Desarrollo de sistema inmótico basado en plataforma Arduino

Subproyecto: Servidor Web

Septiembre de 2012

Autores:

Jaime Prado Cambeiro jaime.prado.7@gmail.com

César Gallego c.gallego.r@gmail.com

Introducción.

En el **Desarrollo colaborativo de un sistema inmótico** basado en plataforma Arduino se incluyen funcionalidades como: control de accesos, control de aire acondicionado, control de alumbrado, control energético, alarmas técnicas, integración con tecnologías móviles, etc. Así, a cada uno de los centros que participan en el presente proyecto, se les ha asignado uno o varios subproyectos en función de las capacidades aportadas por el centro y el personal de dicho centro. En el caso que nos ocupa, el Colegio de Fomento Montecastelo se ha encargado de la implementación de un **Servidor Web empotrado en el sistema domótico Arduino**.

Tabla de Contenidos

Contenido

1. Introducción al Servidor Web

Un **servidor web** o **servidor HTTP** es un programa informático que procesa una aplicación del lado del servidor realizando conexiones bidireccionales y/o unidireccionales y síncronas o asíncronas con el cliente generando o cediendo una respuesta en cualquier lenguaje o Aplicación del lado del cliente. El código recibido por el cliente suele ser compilado y ejecutado por un navegador web. Para la transmisión de todos estos datos suele utilizarse algún protocolo. Generalmente se utiliza el protocolo HTTP para estas comunicaciones, perteneciente a la capa de aplicación del modelo OSI. El término también se emplea para referirse al ordenador que ejecuta el programa.

1.1. Petición GET

Un servidor web opera mediante el protocolo HTTP, de la capa de aplicación del Modelo OSI. Al protocolo HTTP se le asigna habitualmente el puerto TCP 80. Las peticiones al servidor suelen realizarse mediante HTTP utilizando el método de petición GET en el que el recurso se solicita a través de la url al servidor web.

```
GET /index.html HTTP/1.1 HOST: www.host.com
```

En la barra de URL de un navegador cualquiera la petición anterior sería análoga a la siguiente dirección Web:

```
www.host.com/index.html
```

1.1.1. Esquema de una petición GET

El navegador por medio de la interfaz de usuario permite al usuario realizar una o varias peticiones web. La interfaz de usuario o entorno de usuario es el conjunto de elementos del navegador que permiten realizar la petición de forma activa. Una petición Web no sólo puede ser realizada mediante un navegador sino con cualquier herramienta habilitada para tal fin, como una consola de comandos Telnet.

Elementos del entorno de usuario más comunes en navegadores Web visuales:

Nombre	Descripción

Hipervínculo enlace o **link**	Es una porción de contenido Web, texto, imagen y otros elementos, que enlaza con una dirección Web. Al pulsar un hipervínculo el navegador genera una petición GET automática a la dirección URL de dicho link.
Formulario web	Al realizar el envío satisfactorio de los datos de un formulario, el navegador Web genera una petición GET o POST (comúnmente POST) automática a la par que envía los datos al servidor.
Barra de direcciones	Todos los navegadores incluyen una barra de direcciones mediante la cual puede accederse manualmente a cualquier dirección URL, de modo que el navegador generará una petición GET automática a dicha URL cada vez que el usuario lo desee.
Script activo o **pasivo**	Cualquier aplicación Javascript tiene acceso al estado del navegador, cómo puede modificar los datos que describen tal estado, de forma pasiva (sin medio de la intervención del usuario) o de forma activa (mediante alguna acción del usuario).

1.2. Petición POST

Es el segundo tipo de petición HTTP más utilizado. Los datos a enviar al servidor se incluyen en el cuerpo de la misma petición con las cabeceras HTTP asignadas correspondientemente respecto al tipo de petición. Generalmente se asocia con los formularios web en el que los datos suelen ser cifrados para enviarlos de manera segura al servidor.

Por motivos de convención se incluye en la petición la cabecera application/x-www-form-urlencoded que indica el formato o codificación de los datos a enviar; esta es:

variable->valor en el formato: variable=valor separada cada par variable->valor por &.

Esta cabecera, en los formularios HTML se envía automáticamente, pero en otras tecnologías web tal como AJAX, si se desea hacer correctamente una petición POST debe ser especificado o instanciado el objeto:

setRequestHeader("Content-type:application/x-www-form-rlencode");ajax.send(data);

Si se utilizase el método GET los datos deberían de ser añadidos a la URL, lo que los expondría a ser vistos de forma directa.

1.2.1. Estructura de una petición POST

Estructura típica de una petición POST		Muestra
Petition type	POST url HTTP/1.1	POST comment.php HTTP/1.1
Referer	http-url-referer	index.php
Content-Length	contentlenght-int	63
Origin	http-url-origin	http://es.wikipedia.org
User-Agent	useragent-string	Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) ...
Content-Type	content-type-string	application/x-www-form-urlencoded
Accept	mimetypes-accepted-string	application/xml,application/xhtml+xml ...
Accept-Language	language-accepted-string	es-ES,es;q=0.8
Accept-Charset	charset-accepted-string	ISO-8859-1,utf-8;q=0.7,*;q=0.3
Cookie	phpsessid-string	PHPSESSID=gm0ugf96iojuldio8i51u92716
Accept-Encoding	accept-encoding-string	gzip,deflate,sdch
Content	Content-string	&data=4&lang=es+es

- **Petition type:** Especifica el tipo de petición HTTP. *(Esta cabecera no tiene nombre, se envía tal cual)*
- **Referer:** Especifica la url desde la cual se hizo la petición POST.

- **Content-Length:** Especifica la longitud en bytes de los datos enviados en el cuerpo de la petición.
- **Origin:** Especifica la url principal del sitio.
- **User-Agent:**Especifica el identificador del navegador Web desde el cual se hizo la petición.
- **Content-Type:** Especifica el formato o MIME de los datos enviados en el cuerpo de la petición.
- **Accept:** Especifica el MIME que se espera en la respuesta.
- **Accept-Language:** Especifica el código del lenguaje esperado en la respuesta.
- **Accept-Charset:** Especifica la codificación que se espera en la respuesta.
- **Cookie:** Especifica un identificador de sesión en la petición derivado de un cookie.
- **Accept-Encoding:** Especifica el tipo de codificación (generalmente compresión) que se espera de la respuesta. *(No todos los navegadores envían esta cabecera)*

2. Librería WebDuino

2.1. Servidor Web basado en el código de Alessandro Calzavara versus WebDuino

Para la implementación del servidor Web, se optó en primer lugar por la modificación del código desarrollado por Alessandro Calzavara. Un código bastante sencillo que permite la realización de páginas web a base de strings con código estructurado en lenguaje “html”.

El problema nos lo encontramos al tratar de enviar o capturar variables por los típicos métodos POST y GET. Al estar almacenados los códigos del servidor en memoria Flash, se hacía muy difícil poder acceder a ellas para consultarlas o modificarlas (ver instrucción PROGMEN). Un problema con solución, pero el caso es que investigando la vía para atacar el problema, nos topamos con una demo de servidor web llamada “WebDuino” que de un plumazo resolvía la forma de capturar variables, y que además nos ofrecía múltiples funcionalidades como la página de “Login”.

De todas formas, lo que más gustó de la librería WebDuino fue la sobrecarga del operador “<<”, con la que se pueden cargar los textos en formato html que conforman las páginas del servidor web.

2.2. Funcionalidades librería WebDuino

La librería WebDuino está incluida en “WebServer.h”, la cual depende a su vez de la librería suministrada por la plataforma Arduino “Ethernet.h”.

Así, podemos crear en nuestro código una clase denominada “WebServer”, la cual contiene los siguientes tipos y métodos:

2.2.1. Tipos de la clase WebServer

- WebServer::ConnectionType
    - INVALID
    - GET
    - HEAD
    - POST
- WebServer::Commnad

2.2.2. Métodos de la clase WebServer

- **WebServer(urlPrefix, int port)**
- **begin()**
- **processConnection(buffer, bufferLen)**
- **processConnection()**

- **setDefaultCommand(cmd)**
- **setFailureCommand(cmd)**
- **addCommand(verb, cmd)**

- **print(...)**
- **printCRLF()**
- **printP(progStr)**

- **write(data, length)**
- **writeP(progData, length)**

- **readPOSTparam(name, nameLen, value, valueLen)**
- **nextURLparam(tail, name, nameLen, value, valueLen)**

- **radioButton(name, val, label, selected)**
- **checkBox(name, val, label, selected)**

- **int read()**
- **push(ch)**
- **bool expect(expectedStr)**

- **httpFail()**
- **httpSuccess(contentType, extraHeaders)**
- **httpSeeOther(otherURL)**

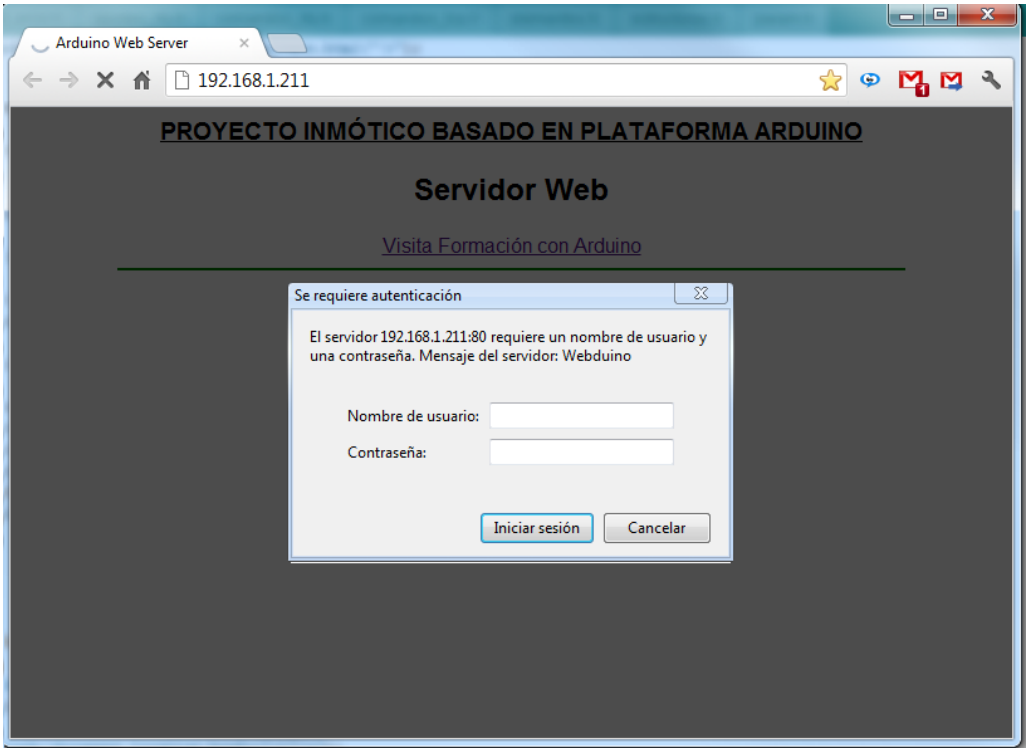
3. Funcionalidad Servidor Web

En esta primera versión del servidor en nuestra plataforma, hemos pensado en dotarle de las siguientes funcionalidades:

- Pantalla de Login
- Activación Leds. Nos permite trabajar con los leds rojo y verde incorporados al WK0500
- Formulario datos IP y MAC. Página para poder cambiar los datos de la IP y MAC del servidor
- Activación de salidas en esclavos. Desde aquí, podremos atacar sobre las salidas de un esclavo definido internamente en el código.
- Parámetros comunicación ModBus. Página muy interesante de cara a mantenimiento y puesta en marcha. Desde aquí se podrán modificar los principales parámetros de comunicación ModBus.

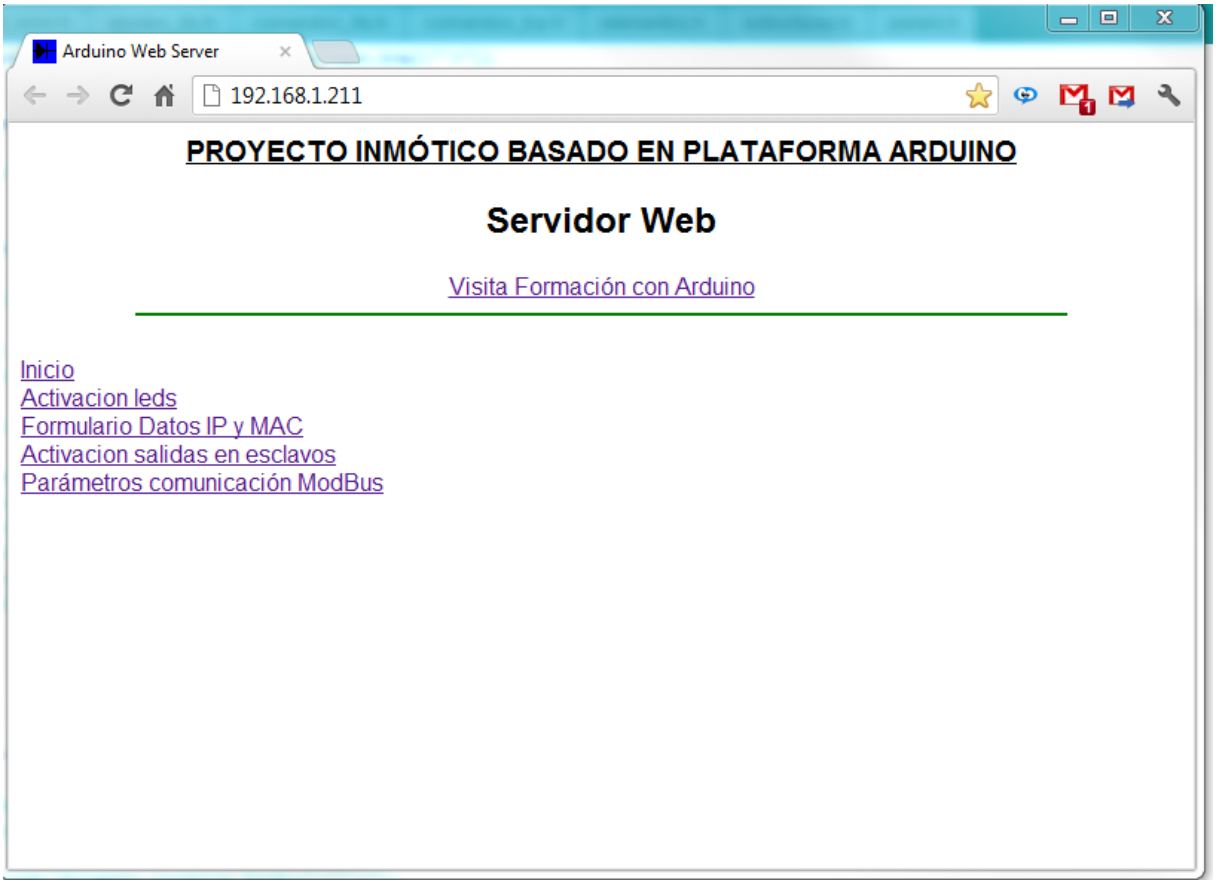
3.1. Login

Si un usuario accede a nuestro servidor (desde cualquier parte del mundo), éste deberá loguearse:



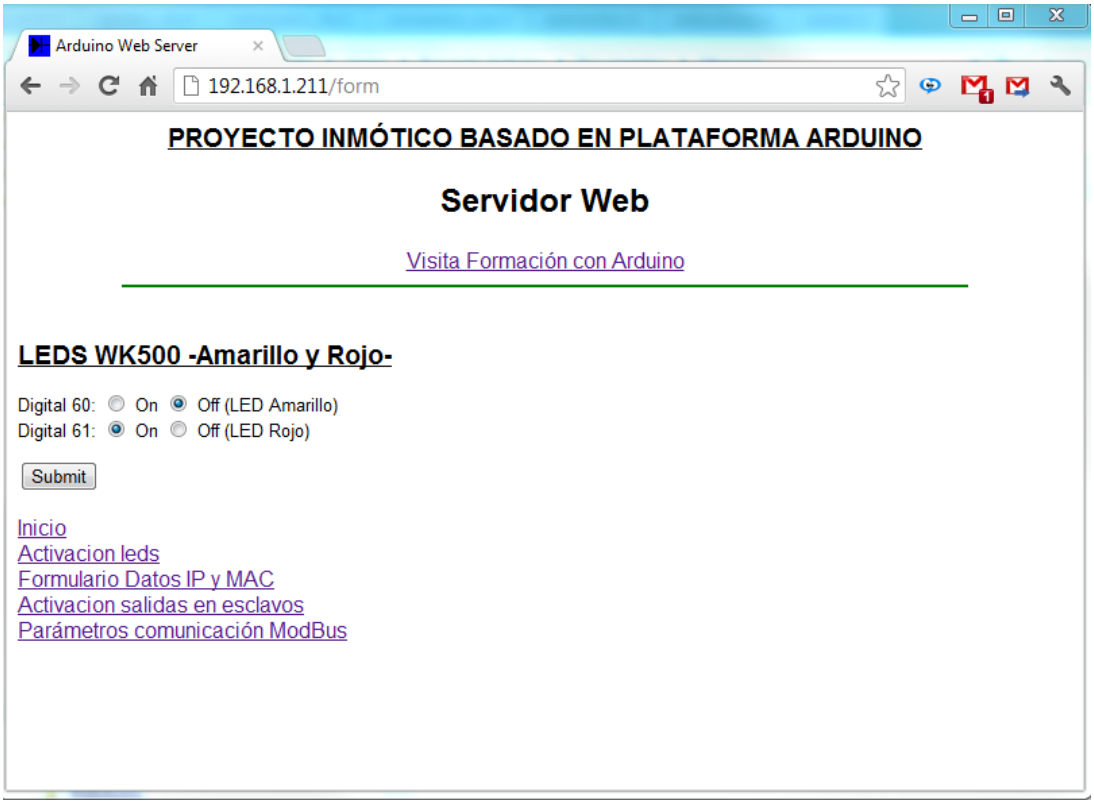
En la parte de implementación del código se define cómo podemos crear nuestros propios usuarios y cómo encriptar la contraseña.

3.2. Pantalla de inicio



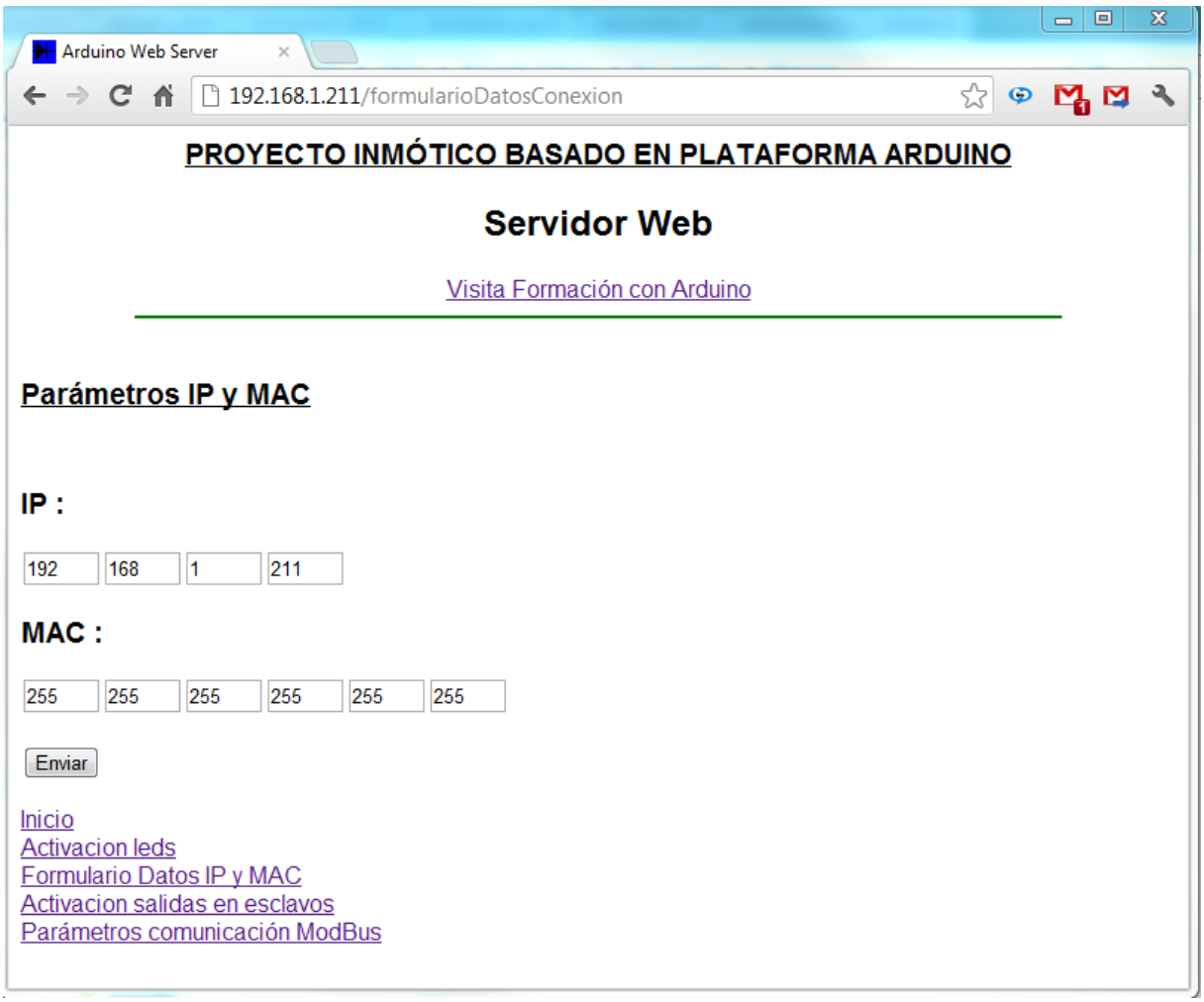
Lo primero que nos ofrece esta pantalla es un link central desde la que podemos acceder a la página de la plataforma de formación con Arduino. A continuación, en la parte izquierda, un menú nos permitirá acceder a las funcionalidades descritas.

3.3. Activación leds



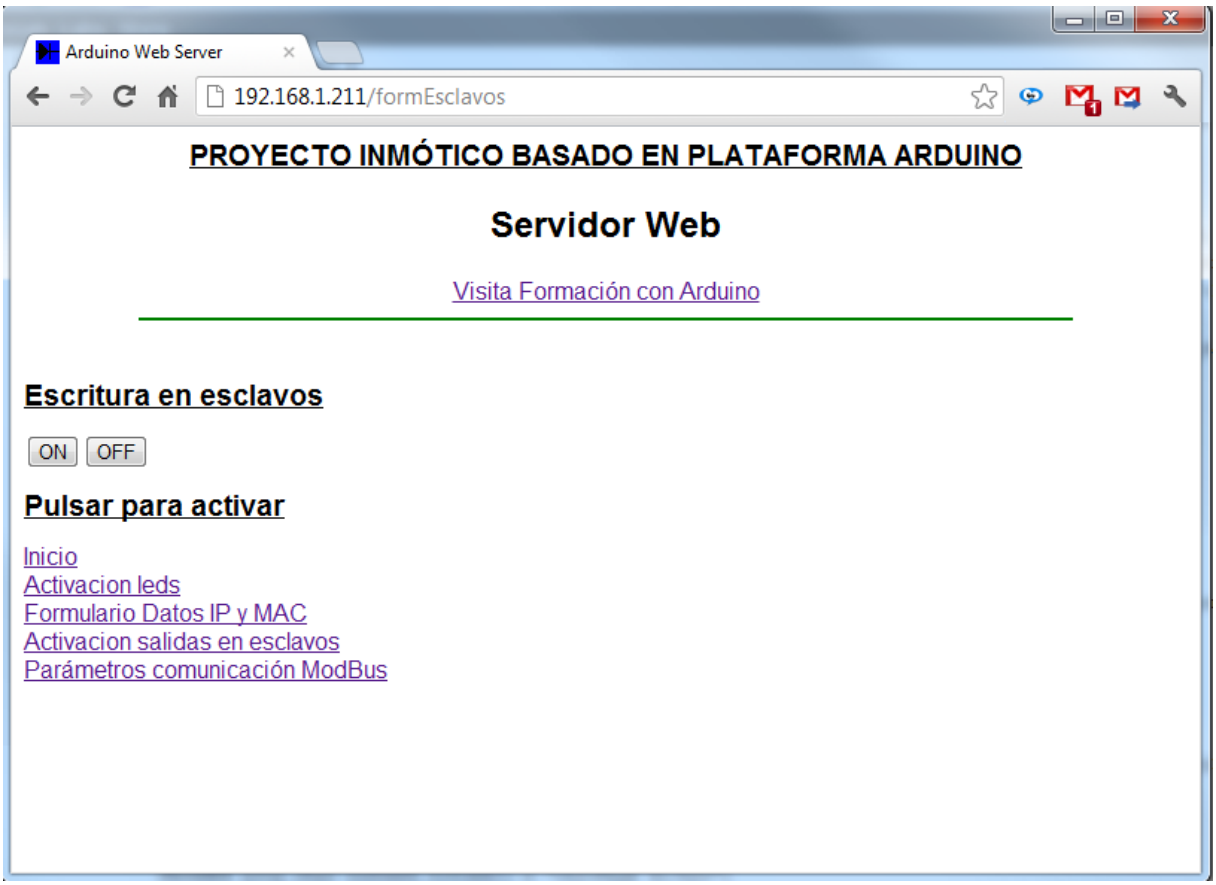
El WK0500 cuenta con dos leds, una amarillo y otro rojo. En esta pantalla, podemos probar a modificar el estado de estos dos leds.

3.4. Formulario datos IP y MAC



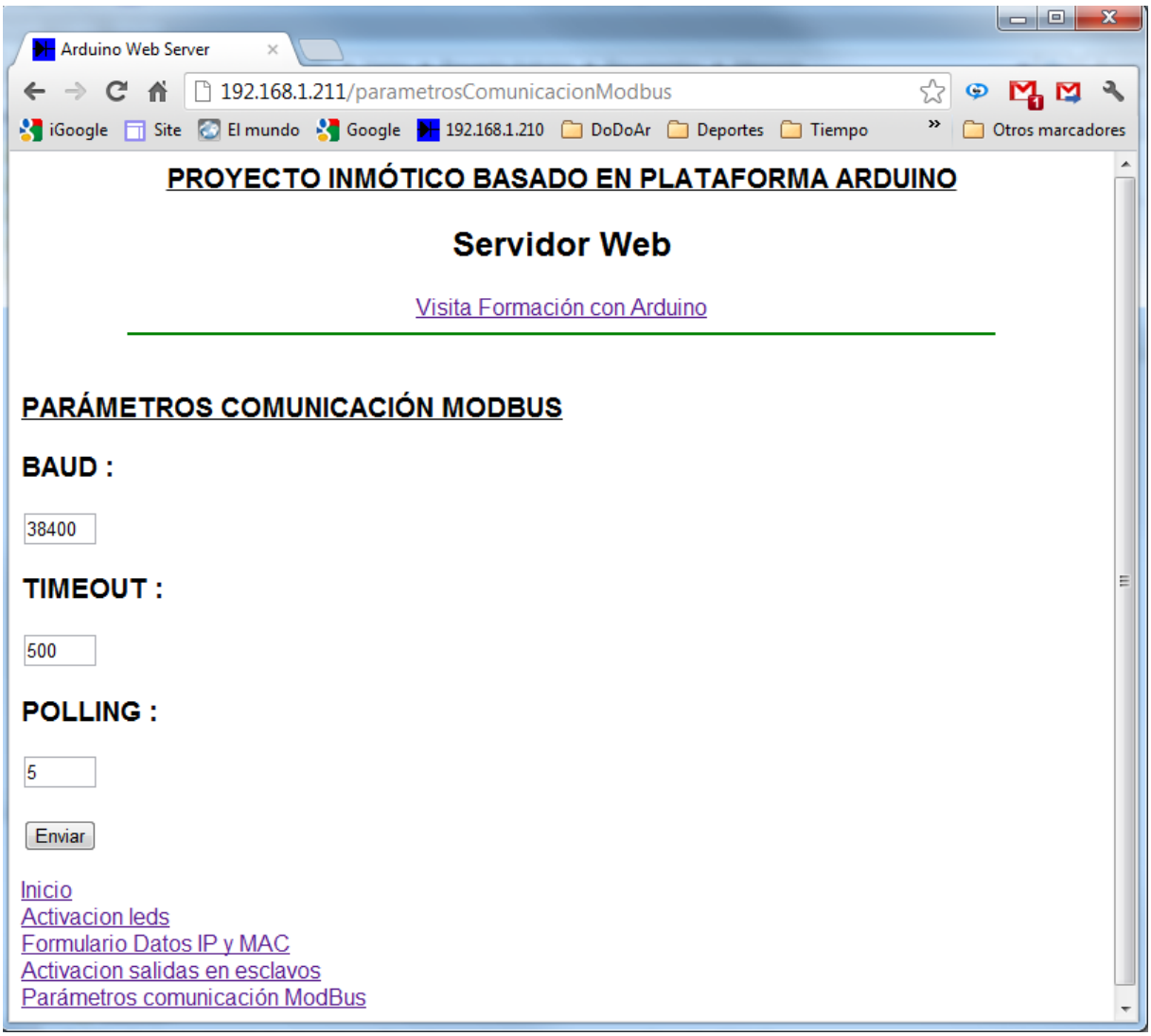
Se trata de poder cambiar los parámetros básicos de un Servidor: su dirección IP y MAC. Al pulsar enviar, los nuevos datos se almacenan en EEPROM. Éstos, son cargados en el servidor web al arrancar el módulo WK0500. Por ello, para activar los nuevos datos, deberá ser preciso reiniciar el módulo.

3.5. Activación de salidas den esclavos



Para esta parte del código, como se verá más adelante, será necesario diseñar una función a medida para el esclavo al que se desee atacar. El código será a medida de la aplicación, restringiendo de esta manera los esclavos a los que está permitido acceder desde el servidor web.

3.6. Parámetros comunicación ModBus



En este formulario se accede a la consulta y/o modificación de los parámetros de comunicación ModBus. Muy útil para puestas en marcha de las instalaciones.

4. Implementación del código

Cómo se indicó en la introducción, el servidor depende de la librería “WebServer.h”, que a su vez depende de “Ethernet.h”.

Así, declaramos al inicio del código lo que será nuestra clase principal, donde iremos volcando todas las páginas web:

```
WebServer webserver(PREFIX, 80);
```

En la parte de void setup() se inicializa la librería ethernet con la MAC y la IP guardadas en EEPROM, a la vez que se lanza el miembro de la clase webserver:

```
Ethernet.begin(mac, ip);
webserver.begin();
addPagesHTML();
```

En la parte de void loop(), el método processConnection() se encarga de procesar en todo momento el servidor, atendiendo a las peticiones que le llegan desde los clientes.

Con el método addCommand, iremos introduciendo en el servidor las distintas páginas confeccionadas en forma de funciones:

```
void addPagesHTML()
{
  webserver.setDefaultCommand(&inicioCmd);
  webserver.addCommand("private.html", &privateCmd);
  webserver.addCommand("form", &formCmd);
  webserver.addCommand("formularioDatosConexion", &formularioDatosConexion);
  webserver.addCommand("almacenarDatosConexion", &almacenarDatosConexion);
  webserver.addCommand("formEsclavos", &formEsclavos);
  webserver.addCommand("writeEsclavos", &writeEsclavos);
  webserver.addCommand("parametrosComunicacionModbus", &parametrosComunicacionModbus);
  webserver.addCommand("almacenarParametrosMB", &almacenarParametrosMB);
}
```

Además, se ha incluido una función “defineIP()” que nos dará un IP y MAC por defecto. Muy útil en caso de reinicio del sistema. Siempre sabremos a qué IP atacar en caso de

que no nos acordemos (ya ha pasado, que después de haber cambiado una IP, no recordarlo, y haber sido necesario borrarla EEPRO para recuperar el servidor web).

4.1. InicioCmd()

Aparte de la cabecera del servidor web con el link a formación con Arduino, observamos en esta función como accederemos al menú principal siempre y cuando estemos logueados en el sistema. En caso negativo, se nos remite a una página de "login" donde se introduce el nombre de usuario y el pass. Esta parte está implementada en la función privateCmd.

Simplemente indicar que la construcción del usuario y el pass está codificado en Base64. Existen múltiples páginas web donde podremos encriptar o desencriptar nuestro texto a base64, por ejemplo: http://www.motobit.com/util/base64-decoder-encoder.asp

Tal como está el código, nuestro usuario por defecto será user3 y el pass user3

```
if (server.checkCredentials("dXNlcjM6dXNlcjM=")) //user3:user3
```

Para terminar con InicioCmd(), observar como el código html puede introducirse en nuestra clase con los métodos "printP", o bien, indistintamente, con la sobrecarga del operador "<<":

```
void inicioCmd(WebServer &server, WebServer::ConnectionType type, char *url_tail, bool tail_complete)
{
  P(htmlHead) =
  "<html>"
  "<head>"
  "<title>Arduino Web Server</title>"
  "<style type=\"text/css\">"
  "BODY { font-family: sans-serif }"
  "H1 { font-size: 14pt; text-decoration: underline }"
  "P  { font-size: 10pt; }"
  "</style>"
  "</head>"
  "<body>";
```

```
 server.httpSuccess();
 server.printP(htmlHead);
 server.printP(cabecera);
 if (logueado)
 {
  server << "<a href='/'>Inicio</a></br>";
  server << "<a href='/form'>Activacion leds</a></br>";
  server << "<a href='/formularioDatosConexion'>Formulario Datos IP y MAC</a></br>";
  server << "<a href='/formEsclavos'>Activacion salidas en esclavos</a></br>";
  server << "<a href='/parametrosComunicacionModbus'>Parámetros comunicación ModBus</a></p>";
  server << "</body></html>";
 }
 else
 {
  server.print("<body onLoad='document.location.href=\"private.html\"'>");
 }
}
```

Aclarar únicamente, que a la función se le pase el objeto de la clase webserver como referencia por &server.

4.2. formCmd()

Desde esta función, se detecta si se ha requerido por parte del cliente un paso de parámetro por el método POST.

```
if (type == WebServer::POST)
```

De esta manera, se llama a la función outputPins() para, o bien mostrar el estado de los leds de salida del WK0500, o bien capturar una petición a “true” o “false” de uno de ellos y consecuentemente modificar su estado.

4.3. FormularioDatosConexion()

Desde esta función, se crea la página para poder modificar los datos de conexión IP y MAC. Aparte de implementar el código para crear los campos correspondientes (cuatro por IP y seis por MAC), se crea una función en html, ValidarCampos() que permite corroborar la validez de los datos introducidos. Una vez comprobados, la función almacenarDatosConexión() almacena en EEPROM los valores, siendo necesario reiniciar el módulo.

4.4. formEsclavos() y writeEsclavos()

Desde formEsclavos() se configuran dos botones, uno para activar una salida concreta de un esclavo determinado, y otra para desactivarla:

```
server << "<input type=button onClick=""location.href='/writeEsclavos?valor=1'"" value='ON'>";

server << "<input type=button onClick=""location.href='/writeEsclavos?valor=0'"" value='OFF'><br/><h1>Pulsar para activar</h1>";
```

Como se observa en el código, la acción sobre los botones llama a writeEsclavos, pasando el valor ‘On’ u ‘OFF’ por método POST.

Observemos ahora el siguiente código escrito en writeEsclavos() que se ejecuta al capturar el evento de los botones en el servidor:

```
if (atoi(value) == 0)
 {
   Mb.R[MB_OFFSET_COMANDO] = (Id_TCP_Web << 12) | oDESACTIVAR_LUZ_SIMPLE;   // 0x2009
   Mb.R[MB_OFFSET_COMANDO+1] = ELEM_48;                                     // 0x0030
   Serial.println("Desactiva salida elemento 48");
 }
 else
 {
   Mb.R[MB_OFFSET_COMANDO] = (Id_TCP_Web << 12) | oACTIVAR_LUZ_SIMPLE;      // 0x2008
   Mb.R[MB_OFFSET_COMANDO+1] = ELEM_48;                                     // 0x0030
   Serial.println("Activa salida elemento 48");
 }
```

Los botones actúan sobre la salida correspondiente al ELEM_48. Esta definición la podemos encontrar en param.h:

#define ELEM_48 0x30

Se trata pues de la salida 0 del esclavo 3. Es importante recordar que esta parte del código será necesaria diseñarla para cada instalación en particular, y en ella deberemos decidir que parte de la instalación podrá ser accesible desde el exterior. En ningún caso será dinámica, es decir, nunca se podrá acceder libremente a cualquier esclavo para ejecutar cualquier acción que esté a disposición del usuario dentro de la propia instalación.

4.5. parametrosComunicacionModbus()

Para terminar, esta función permite la consulta y/o modificación de los parámetros de comunicación modbus. Será muy útil en la parte de puesta en marcha y mantenimiento. Desde ella se pueden modificar los siguienes parámetros:

- Baud. Velocidad del bus. Por defecto 38400
- TimeOut: Tiempo máximo respuesta. Por defecto 500
- Pollin: Por defecto 5. Nº reintentos.

Al pulsar enviar, se llama a la última página del servidor, almacenarParametrosMB(). Igual que en writeEsclavos(), captura los parámetros por método POST y los almacenta en EEPROM. Siendo neceario reinciar el WK0500 para que surtan efecto.