in the server and show them on real time.

We will configure a VPN (Virtual Private Network) which will allow us inside our local network from wherever we are.

Besides, we will add more services to the server: Owncloud platform will allow us to have our own data storage online, on the other hand, the printout server will allow us to print from anywhere on the network.

**Keywords :** in IoT, Plataforma Web, PHP, JAVA, HTML, CSS, JavaScript/jQuery, Mysql, Owncloud, Arduino, Raspberry.

# Tabla de contenidos

# 1. Introducción


*Ilustración 1: El internet de las cosas*

Este proyecto se ha desarrollado en base a un concepto, el internet de las cosas (IoT). IoT hace referencia a la interconexión de objetos mundanos a internet. Presentaría un punto donde un gran número de objetos cotidianos, que actualmente podemos encontrar en la mayoría de hogares, estén interconectados a nuestras redes domésticas con el fin de controlarlas o monitorizarlas.

Con esa premisa se ha desarrollado este servidor de domotización, que nos ayudara tanto a centralizar nuestros servicios en red como permitirnos controlar microcontroladores Arduino. Este servidor se ha diseñado para utilizar microcontroladores Arduino para los nodos y Raspberry PI para el servidor, ambas son open-source (código abierto), lo que significa que tanto el software como el hardware son de libre distribución. La interconexión de los microcontroladores se realiza vía Ethernet a través de nuestra red de área local, y la comunicación utilizada es UDP (User Datagram Protocol). Los microcontroladores se identifican dentro de la red mediante un proceso de identificación, el cual es iniciado por el servidor, tras el cual es microcontrolador puede tanto enviar información como recibir. Por cuestiones de comodidad se tomó la decisión de dividir los posibles nodos en dos tipos actuadores y sensores. Los actuadores son elementos que podemos controlar a través de nuestro servidor (servomotores, relés,…), mientras que los sensores son elementos que sólo envían información al servidor para ser monitorizada.

El servidor cuenta también con un contenedor web, en el cual se encuentra alojada la web donde está la interfaz de usuario. Esta web nos muestra los diferentes nodos conectados a nuestra red, con los datos enviados al servidor. A través de esta interfaz se lleva a cabo el proceso que termina registro de los nodos conectados y enviar datos de forma cómoda a los diferentes nodos conectados. La web cuenta además con una serie de eventos, totalmente transparentes al usuario, que actualizan los datos de los nodos y proporcionan un sistema de alertas para el usuario.

Anteriormente mencionábamos que el servidor contaría con una serie de servicios en red, entre ellos podemos destacar el servidor de impresión que nos permite imprimir desde cualquier punto de la red y la VPN (Virtual Private Network), que nos permite conectarnos a nuestra red

doméstica desde cualquier punto de internet de manera segura.

# 2. Análisis

En el siguiente apartado se presentan los diferentes componentes que necesitamos para cada uno de los elementos que forman parte del sistema. También presentamos la tecnología seleccionada para realizar la tarea concreta.

## 2.1 Microcontroladores


*Ilustración 2: Logo Arduino*

Para poder interconectar los diferentes objetos cotidianos a nuestra red, vamos a utilizar microcontroladores. Utilizaremos esta electrónica para gestionar la comunicación entre los diferentes elementos y el servidor que almacena los datos. Concretamente los microcontroladores son circuitos integrados programables, en los que podemos albergar nuestros propios códigos. En nuestro caso estos códigos establecerán la conexión con el servidor, con el fin de dar de alta al elemento dentro del sistema y enviar la información pertinente. En el mercado he encontrado muchas alternativas de entre las que cabe destacar Arduino y Nanode.

Hemos elegido para este proyecto la tarjeta Arduino Mega. Arduino es un conjunto bastante grande de microcontroladores basados en el microcontrolador ATmel, basado en una arquitectura RISK de 8 bits. La iniciativa de estas tarjetas surge como el proyecto de unos estudiantes italianos, que querían obtener microcontroladores de bajo coste y consumo.

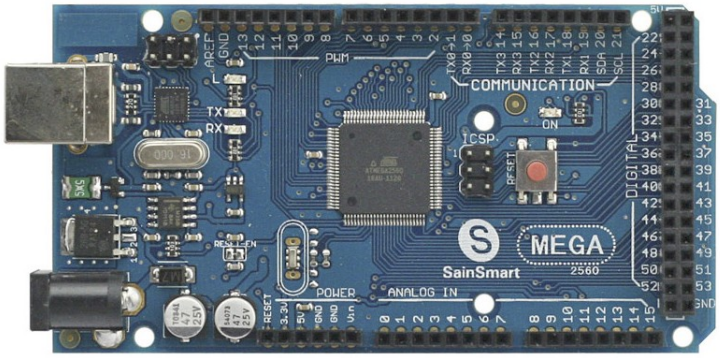



*Ilustración 3: Arduino Mega R3*
**Arduino Mega**

Las especificaciones técnicas de la tarjeta son:

| Características | Arduino Mega |
|---|---|
| Microcontrolador | Atmega2560 |
| Voltaje de alimentación | 5V |
| Pines digitales de E/S | 54 donde 15 son pwd |
| Pines analógicos | 16 |
| Memoria Flash | 248 KB utilizables |
| SRAM | 8KB |
| EEPROM | 4KB |
| Velocidad de Reloj | 16MHz |

Se ha elegido esta tarjeta esta tarjeta de entre todas las que podemos encontrar actualmente en el mercado, por su versatilidad. Por un lado es muy cómoda de utilizar, puesto que esta marca concreta cuenta con un puerto USB de tipo b que nos permite conectarla a cualquier equipo. Además también es compatible con todas las tarjetas de expansión que han sido diseñados para esta familia, entre las que podemos encontrar toda clase de circuitos externos que podemos añadir a nuestro microcontrolador para dotarla de nuevas funcionalidades.

### 2.1.1 Módulo de Ethernet

Como este dispositivo no puede ser conectado de forma directa a la red, vamos a necesitar algo de ayuda, por lo que vamos a utilizar la shield de expansión para Ethernet compatible con Arduino.

Actualmente podemos encontrar una gran variedad de circuitos de expansión para poder conectar nuestra tarjeta a nuestra red, entre ellas podemos encontrar tanto versiones para conectarte a través de Ethernet como WIFI. Se han estudiado varias alternativas para esta

conexión, el módulo ENC28j60 tuvo que ser descartado debido a que no termina de ser compatible con el Arduino Mega al 100%, debido a un problema de diseño sobre uno de los pines de este. Otro modulo que se investigo fue el esp8266, este circuito es muy fácil de conectar a nuestro Arduino y nos permite conectarlo a la red a través de la red WIFI.

Finalmente se optó por la opción más sencilla, la tarjeta de expansión shield Ethernet de la propia marca Arduino.

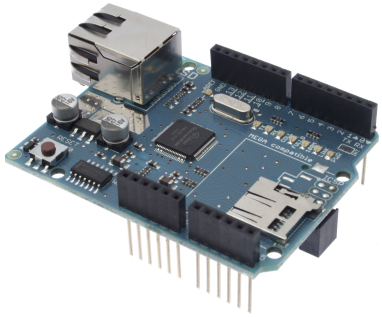


*Ilustración 4: Shield Arduino Ethernet*

Esta tarjeta tiene el mismo formato físico que la versión UNO de Arduino. Concretamente esta versión cuenta además con un zócalo de expansión para tarjetas SD, para utilizar como sistema de almacenamiento ya que nos permitirá escribir y leer todos los datos que necesitemos. Por último destacar que el conector de RJ47 cuenta con la tecnología POE (Power One Ethernet), la cual nos permite alimentar el sistema a través del cable de Ethernet al mismo tiempo que viajan los datos hacia la red y viceversa.

La estructura de la comunicación entre los nodos y el servidor estará formada por tres funciones. Por un lado la primera función se encargará de establecer la conexión con el servidor para obtener un identificador válido para el nodo, esta función devolverá el identificador nodo que se utilizara para marcar los mensajes. Por otro lado la segunda función esperara recibir los datos que serán enviados al servidor, esta función montará los mensajes UDP marcados con el identificador y estructurara los datos, para que sea posible identificar el nodo y su sensor o actuador. Por último nos hará falta una función que se encargue de escuchar posibles mensajes que modifiquen el estado del nodo, de esta forma podremos enviar mensajes a un puerto para modificar partes concretas del nodo.

## 2.2 Computador

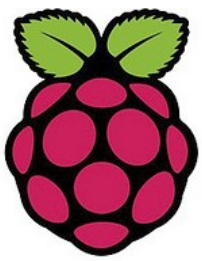


*Ilustración 5: Logo Raspberry PI*

Para ejecutar el servidor y los diferentes servicios de los que dispondremos, necesitamos un ordenador donde ejecutar los diferentes sistemas y programas. Se podría utilizar cualquier ordenador para este trabajo, pero debido a que el sistema domótico tendrá que estar en funcionamiento a lo largo de todo el día, necesitamos un computador que tenga un consumo eléctrico muy bajo. En este ordenador se ejecutaran programas como el demonio que identifica los nodos, la base de datos donde guardaremos la información, los servicios externos y la página web que utilizaremos a modo de interfaz gráfica.

En el Mercado hay toda clase de ordenadores disponibles, pero los que más llamaron mi atención fueron los denominados "ordenadores de placa reducida". Este tipo de equipos son denominados así porque todos los componentes del ordenador son integrados en la misma placa.

De entre todas las posibilidades de las que disponemos, se ha seleccionado la placa Raspberry Pi B+.

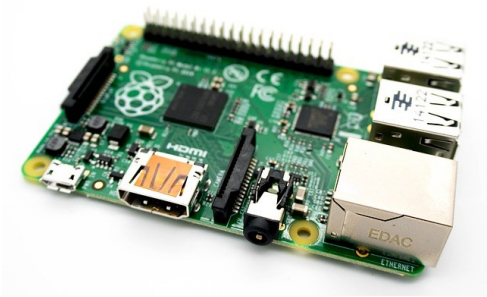


*Ilustración 6: Raspberry PI B+*

Las especificaciones técnicas de la tarjeta son:

| Características | Raspberry PI B+ |
|---|---|
| **Procesador** | Broadcom BCM2835 SoC full HD |
| **RAM** | 512 MB SDRAM 400 MHz |
| **Almacenamiento** | Ranura para tarjeta microSD |
| **USB** | 4 puertos USB 2.0 |
| **Energía** | 600mA hasta 1.8A a 5V |
| **Pines GPIO** | 40 pines |

Raspberry PI es un ordenador de bajo coste que se ajusta perfectamente a nuestras necesidades. Con el tamaño de una tarjeta de crédito y con los recursos suficientes para ejecutar nuestro proyecto, lo convierten en la mejor opción. Funciona con una alimentación de 5v, lo que es un consumo muy bajo si lo comparamos con un ordenador convencional. Esta placa fue diseña para extender el mundo de la programación entra la población, por lo que es perfecta para todas clase de experimentos, lo que deja a nuestra disposición una comunidad de desarrolladores.

Para el sistema operativo se va a utilizar una tarjeta micro SD, con una capacidad de 4GB, en la que instalaremos el sistema Raspbian. Este SO nos lo proporciona la propia organización de Raspberry, es una versión de Debian adaptada para la arquitectura ARM de la tarjeta.


*Ilustración 7: Logo de Raspbian*

El sistema operativo será clave para seleccionar el software y las herramientas que vamos a utilizar. Gracias a la utilización de usar este tipo de software, encontramos gran cantidad de programas gratuitos y procesos que modificaremos para propósitos propios del servidor. Un buen ejemplo es el uso hilo de ejecución crontab para realizar procesos de mantenimiento del servidor. Pero el más importante es un demonio que dejaremos continuamente en ejecución, este servicio abrirá uno de los puertos de nuestra tarjeta para recibir mensajes vía UDP. El proceso tratará los datos que se reciban y actuará en función del tipo de mensaje recibido.

```
Programa;
        AbrimosPuerto;
                bucle 1;
                        Si mensaje = Alta
                                Consulta a la base de datos de la IP
                                Si existe
                                        Devolvemos el id asociados
                                Si NO
                                        Recuperamos el ultimo id
                                        Enviamos (ultimo id +1)
                                FinSi
                        FinSi
                        Si mensaje = Datos
                                Se trata el mensaje para establecer relacion sensor/valor
                                Se introduce en la base de datos
                        FINSI
                finBucle
        CerramosPuerto
FinPrograma
```

## 2.3 Base de datos


*Ilustración 8: Logo MySQL*

Los datos que son enviados continuamente de los diferentes nodos, contienen información acerca del estado en el que se encuentran, esta información debe ser almacenada y para ello vamos a utilizar una base de datos. De entre todas las alternativas de libre distribución que

podríamos utilizar, se ha escogido MySQL. Los motivos de haber elegido este sistema concreto están relacionados con la interfaz web. Como la web tendrá que realizar muchas consultas rápidas hacia la base de datos, para mantener la información a tiempo real. Aunque es mucho menos robusta que por ejemplo PostgreSQL, esta optimizada para realizar consultas pequeñas y está muy ligada al desarrollo web.

La base de datos deberá estar formada por dos tablas, una de ellas contendrá información acerca de los nodos y otra guardara toda la información. Utilizamos MySQL para poder establecer claves ajenas que relacionen ambas tablas, de esta forma podremos relacionar las medidas obtenidas con el nodo que la genero.

La tabla que guarda los nodos mantendrá la información acerca de la IP y puerto de estos, y como es posible que sea muy útil relacionar varios nodos entre si, usaremos un campo para guardar un identificador de grupo. Por otro lado la tabla de datos guardara el identificador del nodo que genero los datos, así como la fecha en que se recibió.

## 2.4 Contenedor web

La selección del contenedor web fue bastante complicada, puesto que la elección de este marcaba el desarrollo de la aplicación. Entre las alternativas se encontraban Tomcat y Apache.

*Ilustración 9: Logo Tomcat*

Tomcat por un lado, es un contenedor basado en servlets,  una buena alternativa para este proyecto hubiera sido la publicación de la interfaz web con Java. Debido a las limitaciones de la Raspberry se decidió no utilizar esta alternativa, por miedo a que la máquina virtual de Java fuera demasiado para esta. Además en caso de querer realizar cambios en la web se debería de recompilar todo el código JAVA.


*Ilustración 10: Logo Apache*

Por otro lado Apache podría ser una alternativa más recomendable, ya que utilizaremos PHP para el código de la web. Como este código estará visible, podremos modificarlo en cualquier momento, y cambiar todo aquello que consideremos necesario en un futuro.

## 2.5 Otras tecnologías

*Ilustración 11: Logo Jquer* *Ilustración 12: Logo Php* *Ilustración 13: Logo Java*

Otras tecnologías que se utilizarán para el desarrollo de todas las funcionales que componen el sistema son:

Utilizaremos Java para el desarrollo del demonio que estará ejecutándose continuamente. Este creara los sockets y se encargará de comunicar la información pertinente a cada uno de los nodos para identificarse en el sistema. Hemos elegido este lenguaje en concreto por su universalidad, que nos permitirá ejecutar la aplicación en cualquier dispositivo y por qué fue diseñado para potenciar la interacción a través de internet.

Usaremos PHP para el desarrollo de la interfaz web, puesto que cuenta con una muy completa API que nos permite realizar casi cualquier acción sobre el servidor.

Gracias a JavaScript podremos ejecutar código en los navegadores cliente, de modo que dará a nuestra interfaz la apariencia de los datos y estados a tiempo real.

Por último destacar las librerías que serán utilizas para simplificar el trabajo de desarrollo, son la librería JQuey y las librerías de Arduino. La librería JQuery nos facilitara el acceso a los elementos del DOM (Document Object Modelo), para poder modificarlos, y las diferentes librerías Arduino que se han diseñado para los diferentes módulos o sensores, en este caso usaremos la librería Ethernet, la librería Servo y la DHT11.

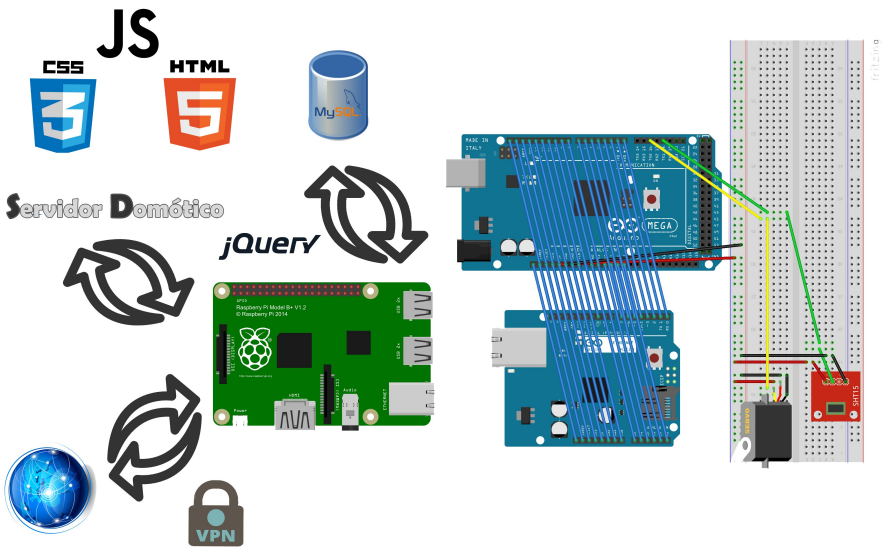


*Ilustración 14: Esquema de conexiones del proyecto.*

# 3. Presupuesto

| Material | coste(€) |
|---|---|
| Arduino Mega 2560R3 | 17,5 |
| Shield Enthernet | 11,9 |
| Rasberry Pi B+ | 40,6 |
| Tarjeta SD 4 GB | 3,4 |
| Total | 73,4 |

# 4. Diseño

En este apartado vamos a centrarnos en la estructura de los elementos que conformaran nuestro sistema, estas estructuras marcarán las pautas que se seguirán a la hora de programar la comunicación entre los nodos y el servidor, la base de datos y la web.

## 4.1 Protocolo de comunicación

Para la comunicación a través de la red se ha elegido el protocolo UDP (User Datagram Protocol), que parece una mejor elección que TCP para este proyecto. La razón de haber elegido este protocolo es por cuestiones de sobre saturación de la red, puesto que como TCP tiene que establecer una sesión antes de empezar a enviar mensajes puede generar una latencia muy alta en nuestra red. Un punto negativo que tiene este protocolo es la baja fiabilidad de entrega de los paquetes, pero en este sistema, la pérdida de paquetes puntuales no genera problemas de funcionamiento.

El protocolo de comunicación se ha dividido en fases para simplificar la identificación de los nodos, dentro de la red, así como de enviar la información acerca del estado del nodo. Para la identificación se ha diseñado un pequeño protocolo de “handshake”, que permitirá al controlador obtener un identificador único dentro del sistema para poder marcar los mensajes que contendrán los registros del estado del sistema.

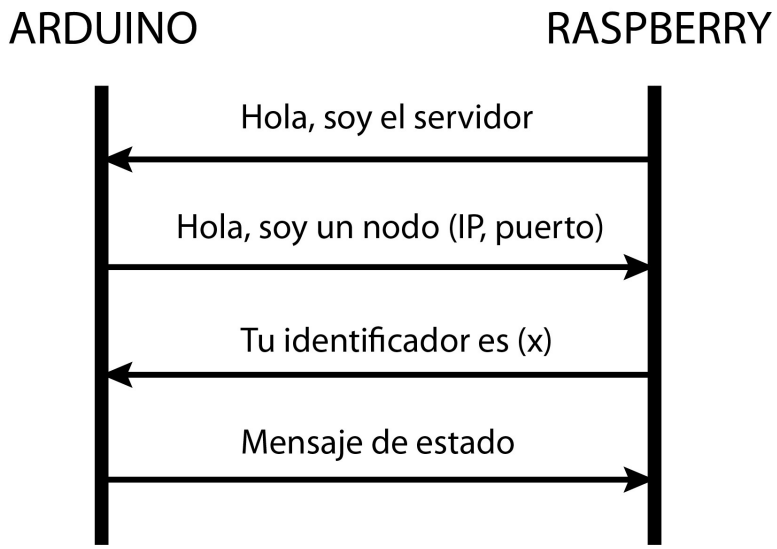



*Ilustración 15: Esquema de comunicación del protocolo*

El servidor será el encargado de enviar mensajes de invitación a través de la red, para que los nodos puedan descubrir cuál es la dirección donde está situado el servidor. Este mensaje será enviado por el servidor a intervalos de tiempos para poder mantener el servicio actualizado. Los nodos esperan recibir este mensaje, cuando reciben este mensaje envían un mensaje al servidor indicándole que quiere identificarse dentro de la red. Cuando el servidor recibe un mensaje de identificación, el servidor tendrá que comprobar si ese nodo ya existía dentro del sistema o es

necesario darle de alta. Una vez el nodo obtiene el identificador, ya puede empezar a realizar los envíos necesarios o escuchar un puerto determinado.

## 4.2 Base de datos

Para el almacenamiento del estado de los nodos vamos a utilizar una estructura de datos bastante sencilla, utilizando para ello la base de datos MySQL. Como se trata de una base de datos relacional, guardara la relación entre las diferentes tablas que guardaran toda la información. La estructura que se ha diseñado es la siguiente:

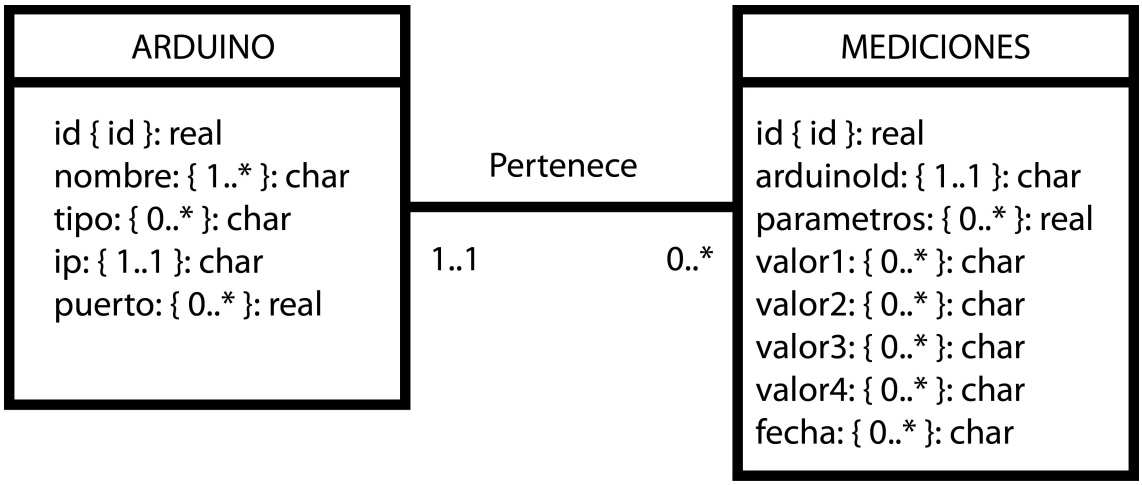



*Ilustración 16: Esquema de la base de datos*

Finalmente se ha optado por dividir la información en dos tablas relacionadas, Arduino y mediciones. Por un lado la tabla Arduino se encargará de guardar la información relativa al nodo, por otro lado la tabla mediciones guardará los datos enviados por los nodos. Estas dos tablas estarán relaciona a través del campo id de la tabla Arduino y el campo arduinoId de mediciones, quedando el siguiente esquema relacional.

Arduino (id:real, nombre:char(20), tipo:char(20), ip:char(20),puerto:real)

CP:{id}

Mediciones (id:real, arduinoId:char(20), parametros:real, valor1:char(20), valor2:char, valor3:char(20), valor4:char(20), fecha:date)

CP:{id}

CAj:{ arduinoId } -> Arduino{id}

Los campos elegidos para la tabla Arduino son:

- Id: identificador que se le dará a cada uno de los nodos, se utilizará para marcar la información que se envíe a la red, por lo que tiene que ser único para cada nodo.
- Nombre: es un valor auxiliar que se utilizará para relacionar diferentes nodos. Si por ejemplo tenemos varios nodos en un mismo lugar, podemos utilizar este campo para establecer una relación entre ellos asociándoles el mismo valor.
- Tipo: existen dos tipos de nodos y por lo tanto dos posibles valores, sensor o actuador.
- Ip: guardara la IP del nodo.
- Puerto: guardara el puerto del nodo.

Los campos que se utilizan para la tabla Mediciones son:

- Id: es el identificador de la entrada dentro de la tabla.
- ArduinoId: este campo se utilizará para relacionar las dos tablas, de este modo podremos identificar los datos asociándolos a un nodo.
- Parámetros: es un valor numérico que nos indica cuantos datos se están enviando al servidor. En este diseño se ha optado por limitarlo a 4 puestos que es el número de puertos serie de los que dispone nuestra Arduino Mega.
- Valor1, Valor2, Valor3, Valor4: son los campos que se utilizan para almacenar la información recibida de la red, cada uno de los campos corresponde a un sensor o actuador diferente.
- Fecha: este campo guardará la fecha y hora en la que se recibió el mensaje.

## 4.3 Estructura web

Para la estructura de la web hemos decido seguir una estructura por bloques. La página estará dividida en tres zonas diferenciadas, el menú, el contenedor y la lista.

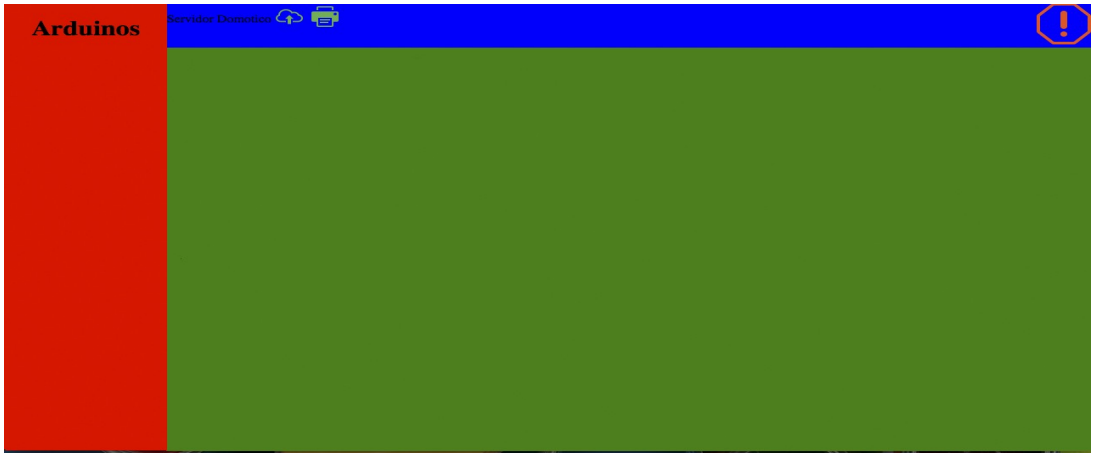



*Ilustración 17: Esquema de la página web*

La parte roja de la página corresponde a una lista de los identificadores que se utilizan para identificar los nodos. Los elementos de esta lista serán clicables y nos permitirán mostrar en la

página solo aquellos nodos que compartan este identificador. De este modo podremos tener un acceso rápido a todos aquellos nodos que se encuentran por ejemplo en la misma habitación.

La parte azul es el menú de nuestra interfaz en la que encontraremos todos aquellos servicios que instalaremos en nuestro servidor, de este modo conseguiremos un acceso centralizado a todos los servicios del servidor. En este apartado además añadiremos el aviso visual naranja, que actuará de sistema de aviso cuando un nuevo nodo se conecte a la red. Cuando se pulse nos permitirá terminar de dar de alta al nodo en nuestro sistema, rellenando los datos que no se han completado por el demonio.

Por último la parte verde será el contenedor, en esta parte iremos añadiendo los datos que nos enviarán los nodos. Cada nodo tendrá una entrada en la que se publicarán los datos que la Arduino envíe en tiempo real, si el nodo dispone de algún actuador, aparecerá un pequeño formulario que permitirá enviar datos al nodo.

Utilizaremos eventos temporales mediante javascript y el crontab del servidor, para ejecutar el sistema de alertas y avisos que dotarán al sistema de datos en tiempo real.

# 5. Implementación

## 5.1 Preparación del entorno

Para comenzar con la implementación del sistema, debemos preparar el entorno donde ejecutaremos el servidor. Empezaremos preparado la Raspberry, una vez que tengamos el sistema operativo listo, podremos empezar a instalar las herramientas necesarias para ejecutar la aplicación.

En la página web oficial de Ardruino podemos encontrar una gran variedad de sistemas operativos, pero el que realmente nos interesa es Raspbian. Este sistema es una versión adaptada de Debian que ha sido optimizado para Raspberry, cuenta con los programas básicos que se pueden ejecutar en nuestra tarjeta. Nosotros vamos a utilizar la versión Wheezy, y utilizaremos la aplicación Apple-Pi-Baker para montar la imagen con el sistema en nuestra tarjeta SD.

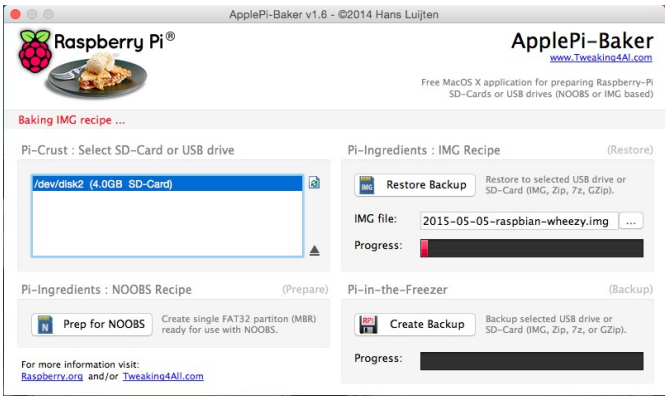



*Ilustración 18: Captura de pantalla ApplePi-Baker*

Como se puede ver en la imagen superior, la interfaz del programa utilizado para instalar el SO es bastante sencilla. Apple-Pi-Baker es un programa gratuito que nos permite realizar instalaciones de sistemas operativos, realizar y recuperar copias de seguridad y la utilización de NOOBS en caso de que queramos tener instalados varios sistemas operativos al mismo tiempo. Cuando el programa termine, podremos colocar la tarjeta de memoria en nuestra Raspberry y conectarla tanto a la alimentación como a la red.

Podemos utilizar nuestro router para acceder a su tabla ARP o podemos utilizar el comando arp si disponemos de un SO con kernel basado en linux, para descubrir cuál es la IP de nuestra Raspberry. En nuestro caso la IP que le ha dado el servidor de DHCP ha sido la 192.168.1.16, de modo que vamos a conectarnos a ella a través del protocolo de SSH. Este protocolo nos permite acceder y controlar máquinas de modo remoto a través de una red.

**ssh pi@192.168.1.16**

Con este comando podremos acceder a nuestra Raspberry con el usuario “pi”, cuando nos pregunte la contraseña solo tenemos que introducir la password por defecto que tiene el SO, “raspberry”. Una vez dentro del sistema podemos empezar a instalar todo lo que necesitemos.

Antes de empezar a instalar software en nuestra Raspberry, vamos a configurar algunos ficheros que pueden resultar útiles en un futuro. Para configurar nuestra Raspberry vasta con ejecutar el comando:

**sudo raspi-config**

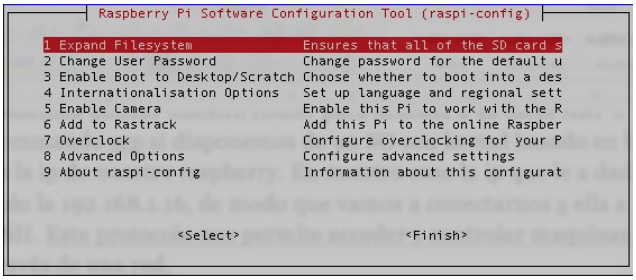



*Ilustración 19: Captura del menú de configuración Raspbian*

En este menú encontramos todas las opciones de configuración a las que tenemos acceso, en nuestro sistema. Vamos a destacar las más importantes:

- Expand Filesystem nos permitirá aumentar el tamaño del sistema de archivos, esto es muy importante si estamos utilizando tarjetas micro SD de más de 2GB, ya que por defecto en el SO viene limitado a dos 2GB.
- Enable Boot to Desktop/Scratch permite cambiar el tipo de arranque del sistema operativo, podemos elegir que arranque en modo consola o modo gráfico directamente.
- Internationalisation Options estas son quizás las opciones más importantes de la configuración, en ellas se debe configurar la codificación de caracteres que se va a utilizar así como la zona horaria. En este caso se ha utilizado una codificación para el teclado “spanish”, se ha seleccionado la zona horaria europea en Madrid y se ha cambiado el idioma a “en_esp_UTF-8”.

Para terminar la configuración vamos a cambiar el archivo config.txt para cambiar los formatos de salida del audio y el vídeo de nuestro Raspberry para adaptarlo a los estándares. Utilizando este comando se obtiene acceso al fichero:

**sudo nano /boot/config.txt**

Una vez dentro del fichero vamos a añadir unas líneas para configurar la salida de vídeo de nuestra tarjeta, por si en algún momento es interesante pasar al modo gráfico:

- Sdtv_mode=2 seleccionaremos como salida analógica el formato PAL. PAL es el tipo de video de codificación de vídeo que se usa en España.
- hdmi_group=2, en este parámetro estamos configurando el tipo de monitor que se va a utilizar. Los posibles valores son 1, en caso de que estemos utilizando una televisión ya que corresponde con el formato CEA o 2 DMT, que es un formato para monitores.

- hdmi_mode=16, por último este parámetro hace referencia a la resolución de salida del vídeo, una buena forma de saber qué alternativas de resolución se disponen es usar el comando:

**/opt/vc/bin/tvservice –m TIPO_DE_MONITOR**

Una vez todo configurado ya podemos comenzar instalando el Apache y PHP en el nuestra tarjeta, de modo que primero de todo necesitamos dar permisos dentro del sistema al grupo que usa apache por defecto.

**Sudo addgroup www-data**

**scudo usermod -a -G www-data www-data**

Una vez creado el grupo, vamos a actualizar nuestro sistema y después instalaremos los paquetes de Apche y PHP.

**sudo apt-get update**

**sudo apt-get install apache2 php5 libapache2-mod-php5**

A continuación vamos a instalar la máquina virtual de java para poder ejecutar programas en JAVA. Oracle tiene una versión del JDK para arquitecturas ARM, en su web podemos encontrar el enlace a la versión más actualizada. Con el siguiente comando podemos descargarnos esa versión.

**wget http://download.oracle.com/otn-pub/java/jdk/8u51-b16/jdk-8u51-linux-arm-vfp-hflt.tar.gz**

Una vez descargado sólo tenemos que moverlo al directorio que le corresponde y descomprimirlo.

**sudo mv jdk-8u51-linux-arm-vfp-hflt.tar.gz /usr/local/**

**cd /usr/local/**

**sudo tar -xzvf jdk-8u51-linux-arm-vfp-hflt.tar.gz**

Por último lo añadiremos al path para que sea ejecutable desde cualquier lugar mediante este comando.

**PATH=/usr/local/jdk1.8.0_51/bin:$PATH**

**export PATH=/usr/local/jdk1.8.0_51/bin:$PATH**

## 5.2 Instalación de MySQL

Es hora de instalar la base de datos, además instalaremos la aplicación PhpMyAdmin para facilitar el uso de esta. PhpMyAdmin es una herramienta escrita en PHP de libre distribución, está diseñada para la administración de bases de datos MySQL a través de la web. Antes de

empezar la instalación debemos de activar la interfaz loopback de la Raspberry, para evitar problemas de instalación de Mysql. Podemos activarla con el siguiente comando:

**sudo ifup lo**

Para instalarlo todo en nuestra Raspberry deberemos de ejecutar el siguiente comando en el terminal.

**sudo apt-get install mysql-server mysql-client php5-mysql phpmyadmin**

Durante la instalación se nos preguntará qué contraseña queremos utilizar para acceder a nuestra base de datos, en este caso se ha introducido la misma contraseña para todo con el fin de simplificar un poco la implementación del sistema. Una vez esté instalado debemos de modificar el archivo de php.ini:

**sudo nano /etc/php5/apache2/php.ini**

Buscamos el aparatado “Dynamics Extensions”, y añadimos la siguiente línea:

**extension=mysql.so**

Con esto daremos de alta la extensión de mysql para php. Por último crearemos un enlace simbólico para poder acceder a PhpMyAdmin.

**sudo ln -s /etc/phpmyadmin/apache.conf /etc/apache2/conf.d/phpmyadmin.conf**

**sudo /etc/init.d/apache2 reload**

Una vez hecho esto, podemos acceder al panel de administración a través de la url, 192.168.1.16/phpmyadmin. Nos aparecerá un panel de registro, una vez dentro tendremos todo a nuestra disposición para generar las tablas que utilizaremos en nuestro servidor.


*Ilustración 20: Captura de la página de registro phpMyAdmin*

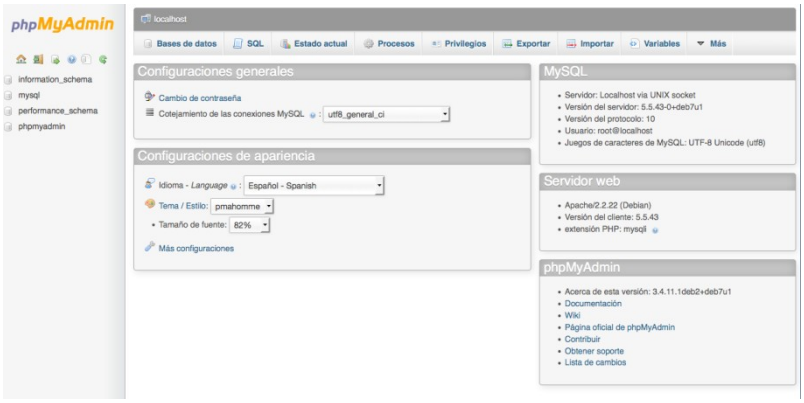



*Ilustración 21: Captura de la página de inicio phpMyAdmin*

Dentro del apartado base de datos podemos crear la base de datos que contendrán las tablas que guardarán la información. Una vez creada la base datos podremos acceder a ella y crear la tabla, a través de un sencillo formulario, la interfaz nos permite crear las tablas que se han mostrado en el punto anterior. Destacar que cuando elijamos el tipo de cotejamiento de los datos,

seleccionar el utf8_unicode_ci, esto evitara que tengamos problemas cuando se inserten o se busquen los datos.

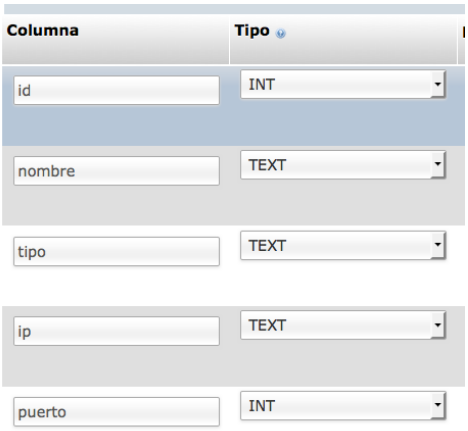



*Ilustración 22: Captura configuración base de datos*

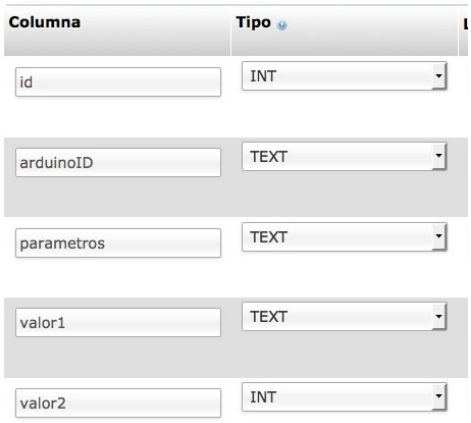



*Ilustración 23: Captura configuración base de datos*

## 5.3 VPN

Ahora vamos a montar un servicio de VPN (Virtual Private Network), que nos permitirá acceder a nuestro servidor desde cualquier punto del mundo. En el servidor necesitaremos tener instalando un servicio de NO-IP para dar soporte a la VPN.

El servicio NO-IP nos proporcionará todo lo necesario para que nuestra Raspberry sea visible a través de internet, sin tener que tener una dirección pública fija. Es un demonio que se puede descargar en la propia web del servicio, es gratuito y se encarga de actualizar el servidor de DNS. El DNS se actualiza continuamente con la información de la dirección actual del equipo, por lo que aunque se reinicie el router siempre sabrá donde esta nuestra Raspberry. Para instalarlo solo tenemos que acceder a la página web de www.no-ip.com y registrarnos. Esta página nos permite utilizar una gran cantidad de dominios y asignárselos a los host que queramos.

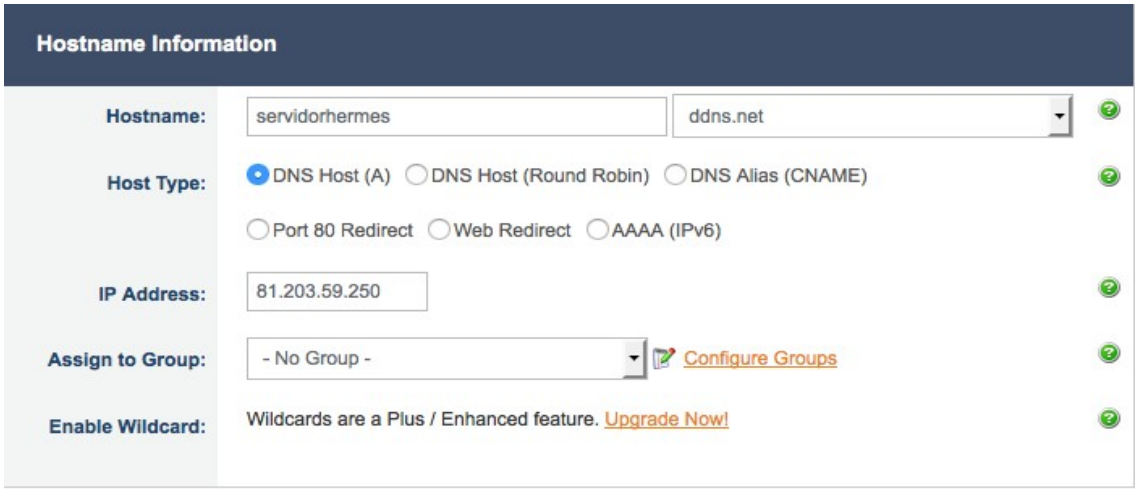
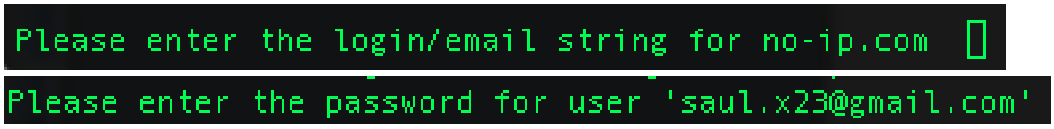



*Ilustración 24: Captura del menú de NoIP*

Para este proyecto vamos a utilizar un dominio de la sección gratuita que nos ofrece la web y un nombre de dominio, en este caso el dominio sería “servidorhermes.ddns.net”. Para instalar el servicio que actualizará el router para re direccionar el tráfico hacia nuestra Arduino vamos a ejecutar los siguientes comandos:

**mkdir noip**
**cd noip/**
**wget** http://www.no-ip.com/client/linux/noip-duc-linux.tar.gz
**tar -zxf noip-duc-linux.tar.gz**
**cd noip-2.1.9-1/**
**make**
**sudo make install**

Primero de todo vamos a crear una carpeta donde descargar el archivo comprimido de la web oficial de no-ip. Una vez descomprimido este archivo ya tenemos los códigos de la aplicación, solo tenemos que compilar el código para que se generen todos los archivos necesarios para la instalación del servicio. Una vez todo compilado ya podemos instalar el demonio, durante la instalación se nos pide que introduzcamos los datos de la cuenta donde hemos creado el host.




Ahora vamos a crear un fichero para que el servicio se levante cada vez que reiniciamos nuestra tarjeta:

**sudo vim /etc/init.d/noip2**

Con el siguiente script y los comandos que le dan al fichero los permisos necesarios para levantar el servicio, ya tenemos todo listo para re direccionar el dominio a nuestra Raspberry.

```
#! /bin/sh
# /etc/init.d/noip
case "$1" in
  start)
    echo "Arrancando noip"
    /usr/local/bin/noip2
    ;;
  stop)
    echo "Parando noip"
    killall noip2
    ;;
  *)
    echo "Uso: /etc/init.d/noip {start|stop}"
    exit 1
    ;;
esac

exit 0
```

**sudo chmod +x /etc/init.d/noip2**

**sudo update-rc.d noip2 defaults**

Ahora vamos a configurar nuestra VPN utilizando para ello PPTP (Point to Point Tunnelling Protocol). Este es un protocolo que permite establecer conexiones seguras entre dos puntos cifrando para ello los datos antes de ser enviados. Para instalarla primero de todo vamos a instalar el servidor de PPTP a través del comando:

**sudo apt-get install pptpd**

Una vez instalado el programa se ha de configurar, comenzaremos por asignar la IP local de nuestra Raspberry y el rango de IPs que vamos a distribuir cuando alguien se conecte a la VPN. Sólo tenemos que acceder al fichero con el siguiente comando y descomentar y configurar los campos “localip” y ”remoteip” mediante el comando:

**sudo vim /etc/pptpd.conf**

Ahora vamos a añadir las siguientes instrucciones dentro del fichero pptpd-options para configurar los datos de red, en este caso se ha utilizado el servidor DNS de “google”, 8.8.8.8.

**sudo nano /etc/ppp/pptpd-options**

Y añadimos las líneas:

```
ms-dns 8.8.8.8
nobsdcomp
noipx
mtu 1490
mru 1490
```

Por último podemos crear un usuario para acceder a la VPN, en el fichero chap-secret que se encuentra en la ruta:

**/etc/ppp/**

Tras reiniciar el servicio y abrir el puerto 1723 de TCP/IP en nuestro router, ya tendremos listo un acceso privado a nuestro servidor. Por temas de seguridad, cuando un usuario acceda a nuestra red, solo tendría acceso a nuestra tarjeta. De este modo nos aseguramos de que en caso de una intrusión, no se vea corrompida la red. Como se trata de un servidor que se ejecuta localmente, no corre peligro de ser atacado, la VPN es el elemento que conecta nuestra red a través de internet y necesita de credenciales de usuario para poder conectarse. Además destacar que la comunicación que se realiza entre los dos puntos se realiza de forma cifrada, lo que puede evitar que suframos un ataque de “Man-in-the-middle”.

## 5.4 Servicios adicionales

En un proyecto como este que busca la conexión de objetos comunes a nuestra red, no podíamos dejar de hablar del servidor de impresión. Vamos a utilizar un servicio web llamado CUPS (Common Unix Printing System) que nos permitirá subir ficheros al servidor para que sean imprimidos por una impresora, la idea es utilizar una impresora USB convencional como una impresora en red moderna. Empezaremos instalando el repositorio mediante el siguiente comando:

**sudo apt-get install cups**

Una vez se instale la aplicación debemos darle permisos, de este modo podrá administrar las impresoras y enviarles los documentos a imprimir.

**sudo usermod -a -G lpadmin pi**

El programa por defecto solo deja que las impresoras sean administradas de forma local, pero si modificamos el fichero de configuración del programa, podemos acceder al fichero con el siguiente comando:

**sudo nano /etc/cups/cupsd.conf**

Una vez dentro del fichero podemos empezar modificando algunos valores para que queden de la siguiente manera:

```
# Only listen for connections from the local machine.
Listen *:631
Listen /var/run/cups/cups.sock


# Restrict access to the server...
<Location />
  Order allow,deny
 Allow 192.168.1.*
</Location>

# Restrict access to the admin pages...
<Location /admin>
  Order allow,deny
 Allow 192.168.1.*
</Location>

# Restrict access to configuration files...
<Location /admin/conf>
  AuthType Default
 # Require user @SYSTEM
 Allow 192.168.1.*
  Order allow,deny
</Location>
```

Una vez modificado el fichero, ya podemos acceder a nuestro servidor de impresión desde cualquier punto de la red, solo tenemos que utilizar la url 192.168.1.16:631.


*Ilustración 25: Captura de pantalla Cups*

Ya sólo nos queda configurar nuestra impresora, solo tenemos que conectarla a nuestra Raspberry a través del puerto USB y configurarla en el apartado impresoras del menú de CUPS.

Ahora vamos a instalar Owncloud, que es un servicio que nos proporcionará nuestra propia nube personal. Gracias a esto podremos almacenar en nuestro servidor toda la información que queramos, vídeos, música, imágenes…
Owncloud es una aplicación web basada en PHP y de libre distribución, es una herramienta muy útil ya que es compatible con la mayoría de las bases de datos para facilitar su implantación. Esta aplicación no solo nos proporciona un almacenamiento seguro de nuestros datos, también podemos encontrar en su web gran cantidad de plugins que facilitan la experiencia del usuario, como un reproductor multimedia online, un calendario de eventos, un visor de archivos o la posibilidad de administrar usuarios y grupos. Para poder instalar la aplicación, antes debemos de instalar una serie de librerías necesarias para ejecutarla, mediante el comando:

**sudo apt-get install php5-json php-xml-parser php5-gd curl libcurl3 libcurl3-dev**

Cuando se instalen todas las librerías, solo debemos de movernos hasta el directorio /var/www y descargar el archivo comprimido de la aplicación que podemos encontrar en la página web oficial. Una vez descomprimido sólo tenemos que configurarlo, los comandos utilizados son:

**cd /var/www**
**sudo wget download.owncloud.org/community/owncloud-5.0.0.tar.bz2**
**tar -xjf owncloud-5.0.0.tar.bz2**
**sudo chown -R www-data:www-data /var/www**

Para configurarlo debemos de acceder al archivo de configuración de PHP y modificar el tamaño máximo de archivo que se puede subir al servidor, de este modo eliminaremos la restricción de 8M que viene por defecto en la instalación de PHP. Para llevar a cabo esta configuración debemos cambiar el parámetro siguiente:

**sudo nano /etc/php5/apache2/php.ini**

**; Maximum size of POST data that PHP will accept.**
**; Its value may be 0 to disable the limit. It is ignored if POST data reading**
**; is disabled through enable_post_data_reading.**
**; http://php.net/post-max-size**
**post_max_size = 100000M**

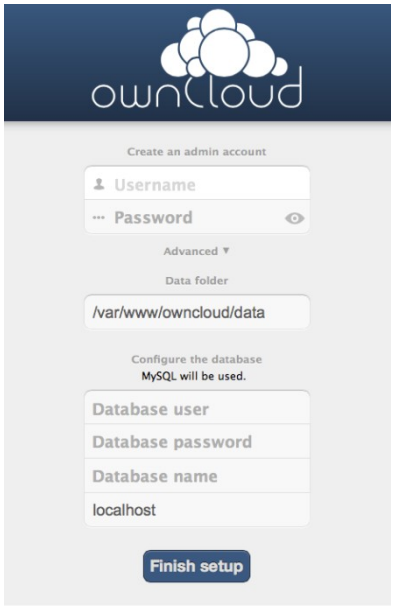



*Ilustración 26: Captura pantalla de registro Owncloud*

Podremos acceder a nuestra nube a través de la url 192.168.1.16/owncloud, la primera vez que accedamos al servicio tendremos que configurar la base de datos y el usuario que vamos a utilizar para administrar y configurar el servicio. Primero deberemos crear una base de datos nueva con PhpMyAdmin, luego podremos usar la interfaz del owncloud para utilizarla para guardar la información acerca de los usuarios. El espacio de almacenamiento en el sistema es algo que nos preocupa, pero este sistema nos permite en el apartado de configuración ampliar el tamaño de nuestra nube utilizando todo clase de dispositivos, en nuestro caso vamos a utilizar un servidor NAS que tenemos en la red y que será perfecto para almacenar nuestros datos.

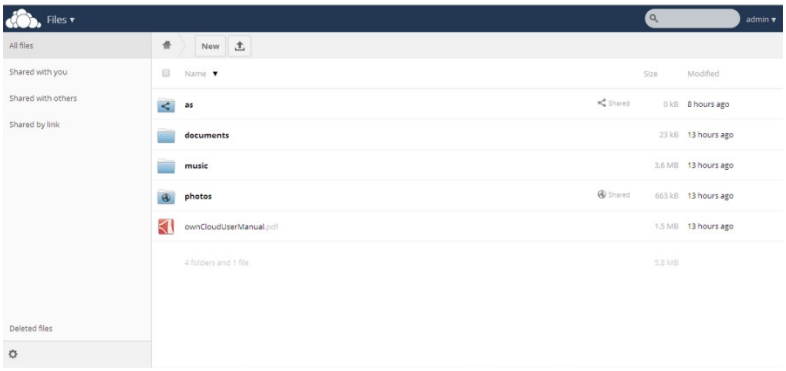



*Ilustración 27: Captura pantalla principal Owncloud*

Otra aplicación muy interesante y que nos será útil para el sistema de alertas y avisos es Mutt. Mutt es un cliente de correo electrónico, para sistemas UNIX y de libre distribución, que funciona a través de comandos introducidos por consola. Esto nos será muy útil para crear un sistema de alertas que podremos utilizar para obtener correos regularmente, con aquella información que creamos pertinente, en este caso avisaremos de las nuevas incorporaciones al servidor y avisar en caso de que se supere cierta temperatura. Para instalarlo basta con ejecutar el siguiente comando y crear el fichero de configuración:

**sudo apt-get install mutt**
**sudo nano /root/.muttrc**

Una vez dentro del fichero debemos de añadir los siguientes parámetros :

- set from = nuestroMail@gmail.com
- set realname = "Identificador de remitente"
- set imap_user = nuestroMail@gmail.com

- set imap_pass = "password"
- set folder = "imaps://imap.gmail.com:993"
- set spoolfile = "+INBOX"
- set postponed ="+[Gmail]/Drafts"
- set header_cache =~/.mutt/cache/headers
- set message_cachedir =~/.mutt/cache/bodies
- set certificate_file =~/.mutt/certificates
- set smtp_url = "smtp:// nuestroMail@smtp.gmail.com:587/"
- set smtp_pass = "password"

De este modo podemos utilizar nuestra cuenta de correo de “Gmail” para enviar correos desde nuestra Raspberry. Solo tenemos que introducir nuestros datos de configuración y podremos enviar correos electrónicos desde el terminal de Linux con el siguiente comando:

**echo "Mensaje" | mutt -s "Asunto" cuentaDestino@gmail.com**

Para terminar vamos a instalar la aplicación Motion, este software gratuito permite la visualización y configuración de webcams USB a través de la red. Este programa es compatible con una gran variedad de modelos de webcam, es compatible tanto con las base de datos MySQL como PostgreSQL aunque en este trabajo no se va a utilizar dicha configuración. El programa funciona realizando capturas de pantallas a través de la webcam y publicándolas en el puerto que queremos, una vez conectado el servicio solo debemos utilizar una etiqueta “<img>” escuchando el puerto configurado para ver a tiempo real lo que está sucediendo. Para instalarlo solo tenemos que conectar la webcam al USB de nuestra Raspberry y ejecutar los siguientes comandos:

**sudo apt-get upgrade**
**sudo apt-get install motion**

Una vez instalada la nueva aplicación sólo tenemos que configurarla para que se ajuste a nuestro sistema. Para ello podemos usar el comando siguiente para acceder al fichero de configuración de la aplicación.

**sudo nano /etc/motion/motion.conf**

Dentro de este archivo tenemos que cambiar los siguientes parámetros:

- Debemos de activar el demonio con el parámetro Daemon on
- Snapshot_interval nos permite seleccionar cada cuanto tiempo se realizan las capturas de pantallas por parte del programa, para este proyecto se ha elegido que se tome una imagen cada segundo (snapshot_interval 1).
- Webcam_port aquí podemos seleccionar el puerto que queremos para mostrar las imágenes.
- Webcam_localhost off, es muy importante deshabilitar esta opción para poder acceder a servicio desde cualquier punto de la red.
- Control_port a diferencia del anterior, este puerto nos dará acceso a la interfaz de control del programa.
- Control_localhost off, deshabilitando esta opción podremos configurar de forma remota el servicio.

Cuando lo tengamos todo instalado y configurado solo tenemos que lanzar el servicio, como es posible que encendamos y apaguemos la Raspberry, vamos a asegurarnos de que cada vez que encendamos nuestra tarjeta se levante el servicio automáticamente. Para realizar esta tarea

vamos a modificar el archivo rc.local y añadiremos la instrucción que levanta el servicio. Podemos acceder al fichero con la instrucción:

**sudo nano /etc/rc.local**

Al final del archivo deberemos de añadir la siguiente instrucción:

**sudo motion –n**

## 5.5 Código Arduino

Para la programación de todos los códigos que se utilizan dentro de los microcontroladores Arduino vamos a utilizar el IDE oficial, que podemos encontrar en la página web de la organización. Es un programa muy sencillo y ligero que nos facilita la compilación de los códigos y la corrección de nuestros códigos.

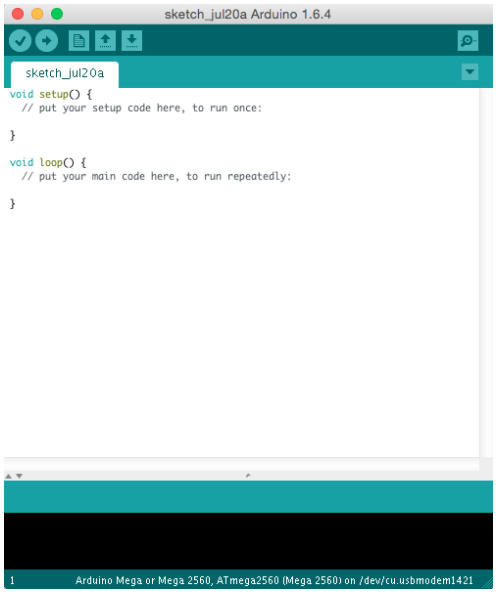



*Ilustración 28: Captura IDE Arduino*

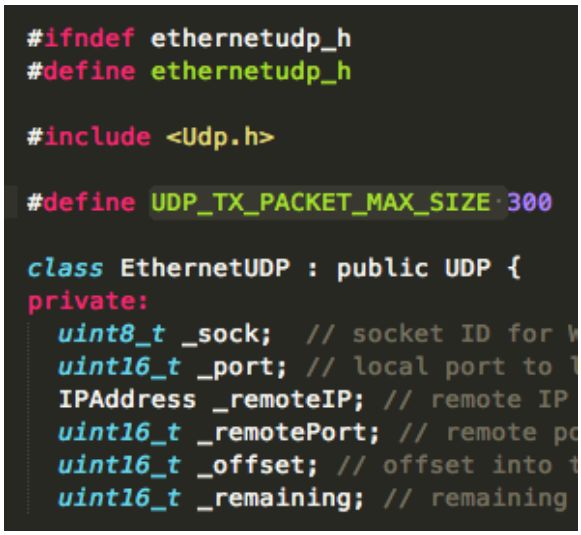



*Ilustración 29: Captura de la librería Ethernet*

Antes de empezar a programar tenemos que hacer un cambio dentro de las librerías que vienen por defecto en este programa. Debemos de modificar el tamaño máximo de mensaje que permite enviar esta librería dentro de un mensaje UDP. Por defecto el tamaño está limitado a 24 bytes, una vez cambiando a 300 tendremos un tamaño lo suficientemente grande como para enviar los mensajes que sean necesarios desde nuestra Arduino. El fichero de cabeceras que vamos a modificar se encuentra en el directorio de instalación del programa, dentro de la siguiente ruta:

**Arduino/Contents/Resources/Java/libraries/Ethernet/EthernetUdp.h**

Esta librería que es compatible con la shield Ethernet que estamos utilizando, para dar soporte a nuestra Arduino. Esta librería complementa la básica Ethernet y nos permite comunicarnos con la shield de expansión y utilizar la comunicación UDP. Ethernet nos permite asignar a nuestra

Arduino las direcciones IP y MAC, cuando se conecte a la red, el router utilizará esta dirección para enviar el tráfico. Por otro lado EthernetUdp nos permite escuchar el puerto que queramos para recibir mensajería UDP.

Además de esta librería también vamos a utilizar las librerías DHT y Servo. La librería Servo viene incluida entre las librerías del programa, esta librería nos permite configurar un servomotor en uno de los pines de nuestra tarjeta y controlarlo. Cabe destacar la función "write(Int valor)", que nos permite modificar el ángulo de nuestro servomotor. La librería DHT es una librería externa, desarrollada para el módulo DHT11 que usaremos en este proyecto. Esta librería nos la proporciona el propio distribuidor que nos vendió el módulo DHT11, este módulo toma medidas sobre la temperatura y la humedad local. Vale la pena destacar las funciones "readHuminity()" y "readTemperature()", las cuales nos devuelven un string con la humedad y la temperatura que está registrando el modulo.

El código se ha dividido lo máximo posible en funciones, éstas realizan tareas concretas como la identificación o el envío de información. Se ha tomado la decisión de usar funciones para desarrollar una librería, de este modo si en un futuro se necesita desarrollar nuevas funciones, solo sería necesario añadirla al fichero.

Dentro de la función "setup()" , que es el código que se ejecuta en primer lugar en nuestra Arduino, vamos a declarar todas las variables que serán utilizadas a lo largo de todo el programa. Esta función solo se ejecuta una vez cuando nuestra tarjeta es alimentada, en nuestro caso :

```
void setup()
{
        myservo.attach(9);
        Ethernet.begin(mac,ip);
        Udp.begin(localPort);
        Udp2.begin(localPort2);
        Serial.begin(9600);
        id = procesoRegistro();
}
```

En esta función declaramos todos los elementos que se utilizan en este proyecto, en orden sería la declaración servomotor, la asignación de IP y la MAC a la shield, escuchar dos puertos UDP, escuchar el puerto serie y asignar a la variable "id" la respuesta de la función procesoRegistro. Esta función devuelve el valor que el servidor utiliza para identificar la dirección IP, una vez que tengamos este valor ya podemos empezar a enviar mensajes añadiéndoles este identificador.

```
String procesoRegistro()
{
 while(true)
 {
  int packetSize = Udp.parsePacket();
  if(packetSize)
  {
   IPAddress remote = Udp.remoteIP();
   for (int i =0; i < 4; i++)
   {
   ipServidor=Udp.remoteIP();
   puertoServidor=Udp.remotePort();
```

```
    Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE);
    if (strcmp(packetBuffer,respuestaReconocimiento) == 0)
    {
     msg="iniSess/"+String(tipo);
     Udp.beginPacket(Udp.remoteIP(),Udp.remotePort());
     msg.toCharArray(envio,UDP_TX_PACKET_MAX_SIZE);
     Udp.write(envío);
     Udp.endPacket();
    }
    int separador=String(packetBuffer).indexOf("/");
    String res="";
    res = String(packetBuffer).substring(0,separador);
    if (res=="ident")
    {
     Udp.flush();

     return String(packetBuffer).substring(separador+1);
    }

   }
  }
}
```

Lo que hace esta función es comunicarse con el servidor para que la tarjeta Arduino pueda darse de alta dentro del servidor. Para ello la función introduce la tarjeta en un bucle infinito esperando recibir un mensaje de descubrimiento, este mensaje se envía desde el servidor. Una vez que la tarjeta recibe un mensaje en el puerto que está escuchando, automáticamente le envía un mensaje con el prefijo "iniSess" que indica al servidor que se trata de un microcontrolador que quiere darse de alta en el servicio. La tarjeta tras enviar este mensaje sigue escuchando el puerto hasta recibir la respuesta del servidor respecto al proceso de registro. Para evitar que nuestro microcontrolador reaccione ante otro tipo de mensajes que el espera, la Arduino solo reacciona ante los mensajes con el identificativo "ident" que contendrá la información que esperamos. Los mensajes que se reciben o se envían desde el servidor a los nodos y viceversa, tiene la siguiente estructura:

**Identificativo/valor**

Sólo tenemos que separar el mensaje por el delimitador "/" que estamos usando, y en función de los datos coger o no el valor. Cuando la tarjeta tiene este valor, sale del bucle y ya está lista para enviar datos. Otras funciones que se han desarrollado para el sistema son "enviarDatos(int numDat, String array [])" y "recibirDatos()".

La función enviarDatos se encarga de ejecutar todas las instrucciones necesarias para enviar el mensaje que le pasamos como parámetro y el número de sensores o actuadores de los que se dispone.

```
void enviarDatos(int numDat, String array [])
{
 String dat="";
    for (int i=0; i <= numDat - 1;i++)
    {
     dat +="/"+array[i];
```

```
    }
    msg="data/"+id+"/"+numDat+""+dat;
    Udp.beginPacket(ipServidor,puertoServidor);
    msg.toCharArray(envio,UDP_TX_PACKET_MAX_SIZE);
    Udp.write(envío);
    Udp.endPacket();

}
```

Esta función recorre el array con los datos que le hemos pasado en función al número de datos que también le pasamos. De esta forma podemos crear un String con el parámetro que ha recibido, una vez listo podemos reutilizar el puerto que teníamos creado para identificar el nodo para enviar los datos. En el parámetro numDat le pasaremos a la función el número de sensores y actuadores total de los que disponemos en el nodo. En el array vamos a introducir una serie de String que contiene la información de cada uno de los sensores o actuadores. Por ejemplo en nuestro caso disponemos de un sensor de temperatura, uno de humedad y un servomotor, por lo que nuestra array tendría la siguiente estructura:

```
String myStrings[] =
{
        "Temperatura:"+String(dht.readTemperature()),
        "Humedad:"+String(dht.readHumidity()),
        "Servomotor:input:8887"
};
```

Cada uno de los elementos de los que disponemos tiene una entrada en esta array, como se puede ver todos los elementos siguen la misma la estructura. La estructura será utilizada por el servidor web para darle coherencia a los datos que se van a mostrar de modo que para simplificarla hemos utilizado el separador “:” para separar los valores del tipo de medida. La estructura varía en función de si se trata de un actuador o un sensor.

Si se trata de un sensor la estructura del mensaje a utilizar es:

**tipoDeMedida:valorActual**

Mientras que en caso de tratarse de un actuador, la estructura del mensaje es algo más compleja:

**Nombre:input:puerto**

En este caso estamos indicando el nombre del dispositivo, que se trata de un dispositivo del tipo actuador gracias al dato “input” y el puerto donde escuchará a la espera de recibir mensajes UDP con información para cambiar el estado. La razón de haber declarado dos puertos es precisamente esta, en este caso se ha utilizado un servomotor, pero con esta estructura se podrían controlar todo clase de actuadores como Relés o motores.

La función recibirDatos, que como se ha dicho antes, se utiliza para escuchar un puerto a la espera de recibir un mensaje con el contenido adecuado que modifique el estado de algún actuador es la siguiente:

```
String recibirDatos()
{
```

```
 boolean recibido=true;
 String res ="" ;
 int packetSize = Udp2.parsePacket();
 if(packetSize)
 {
        IPAddress remote = Udp2.remoteIP();
        char packetBuffer2[UDP_TX_PACKET_MAX_SIZE]={""};
   Udp2.read(packetBuffer2,UDP_TX_PACKET_MAX_SIZE);
   recibido=false;
   Udp.flush();
   res=String(packetBuffer2);
 }
 return res ;
}
```

Esta función nos retorna un “String” con el mensaje que se recibe, una vez que tengamos el mensaje solo debemos de tratarlo. La estructura que se utiliza para estos mensajes es:

**Identificador/nuevoValor**

El identificador especificará el actuador al que va dirigido el mensaje, la otra parte contiene el valor que se ha enviado. En el proyecto usaremos este tipo de mensajes para alterar el ángulo de giro del servomotor que sostiene la cámara USB que visualizamos gracias al servicio Motion, creando una pseudo camaraIP.

## 5.6 Código Java

El código que se ha desarrollado utilizando java, actuará de intermediario entre la red y la base datos. Se encargará de almacenar los datos una vez tratados, que son enviados por los nodos, además de registrar los nodos e identificarlos. La idea es desarrollar un demonio que cree un socket en un determinado puerto y escuche la información entrante, este dispondrá de dos hilos. Un hilo principal que trate los mensajes y añada los datos a la base de datos, y el otro dé a conocer a los nodos la dirección del servidor.

Se han utilizado varias librerías para desarrollar los diferentes elementos que conforman el programa, entre ellas cabe desatacar las siguientes:

- **DatagramPacket:** esta librería nos proporciona todos los constructores necesarios para usar las instancias. Estas instancias codificarán los datos en un mensaje UDP que puede ser transmitido, la forma que tendría la instancia es la siguiente:

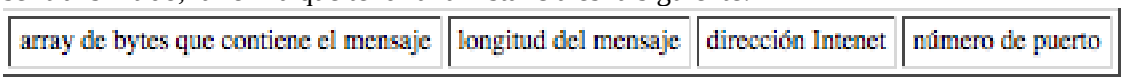



*Ilustración 30: Partes de un mensaje UDP*

  De este modo podemos acceder a cada una de las partes del mensaje a través de métodos get.

- **DatagramSocket:** con esta librería se nos proporciona todo lo necesario para utilizar y crear sockets. Estos sockets son los encargados de transmitir y recibir datagramas UDP.

Cuando creemos las instancias de los mensajes de la librería DatagramPacket, serán enviados a través del socket creado gracias a DatagramSocket.

- **InnetAddress:** Es una librería que nos ayuda a crear instancias de direcciones IP, algo que nos será útil para tratar las direcciones de los nodos cuando se reciban.

- **Librerías SQL**: las libras que se ha usado es JDBC, concretamente se importó un archivo sqljdbc.jar. Este archivo nos proporciona un conjunto de interfaces y métodos, con todo lo necesario para establecer y mantener conexiones entre la base de datos y el código JAVA.

La clase que se ha utilizado para el código tiene que extender de la clase “Thread”, de este modo podremos crear varios hilos de ejecución en el programa. De este modo el hilo principal estará en continua ejecución tratando los datos, y el hilo secundario enviara a los nodos el identificador que usaran en el sistema. El hilo secundario no permanecerá activo todo el tiempo, puesto que se ejecutará una vez cada minuto.

```
try {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
                conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/hermes", "root", "");
                comunicacionSql = (Statement)
conn.createStatement();
            } catch (InstantiationException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            } catch (SQLException e) {
                e.printStackTrace();
            }
```

Para establecer la conexión con la base de datos, primero tenemos que importar el driver que nos ayudará a establecer la conexión entra ambas tecnologías. Una vez cargado el “Driver” podemos solicitar una conexión con la base datos mediante el método “getConnection”. Este método espera recibir como parámetro la dirección de acceso a la base de datos, el usuario de acceso y la contraseña del usuario. Como en este proyecto es el mismo servidor el que ejecuta el código lo configuraremos localmente, a través del puerto 3306 y seleccionando la base de datos donde hemos creado las tablas Arduino y Mediciones. Una vez establecida se puede guardar en un objeto de tipo “Statement”, para ser utilizada cuando sea oportuno.

```
while(true)
{
DatagramPacket dgpRecep = new DatagramPacket(mensajeRecibido,
mensajeRecibido.length);
socketEnvio.receive(dgpRecep);
msg = new String(dgpRecep.getData(),0,dgpRecep.getLength());
```

```
datosRecibidos = msg.split("/");

if(datosRecibidos[0].equals("iniSess"))
{
		String tipo = datosRecibidos[1];
		flag=false;
		int res = procesoRegistro(dgpRecep.getAddress(),tipo,dgpRecep.getPort());
		String mensaje ="ident/"+res;
		byte[] dato = mensaje.getBytes();
		DatagramPacket dgp;
		dgp = new DatagramPacket(dato, dato.length,
InetAddress.getByName("192.168.1.255"),puertoArduino);
		socketEnvio.send(dgp);
}

if(datosRecibidos[0].equals("data"))
{
		int cont = Integer.valueOf(datosRecibidos[2]);
		int id = Integer.valueOf(datosRecibidos[1]);
		String datos="";
		int param=4;
		for(int i=0;i<4;i++)
		{
			if(i<cont){datos +=",'"+datosRecibidos[i+3]+"'";}

			else{datos +=",'';}}
		java.util.Date fecha = new Date();
		String sql="INSERT INTO `Mediciones`( `arduinoId`, `parámetros`, `valor1`,
`valor2`, `valor3`, `valor4`, `fecha`) VALUES
('"+id+"','"+cont+"'"+datos+",'"+fecha+"')";
		comunicacionSql.executeUpdate(sql);
}}}
```

La comunicación con la red se hace a través del puerto 8888, el hilo principal espera recibir toda la información en ese puerto, cuando le llegan los mensajes los filtra en función de la cabecera que contengan. En este caso filtramos en caso de dos posibles cabeceras, “iniSess” y “data”. En caso de tratarse de un mensaje con la cabecera ident, el programa da por hecho que se trata de un mensaje de inicio de sesión de un nodo, es decir que un nuevo nodo se ha conectado a la red.

Si por el contrario se trata de un mensaje de tipo data, es algo más complicado, puesto que tenemos que acceder al contenido del mensaje y tratarlo. El programa debe dividir el mensaje mediante el separador “/”, que utilizábamos en el código de Arduino y usar los parámetros que le proporcionamos. Destacar que el número de elementos introducido es muy importante, ya que con el conseguimos rellenar la sentencia sql que se encarga de la inserción de los datos en función de los elementos de los que disponga nuestro nodo.

```
while (true)
{
		dgp = new DatagramPacket(dato, dato.length,
```

```
InetAddress.getByName("192.168.1.255"),puertoArduino);
            socketEnvio.send(dgp);
            sleep(6000);
}
```

El hilo secundario por otra parte se encarga de utilizar el mismo socket que utilizamos para recibir los mensajes, para enviar mensajes de descubrimiento a los nodos. Cuando un nodo reciba este mensaje, actuará a modo de baliza localizando el servidor. Como se ha utilizado un bucle para que el nodo espere recibir un mensaje, vamos a utilizar el método “sleep” para que el hilo se mantenga en suspensión durante un minuto. Se ha utilizado el constructor del hilo para que cuando creamos el hilo, podamos pasarle la cabecera que será utilizada en mensajes de baliza.

```
public static int procesoRegistro( InetAddress ip, String tipo, int puerto)
{
try {
            String sql = "SELECT COUNT(id) FROM Arduino WHERE
`ip`='"+ip.toString().replace("/", "")+"'";
            respuestaSql = (ResultSet) comunicacionSql.executeQuery(sql);
            respuestaSql.next();

            if(respuestaSql.getInt(1)==0)
            {
                        sql = "INSERT INTO `Arduino`(`tipo`, `ip`,`puerto`) VALUES
('"+tipo+"','"+ip.toString().replace("/", "")+"','"+puerto+")";
                        comunicacionSql.executeUpdate(sql);
                        sql = "SELECT id FROM Arduino WHERE
`ip`='"+ip.toString().replace("/", "")+"'";
                        respuestaSql = (ResultSet) comunicacionSql.executeQuery(sql);
                        respuestaSql.next();
                        return Integer.parseInt(respuestaSql.getString("id"));
            }
            else
            {
                        sql = "SELECT id FROM Arduino WHERE
`ip`='"+ip.toString().replace("/", "")+"'";
                        respuestaSql = (ResultSet) comunicacionSql.executeQuery(sql);
                        respuestaSql.next();
                        return Integer.parseInt(respuestaSql.getString("id"));
            }
} catch (SQLException e) {e.printStackTrace();}
return 0;
}}
```

La función que se encarga del registro de los nodos debe recibir los parámetros IP, tipo y puerto. Estos parámetros son extraídos del mensaje que se recibe, con el que los nodos se intentan dar de alta. Lo que el programa hace es utilizar la IP para consultar si hay alguna entrada dentro de la base de datos relacionada con esa IP. En caso de encontrar una entrada se devuelve el identificador que le ha asociado el sistema, por el contrario si no se encuentra ninguna entrada realiza la inserción de los datos asociados y devuelve el identificador que se le asigno.

Para que sea fácil de ejecutar se ha generado un archivo .jar con el código del programa ya compilado para que pueda ser ejecutado desde nuestra Raspberry. Para asegurarnos de que se van a ejecutar, vamos a añadir un script que se encargue de lanzar el programa. Este script será añadido al archivo rc.local para que se ejecute cada vez que reiniciemos la tarjeta. El archivo lo podemos encontrar en la ruta:

**/etc/rc.local**

En este archivo añadiremos la ruta hacia el siguiente script que se encargará de levantar el servicio:

```
#!/bin/bash
#Script de reinicio
java -jar /home/pi/servidorDomotico.jar &&
```

## 5.7 Código de la página Web

Por último pero no menos importante vamos a hablar acerca de  la web que se utiliza en este proyecto, esta web es muy importante puesto que actuará a modo de interfaz gráfica entre el usuario y el servidor. Para el desarrollo vamos a utilizar la estructura principal que se explicó en el apartado de análisis. Para cargar el contenido de cada parte del sistema se han desarrollado códigos PHP que cargan el contenido de cada una de ellas, de esta forma podremos renovar el contenido cuando nos interese.

Hablando de la estructura tenemos que aclarar que se ha utilizado una estructura de directorios determinada, de esta forma podremos mantener la coherencia entre nuestros códigos y localizar cada tipo de fichero en un mismo contenedor.

- Directorio **css**: en este directorio guardaremos aquellos archivos que utilicemos para darle estilo en nuestra web. En estos archivos aparecerán reflejados los identificadores de los diferentes elementos de nuestra web, justo con el conjunto de reglas que especifican su aspecto visual en el navegador.

- Directorio **img**: la carpeta albergará todas las imágenes que aparecerán referenciadas en nuestra web.

- Directorio **js**: en esta carpeta se guardarán todos los códigos javascript que el usuario se descargará a través del navegador. Estos ficheros contienen todas las funciones que se pueden utilizar en los ficheros de la web. Estos códigos se ejecuta en el navegador del cliente, por lo que la carga de proceso recae sobre el dispositivo. Destacar que hemos añadido las librerías de JQuery para facilitar el acceso a los elementos del documento DOM, que contendrá a la interfaz. De este modo cuando tengamos que referenciar cualquier elemento o queramos realizar llamadas AJAX, entre otras muchas funciones, tendremos que utilizar muchas menos instrucciones simplificando de esta manera el código. La tecnología AJAX nos permitirá realizar llamadas al servidor y traernos de

este fichero concreto, de este modo podremos cargar de manera dinámica los ficheros PHP.

- Directorio **PHP**: este directorio contiene los archivos PHP que complementan la estructura de la web. Esos ficheros devuelven el contenido que luego será introducido en las diferentes partes de la web, al utilizar estos ficheros junto a la tecnología AJAX mantendrán los datos de la web a tiempo real.

Para la estructura principal del documento se han diseñado los elementos “listaArduinos”, “listaModulos”, “header” y “altaArduino”. El nombre de estos elementos corresponde al identificador que utilizamos el código HMTL para identificarlos. Estos elementos se han etiquetado bajo la etiqueta “div” tienen asociado un estilo en a los archivos css que puede ser modificado en cualquier momento de manera rápida.

**ListaArduino** es un elemento div que corresponde a la zona roja del diseño que se presenta en el apartado de análisis. Este bloque contendrá la lista de contenedores Arduino de los que se disponga en la base de datos, estos contenedores corresponden al nombre que le damos al nodo cuando terminamos el registro de mismo en la web. De este modo podemos asociar varios nodos a un identificador de grupo, esto es muy útil si por ejemplo queremos asociar varios nodos a un mismo lugar como es nuestro caso. Esta zona contendrá una lista de elementos que podremos pulsar para que en la web sólo aparezcan los elementos asociados a ese identificador, estos elementos como el contenedor, tienen asociados códigos css que lo mantienen en la parte izquierda de la pantalla ocupando la totalidad del alto de esta. En caso de que la lista de elementos supere el tamaño del alto de la pantalla se utiliza un *scroll* para tratar el problema de desbordamiento. La lista de elementos identificador se carga mediante una llamada a la siguiente función javaScript:

```
función cargarArduinos()
{
   $.ajax({url:   'arduinosList.php',
         type:  'get',
         success:  función (response) {
         $("#listaArduinos").append(response);}});}
```

Esta función utiliza la librería jQuery para llamar desde el navegador cliente un fichero PHP y esperará a que éste le devuelva los datos que posteriormente serán utilizados para rellenar el elemento listaArduino. El archivo PHP hace una llamada a la base de datos y se trae los datos asociados a los nodos.

```
$query = "SELECT DISTINCT(`nombre`),`id` FROM `Arduino`";
```

Gracias a esta sentencia sql podemos devolver el nombre de todos los nodos asociados, pero devolviendo una entrada por cada campo de tipo nombre. Como este campo puede tener varias entradas idénticas dentro de la tabla, sólo devolverá uno por cada grupo.

Una vez que tengamos los elementos que tenemos que introducir los introducimos dentro de elementos “div” en los cuales atraparemos el evento onclick. Cada uno de estos elementos está asociado a un clase referenciada dentro del archivo de estilos css, de este modo cuando se realicen cambios estéticos sobre la web, se aplicarán a todos los elementos simultáneamente. Al atrapar el evento onclick, podemos ejecutar una función javascript que cambiará el contenido de la web por los nodos asociados a este identificador.

```
<div class="listado" onclick="arduinoEspecifica('45')">
```

Este es aspecto de la lista de elementos que devolverá el archivo “arduinoLista.php”. La función a la que llamamos cuando un usuario pulsa sobre el elemento, llama al archivo “modulosLista.php”, pasándole como parámetro el identificador de elemento pulsado para que cambie los nodos que aparecen en la web sin recargar la página.


*Ilustración 31: Captura de la lista de Arduinos*

**Header** es el elemento div que corresponde a la zona azul del diseño que se presentaba en la parte de análisis. En esta zona se colocan todos los servicios de los que dispondrá nuestro servidor, en este caso vamos a poner enlaces para el servicio CUPS, Owncloud y configuración de la web cam. Se ha diseñado una estructura sencilla de HTML donde podemos colocar la

información de los servicios. Este código es replicable, por lo que en caso de añadir nuevos servicios, basta con copiar la estructura con los datos asociados a ese servicio.

```
<a href=""><img class="servicioExterno" src="img/"></a>
```

Para añadir el servicio solo tenemos que añadir la dirección “localhost:” con el puerto asociado a este y una imagen, para darle estilo dentro de la web se ha creado una clase específica para estos iconos. Esta imagen es el icono que aparecerá en el menú para enlazar la web con el servicio.


*Ilustración 32: Captura del menú de la página web*

**listaArduino** corresponderá a la zona verde del esquema, en esta zona es donde aparecerá la información asociada a cada uno de los nodos que conectemos a nuestra red. Se ha desarrollado un función en JavaScript que cargará el contenido de este elemento div.

```
función cargarModulos()
{
   $.ajax({
            url:   'modulosList.php',
            type:  'get',
            success:  función (response) {
            $("#listaModulos").append(response);

            }});}
```

En este caso se hace una llamada al archivo “moduList.php” sin pasarle ningún parámetro, por lo que devolverá el todos los nodos que se encuentren en la base de datos. Los elementos que devuelve este archivo siguen una estructura determina, cada parte tiene asociada una de las dos clases de las que se dispone “sensor” y  “actuador”.

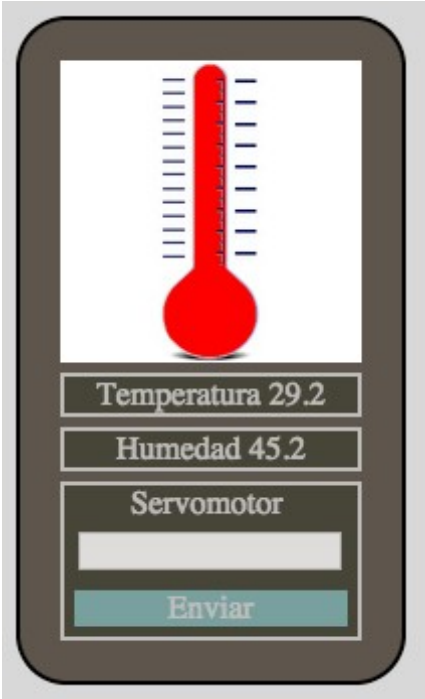



*Ilustración 33: Captura de un nodo de la página web*

Como se puede apreciar lo que más llama la atención es la imagen que se encuentra en el módulo, en este caso se ha puesto un enlace al puerto del servicio M*otion*. De esta forma tendremos una cámara ip que podremos controlar, ya que en este proyecto hemos montado la webcam sobre un servomotor que podremos manejar desde la página web. En caso de que queramos cambiar la imagen asociada a cada elemento, solo tenemos que ejecutar una función que cambie la imagen. Como el código javaScript puede ejecutarse tanto mientras se carga la página, como cuando ya está cargada, solo tenemos que introducirla justo después de cerrar el div listaArduino.

Como se puede ver se ven dos tipos de datos dentro del nodo, dos de ellos corresponden a un sensor y el otro aún actuador. En este caso los datos que se están mostrando son los datos recibidos del módulo HDT11 que se utiliza para medir la temperatura y la humedad ambiental, el actuador que corresponde al servomotor que se sitúa debajo de la webcam.

Los datos del sensor son tratados antes de ser mostrados, para conseguir una representación de la temperatura más realista. La función encargada de añadir los datos asociados a los sensores de este tipo aplica la media, obtiene las últimas 5 medidas que se ha añadido ese sensor para obtener una visión del sensor más representativa y no estar mostrando la temperatura momentánea.

```
while ($fila2 = mysql_fetch_assoc($result2))
        {
                if($fila2['valor1'] != "")
                {
                        $arrayAux = split(":", $fila2['valor1']);
                        $clave1=$arrayAux[0];
                        array_push($cont1, $arrayAux[1]);
                }
```

```
                if($fila2['valor2'] != "")
                {
                        $arrayAux = split(":", $fila2['valor2']);
                        $clave2=$arrayAux[0];
                        array_push($cont2, $arrayAux[1]);
                }
                if($fila2['valor3'] != "")
                {
                        $arrayAux = split(":", $fila2['valor3']);
                        $clave3=$arrayAux[0];
                        array_push($cont3, $arrayAux[1]);
                }
                if($fila2['valor4'] != "")
                {
                        $arrayAux = split(":", $fila2['valor4']);
                        $clave4=$arrayAux[0];
                        array_push($cont4, $arrayAux[1]);
                }

        }
```

Para llevar a cabo el cálculo de la media, la función crea un array bidimensional relacionando los campos de tipo valor y los valores asociados. Cuando tenemos este array basta con recorrerlo acumulando los diferentes sumatorios y dividir entre el número de elementos del array interior.

El formato de los actuadores por el contrario consta de un pequeño formulario donde introducir los datos. Para enviar la información basta con rellenar el campo de texto con los datos que queramos enviar y pulsar el botón. Cuando se ejecuta el formulario se ejecuta la siguiente función:

```
función enviarCambio(datos)
{

   var valor = $("#valor_"+datos).val();
   alert("el valor es "+valor);
   var nombre = $("#nombre_"+datos).val();
   $.ajax({
            data: 'datos='+datos+'&mod='+nombre+"&valor="+valor,
            url:   'mensajeArduino.php',
            type:  'get',
            success:  función (response) {
           }});}
```

Esta función también utiliza de la tecnología AJAX para llamar al archivo “mensajeArduino.php”. Al contrario que otros ficheros anteriormente comentados este no devolverá información que modifica el estado de la página sino el de nuestro nodo. Gracias a la

extensísima API de PHP, se ha utilizado esta llamada para crear un socket UDP y enviar un mensaje (respetando la estructura utilizada) al nodo.

```
$fila = mysql_fetch_assoc($result);
$sock = socket_create(AF_INET, SOCK_DGRAM, SOL_UDP);
$msg = "".$_GET['mod']."/".$_GET['valor'];
$len = strlen($msg);
echo "la ip es ".$fila['ip']." el puerto ".$fila['puerto'];
socket_sendto($sock, $msg, $len, 0, $fila['ip'],8887/*$fila['puerto']*/);
socket_close($sock);
```

Gracias a las funciones socket, no tenemos por qué comunicar el código de la web con el código java del servidor. De esta forma conseguimos un acceso directo hasta los nodos, gracias a la información que se obtiene de la base de datos, y reducimos el consumo de recursos de nuestra Raspberry, al añadir más carga al código JAVA.

Por último aclarar que se utiliza la función setInterval para ejecutar funciones cada cierto tiempo, entre ellas se encuentran las funciones que cargan los nodos y los identificadores para mantener los datos actualizados a tiempo real.

**altaArdiono** corresponde a la señal de color naranja que podía verse en el esquema que se presenta durante el análisis de la página web. Este elemento no es visible en la web de forma habitual, como anteriormente se explicaba se ha utilizado un setInterval para ejecutar de forma constante ciertas funciones para este elemento es muy importante “nuevosElementos”. Esta función se ejecuta cada cierto tiempo, es muy sencillo de modificar sobre el código, realiza una llamada AJAX para comprobar si existen nuevos nodos sin dar de alta en el servidor.

```
función nuevosElementos()
{
        $.ajax({
           url:   'nuevosElementos.php',
           type:  'get',
           success:  función (response) {
                if (response != 0)
                {
                        $("#aviso").css("display","inline");
                }}});}
```

Esta función es diferente a las anteriores, en este caso cuando el código PHP se ejecuta sin devolver nada de información significa que no hay nuevos nodos conectados en la red. Pero en caso de que ese código devuelva información se hará visible el aviso que indicará al usuario que tiene información que configurar.

*Ilustración 34: Icono alerta de la página web*

Cuando el usuario pulsa sobre el icono que aparece en la esquina superior izquierda se ejecuta la función “darAlta”. Esta función hace aparecer un bloque de configuración con una animación deslizante desde la izquierda. La información de este formulario se rellena mediante otra función AJAX.

```
función darDeAlta()
{
   clearInterval(refrescarAviso);
        $("#aviso").css("display","none");
        $.ajax({
                url:   'nuevosDatos.php',
           type:  'get',
           success:  función (response) {
           $("#altaArduino").append(response);

           }
   });
        $("#altaArduino").css("display","inline");
        $("#altaArduino").animate({'width': "250px",});}
```

Este código PHP realiza una llamada a la base de datos y obtiene la información relaciona con los datos de aquellos nodos que no tienen asociado nada en el campo nombre. Con la información se utiliza para generar el formulario con el que se termina el proceso de registro del nuevo nodo.

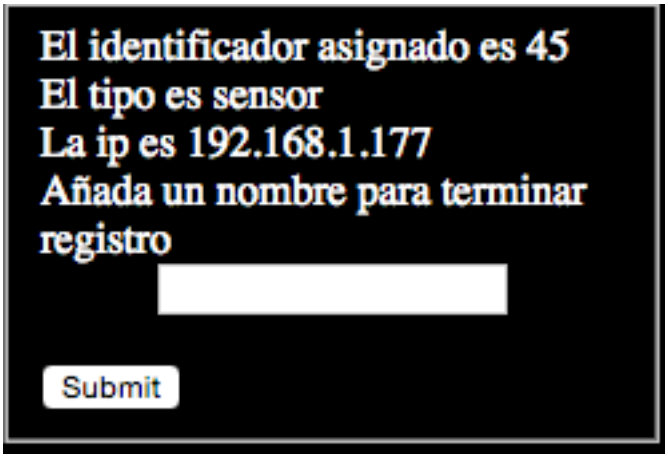



*Ilustración 35: Captura del menú de registro de nodos*

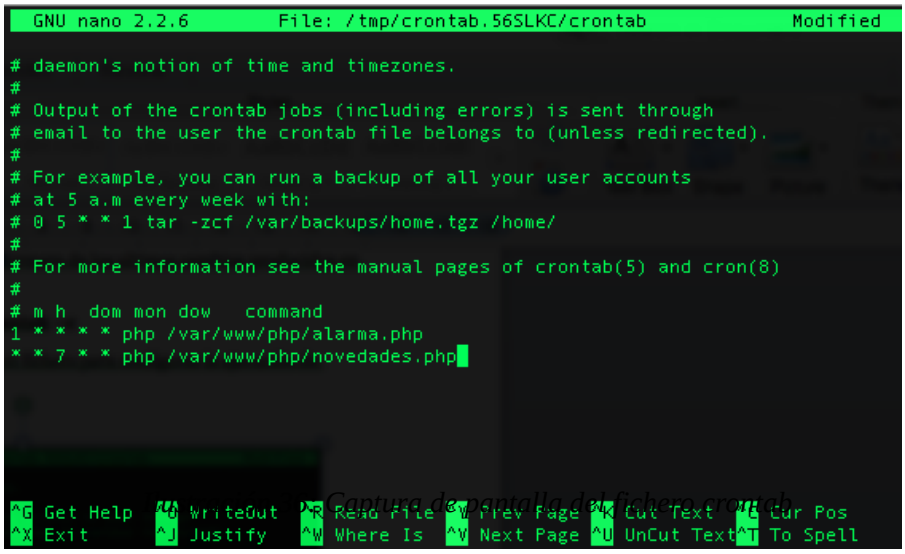
Sólo hay que rellenar el campo del nombre, que es el identificador que usamos para agrupar Arduinos y estará listo. Cuando se refrescan los datos de la web aparecerá este nuevo elemento aparecerá con el resto de datos y desaparecerá el menú de configuración junto con la alerta de nuevo nodo.

## 5.8 Crontab

El crontab es un fichero que se encuentra en los servidores y permite ejecutar scripts o PHP regularmente. Esto nos es muy útil para ejecutar funciones que actuarán a modo de alertas, en este proyecto vamos a ejecutar dos script. Un script se ejecutará cada minuto y consultará la última medida introducida, en caso de que ésta supere el tope introducido nos enviará un correo electrónico avisándonos del problema. El otro código se ejecutará una vez a la semana y nos informara de los últimos cambios producidos en la red.

**crontab –e**

Esta función nos permite entrar dentro del fichero para configurar la ejecución de nuestros scripts.




*Captura de pantalla del fichero crontab*

En este archivo podemos ejecutar toda clase de acciones en el servidor, como por ejemplo backups de seguridad o toda clase de alertas o actualizaciones. El código PHP que se ejecuta es un conjunto de código PHP y bash. Utilizaremos el PHP para obtener la información de la base de datos que nos interesa y rellenar la información del mensaje que luego se le pasa a la función “shell_exec”, ésta ejecutará los comandos necesarios para utilizar el gestor de correo para enviar mensajes al correo electrónico.

## 5.9 Manual de usuario

Para utilizar este servidor sólo tenemos que utilizar las funciones Arduino diseñadas para configurar el esquema de nuestro nodo y las librerías necesarias. Una vez que inicialicemos el identificador del nodo gracias a la función “procesoRegistro()” utilizando la variable “id”, podemos utilizar las otras funciones para recibir datos o enviarlos dependiendo de la utilidad que queramos darle a nuestro nodo. Los datos que son enviados al servidor han de seguir la estructura que se exponía anteriormente, para mantener la coherencia de datos en el servidor.

Una vez listo y cargado el código en nuestra Arduino, solo tenemos que conectarla a la red. El servidor se encargará de registrar la dirección IP y el puerto del nodo para ser configurado

posteriormente. Si entramos en la web podemos ver que aparecen los datos asociados al nodo, pero este no tiene datos asociados. Tras esperar dentro de la web 1 minuto podemos ver que nos aparece una señal que nos indica que se ha detectado un nuevo nodo.

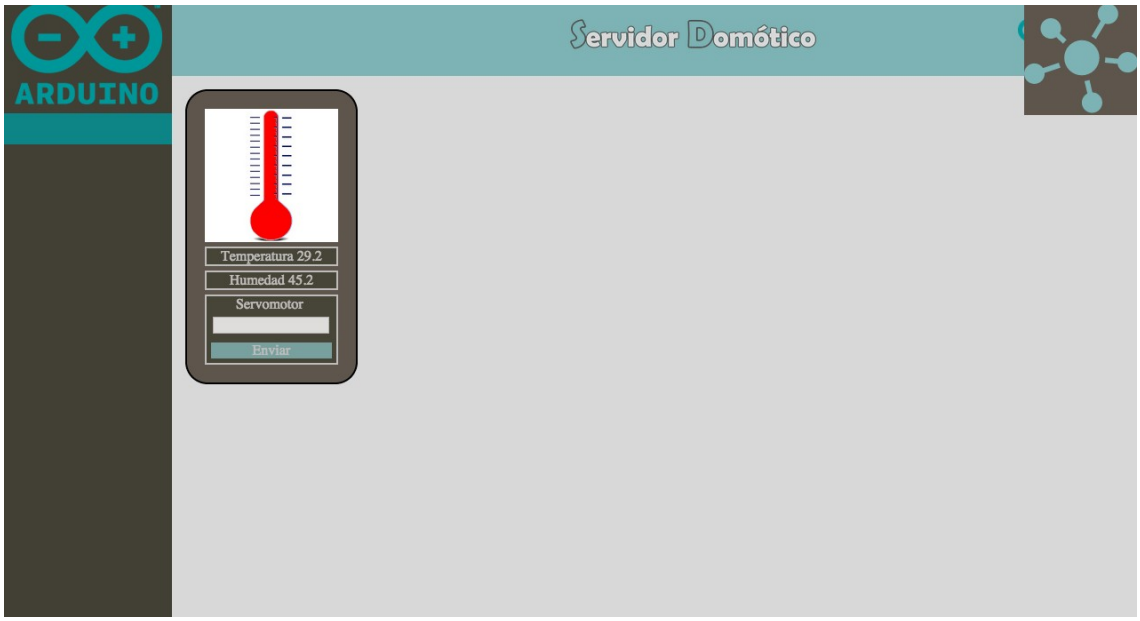



*Ilustración 37: Captura de pantalla, aviso página web*

Si pulsamos sobre el icono de conexión que ha aparecido en la parte superior izquierda nos aparecerá un formulario, mediante una función deslizante. En este formulario se configura el identificador de grupo del nodo que acabamos de conectar.

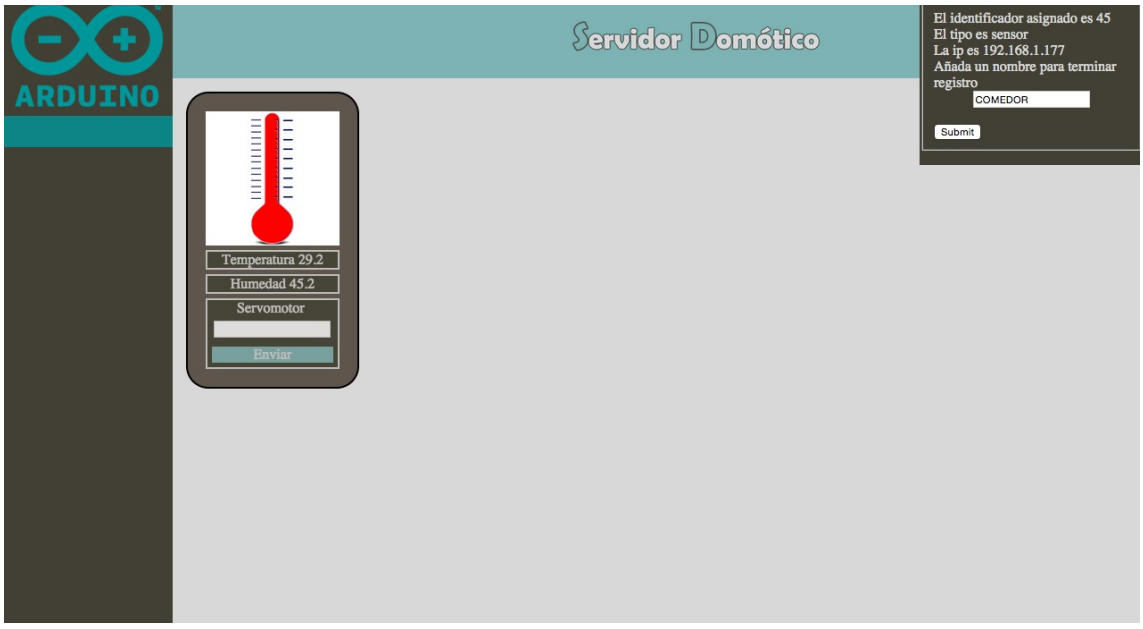



*Ilustración 38: Captura de pantalla, menú de configuración de nodo*

Vamos a utilizar el nombre “COMEDOR” para identificar este módulo. Cuando mandamos los formularios, con el fin de que se ejecute en el servidor el php que añade la nueva configuración al nodo. Tras enviar la información se nos recargara la página con la nueva información.

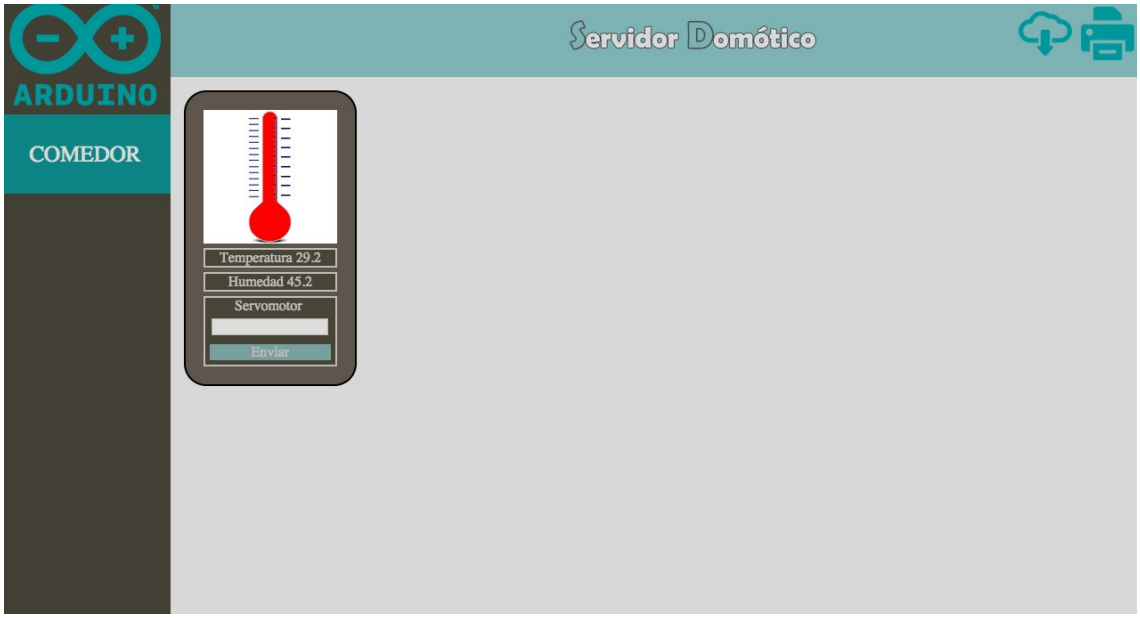



*Ilustración 39: Captura de pantalla, página web configurada*

Una vez hecho esto ya está el nuevo nodo configurado dentro de nuestro servidor. De esta forma ya está listo para enviarle la información que queramos o ver los datos que está enviando.

# 6 Conclusiones

En conclusión este es un trabajo con mucho potencial que puede explotarse de muchas maneras. Este proyecto abre la puerta a la imaginación del desarrollador, por un lado permite que podamos tener conectados toda clase de elementos de nuestra vida cotidiana a nuestra red como las luces del comedor o la cafetería. Es tan sencillo como conectar un relé a una Arduino y podremos activarlos o desactivarlos desde cualquier lugar del mundo. Esto es algo fascinante, el hecho de que podamos conectar la cafetera al despertarnos por la mañana, parece digno de estudio.

Al ver el proyecto terminado me he dado cuenta de que se tomó la decisión correcta en lo que se refiere al uso de apache. La decisión de no usar servlets para comunicar el código web y el código java podría haber generado muchos problemas. Como hemos utilizado HTML para el contenido de la web y se han importado todas las librerías jQuery, se me ocurren millones de posibilidades para utilizar el formulario que nos proporciona el servidor. Por ejemplo en caso de que tengamos un actuador que encienda las luces del comedor, se pueden añadir funciones que modifiquen el aspecto del formulario. Como este código se ejecuta sobre el navegador del cliente no nos preocupa de que se sobrecargue nuestro servidor, por lo que cualquier modificación que se haga no afectara al funcionamiento del servidor.

En mi opinión a este servidor le faltan muchas mejoras, como activar el puerto de HTTPS para cifrar el trafico web en caso de que alguien espíe nuestra red, cerrar todos los puertos que puedan suponer un problema en caso de un ataques maliciosos, configurar el cortafuegos que podemos instalar en nuestra tarjeta tratando todo el tráfico entrante y saliente, crear un sistema de usuarios y permisos para limitar el acceso a determinadas partes o mejorar el sistema de alertas.

Ha sido una lástima no contar con presupuesto para probar la última versión de Raspberry, me hubiera gustado poner a prueba el nuevo hardware. Es mucho más potente que la que se ha utilizado para este proyecto al contar con un procesador de 4 núcleos y 1 GB de RAM, lo que supondría un mayor rendimiento en el servidor.

En resumen he disfrutado mucho con el desarrollo de este proyecto, me ha parecido un tema interesantísimo con un potencial increíble. Puede ser un primer paso para abrir las puertas a un futuro en el que tendremos control de toda clase cosas como plantas o cerraduras.

# 7 Bibliografía


[1] Página web oficial de Arduino, en ella se puede encontrar información acerca de la información y distribución de sus productos: https://www.arduino.cc/

[2] Blog oficial de Arduino donde podemos encontrar información sobre proyectos y ayudas: https://blog.arduino.cc/

[3] Blog con un gran numero de manuales sobre como configurar y explotar al máximo nuestra Raspberry: https://geekytheory.com

[4] Página web oficial de Raspberry PI, en ella se puede encontrar información acerca de la información, distribución de sus productos y sus sitemasar operativos: https://www.Raspberrypi.org/

[5] Api de Java: http://docs.oracle.com/javase/7/docs/api/

[6] Api de html y css: http://www.w3schools.com/html/

[7] Api de PHP: http://php.net/

[8] Tutorial sobre la clase DatagramSocket para UDP: http://tutorials.jenkov.com/java-networking/udp-datagram-sockets.html

[9] Comparativa entre TCP y UDP: http://es.diffen.com/tecnologia/TCP-vs-UDP

[10] Analisis comparativo de los diferentes modelos Arduinos: http://comohacer.eu/analisis-comparativo-placas-arduino-oficiales-compatibles/

[11] Analisis comparativo de los diferentes modelos Raspberry PI: http://comohacer.eu/comparativa-y-analisis-Raspberry-pi-vs-competencia/

[12] Almacen de iconos gratuitos: http://www.flaticon.es/

[13] Almacen de iconos gratuitos: http://www.freepik.es/iconos-gratis

[14] Comparativa entre Mysql y PostgresSql: https://blog.udemy.com/mysql-vs-postgresql/

[15] James F. Kurose & Keith W. Ross (2011), “Redes de Computadoras: un enfoque descendente”, 5ta Edición de Prentice Hall

[16] Ribas Lequerica, Joan (2013), “Manual imprescindible de Arduino práctico”, Anaya Multimedia

[17] Upton, Eben (2013), “Raspberry Pi. : guía del usuario”, Anaya Multimedia

[18] Arias, Ángel (2014), “Aprende a programar Ajax y jQuery”, Amazon

[19]

# 8 Anexos

## 8.1 Función de registro (Arduino)

```
String procesoRegistro()
{

   while(true)
  {

   int packetSize = Udp.parsePacket();
   if(packetSize)
      {
       IPAddress remote = Udp.remoteIP();
        ipServidor=Udp.remoteIP();
        puertoServidor=Udp.remotePort();
        Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE);
        if (strcmp(packetBuffer,respuestaReconocimiento) == 0)
        {
          msg="iniSess/"+String(tipo);
          Udp.beginPacket(Udp.remoteIP(),Udp.remotePort());
          msg.toCharArray(envio,UDP_TX_PACKET_MAX_SIZE);
          Udp.write(envio);
          Udp.endPacket();
        }
        int separador=String(packetBuffer).indexOf("/");
        String res="";
        res = String(packetBuffer).substring(0,separador);
        if (res=="ident")
        {
         return String(packetBuffer).substring(separador+1);
        }
       }
  }
}
```

## 8.2 Funcion para enviar datos (Arduino)

```
void enviarDatos(int numDat, String array [])
    {
     String dat="";
```

```
for (int i=0; i <= numDat - 1;i++)
    {
        dat +="/"+array[i];
      }
      msg="data/"+id+"/"+numDat+""+dat;
      Udp.beginPacket(ipServidor,puertoServidor);
      msg.toCharArray(envio,UDP_TX_PACKET_MAX_SIZE);
      Udp.write(envio);
      Udp.endPacket();
  }
```

## 8.3 Función para recibir datos (Arduino)

```
String recibirDatos()
    {

      Serial.println("entramos recivir");
      boolean recibido=true;
      String res ="" ;
      int packetSize = Udp2.parsePacket();
      if(packetSize)
      {

        IPAddress remote = Udp2.remoteIP();
        char packetBuffer2[UDP_TX_PACKET_MAX_SIZE]={""};
        Udp2.read(packetBuffer2,UDP_TX_PACKET_MAX_SIZE);
        recibido=false;
        Serial.println("El contenido es");
        Serial.println(String(packetBuffer2));
        Udp.flush();
    res=String(packetBuffer2);
      }
      return res ;
    }
```

# 8.4 Código java

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
```

```
import java.net.UnknownHostException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Date;

import com.mysql.jdbc.ResultSet;
import com.mysql.jdbc.Statement;


public class Servidor extends Thread
{
	private	static int puertoArduino=8888;
	private static DatagramSocket  socketEnvio;
	private String id;
	private static String msg;
	private static String[] datosRecibidos;
	private static Statement comunicacionSql;
	private static ResultSet respuestaSql;
	private static Connection conn;
	private static boolean flag = true;

	public static void main(String[] args)
	{

		try
		{
			try {

	Class.forName("com.mysql.jdbc.Driver").newInstance();
				conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/hermes",
"root", "");
				comunicacionSql = (Statement)
conn.createStatement();
			} catch (InstantiationException e) {
				e.printStackTrace();
			} catch (IllegalAccessException e) {
				e.printStackTrace();
			} catch (ClassNotFoundException e) {
				e.printStackTrace();
			} catch (SQLException e) {
				e.printStackTrace();
			}

			socketEnvio = new DatagramSocket(8888);
			byte [] mensajeRecibido = new byte [1024];
			Servidor h = new Servidor ("comun");
			h.start();
```

```
                while(true)
                {
                        DatagramPacket dgpRecep = new
DatagramPacket(mensajeRecibido, mensajeRecibido.length);
                        socketEnvio.receive(dgpRecep);
                        msg = new
String(dgpRecep.getData(),0,dgpRecep.getLength());

                        datosRecibidos = msg.split("/");

                        if(datosRecibidos[0].equals("iniSess"))
                        {
                                String tipo = datosRecibidos[1];
                                flag=false;
                                System.out.println("Proceso de
registro");
                                int res =
procesoRegistro(dgpRecep.getAddress(),tipo,dgpRecep.getPort());
                                String mensaje ="ident/"+res;
                                byte[] dato = mensaje.getBytes();
                                DatagramPacket dgp;
                                dgp = new DatagramPacket(dato,
dato.length,
InetAddress.getByName("192.168.1.255"),puertoArduino);
                                socketEnvio.send(dgp);
                                System.out.println("El identificador es
->"+res);
                                System.out.println("OK");
                                System.out.println(datosRecibidos[1]);
                        }
                        if(datosRecibidos[0].equals("data"))
                        {
                                int cont =
Integer.valueOf(datosRecibidos[2]);
                                int id =
Integer.valueOf(datosRecibidos[1]);
                                String datos="";
                                int param=4;
                                for(int i=0;i<4;i++)
                                {
                                        if(i<cont)
                                        {
                                                datos
+=",'"+datosRecibidos[i+3]+"'";
                                        }
                                        else
                                        {
                                                datos +=",''";
                                        }
                                }
```

```
                                java.util.Date fecha = new Date();
                                String sql="INSERT INTO
`Mediciones`( `arduinoId`, `parametros`, `valor1`, `valor2`, `valor3`,
`valor4`, `fecha`) VALUES
('"+id+"','"+cont+"'"+datos+",'"+fecha+"')";
                                comunicacionSql.executeUpdate(sql);
                        }
                }
        }
        catch (SocketException e)
        {
                System.out.println("Fallo al crear el socket");
                System.out.println(e);
        }
        catch (SecurityException e)
        {
                System.out.println("Fallo de seguridad");
                System.out.println(e);
        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }

}
public void run()
{

        String mensaje = id;
        byte[] dato = mensaje.getBytes();
        DatagramPacket dgp;
        try {
                while (true)
                {
                        dgp = new DatagramPacket(dato, dato.length,
InetAddress.getByName("192.168.1.255"),puertoArduino);
                        socketEnvio.send(dgp);
                        sleep(5000);
                }

        }
        catch (UnknownHostException e)
        {
                e.printStackTrace();
        }
        catch (IOException e)
        {
```

```
                e.printStackTrace();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

        }
        public Servidor(String id)
        {
            this.id = id;
        }

        public static int procesoRegistro( InetAddress ip, String tipo, int
puerto)
        {
            try {

                String sql = "SELECT COUNT(id) FROM Arduino
WHERE `ip`='"+ip.toString().replace("/", "")+"'";
                respuestaSql = (ResultSet)
comunicacionSql.executeQuery(sql);
                respuestaSql.next();

                if(respuestaSql.getInt(1)==0)
                {
                    sql = "INSERT INTO `Arduino`(`tipo`,
`ip`,`puerto`) VALUES ('"+tipo+"','"+ip.toString().replace("/", "")
+"',"+puerto+")";
                    comunicacionSql.executeUpdate(sql);
                    sql = "SELECT id FROM Arduino WHERE
`ip`='"+ip.toString().replace("/", "")+"'";
                    respuestaSql = (ResultSet)
comunicacionSql.executeQuery(sql);
                    respuestaSql.next();
                    return
Integer.parseInt(respuestaSql.getString("id"));
                }
                else
                {
                    sql = "SELECT id FROM Arduino WHERE
`ip`='"+ip.toString().replace("/", "")+"'";
                    respuestaSql = (ResultSet)
comunicacionSql.executeQuery(sql);
                    respuestaSql.next();
                    return
Integer.parseInt(respuestaSql.getString("id"));
                }


            } catch (SQLException e) {
                // TODO Auto-generated catch block
```

```
                e.printStackTrace();
            }

            return 0;

        }
}
```

## 8.5 Servidor.php (Interfaz)

```
<html>
<?php
if($_GET)
{
        $link = mysql_connect('localhost', 'root', '') or die('No se pudo
conectar: ' . mysql_error());
        mysql_select_db('hermes') or die('No se pudo seleccionar la
base de datos');

        for ($i=0; $i < $_GET['numero']; $i++) {
                $nombre = "nombre".$i;
                $id = "id".$i;
                $sql = "UPDATE `Arduino` SET `nombre`='".
$_GET[$nombre]."' WHERE `id`=".$_GET[$id];
                //echo $sql."\n";
                mysql_query($sql);
                //echo $sql;
        }

}
?>

<head>
 <meta name="tipo_contenido"  content="text/html;" http-
equiv="content-type" charset="utf-8">
<title>Servidor Domotico</title>
<link rel="stylesheet" type="text/css" href="css/estilos.css"/>
<script type="text/javascript" src="js/jquery-
1.11.3.min.js"></script>
<script type="text/javascript" src="js/scripts.js"></script>
<script type="text/javascript">

        $(document).ready(function($){
                cargarArduinos();
                cargarModulos();
                refrescarAviso=setInterval('nuevosElementos()',60000);
        });
```

```
</script>
</head>

<body>

<div class="pagina">
<div id="listaArduinos" class="lista">
	<center>
		<img src="img/Arduin_logo.png">
	</center>
</div>
<div class="header">
	<img src="img/titulo.png" width="300px" style="margin-
left:40%;">
	<img class="servicioExterno" src="img/nube.png"
width="60px">
	<img class="servicioExterno"src="img/impresora.png"
width="60px">
	<div onclick="darDeAlta()" id="aviso"> <img
src="img/aviso.png"></div>
</div>
<div id="listaModulos" class="contenido"></div>
<div id="altaArduino"></div>
</div>
</body>

</html>
```

## 8.6 mensajeArduino.php

```
<?php
	$link = mysql_connect('localhost', 'root', '') or die('No se pudo
conectar: ' . mysql_error());
	mysql_select_db('hermes') or die('No se pudo seleccionar la
base de datos');
	$aux= split("-", $_GET['datos']);
	$query = "SELECT `ip`,`puerto` FROM `Arduino` WHERE
`id`='".$aux[0]."'";
	$result = mysql_query($query) or die('Consulta fallida: ' .
```

```
mysql_error());

        if (mysql_num_rows($result) != 0)
        {
                $fila = mysql_fetch_assoc($result);
                $sock = socket_create(AF_INET, SOCK_DGRAM, SOL_UDP);
                $msg = "".$_GET['mod']."/".$_GET['valor'];
        $len = strlen($msg);
        echo "la ip es ".$fila['ip']." el puerto ".$fila['puerto'];
                socket_sendto($sock, $msg, $len, 0,
$fila['ip'],8887/*$fila['puerto']*/);
        socket_close($sock);

        }

?>
```

## 8.7 modulosList.php

```
<?php

$link = mysql_connect('localhost', 'root', '') or die('No se pudo
conectar: ' . mysql_error());
mysql_select_db('hermes') or die('No se pudo seleccionar la base de
datos');

$query = "SELECT DISTINCT (`arduinoId`) FROM `Mediciones`";
$result = mysql_query($query) or die('Consulta fallida: ' .
mysql_error());

if (mysql_num_rows($result) != 0)
{
   while ($fila = mysql_fetch_assoc($result))
   {
        $sql = "SELECT * FROM `Mediciones` WHERE `arduinoId`='".
$fila['arduinoId']."' ORDER BY `fecha` DESC LIMIT 5";
        $result2 = mysql_query($sql) or die('Consulta fallida: ' .
mysql_error());
        echo "<div class=\"moduloArduino\">";
        echo "<img src=\"img/termometro.jpg\" width=\"150\"
height=\"150\">";
        $cont1 =array();
        $clave1="";
        $cont2 =array();
        $clave2="";
        $cont3 =array();
```

```
$clave3="";
$cont4 =array();
$clave4="";
$total=array();
while ($fila2 = mysql_fetch_assoc($result2))
{
	if($fila2['valor1'] != "")
	{
		$arrayAux = split(":", $fila2['valor1']);
		$clave1=$arrayAux[0];
		array_push($cont1, $arrayAux[1]);
	}

	if($fila2['valor2'] != "")
	{
		$arrayAux = split(":", $fila2['valor2']);
		$clave2=$arrayAux[0];
		array_push($cont2, $arrayAux[1]);
	}
	if($fila2['valor3'] != "")
	{
		$arrayAux = split(":", $fila2['valor3']);
		$clave3=$arrayAux[0];
		array_push($cont3, $arrayAux[1]);
	}
	if($fila2['valor4'] != "")
	{
		$arrayAux = split(":", $fila2['valor4']);
		$clave4=$arrayAux[0];
		array_push($cont4, $arrayAux[1]);
	}

}
if(!empty($cont1)) $total[$clave1]=$cont1;
if(!empty($cont2)) $total[$clave2]=$cont2;
if(!empty($cont3)) $total[$clave3]=$cont3;
if(!empty($cont4)) $total[$clave4]=$cont4;

foreach (array_keys($total) as $key) {
	$media=0;
	$sensor=false;
	$cont=0;
	for ($i=0; $i < count($total[$key]); $i++)
	{
		if(!is_numeric($total[$key][$i]))
		{
			if($total[$key][$i] == "input")
			{
				$cont++;
				$actuador =  "<div class
```

```
= \"sensor\"><center>".$key." <input id=\"nombre_".
$fila['arduinoId']."-".$cont."\" type=\"hidden\" value=\"".
$key."\"><input class =\"nuevoValorEnviar\" type=\"text\"
id=\"valor_".$fila['arduinoId']."-".$cont."\" value=\"\"><div
onclick=\"enviarCambio('".$fila['arduinoId']."-".$cont."')\"
class=\"boton\">Enviar</div></center></div>";
                                        break;
                                }
                        }
                        else
                        {
                                $sensor =true;
                                $media = $media+$total[$key][$i];
                        }

                }
                if($sensor)
                {
                        echo "<div class = \"sensor\"><center>".$key."  ".
$media/count($total[$key])."</center></div>";

                }
                else
                {
                        echo $actuador;
                }
        }


        echo"</div>";
    }
}
;?>
```

## 8.8 nuevosDatos.php

```
<?php

$link = mysql_connect('localhost', 'root', '') or die('No se pudo
conectar: ' . mysql_error());
mysql_select_db('hermes') or die('No se pudo seleccionar la base de
datos');

$query = "SELECT * FROM `Arduino` WHERE `nombre`=\"\"";
$result = mysql_query($query) or die('Consulta fallida: ' .
mysql_error());
```

```
 echo "<form id=\"formAltaArduino\" onsubmit=\"return
validacion('".mysql_num_rows($result)."')\" ><fieldset>";
 $cont=0;
   while ($fila = mysql_fetch_assoc($result))
   {

       echo "<div class\"identificador\">El identificador asignado es ".
$fila["id"]."</div>";
       echo "<div class\"identificador\">El tipo es ".
$fila["tipo"]."</div>";
       echo "<div class\"identificador\">La ip es ".$fila["ip"]."</div>";
       echo "Añada un nombre para terminar registro";
        echo "<center>";
       echo "<input type=\"text\" id=\"nombre".$cont."\"
name=\"nombre".$cont."\" value=\"\">";
       echo "</center>";
       echo "<input type=\"hidden\" name=\"id".$cont."\" value=\"".
$fila["id"]."\">";
       echo "<br>";
       $cont++;
   }
   echo "<input type=\"hidden\" name=\"numero\"
value=\"".mysql_num_rows($result)."\">";
   echo "<input type=\"submit\"  value=\"Submit\"></fieldset>";
   echo "</form>";


;?>
```

## 8.9 nuevosElementos.php

```
<?php

$link = mysql_connect('localhost', 'root', '') or die('No se pudo
conectar: ' . mysql_error());
mysql_select_db('hermes') or die('No se pudo seleccionar la base de
datos');

$query = "SELECT COUNT(*) FROM `Arduino` WHERE `nombre`=\"\"";
$result = mysql_query($query) or die('Consulta fallida: ' .
mysql_error());
$line = mysql_fetch_array($result);
echo $line[0];

;?>
```

## 8.10 arduinoList.php

```
<?php

$link = mysql_connect('localhost', 'root', '') or die('No se pudo
conectar: ' . mysql_error());
mysql_select_db('hermes') or die('No se pudo seleccionar la base de
datos');

$query = "SELECT DISTINCT(`nombre`),`id` FROM `Arduino`";
$result = mysql_query($query) or die('Consulta fallida: ' .
mysql_error());

if (mysql_num_rows($result) != 0)
{
   while ($fila = mysql_fetch_assoc($result))
   {
      echo "<center>";
      echo "<div class=\"listado\" onclick =\"arduinoEspecifica('".
$fila['nombre']."')\"><p>".$fila['nombre']."</p></div>";
      echo "</center>";
   }
}
;?>
```

## 8.11 unicoModulo.php

```
<?php

$link = mysql_connect('localhost', 'root', '') or die('No se pudo
conectar: ' . mysql_error());
mysql_select_db('hermes') or die('No se pudo seleccionar la base de
datos');

$$sql = "SELECT * FROM `Arduino` a,`Mediciones` m WHERE
a.`id`=m.`arduinoId` AND a.`nombre`= '".$_GET['datos']."' ";
$result = mysql_query($query) or die('Consulta fallida: ' .
mysql_error());

if (mysql_num_rows($result) != 0)
{
   while ($fila = mysql_fetch_assoc($result))
```

```
   {
        $sql = "SELECT * FROM `Mediciones` WHERE `arduinoId`='".
$fila['arduinoId']."' ORDER BY `fecha` DESC LIMIT 5";
        $result2 = mysql_query($sql) or die('Consulta fallida: ' .
mysql_error());
        echo "<div class=\"moduloArduino\">";
        echo "<img src=\"img/termometro.jpg\" width=\"150\"
height=\"150\">";
        $cont1 =array();
        $clave1="";
        $cont2 =array();
        $clave2="";
        $cont3 =array();
        $clave3="";
        $cont4 =array();
        $clave4="";
        $total=array();
        while ($fila2 = mysql_fetch_assoc($result2))
        {
                if($fila2['valor1'] != "")
                {
                        $arrayAux = split(":", $fila2['valor1']);
                        $clave1=$arrayAux[0];
                        array_push($cont1, $arrayAux[1]);
                }

                if($fila2['valor2'] != "")
                {
                        $arrayAux = split(":", $fila2['valor2']);
                        $clave2=$arrayAux[0];
                        array_push($cont2, $arrayAux[1]);
                }
                if($fila2['valor3'] != "")
                {
                        $arrayAux = split(":", $fila2['valor3']);
                        $clave3=$arrayAux[0];
                        array_push($cont3, $arrayAux[1]);
                }
                if($fila2['valor4'] != "")
                {
                        $arrayAux = split(":", $fila2['valor4']);
                        $clave4=$arrayAux[0];
                        array_push($cont4, $arrayAux[1]);
                }

        }
        if(!empty($cont1)) $total[$clave1]=$cont1;
        if(!empty($cont2)) $total[$clave2]=$cont2;
        if(!empty($cont3)) $total[$clave3]=$cont3;
        if(!empty($cont4)) $total[$clave4]=$cont4;
```

```
        foreach (array_keys($total) as $key) {
                $media=0;
                $sensor=false;
                $cont=0;
                for ($i=0; $i < count($total[$key]); $i++)
                {
                        if(!is_numeric($total[$key][$i]))
                        {
                                if($total[$key][$i] == "input")
                                {
                                        $cont++;
                                        $actuador =  "<div class
= \"sensor\"><center>".$key." <input id=\"nombre_".
$fila['arduinoId']."-".$cont."\" type=\"hidden\" value=\"".
$key."\"><input class =\"nuevoValorEnviar\" type=\"text\"
id=\"valor_".$fila['arduinoId']."-".$cont."\" value=\"\"><div
onclick=\"enviarCambio('".$fila['arduinoId']."-".$cont."')\"
class=\"boton\">Enviar</div></center></div>";
                                        break;
                                }
                        }
                        else
                        {
                                $sensor =true;
                                $media = $media+$total[$key][$i];
                        }

                }
                if($sensor)
                {
                        echo "<div class = \"sensor\"><center>".$key."  ".
$media/count($total[$key])."</center></div>";

                }
                else
                {
                        echo $actuador;
                }
        }


        //echo "<div class = \"sensor\">Humedad</div>";

        echo"</div>";
   }
}
;?>
```

## 8.12 script.js

```
var refrescarAviso;
function nuevosElementos()
{
        $.ajax({
              url:   'nuevosElementos.php',
              type:  'get',
              success:  function (response) {
                if (response != 0)
                {
                        $("#aviso").css("display","inline");
                }


              }
    });
}

function darDeAlta()
{
    clearInterval(refrescarAviso);
        $("#aviso").css("display","none");
        $.ajax({
                url:   'nuevosDatos.php',
              type:  'get',
              success:  function (response) {
              $("#altaArduino").append(response);

              }
    });
        $("#altaArduino").css("display","inline");
        $("#altaArduino").animate({'width': "250px",});

}
function validacion(cont)
{
    for (var i = 0; i < cont; i++) {

     var ident = "#nombre"+i;

       if ($(ident).val()=="")
       {

          $(ident).css("border","2px solid #990000");
          return false;
       }
```

```
    }


    return true;
}

function cargarArduinos()
{
    $.ajax({
            url:   'arduinosList.php',
            type:  'get',
            success:  function (response) {
            $("#listaArduinos").append(response);

            }
    });


}
function enviarCambio(datos)
{

    var valor = $("#valor_"+datos).val();
    alert("el valor es "+valor);
    var nombre = $("#nombre_"+datos).val();
    $.ajax({
            data: 'datos='+datos+'&mod='+nombre+"&valor="+valor,
            url:   'mensajeArduino.php',
            type:  'get',
            success:  function (response) {
            alert(response);

            }
    });
}
function arduinoEspecifica(datos)
{
    $.ajax({
            data: 'datos='+datos,
            url:   'unicoModulo.php',
            type:  'get',
            success:  function (response) {
            $("#listaModulos").html(response);
            }
    });
}
function cargarModulos()
{
    $.ajax({
            url:   'modulosList.php',
```

```
            type:  'get',
            success:  function (response) {
            $("#listaModulos").append(response);

            }
    });


}
```