UNIVERSIDAD DE EL SALVADOR

FACULTAD DE INGENIERÍA Y ARQUITECTURA

ESCUELA DE INGENIERÍA ELÉCTRICA

Diseño y construcción de un data logger multiparámetro con comunicación vía internet.

PRESENTADO POR:

LUIS ALFREDO GÓMEZ CALIDONIO

WILLIAN ERICK HERNÁNDEZ IRAHETA

PARA OPTAR AL TÍTULO DE:

INGENIERO ELECTRICISTA

CIUDAD UNIVERSITARIA, MARZO DE 2013

UNIVERSIDAD DE EL SALVADOR

RECTOR :

ING. MARIO ROBERTO NIETO LOVO

SECRETARIA GENERAL :

DRA. ANA LETICIA ZAVALETA DE AMAYA

FACULTAD DE INGENIERÍA Y ARQUITECTURA

DECANO :

ING. FRANCISCO ANTONIO ALARCÓN SANDOVAL

SECRETARIO :

ING. JULIO ALBERTO PORTILLO

ESCUELA DE INGENIERÍA ELÉCTRICA

DIRECTOR :

ING. JOSÉ WILBER CALDERÓN URRUTIA

UNIVERSIDAD DE EL SALVADOR

FACULTAD DE INGENIERÍA Y ARQUITECTURA

ESCUELA DE INGENIERÍA ELÉCTRICA

Trabajo de Graduación previo a la opción al Grado de:

INGENIERO ELECTRICISTA

Título :

Diseño y construcción de un data logger multiparámetro con comunicación vía internet.

Presentado por :

LUIS ALFREDO GÓMEZ CALIDONIO

WILLIAN ERICK HERNÁNDEZ IRAHETA

Trabajo de Graduación Aprobado por:

Docente Director :

ING. JOSÉ WILBER CALDERÓN URRUTIA

CIUDAD UNIVERSITARIA, MARZO DE 2013

Trabajo de Graduación Aprobado por:

Docente Director :

ING. JOSÉ WILBER CALDERÓN URRUTIA

DEDICATORIA

A Dios Todopoderoso por permitirme conocer la verdad, y brindarme sabiduría, amor y paciencia, me ayuda en los momentos más difíciles brindándome valores que me fortalecen no solo académicamente, si no como persona.

A mis padres José Santana Hernández y María Teresa Iraheta puesto que me brindaron apoyo y fortaleza en el desarrollo y transcurso de este, ayudándonos a concluir satisfactoriamente nuestro proyecto.

A mis hermanos, Norma, Maritza, Roxana, Erika y José por estar siempre conmigo como familia, la mejor que un ser humano puede tener, por su apoyo incondicional en todo momento.

A mi esposa Marina Isabel Soriano por brindarme ese apoyo incondicional cada día, y hacer que mi vida sea maravillosa siempre, te amo con todo mi corazón.

A mi hija Meylin Nicole Hernández Soriano, mi chiquita hermosa que papito Dios nos ha prestado, eres una razón especial para salir adelante frente a todos los obstáculos que se presentan cada día, te dedico este trabajo con todo mi amor y cariño.

A toda mi familia y amigos, por estar conmigo siempre de una o de otra manera, por el apoyo, por sus buenos consejos, por estar ahí cuando los necesito.

A todos ustedes los quiero mucho, y que Dios les bendiga siempre.

Willian Hernández

DEDICATORIA

Al finalizar un trabajo tan arduo y lleno de dificultades es inevitable caer en el egocentrismo que me lleva a concentrar la mayor parte del mérito en el aporte que he hecho. Sin embargo, el análisis objetivo me muestra inmediatamente que la magnitud de ese aporte hubiese sido imposible sin la participación de personas e incluso instituciones que han facilitado las cosas para que este trabajo llegue a un feliz término. Por ello, es para mí un verdadero placer utilizar este espacio para ser justo y consecuente con ellas, expresándoles mis agradecimientos

A Dios.
Por haberme permitido llegar hasta este punto, haberme dado salud para lograr mis objetivos y por haber puesto en mi camino a aquellas personas que han sido mi soporte y compañía durante todo el periodo de estudio.

A mis padres.
Por darme la vida y nunca dejar de creer en mi que aunque las limitaciones económicas no les permitieron apoyarme económicamente, me apoyaron moralmente y con sus oraciones.

A mi esposa.
Marina del Carmen Zeceña, por siempre estar a mi lado, brindándome todo su amor, entrega, dedicación y conocimiento y sobre todo tenerme mucha comprensión y paciencia durante estos años de mi vida y quien ha sido una pieza clave en mi desarrollo profesional. Mil gracias porque siempre estas a mi lado sin condiciones.

A mi hija.
Karen Melissa Gómez, que vino hacer el más grande motivo para luchar, sin importar que tan cansado, desvelado o sin ánimos me sienta ella siempre me llena de alegría.

Un agradecimiento muy especial a mi tía Esperanza Calidonio y al Ing. José Alfredo Aguilar, que me apoyaron tanto moral como económicamente justo cuando más lo necesitaba.

A toda mi familia y amigos, por estar conmigo siempre de una o de otra manera, por el apoyo, por sus buenos consejos, por estar ahí cuando los necesito.
Que dios los bendiga siempre

Luis Gómez

Contenido

Índice de figuras

Índice de tablas

Índice de códigos

Introducción

Algunas veces se requiere medir ciertas variables físicas presentes en un determinado ambiente, para llevar un control detallado a intervalos de tiempo del comportamiento de dichas variables dentro del sistema, para ello existen herramientas electrónicas capaces de censar estas variables físicas y almacenarlas en una memoria extraíble para que puedan ser procesadas como se desee, a estas herramientas se les denomina DATA LOGGER.

A groso modo, un Data Logger es un dispositivo electrónico que registra mediciones ordenadas en el tiempo, provenientes de diferentes sensores. Luego cada medición es almacenada en una memoria, junto con su respectiva fecha y hora.

En general los Data Loggers son pequeños, y alimentados por baterías, y están conformados por un microprocesador, una memoria para el almacenamiento de los datos y diferentes sensores.

En el presente trabajo se ha implementado uno de estos dispositivos Data Logger, utilizando hardware y software Arduino los cuales son de libre distribución, tomando en cuenta determinadas características que hacen que el aparato tenga aplicación relevante y a un costo relativamente bajo.

Para cumplir con los estándares el Data Logger debe ser portable, tener un bajo costo, tener intervalos de muestreo programables, tener una buena capacidad de almacenamiento de datos, disponibilidad y bajo costo en los componentes y tener una interface con el usuario.

Mas que un Data Logger, hemos implementado un sistema de monitoreo vía web, en el cual se utiliza un dispositivo que censa las variables físicas influyentes en un determinado ambiente a intervalos de tiempo (DATA LOGGER), cuyo objetivo principal es enviar estas lecturas a un servidor WEB externo, para que sean presentadas al usuario mediante aplicaciones web, ya sea gráficos, tablas, descarga de datos, etc. Además si en algún momento no es posible la conexión con este servidor externo, el sistema Data Logger es capaz de levantar un servidor web de forma más básica para presentar los datos censados en ese preciso momento en el

que dura la falla, además de respaldar estos valores salvando cada cierto tiempo en una tarjeta MicroSD; en el momento de recuperar la conexión con el servidor externo, el sistema se encarga de enviar los datos almacenados en la memoria MSD al servidor externo, para que puedan almacenarse en la base de datos, y puedan ser presentados al usuario cuanto sea necesario.

Cuando el usuario lo desee puede descargar los datos censados por fecha y hora desde la web de monitoreo, en formato de hoja de cálculo, para verificar el comportamiento del ambiente y realizar proyecciones o para lo que fuere necesario.

En el sistema hay muchos parámetros importantes que son configurables por el administrador mediante un teclado y una pantalla LCD, parámetros como tipos de sensores conectados, tiempos de muestreo, etc. Se conecta a la red y se auto configura aunque si el usuario lo desea puede configurar los parámetros de red manualmente.

I. Generalidades

1.1. Aplicaciones de los data loggers

Los Data Loggers pueden ser construidos para controlar todo tipo de datos ambientales. La temperatura y la humedad son las más comunes.
Según sea la necesidad pueden servir para otras aplicaciones. Por ejemplo han sido utilizados por muchos años en los servicios meteorológicos para medir la humedad, presión atmosférica, los niveles de precipitación, etc.
También un Data Logger puede ser transportado, por ejemplo, para registrar la temperatura que hay dentro del contenedor de un camión de carga desde una planta hasta el centro de distribución.

1.2. Características principales

Para desarrollar este proyecto se plantearon una serie de características que debía cumplir el sistema, para ser competitivo frente a productos similares existentes comercialmente. Este dispositivo por tanto debe:

- Ser portátil, es decir, funcionar con baterías y tener un peso y tamaño que le permita ser transportado con facilidad.
- Tener un bajo consumo, hay que tener en cuenta que este tipo de sistemas pueden trabajar durante semanas, meses y hasta años, por lo tanto es muy importante tomar en cuenta el consumo nominal y la capacidad de las baterías.
- Intervalos de muestreo configurables y con la mayor flexibilidad posible, desde segundos hasta horas. Esto permite registrar variables con diferente velocidad de cambio respecto al tiempo.
- Tener una buena capacidad de almacenamiento de datos. En este punto entran en juego las características anteriores, por lo tanto hay que determinar una cierta cantidad de memoria, teniendo en cuenta que la duración de las baterías depende del consumo, y el tiempo de trabajo dependerá del intervalo entre muestras y la capacidad de memoria.
- Costo bajo de los componentes y disponibilidad. Es muy importante ya que para ciertas aplicaciones se podrían necesitar varios equipos, o bien, puedan ser utilizados en lugares donde se corra el riesgo de ser destruidos.
- Una interface con el usuario a través de teclado, pantalla y una PC, donde el usuario o administrador puede configurar y leer el dispositivo de una manera sencilla y rápida, la configuración se hace por medio de una

pantalla LCD y un teclado para la introducción de datos de red, hora, descripción, etc., para leer los datos se está haciendo uso de un servidor web al cual el Data Logger está enviando los datos cada cierto tiempo para presentarlos al usuario mediante una página web que podría ser vista desde cualquier lugar del mundo.

1.3. Materiales y métodos

Especialmente son tres las partes principales que componen a un Data Logger: microcontrolador, memoria y sensores. El microcontrolador es la parte fundamental del sistema, pero también la más costosa. Para este desarrollo se utilizó el microcontrolador AVR ATMEGA2560 con un amplio espacio de memoria para programar y corriendo a 16Mhz. Este es el controlador de la placa Arduino Mega 2560, la cual estamos utilizando en este proyecto por ser de bajo costo y de libre distribución.

La memoria de datos utilizada es del tipo MicroSD la cual está integrada al modulo Ethernet Arduino el cual está siendo usado en este trabajo para la salida de datos a la red global.

Los sensores son la otra pieza fundamental, en este caso el sistema no responde a un solo tipo de sensor. Ofrece al usuario la oportunidad de implementar el tipo de sensor que sea de utilidad para una aplicación especifica que puede programar en un sistema operativo Linux en una PC, en el software Arduino se escribe el código que se va a cargar al controlador mediante una conexión USB entre el ordenador y el Data Logger.

Para leer el dispositivo se ha implementado un servidor web externo al Data Logger, al cual se envían los datos censados en tiempo real, mediante un modulo Ethernet, para ser presentados al usuario mediante un explorador web, además el Data Logger incorpora un servidor web interno como respaldo, en caso de que el servidor externo no pueda ser accedido.

1.4. Funcionamiento general

El usuario mediante un teclado y una pantalla LCD configura al equipo en cuanto a la cantidad (hasta 12) y tipo de sensores, intervalos de muestreo (desde 5 segundos hasta un máximo considerable) y el tiempo de operación, en este último se puede configurar la fecha y hora de inicio y finalización de la toma de muestras, además se configura el tiempo de envió de datos al servidor y de almacenamiento.

La capacidad de almacenamiento de las muestras no es un parámetro que se considere critico, ya que las muestras se almacenan en la base de datos del servidor externo el cual está corriendo sobre un ordenador con una capacidad de almacenamiento de Gigabytes, por tanto no se podría determinar el límite de lecturas que soporte la memoria, es suficiente decir que son muchas.
Sin embargo cuando el servidor externo no está accesible desde el Data Logger, éste cuenta con una memoria de 512 Megabytes para almacenar las lecturas tomadas durante el tiempo de falla, la cual es capaz de almacenar poco más de 400,000 lecturas.
La hora en el Data Logger se carga desde la PC cuando este se programa con el software, una vez configurado el Data Logger, comienza con la ejecución de tareas. Mientras esta en operación el Data Logger informa al usuario mediante la pantalla LCD si este está en modo CLIENTE, es decir, enviando datos al servidor externo, o en modo SERVIDOR, es decir, que está levantado el servidor web desde el Data Logger.
El usuario puede leer los datos recopilados en cualquier momento, mediante la página web, aunque el equipo se encuentre trabajando, ya que el monitoreo es en tiempo real.
Además es posible leer y calibrar cada sensor desde servidor web externo, como administrador y agregar nuevos sensores indicando su ecuación de transferencia.

1.5. Definiciones y conceptos básicos

1.5.1. Definición y estandarización de conceptos

El Data Logger implementado en este proyecto se realizo con criterio de diseño propio, es decir, no se tomo como referencia o modelo otro equipo diseñado para el mismo fin, por lo tanto, comercialmente pueden existir muchas definiciones referente a partes o funciones que hayan sido utilizadas en la realización de este proyecto, por esto es importante estandarizar los conceptos más generales que se han utilizado en este trabajo para evitar confusión al lector.
Para efecto de aplicación en los capítulos a seguir, se utilizarán los siguientes conceptos básicos:

- **Data Logger –** Sistema autónomo de adquisición de datos portátil, es decir, es un dispositivo electrónico que registra mediciones ordenadas en el tiempo, provenientes de diferentes sensores y las almacenas por hora y fecha de adquisición.
- **Arduino -** Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar.

- **Software Arduino –** Es un entorno de código abierto que hace fácil escribir código y cargarlo a la placa E/S. Funciona en Windows, Mac OS X y Linux. El entorno está escrito en Java y basado en Processing, avr-gcc y otros programas también de código abierto.
- **Modulo Ethernet –** Es un modulo que permite a una placa Arduino conectarse a internet. Está basada en el chip Ethernet Wiznet W5100, el cual provee de una IP y es capaz de soportar los protocolos TCP y UDP.
- **Memoria SD –** Es un formato de tarjeta de memoria flash más pequeña que la MicroSD, desarrollada por SanDisk; adoptada por la Asociación de Tarjetas SD bajo el nombre de MicroSD.
- **Servidor –** es una computadora que forma parte de una red que puede ser local o externa y que se encarga de brindar servicios a otras computadoras denominadas clientes.
- **Sensor -** Es un dispositivo capaz de detectar magnitudes físicas o químicas, y transformarlas en variables eléctricas.
- **RTC –** Reloj en tiempo real (en inglés, real-time clock, RTC), es un reloj de un dispositivo, incluido en un circuito integrado, que mantiene la hora actual.
- **Jumper –** En este trabajo, son los cables que interconectan a la placa Arduino con los diferentes dispositivos que forman parte del Data Logger, ya sea sensores, RTC, pantalla LCD, teclado, etc.
- **Dirección IP -** es una etiqueta numérica que identifica, de manera lógica y jerárquica, a un interfaz de un dispositivo dentro de una red que utilice el protocolo IP (Internet Protocol), que corresponde al nivel de red del protocolo TCP/IP.
- **Mascara de red -** Es una combinación de bits que sirve para delimitar el ámbito de una red de computadoras. Su función es indicar a los dispositivos qué parte de la dirección IP es el número de la red, incluyendo la subred, y qué parte es la correspondiente al host.
- **Gateway –** Puerta de enlace en español, es la dirección IP asignada al enrutador dentro de una red. Y se utiliza para indicarle al host hacia donde enviar el tráfico de red.
- **Framework –** Define estándares, reglas, criterios, etc. Con el fin de facilitar la solución de un problema, ejemplos de esto son Rails, Jquery y Bootstrap, todos ellos usados en el desarrollo del servidor web.

1.5.2. Historia, funcionalidad y tipo de arduino usado

1.5.2.1. Historia de arduino

Vamos a exponer un poco acerca de la historia de Arduino, para que conozcan un poco de donde viene este maravilloso prototipo con el cual se es capaz de realizar una gran cantidad de proyectos interesantes, resumimos a continuación un documento de la revista IEEE Spectrum.

Arduino nació como un proyecto educativo en el año 2005 sin pensar que algunos años más tarde se convertiría en líder del mundo DIY (Do It Yourself). Su nombre viene del nombre del bar Bar di Re Arduino donde Massimo Banzi pasaba algunas horas, el cual a su vez viene del nombre de un antiguo rey europeo allá por el año 1002. Banzi dice que nunca surgió como una idea de negocio, es más nació por una necesidad de subsistir ante el eminente cierre del Instituto de diseño Interactivo IVREA en Italia. Es decir, al crear un producto open hardware (de uso público) no podría ser embargado. Es más hoy en día Arduino tiene la difícil tarea de subsistir comercialmente y continuar en continuo crecimiento. A la fecha se han vendido más de 250 mil placas en todo el mundo sin contar las versiones clones y compatibles.

Para su creación participaron alumnos que desarrollaban sus tesis como Hernando Barragan (Colombia) quien desarrollo la plataforma de programación Wiring con la cual se programa el microcontrolador. Hoy en día con Arduino se pueden fabricar infinidad de prototipos y cada vez su uso se viene expandiendo más. Desde cubos de leds, sistemas de automatización en casa (demótica), integración con el Internet, displays Twitter, kit analizadores de ADN.

Google ha apostado por el proyecto y ha colaborado en el Android ADK (Accesory Development Kit), una placa Arduino capaz de comunicarse directamente con spmartphons Android para obtener las funcionalidades del teléfono (GPS, acelerómetros, GSM, a bases de datos) y viceversa para que el teléfono controle luces, motores y sensores conectados de Arduino.

El primer prototipo fue desarrollado en el instituto IVRAE pero aún no se llamaba Arduino.

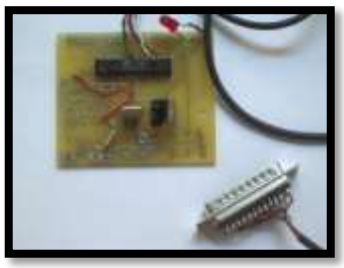

Figura 1. Primer prototipo Arduino.

Para la producción en serie de la primera versión se tomaron en cuenta algunas consideraciones: Economía (menor a 30 Euros), debía ser Plug and Play, utilizaron el color azul para marcar una diferencia con las placas convencionales, trabajar en todas las plataformas (Mac, Windows y Linux). La primera producción fue de 300 unidades y se las dieron a los alumnos del Instituto IVRAE, (las ganancias fueron de sólo 1 dollar por placa), con el fin de que las probaran y empezaran a diseñar sus primeros prototipos. Uno de los primeros proyecto fue un reloj alarma, el cual no se apagaría hasta que no te pararas de la cama.

Tom Igoe, profesor y padre de la computación física se unió al proyecto luego que se enterara del mismo a través de la web. El ofreció su apoyo para desarrollar el proyecto a grandes escalas.
Hoy por hoy Arduino te permite crear cosas por ti mismo. Varias universidades como Standford y Carnegie Mellon y el MIT usan Arduino en sus campus. En la feria Maker Fair del 2011 se presentó la primera placa Arduino 32 Bit para trabajar tareas más pesadas. Entre ellas se presentó la impresora en 3D de MakerBot capaz de de imprimir en resina cualquier modelo en 3D.

1.5.2.2. Funciones de la placa arduino

Arduino puede tomar información del entorno a través de sus pines de entrada de toda una gama de sensores y puede afectar aquello que le rodea controlando luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino (basado en Processing). Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un ordenador, si bien tienen la posibilidad de hacerlo y comunicar con diferentes tipos de software.

Las placas pueden ser hechas a mano o compradas montadas de fábrica; el software puede ser descargado de forma gratuita. Los ficheros de diseño de referencia (CAD) están disponibles bajo una licencia abierta, así pueden ser adaptadas a las necesidades.

1.5.2.3. Arduino mega

Existen muchos modelos de Arduino en el mercado, sin embargo, se determino que el Arduino MEGA era el más adecuado para este proyecto, por sus características. Es importante conocer estas detalladamente, es por eso que a continuación se presentan las principales.

Figura 2. Arduino MEGA

El Arduino Mega es una placa que posee un microcontrolador ATmeg1280. Tiene 54 entradas/salidas digitales (de las cuales 14 proporcionan salida PWM), 16 entradas digitales, 4 UARTS (puertos serie por hardware), un cristal oscilador de 16MHz, conexión USB, entrada de corriente, conector ICSP y botón de reset. Contiene todo lo necesario para hacer funcionar el microcontrolador. Simplemente necesita ser conectado al ordenador con el cable USB o alimentado con un trasformador o batería para empezar a trabajar. El Mega es compatible con la mayoría de shields diseñados para el Arduino Duemilanove o Diecimila.

Tabla 1. Especificaciones Arduino mega

Microcontrolador	ATMEGA 1280
Voltaje de funcionamiento	5 V
Volteje de entrada	7 – 12 V
Voltaje de entrada limite	6 – 20 V
Pines E/S digitales	54
Pines de entrada analógica	16
Intensidad por pin	40 mA
Intensidad en pin 3.3 V	50 mA
Memoria flash	128 KB
SRAM	8 KB
EEPROM	4KB
Velocidad de reloj	16 MHZ

II. Diseño, construcción y configuración del Data Logger

2.1. Criterios de construcción del Data Logger

Un Data Logger es un dispositivo que obtiene valores de variables físicas de un ambiente mediante un sensor, y las almacena en una memoria, para que estos datos puedan ser procesados luego por el usuario, sin embargo, este trabajo no solo comprende el diseño de un Data Logger como tal, más bien es un sistema de monitoreo vía web, por lo tanto, el diseño es un poco más complejo.

Para que este equipo sea competitivo frente a productos similares en el mercado, se plantearon una serie de características que el sistema debe cumplir, enunciadas a continuación.

- ✓ Debe de ser portátil.
- ✓ Tener un bajo consumo de energía.
- ✓ Con intervalos de muestreo programables.
- ✓ Buena capacidad de almacenamiento.
- ✓ Costos de fabricación y mantenimiento muy bajo.
- ✓ Una interface con el usuario para configurar y leer el dispositivo.

Como punto de partida solo se tenía claro que se iba a trabajar con la placa Arduino y algunos de sus módulos, considerando la capacidad de cada uno de ellos se eligió la placa Arduino mega como controlador o cerebro del dispositivo, esta placa posee suficientes pines de Entrada analógica y E/S digitales, comunicación USB con la PC, y es compatible con casi todos los módulos Arduino.

2.1.1. Partes del sistema

- Arduino MEGA, es el cerebro del Data Logger, es el encargado de manejar los dispositivos de entrada y salida, es decir, es el que se encarga de la adquisición de datos, controlar una pantalla LCD y un teclado, en fin todos los dispositivos conectados a este.
- Modulo Arduino Ethernet Shield, se encarga de transmitir los datos desde la placa Arduino hacia internet (servidor web externo), cuando el servidor web externo no está accesible este carga una página web básica y muestra los datos censados en tiempo real. Además el modulo Ethernet posee soporte para un tarjeta de memoria MicroSD con capacidad hasta de 2 GB probado, sin embargo se está usando una memoria de 512 MB, suficiente para almacenar una importante cantidad de muestras.
- RTC, el reloj en tiempo real se encarga de proporcionarle la fecha y hora al Data Logger, la fecha y hora se carga al RTC cuando el Data Logger es

programado desde el entorno de programación Arduino, el dispositivo toma la hora de la PC desde donde se está programando.

- Pantalla LCD, la pantalla tiene la función de informar al usuario de los procesos de configuración necesarios para la operación adecuada del Data Logger, esto mediante mensajes cortos que solicitan los parámetros adecuados de operación. Se está usando una pantalla LCD de 84 x 48 pixeles del tipo LCD 5110.
- Teclado 4x3, este es un teclado sumamente básico que consta de 12 teclas, números del 1 al 9 y el 0, el asterisco "*" utilizado como "," y la tecla numeral "#" utilizada como ENTER para procesar una orden de configuración. Es el encargado de pasar los parámetros de configuración solicitados al usuario.
- Sensores, son los encargados de detectar magnitudes físicas o químicas y transformarlas en variables eléctricas, cuando el Data Logger lo solicite, los sensores utilizados son, sensor de temperatura analógico LM35, sensor de temperatura digital DS18B20, sensor de humedad y temperatura digital DHT22, y un sensor que incorpora un anemómetro un pluviómetro y que además mide la dirección del viento.
- Conectores hembra 3.5 mm, para conectar los sensores a la placa Arduino.

Cables o jumper, estos son utilizados para interconectar físicamente algunas de las partes dentro del Data Logger.

2.1.2. Tamaño del data Logger

Uno de los requerimientos esenciales de construcción del Data Logger es la portabilidad que el equipo ofrece, para transportarlo hasta los lugares donde es necesario realizar las mediciones de un ambiente en particular, y para cumplir con este detalle, es necesario que el dispositivo sea lo suficientemente pequeño y liviano, de tal manera que pueda ser instalado en espacios pequeños.

Las partes del Data Logger se han acoplado eficientemente dentro de una caja metálica, evitando usar mucho espacio procurando que las partes no queden ni muy sobradas ni muy ajustadas; las dimensiones del Data Logger son por tanto de:

- ✓ 10 centímetros de ancho
- ✓ 15 centímetros de largo
- ✓ 5 centímetros de alto

Dimensiones que son aceptables comparadas con un dispositivo similar en el mercado.

2.1.3. Características y funcionamiento de los sensores

- **Sensor de temperatura analógico LM35**

La serie de circuitos integrados LM35 son sensores de precisión de temperatura, cuya tensión de salida es linealmente proporcional a la temperatura en grados Celsius (centígrados). El LM35 tiene así una ventaja sobre los sensores de temperatura lineales calibrado en Kelvin, cuando el usuario no quiere acarrear una constante de tamaña no considerable cuando se los grados Kelvin para convertir a Celcius. El LM35 no requiere ninguna calibración externa, precisiones típicas de grados +- 1/4 °C a temperatura ambiente y +- 3/4 °C sobre un rajo de -55 a +150 °C.

Figura 3. Sensor de temperatura LM35

El bajo costo está asegurado. El LM35 es de baja impedancia de salida, salida lineal, y la calibración precisa para hacer interfaz con lectura o circuitos de control especialmente. Se puede usar con fuentes de alimentación individuales, con sólo que suministre 60 microA de corriente, tiene muy bajo calentamiento, menos del 0,1 °C en aire libre. El LM35 es óptimo para funcionar en un rango de-55 °C a +150 °C de temperatura. La serie LM35 esta disponibles en paquetes herméticos con transistores empaquetados tipo TO-46.

Características:

- ✓ Calibrado directamente en grados Celsius (centígrados)
- ✓ Un factor de escala Lineal de 10,0 mV / °C
- ✓ Precisión garantizada de 0.5 °C (a +25 °C)
- ✓ Con el rango completo de temperatura - 55 °C a + 150 °C
- ✓ Adecuado para aplicaciones remotas
- ✓ De bajo costo.
- ✓ Opera 4 a 30 voltios
- ✓ Menos de 60 mA de corriente de fuga
- ✓ Bajo calentamiento 0.08 °C en aire libre
- ✓ No linealidad sólo en 1/ 4 °C típico.
- ✓ Salida de baja impedancia, 0.1 ohms para 1 mA de carga

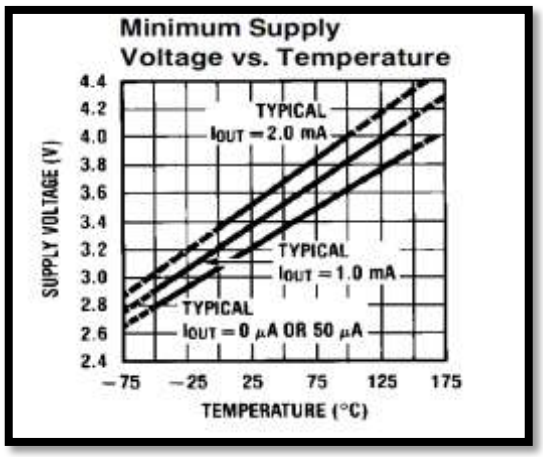

Figura 4. Curva de voltaje vrs. Temperatura de salida

- **Sensor de temperatura digital DS18B20**

El termómetro digital de temperatura en grados Celcius DS18B20 ofrece una medición de 9-bit a 12-bit de precisión y tiene una función de alarma no volátil programada por el usuario con los puntos de activación superior e inferior. El DS18B20 se comunica a través de un bus 1-Wire, que por definición, requiere solamente una línea de datos (y de tierra) para la comunicación con un microprocesador. Tiene un rango de operación de temperatura que va desde -55 ° C a +125 ° C y tiene una precisión de ± 0,5 ° C en el rango de -10 ° C a +85 ° C.

Además, el DS18B20 puede obtener energía directamente desde la línea de datos ("corriente parásita"), eliminando la necesidad de una fuente de alimentación externa.

Cada DS18B20 tiene un único código de 64-bit serie, lo que permite que múltiples DS18B20 puedan funcionar en el mismo bus 1-Wire. Por lo tanto, es sencillo de utilizar un microprocesador para controlar muchos DS18B20 distribuidos sobre un área grande. Las aplicaciones que pueden beneficiarse de esta característica incluyen control ambiental, monitoreo de temperatura sistemas dentro de los edificios, equipos o maquinaria, y el proceso de seguimiento y control sistemas.

Figura 5. Sensor de temperatura digital DS18B20

Características:

- ✓ Requiere sólo un cable para la comunicación por un solo pin (1-wire)
- ✓ Cada dispositivo tiene un único código de 64 bits Serial almacenado en ROM.
- ✓ Capacidad Multipunto, Simplifica Aplicaciones Distribuyendo Sensores de temperatura.
- ✓ No requiere componentes externos
- ✓ Puede ser alimentado por línea de datos; fuente de alimentación cuyo rango es de 3.0V a 5.5V-
- ✓ Mide temperaturas de -55 ° C a +125 ° C (-67 ° F a +257 ° F)
- ✓ ± 0,5 ° C de precisión en el rango de -10 ° C a +85 ° C
- ✓ Resolución es seleccionable por el usuario 9 a 12 Bits
- ✓ Convierte la temperatura en una Digital Word de 12-Bit en 750 ms (máx.)

- **Sensor de humedad y temperatura digital DHT22**

Sensor de humedad y temperatura con salida digital, el DHT22 con salida calibrada de la señal digital. Utiliza la exclusiva señal digital-collecting-technique y tecnología de detección de la humedad, asegurando su fiabilidad y estabilidad. Este elemento de detección está conectado con 8-bits a un solo chip.

Cada sensor de este tipo es compensado y calibrado en una cámara de calibración precisa con un coeficiente de calibración almacenado en la memoria OTP, cuando el sensor está censando, se afecta cada lectura con el coeficiente de la memoria.

Bajo consumo y tamaño pequeño y distancia de transmisión larga (20m), permite que se adapte a todo tipo de ocasiones toscas o en condiciones extremas.

Empaquetado con cuatro pines, haciendo la conexión muy conveniente.

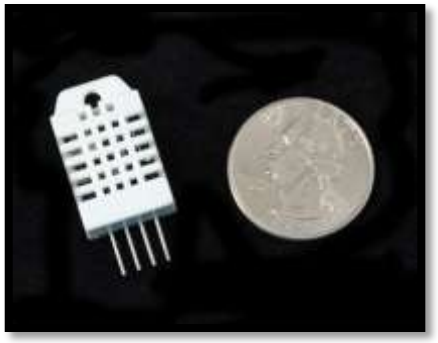

Figura 6. Sensor de humedad digital DHT22

Características:

- ✓ Amplia gama de temperatura compensada
- ✓ Mide humedad relativa y temperatura
- ✓ Señal digital calibrada
- ✓ Sobresaliente estabilidad a largo plazo
- ✓ No necesita componentes extras
- ✓ Larga distancia de transmisión
- ✓ Bajo consumo
- ✓ Empaquetado con 4 pines que pueden ser removidos según la aplicación.

Especificaciones técnicas:

- ✓ Modelo: AM2303
- ✓ Fuente de alimentación: 3.3-6V DC
- ✓ Señal de salida: señal digital a través de un solo cable
- ✓ Elemento de detección: humedad a través de condensadores Polímeros y un DS18B20 para detectar la temperatura.
- ✓ Rango de Medición: 0-100% de humedad relativa, temperatura - 40 a +125 Celsius

- **Resistencia variable potenciómetro**

Figura 7. Potenciómetro (resistencia variable)

Un potenciómetro es un componente electrónico similar a los resistores pero cuyo valor de resistencia en vez de ser fijo es variable, permitiendo controlar la intensidad de corriente a lo largo de un circuito conectándolo en paralelo ó la caída de tensión al conectarlo en serie.

Un potenciómetro está compuesto por una resistencia de valor total constante a lo largo de la cual se mueve un cursor, que es un contacto móvil que divide la

resistencia total en dos resistencias de valor variable y cuya suma es la resistencia total, por lo que al mover el cursor una aumenta y la otra disminuye. A la hora de conectar un potenciómetro, se puede utilizar el valor de su resistencia total o el de una de las resistencias variables ya que los potenciómetros tienen tres terminales, dos de ellos en los extremos de la resistencia total y otro unido al cursor.
Se pueden distinguir varios tipos de potenciómetros.

- Según la forma en la que se instalan: para chasis o para circuito impreso.
- Según el material: de carbón, de alambre ó de plástico conductor.
- Según su uso: de ajuste, normalmente no accesibles desde el exterior, ó de mando, para que el usuario pueda variar parámetros de un aparato, estos a su vez pueden ser: rotatorios, se controlan girando su eje, deslizantes, cuya pista resistiva es recta y el cursor cursor se mueve en línea recta ó múltiples.
- Según su respuesta al movimiento del cursor pueden ser: lineales, logarítmicos, sinusoidales y anti logarítmicos.
- Potenciómetros digitales: son circuitos integrados con un funcionamiento similar a un potenciómetro analógico.

Los usos más comunes del potenciómetro son los referidos a al control de funciones de equipos eléctricos, como el volumen en los equipos de audio y el contraste ó el brillo en la imagen de un televisor.
En este caso se esta usando para simular cualquier variable que afecte a un sistema determinado, es decir, se varia la resistencia en el potenciómetro para simular los cambios de una variable "x" representada por la caída de tensión en el potenciómetro, de esta forma se obtiene una prueba rápida que el sistema esta monitorizando en tiempo real.

- **Estación meteorológica (veleta anemómetro y Pluviómetro).**

La estación meteoróloga consta de tres sensores, uno para medir la dirección del viento, otro para medir la velocidad del viento y una mas para medir la cantidad de lluvia precipitada durante una tormenta, para conocer un poco acerca de las características de cada uno de estos dispositivos se expone a continuación un poco de lo que revela la hoja de datos de este sistema de monitoreo meteorológico comercializado por sparkfun.
El kit incluye veleta, anemómetro de cazoletas y una cubeta tipo bascula como pluviómetro, y con soportes para el montaje incluido. Estos sensores no contienen componentes electrónicos activos, en su lugar de utilizan láminas magnéticas cerrando interruptores e imanes para tomar medidas. Un voltaje de alimentación debe estar suministrado a cada instrumento para producir una salida. Sobre un brazo plástico se montan los sensores de viento, veleta y anemómetro sobre otro

brazo se apoya también el anemómetro, estas dos piezas plásticas van sujetadas a un mástil de metal. Un cable corto conecta a los dos sensores de viento. Los tornillos se proporcionan para fijar los sensores en el brazo.
El sensor de lluvia puede estar montado en el mástil inferior utilizando su propio brazo de montaje y el tornillo, o puede ser montado de forma independiente. Pluviómetro.

Figura 8. Estación meteoróloga.

El pluviómetro es un tipo de auto-vaciamiento de balancín. Cada 0,011" (0,2794 mm) de lluvia provoca un cierre de contacto momentáneo que puede registrarse con un contador digital o entrada de interrupción del microcontrolador. El interruptor está conectado a los dos conductores centrales de los conectores RJ11. La medida con el anemómetro para censar la velocidad del viento se hace mediante el cierre de un contacto que se cierra magnéticamente con un imán a medida las cazoletas giran. Una velocidad de viento de 1,492 MPH (2,4 km/h) hace que el interruptor se cierre una vez por segundo.
El interruptor del anemómetro está conectado a los dos conductores internos del cable RJ11 compartida por la veleta (pines 2 y 3).

La veleta es la más complicada de los tres sensores. Tiene ocho interruptores, cada uno conectado a una resistencia diferente. La paleta del imán puede cerrar dos interruptores a la vez, lo que permite tener hasta 16 diferentes posiciones como puede verse en la figura siguiente.

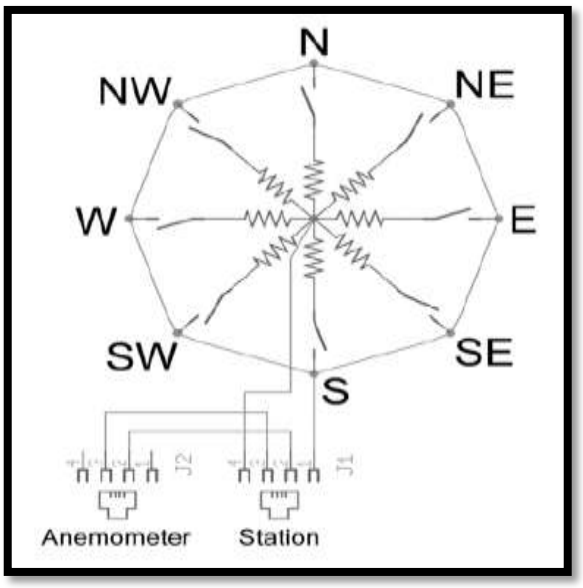

Figura 9. Posiciones que puede tomar la veleta.

Una resistencia externa se puede utilizar para formar un divisor de voltaje, produciendo una salida de tensión que se puede medir con un convertidor analógico a digital, como se muestra a continuación.

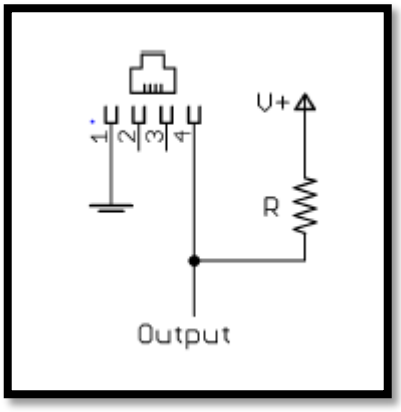

Figura 10. Divisor de tensión para obtener las lecturas de la dirección del viento.

Puede consultar la tabla 3 para visualizar los valores de resistencia para todas las posibles posiciones (16). Los valores de resistencia para las posiciones entre los que se muestran en el diagrama son el resultado de dos resistores adyacentes conectados en paralelo cuando el imán de la veleta activa dos interruptores simultáneamente.

2.1.4. Circuito impreso (PCB)

Fue necesario diseñar un circuito impreso para montar todos los dispositivos físicos que componen al Data Logger, es decir, jacks de 3.5 mm, integrados, resistencias, pines de conexión entre E/S y pines de placa Arduino etc., esto con el fin de que el dispositivo tenga muy buena presentación, y así evitar las conexiones con cables o "jumper" como normalmente se hacen en los proyectos con Arduino.

Tomando en cuenta las dimensiones que tiene el Data Logger, el circuito impreso fue diseñado así:

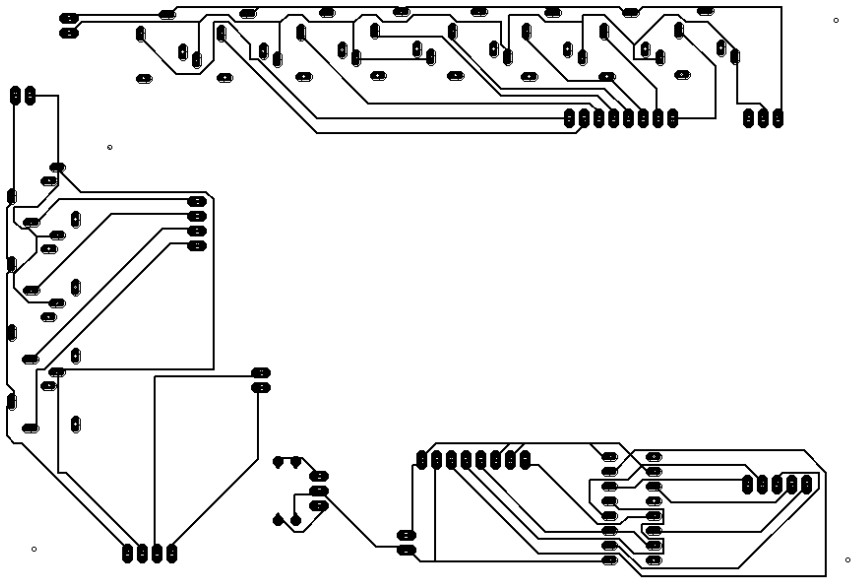

Figura 11. Circuito impreso Data Logger.

En la figura anterior solo se observan las pistas o conexiones hacia los diferentes dispositivos utilizados, sin embargo, puede ver algunas imágenes fotográficas en el anexo D (Diseño y construcción). Además puede consultar el archivo PCB.sch que se encuentra en el CD anexo y abrirlo con el programa EAGLE, para ver todos los detalles.

2.2. Esquema del sistema

La placa Arduino mega controla todas las operaciones que se realizan en el Data Logger, por tanto, todos las partes están conectadas a esta placa, sensores, pantalla, teclado, modulo Ethernet, alimentación, etc.

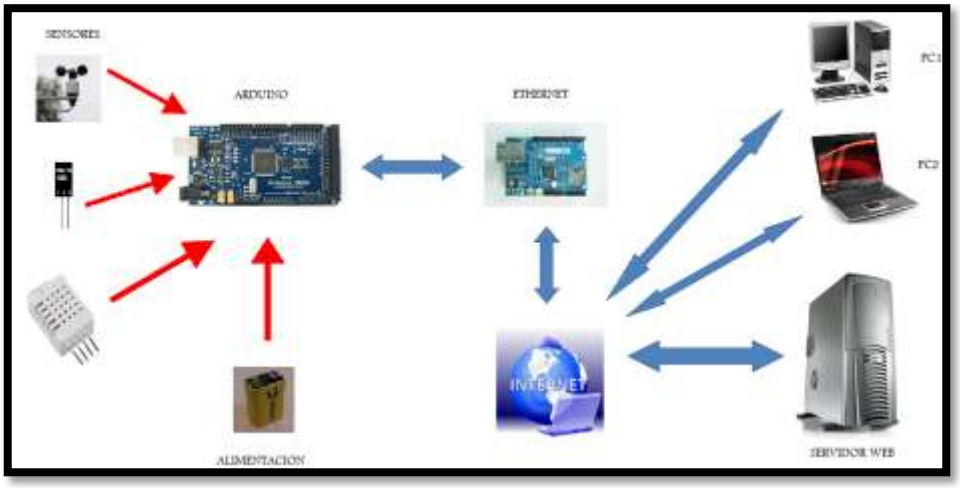

Figura 12. Esquema básico del Data Logger.

Como puede verse en la figura 3.2, la placa Arduino mega se encarga de comunicarse con todos los periféricos que tiene conectados, el funcionamiento de cada parte está controlado por la placa. El Arduino solicita la lectura a los sensores conectados al sistema cada cierto tiempo configurado por el usuario, le coloca la fecha y hora de adquisición gracias al RCT conectado al sistema y transmite los datos al modulo Ethernet para que éste, por medio de una conexión física de un cable de red UTP con conector RJ45 los envié al servidor web externo, este ultimo presenta los datos al usuario mediante herramientas o aplicaciones web.

Si el servidor esta fuera del alcance, es decir, no hay conexión física con el servidor externo, el Data Logger Arduino mediante el modulo Ethernet levanta un servidor web básico, de tal manera que el usuario pueda ver los datos que se están cenando mientras el sistema este fallado, además los datos censados son guardados en una memoria MSD con capacidad de 512 MB.

Al momento de restablecer la comunicación con el servidor web externo, el Data Logger se encarga de enviar los datos almacenados en la memoria que se respaldaron durante el tiempo que duro la falla, mientras se están enviando los datos al servidor externo el sistema no puede seguir monitoreando las variables físicas del ambiente; una vez terminada la transferencia de datos hacia el servidor externo, el sistema sigue censando las variables físicas y enviándolas al servidor externo.

2.3. Data Logger producto terminado y ultimas especificaciones

En la siguiente imagen se muestra una fotografía del producto terminado en donde se pueden notar los componentes de este, disponibles al usuario.

Figura 13. Fotografía de producto terminado

Los componentes son:

- Botón para luz de pantalla
- Puerto USB para programación
- Alimentación de voltaje
- Entradas analógicas
- Entradas digitales
- Teclado
- Pantalla LCD

2.4. Configuración y operación del data Logger

2.4.1. Menú de configuración del data Logger

Figura 14. Logo Minerva UES, en la LCD

Todo dispositivo similar al implementado en este proyecto debe de tener ciertos parámetros que deberían ser configurables, intervalo de muestreo, hora de inicio, entre otras. La configuración del Data Logger que se ha implementado en el presente trabajo es básica e incorpora la configuración de hora de inicio, hora de fin, tipo de sensores conectados a cada pin de E/S e intervalo de tiempo de envió de datos. Ya que este dispositivo no solo se encarga de censar y almacenar estos datos en una memoria como lo hacen otros, sino que también, envía los datos capturados hacia el servidor externo, se llegaría a pensar que el dispositivo necesita configuración de red, sin embargo, el sistema incorpora un servidor dhcp que obtiene la dirección IP automáticamente del modem o router al que esté conectado.

Figura 15. Pantalla de Configuración del Data Logger.

En la figura 4.2 observamos el mensaje que pregunta al usuario si desea realizar el procedimiento completo de configuración. Si se desea configurar los parámetros el usuario debe presionar "*" y si no "#" para dejar que el Data Logger trabaje con los parámetros por defecto. Los parámetros por defecto, están diseñados para que el arduino trabaje correctamente con los sensores que hemos mencionado anteriormente, conectados en pines específicos, si desea cambiar deposición algún

sensor habrá que configurarlo, en caso contrario se puede usar el arduino sin necesidad de configurar absolutamente nada.
Por tal motivo es de vital importancia dar a conocer los parámetros que se deben configurar y de qué forma se deben configurar, para ello el Data Logger se diseño con una pantalla LCD para mostrar mensajes al usuario, y un teclado de 4x3 para la introducción de datos del usuario hacia el sistema, para validar una opción el usuario debe presionar "*" o "#". El menú principal que presenta el Data Logger es el siguiente.

2.4.1.1. Configuración de pines

Figura 16. Configuración de pines.

El Data Logger necesita saber el tipo de sensor y en qué pin será conectado dicho sensor, es decir, el Data Logger como se ha mencionado anteriormente, tiene seis de sus E/S analógicas y otras 6 digitales, por tanto, es necesario especificar cual sensor se está conectando al sistema ya que el Data Logger fue diseñado de tal manera de que algunos de sus pines son fijos para algún tipo de sensor, esto es por su conexión física dentro del sistema. Esto es así solo con los sensores del tipo digital, en el caso de los sensores analógicos, estos pueden ser conectados en cualquier E/S analógica del Data Logger. A continuación se muestran los mensajes que solicitan la configuración.

a) En la pantalla el aparecerá el mensaje: ***"Configuración de pines"***
   El usuario debe analizar los sensores que va a conectar al sistema y en que pines los va a conectar, por ejemplo, si va a conectar un sensor del tipo DS18B20 este debe ser conectado en el pin 11 de Data Logger, un sensor del tipo DHT22 debe ser conectado en el pin 12. Los sensores que conforman la estación meteorológica también tienen los pines a los que van conectados y puede verse todos estos detalles en el manual de usuario.

b) Luego aparecerá el mensaje: ***"Digite el pin"***

Figura 17. Solicitando digitar el pin de conexión.

En este punto el usuario debe introducir mediante el teclado numérico, el número de pin al que desea conectar el sensor tomando en cuenta las características del mismo y el tipo de sensor que es conectado en cada pin.

Figura 18. Pin de conexión digitado por el usuario

c) A continuación aparecerá el mensaje en pantalla: ***"Digite el tipo"***

Figura 19. Digitando el tipo de sensor a conectar

Cada vez que se introduce el pin y tipo de sensor que se conectara a dicho pin, el Data Logger preguntara si desea ingresar otro pin y tipo o desea finalizar ya la configuración de pines, para ello se mostrara en pantalla el mensaje que se muestra en la siguiente imagen.

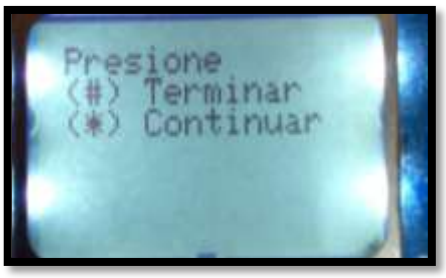

Figura 20. Pregunta si desea terminar o seguir configurando.

Para evitar trabajar con letras para describir el sensor que se va conectar al sistema, se dio la idea de trabajar con un numero que represente al tipo de sensor que se está utilizando, así pues el sistema es capaz de entender fácilmente el tipo de sensor y por tanto la ecuación que debe aplicarse para transformar un valor eléctrico o digital, en un valor razonable para el ser humano. A continuación se presenta la tabla con los tipos de sensores y su equivalente en número:

Tabla 2. Tipo de sensores usados y su equivalente numérico

Numero	**Descripción**
1	Temperatura lm35
2	Otro (analógico)
3	Temperatura ds18b20
4	Temp. y humedad DHT22
5	Dirección del viento
6	Anemómetro
7	Pluviómetro

Después de introducir cada pin y tipo de sensor conectado, el sistema le pedirá validar la información presionando la tecla "*" o "#" para anular la introducción anterior. Pero todos estos detalles los puede consultar en el manual de usuario.

2.4.1.2. Hora de inicio

Figura 21. Configurando hora de inicio.

Con la configuración de **"HORA DE INICIO"**, podemos programar al Data Logger para que este comience la adquisición de datos exactamente el día y la hora que se necesite censar.
Es tan fácil como introducir AÑO/MES/DIA y HORA/MINUTOS/SEGUNDOS, a medida que el dispositivo lo solicite.

Figura 22. Introduciendo hora de inicio del Data Logger.

Cabe mencionar que no se puede colocar un día inválido por ejemplo un día 32, un mes 14, un año 2000, aparecerá un mensaje en pantalla que me indica que el formato introducido es inválido o simplemente no es aceptado el valor.
Así como tampoco la hora con formato inválido, es decir, sobrepasar de 23:59:59,
Si no se configura esta hora de inicio el Data Logger comienza inmediatamente después que se termina de configurar todos los parámetros, sin hacer tiempo de espera para iniciar y finalizara hasta que el usuario desconecte la alimentación del sistema o hasta que la batería se agote en caso de esta conectado con batería.

2.4.1.3. Hora de finalización

Figura 23. Configurando hora de finalización.

El proceso de configuración es el mismo que el de hora de inicio, con el la diferencia que esta configuración es para indicarle al sistema en qué fecha y hora especifica se apagara el sistema, o dejara de censar datos.

2.4.1.4. Tiempo de muestreo

Con esta configuración se logra indicar al sistema cada cuanto tiempo necesita que se envíen los datos al servidor externo y el mismo tiempo de almacenamiento en la SD. Cabe mencionar que este tiempo es en segundos y es el intervalo de tiempo que el Data Logger hará peticiones al servidor.

Figura 24. Configuración de tiempo de muestreo.

2.5. Especificaciones de entradas y salidas del data Logger

Como se ha mencionado anteriormente el Data Logger ah sido diseñado para soportar algunos de los sensores disponibles comercialmente, específicamente son, el sensor de temperatura LM35 de tipo analógico, el sensor de temperatura DS18B20 de tipo digital, el sensor de humedad y temperatura DHT22 de tipo digital, y la estación meteorológica distribuida por la empresa Sparkfun la cual

consta de un sensor de que proporciona la dirección de viento de tipo analógico, un sensor que proporciona la velocidad del viento de tipo digital y un pluviómetro que es de tipo digital.

Sin embargo el sistema ha sido diseñado de tal manera que cualquier persona que conozca poco o mucho de electrónica sea capaz de diseñar o modificar el diseño realizado. Por tanto fácilmente podría agregarse mas sensores al sistema simplemente agregando más entradas y salidas al sistema ya que consta con 16 entradas analógicas y alrededor de 45 entradas digitales disponibles, de las cuales se ha tomado a bien utilizar solo 6 analógicas y 6 digitales ya que son suficientes para realizar una buena aplicación para el sistema diseñado.

A continuación en la siguiente tabla se explica en detalle todas las entradas y salidas de comunicación que posee el sistema Data Logger y a que pines deben conectarse los sensores.

Tabla 3. Conexión de sensores en cada pin del Data Logger.

PIN	**TIPO**	**CONEXIÓN DE SENSOR**
1	Analógico	Cualquier sensor analógico
2	Analógico	Cualquier sensor analógico
3	Analógico	Cualquier sensor analógico
4	Analógico	Cualquier sensor analógico
5	Analógico	Cualquier sensor analógico
6	Analógico	Cualquier sensor analógico
7	Analógico	Cualquier sensor analógico
8	Analógico	Cualquier sensor analógico
9	Digital	Sensor DS18B20
10	Digital	Sensor DHT22
11	Digital	Anemómetro
12	Digital	Pluviómetro

Como puede observarse en la tabla 2, los pines de 1 al 8 son del tipo analógico y pueden ser usados para conectar cualquier sensor de tipo analógico, puede conectar por ejemplo 5 sensores LM35 y 3 potenciómetros para cubrir los 8 pines. Los pines digitales que están enumerados del 7 al 12, en este caso si hay algunos

pines que están fijos para ciertos sensores, como puede verse en la tabla, el pin 9 está reservado para el anemómetro, el pin 10 está reservado para el pluviómetro, el 11 para el DHT22 y el 12 para el DS18B20. Por ello estos pines no pueden ser utilizados para conectar otro tipo de sensor, sin embargo, hay dos pines digitales disponibles para su conexión, el pin 7 y 8.

El código con el que se ha programado el Data Logger puede modificarse para agregar más pines digitales y/o analógicos, solo hay que tomar en cuenta en colocar la ecuación correcta para la conversión de los datos eléctrico o digitales del sensor para una excelente comprensión humana.

2.6. Programación del Data Logger

2.6.1. Software arduino

El entorno de código abierto Arduino hace fácil escribir código y cargarlo a la placa que se esté utilizando. Funciona en Windows, Mac OS X y Linux. El entorno está escrito en Java y basado en Processing, avr-gcc y otros programas también de código abierto. El entorno es muy amigable ya que posee un editor de texto que resalta las instrucciones propias del lenguaje para mejor comprensión, además no necesita otro programa para compilar y cargar el software a la placa, basta con hacer clic en el icono compilar o verificar y otro en el botón cargar, así de sencillo.

Por su gran parecido al lenguaje de programación en C, C++, es muy intuitivo de usar ya que las instrucciones son similares, aunque no se necesita sintaxis tan complicada como c lo cual es una ventaja. Pueden crearse muchas aplicaciones ya que en la web se encuentran una gran cantidad de librerías para controlar diversos dispositivos que se encuentran en el mercado, como sensores, RTC, pantallas LCD, teclados, etc.

2.6.2. Descarga del programa

El software está disponible en la web en el sitio oficial de Arduino.cc, sin embargo, en el contenido del CD anexo puede encontrar la versión más reciente hasta la fecha (noviembre/2012), en la cual ha sido desarrollado este proyecto.
Cabe mencionar que tanto hardware como software Arduino no tiene ningún costo, ya que están basados y desarrollas con licencia libre, para que cualquier principiante o profesional pueda realizar sus aplicaciones sin pagar ni un centavo por derechos, o cualquier otro tipo de cargo.

2.6.3. Código de programación

El código de programación es bastante intuitivo, si se tiene un poco de conocimiento del lenguaje de programación en C, la programación en esta plataforma Arduino le será bastante fácil.
Los programas hechos con Arduino se dividen en tres partes principales: estructura, valores (variables y constantes), y funciones. El Lenguaje de programación Arduino se basa en C/C++.

2.6.3.1. Estructura

La estructura de programación es bastante simple, todo código realizado en Arduino debe llevar dos campos obligatorios y son:

- Setup()
- Loop()

El setup() es una función de inicialización de todas las variables y librerías que se serán usadas dentro y fuera de este función, esta función solo se ejecuta una vez.
El loop(), es una función repetitiva que se encarga propiamente de repetir los ciclos completos de todo lo que se encuentra dentro de este campo, es decir esta función es un bucle infinito.
La estructura completa de estas dos funciones es la siguiente.

Código 1. Método setup llamado al iniciar el arduino

Setup

```
void setup() {
  //Instrucciones
}
```

Código 2. Método loop método llamado repetitivamente mientras se ejecuta el programa

Loop

```
void loop () {
  //Instrucciones
}
```

2.6.3.2. Variables y constantes

Las variables y constantes como en todo lenguaje de programación son utilizadas frecuentemente para almacenar variables de todo tipo, numéricos, alfabéticos, ASCII, etc. Este software no es la excepción, en el condigo desarrollado se han utilizado muchas variables para manejar los datos y hacer referencia a un tipo de dato especifico. Las variables pueden ser de diversos tipos, boolean, char, byte, int, float, double, string, array, void, etc. Se utilizan como convenga en la estructura del programa, la constantes sin embargo, son de tipo fijo, HIHG, LOW, INPUT, OUTPUT, true, false, etc. Las constantes y variables que vienen predefinidas en el lenguaje de Arduino. Se usan para facilitar la lectura de los programas. Las constantes se clasifican en grupos.

a) **Las que definen niveles lógicos, verdadero (true) y falso (false) (Constantes Booleanas)**

   Existen dos constantes para representar si algo es cierto o falso en Arduino: true, y false.

   False: false es el más sencillo de definir. false se define como 0 (cero).

   True: true se define la mayoría de las veces como 1, lo cual es cierto, pero tiene una definición más amplia. Cualquier entero que no es "0" es TRUE, en un sentido Booleano. Así, en un sentido Booleano, -1, 2 y -200 son todos true.

   Hay que tener en cuenta que las constantes *true* y *false* se escriben en minúsculas, al contrario que HIGH, LOW, INPUT, y OUTPUT.

b) **Las que definen el nivel de los pines, nivel alto (HIGH) y nivel bajo (LOW)**

   Cuando se lee o escribe en un pin digital, existen sólo dos valores que podemos obtener o asignar: **HIGH** y **LOW**.

   HIGH: El significado de HIGH (en referencia a un pin) depende de si el pin está configurado como entrada (INPUT) o como salida (OUTPUT). Cuando un pin se configura como entrada (INPUT) usando pinMode, y se lee con digitalRead, el microcontrolador nos retornará HIGH si en el pin hay 3 voltios o más. Un pin puede ser configurado como entrada (INPUT) usando pinMode, y después establecerlo a HIGH con digitalWrite, esto conectará el pin a 5 Voltios a través de una resistencia interna de 20K, resistencia pull-up , la cual establecerá el pin al estado de lectura HIGH a menos que la conectemos a una señal LOW a través de un circuito externo.

Cuando un pin se configura como salida (OUTPUT) con pinMode, y se establece a HIGH con digitalWrite, el pin tiene 5V. En este estado puede usarse como fuente de corriente, ya sea un LED que se conecte a través de resistencias en serie a masa (tierra), o a otro pin configurado como salida y establecido a LOW.

LOW: El significado de LOW difiere también según esté configurado como entrada (INPUT) o como salida (OUTPUT). Cuando un pin está configurado como entrada (INPUT) con pinMode, y se lee con digitalRead, el microcontrolador retornará LOW si el voltaje presente en el pin es de 2V o menor.

Cuando un pin es configurado como salida (OUTPUT) con pinMode, y establecido LOW con digitalWrite, el pin tiene 0 voltios. En este estado puede absorber corriente, como un LED que se conecte a través de resistencias en serie a +5 voltios, o a otro pin configurado como salida, y establecido a HIGH.

c) **Las que definen los pines digitales, INPUT y OUTPUT**

Los pines digitales pueden ser usados como **entrada (INPUT)** o como **salida (OUTPUT)**. Cambiando un pin de INPUT a OUTPUT con pinMode() el comportamiento eléctrico del pin cambia drásticamente.

Pines configurados como entradas

Los pines de Arduino (Atmega) configurados como **INPUT** con pinMode() se dice que se encuentran en un estado de alta impedancia. Una forma de explicar esto es que un pin configurado como entrada se le aplica una muy baja demanda, es decir una resistencia en serie de 100 Megaohms. Esto lo hace muy útil para leer un sensor, pero no para alimentar un LED.

Pins configurados como salidas

Los pins configurados como **salida (OUTPUT)** con pinMode() se dice que están en estado de baja impedancia. Esto implica que pueden proporcionar una sustancial cantidad de corriente a otros circuitos. Los pins de Atmega pueden alimentar (proveer de corriente positiva) o absorber (proveer de masa) hasta 40 mA (miliamperios) de corriente a otros dispositivos/circuitos. Esto los hace muy útil para alimentar LED's pero inservible para leer sensores. Los pines configurados como salida pueden deteriorarse o romperse si ocurre un cortocircuito hacia los 5V o 0V. La cantidad de corriente que puede proveer un pin del Atmega no es suficiente para la mayoría de los relés o motores, y es necesario añadir circuitería extra.

2.6.3.3. Librerías utilizadas

Las librerías proporcionan funcionalidad extra para la utilización en "sketches", por ejemplo para trabajar con hardware o manipular datos. Para utilizar una librería en un "sketch", debe seleccionar el menú Sketch > Import Library. Esto insertará una o más sentencias #include al principio del "sketch" y compilará la librería con su "sketch". Debido a que las librerías se vuelcan a la placa junto con su "sketch", incrementan la ocupación del espacio disponible. Si un "sketch" no precisa de una librería, simplemente se borra su sentencia #include en la parte inicial de su código.
Existe una Lista de librerías en las referencias. Algunas librerías están incluidas en el software Arduino, otras pueden ser descargadas desde una gran variedad de fuentes. Para instalar estas librerías de terceros, se debe crear un directorio denominado libraries en su Directorio sketchbook. Después descomprimir la librería allí. Por ejemplo, para instalar la librería Data Time, sus ficheros deberían estar en una subcarpeta /libraries/DateTime en su carpeta de sketchbook.

Algunas de las librerías más importantes usadas para implementar este sistema son:

#include <SPI.h>

Utilidad: Esta biblioteca le permite al Arduino comunicarse con los dispositivos SPI, con Arduino como dispositivo maestro.

#include <Ethernet.h>

Utilidad: Para conectar a internet usando el Ethernet Shield.

#include "DHT.h"

Utilidad: Permite al Arduino comunicarse con el sensor de humedad y temperatura DHT21 y DHT22.

#include <OneWire.h>

Utilidad: Controla dispositivos (de Dallas Semiconductor) que usan el protocolo One Wire.

#include <Wire.h>

Utilidad a través de una red de dispositivos y sensores.

#include <DallasTemperature.h>

Utilidad: Esta librería permite que la placa Arduino se comunique con la mayoría de los sensores provistos por Dallas Texas, en este caso el DS18B20.

#include "RTClib.h"

Utilidad: Es la librería que controla al reloj usado en este proyecto.

#include "PCD8544.h"

Utilidad: Arregla bugs de inicialización de LCD de la librería LCD oficial de adafruit.com.

#include "Graphic.h"

Utilidad: Es un archivo que contiene el mapa de bits de las imágenes que se presentan en la pantalla LCD, en este caso se usando la imagen de la diosa minerva.

#include <SD.h>

Utilidad: Permite la comunicación entre la placa Arduino y tarjeta micro SD

#include <Keypad.h>

Utilidad: Permite la comunicación con el teclado 4X3 utilizado en este proyecto.

Puede encontrar información más amplia acerca de estas librerías accediendo a los sitios web del proveedor de cada sensor que usted encontrara en la bibliografía de este documento.

2.6.3.4. Configuración de los sensores

Cada sensor utilizado en este proyecto necesita ser configurado para que la placa Arduino pueda comunicarse con este sensor mediante su librería, para entender un poco de esta configuración vamos a explicar un poco del código Arduino que permite esta configuración.

Como se ha mencionado anteriormente el Data Logger está diseñado para tomar lecturas de 12 pines E/S, de los cuales seis pueden ser entradas analógicas y otros seis E/S digitales. Por tanto en el código Arduino se declara un arreglo de enteros de longitud 12 el cual almacena el tipo de sensor que se va a conectar a cada pin, así:

```
int typeSensor[]   = { 2,  5,  1,  2,  0,  0,  0,  0,  0,  3,   4,   6,   7,   8 };
```

En el orden: A0, A1, A2, A3, A4, A5, A6, D7, D8, D9, D10, D11, D12

Con el significado siguiente:

Tipo 1 = lm35

Tipo 2 = otro (potenciómetro)

Tipo 3 = ds18b20

Tipo 4 = dht

Tipo 5 = vane

Tipo 6 = anemómetro

Tipo 7 = rain gauge

Por lo que podemos deducir que en los doce pines del Data Logger tenemos unos parámetros por defecto, es decir, en el pin 1 se puede conectar un potenciómetro, en el 2 un sensor LM35, los pines del 3 - 7 están deshabilitados no tienen configurados ningún tipo de sensor, el pin 8 tiene un sensor de de la estación meteorológica el cuál corresponde a la veleta, el pin 9 está configurado para el anemómetro, el pin 10 corresponde al pluviómetro, el pin 11 pre-configurado para el sensor de humedad y temperatura DHT22 y el pin 12 para el sensor de temperatura DS18B20.

Inicialización de cada sensor:

- POTENCIOMETRO: El potenciómetro no necesita configuración ya que proporciona un dato puramente analógico, por lo tanto solo se utilizan la instrucción para leer el pin al que está conectado así:
  analogRead(pin);
  Instrucción con la cual se está leyendo el dato proporcionado por la caída de tensión que hay en el potenciómetro, puede servir para controlar niveles de un sistema de interés, por ejemplo, nivel de luz, velocidad, tiempo, etc.

- LM35: Al igual que el potenciómetro este no necesita configuración para el funcionamiento, simplemente obtener la lectura con el mismo comando y aplicar la formula correspondiente para la interpretación correcta de temperatura,
  analogRead*(pin * 5.0 * 100.0 / 1023.0)*
  Con esta ecuación se obtiene la temperatura correspondiente a cada lectura, ya que el sensor proporciona 0.1 mV/°C, y las entradas analógicas del Arduino proporcionan una resolución de 10 bits (1024).

- DHT22: La manipulación de este sensor de humedad y temperatura se hace bastante fácil gracias a la librería que permite la comunicación con el Arduino, por supuesto que se debe incluir la librería DHT.h mencionada anteriormente y proporcionar la configuración adecuada para su optimo funcionamiento, podemos ver a continuación lo fácil que resulta obtener una lectura de este sensor.

Código 3. Configuración del sensor DHT

Código datalogger

```
#define DHTPIN 46
#define DHTTYPE DHT22
#define ONE_WIRE_BUS 47
DHT dht(DHTPIN, DHTTYPE);
dht.begin(9600);
```

Primero se define el pin digital de Arduino al que se va a conectar físicamente el sensor, en este caso, en el pin 46 de Arduino mega, luego se define el tipo de sensor ya que la misma librería puede controlar los tipos DHT21 y DHT22, con estas dos macros define el sensor llamado en este caso "dth", por supuesto puede ser cualquier otro nombre que se crea conveniente.
Con la instrucción anterior se inicializa la librería que permite al Arduino comunicarse con el sensor de humedad y a partir de este punto ya se puede pedir la lectura del sensor así:

float h = dht.readHumidity();
float t = dht.readTemperature();

La variable "h" contiene el valor de la humedad relativa y la variable "t" el dato de temperatura. Y de esta manera los datos están listos para ser mostrados en pantalla, monitor serial del software Arduino, almacenamiento en la MicroSD o para ser enviados al servidor externo.

- DS18B20: El procedimiento es similar al sensor DHT22, con algunas diferencias en las instrucciones, se deben incluir en el sketch las librerías OneWire.h, DallasTemperature.h.
  Se define primeramente el pin digital de Arduino al cual será conectado, en este caso al pin 48 de Arduino mega.
  #define ONE_WIRE_BUS 48

  Se configura el DS18B20 con el nombre deseado y el pin definido anteriormente.
  OneWire oneWire(ONE_WIRE_BUS);
  Y con esto ya está listo para extraerle lecturas cuando sea necesario, con la siguiente instrucción.

El index "0" se utiliza porque se está usando la librería que permite leer varios dispositivos en un solo cables, por tanto, pueden conectarse muchos sensores del mismo tipo en el mismo nodo, simplemente cambiando el índice, por el correspondiente configurado en el dispositivo.

- ANEMÓMETRO: El anemómetro es el sensor que permite medir la velocidad del viento y la configuración de este es un poco más difícil que los anteriores, sin embargo no representa imposibilidad alguna. La medición de este sensor es bastante básica ya que prácticamente solo es un switch que indica que la veleta ha realizado una vuelta completa, esta vuelta es capturada mediantes interrupciones al Arduino.
  Se declara el pin al que está conectado el anemómetro, para este proyecto hemos usado el pin digital 2, por tanto la variable anem_pin corresponde al pin de conexión del sensor,
  int anem_pin = 2;
  La siguiente variable es usada para proporcionar el periodo de tiempo con el que se hará el cálculo, es decir, cada vuelta que da la veleta se cuenta en un intervalo de tiempo correspondiente a esta variable.
  int time_for_anem = 5;
  Se necesita saber cada cuanto tiempo se proporcionara los datos del sensor de velocidad del viento, para ello se creó una variable de tipo entero, la cual configura cada cuanto tiempo se procesaran los conteos equivalente a cada vuelta del sensor.
  unsigned long next_calc_anem = 0;
  Se da un tiempo al Arduino para que pueda procesar los conteos, para ello se usa la siguiente variable, para estar seguros que no se hará una medición mientras no haya transcurrido el tiempo de descanso.
  unsigned long last_anem_count = 0;
  La variable count_anem almacena los conteos equivalentes a cada vuelta de la veleta y que se obtienen mediante interrupciones que se describen más adelante.
  int count_anem = 0;
  La última variable corresponde a la velocidad de viento, que hace tomando las referencias que proporciona el diseñador del anemómetro que es aproximadamente de 2.4 km/s por vuelta.
  float anem_value = 0;
  A continuación se muestra la función para el procesamiento de los conteos que se toman del anemómetro.

Código 4. Función para calcular el valor de la velocidad del viento con un anemómetro

Código datalogger

```
void calc_anem(){
  if (millis() >= next_calc_anem*1000){
    next_calc_anem = (millis()+time_for_anem*1000)/1000;
    anem_value = 2.4*count_anem/time_for_anem;
    count_anem = 0;
  }
}
```

Puede observar que solo es una multiplicación que se hace, tomando en cuenta el conteo de las interrupciones (count_anem) por 2.4 km/s que equivalen a cada vuelta esto entre el intervalo de tiempo de muestreo.
Se muestra a continuación el código bastante simple, utilizado para contar las interrupciones.

Código 5. Función que atiende la interrupción generada por el anemómetro

Código datalogger

```
void countAnem(){
  if((micros() - last_anem_count) > 500){
    count_anem++;
    last_anem_count = micros();
  }
}
```

Solo debe cumplirse que el tiempo transcurrido sea mayor a 500 microsegundos tomando en cuenta una precisión considerable porque es improbable que el anemómetro gire a más de 2000 revoluciones por segundo.

- PLUVIOMETRO: El procesamiento de datos que provienen de este sensor es prácticamente el mismo que el del anemómetro, ya que se trata de las mismas interrupciones.

Se mostrara solamente las funciones que realizan el proceso de muestreo para que pueda comprobarse la similitud con el proceso anterior.

Código 6. Código para cálculo y atención de interrupciones del pluviómetro

Código datalogger

```
void calc_rain(){
  if (millis() >= next_calc_rain*1000){
    next_calc_rain = (millis()+time_for_rain*1000)/1000;
    rain_value = 0.2794*count_rain;
  }
}
void countRain(){
  if((micros() - last_rain_count) > 500){
    count_rain++;
    last_rain_count = micros();
  }
}
```

- **Dirección del viento (vane o cazoletas):** La manipulación de este sensor es bastante simple, ya que este proporciona un voltaje según corresponda con la dirección a la que apunta la flecha obligada por el viento. Entonces es bien práctico, la dirección del viento se calcula comparando en una tabla el voltaje del sensor con los grados y dirección proporcionados en los datos del diseñador.

Tabla 4. Dirección del viento según el voltaje proporcionado

Dirección (Grados)	**Resistencia (Ω)**	**Voltaje (V)**
0	33k	3.84v
22.5	6.57k	1.98v
45	8.2k	2.25v
67.5	891	0.41v
90	1k	0.45v
112.5	688	0.32v
135	2.2k	0.90v
157.5	1.41k	0.62v

180	3.9k	1.40v
202.5	3.14k	1.19v
225	16k	3.08v
247.5	14.12k	2.93v
270	120k	4.62v
292.5	42.12k	4.04v
315	64.9k	4.78v
337.5	21.88k	3.43v

Esta tabla se describe en código Arduino de la siguiente manera

Código 7. Variables utilizadas para la adecuada conversión en la veleta de dirección de viento

Código datalogger

```
int num_values_vane = 16;
int analog_vane_read[] =      {    65,    83,  92,    126,  184,    243,  286,   405,  460,
600,  630,    702,  786,    827,  946,  978 };
float degree_vane_convert[] = { 112.5,  67.5,  90,  157.5,  135,  202.5,  180,  22.5,   45,
247.5,  225,  337.5,    0,  292.5,  270,  315 };
```

La primera variable indica al Arduino el numero de posiciones que pueden interpretarse a cerca de la dirección de viento, es decir, si son 16 posiciones, el sensor tiene una precisión de 22,5 °.

El arreglo analog_vane_read corresponde al número digital que el Arduino convierte a la entrada del pin analógico al cual está conectado el sensor y que equivale al voltaje que proporciona el sensor. El siguiente arreglo que observa solo se utiliza para convertir el voltaje a grados direccionales para comprensión humana.

Entonces, el código Arduino para la lectura y procesamiento de los datos de este sensor se vuelve muy simple y se reduce a una simple comparación de datos.

Código 8. Código para calcular la dirección del viento

Código datalogger

```
float calc_vane(int pin){
  int val = analogRead(pin);
  int x = 0;
  for(x=0;x<num_values_vane;x++){
    if(analog_vane_read[x] >= val){break;}
  }
  return degree_vane_convert[x];
}
```

Bastante simple, se hace un lazo "for" para comparar el voltaje (numero), con el dato extraído del sensor, y luego se retorna un valor del arreglo en grados, listo para ser mostrado al usuario.

2.6.3.5. Configuración de pantalla LCD

La configuración es simple gracias a la librería que permite la comunicación con el Arduino, solo necesita se inicializada y luego utilizar las instrucciones correspondientes para escribir, mostrar y limpiar pantalla.

Primero deben definirse el nombre y a que pines está conectada la pantalla LCD, es una definición global.

PCD8544 lcd = PCD8544(8, 7, 6, 5, 4);

A continuación se inicializa la pantalla dentro de la función setup(), y se configura el contraste necesario, en este caso el 60%.

lcd.init();
lcd.setContrast(60);

Esta función imprime una imagen de mapa de bits almacenado en la variable el archivo "Graphics.h".

drawMinerva();

Se necesita indicar en posición de pantalla se va a imprimir, cabe mencionar que la pantalla soporta 14 caracteres en las columnas y 6 en las filas, para hacer esto la librería implementa una función, la cual se usa de la siguiente manera.

lcd.setCursor(0, 0);

Los parámetros corresponden a las filas y columnas.
Imprimimos un mensaje en pantalla de la siguiente manera,

lcd.print("Este es un mensaje mostrado en la pantalla LCD");
lcd.display();

Para borrar texto de la pantalla se usa la siguiente función,

lcd.clear();

2.6.3.6. Configuración de teclado

El teclado no necesita una configuración de inicialización, la librería <Keypad.h> solo necesita saber en qué pines del Arduino está conectado el teclado, y un lector de eventos para capturar una tecla del mismo.

Con este segmento de código se indican los pines a los que están conectadas las filas y columnas del teclado, y se crea el teclado por medio de funciones que provee la librería para mapear el teclado,

Código 9. Configuración del teclado

Código datalogger

```
const byte ROWS = 4; // Cuatro filas
const byte COLS = 3; // Tres columnas
char numberKeys[ROWS][COLS] = {
    { '1','2','3' },
    { '4','5','6' },
    { '7','8','9' },
    { '*','0','#' }
};
char* keypadMap = makeKeymap(numberKeys);
byte rowPins[ROWS] = {32, 34, 36, 38};
byte colPins[COLS] = {40, 42, 44};
Keypad keypad = Keypad(keypadMap, rowPins, colPins, sizeof(rowPins),
sizeof(colPins));
```

Las filas están conectadas a los pines digitales 32, 34, 36 y 38 de Arduino mega respectivamente.

2.6.3.7. Configuración de RTC

La librería "RTClib.h" proporciona las funciones útiles para la lectura del reloj en tiempo real utilizado en este proyecto, algunas para funciones un poco complejas como proyectar o extrapolar el tiempo, pero en este caso el funcionamiento es simple.

Lo primero es definir un nombre para el RTC, esto se hace en el campo para macros para que la variable sea global,

RTC_DS1307 RTC;

En este caso el nombre utilizado es lógico RTC, aunque puede ser cualquier otro considerando un nombre corto y comprensible.
Luego en la función principal setup(), debe utilizarse algunas instrucciones para inicializar el reloj.

Código 10. Inicialización de la RTC

Código datalogger

```
Wire.begin();
RTC.begin();
if (! RTC.isrunning()) {
  RTC.adjust(DateTime(__DATE__, __TIME__));
}
```

Se inicializa la librería Wire, la cual es utilizada por este tipo de RTC para acoplarla con Arduino y la lectura en un solo cable, y luego se inicializa la RTC, luego, si la RTC no está en operación se configura la hora mediante la función RTC.adjust(). Después de esto esta lista la RTC para entregar la hora y fecha correspondiente.
Para extraer hora y fecha del RTC se usan los siguientes comandos.

Código 11. Obteniendo una string con la fecha y hora

Código datalogger

```
//Obtiene la fecha y hora actual
String getTime(){
  DateTime now = RTC.now();
  String time = String(now.year(), DEC);
  time += "/";
  time += String(now.month(), DEC);
  time += "/";
  time += String(now.day(), DEC);
  time += "%20";
  time += String(now.hour(), DEC);
  time += ":";
  time += String(now.minute(), DEC);
  time += ":";
  time += String(now.second(), DEC);
  return time;
}
```

Claramente puede observarse que la variable time está definida como una cadena de caracteres para almacenar la fecha y hora extraída del RTC con las respectivas instrucciones de año, mes, día, hora, minutos y segundos en formato decimal.

2.6.3.8. Puerto serial

El puerto serial es útil para mostrar mensajes en el monitor serial que provee el software Arduino, y solo necesita ser inicializado en la función setup(), con el comando siguiente,

Serial.begin(9600);

El argumento de la función serial.begin corresponde a los baudios con los que se desea comunicar con el puerto serial del ordenador, típicamente 9600, sin embargo puede comunicar también a 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, o 115200 siempre y cuando se configure esta velocidad en el puerto serial.

2.6.3.9. Loop del programa o cuerpo principal

Al usar funciones la programación en Arduino se vuelve más fácil y más comprensible, ya que una función puede realizar un proceso completo y se evita colocar repetidas instrucciones en un solo lugar. Esto puede verse en el loop del

código implementado en este proyecto, este loop es el proceso principal de un código en Arduino ya que es la ejecución permanente de las instrucciones de programación siguiendo una secuencia lógica según la aplicación para la cual ha sido creado.

Código 12. Código para el loop del programa

Código datalogger

```
void loop() {
  if (read_now()){
    if(is_conected){
      toServer();
    }else{
      toSd();
    }
    if (putAnswerServer){getAnswerServer();}
    calc_anem();
    calc_rain();
  }else{
    if ((millis() - last_msg_time) > show_ip_msg*1000){
      showTime();
      last_msg_time = millis();
    }else      if((millis()      -      last_msg_time)      >
show_ip_msg*1000*1.5){
      showIP();
      last_msg_time = millis();
    }
  }
  playServer();
}
```

Puede observarse lo simple que se vuelve el código cuando se usan funciones, es decir, conlleva el mismo trabajo de programación que cuando no se usan funciones pero el código se vuelve entendible y cualquier error se sabe donde corregirlo.

La lógica es simple, el Data Logger constantemente pregunta si hay un cliente conectado y de ser así, usa la función toServer(), para la captura de datos de los diferentes sensores y se envían al servidor externo. Al mismo tiempo el data Logger levanta un servidor local en el cual se pueden ver los datos que están

siendo leídos en tiempo real, colocando la dirección IP asignada al Data Logger en el explorador web de un equipo que se encuentre en la misma red local que el dispositivo. Luego si no existe un cliente, porque no hay red de internet, es decir, no se puede establecer con el servidor, el Data Logger almacena los datos en la tarjeta micro SD para su posterior procesamiento de datos.
Las lecturas de la estación meteorológica se hacen siempre ya que están gobernadas por interrupciones y siempre hay que estar atentos a capturarlas para mantener almacenado el valor y proporcionarlo cuando sea necesario.

Al revisar el código Arduino contenido en los anexos, usted comprenderá fácilmente el proceso de programación y cómo funciona cada una de las funciones. Consulte el manual de usuario para comprender como operar al Data Logger y comparar el proceso del código Arduino.

2.6.3.10. Funciones útiles

El código Arduino estructurado con funciones que realizan tareas especificas para que el lector pueda comprender mejor el código y para depuración de errores y desde luego poder añadir aplicaciones. Detallamos una función para explicar cómo es la definición de la función y su llamada.

La función getTime() que se muestra en el código 12, es la que obtiene la hora del RTC y tiene una operación bien simple.
La primera línea que muestra este código es la definición de la función abriendo corchetes para contener al cuerpo de la función y luego cerrando para indicar el fin de la función.

Como puede verse la función solo retorna una variable de tipo String que contiene la fecha y hora en el formato *AÑO/MES/DIA HORA/MINUTOS/SEGUNDOS*, con un espacio de por medio que equivale a un %20 para enviarse al servidor externo.
Para utilizar esta función en cualquier lugar del programa, basta con hacer el llamado de la función, por ejemplo:

String hora_fecha = getTime();

Con ello queda almacenada la hora y fecha que retorna la función desde la RTC en la cadena hora_fecha.

Algunas de las funciones creadas se presentan a continuación con una breve descripción a lado,

setUpRtc();	Configuración de RTC
setUpSd();	Configura Memoria MicroSD
generalConfig();	Presenta menú de configuración
configStation();	Configura la estación meteorológica
showIP();	muestra la IP asignada al Data Logger
calc_anem();	Calcula velocidad del viento
calc_rain();	Calcula cantidad de lluvia
playServer();	Levanta el servidor Arduino
getData()	Extrae los datos de los sensores conectados

2.6.3.11. Envió de datos al servidor externo

Para enviar los datos al servidor externo se creó la función toServer(), la cual se encarga de obtener los datos de los sensores conectados y enviarlos al servidor.

Código 13. Función que se encarga de obtener los datos y enviarlos al servidor auxiliándose de dos funciones más previamente creadas

Código datalogger

```
void toServer(){
  String url = getData();
  httpRequest(url);
}
```

Siempre y cuando se cumpla con el tiempo establecido para la conexión con el servidor, es decir, que haya transcurrido un tiempo mayor o igual que el tiempo de muestreo configurado por el usuario después de la ultima conexión con el servidor, se obtiene los datos de todos los sensores conectados al Data Logger utilizando la función getData(), y estos son enviados al servidor externo utilizando la función httpRequest(), que pasa como parámetro los datos extraídos de los sensores.

La lógica y funcionamiento de estas funciones puede comprenderse mejor revisando el código Arduino completo incluido en los anexos, ya que contiene comentarios básicos que explican la función o proceso de cada línea de código.

2.6.3.12. Servidor Arduino

Debe conocer un poco de programación web html para comprender lo que esta función hace, ya que se necesita el protocolo http para que el explorador web en un equipo de computo pueda interpretar correctamente los datos enviados.

Se usa la librería Ethernet.h para la comunicación entre Arduino y el modulo Ethernet Shield Arduino, esta librería posee funciones que permiten realizar este proceso muy fácilmente.

Código 14. Código que atiende las peticiones hechas desde un navegador a arduino directamente.

Código Datalogger

```
String playServer(){
  clientServer = server.available();
  String str = "";
  if (clientServer) {
    info("Nuevo cliente");
    boolean currentLineIsBlank = true;
    while (clientServer.connected()) {
      if (clientServer.available()) {
        char c = clientServer.read();
        str += c;
        if (c == '¥n') {currentLineIsBlank = true;}
        else if (c != '¥r') {currentLineIsBlank = false;}
        if (c == '¥n' && currentLineIsBlank){break;}
      }
    }
    info(str);
    char line[50];
    getFileName(str).toCharArray(line, 50);
    sendFile(line);
    delay(1);
    clientServer.stop();
    info("cliente desconectado");
  }
  return str;
}
```

Sin entrar en mayores detalles, a continuación se explica un poco sobre el proceso que realiza esta parte del código Arduino, correspondiente al servidor Arduino. El Data Logger pregunta si hay un cliente disponible, y mientras este cliente se encuentre conectado se leen las peticiones del cliente para poder servirle los archivos que se encuentran almacenados en la micro SD y que contienen el código html que será presentado en el explorador del ordenador del cliente, este proceso se repite cada vez que el cliente no envié una línea en blanco, lo cual significa que la petición a finalizado. Cabe mencionar que siempre abran peticiones al servidor Arduino ya que el código html contiene un refresh que actualiza cada cinco segundos los datos que son leídos de los sensores y presentados en una tabla en el explorador web.

2.6.3.13. Almacenamiento en la memoria MicroSD

Lo primero es consultar si ya es posible la conexión con el servidor, de ser así, se envían los datos capturados durante el tiempo de desconexión contenidos en la memoria micro SD. Si no es posible aun la comunicación con el servidor, se procede a guardar los datos de los sensores en la memoria micro SD, siempre con el mismo intervalo de tiempo de muestreo (read_now).

Código 15. Código que envía los datos a la SD

Código Datalogger

```
void toSd(){
  if (client.connect(serverName, 80)){
    info("¥n¥n¥n
==================¥n
         Conexion restaurada¥n
==================¥n¥n");
    is_conected = true;
    client.stop();
    sendSdToServer();
  }
  String url = getData();
  info("SD: " + url);
  printToSd(url);
  lastConnectionTime = millis();
}
```

III. Servidor web Ruby on Rails

3.1. Acerca de el servidor web RoR

Cuando se es un estudiante de Ingeniería Eléctrica, poco se sabe de las tecnologías actuales de programación en la web, básicamente un estudiante sabe que existe un lenguaje de hipertexto (HTML) que interpretan los navegadores, conoce un poco de la arquitectura cliente servidor y que la parte dinámica de un sitio es programada en algún lenguaje como, PERL, PHP, etc. Con estos lenguajes y otros se pueden escribir CGI's, si el estudiante ha puesto atención en clases tal vez se conozca un poco de JSP´s y algún framework como Hibernate aunque solo vagamente. Entonces a la hora de comenzar un proyecto ¿qué lenguaje elegir? y ¿Se usara algún framework? Estas son preguntas que deben hacerse.
Bueno puedo decir con mucha seguridad después de probar PERL, PHP y JAVA con sus JSP's, que la mejor elección para una aplicación web es RUBY ON RAILS.

3.2. Historia y antecedentes

3.2.1. Ruby (www.ruby-lang.org)

Figura 25. Logo de RUBY.

Ruby es un lenguaje de programación orientado a objetos, todo en ruby es un objeto, por tanto tienen métodos que nos permiten realizar tareas de una forma más fácil. Ruby es OpenSource, se puede usar, copiar y modificar su código fuente, Su creador ha mezclado partes de sus lenguajes favoritos (Perl, Smalltalk, Eiffel, Ada, y Lisp) para formar un nuevo lenguaje que incorporara tanto la programación funcional como la programación imperativa.
Los fanáticos de ruby expresan que ruby es un lenguaje hermoso y artístico.
Al leer un código en ruby es muy fácil entender su funcionalidad, ya que su creador Yukihiro "matz" Matsumoto ha tratado de hacer que ruby sea natural, que se asemeje a la vida real, no pretende que ruby sea simple.

En RUBY, todo es un objeto. Se le puede asignar propiedades y acciones a toda información y código. La programación orientada a objetos llama a las propiedades variables de instancia y las acciones son conocidas como métodos. La orientación a objetos pura de RUBY se suele demostrar con un simple código que aplica una acción a un número.

Código 16. En ruby todo es un objeto

```
irb>

5.times { print "Nos *encanta* Ruby -- ¡es fuera de serie!" }
```

3.2.2. Rails (http://rubyonrails.org)

Figura 26. Logo de RAILS.

Rails es un framework para crear aplicaciones web en Ruby, con Rails se pueden construir aplicaciones web que antes podrían haber tomado semanas o incluso meses en tan solo unos días.

Rails sigue al igual que los mejores framework la arquitectura modelo vista controlador o MVC con lo cual se obtiene como resultado un proyecto muy ordenado, fácil de extender y depurar.

¿Quienes ya están usando Rails?

Solo aquí en El Salvador en tan poco tiempo se han desarrollado muchas sitios, para el interior del país y para el exterior. Buscando encontré más de 75 sitios, entre los más populares están.

- www.integral.com.sv
- www.laescalon.org
- www.libroslaceiba.com
- www.salvasol.com
- www.melher.com
- www.cta.com.sv
- www.tiendascarrion.com

Este proyecto ha sido desarrollado en con la versión 3.2.8 de Rails, que es la ultima versión disponible en el momento del desarrollo del proyecto. La primera versión de Rails fue publicada el 13 de diciembre de 2005, Rails 3.2.0 fue publicada el 20 de enero de 2012, en el momento de terminar este trabajo ya esta disponible la versión 3.2.11, se recomienda actualizarse rápidamente a esta versión de Rails, ya que hace unos pocos días, (el 8 de enero/2012) fueron reportadas dos vulnerabilidades de seguridad de Rails un atacante puede lograr ejecutar código arbitrario en la aplicación, la versión 3.2.11 no tiene esa vulnerabilidad y este proyecto es compatible con dicha versión.

La filosofía de Rails incluye varios principios:

- DRY - "Dont Repeat Yourself" - sugiere que escribir el mismo código una y otra vez no es correcto.
- Convención sobre configuración - significa que Rails hace suposiciones acerca de lo que se quieren hacer y cómo lo vamos a hacer, en lugar de pedirle que especifique cada pequeña cosa a través de los interminables archivos de configuración.
- REST es el mejor patrón para aplicaciones web - Organiza su aplicación en torno a los estándares HTTP, es la manera más rápida de avanzar.

3.3. Instalando el ambiente de trabajo para RoR

Hay muchas formas de instalar RoR. Para Windows, básicamente se descarga el instalador y ya, pero tiene sus desventajas, cuando se quiere trabajar varios proyectos, con diferentes versiones de RUBY y diferentes versiones de RAILS. En Linux se puede instalar desde el gestor de paquetes (similar a Windows) con iguales desventajas.
Quizá la mejor opción es usar RVM (Ruby Versión Manager).

3.3.1. RVM (http://rvm.io/)

Con RVM se pueden tener instaladas varias versiones de RUBY al mismo tiempo, además permite trabajar con diferentes gemset (Conjuntos de gemas), de esta forma se puede especificar diferentes versiones de las gemas para los diferentes proyectos, RAILS es una gema, se puede estar usando RUBY 1.8.7 con Rails 2.3 y otro gemset para RUBY 1.9.2 con RAILS 3.0.7 o Rails 3.2.8, o cualquier combinación que queramos.

Para instalar el RVM también se puede hacer con el gestor de paquetes pero de igual forma muy probablemente no estará disponible la última versión estable, por tanto lo instalaremos desde la página oficial.

Código 17. Instalando el ambiente de trabajo

```
terminal $

# Instalando CURL que servirá para instalar el RVM.
sudo apt-get install curl
# Luego se instala el RVM con curl.
curl https://raw.github.com/wayneeseguin/rvm/master/binscripts/rvm-installer
| bash -s stable
# Se carga el RVM
source ~/.rvm/scripts/rvm
# El RVM devuelve con el siguiente comando las dependencias
rvm requirements
# Se procede ha instalar lo que aparece en pantalla
sudo apt-get install build-essential openssl libreadline6 libreadline6-dev
curl git-core zlib1g zlib1g-dev libssl-dev libyaml-dev libsqlite3-dev
sqlite3 libxml2-dev libxslt-dev autoconf libc6-dev ncurses-dev automake
libtool bison subversion
# Se instala ruby, en este caso se esta usando la versión 1.9.2, pero ya
está disponible la versión 1.9.3 como se menciono, es posible instalar
varias versiones de ruby con el RVM, así que pueden probar entre versión y
versión.
rvm install 1.9.2
# Se crea un set de gemas llamado "arduino", aquí se almacenaran las gemas
como rails y muchas otras
rvm gemset create arduino
# Se especifica el gemset a utilizar junto con la versión de ruby.
rvm use 1.9.2@Arduino --default
# por ultimo se instala la gema RAILS
gem install rails -v 3.2.8
```

3.4. Creando un nuevo proyecto

Ya que se tiene el ambiente de trabajo instalado se procede, a crear un nuevo proyecto al cual se ha llamado "blog", se ejecuta en consola las siguientes sentencias.

Código 18. Creando un nuevo proyecto

```
terminal $

rails new blog
cd blog/
```

El comando "rails new blog" creara una carpeta en su directorio de trabajo denominado "blog". La carpeta blog tiene un número de carpetas auto generadas que componen la estructura de una aplicación Rails. La mayor parte del trabajo es en la carpeta "app/", pero aquí hay un resumen básico de la función de cada uno de los archivos y carpetas que Rails ha creado:

Tabla 5. Contenido de un proyecto en rails

Archivo / Carpeta	Propósito
app/	Contiene los controladores, modelos, vistas y recursos para su aplicación. Es la carpeta donde más se trabajara.
config/	Configurar las reglas de ejecución de la aplicación, rutas, bases de datos y mucho más.
config.ru	Configuración para servidores basados en rack utilizados para iniciar la aplicación.
db/	Contiene el esquema de base de datos actuales, así como las migraciones de la base de datos.
doc/	Documentación en general
Gemfile y Gemfile.lock	Estos archivos permiten especificar qué dependencias de gemas son necesarias para su aplicación Rails.
lib/	Módulos para su aplicación.
log/	Aquí se registra toda la actividad del sitio.
public/	La única carpeta que se muestra al mundo tal como es. Contiene los archivos estáticos y activos compilados.
Rakefile	Esto localiza archivos y tareas las cargas que se pueden ejecutar desde la línea de comandos. Las definiciones de tareas se definen a través de los componentes de Rails. En lugar de cambiar Rakefile, debe agregar sus propias tareas mediante la adición de archivos en el directorio lib/ tareas de su solicitud.
README. rdoc	Este es un breve manual de instrucciones para su aplicación. Usted debe editar este archivo para decir a los

	demás lo que su aplicación hace, cómo configurarlo, y así sucesivamente.
script/	Contiene el script de rails que inicia su aplicación y pueden contener otras secuencias de comandos que se utilizan para implementar o ejecutar la aplicación.
test/	Las pruebas unitarias, accesorios y otros aparatos de prueba.
tmp/	Los archivos temporales
vendor/	Un lugar para todo el código de terceros. En una aplicación Rails típica, esto incluye RubyGems, el código fuente de Rails (si tiene la opción de instalarlo en su proyecto) y los plugins que contienen funcionalidad adicional pre envasados.

3.4.1. Configurando la base de datos

Casi todas las aplicaciones Rails tienen que interactuar con una base de datos. La base de datos a usar se especifica en un archivo de configuración "config/database.yml". Si abre este archivo en una nueva aplicación Rails, verás una base de datos por defecto configurado para usar SQLite3. El archivo contiene secciones de tres ambientes diferentes en los que RAILS se puede ejecutar de forma predeterminada:

- El entorno de desarrollo (development): se utiliza en el equipo de desarrollo/local mientras interactúas manualmente con la aplicación.

- El entorno de prueba (test): se utiliza al ejecutar pruebas automatizadas.

- El entorno de producción (production): se utiliza al implementar la aplicación para que el mundo la pueda usar.

Cada una de las anteriores configuraciones especifica una base de datos a utilizar, por tanto se puede usar diferentes motores de base de datos para cada ambiente de trabajo, no se debería dejar en el ambiente de producción la base de datos en SQLite, se recomienda usar un motor mas potente como MySQL o postgreSQL.

Código 19. Configuración de la base de datos

config/Database.yml

```
development:
  adapter: sqlite3
  Database: db/development.sqlite3
  pool: 5
  timeout: 5000
test:
  adapter: sqlite3
  Database: db/test.sqlite3
  pool: 5
  timeout: 5000
production:
  adapter: sqlite3
  Database: db/production.sqlite3
  pool: 5
 timeout: 5000
```

Se puede usar otra base de datos como MySQL, PostgreSQL, Oracle, etc. La configuración es similar.
Luego de esto se puede levantar el proyecto, para esto es necesario un servidor se puede usar varios servidores entre estos Apache, pero para pruebas trae incluido uno por defecto WEBrick. Uno muy bueno para aplicaciones Rails es THIN. Que se instala con solo agregar una gema más en el proyecto, por el momento nos basta WEBrick.

Código 20. Levantando el servidor WEBrick

```
terminal $
rails s
```

Esto devolverá en consola:

```
=> Booting WEBrick
=> Rails 3.2.8 application starting in development on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
[2012-09-04 14:04:50] INFO  WEBrick 1.3.1
[2012-09-04 14:04:50] INFO  ruby 1.9.2 (2012-04-20) [i686-linux]
[2012-09-04 14:04:50] INFO  WEBrick::HTTPServer#start: pid=1340 port=3000
```

Ya podemos ver la primera página web ingresando a http://localhost:3000
Veremos lo siguiente.

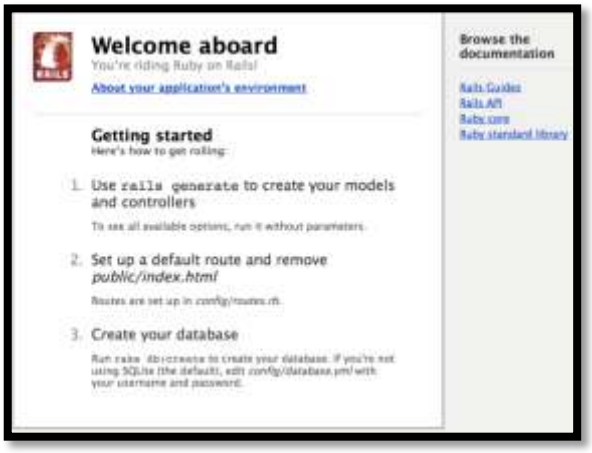

Figura 27. Página de inicio.

3.4.2. Rutas

La pagina que se esta viendo es una página estática se encuentra en public/index.html, como lógicamente no se desea estar viendo esta página al inicio hay que borrar el archivo.
Cualquier archivo que se encuentre en la carpeta public puede ser vista por un usuario desde su navegador. Dichas rutas no es necesario especificarlas, en la carpeta public se tendrá comúnmente imágenes, Hojas de estilos, librerías javascript, y cualquier otro documento que se quiera mostrar al usuario. Pero cuando se necesita un contenido activo, como consultas a la base de datos necesitamos atender peticiones especificas, por ejemplo:
http://localhost:3000/systems/sensor_01/edit
Con una estructura orientada a vistas (como PHP, PERL y otros), se necesitaría una estructura de carpetas, de tal manera que en la carpeta publica se tenga la carpeta system, en la carpeta system otra llamada sensor_01 y en la carpeta sensor_01 un script o pagina estática cuya salida es enviada al cliente, en rails esto indica que se va a editar el system llamado sensor_01, que básicamente es lógico pensarlo así. Si se quiere editar en PHP o en otro similar debería ser como la siguiente ruta.
http://localhost:3000/systems/edtit.php?system_path=sensor_01

En RAILS se especifican las rutas que se atenderán en el archivo config/routes.yml, con la línea "resources :systems", se agregan las rutas comunes para administrar una tabla de la base de datos. Además se crean automáticamente

helpers que permiten imprimir las rutas en el contenido de la página web. RAILS crea las rutas tomando como base la línea "resources :systems", son:

Tabla 6. Rutas por defecto de Rails

Helper	**Método**	**Ruta**	**Acción**
systems	GET	/systems	Index
new_system	GET	/systems/new	new
systems	POST	/systems	create
edit_system	GET	/systems/ID/edit	edit
system	PUT	/systems/ID	update
system	GET	/systems/ID	show
System	DELETE	/systems/ID	Destroy

Como se puede ver muchas rutas se repiten, por ejemplo "/systems/ID", si se envía por método GET (desde la URL del navegador) la petición la atiende la acción show, si se envía por método PUT, la petición la atiende la acción update y si se envía por método DELETE la petición la atiende la acción destroy.

Con estas rutas se tiene suficiente para la administración completa de la tabla systems en la base de datos, cada una de las acciones anteriores están definidas en el controlador de systems (el archivo systems_controler.rb), un controlador es una clase de ruby. Otro detalle que se puede ver en algunas rutas como, EDIT, DESTROY y SHOW, es que para ambas se debe especificar un identificador del registro sobre el cual se desea aplicar una acción (ID), este por defecto es el id en la base de datos (un numero único y auto incremental), pero se puede especificar cualquier otro campo que lo identifique como seo_url, url, etc. En el desarrollo de la aplicación se eligió un campo llamado PATH.

Si se quieren rutas extras al mismo controlador, o no se quieren todas las que se han agregado se pueden especificar cómo se muestra en el siguiente trozo de código:, en el cual se especifica que no se quieren todas las rutas, sino solo dos INDEX y SHOW, además agregamos otra por método GET, llamada "hardwares"

Código 21. Modificando las rutas.

config/routes.rb

```
resources :systems, :only => [:index, :show] do
  get 'hardwares', :on => :collection
end
```

En las rutas GET, se pueden enviar parámetros extras, por ejemplo si se quieren paginar los registros a mostrar en la index de systems, podemos poner una ruta como la siguiente:
http://localhost:3000/systems?page=3
El ID que se envía en una ruta como la edit
(http://localhost:3000/systems/ID/edit), también es un parámetro.
Ambos se pueden obtener de forma similar en el controlador con la instrucción "params[:page]" y "params[:id]" respectivamente.
Hay otras formas de agregar rutas, por ejemplo un "match", también se debe especificar la ruta de la index ósea que controlador atenderá la petición http://localhost:3000/. Esto se hace de la siguiente forma:
root :to => "systems#index"
Donde se está especificando que la ruta ROOT apuntara hacia un controlador llamado "systems" a la acción "index", por tanto se podrá obtener el mismo resultado ingresando desde el navegador a http://localhost:3000/ o http://localhost:3000/systems/.

3.4.3. Controladores

En el controlador procesamos los datos que vienen del cliente, por ejemplo si la ruta fuese la index http://localhost:3000?times=5, como vemos hemos agregado un parámetro llamado times que tiene el valor de 5, entonces en el controlador "systems" en la acción index, (según lo especificamos en las rutas), actuamos sobre este parámetro.

Código 22. Obteniendo parámetros en el controlador.

app/controllers/systems_controller.rb

```
class HomeController < ApplicationController
  def index
    if params[:times]
      ...
    else
      ...
    end
  end
end
```

3.4.4. Vistas

Las vistas es lo que regresa al cliente, después de una petición, por ejemplo http://localhost:3000/systems/sensor_01, el usuario espera ver detalles del sistema llamado sensor_01, por tanto, en el controlador se hace la consulta a la base de datos, se guarda el objeto devuelto en una variable y en la vista se muestra dicho objeto en código HTML (o cualquier otro formato que manejemos). Un ejemplo sencillo de esto es suponiendo que tenemos, un sensor_01 con un campo llamado "name", se hace la consulta a la base de datos en el controlador y se guarda el resultado el resultado en una variable llamada "@system", entonces en la vista mostraríamos el nombre de la siguiente forma.

Código 23. Vista en html sencilla.

app/views/systems/show.html.erb

```
<html>
  <body>
    <h1><%= @system.name %> </h1>
  </body>
</html>
```

En el código anterior estamos mostrando en formato HTML, el nombre del sistema consultado.

3.4.5. Conectándose a la base de datos (modelos)

Los modelos sirven para conectarnos a la base de datos sin tener que hacer consultas SQL especificas para una base de datos en particular, además, no hay que especificar el nombre de los campos en la base de datos, es más, no hay que hacer ninguna configuración, solo creamos el modelo. El modelo no es más que al igual que el controlador una clase de RUBY. Pero el modelo se hereda de activerecord, que es el que se encarga de hacer las consultas, por ejemplo si en la base de datos se tiene la tabla systems, con los campos name, description y id, entonces se crea en "app/models/"un archivo llamado "system.rb", dentro del se define la clase de la siguiente forma:

Código 24. Creando un modelo

app/models/system.rb

```
class System < ActiveRecord::Base
end
```

Ahora se pueden hacer consultas a la base de datos, por ejemplo

Código 25. Consultas a la base de datos a través de modelos.

irb>

```
System.all
System.first
systems = System.where("id > 10").order("id ASC")
for system in systems
  puts  "#{system.id} #{system.name} #{system.description}"
end
```

Pero ¿en qué momento se especifico la tabla de la base de datos donde están los registros, los campos de la base de datos, etc.? La respuesta es que en el modelo no se necesita ninguna configuración ActiveRecord asume que si el modelo se llama system, la base de datos debe de llamarse systems y obtiene de forma automática los campos de la base de datos. Pero y ¿si no quiero seguir las convenciones de rails o otra persona creo las tablas? No hay problema, también se puede configurar pero si se quiere ahorrar tiempo y trabajo, lo mejor es trabajar de acuerdo a la segunda filosofía de rails "Convención sobre configuración".

3.4.6. Administración de base de datos en pocos minutos (scaffold)

Si lo que se necesita es una administración de una tabla en la base de datos, Rails tiene un generador, que crea todo lo necesario para administrar la tablas, incluido las rutas, controlador con las siete acciones básicas, vistas en HTML, modelos, y otros más. Tenemos lo necesario y nos dedicamos a dar estilos, validaciones y programación de funcionalidades específicas. A continuación se muestra un scaffold en el cual se generara todo lo necesario para administrar publicaciones para un blog.

Código 26. Creando un scaffold.

```
terminal $

rails generate scaffold Post name:string title:string content:text
rake db:migrate
```

Automáticamente se generan una serie de archivos, que se muestran en la siguiente tabla con una breve descripción.

Tabla 7. Archivos autogenerados con scaffold

db/migrate/20121010214725_create_posts.rb	La migración es la encargada de crear la tabla de posts en la base de datos, la segunda línea del código corre la migración, que en este caso le indica crear una tabla en la base de datos
app/models/post.rb	El modelo Post con el cual realizaremos las consultas y otras tareas sobre la tabla posts.
config/routes.rb	Crea las rutas de las que se hablo antes, el archivo ya existe solo agrega una línea más "resources :posts"
app/controllers/posts_controller.rb	El controlador a donde apuntaran las rutas, ahí se definen las acciones básicas sobre la tabla posts, edit, create, etc.
app/views/posts/index.html.erb	Muestra todos los posts con sus respectivos campos en una tabla.
app/views/posts/edit.html.erb	Sirve para enviar un formulario donde se pueden editar los campos de un post ya existente

app/views/posts/show.html.erb	Muestra un post con todos sus detalles
app/views/posts/new.html.erb	Sirve para enviar un formulario donde se pueden llenar los campos para crear un nuevo post
app/views/posts/_form.html.erb	Como se menciono antes la primera filosofía de rails, DRY (No repetir código), el formulario de edit y new, es el mismo, por tanto se crea un archivo que lo contiene y en las vistas new y edit, solo se llama al archivo.

Se generan algunos otros archivos que no interesan por el momento. El archivo "_form.html.haml", es común para la vista edit y new, estos archivos son llamados "partial".
Ahora podemos navegar entre las diferentes rutas que se nos generaron.
Podemos hacer que http://localhost:3001/, apunte a http://localhost:3001/posts, para esto borramos el archivo "public/index.html" y editamos el archivo de las rutas, nos quedaría así:

Código 27. Rutas para los posts.

config/routes.rb

```
Blog::Application.routes.draw do
  resources :posts
  root :to => "posts#index"
end
```

En el navegador podemos acceder a las siguientes rutas.

- http://localhost:3000/posts o http://localhost:3000/ Muestra la lista de todos los posts existentes, si no hemos creado ninguno, aparecerá una tabla vacía. En la siguiente imagen se muestra una captura de cómo debería de verse.

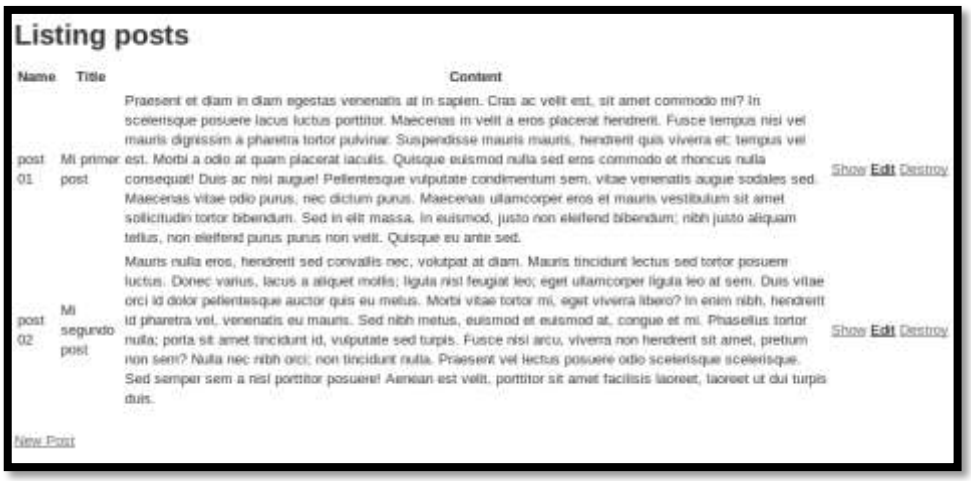

Figura 28. Listado de los posts creados.

- http://localhost:3000/posts/new Para crear un nuevo post, se verá un formulario, con los campos en la base de datos.

Figura 29. Formulario para la creación de un nuevo post.

- http://localhost:3000/posts/1/edit Edita un post existente con un formulario similar a la anterior pero con los campos pre llenados con los datos del post que se está editando.

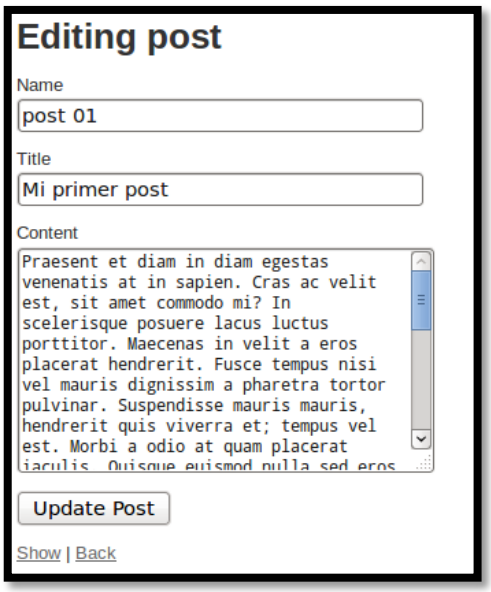

Figura 30. Formulario para la edición de un post.

- http://localhost:3000/posts/1 Muestra un post con todos los datos del post.

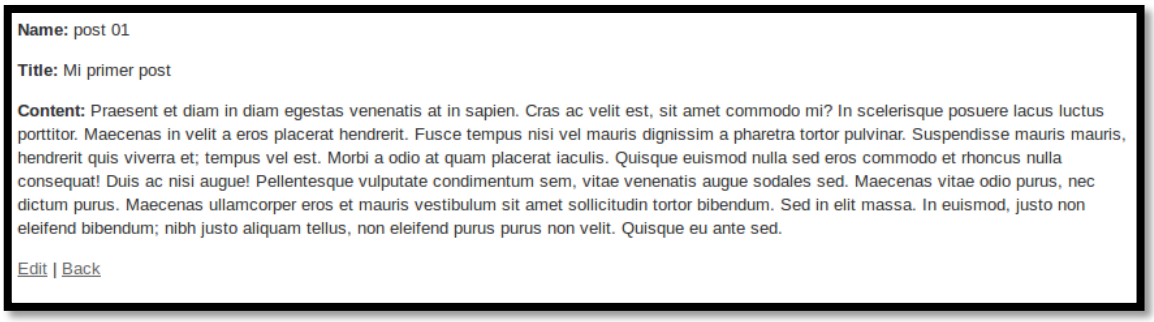

Figura 31. Vista en detalle del un post.

Quiero comentar que hay una materia donde se hace esto mismo con PERL, y nos tardamos 16 semanas en lograr el mismo resultado que aquí ha tomado unos cinco minutos.
Hay que notar, que en cada pantalla se ofrece una serie de enlaces, para que el usuario, pueda acceder a las demás rutas.

3.5. Creando un gran proyecto (arduino Data Logger)

Ahora bien, si se quiere administrar la base de datos y sus contenidos, pues ya hay administradores como phpmyadmin, phppgadmin, pgAdmin3, sqliteManager, etc. Pero cuando se necesita mostrar datos a clientes, graficas, procesamiento de información, etc. no basta con esto. Aquí es donde comienza el largo trabajo, dar estilos (CSS), graficar (JavaScrip – jquery – flot), relacionar tablas, etc.

3.5.1. Tablas y modelos

Lo primero que se debe hacer al comenzar un proyecto es diseñar una estructura de base de datos, cuan do el proyecto crece, la estructura puede ser muy compleja, en este caso se tendrán las siguientes tablas.

Tabla 8. Tablas en la base de datos

Tabla	Descripción
users	Los datos que un Arduino (hardware) este censando, son enviados al sistema, almacenándolos en la base de datos, pero ¿a quién se le mostraran?, para esto se creó una tabla donde se almacenan los usuarios (users), un usuario tiene acceso solo a los datos guardados por su Hardware, no tiene acceso, a otros datos.
hardwares	Se diseño el proyecto para que pueda manejar más de un solo Arduino, así que se creó la tabla hardwares, en esta se guarda básicamente un identificador para cada Arduino, de modo que no cualquier Arduino pueda enviar sus datos, solo aquellos autorizados, además, cada registro está ligado a un usuario, por tanto un usuario puede tener muchos harwares y por tanto un usuario solo podrá ver los datos de hardwares que estén ligados a él.
systems	Esta es una tabla donde se almacenan los detalles de un sensor (system), como un hardware pertenece a un usuario, un system pertenece a un Hardware, y al mismo tiempo a un usuario, por tanto un usuario solo podrá ver datos de los sensores ligados a sus Arduinos.
lectures	En esta tabla se guardan todas las lecturas (lecture) de todos los sensores conectados a cualquiera de todos los

	Arduinos. Y de forma idéntica a las anteriores, una lectura está ligada a un sensor. Se guarda tanto el valor digital enviado por el Arduino como el valor analógico que se calcula al momento de guardar el registro.
kinds	¿Cómo convertir los diferentes tipos de datos digitales a un valor físico? se creó esta tabla donde se guardan tipos de sensores, un sensor está ligado a un registro de esta tabla, donde se le especifica la ecuación de conversión de datos.
Arduino_types	Un hardware puede ser de varios tipos, MEGA, UNO, etc., se creó una administración para esto, un hardware está ligado a un registro en esta tabla, de esta manera se puede saber si un hardware es de tipo MEGA, UNO, o cualquier otro.

A continuación se muestra un esquema que muestra las relaciones entre las diferentes tablas.

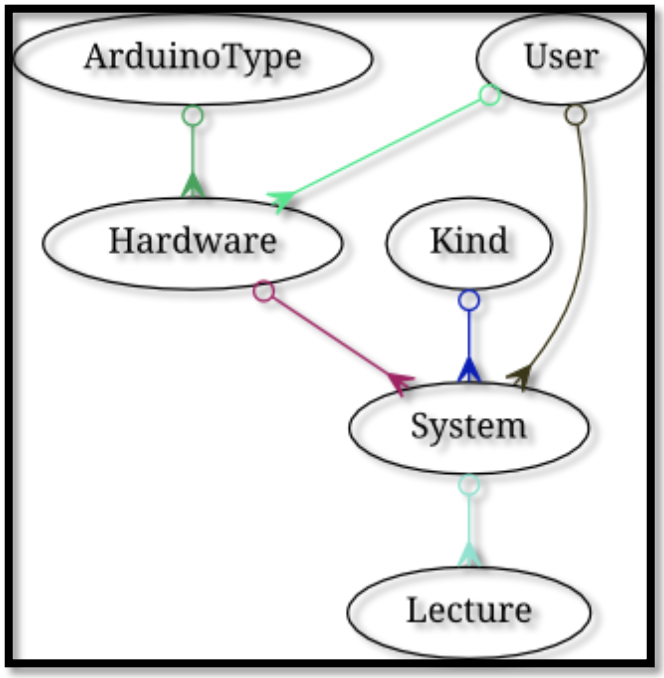

Figura 32. Esquema de relaciones entre las tablas.

En la siguiente figura se muestra la misma relación, pero especificando, todos los campos de cada tabla.

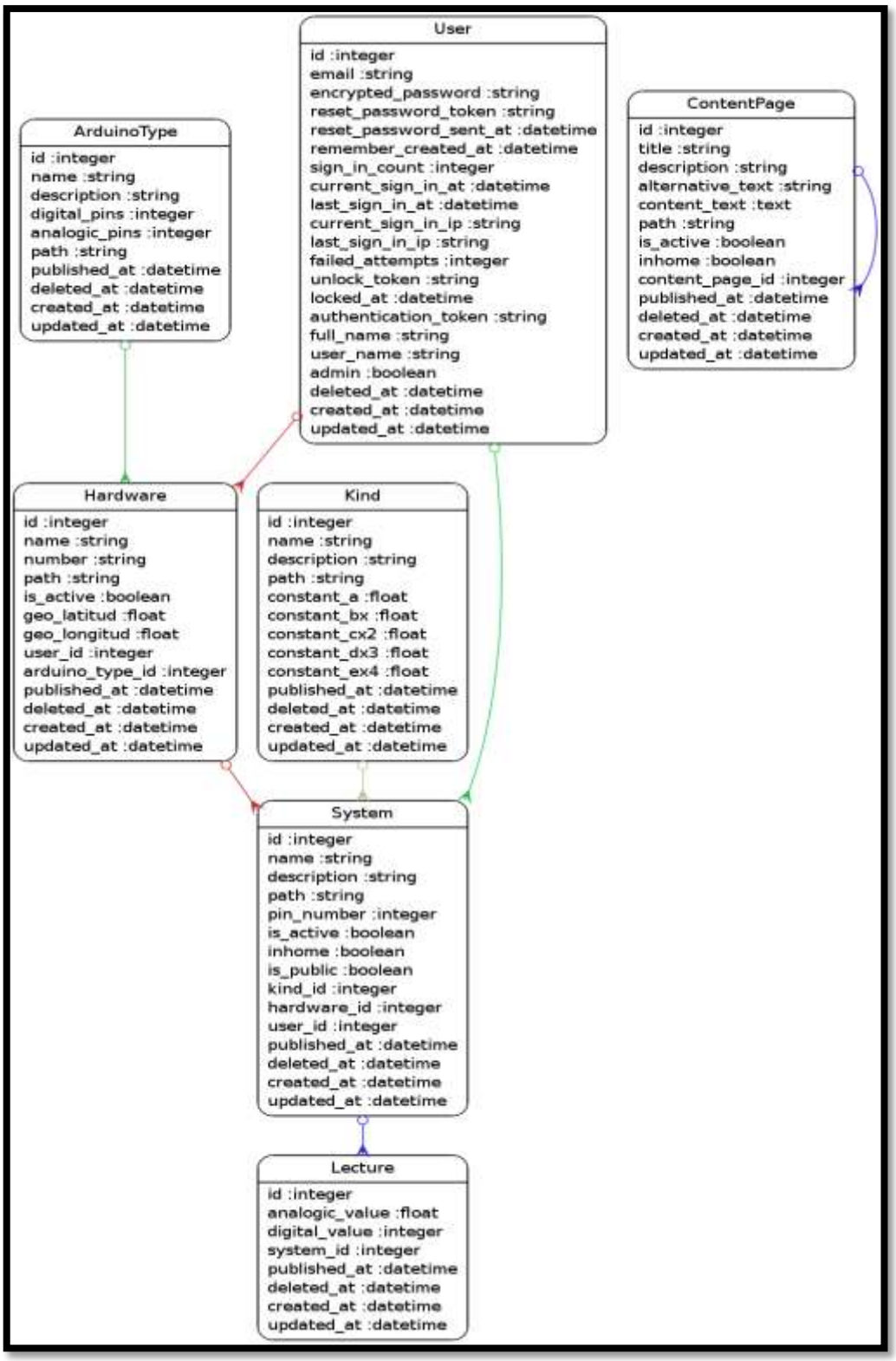

Figura 33. Esquema de relación de tablas con detalles.

Los Modelos aunque no es necesario modificarlos para que funcionen correctamente se agregaron unas pocas líneas, principalmente para las relaciones y validaciones.
Por ejemplo como se desea que un sensor (system) pertenezca a un Arduino (hardware), entonces en el modelo System declaramos "belongs_to :hardware", y en el modelo Hardware "has_many :systems", ahora podemos hacer consultas relacionadas.

Como esta la aplicación, se puede guardar un registro con los valores que se quiera, de hecho se puede guardar sin ningún valor, es decir completamente vacío, pero no se quiere que esto pase, por tanto habrá que validar algunos campos, especificando, que no puede ir vacío por ejemplo el nombre de un sensor, el formato correcto de un correo, la longitud mínima de caracteres de una contraseña, que no hayan nombres de usuarios repetidos, etc. A continuación se muestran las validaciones usadas en el modelo de System, similar se ha hecho en los otros modelos.

Código 28. Validaciones y atributos accesibles para el modelo System

app/models/system.rb

```
Class System < ActiveRecord::Base
  attr_accessible   :name, :description, :path, :pin_number, :is_active, :inhome, :is_public,
                    :kind, :user, :hardware, :kind_id, :user_id, :hardware_id, :published_at,
                  :deleted_at
  validates   :name, :description, :pin_number, :kind_id, :hardware_id, :presence => true
  validates   :pin_number, :numericality => true


.
.
.
end
```

3.5.2. Rutas

Rutas, básicamente se crean rutas para administrar la base de datos, dentro del namespace "admin", de modo que para ver la lista de sensores (tabla systems), se debe ingresar a la ruta, http://localhost:3000/admin/systems, similar para las demás rutas, esto se muestra en el siguiente código.

Código 29. Rutas accesibles por el administrador.

config/routes.rb

```
namespace :admin do
  resources :users, :only => [:index, :show, :destroy]
  resources :systems
  resources :lectures
  resources :kinds
  resources :hardwares
  resources :Arduino_types
end
```

Para las rutas accesibles a los usuarios no administradores, se crearon solo las rutas necesarias para mostrar datos. Esto se muestra en el siguiente código.

Código 30. Rutas accesibles por los usuarios loggeados.

config/routes.rb

```
resources :hardwares, :only => [:index, :show] do
  resources :systems, :only => [:show] do
  end
end
```

Se uso una gema DEVISE que crea rutas que sirven para registrar usuarios, crear sesión, cerrar sesión, etc. Estas se modificaron un poco para que apuntaran a nuestro controlador ya que por defecto cualquier persona puede registrase, y en esta aplicación solo el administrador puede registrar nuevos usuarios. Estas rutas se muestran en el siguiente código.

Código 31. Rutas para manejo de sesión con DEVISE.

config/routes.rb

```
devise_for :users,
  :controllers => {
    :sessions => "users/sessions",
    :registrations => "users/registrations",
    :passwords => "users/passwords",
    :unlocks => 'users/unlocks' } do
end
```

Ahora se mostrara cómo se administra una tabla por ejemplo systems, ya que se han generado las rutas, se puede acceder al index de systems, como administrador en la siguiente url.
http://localhost:3000/admin/systems.

Cuando se crea un nuevo registro o se edita el campo de referencia a otra tabla, por ejemplo si se estuviese editando un sensor (System) y se quiere que este sensor pertenezca a un hardware llamado "Mi primer Arduino", que además el sensor sea de tipo (kind) "Temperatura DS18B20", al registro que se va a crear hay que especificarle las referencias hardware_id (especifica el hardware al cual estará asociado) y (kind_id especifica el tipo de sensor que se usara) estos campos son de tipo entero, pero si el usuario ve un campo system_id y otro kind_id, donde debe ingresar un numero entero de referencia a su hardware y tipo de sensor respectivamente, le parecerá difícil la operación, principalmente porque le será difícil recordar el número identificador de cada hardware y tipo de sensor, entonces para facilitarle el trabajo se crea un campo de selección múltiple donde se muestran los hardwares y otro donde se muestran los tipos de sensores, para que el elija "por nombre" el que desee. Al seleccionar el hardware "Mi primer Arduino" sin darse cuenta esta seleccionando el ID de este hardware.
En la siguiente imagen se muestra como se ve el formulario con el campo de selección.

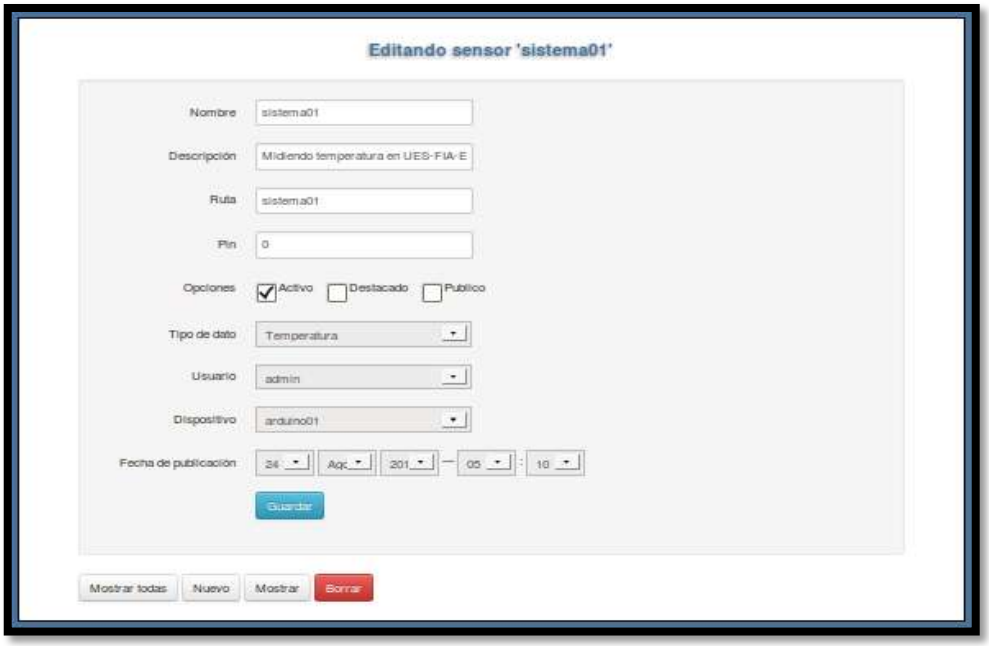

Figura 34. Formulario con campos de selección múltiple.

Se usaron hojas de estilos y javascript para poder lograr el diseño de este sitio, lógicamente este es un arduo trabajo, las hojas de estilos y la programación en JavaScript queda fuera de los objetivos de este documento, sin embargo cuando sea necesario, se explicaran algunos fragmentos de código, principalmente de JavaScript como por ejemplo crear graficas con jQuery-Flot.

3.5.3. Vistas y controladores para la administración de tablas

En esta sección se mostrara el desarrollo del código que hace posible administrar los registros en la base de datos, siendo tantas tablas y se mostrara como se a realizado este trabajo únicamente para la tabla hardwares, los pasos son idénticos para las tablas restantes.

En la página principal se muestran los enlaces de navegación, el usuario debe estar loggeado como administrador e ingresar en el enlace que dice "Arduinos".
En la imagen siguiente se muestra el enlace.

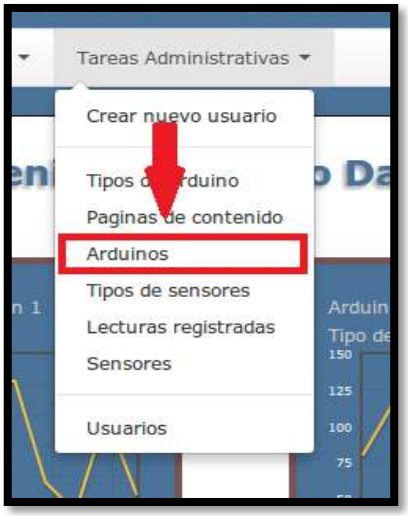

Figura 35. Tareas administrativas.

El controlador que atiende las peticiones para las rutas de hardwares es el que se muestra en el siguiente código.

Código 32. Controlador para hardwares

app/controllers/admin/hardwares_controller.rb

```
class Admin::HardwaresController < Admin::AdminController
  before_filter :load, :only => [:show, :edit, :destroy, :update]
  def index
    @users = User.all
  end
  def new
    @hardware = Hardware.new
  end
  def create
    @hardware = Hardware.new(params[:hardware])
    if @hardware.save
      redirect_to [:admin, @hardware], :notice => t(:successfull)
    else
      render :action => 'new'
    end
  end
  def update
    if @hardware.update_attributes(params[:hardware])
      redirect_to [:admin, @hardware], :notice => t(:successfull)
    else
      render :action => 'edit'
    end
  end
  def destroy
    @hardware.update_attributes(:deleted_at => Time.now)
    redirect_to admin_hardwares_url, :notice => t(:successfull)
  end
  private
    def load
      @hardware = Hardware.find_by_path(params[:id])
    end
end
```

A continuación se explicaran cada una de las acciones de este controlador, asociadas a una ruta. Al dar clic en Arduinos se ha ingresado ya a este controlador.

a) Con esto se ha entrado a la ruta index de hardwares, en esta se muestran todos los Arduinos registrados, esta ruta es atendida por el controlador

hardwares acción index, este se muestra en la siguiente ruta. La ruta index http://localhost:3000/admin/hardwares/ , muestra todos los registros, para esto hay que hacer en el controlador una consulta que devuelva todos los registros.

Si revisamos el código anterior vemos que se muestra en la acción index, una consulta donde se obtienen todos los usuarios, en algunas ocasiones el numero de registros en este caso usuarios puede ser muy alto, en este caso habría que mostrar los usuarios en varias paginas, esto fue necesario en la administración de lecturas registradas, ya que se tenían miles de lecturas, para esto se uso una gema llamada will_paginate.

Código 33. Acción index del controlador de hardwares

app/controllers/admin/hardwares_controller.rb

```
  def index
    @users = User.all
  end
```

La vista tiene el siguiente código donde se crea una tabla con todos los registros. Primero se recorren todos los usuarios y para cada uno de ellos se obtienen los hardwares asociados a él (user.hardwares), luego se recorre cada uno de los hardwares que le pertenecen dibujándolos debajo de su nombre.

Código 34. Vista para la acción index de hardwares

app/views/admin/hardwares/index.html.haml

```
- title "Arduinos"
%table.table.table-bordered
  %thead
    %tr
      %th= 'Nombre'
      %th= 'Tipo de Arduino'
      %th= 'Activo'
      %th= 'Usuario'
      %th= ' '
  %tbody
    - @users.each do |user|
      %tr
```

```
        %th(colspan='5')
          %p(style='color:#4B7399')= "Usuario: #{user.full_name} #{user.user_name})"

      - user.hardwares.each do |hardware|
        %tr
          %td= hardware.name
          %td= hardware.Arduino_type.name rescue nil
          %td= hardware.is_active ? 'SI' : 'NO'
          %td= hardware.user.user_name
          %td= render :partial => 'links', :locals => {:mylocal => hardware,
:no_show_new => true}

= render 'links'
```

El código html enviado al navegador es el que se muestra a continuación. Donde se puede notar que es mucho más extenso, el código escrito en HAML es mucho más compacto y ordenado.

Código 35. HTML generado para la acción index de hardwares

http://localhost:3000/admin/hardwares/

```
<table class='table table-bordered'>
  <thead>
    <tr>
      <th>Nombre</th>
      <th>Tipo de Arduino</th>
      <th>Activo</th>
      <th>Usuario</th>
      <th> </th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th colspan='5'>
        <p style='color:#4B7399'>Usuario: Luis Gómez (usuario01)</p>
      </th>
    </tr>
    <tr>
      <td>Arduino02</td>
      <td>Mega</td>
```

```
      <td>SI</td>
      <td>usuario01</td>
      <td>
        <p>
          <a                href="/admin/hardwares/hardware_1349754144/edit"
class="btn">Editar</a>
          <a                      href="/admin/hardwares/hardware_1349754144"
class="btn">Mostrar</a>
          <a  href="/admin/hardwares/hardware_1349754144"  class="btn  btn-
danger" Data-confirm="¿Esta seguro?" Data-method="delete">Borrar</a>
        </p>
      </td>
    </tr>
    <tr>
      <td>Arduino03</td>
      <td>Mega</td>
      <td>SI</td>
      <td>usuario01</td>
      <td>
        <p>
          <a                href="/admin/hardwares/hardware_1349754164/edit"
class="btn">Editar</a>
          <a                      href="/admin/hardwares/hardware_1349754164"
class="btn">Mostrar</a>
          <a  href="/admin/hardwares/hardware_1349754164"  class="btn  btn-
danger" Data-confirm="¿Esta seguro?" Data-method="delete">Borrar</a>
        </p>
      </td>
    </tr>
    <tr>
      <th colspan='5'>
        <p style='color:#4B7399'>Usuario: Luis Alfredo (usuario02</p>
      </th>
    </tr>
    <tr>
      <th colspan='5'>
        <p style='color:#4B7399'>Usuario: Luis calidonio (admin)</p>
      </th>
    </tr>
    <tr>
      <td>Arduino01</td>
      <td>Mega</td>
      <td>SI</td>
```

```
      <td>admin</td>
      <td>
        <p>
          <a                    href="/admin/hardwares/hardware_1349649481/edit"
class="btn">Editar</a>
          <a                         href="/admin/hardwares/hardware_1349649481"
class="btn">Mostrar</a>
          <a   href="/admin/hardwares/hardware_1349649481"   class="btn   btn-
danger" Data-confirm="¿Está seguro?" Data-method="delete">Borrar</a>
        </p>
      </td>
    </tr>
  </tbody>
</table>
```

Por último el usuario lo que podrá ver es una tabla esta se muestra en la siguiente imagen. En donde se puede ver todos los usuarios en color azul, y los Arduinos debajo de cada usuario, en este caso el "usuario01", tiene dos Arduinos, el "usuario02" no tiene ninguno y el usuario "admin" tiene solo un Arduino.

Figura 36. Home page del sitio

Se está usando la gema HAML, que facilita el escribir código HTML, ya que no hay que estar cerrando etiquetas, y además obliga a usar identación el código es más legible.
Podemos con los enlaces disponibles crear nuevos Arduinos, el ultimo enlace, es para registrar un nuevo Arduino. Como se puede notar también se ha omitido el código tanto HAML como HTML para el layout ósea el contenedor de la tabla que lleva las etiquetas <html><head><body>… ya que esto se repite en cada vista solo hay que escribirlo una vez.

b) Al hacer clic en el enlace para crear un nuevo hardware http://localhost:3000/admin/hardwares/new, en ese momento se hace una petición al servidor, la ruta apunta al controlador HARDWARES acción NEW. La mejor forma de crear un registro en rails es con los "helpers" que este nos provee, con el helper "form_for", se puede recorrer un objeto dibujando el formulario, junto con sus campos y los valores que trae, si el objeto se obtuvo de una consulta en la base de datos, se tienen valores guardados se colocaran esos valores en los campos respectivos, si el objeto es una instancia aun no guardada, como en este caso, el formulario aparecerá en blanco, una ventaja más de crear un formulario de esta forma es que puede ser usado tanto para la acción new como para la acción edit.

Código 36. Acción new en el controlador hardwares

app/controllers/admin/hardwares_controller.rb

```
  def new
    @hardware = Hardware.new
  End
```

Se crea un objeto hardware pero no se guarda en la base de datos, como no se especifica la vista a ser enviada al cliente, se envía la vista con el mismo nombre que la acción en el controlador, esta se muestra en el siguiente bloque de código.

Código 37. Vista de la acción new

app/views/admin/hardwares/new.html.haml

```
- title "crear nuevo dispositivo"
= render 'form'
```

El "render" llama a otro trozo de código, este se muestra a continuación.

Código 38. Creación del formulario para un hardware

app/views/admin/hardwares/_form.html.haml

```
- content_for :head do
  = javascript_include_tag "rails.validations"
= form_for [:admin, @hardware], :validate => true, :html => {:class => 'well form-horizontal'} do |f|
  %fieldset
    .control-group
      = f.label :name, 'Nombre', :class => 'control-label'
      .controls
        = f.text_field :name
    .control-group
      = f.label :Arduino_type_id, 'Tipo de Arduino', :class => 'control-label'
      .controls
        = f.collection_select :Arduino_type_id, ArduinoType.all, :id, :name
    .control-group
      .controls
        %label.checkbox
          = f.check_box :is_active
          = f.label :is_active, 'Activo'
    .control-group
      = f.label :user_id, 'Usuario',:class => 'control-label'
      .controls
        = f.collection_select :user_id, User.all, :id, :user_name
    .control-group
      = f.label :geo_latitud, 'Latitud', :class => 'control-label'
      .controls
        = f.text_field :geo_latitud
    .control-group
      = f.label :geo_longitud, 'Longitud', :class => 'control-label'
      .controls
        = f.text_field :geo_longitud
    .control-group
      = f.label :published_at, 'Fecha de publicación', :class => 'control-label'
      .controls
        = f.datetime_select :published_at, {}, :style => 'width:60px'
    .control-group
      = f.label :path, 'Ruta', :class => 'control-label'
      .controls
        = f.text_field :path
    .control-group
      .controls
```

```
        = f.submit 'Guardar', :class => 'btn btn-info'
= render :partial => 'links', :locals => {:mylocal => @hardware}
```

El código que el cliente puede ver es el siguiente.
Los campos de selección múltiple, tienen más opciones, se han recortado para hacer más compacto el código. También se omitido todo el layout.

Código 39. HTML generado para la vista new de hardwares

http://localhost:3000/admin/hardwares/new

```
<form accept-charset="UTF-8" action="/admin/systems" class="well form-horizontal" data-validate="true" id="new_system" method="post" novalidate="novalidate">
  <div style="margin:0;padding:0;display:inline">
    <input name="utf8" type="hidden" value="&#x2713;" />
    <input name="authenticity_token" type="hidden" value="OxwmB849bv8VNxcf8IiKaoGI6cN1CAMFjn6WKUYnUdc=" />
  </div>
  <fieldset>
    <div class='control-group'>
      <label class="control-label" for="system_name">Nombre</label>
      <div class='controls'>
        <input data-validate="true" id="system_name" name="system[name]" size="30" type="text" />
      </div>
    </div>
    <div class='control-group'>
      <label class="control-label" for="system_description">Descripción</label>
      <div class='controls'>
        <input data-validate="true" id="system_description" name="system[description]" size="30" type="text" />
      </div>
    </div>
    <div class='control-group'>
      <label class="control-label" for="system_path">Ruta</label>
      <div class='controls'>
        <input id="system_path" name="system[path]" size="30" type="text" />
      </div>
    </div>
    <div class='control-group'>
      <label class="control-label" for="system_pin_number">Pin</label>
      <div class='controls'>
```

```
        <input  data-validate="true"  id="system_pin_number"  name="system[pin_number]"
type="number" />
      </div>
    </div>
    <div class='control-group'>
      <label class='control-label'>Opciones</label>
      <div class='controls'>
        <label class='checkbox inline' for='system_is_active'>
          <input    name="system[is_active]"    type="hidden"    value="0"    /><input
id="system_is_active" name="system[is_active]" type="checkbox" value="1" />
          <span>Activo</span>
        </label>
        <label class='checkbox inline' for='system_inhome'>
          <input     name="system[inhome]"     type="hidden"     value="0"     /><input
id="system_inhome" name="system[inhome]" type="checkbox" value="1" />
          <span>Destacado</span>
        </label>
        <label class='checkbox inline' for='system_is_public'>
          <input    name="system[is_public]"    type="hidden"    value="0"    /><input
id="system_is_public" name="system[is_public]" type="checkbox" value="1" />
          <span>Publico</span>
        </label>
      </div>
    </div>
    <div class='control-group'>
      <label class="control-label" for="system_kind_id">Tipo de dato</label>
      <div class='controls'>
        <select Data-validate="true" id="system_kind_id" name="system[kind_id]"><option
value="1">Temperatura</option>
        <option value="2">Humedad</option></select>
      </div>
    </div>
    <div class='control-group'>
      <label class="control-label" for="system_user_id">Usuario</label>
      <div class='controls'>
        <select           id="system_user_id"           name="system[user_id]"><option
value="1">admin</option>
        <option value="2">calidonio</option>
        <option value="3">Luis</option></select>
      </div>
    </div>
    <div class='control-group'>
      <label class="control-label" for="system_hardware_id">Dispositivo</label>
```

```
      <div class='controls'>
        <select              Data-validate="true"              id="system_hardware_id"
name="system[hardware_id]"><option value="1">Arduino01</option>
        <option value="2">Arduino02</option>
        <option value="3">Data Logger</option></select>
      </div>
    </div>
    <div class='control-group'>
      <label       class="control-label"       for="system_published_at">Fecha       de
publicación</label>
      <div class='controls'>
        <select       id="system_published_at_3i"       name="system[published_at(3i)]"
style="width:60px">
          <option value="1">1</option>
          <option value="2">2</option>
        </select>
        <select       id="system_published_at_2i"       name="system[published_at(2i)]"
style="width:60px">
          <option value="7">Julio</option>
          <option value="11">Noviembre</option>
          <option value="12">Diciembre</option>
        </select>
        <select       id="system_published_at_1i"       name="system[published_at(1i)]"
style="width:60px">
          <option value="2013">2013</option>
          <option value="2014">2014</option>
          <option value="2015">2015</option>
          <option value="2016">2016</option>
          <option value="2017">2017</option>
        </select>
         &mdash;
        <select       id="system_published_at_4i"       name="system[published_at(4i)]"
style="width:60px">
          <option value="16">16</option>
          <option value="17">17</option>
          <option value="21">21</option>
          <option value="22">22</option>
          <option value="23">23</option>
        </select>
        <select       id="system_published_at_5i"       name="system[published_at(5i)]"
style="width:60px">
          <option value="53">53</option>
          <option value="54">54</option>
```

```
          <option value="55">55</option>
          <option value="56">56</option>
          <option value="57">57</option>
          <option value="58">58</option>
          <option value="59">59</option>
        </select>
      </div>
    </div>
    <div class='control-group'>
      <div class='controls'>
        <input class="btn btn-info" name="commit" type="submit" value="Guardar" />
      </div>
    </div>
  </fieldset>
</form>
```

Ahora el cliente tiene un formulario que llenar, el diseño como puede ser visto por el usuario a través del navegador se muestra en la siguiente imagen.

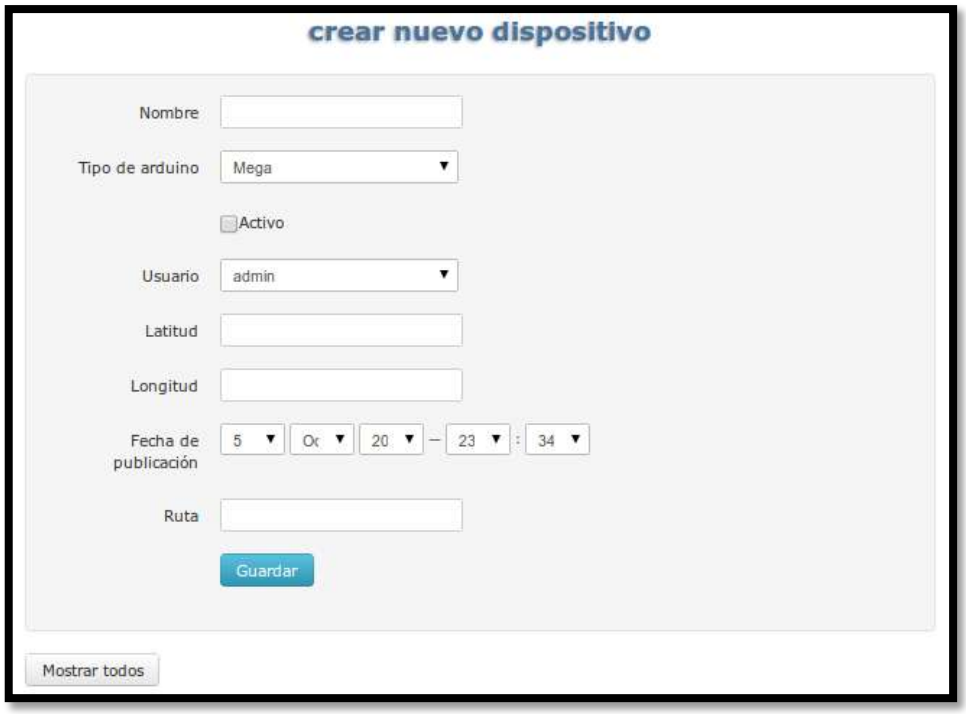

Figura 37. Crear un nuevo Data Logger.

Note que en las primeras líneas del código se ha escrito:

Código 40. Incluyendo validaciones con JavaScript

app/views/admin/hardwares/_form.html.haml

```
- content_for :head do
  = javascript_include_tag "rails.validations"
```

Esto nos incluye un archivo, que nos permite validar en el lado del cliente (navegador), todas las validaciones que hicimos en el modelo, esto se logra a través de una gema llamada Client Side Validator (http://railscasts.com/episodes/263-client-side-validations), y no se puede enviar el formulario, hasta que satisfaga cada una de las validaciones. Podría ser que el usuario configure su navegador para que trabaje sin JavaScript, entonces podría enviar los datos sin validarse, por ejemplo podría ingresar un correo repetido, etc.

c) Al hacer el submit del formulario, se envían los datos al controlador, que los procesa sin importar si fueron validados por JavaScript o no y trata de guardarlos. El código en el controlador en la acción CREATE es el siguiente.

Código 41. Acción create de hardwates

app/controllers/admin/hardwares_controller.rb

```
def create
  @hardware = Hardware.new(params[:hardware])
  if @hardware.save
    redirect_to [:admin, @hardware], :notice => t(:successfull)
  else
    render :action => 'new'
  end
end
```

Como se puede ver se valida con un IF (if @hardware.save), En el momento de guardar se corren las validaciones que tengamos en el modelo automáticamente, y ya que a esto el usuario no tiene acceso, de nada le habrá servido saltarse las validaciones con JavaScript. Si no se puede guardar se envía la vista NEW, pero el objeto que se recorre es el @hardware que se acaba de tratar de guardar, por tanto los campos aparecerán llenos con los datos que había escrito con anterioridad y por lo tanto y para suerte del usuario no será necesario que vuelva a llenar todo de nuevo, solo deberá corregir lo que se le indique. Si en cambio, el registro fue guardado satisfactoriamente, se re-direcciona hacia la ruta SHOW, de dicho

hardware, o a otra ruta conveniente. Como se puede notar, no hay una vista para la acción create, se renderiza la acción new, o se redirige a la ruta show.

d) A la ruta show se puede acceder desde http://localhost:3000/admin/harwares/path, donde path es el identificador del hardware, y puede ser obtenido de los params con "params[:id]". En el controlador no hay un método show, ya que el trabajo de este método seria encontrar en la base de datos el objeto que vamos a mostrar, sin embargo esto también se hace en las acciones edit, update y destroy, por tanto ejecutamos un método común para estas acciones este se muestra en el siguiente trozo de código.

Código 42. Buscando el objeto solo para las acciones show, edit, destroy y update

app/controllers/admin/hardwares_controller.rb

```
before_filter :load, :only => [:show, :edit, :destroy, :update]
private
  def load
    @hardware = Hardware.find_by_path(params[:id])
  end
```

Luego de haber encontrado el registro que se desea mostrar se dibuja la vista, en este caso se dibuja una tabla con los nombres y valores de los campos de este registró.

Código 43. Vista de la acción show para el controlador hardwares

app/views/admin/hardwares/show.html.haml

```
- title "Dispositivo '#{@hardware.name}'"
%table.table.table-bordered
  %tbody
    %tr
      %td
        %strong= 'Nombre'
      %td= @hardware.name
    %tr
      %td
        %strong= 'Numero'
```

```
      %td= @hardware.number
    %tr
      %td
        %strong= 'Usuario'
      %td= @hardware.user.user_name
    %tr
      %td
        %strong= 'Tipo de Arduino'
      %td= @hardware.Arduino_type.name
    %tr
      %td
        %strong= 'Fecha de publicación'
      %td= @hardware.published_at.strftime('%d/%m/%Y %H:%M') rescue nil
    %tr
      %td
        %strong= 'Activo'
      %td= @hardware.is_active ? 'SI' : 'NO'
    %tr
      %td
        %strong= 'Latitud'
      %td= @hardware.geo_latitud
    %tr
      %td
        %strong= 'Longitud'
      %td= @hardware.geo_longitud
= render :partial => 'links', :locals => {:mylocal => @hardware}
```

El código HTML generado es el siguiente

Código 44. Código HTML generado de la vista por la acción show del controlador hardwares

http://localhost:3000/admin/harwares/path

```
<table class='table table-bordered'>
  <tbody>
    <tr>
      <td>
        <strong>Nombre</strong>
      </td>
      <td>Arduino02</td>
    </tr>
    <tr>
```

```
      <td>
        <strong>Numero</strong>
      </td>
      <td>FMGNPYMMWCL</td>
    </tr>
    <tr>
    <td>
      <strong>Usuario</strong>
    </td>
    <td>usuario01</td>
    </tr>
    <tr>
      <td>
        <strong>Tipo de Arduino</strong>
      </td>
      <td>Mega</td>
    </tr>
    <tr>
    <td>
      <strong>Fecha de publicación</strong>
    </td>
    <td>08/10/2012 21:42</td>
    </tr>
    <tr>
      <td>
        <strong>Activo</strong>
      </td>
      <td>SI</td>
    </tr>
    <tr>
      <td>
        <strong>Latitud</strong>
      </td>
      <td>0.0</td>
    </tr>
    <tr>
      <td>
        <strong>Longitud</strong>
      </td>
      <td>0.0</td>
    </tr>
  </tbody>
</table>
```

En el navegador después de establecer los estilos necesarios se ve como se muestra en la siguiente imagen.

Figura 38. Resumen del nuevo sistema creado

e) La ruta edit http://localhost:3000/admin/hardwares/hardware-name/edit es similar a la new, solo que el objeto que se recorre ya tiene datos, a demás la acción del formulario va por método PUT, hacia la acción UPDATE y no a la CREATE, como en el NEW.

En la acción edit del controlador no hay código, pero al igual que en la acción show, se tiene que hacer la consulta a la base de datos con la función load. Luego se renderiza la vista, que se muestra en el siguiente código.

Código 45. Vista de la acción edit del controlador hardwares

app/views/admin/hardwares/edit.html.haml

```
- title "Editando dispositivo '#{@hardware.name}'"
= render "form"
```

Como se puede notar también se renderiza el formulario pero en este caso el objeto que se recorre ya tiene datos por tanto, el formulario aparecerá pre-llenado con los datos que el registro tiene almacenado. Esto se muestra en la siguiente figura. Además, de los valores por defecto también cambia la acción y el método del formulario, ahora es dirigido a ala acción update por método "put".

Figura 39. Edición de un dispositivo

f) El submit de formulario anterior lleva a la acción update, a esta acción se accede por método put, el código en esta acción es el siguiente.

Código 46. Acción create en el controlador de hardwares

app/controllers/admin/hardwares_controller.rb

```
def update
  if @hardware.update_attributes(params[:hardware])
    redirect_to [:admin, @hardware], :notice => t(:successfull)
  else
    render :action => 'edit'
  end
end
```

Como se puede notar, similar a la acción create, el registro se trata de actualizar con el método "update_attributes" al cual se le pasan los parámetros que se han ingresado en el formulario, y de nuevo se corren las validaciones que se han hecho en el modelo, si el registro se actualizo correctamente, se envía al detalle de este, caso contrario, se vuelve a renderizar el formulario.

g) La ruta destroy aunque tiene la misma url que la show http://localhost:3000/admin/hardwares/hardware-name, se diferencian en que el método usado para esta acción es el método DELETE, y comúnmente lo que hace la acción es borrar un registro, pero en muchas ocasiones se prefiere no borrar el registro sino solo colocar una valor que indique que el registro fue borrado y ya no presentarlo, aunque en realidad nunca se borre, para estos se destino un campo llamado "deleted_at", la acción destroy tiene el siguiente código. Antes de borrarlo se le pide al usuario que confirme su petición ya que pudo ver dado clic al enlace por error.

Figura 40. Confirmación antes de borrar

Código 47. Acción destroy del controlador hardwares

app/controllers/admin/hardwares_controller.rb

```
def destroy
  @hardware.update_attributes(:deleted_at => Time.now)
  redirect_to admin_hardwares_url, :notice => t(:successfull)
end
```

Recordemos que en la acción se ejecuta el método load, que busca el hardware, luego se actualiza el registro por último, se redirige a la index de hardwares, por lo que esta acción no tiene un vista.

En las acciones que no tienen una vista comúnmente se dirigen a otra acción o ruta pero se coloca una variable ":notice => 'texto'", que lleva un mensaje indicando que el registro fue actualizado.

NOTA: Todo este trabajo aquí resumido para la tabla hardwares, se realiza de igual forma para cada tabla de la base de datos.

Rutas visibles para un usuario

Se crearon rutas donde solo se utilizan la index y show, el cliente en este caso no tiene acceso a borrar, editar ni crear registros.

Se crearon las rutas de tal manera que el usuario siga un numero de pasos lógicos para llegar a lo que está buscando, selecciona su Arduino, en este se muestran los sensores que tiene conectado, al seleccionar el sensor se muestra una gráfica. Veamos el proceso paso por paso.

Se supone que el usuario esta loggeado, y en la ruta http://localhost:3000/, desde ahí puede acceder a enlaces que se le proporcionan.

Puede ver todos sus hardwares, en la siguiente imagen se muestra usuario que tiene un hardware al cual ha llamado "Arduino01" y a el tiene conectados cuatro sensores, estos enlaces están disponibles desde el menú.

Figura 41. Enlace a los Arduinos y sensores del usuario.

Aquí el usuario puede elegir, dar clic a Arduino01, en ese momento se entrara en la ruta http://localhost:3000/hardwares/path.

La ruta anterior es la ruta show de hardwares, como puede notarse la diferencia a la ruta show de hardwares del administrador es que no lleva la palabra admin.

Aquí el usuario ve una tabla de descripción general de su Arduino, además ve el listado de sensores conectados a su Arduino y aun mas, tiene acceso a un mapa que le muestra la ubicación de su Arduino. Lo que se necesita entonces es hacer una consulta para obtener el hardware y los sensores conectados a el, para dibujar su descripción entonces la consulta en el controlador es como se muestra en el siguiente código.

Código 48. Acción show de hardwares

app/controllers/hardwares_controller.rb

```
class HardwaresController < ApplicationController
  before_filter :authenticate_user!
  def show
    @hardware = current_user.hardwares.where(:is_active => true, :path =>
params[:id]).first
    @systems = @hardware.systems.where(:is_active => true)
  end
  def index
    @hardwares = current_user.hardwares.where(:is_active => true)
  end
end
```

Como se puede notar en la acción show se obtienen tanto el hardware como los sensores conectados a el, además se ejecuta antes de esto un el método "authenticate_user!" de devise, que valida que sea un usuario registrado el que consulta dicha información., caso contrario se redirecciona a loggearse.
Si el usuario esta loggeado, se renderiza una vista con el siguiente código.

Código 49. Vista de la acción show de hardwares.

app/views/hardwares/show.html.haml

```
- content_for :footer do
  = javascript_include_tag
'http://maps.google.com/maps?file=api&v=2&key=AIzaSyA5BemLlgSaAsajfq47oiXTm1Ad6VhZVvc',
'jquery.gmap'
- title "Arduino '#{@hardware.name}'"
%table.table.table-bordered
  %tbody
    %tr
      %td
        %strong= 'Nombre'
      %td= @hardware.name
    %tr
      %td
        %strong= 'Numero'
      %td= @hardware.number
    %tr
      %td
```

```
        %strong= 'Tipo de Arduino'
      %td= @hardware.Arduino_type.name
    %tr
      %td
        %strong= 'Fecha de publicación'
      %td= @hardware.published_at.strftime('%d/%m/%Y %H:%M') rescue nil
    %tr
      %td
        %strong= 'Latitud / Longitud'
      %td
        = @hardware.geo_latitud
        = '/'
        = @hardware.geo_longitud
    %tr
      %td= ' '
      %td
        = link_to 'Ver mapa', '#modal', :class => 'btn btn-info', :Data => {:toggle =>
'modal'}
= link_to '<< Ver todos mis dispositivos', hardwares_path, :class => 'btn btn-info'
%br/
%h3= 'Sensores'
%br/
%br/
%table.table.table-bordered.table-condensed
  %thead
    %tr
      %th= 'Nombre'
      %th= 'Pin'
      %th
        = 'Destacado'
        = ' / '
        = 'Publico'
      %th= 'Tipo'
      %th= ' '
  %tbody
    - for system in @systems
      %tr
        %td= system.name
        %td= system.pin_number
        %td
          = system.inhome ? 'SI' : 'NO'
          = ' / '
          = system.is_public ? 'SI' : 'NO'
```

```
        %td= system.kind.name
        %td= link_to 'Ver grafica', hardware_system_path(@hardware, system), :class =>
'btn btn-primary'
#modal.modal.hide(style='margin-top:50px;top:0;')
  .modal-header
    %h2= "Mapa de localizacion de #{@hardware.name}"
  .modal-body
    #map.map
  .modal-footer
    %button.btn.btn-danger(Data-dismiss="modal" aria-hidden="true")= "cerrar"
:javascript
  var cont = 0;
  $(document).ready(function(){
    $('#modal').on('shown', function () {
      if(cont == 0){
        cont += 1;
        $("#map").gMap({
          markers: [{
            latitude: #{@hardware.geo_latitud},
            longitude: #{@hardware.geo_longitud},
            html: "#{@hardware.name}",
            popup: true
          }],
          //maptype: G_PHYSICAL_MAP,
          zoom: 14
        });
      }
    })
  });
```

En resumen este código hace lo siguiente, primero agrega unas librerías de javascript que nos permiten trabajar con GoogleMap, Luego se crean dos tablas la primera con la descripción del hardware y la segunda con los sensores conectados a el, por ultimo se crea un "MODAL" que es una especie de ventana que se mostrara al hacer clic en el botón ver mapa y por ultimo creamos el mapa.
El usuario podrá ver una pantalla como la que se muestra en la siguiente imagen.
Vista y descripción de un hardware del usuario.

Figura 42. Vista mostrada al usuario, en la acción show de hardwares

Desde aquí básicamente se pueden ver tres enlaces importantes, Ver un mapa en el cual se indica la posición del Arduino, regresar y ver el listado de todos los Arduinos, y ver una grafica correspondiente a un sensor conectado a ese Arduino. Al hacer clic sobre ver mapa se cargara el modal con el mapa, como se muestra en la siguiente imagen.

Figura 43. Mapa que muestra la ubicación del Arduino seleccionado

Al dar clic en ver grafica de algún sensor nos dirigimos a una página donde tendremos acceso al detalle del sensor incluido una grafica de los valores registrados por este mismo.
En el controlador de sensores tendremos que hacer las consultas a la base de datos como se muestra en el siguiente código.

Código 50. Controlador para los sensores (system)

app/controllers/systems_controller.rb

```
class SystemsController < ApplicationController
  before_filter :authenticate_user!, :except => [:Data]
  def show
    @hardware = current_user.hardwares.where(:path =>
params[:hardware_id]).first
    @system = @hardware.systems.where(:path => params[:id]).first if @hardware
    render(:file => "#{Rails.root}/public/404.html", :layout => false, :status
=> 404) and return unless @system
  end

  def data
    @system = System.find_by_id(params[:id].to_i)
    render(:file => "#{Rails.root}/public/404.html", :layout => false, :status
=> 404) and return unless @system
    respond_to do |format|
      format.csv {
        if params[:start_at] and params[:stop_at]
          send_Data @system.getDataCsv(
            :start_at => params[:start_at].to_datetime,
            :stop_at => params[:stop_at].to_datetime
          )
        else
          send_Data @system.getDataCsv();
        end
      }
      format.js {
        if params[:start_at] and params[:stop_at]
          @Data = @system.getDataJs(
            :start_at => params[:start_at].to_datetime,
            :stop_at => params[:stop_at].to_datetime
          )
        else
          @Data = @system.getDataJs();
```

```
        end
      }
      format.xls {
        if params[:start_at] and params[:stop_at]
          @lectures = @system.getData(
            :start_at => params[:start_at].to_datetime,
            :stop_at => params[:stop_at].to_datetime
          )
        else
          @lectures = @system.getData();
        end
      }
    end
  end
end
```

Básicamente hay dos acciónes, mostrar la pagina "acción show" y obtener las lecturas "acción Data"
La acción show muestra una página con pocos detalles del sensor y un espacio donde se dibujara una grafica de los valores que este sensor ha registrado.
La vista tiene el siguiente código.

Código 51. Vista para la acción show de systems

app/views/systems/show.html.haml

```
- content_for :head do
  =   javascript_include_tag   'jquery.flot',   'grafica',   'jquery-ui',   'jquery-ui-
timepicker-addon'
  = stylesheet_link_tag 'jquery-ui-timepicker-addon', 'jquery-ui'
- title "Sensor '#{@system.name}'"
%p
  %strong= ">> Midiendo #{@system.kind.name} en el pin #{@system.pin_number}"
%p
  %strong= 'Descripción: '
  = @system.description
.content_graph
  .graph{:id => "graph_"+@system.id.to_s}
%table.table
  %tr
    %td= 'Desde'
    %td= 'Hasta'
```

```
  %tr
    - format = '%Y/%m/%d %H:%M:%S'
    %td= text_field_tag :start_at, (1.day.ago).strftime(format), :readonly => true,
:class => 'time'
    %td= text_field_tag :stop_at, (1.day.since).strftime(format), :readonly => true,
:class => 'time'
  %tr
    %td= 'Obteber los datos de la grafica mostrada en:'
    %td
      = link_to raw(image_tag('opx.png')+' CSV'), '#', :class => 'btn btn-info', :id =>
'csv'
      = link_to raw(image_tag('xls.png')+' XLS'), '#', :class => 'btn btn-info', :id =>
'xls'
  - if (@system.is_public)
    %tr
      %td= "Enlace público"
      %td= link_to public_url(@system), public_url(@system)
```

Aquí se han incluido unas librerías de javascript para generar graficas llamada "FLOT", luego se muestra una pequeña descripción del sensor, también se muestran unas casillas de texto donde el usuario puede elegir el rango de fechas en el cual desea obtener datos por defecto se muestran solo los datos desde hace un día, por ultimo se muestran dos enlaces que permiten descargar los datos de la grafica que se esta mostrando.
Aunque esto se resume en unas pocas palabras, no fue así el tiempo de trabajo dedicado a esto, con la librería Jquery-ui y el siguiente código se obtuvo un bonito calendario, donde el usuario puede elegir fechas con precisión de minutos para mostrar datos.

Código 52. Usando datepicker de jquery-ui.

app/assets/javascript/grafica.js

```
  var a = {
    separator: ' ',
    timeText: 'Tiempo',
    hourText: 'Horas: ',
    minuteText: 'Minutos: ',
    currentText: 'Ahora',
    closeText: 'Aceptar',
    hourGrid: '3',
```

```
    minuteGrid: '10',
    dateFormat: '20y/m/d'
  }
  $('.time').datetimepicker(a);
```

Básicamente se crea una variable donde se guardan las configuraciones que se desea aplicar y por ultimo a la casilla de texto que previamente se le asigno la clase "time", se le aplica la función datetimepicker.

Ahora uno de los pasos mas importantes es cargar la grafica, y que se siga actualizando con el tiempo, el código que hace esto es el siguiente.

Código 53. JavaScript para generar graficas auto recargables

app/assets/javascript/grafica.js

```
$(document).ready(function(){
  $('.graph').each(function(){
    getData($(this).attr('id'));
  });
});

function getData(div_id){
  var div = $('#'+div_id);
  var start_at = $('#start_at').val();
  var stop_at = $('#stop_at').val();
  var system_id = div_id.split('_')[1]
  $('#csv').attr('href',
"/Data.csv?id="+system_id+'&start_at='+start_at+'&stop_at='+stop_at);

  $('#xls').attr('href',
"/Data.xls?id="+system_id+'&start_at='+start_at+'&stop_at='+stop_at);
  $.getScript(
    "/Data?id="+system_id+'&start_at='+start_at+'&stop_at='+stop_at,
    function(){
      setTimeout("getData('"+div_id+"')", 5000);
    }
  );
}

function setPlot(div, d){
  plot = $.plot(div,[d],{
```

```
    xaxis: {
      mode: "time",
      timeformat: "%d/%m/%y - %H:%M:%S",
    }
  });
}
```

Explicación: No se trata de aprender javascript en este momento ni mucho menos jquery, pero la función de este código es lo que logra que el usuario pueda visualizar los cambios en los sensores casi instantáneamente.

- Primero "$(document).ready", jquery permite de esta forma ejecutar nuestro código hasta que el DOM de la pagina ha cargado por completo (http://es.wikipedia.org/wiki/Document_Object_Model).
- Luego "$('.graph').each" selecciona los elementos con la clase "graph", y los recorre ejecutando la función que se le pase como parámetro
- La función pasada como parámetro ejecuta "getData($(this).attr('id'));", obteniendo el atributo id del elemento que este recorriendo en ese momento, aunque todos los elementos tienen la misma clase, el atributo "id" es único para cada uno de ellos por ejemplo:
  `<div class='graph' id='graph_3' style='height:200px'></div>`
- La función "getData" envía el id del elemento "div" hacia la función getData, donde se obtiene el elemento ya no por clase si no por id "var div = $('#'+div_id);" se obtienen los valores de el rango de fechas a mostrar y por ultimo el id del sensor (no confundir con el id del elemento div)
  - id del elemento: graph_3
  - id del sensor: 3 (se hace un Split para obtenerlo)
- Con esto se arma una url
  ("/data?id="+system_id+'&start_at='+start_at+'&stop_at='+stop_at")
  y se hace la petición get que devuelve un script que contiene los datos, la url para la petición con ajax es similar a la siguiente:
  http://localhost:3000/data?id=4&start_at=2012/10/1%2000:00&stop_at=2012/10/18%2000:32:00&_=1350455599367
  Donde se nota que se ha incluido el rango de fechas y un id del sensor, con el cual se obtendrán los datos para la grafica.
- Esta url devuelve un script como el siguiente
  Script devuelto por ajax

Código 54. Javascript devuelto con ajax

http://localhost:3000/Data?id=4&start_at=2012/10/1%2000:00&stop_at=2012/10/18%2000:32:00&_=1350455599367

```
var d = [[1349655626000, 91.796875], [1349677226000, 237.3046875], [1349684426000, 388.18359375], [1349698826000, 253.90625],
...CONTINUA…
[1350354028000, 493.65234375], [1350364829000, 10.7421875]];
var div = $('#graph_4');
setPlot(div, d);
```

- Al terminar de ejecutarse el script se vuelve llamar a la función "getData" con un retardo de 5 segundos, con esto se logra que se este actualizando la grafica automáticamente, pero no toda la pagina, minimizando el tiempo de espera
- Por último es importante notar que los valores del rango de fechas son también utilizados para cambiar constantemente la url ala que apuntan los enlaces de descargar datos en formatos "CSV" y "XLS".

En el controlador se puede ver la obtención de datos ya sea para formato "JS", "CSV" o "XLS" (ver acción Data en systems_controller.rb)
Como pequeño resumen, en esta ultima pantalla se tiene tanto la carga de la pagina, acción show, si se puso atención habremos notado que aunque es la acción show solo se prepara la pagina, porque en el momento que cargué no tendrá grafica alguna, de hecho no se mandan datos para ser graficados, pero luego de haber cargado se hace una petición ajax que devuelve los datos y grafica por fin los datos. A continuación la vista de la petición ajax, en el código anterior se ve el resultado, pero ahora veremos como se genero.

Código 55. Vista de la acción "Data" en system

app/views/systems/Data.js.erb

```
var d = <%= @Data %>;
var div = $('#graph_1 %>');
setPlot(div, d);
```

Aunque parece muy sencillo detrás de esto hay mucho escondido, en el código de "app/assets/javascript/grafica.js" que explicamos en las líneas anteriores, se incluyo una función llamada "setPlot" pero parecería nunca la se utiliza, en

realidad la utiliza este código, se dejo en el anterior, para que cuando se ejecute este código no se tenga que estar enviando la función por cada petición, esto con el afán de disminuir el trafico y el tiempo entre peticiones.

La explicación del código anterior es relativamente sencilla, los datos consultados en la base de datos que se obtuvieron en el controlador y se guardaron en la variable "@data" ahora son asignados a la variable "d" del código javascript, luego se selecciona el elemento donde se graficara, (un div con id "graph_#"), tanto los datos que se encentran en un arreglo como el elemento donde se graficara son enviados a la función "setPlot", esta los recibe e invoca a la función plot del plugin de jquery llamado "flot" "plot = $.plot(div,[d], {…}", donde se le pasa de nuevo, el elemento donde se graficara, los datos y unas configuraciones, que hemos dejado por defecto, por ejemplo el formato de la fecha.

Resumiendo, cada cinco segundos (aproximadamente) se ejecuta un script que lee los valores de fechas elegidas, con estas se actualiza el enlace al que apuntan las descargas de archivos "XLS" y "CSV" luego se hace una petición vía ajax, que devuelve un script, que actualiza la grafica.

A continuación se muestra la grafica que el cliente puede ver.

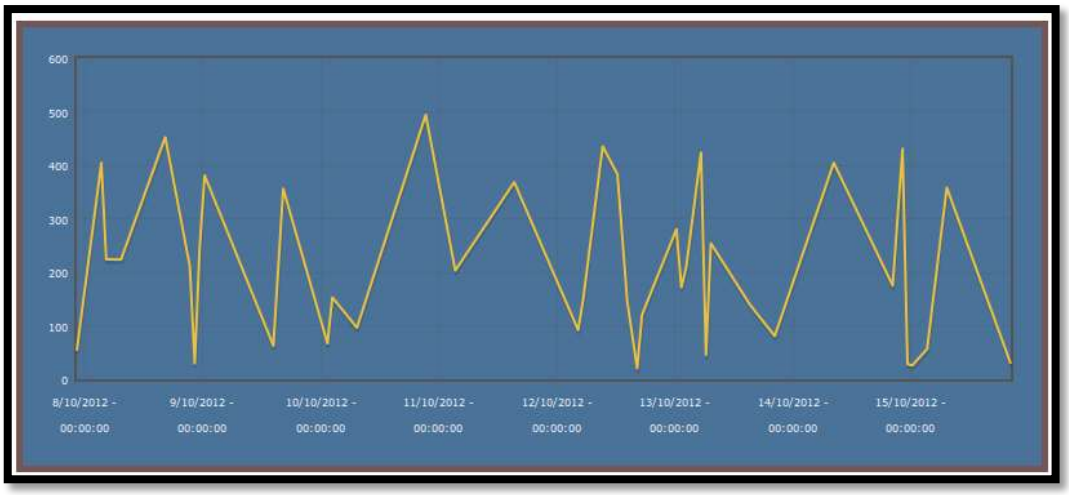

Figura 44. Grafica generada con "flot"

Al dar clic en uno de los campos de texto, aparece un calendario que debería de verse como en la siguiente imagen.

Figura 45. Calendario para seleccionar fechas.

Al dar clic en cualquiera de los enlaces para descarga de archivos se obtendrá un archivo ya sea en formato csv o xls. Al abrir el archivo con el procesador de hojas de calculo que se prefiera, se podrá entre otras cosas graficar los datos, obtener las tendencias, etc. En la siguiente imagen se muestra el archivo obtenido, y además una grafica generada con estos datos.

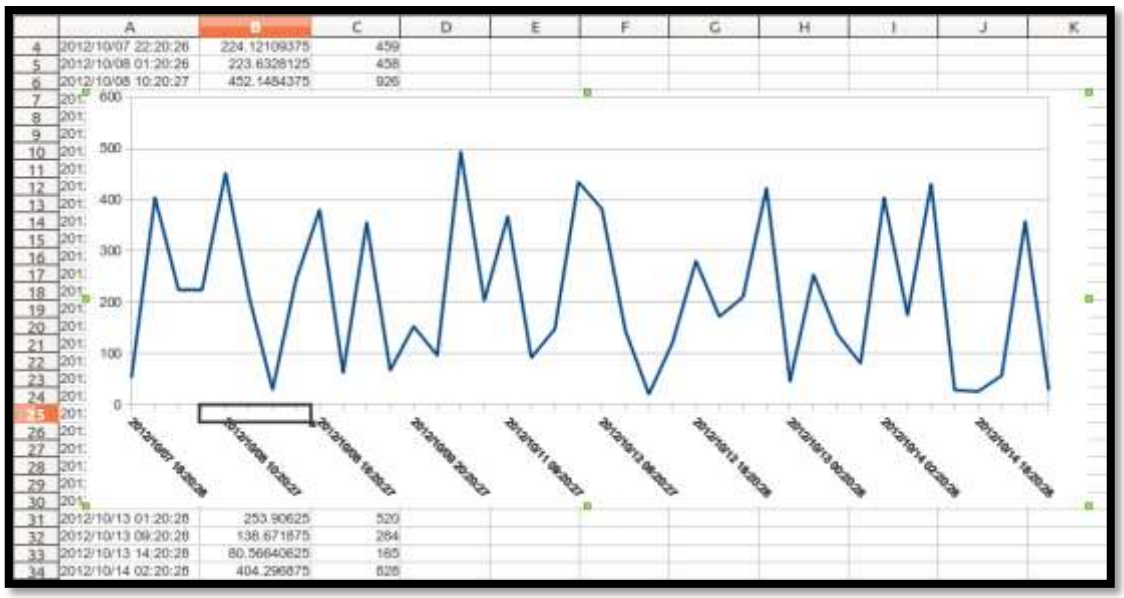

Figura 46. Datos descargados y grafica de los mismos.

En la siguiente imagen mostramos una comparación de la grafica vista desde el navegador, desde Libre Office y desde Excel, respectivamente.

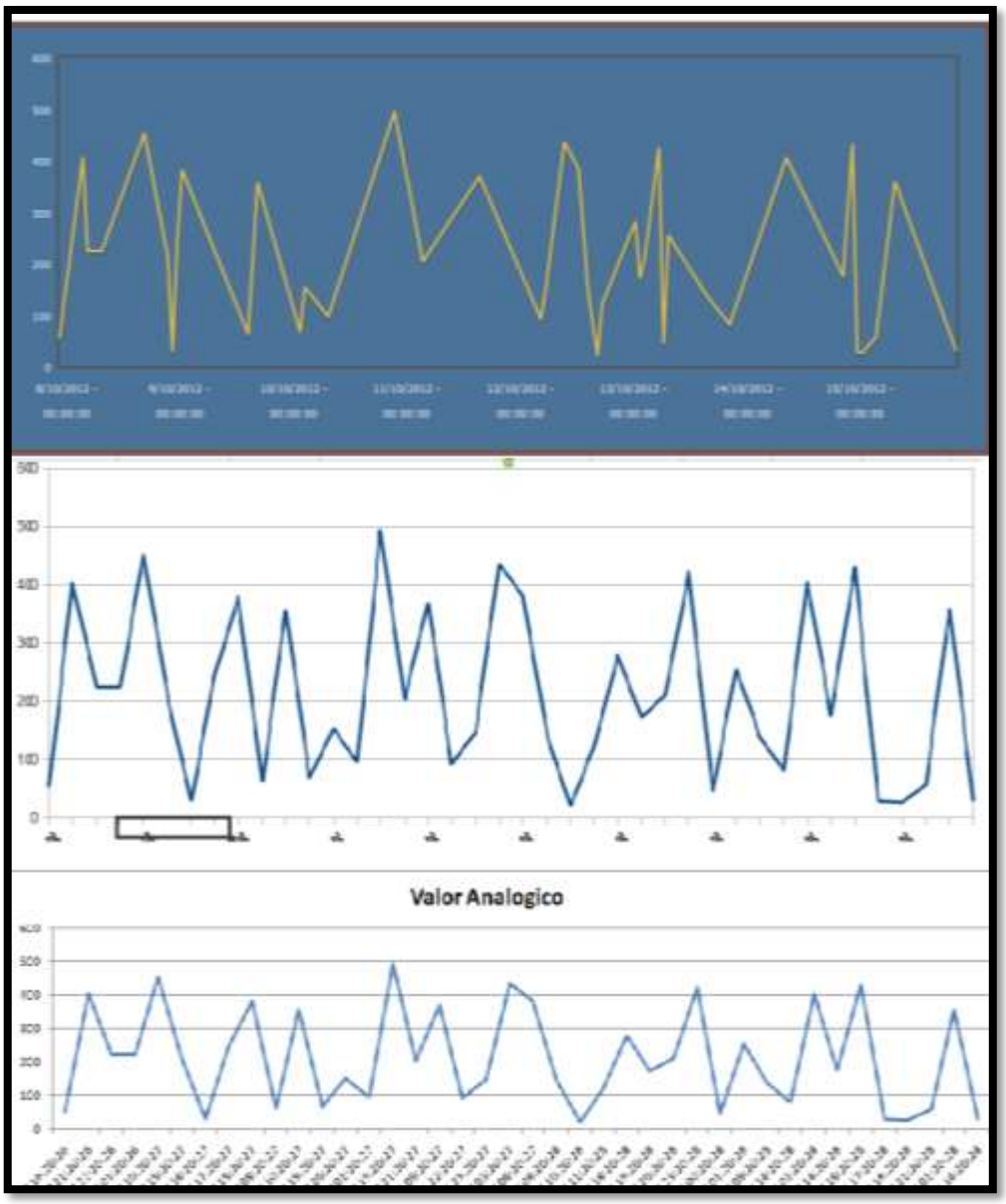

Figura 47. Graficas de los datos, desde el navegador, Libre Office y Excel respectivamente.

Un último dato que hay que notar es que se grafican en todos los div con la clase "$('.graph').each" ¿para qué recorrerla si solo tenemos una grafica en la pagina?, la razón es que en la página principal se podrá tener más de una. Esto se muestra en la siguiente imagen que muestra cuatro graficas en la página principal.

Figura 48. Página de principal mostrando cuatro graficas simultáneamente.

Se crearon mas rutas, más vistas, y desde luego mas código, pero con lo que hemos visto se explican casi todas las demás. La gema devise nos genero rutas para registro y sesión de usuarios, se trabajo también mucho en esto pero la idea básica es la misma.

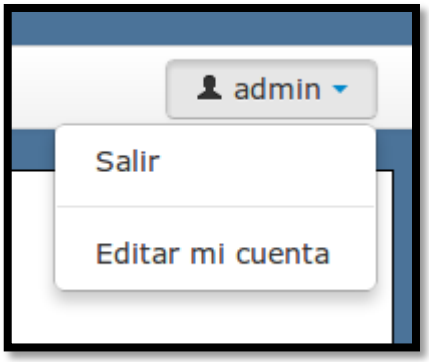

Figura 49. Usando las rutas de devise.

3.6. Gemas

Se ha estado hablando de las gemas que se han utilizado, pero quizá no quede muy claro ¿Qué es una gema?, básicamente una gema es lo que se conoce en otros lenguajes como librerías. Su propósito es añadir nuevas funcionalidades al proyecto (como devise para registro de usuarios) o facilitar el desarrollo de algún proceso (will_paginate). A continuación se listan las gemas usadas en este proyecto junto a una breve descripción. Puede encontrar la mayoría de gemas en http://gems.rubyforge.org/

- Rails: Esta es la más importante, nos crea la estructura del proyecto basado en la arquitectura MVC (Modelo Vista Controlador), además instala a su ves otras dependencias como ActiveRecord, que permite realizar consultas a la base de datos de una manera elegante y fácil de entender, al estilo de RUBY (Humano). Como todo framework se basa en un lenguaje, en este caso ruby, una de las ventajas de rails es que no se necesita ser experto en ruby para trabajar con rails.
  Algunos enlaces que podrían ayudar a entender esta gema son:
  - http://es.wikipedia.org/wiki/Ruby_on_Rails
  - http://guides.rubyonrails.org/getting_started.html

- pg: Cuando se va a usar una base de datos (casi siempre), hay que especificar qué base de datos se va a utilizar, "pg" en este caso nos permite trabajar con postgresql. De esta manera activerecord puede crear las conexiones, para hacer consultas, crear tablas, borrar tablas, etc. Cada base de datos es diferente y existe una gema para trabajar con ella.

- sass-rails: Simplifica el trabajo para escribir hojas de estilos, tiene muchas ventajas, pero principalmente la reducción de líneas a escribir.
  Su página oficial es:
  - http://sass-lang.com/
- coffee-rails: se instala por defecto en las últimas versiones de rails, pero no se ha usado en este proyecto, es similar a sass, permite escribir menos código pero esta vez para JavaScript.
  - http://coffeescript.org/

- jquery-rails: descarga e instala las librerías de jQuery en el proyecto, como se menciono antes, permite trabajar con javascript, de una manera más sencilla, además muchas librerías trabajan con jquery, en este proyecto se ha incluido, flot para graficar, pero esta librería trabaja también con jquery, otra es jquery-ui, que se ha usado para el calendario.
  Enlaces acerca de jquery
  - http://jquery.com/
  - http://jqueryui.com/
  - https://github.com/indirect/jquery-rails

- haml: Se ha mencionado mucho y a mi punto de vista quizá sea una de las gemas que mas colaboran en la rapidez de desarrollo, simplifica enormemente el trabajo, además obliga a ser más ordenado, lo cual ayuda a entender mejor los códigos escritos.
  - http://haml.info/

- will_paginate: es una gema que permite paginar resultado, aunque no se ha hablado de ella en este reporte brinda característica que hacen que paginar resultados sea muy fácil, imagine que se tiene un millón de usuarios registrados no sería practico mostrarlos a todos en una sola página, tendríamos que hacer una consulta a la base y obtener solo un pequeño conjunto de estos, es lo que permite will_paginate, con solo invocar al método "paginate".
  - https://github.com/mislav/will_paginate/wiki

- devise: nos permite autenticar usuarios, administrar cuentas, entre otras. A continuación un pequeño resumen de sus funcionalidades.
  - Base de datos autentificable: encripta y almacena una contraseña en la base de datos para validar la autenticidad de un usuario a la vez de iniciar sesión La autenticación se puede realizar tanto a través de las peticiones POST o autenticación básica HTTP.
  - Token autentificable: inicia sesión de un usuario basándose en un token de autenticación (también conocido como "token de acceso único"). El token se puede dar tanto a través de la cadena de consulta o la autenticación básica HTTP.

- Confirmable: envía mensajes de correo electrónico de confirmación con instrucciones y comprueba si una cuenta está ya confirmada al iniciar sesión
- Recuperable: restablece la contraseña del usuario y envía las instrucciones de restablecimiento.
- Registrable: se encarga de firmar los usuarios a través de un proceso de registro, que también les permite editar y destruir su cuenta.
- Rememberable: gestiona la generación y eliminar un símbolo para recordar al usuario de una cookie guardada.
- Trackable: guarda la IP del usuario.
- Timeoutable: expira sesiones que no tienen actividad en un período de tiempo especificado.
- Validable: proporciona validación de correo electrónico y contraseña. Es opcional y se puede personalizar, por lo que es capaz de definir sus propias validaciones.
- Con cierre: bloquea una cuenta después de un número determinado de no inicio de sesión en los intentos. Se puede desbloquear a través de correo electrónico o después de un período de tiempo especificado.

Enlaces:

- https://github.com/plataformatec/devise

- client_side_validations: Permite validar en el lado del cliente con javascript, lo que hemos validado en los modelos.
  - https://github.com/bcardarella/client_side_validations

- Thin: Es un servidor para aplicaciones rails. Rails viene por defecto con un servidor de pruebas llamado webrick, pero no es funcional a la hora de manejar un número considerable de peticiones al mismo tiempo, además no usa mucha ram.
  - http://code.macournoyer.com/thin/

3.7. Hojas de estilos

Aunque se pueden hacer páginas web sin tocar una hoja de estilos, es preferible usar estas, ya que nos permiten dar la apariencia que deseamos a nuestra página web, uno de los principales problemas a la hora de trabajar en el diseño de una página web, es que no se verá igual entre los diferentes navegadores, por ejemplo un párrafo, la mayoría de navegadores ponen un margen entre párrafo y párrafo,

para poder separarlos y que sea más legible por el usuario, el problema es que el margen dado varía entre los distintos navegadores, ejemplos mejores son, los enlaces, los botones, etc. Para esto se han creado normas que los navegadores deben de cumplir, por ejemplo si se especifica en una hoja de estilos que un párrafo tendrá 10px de margen esto debe de aplicarse en todos los navegadores, pero lastimosamente algunos no siguen al pie de la letra las normas, y por esta razón es bueno revisar el sitio web en diferentes navegadores. El navegador Internet Explorer es uno de los más utilizados ya que viene instalado por defecto en Windows, pero también es el que más problemas da, no soporta algunas reglas y otras las interpreta mal. Si se quiere aprender CSS la mejor herramienta es
http://www.w3schools.com/
Aquí se puede probar código además muestran la compatibilidad entre navegadores. En el camino se dará cuenta que IE es el que menos soporta las propiedades, y esto es basado en Internet Explorer 9, si muchas personas aun usan el IE7, que soporta aun menos, y una minoría de personas usan el IE6 o inferior.
Por ejemplo la propiedad box-align
http://www.w3schools.com/cssref/css3_pr_box-align.asp

O por ejemplo border-top-right-radius, solo soportada por el IE9 o mayor.
http://www.w3schools.com/cssref/css3_pr_border-top-right-radius.asp
"The border-top-right-radius property is supported in IE9+"
Es de mucho interés ver
http://www.w3schools.com/html/html5_canvas.asp
Ahí veremos que la etiqueta canvas es soportada solo por el IE9, Las graficas que generamos con FLOT usan la etiqueta canvas, y ya que el IE no la soporta, se tuvo que emular con otra librería de javascript llamada EXCANVAS.

3.8. Conectándose con arduino

Ya se ha hablado mucho del servidor, como crear, borrar y actualizar registros en la base, también ya se vio como Arduino puede leer los datos de los sensores conectados a el, pero ¿en qué momento se logra la conexión con el servidor?, ¿Cómo interpreta estos datos el servidor?, ¿Cómo logra Arduino enviar los datos?, ¿Qué protocolo se está usando?, ¿Se han tratado aspectos de seguridad? Se vera en esta sección como estas nuevas tecnologías, tanto Arduino como rails, al basarse ambos en estándares adoptados mundialmente se pueden conectar sin ningún problema.

3.8.1. Haciendo una petición desde arduino

Esto suena muy lógico al pensar desde el servidor en rails, siendo este un servidor recibe muchas peticiones, de hecho cada vez que se carga una página web que es servida por la aplicación, se hace más de una petición, por ejemplo al cargar la index, carga el código html generado, imágenes, hojas de estilos y códigos javascript, además se ha programado que se hagan peticiones constantemente para recargar la grafica, y el servidor envía todos los datos que se le soliciten. Pero lo anterior carece un poco de sentido cuando se piensa del lado del hardware con Arduino, Arduino lee un sensor cualquiera, ¿y ahora hace una petición al servidor?, no será más bien envía los datos al servidor, pues muy bien si se pensó de esta manera, pero existe un detalle, cuando por ejemplo en una página web cualquiera se hace clic en un enlace, por ejemplo un enlace que se sabe que llevara a ver un video, en ese momento el servidor al analizar la petición responde con el video, sirviendo de esta forma el video, pero ¿Lo notaron?, ¿Cómo sabe el servidor cual video desean ver?, la respuesta es a través del enlace, cada vez que se hace una petición al servidor, también el navegador actúa como servidor, enviando datos, en este caso un identificador único para el video. En un correo podríamos enviar un video, un documento, etc. En ese preciso momento se ha pasado a servir un archivo y enviarlo al servidor, pero puede ser que no se envie un archivo, el simple hecho de hacer clic en un enlace, desencadena un suceso en el cual se sirven muchos datos, observe el siguiente enlace
http://www.youtube.com/watch?v=3eh6yRdJrGM
¿Qué datos se envía?

- Se puede observar que el servidor es youtube.com, el primer parámetro importante que se envía y en el que todos estaremos de acuerdo es "v=3eh6yRdJrGM", que es un identificador único que diferencia este video de todos los demás, de esta manera el de servidor youtube sabe que video enviarnos.
- Pero se envían más datos sin darnos cuenta, por ejemplo el host, el navegador que estamos usando, el sistema operativo en el que estamos trabajando, algunas posibles cookies, etc.

Aprovechando esto, se logra que Arduino envíe datos a nuestro servidor, por ejemplo con la siguiente url, se envían:
http://localhost:300/public/EVOUPXJFJYM/set_Data?time=2012/10/25%2023:59:59&Data[]=0-17.17&Data[]=1-25.3

- Hora de captación de datos 23/10/2012 23:59:59
- Código del arduino que hace la petición
- El pin 0 registro una lectura de 17.17
- El pin 1 registro una lectura de 25.3

Y así se pueden anidar más datos.
De tal modo que como se decía al principio el hardware Arduino solo tiene que hacer una petición de este tipo, es el servidor quien deberá entenderla.
Ahora bien el trabajo del hardware entre otras cosas es generar esta URL, de forma que sea dinámica y se carguen los valores de todos los sensores activos en ella, luego de armar esta URL se procede a usar la librería Ethernet para hacer una petición GET.

Código 56. El código que genera la URL es el siguiente

Data Logger.cpp

```
String getData(){
  String url = "/public/"+Arduino_code+"/set_Data?time=";
  url += getTime();
  for (int pin = 0; pin < 13; pin++) {
    if (typeSensor[pin] != 0)  {
      url += "&Data[]=";
      url += getValueSensor(pin);
    }
  }
  return url;
}
```

Donde se puede ver cómo se va armando la URL concatenando cada uno de los sensores con su número de pin y valor, ahí se puede observar también funciones como getTime encargada de obtener la hora, esta se agrega una sola vez, logrando de esta forma mejorar el tiempo de conexión, también se puede ver otra función importante getValueSensor en esta se obtiene el valor de el sensor con el numero de pin indicado como parámetro de esta función, esto logra hacer mas legible el código además estas funciones se pueden usar en otras partes del código como cuando el hardware actúa como servidor enviando el valor actual de los sensores, en esta función se valida de qué tipo de sensor se esta hablando, para ejecutar la respectiva consulta al sensor.

Luego de tener la URL guardada en una variable se procede a hacer la petición al servidor, el código encargado de esto es bastante similar al mostrado en los ejemplos de Arduino.

Código 57. Función para peticiones desde arduino

Data_logger.cpp

```
void httpRequest(String str) {
  String url = "GET "+str+" HTTP/1.0";
  client.stop();
  if (client.connect(serverName, 80)) {
    info("SEVER: "+url);
    client.println(url);
    client.println("Host: Data Logger-Arduino.herokuapp.com");
    client.println();
    lastConnectionTime = millis();
  } else {
    info("Conexion Fallida");
    client.stop();
    is_conected = false;
    printToSd(str);
  }
}
```

Aunque el código es bastante similar al de los ejemplos de Arduino se pueden notar diferencias importantes.

- Es una función de modo que se puede usar repetitivamente, lo cual es muy conveniente ya que es necesario enviar muchas peticiones al servidor,
- Se incluye el HOST, se ha publicado el servidor en heroku, en este hay muchas otras aplicaciones, esto es algo muy común ahora en día, pero para diferenciar las peticiones no se puede hacer solo por IP, se necesita el HOST. Esto es muy común por ejemplo si trata de entrar a http://173.194.37.14/ (google.com) vera la pagina de google, ya que ellos tienen su servidor dedicado, pero si por el contrario el servidor aloja mas de un sitio web se necesitara mas que la IP, por ejemplo trate entrar a http://78.46.93.44/ vera un mensaje genérico, en esta ip se puede acceder a diferentes sitios, por ejemplo http://shop.taguzplata.com/

3.8.2. Almacenamiento de datos

Ahora el hardware ya es capaz de hacer peticiones al servidor, pero el servidor debe atender estas peticiones, analizarlas y atenderlas. Como se menciono la petición tiene el siguiente formato:
http://localhost:300/public/EVOUPXJFJYM/set_data?time=2012/10/25%2023:59:59&Data[]=0-17.17&Data[]=1-25.3

Vale la pena mencionar, que primero se generaron las rutas que atenderían las peticiones del hardware Arduino, luego con el formato especifico se programo Arduino para que hiciera las consultas tal y como queríamos.

Código 58. La acción encargada de recibir estas peticiones es "set_data"

app/controllers/public_controler.rb

```
  def set_data
    hardware = Hardware.find_by_number(params[:id])
    time = params[:time].to_datetime rescue nil
    ok = (!params[:data].blank? && browser_ok?)
    if ok
      params[:data].each do |datum|
        datum = datum.split('-')
        pin = datum.first.to_i
        value = datum.last.to_i
        system = hardware.systems.where(:pin_number => pin).first
        if pin and value and system
          lecture = Lecture.new(:system_id => system.id, :published_at => time,
:digital_value => value)
        end
      end
    end
    if ok and is_save
      render :text => "Datos guardados con éxito"
    else
    end
  end
```

Los parámetros que le llegan son los que ya se vieron con anterioridad, uno de los más importantes es el identificador del hardware que es único, en el ejemplo que

se esta viendo el ID era “EVOUPXJFJYM”, este es único, y como se puede notar es a través del que se obtiene el hardware que está enviando los datos.

```
hardware = Hardware.find_by_number(params[:id])
```

Luego se convierte la fecha a un formato especifico, esto tiene varias razones, la más importante es validar que no vaya nulo, ya que si no se envía la fecha de nada servirán los datos, no nos conviene ponerle la fecha en que llegaron los datos al servidor, porque puede ser que sean datos que estaban almacenados en la memoria desde hace mucho tiempo.
Los parámetros que llegan en la petición se obtienen en rails a través de un hash, en este caso son los siguientes.

Código 59. Los parámetros que llegan en la petición normal

Params

```
params: {
  "time"=>"2012/10/21 20:45:59",
  "Data"=>[
    "1-17.17",
    "0-25.3"
  ],
  "action"=>"set_Data",
  "controller"=>"public",
  "id"=>"xxxxxxx"
}
```

Como se puede observar el parámetro Data es un arreglo, por tanto se recorre y se divide cada uno de sus contenidos en pin y valor, de esta forma con el hardware ya encontrado y el pin conocido se puede saber el sensor exacto que envió este valor, por último se guarda una nueva lectura para cada uno de estos datos.
Cabe mencionar que existen validaciones que evitan que datos vacios, o ajenos a la aplicación sean guardados, la validación más importante es:

```
ok = (!params[:data].blank? && browser_ok?)
```

Que devuelve un booleano indicando si todo se cumple, si por ejemplo no se procesaran los datos si la fecha no está presente, si no van datos, si no está registrado el hardware que los está mandando.

Por último se agrego un punto más de seguridad, la url es pública, de modo que cualquiera podría escribir esta dirección en su navegador y empezar a inyectar datos, aunque necesitaría el código de un Arduino válido.

Para evitar esto se valida que no se estén haciendo las peticiones desde un navegador, sino desde un Arduino, esta función se llama "browser_ok?", básicamente se revisa el encabezado "HTTP_USER_AGENT", este es enviado por todos los navegadores sin que el usuario se de cuenta, de esta forma los sitios pueden saber que navegador se esta usando, por ejemplo para cambiar el estilo de su web por problemas de compatibilidad con IE, o desde un dispositivo móvil.
Se ha sacado provecho a esto y con esto se valida que sea un Arduino el que los envía. Si nos damos cuenta en la función que realiza la petición desde Arduino se ha agregado

```
client.println("User-Agent: xxxxx");
```

De modo que en el controlador que recibe la petición validamos que el encabezado "HTTP_USER_AGENT" contenga la palabra Arduino. El código es el siguiente.

Código 60. Método utilizado para validar si es un arduino el que se conecta

app/controllers/public_controler.rb

```
ARDUINO_BROWSER = "yyyyyy"
def browser_ok?
  agent = request.headers["HTTP_USER_AGENT"].downcase
  if agent.match(ARDUINO_BROWSER)
    return true
  else
    return false
  end
end
```

Por ejemplo firefox envía como USER_AGENT
mozilla/5.0 (x11; ubuntu; linux i686; rv:16.0) gecko/20100101 firefox/16.0
Por lo tanto no podrá un usuario usando firefox guardar datos a su gusto.
Esto fue probado también en Internet Explorer, Google chrome y Opera.

3.9. HEROKU llevando el proyecto a producción

Ahora que el proyecto esta finalizado y se a probado su funcionalidad, habrá que probar si funciona correctamente en producción, para esto necesitamos un "HOSTING", podemos encontrar varias soluciones a esto en internet,

- Un sitio llamado NO-IP (www.no-ip.com) ofrece de forma gratuita subdominio por ejemplo (www.example.no-ip.com) y asignárselo a la ip de una maquina que deberá permanecer encendida si queremos que alguien pueda navegar en el sitio, además de las limitaciones de nuestro hardware y nuestra conexión a internet, es una fácil solución pero con muchos inconvenientes.
- Quizá la mejor sea comprar un servidor dedicado, un nombre de dominio y montar el ambiente de trabajo. Esto tiene todas las ventajas a favor, ya que se puede manipular el servidor como se desee, darle la capacidad que se quiera, etc. Pero a un precio muy alto.

Por fortuna para nosotros existe HEROKU (http://www.heroku.com/), un servidor donde se pueden alojar sitios de Ruby on Rails gratuitamente, tiene limitaciones como máximo tamaño en MB de sitio, pero para este proyecto es perfecto, El servidor estará disponible siempre aunque nuestra maquina este apagada y con la potencia de un servidor de producción. Ahora podrán ver el sitio desde cualquier parte del mundo.

La siguiente secuencia de pasos muestra lo fácil que es comenzar con heroku.

1.0. Registrarse. Habrá que registrarse en:
https://api.heroku.com/signup/devcenter
Un registro común con solo brindar nuestro correo

2.0. Instalar Heroku Toolbelt
https://toolbelt.heroku.com/
En Linux basta con copiar en la consola
wget -qO- https://toolbelt.heroku.com/install-ubuntu.sh | sh

3.0. Ingresamos desde nuestra maquina.

```
$ heroku login
Enter your Heroku credentials.
Email: adam@example.com
Password:
Could not find an existing public key.
Would you like to generate one? [Yn]
Generating new SSH public key.
Uploading ssh public key /Users/adam/.ssh/id_rsa.pub
```

4.0. Listo ahora se pueden crear aplicaciones para heroku, como en este caso ya se tiene una, en el directorio de la aplicación y usando GIT se crea un repositorio, se recomienda usar cuando se trabaje en un proyecto grande un

gestor de versiones GIT es el mejor, y como ya se ha trabajado el proyecto ya se tiene instalado.

$ git init

$ git add .

$ git commit -m "init"

$ heroku create

Con las líneas anteriores se tiene el proyecto en la base de datos de GIT. Y se ha solicitado un repositorio en Heroku, el cual responde de la siguiente forma:

```
$ heroku create
Creating severe-mountain-793... done, stack is cedar
http://severe-mountain-793.herokuapp.com/ | git@heroku.com:severe-mountain-793.git
Git remote heroku added
```

Se puede subir nuestra aplicación, para esto se ejecuta

$ git push heroku master

Se vera el proceso conocido como “DEPLOY”, cuando este haya terminado se puede ir a la dirección que se mostro en la opción create y ver el sitio funcionando.

http://severe-mountain-793.herokuapp.com/

El nombre se puede cambiar desde la página web de heroku.

Ahora se puede ver desde cualquier parte del mundo.

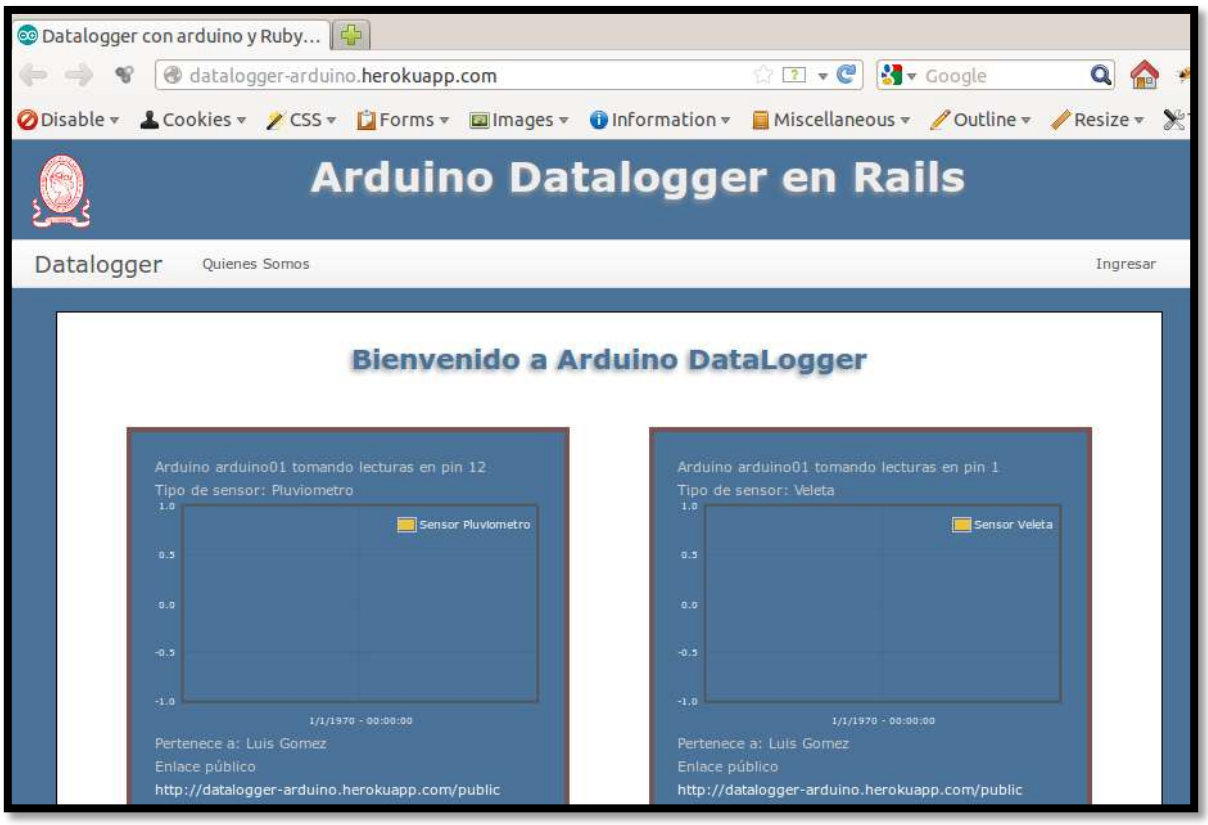

Figura 50. Viendo nuestro sitio web en heroku

Conclusiones

1. Se consiguió un sistema versátil y eficiente con posibilidades de ampliar su capacidad de memoria y su funcionalidad en diseños futuros. Entre estas se podrían nombrar, el agregar canales digitales o analógicos de entrada/salida. Ya que el sistema tiene alrededor de 54 canales digitales, de los cuales se están usando unos 20 para control de dispositivos y solamente 6 para E/S, analógicos tiene 16 de los cuales solo 6 están siendo usados como entrada de datos. Además puede agregarse toma de decisiones en base a los datos analógicos/digitales recopilados, y por su puesto ampliar la funcionalidad del control desde un sitio remoto, etc. Pero siempre teniendo como base el mismo sistema.

2. El sistema puede ser usado para monitorear el ambiente de un determinado lugar, zona, edificio, etc., desde un sitio remoto a este mismo, por lo que, no es necesario estar en el lugar de ubicación del Data Logger para conocer sus valores censados, si no que puede estar a miles de kilómetros de distancia y estar monitoreando vía web gracias a un servidor externo que está permanentemente en contacto con el Data Logger para actualizar las lecturas cada cierto tiempo determinado por el administrador del sistema.

3. Como resultado final de este proyecto se obtuvo un sistema completo para la adquisición, almacenamiento y envió de datos a un servidor externo, el cual está listo para ser implementado en una aplicación que el usuario examine y considere sea de gran utilidad.

4. El sistema implementado tiene características esenciales que le permiten ser muy competitivo en el mercado, entre sus principales características se destacan su bajo costo, tamaño y peso reducido, capacidad de ser portátil, doce canales de entrada, fácil programación y uso, capacidad de almacenamiento, servidor web, etc.

5. Todos los componentes del equipo fueron seleccionados según su disponibilidad en el mercado, que aunque no estén disponibles en el mercado local, pero son accesibles desde el exterior por un bajo costo, lo que permite una fácil reproducción. Cabe mencionar que como componente central se utilizó el microcontrolador ATmega1280 de ATMEL, incorporado

en la placa Arduino MEGA, que gracias a su capacidad de ser un dispositivo programable, le permite al sistema ser prácticamente autónomo.

6. Como todo dispositivo con componentes electrónicos, este sistema Data Logger tiene sus limitantes en cuanto a Temperatura, Humedad, etc. Por lo que hay que considerar que el dispositivo central se encuentre en un lugar donde las condiciones ambientales sean las idóneas para el buen funcionamiento, es decir, no más 30° de temperatura ni más del 50% de humedad relativa. No obstante los sensores pueden usarse en las condiciones nominales para las que están diseñados, por ejemplo el pluviómetro puede estar expuesto permanente a la lluvia, el anemómetro puede estar en ambiente libre, etc. Todas estas características las puede consultar en las hojas de datos de cada dispositivo.

7. El uso de Ruby on Rails acelera de manera increíble la creación de una aplicación web, tiene documentación suficiente y el tiempo de aprendizaje es mucho menor, corregir errores, extender la aplicación, incluir nuevas librerías, etc. es mucho mas fácil. Para un estudiante de ingeniería eléctrica es una opción perfecta.
8. Las gemas de ruby creadas para aplicaciones Rails logran acelerar aun más el desarrollo de aplicaciones, en mi propia experiencia, las mejores son HAML, Will Paginate, Nifty generators, Devise y Client side validations.
9. Heroku nos permite la experiencia de trabajar en diferentes ambientes, probar si nuestras aplicaciones son funcionales, etc. Además con el uso de GIT, podemos manejar diferentes versiones, trabajar un proyecto en grupo, si algo no funciono como se esperaba volver a una versión anterior, etc. Y ver los resultados desde cualquier parte del mundo.

Recomendaciones

1. Una buena actualización del sistema Data Logger seria, realizar una aplicación que permita al Data Logger conectarse a internet inalámbricamente, es decir, que el Data Logger pueda instalarse en la cima de un volcán y mediante una antena con enlace de radio frecuencia sea capaz de enviar sus datos a una base radio terrena y que sea dirigida a la red global (internet). Puede también utilizarse la tecnología moderna 3G o 4G ya que en el mercado hay módulos que permiten al Arduino (central del sistema Data Logger) conectarse a internet.

2. Se recomienda actualizar la versión de Ruby a la 1.9.3 y la versión de Rails a la 3.2.11, ya que el 08 de enero de 2013 fue reportado un BUG de seguridad que permite a un atacante la inyección de SQL y ejecutar código arbitrario.

Bibliografía

- Beginning Arduino, Michael McRoberts
- Arduino Programming Notebook, Bryan W. Evans
- http://www.Arduino.cc/
- http://www.ladyada.net/learn/Arduino/ethfiles.html
- Everyday Scripting with Ruby: For Teams, Testers, and You
  By Brian Marick
- The Ruby Programming Language
  By David Flanagan, Yukihiro Matsumoto
- The Ruby Way, Second Edition: Solutions and Techniques in Ruby Programming (2nd Edition)
  By Hal Fulton
  Consta de más de 400 ejemplos organizados por tema.
- The Rails 3 Way (2nd Edition) (Addison-Wesley Professional Ruby Series)
  By Obie FernandezThe Rails 3 Way ™
- Agile Web Development with Rails (4th edition)
  by Sam Ruby

- Rails guide http://guides.rubyonrails.org/
  Aquí podemos encontrar las guías para Rails para la última versión disponible, en el momento de escribir esto estaba para la versión 3.2.9. Estas guías están diseñadas para hacer que de inmediato su productividad con Rails, y para ayudarle a entender cómo todas las piezas encajan.
- Documentación de Ruby http://www.ruby-lang.org/es/documentation/
  Aquí encontrarás vínculos a manuales, tutoriales y referencias que te serán útiles a la hora de desarrollar con Ruby.
- Documentación de el framework Jquery http://api.jquery.com/
  Aquí se puede encontrar la información sobre todo lo relacionado con Jquery.
- Bootstrap http://twitter.github.com/bootstrap/
  Es un framework para estilos Elegante, intuitivo y potente para el desarrollo web más rápido y más fácil.
- Gmap http://gmap.nurtext.de/
  gMap es un plugin de jQuery ligero que ayuda a integrar Google Maps en tu sitio web. Con sólo 2 KB de tamaño es muy flexible y altamente personalizable
- HAML http://haml.info/

Haml (HTML abstraction markup language) se basa en un principio fundamental: las marcas HTML deben ser hermosas. No solo por decirlo. Haml acelera y simplifica la creación de las plantillas HTML.

- SASS http://sass-lang.com/
  Sass Hace que crear hojas de estilo sea divertido. Sass es una extensión de CSS3, agregando reglas anidadas, variables, mixins, herencia selector, y más.
- DEVISE https://github.com/plataformatec/devise
  Es una solución de autenticación flexible para Rails basadas en Warden, Está basado en rack. Es una solución completa MVC basado en los motores de Rails. Le permite tener funciones múltiples (o modelos / alcances). Se basa en un concepto de modularidad: usar sólo lo que realmente necesita.

Anexos

Anexo A. Especificaciones técnicas del Data Logger

Como todo dispositivo eléctrico el sistema Data Logger tiene sus características eléctricas que lo limitan en cuanto a la alimentación y la carga que puede manejar sin que este se sobrecargue y se quemen algunas de sus entradas o todo el sistema.
En este punto vamos a especificar los parámetros nominales caracterizan a este dispositivo, tomando en cuenta el trabajo para el cual este ha sido diseñado y las características que ofrece la placa Arduino mega que al final es el controlador de todo el sistema Data Logger.

A.1. Voltaje de operación con puerto USB de PC

El puerto USB de una PC generalmente provee un voltaje de 5 V y una corriente nominal de 500 mA, lo cual es suficiente para alimentar al sistema Data Logger, ya que el sistema incorpora elementos de bajo consumo como lo son sensores, RTC, pantalla LCD, etc. Más adelante se mostrara el consumo de los sensores y todos los elementos que componen el sistema.

A.2. Voltaje de entrada recomendado usando alimentación mediante el conector de alimentación secundario

El voltaje de operación del Data Logger (Arduino Mega) que puede ser suministrado varía entre 6 V a 20 V, entendiendo con esto que el voltaje mínimo de operación es de 6V y el máximo de 20 V, fuera de este rango el sistema podría no funcionar o en el peor de los casos quedar inservible debido a sobre voltaje de operación. Esto es suministrando el voltaje a través del conector AC, que puede ser mediante una fuente de alimentación AC-DC o por una batería.
El voltaje que se aconseja usar para la alimentación es de 9 V ya sea con una fuente AC-DC o con una batería de 9V por 150 mA, por supuesto que usando batería, el tiempo de operación quedara limitado a la calidad de la batería usada.

A.3. Características eléctricas de los sensores y dispositivos de e/s utilizados

1. Sensor de temperatura digital ds18b20: según las especificaciones técnicas de construcción este sensor opera con un voltaje de entre 3v a 5v consume entre 1 a 2 mA soportando como máximo una corriente de 5 mA.

2. Sensor de humedad y temperatura dht 22: este sensor especifica en su hoja técnica que puede operar con un voltaje de 3v a 5v y tomar como máximo 2.5 mA en su ciclo de conversión.
3. Real time clock ds1307: este reloj en tiempo real está diseñado de tal forma que opera con una batería de 3.3 v como respaldo para no perder la hora hasta con una pausa de 5 años sin estar conectado al suministro principal en placa arduino, y toma una corriente máxima de operación de 1.5 mA según la hoja técnica del mismo.
4. Sensor de temperatura lm35: este es un sensor de temperatura analógico y no consume más de 10 mA en operación y puede suministrarse un voltaje de 5v.
5. Pantalla lcd 5110: esta pantalla utiliza un buffer para proveer un voltaje constante a la pantalla lcd, este buffer se alimenta de la placa arduino mediante un voltaje de 3.3 v y una corriente máxima que oscila entre 10 mA a 15 mA.
6. Teclado: no usa ninguna carga considerable.
7. Arduino mega y modulo Ethernet: estos dos dispositivos son los más importantes dentro del data logger ya que son en conjunto el centro de operación del sistema data logger y consumen en conjunto aproximadamente 80 mA.

Por supuesto que todas las especificaciones eléctricas de cada uno de los elementos que se están utilizando las podrá ver en el anexo "d" el cual contiene las hojas técnicas de cada dispositivo

Anexo B. Lista de los elementos utilizados en la construcción del Data Logger con arduino

A continuación se presenta una lista detalla con todos los elementos internos y externos que componen al data logger con arduino y la cantidad utilizada, es decir, sensores, módulos, carcasa, tornillos.

- 1 arduino mega, (https://www.adafruit.com/products/191)
- 1 modulo Ethernet arduino, (https://www.adafruit.com/products/201)
- 1 sensor de temperatura analógico del tipo lm35
- 1 sensor de humedad y temperatura dht22 (https://www.adafruit.com/products/393)

- 1 sensor de temperatura digital ds18b20, (https://www.adafruit.com/products/381)
- Estación meteorológica (https://www.sparkfun.com/products/8942)
- 1 rtc ds1307 (https://www.adafruit.com/products/264])
- 1 teclado 3x4 (https://www.adafruit.com/products/419)
- 1 pantalla lcd nokia 3310 (https://www.adafruit.com/products/338)
- 12 jack de audio de 3.5 mm
- 36 pines macho
- 6 tornillos de 1/8 de pulgada por 1.5 pulgadas
- Un socket para circuito integrado til 16 pines
- Una caja metálica de 10 cms por 15 cms
- Un cable USB (mini)

Anexo C. Esquemas de conexión de los diferentes elementos del Data Logger con arduino

C.1. Conexión entre la pantalla LCD y la placa arduino mega

En la figura C.1 se puede observar en detalle la conexión entre de la LCD y la placa Arduino.

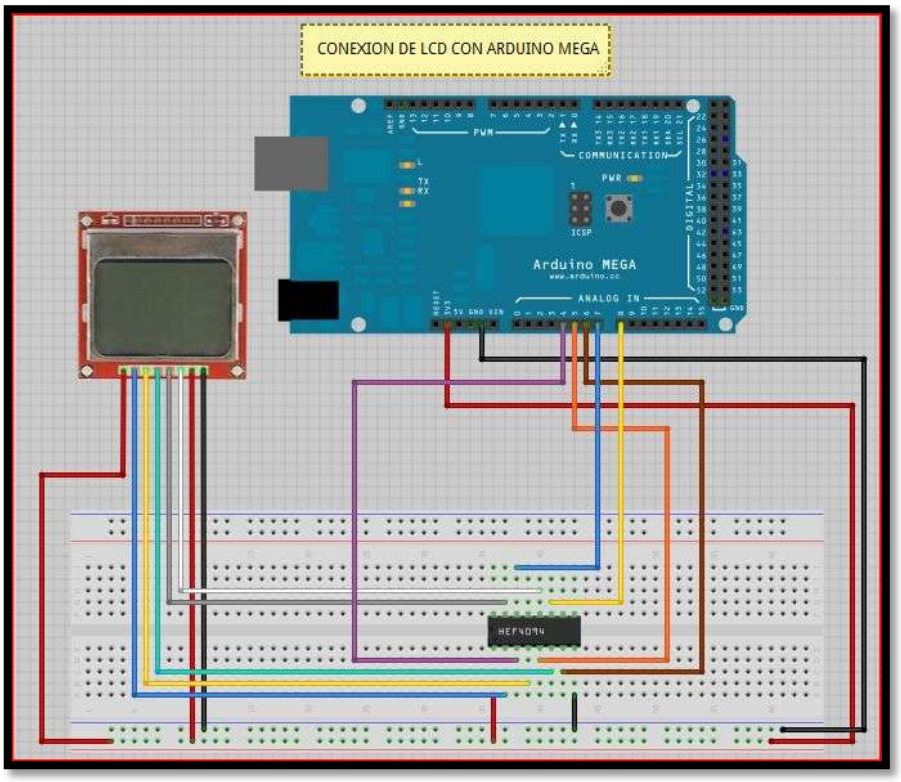

Figura 51. Conexión entre pantalla LCD y placa Arduino mega

C.2. Conexión entre la RTC y la placa arduino mega

Realizar las conexiones tal como se muestran en la figura C.2, obviando el pin SQW, el cual no está siendo utilizado y consiste en un reloj de onda cuadrada para alguna aplicación.

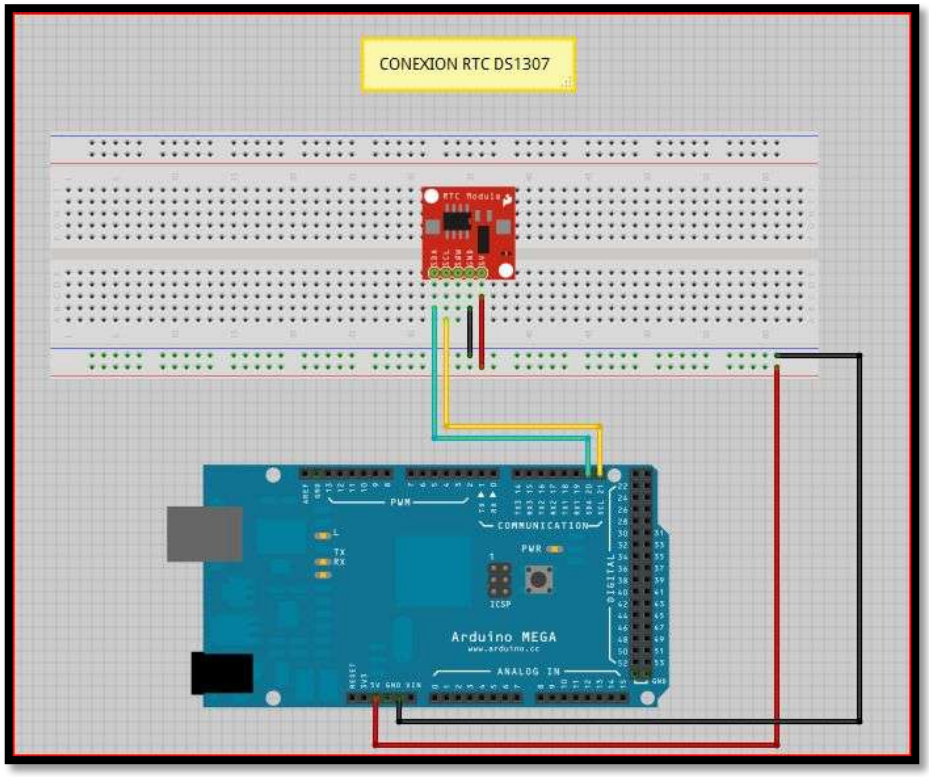

Figura 52. Conexión entre la RTC DS1307 y la placa Arduino mega

Hay que aclarar que la RCT mostrada en la imagen varia un poco respecto a la que está siendo usada en el proyecto, es decir, los pines se encuentran en orden diferente, sin embargo, los nombre son los mismos y por tanto puede ver el esquema completo con el programa FRITZING para ampliar las imágenes y ver detalladamente los pines, los archivos están incluidos en el CD anexo.

C.3. Conexión entre el teclado 4x3 y la placa arduino

La conexión es bastante simple, los pines del teclado de izquierda a derecha viéndolo de frente, van conectados a los pines 44, 42, 40, 38, 36, 34 y 32 respectivamente.

C.4. Conexión entre el modulo Ethernet y la placa arduino

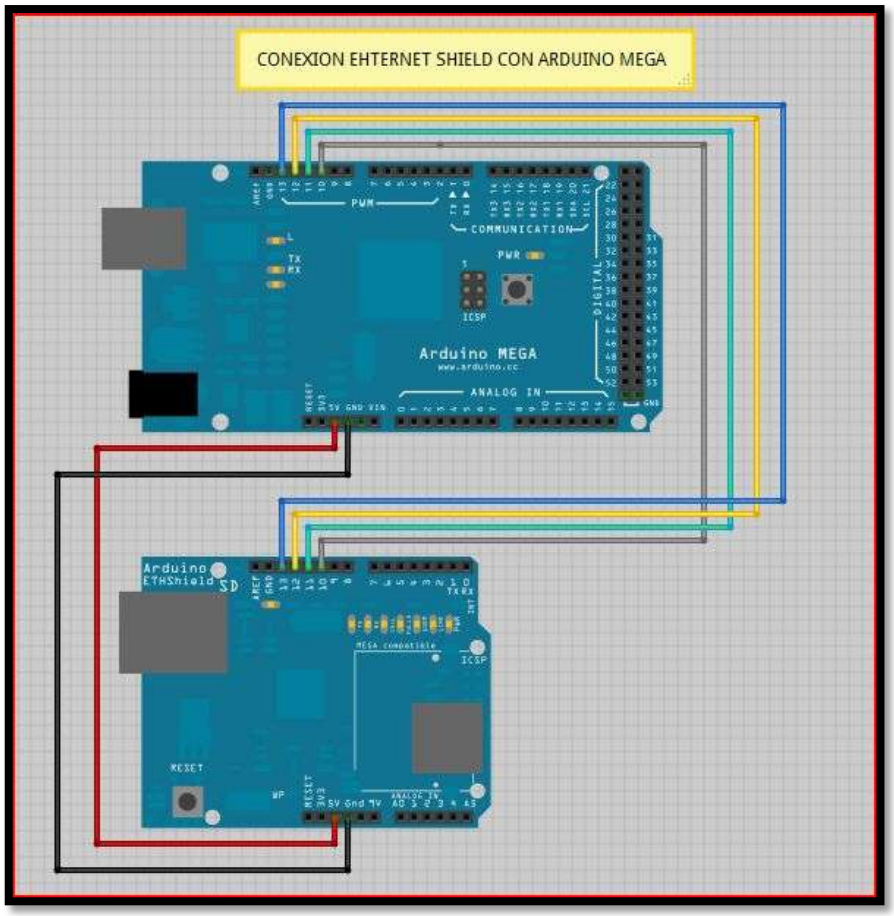

Figura 53. Conexión de la Ethernet Shield y Arduino mega

C.5. Conexión entre el sensor dht22 y la placa arduino

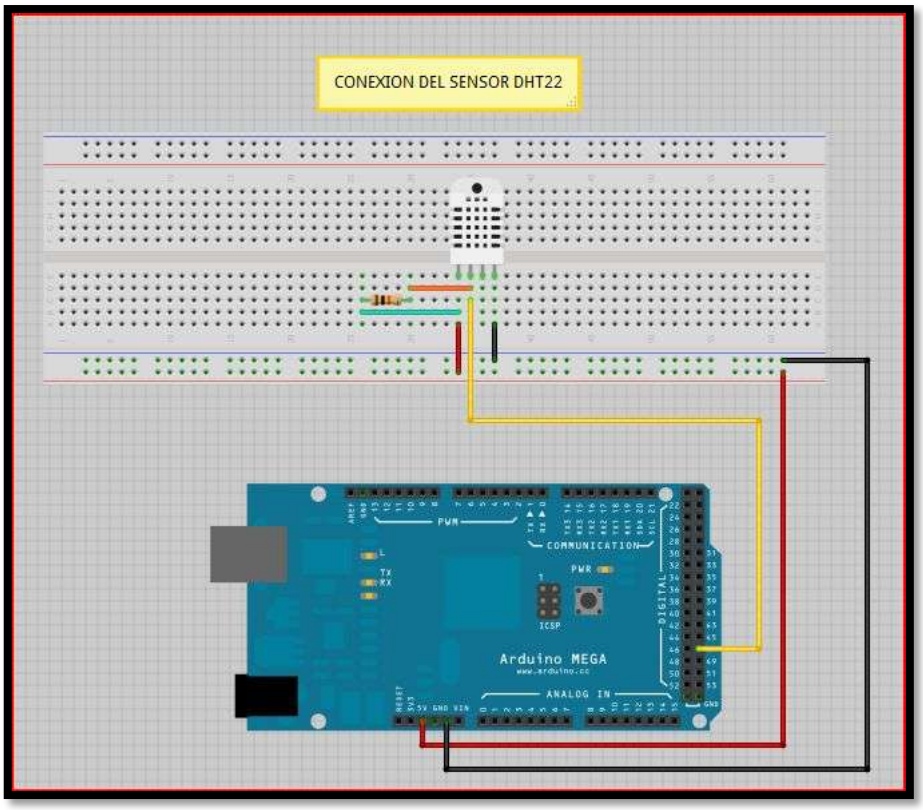

Figura 54. Conexión entre el sensor DHT22 y la placa Arduino Mega

En la figura C.5 puede verse la conexión de este sensor con la placa Arduino, puede notar que el pin 3 del sensor no está siendo utilizado, y una resistencia de 10 kΩ es necesaria entre el pin de datos y el voltaje de alimentación. Sin embargo si usa este sensor proporcionado por www.adafruit.com como el que se ve en la imagen anterior no es necesaria la incorporación de la resistencia ya que este paquete ya incorpora una resistencia de este valor y se ha dibujado solamente por cuestiones didácticas.

C.6. Conexión entre el sensor ds18b20 y la placa arduino mega

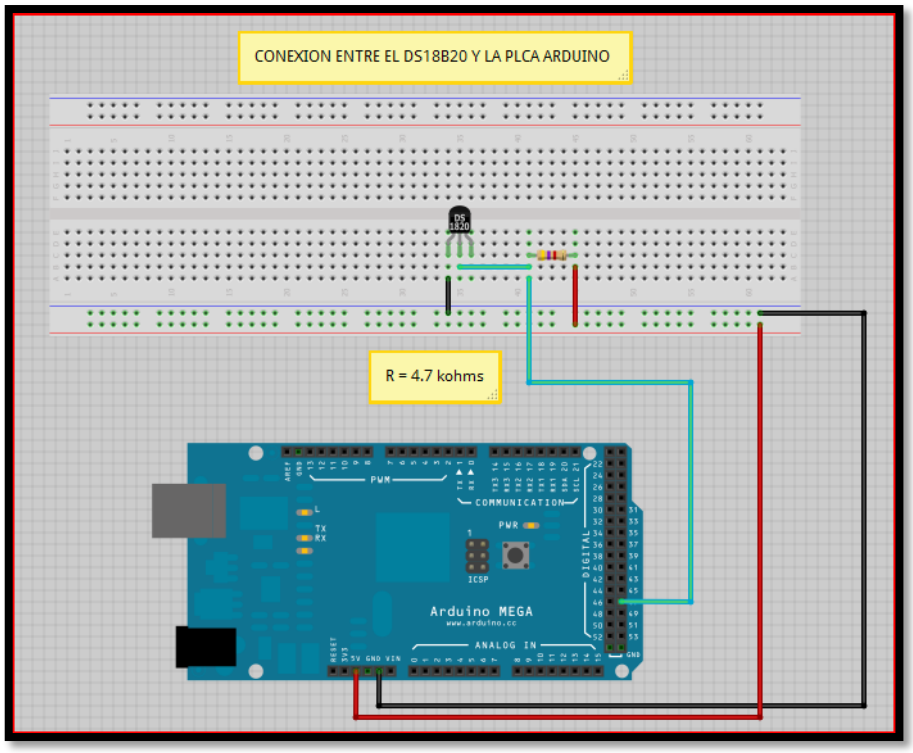

Figura 55. Conexión del Sensor DS18B20 con la placa Arduino

Cabe mencionar que el pin 12 del Data Logger (pin 47 del Arduino Mega), está reservado exclusivamente para el sensor de temperatura digital DS18B20. Por tanto no podrá conectar otro sensor de otro tipo ya que la conexión física que se requiere imposibilita la configuración de tipos, como lo es en el caso de los sensores analógicos que no requieren un pin específico para realizar su conversión a un valor comprensible de temperatura, presión, humedad, etc.

Puede observarse que el pin 3 del sensor (alimentación) puede dejarse sin conexión y aun así el sensor funciona sin problemas, sin embargo, puede conectarlo perfectamente a 5 V como se ha hecho en este proyecto.

C.7. Conexión entre el sensor lm35 y la placa arduino

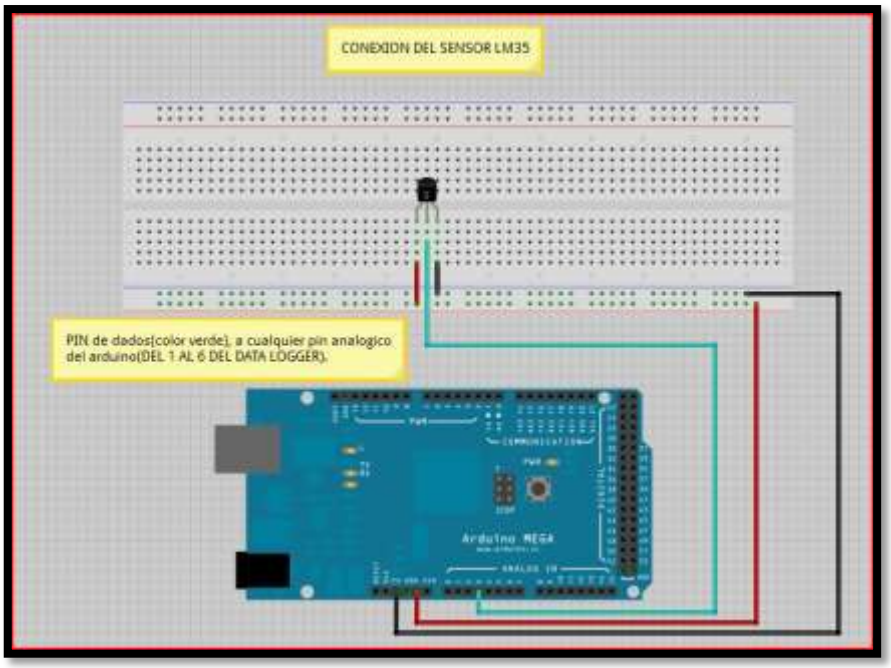

Figura 56. Conexión de sensor LM35 con la placa Arduino Mega

C.8. Conexión entre la placa arduino y la PC

La conexión de la placa Arduino con la PC se realiza mediante un cable USB mini, comunicando así al Data Logger con la PC para poder monitorear el sistema mediante monitor serial del software Arduino. Podrá ver los detalles de esta conexión en el manual de usuario.

Figura 57. Cable MINI USB

Anexo D. Costos del proyecto.

En este apartado vamos a estimar los costos aproximados que conlleva realizar este proyecto, tomando en cuenta todo los materiales y la mano de obra, es decir, tomando en cuenta el costo de todos los elementos que componen al sistema de monitoreo vía web y por su puesto el tiempo invertido en realizar este proyecto el cual se ve reflejado en el costo por hora de trabajo.

En la siguiente tabla puede verse un detalle minucioso del costo de cada uno de los elementos que componen al sistema Data Logger Arduino.

Tabla 9. Costos de los materiales utilizados

DESCRIPCION	**COSTO**
PLACA ARDUINO MEGA	$ 29.0
MODULO EHTERNET SHIELD	$ 32.0
RTC DS1307	$ 9.0
PANTALLA LCD NOKIA 3310	$ 10.0
TECLADO 3X4	$ 3.95
SOCKET PARA INTEGRADO TIL 16	$ 0.4
PINES MACHO (36 PINES)	$ 4.0
CABLE USB	$ 2.0
SENSOR DE TEMPERATURA DS18B20	$ 9.95
SENSOR DE TEMPERATURA LM35	$ 3.0
SENSOR DE TEMP Y HUMEDAD DHT22	$ 15.0
ESTACION METEOROLOGICA	$ 69.95
TORNILLOS (10)	$ 0.50
12 JACKS DE AUDIO DE 3.5 MM	$ 3.0
CAJA METALICA Y PINTURA	$ 20.0
JUMPERS DE CONEXIÓN (20 UNIDADES)	$ 4.5
POTENCIOMETRO DE 10K	$ 0.35
6 RESISTENCIAS	$ 0.6
CIRCUITO IMPRESO (PCB)	$ 10.0
TOTAL	**$ 228.70**

En la tabla anterior pueden verse los costos en detalle de los dispositivos y materiales utilizados, sin embargo, hay ciertos dispositivos que al momento de realizar este proyecto no se encontraban en el país por tano se compraron en el extranjero, en la página web www.adafruit.com y en la página www.sparkfun.com, tomando en cuenta entonces un costo adicional que es el costo de envió del producto el cual se ha estimado que es un 30% del monto de la compra, tomando en cuenta solamente la placa Arduino mega, el modulo Ethernet Shield, RTC DS1307, pantalla LCD Nokia 3310, teclado 3x4, sensor ds18b20, dht22 y la estación meteorológica, haciendo un costo de $178.85.

Por tanto,
Costo de envió = $ 53.65
Haciendo en total un gasto de materiales de: ***$ 282.36***

Luego, tomamos en cuenta el material para realizar el documento final, el cual consiste en páginas de papel bond, impresiones, anillado o empastado. Tomando en cuenta los precios de referencia siguiente,

Pagina de papel bond	: $ 0.01
Impresión	: $ 0.10
Empastado	: $ 2.50

Haciendo un costo aproximado de materiales en la elaboración del documento final: $ 22.0 por documento entregando a las autoridades de la Universidad de El Salvador 3 documentos, lo cual eleva los costos a la cantidad de:
Costo de documento final: $ 66.0

Este proyecto se realizo en aproximadamente 10 meses lo cual hace un total de unas 720 horas de trabajo invertidas por el grupo.
Sin considera el costo de la mano de obra traducida en dinero, el costo total de la elaboración de este proyecto es de:
COSTO TOTAL = $ 350.0 (Aproximadamente)