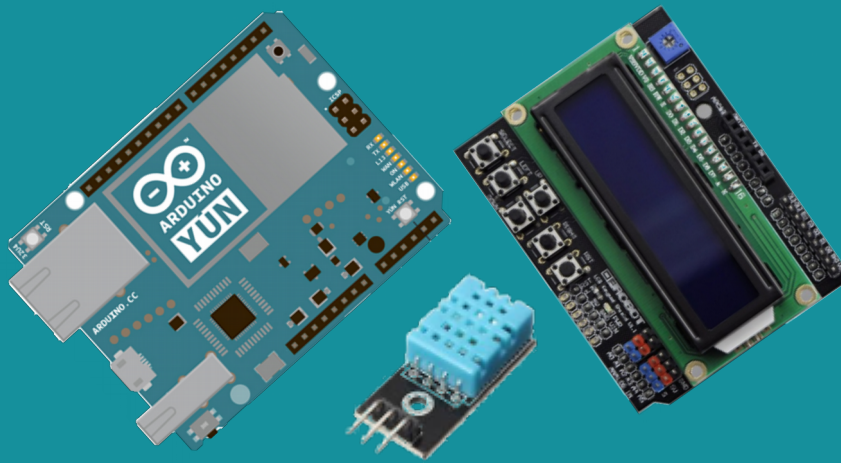


PROYECTO

Entrega Final



xivelyTM
by LogMeln



ROBOT

Sumario

1-Montaje del Dispositivo.....	4
1.1-Elementos utilizados.....	4
1.2-Pines utilizados y su utilidad.....	5
1.3-Esquema de Montaje.....	7
2-Implementación de Funcionalidades.....	10
2.1-Descripción de Funciones del Dispositivo.....	10
2.2-Lectura del Sensor.....	10
2.3-Mostrar Valores de Temperatura y Humedad.....	12
2.4-Creación/Almacenamiento en Base de Datos.....	14
2.5-Control de Errores.....	15
2.6-Modo de Depuración.....	16
3-Funcionalidades añadidas.....	18
3.1-Sincronización de dispositivos.....	18
3.1.1-Descripción.....	18
3.1.2-Herramientas utilizadas.....	18
3.1.3-Implementación.....	18
3.2-Actualización del programa.....	19
3.2.1-Descripción.....	19
3.2.2-Herramientas utilizadas.....	20
3.2.3-Implementación: Script y Sketch.....	23
3.2.4-Ejemplo de funcionamiento.....	26
3.3-Acceso/Modificación de valores mediante API Rest.....	28
3.3.1-Descripción.....	28
3.3.2-Herramientas utilizadas.....	29
3.3.3-Implementación.....	29
3.4-Envío de información a servidor centralizado.....	33
3.4.1-Descripción.....	33
3.4.2-Herramientas utilizadas.....	34
3.4.2.1-Xively IOT.....	34
3.4.2.2- Mosquitto.....	37
3.4.3-Funcionamiento.....	37
3.4.4-Integración en el sketch.....	38
3.5-Envío de E-Mails.....	39
3.5.1-Descripción.....	39
3.5.2-Herramientas utilizadas.....	39
3.5.2.1-MSMTP.....	39
3.5.3-Funcionamiento.....	39
3.5.4-Integración en el Sketch.....	40
3.6-Recepción de datos en smartphone.....	41
3.6.1-Descripción.....	41
3.6.2-Herramientas utilizadas.....	41
3.6.2.1-MQTT Dashboard.....	41
3.6.3-Lectura de valores.....	45

ENTREGA 1

1 Montaje del Dispositivo

1.1 Elementos utilizados

Para el funcionamiento de nuestro dispositivo, hemos empleado los siguientes elementos:

- **Arduino YUN** : Implementa las funciones que debe realizar el dispositivo.



- **Sensor DHT11**: Es el encargado de interactuar con nuestro entorno para recoger los datos de temperatura y humedad de este.



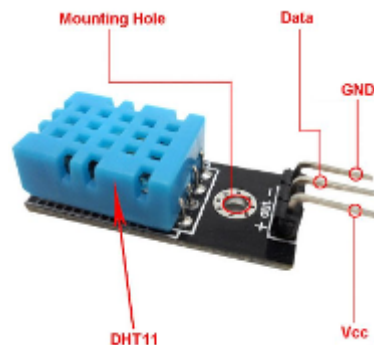
- **LCD Keypad Shield de DFRobot**: Se trata de una Pantalla de cristal líquido ensamblada sobre una placa PCB (la *Shield*) que irá conectada mediante varios de sus pines a la placa Arduino YUN. El objetivo será que muestre los valores de Temperatura y Humedad.



1.2 Pines utilizados y su utilidad

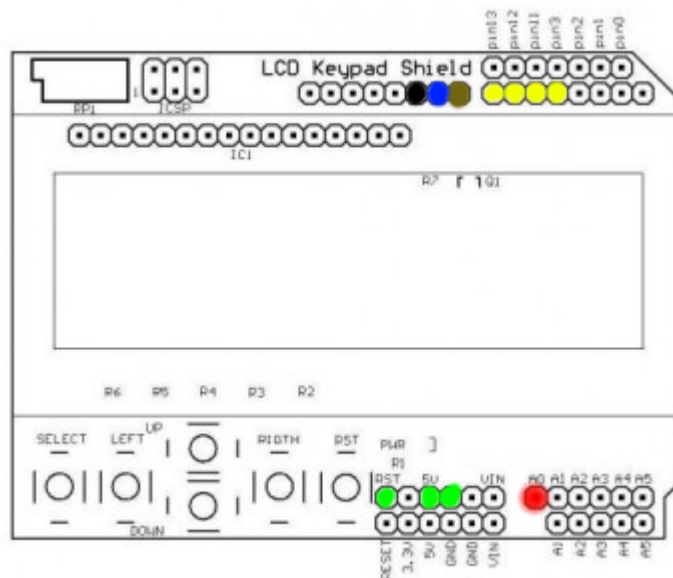
A continuación se hace una breve descripción de los pines utilizados para nuestro dispositivo y su función (lo que se conoce como *pinout*) , tanto del sensor DHT11 como del LCD Shield.

- **DHT11:**



PIN	Descripción
Vcc (+)	Conexión a corriente continua (5V)
GND (-)	Conexión a tierra
Data (out)	Salida de los datos del sensor (digital)

- **LCD Keypad Shield:**



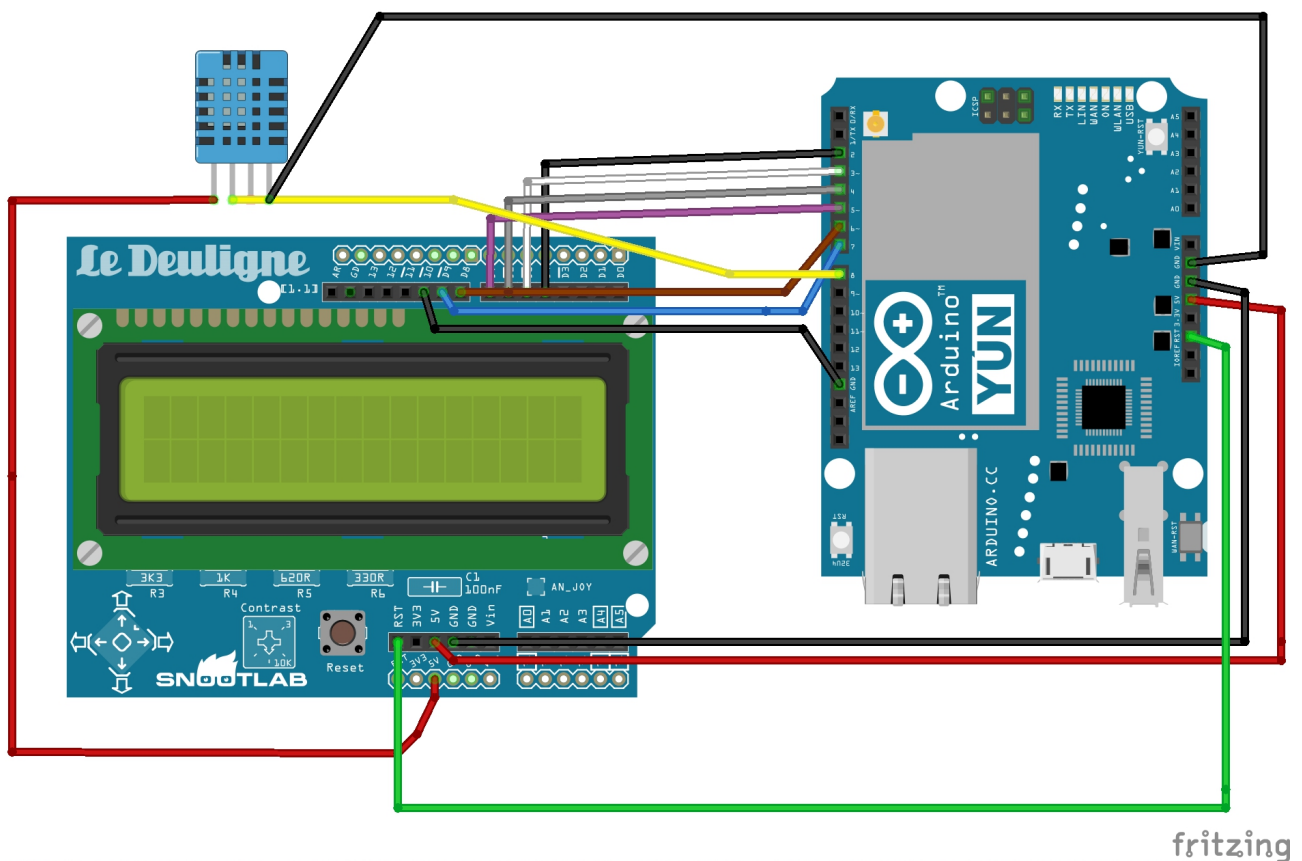
*Los pines en verde, se conectarán a su equivalente en la placa Arduino (RESET, 5V y GND)

PIN	Descripción
D7-D4	Pines DB7-DB4: Son 4 pines digitales bidireccionales para escribir o leer de la LCD desde el Arduino.
D8	Pin RS: Para elegir si se muestran por Display datos o señales.
D9	Pin E (Enable): Para habilitar la lectura/escritura de datos.
D10	Controla la retroiluminación de la LCD.
A0	Controla los botones (Arriba, abajo, Izquierda y Derecha)

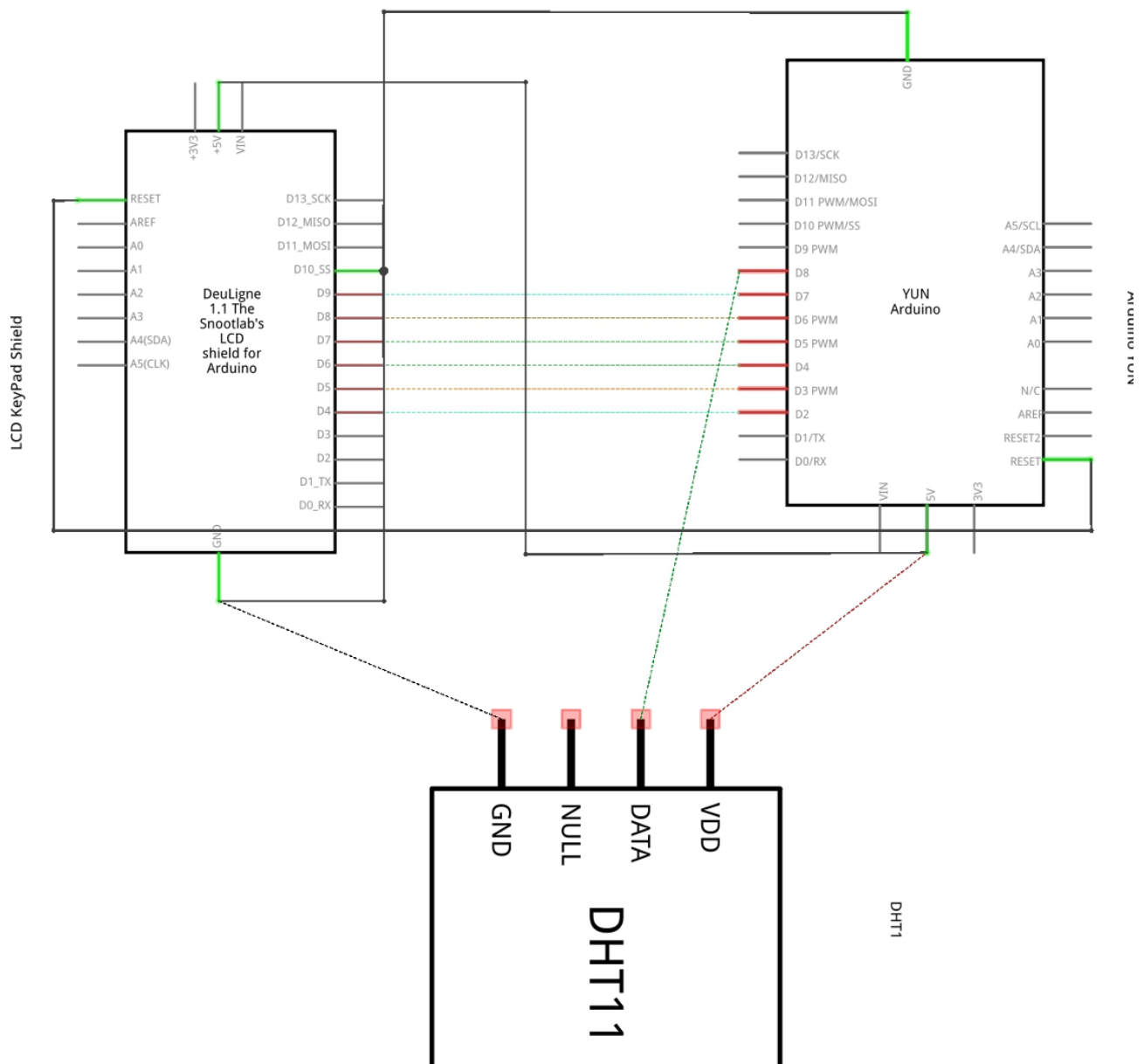
1.3 Esquema de Montaje

Se propone un esquema representativo de las conexiones pertinentes que se han realizado, así como una tabla que recoge dichas conexiones.

- Esquema de Representación



- Diagrama esquemático



- **Tabla de Conexiones**

PIN DHT11	PIN Destino
Vcc (+)	5V Shield
GND (-)	GND Arduino
Data (out)	D8 Arduino

PIN LCD KeyPad Shield	PIN Destino
D4	D2 Arduino
D5	D3 Arduino
D6	D4 Arduino
D7	D5 Arduino
D8 (RS)	D6 Arduino
D9(E)	D7 Arduino
D10 (LCD Backlight Control)	GND Arduino
RST	RESET Arduino
5V	5V Arduino
GND	GND Arduino

2 Implementación de Funcionalidades

2.1 Descripción de Funciones del Dispositivo

En esta primera entrega, el dispositivo realizará una serie de acciones básicas que se irán ampliando a lo largo del desarrollo del proyecto:

1. Lectura del Sensor DHT11
2. Mostrar Valores de Temperatura y Humedad
3. Creación/Almacenamiento de los valores en una Base de Datos .
4. Control de Errores
5. Modo de Depuración

2.2 Lectura del Sensor

Es la principal función, ya que si no se leen los datos, no se podrán gestionar posteriormente.

Consiste básicamente en leer los valores de humedad y temperatura del sensor DHT11 conectado a la placa Arduino y almacenarlos localmente en sendas variables para poder interactuar con ellas.

La lectura se hará cada 100 ms, por lo que habrá que suspender la ejecución durante ese intervalo de tiempo a cada iteración del bucle.

- Librerías necesarias: *DHT_U.h*
- Implementación en el sketch:

```

#include <DHT_U.h>
/** Crear e inicializar sensor**/
#define TIPO_DHT DHT11
#define PIN_DHT 8
DHT dht(PIN_DHT,TIPO_DHT);
int humedad = 0 , temperatura = 0;//Variables para almacenar los valores
boolean lectura = true;//Determina si la lectura ha sido correcta (true) o no(false)
/**Umbrales de temperatura y humedad máximos y mínimos,
según las especificaciones del fabricante.*
#define HUM_MAX 80
#define HUM_MIN 20
#define TMP_MAX 50
#define TMP_MIN
**
* Método que lee los valores de humedad y temperatura
* y los almacena en las variables globales 'humedad' y 'temperatura'
* Return: 'true' si no hay errores (no se pasa el Umbral minimo o máximo
* y el sensor está conectado)
* 'false' si hay errores en alguno de los valores
*/
bool leerValores(){
    humedad = dht.readHumidity();
    temperatura = dht.readTemperature();
    //Control de errores
    if(humedad == 0 && temperatura == 0){//Si el sensor se desconecta
        return false;
    }
    if(humedad > HUM_MAX) {
        return false;
    }
    if(humedad < HUM_MIN){
        return false;
    }
    if(temperatura > TMP_MAX){
        return false;
    }
    if(temperatura < TMP_MIN){
        return false;
    }
    return true;//Lectura Correcta
}
/**Configuración Inicial**/
void setup() {
    ...
    dht.begin();
    ...
}
/**Bucle de servicio**/
void loop() {
    delay(100);//La lectura se hace cada 0,1 segundos
    lectura = leerValores();
    ...
}

```

10

2.3 Mostrar Valores de Temperatura y Humedad

En estado normal representará los valores de temperatura (en grados Celsius °C) y de Humedad (en %).

La primera la mostrará en la fila de arriba(0), y la segunda en la de abajo(1),tal y cómo se indica en la imagen:



- **Librerías Necesarias:**

LiquidCrystal.h

- **Implementación del Sketch**

```
#include <LiquidCrystal.h>
/** Crear e inicializar LCD Display (RS=6,E=7,D4=2,D5=3,D6=4,D7=5).
 */
LiquidCrystal lcd(6, 7, 2, 3, 4, 5);
...
/**
 * Muestra los valores de temperatura (En °C) y humedad (en %)
 * Precondiciones: Los datos de temperatura y humedad han sido
 * leídos previamente
 */
void mostrarLCD(){
  //Mostrar temperatura
  lcd.setCursor(0,0);
  lcd.print("Temperatura: "+String(temperatura) + "°C");
  //Mostrar humedad
  lcd.setCursor(0,1);
  lcd.print("Humedad: "+String(humedad) + "%");
}
void setup() {
  ...
  //Inicialización del LCD Display
  lcd.begin(16,2);
  ...
}
/**Bucle de Servicio**/
void loop() {
  ...
  mostrarLCD();
  ...
}
```

```
#include <Process.h>
/** Variable para el proceso que llama a sistema Linux.*/
Process proceso;
/**Variable que se incrementa cada iteración**/
int seg = 0 ;
/**Base de Datos**/
/**Método para la creación de una base de datos "sensor.db"
 * que almacenará la fecha y hora,el Identificador y los datos de temperatura y
 humedad*/
void crearBD(){
    proceso.runShellCommand("mkdir /tmp/DHT11"); //Crear Carpeta DHT11
    delay(2000); //Esperar a que se haya creado
    //Crear Tabla "Mediciones"
    proceso.runShellCommand("sqlite3 /tmp/DHT11/sensor.db 'CREATE TABLE
MEDICIONES(ID INTEGER PRIMARY KEY AUTOINCREMENT,FECHA DATETIME
NOT NULL,TEMPERATURA REAL NOT NULL,HUMEDAD REAL NOT NULL);'");
}
/** Inserción en Base de Datos**/
void insertarEnBD(){
    String consulta = "INSERT INTO
MEDICIONES(FECHA,TEMPERATURA,HUMEDAD)VALUES(CURRENT_TIMESTAMP,"
+String(temperatura)+"," +String(humedad)+");";
    Serial.println(consulta);
    proceso.runShellCommand("sqlite3 -line /tmp/DHT11/sensor.db " + consulta);
}

/**Configuración Inicial**/
void setup() {
    //Inicializar Bridge
    bridge.begin();
    crearBD();
    . . .
}
```

2.4 Creación/Almacenamiento en Base de Datos

Consiste en crear una base de datos (*sensor.db*) donde almacenar los valores que se van tomando.

Se guardarán los siguientes valores cada 30 segundos:

1. Identificador (Se generará automáticamente)
 2. Fecha y Hora
 3. Temperatura
 4. Humedad
- **Librerías necesarias:**
 - Process.h* – Se necesita llamar al sistema Linux
 - **Implementación en el Sketch**

```
/**Bucle de Servicio**/  
void loop(){  
    ...  
    seg ++; //Se incrementa cada 100 ms, por tanto 30 segundos → seg = 300  
    if (seg == 300){ insertarEnBD(); seg = 0 ;  
    ...  
}
```

2.5 Control de Errores

Para que el desarrollador del dispositivo tenga constancia de que ha habido errores durante el funcionamiento, se dispone un sistema que indique por el Display una situación anómala.

Las situaciones erróneas que se controlarán son:

1. Que el sensor no se haya conectado → Se mostrará un mensaje en el LCD
2. Que la lectura supere uno de los umbrales mínimo o máximo de temperatura o humedad → Se hará parpadear 10 veces la pantalla, hasta que los valores sean normales de nuevo, que la pantalla se quedará sin parpadear.

- **Implementación en el Sketch**

```
/**Variables para el control de parpadeo en caso de error
 * Son contadores que determinarán el número de lecturas
 * (iteraciones del bucle) que tiene que parpadear o no
 */
int parpadea = 0, noParpadea = 0;
boolean bloqueo = false; //Bandera para bloquear que haga 10 lecturas parpadeando o
sin parpadear
...
bool leerValores(){
    ...
    //Control de errores
    if(humedad == 0 && temperatura == 0){ //Si el sensor se desconecta
        lcd.clear();
        lcd.setCursor(2,0);
        lcd.setCursor(0,1);
        lcd.setCursor(3,1);
        lcd.setCursor(0,2);
        return false;
    }
    ...
}
```

2.6 Modo de Depuración

El modo de depuración se activará para que muestre por el monitor serie los pasos realizados por el sketch.

- Activar Modo de Depuración: `#define DEBUG`
- Ejemplo de Depuración:

```
#ifndef DEBUG
Serial.print("Tomando valores de temperatura y humedad...");
#endif
//Se leen los valores de temperatura y humedad
#ifndef DEBUG
Serial.print("OK!\n");//Lectura correcta
#endif
```

```
/**
 * Método para controlar las veces que tiene que estar sin parpadear
 * o parpadeando un número determinado de lecturas (iteraciones del loop)
 * Para ello controla los valores de la lectura anterior y la actual.
 * -Si lectura actual es correcta - No parpadea durante 10 lecturas
 * -Si lectura actual es errónea - Indica al programa
 * que parpadee 10 lecturas consecutivas sean cuales sean las lecturas posteriores
 */
void controlParpadeos(){
  //Se bloquea para que esté 10 lecturas parpadeando o no consecutivamente
  if(bloqueo){
    if (parpadea !=0 || noParpadea !=0){
      if(parpadea > 0){
        lcd.noDisplay();
        lcd.delay(200);
        parpadea --;
      }
      if(noParpadea > 0){
        noParpadea --;
      }
    } else bloqueo = false;
  }
  //Cuando se terminan las 10 lecturas,se puede volver a comprobar
  //que sean correctas para seguir o no parpadeando
  if (!bloqueo) {
    if(lectura){//Si los valores leídos son correctos
      noParpadea = 10;
      bloqueo = true;
    }
    else{
      parpadea = 10;
      bloqueo = true;
    }
  }
  lcd.display();
}
```


ENTREGA 2

3 Funcionalidades añadidas

3.1 Sincronización de dispositivos

3.1.1 Descripción

Debemos sincronizar la placa con un servidor de tiempo mediante el protocolo *NTP* (Protocolo de Tiempo en Red), para que el sistema tenga constantemente la hora lo más precisa posible, para guardar los datos en la Base de Datos con una fiabilidad temporal, así como evitar que la hora se desfase demasiado de la real durante el transcurso del programa debido a la latencia de red y otros problemas.

3.1.2 Herramientas utilizadas

El método para conseguir esta sincronización ha sido posible mediante la herramienta para sistemas Linux *NTPD* (NTP Daemon) ; un proceso demonio que implementa el protocolo NTP versión 4 y que mantiene el sistema sincronizado mediante servidores de tiempo.

Es preciso instalar el paquete *ntpd* en nuestro Linino, ya que no viene por defecto:

```
# opkg update  
# opkg install ntpd
```

3.1.3 Implementación

Simplemente hay que hacer una sincronización con el servidor *pool.ntpg.org*

Desde el sketch se hace la siguiente llamada al inicio del programa (en el *setup()*) y una vez al día (junto con la comprobación de versiones).

```
proceso.runShellCommand("ntpd -qn -p 0.pool.ntp.org");
```

3.2 Actualización del programa

3.2.1 Descripción

Se ha desarrollado un sistema de gestión de las actualizaciones para que el programa pueda cambiar su versión a la más reciente de una forma continua.

La versión del programa consiste se compone de dos partes:

- Sketch: Código de Arduino con el funcionamiento principal del programa previamente compilado y exportado en formato “.hex”
- Scripts: Fichero comprimido (en tar.gz) con los scripts que realizan funciones externas al sketch en el procesador.

La gestión de las versiones consta de dos fases:

1. Subida de los archivos de la última versión al servidor mediante la interfaz web. Dicho servidor contendrá la información de la versión última instalada:
 - 1.1. Se seleccionan los ficheros del sketch
 - 1.2. Se selecciona la versión que se quiere subir y la aplicación web detectará si es posterior o no a la ya existente en el servidor.
 - 1.3. Si la versión es más reciente, se subirán los archivos a un directorio de la versión correspondiente y se actualizará la información de la versión en el servidor.
2. Comprobación de la versión en el servidor y descarga a la placa
 - 2.1. Se realiza en la placa mediante un script.
 - 2.2. Se comprueba que la versión instalada en el Linino (ahora veremos cómo) es más antigua que la que hay en el servidor previamente subida.
 - 2.3. Si es más reciente, se descargan los ficheros, se descomprimen, se recarga el sketch...

Puesto que el hecho de recargar el sketch y los scripts cada vez que se suba una nueva versión supondría un coste de computación y de tiempo demasiado alto, se ha limitado dicha comprobación e instalación a una vez diaria una vez lanzado el programa.

3.2.2 Herramientas utilizadas

Para poder desarrollar los pasos anteriormente descritos, se han necesitado de distintas utilidades para cada una de las partes:

- Para el Servidor Web:
 - Servidor LuCI del Arduino YÚN: Se ha utilizado el servidor web que trae Arduino por defecto para levantar nuestro propio servidor de actualizaciones. Los archivos se guardarán en el directorio de la placa `/www/versiones/actualizaciones`

- PHP:

Es necesario instalarlo y configurarlo, ya que la placa no trae configuradas las funcionalidades. La instalación se ha realizado los pasos siguiendo las directrices de este manual [\[1\]](#) :

#Instalación

```
opkg update
opkg install uci libuci libuci-lua
opkg install php5
opkg install php5-mod-json
opkg install php5-mod-curl
opkg install php5-cli
opkg install php5-cgi
```

#Configuración

```
uci set uhttpd.main.interpreter=".php=/usr/bin/php-cgi"
uci set uhttpd.main.index_page="index.html index.htm default.html default.htm index.php"
uci commit uhttpd
sed -i 's,doc_root.*,doc_root = "",g' /etc/php.ini
sed -i 's,;short_open_tag = Off,short_open_tag = On,g' /etc/php.ini
/etc/init.d/uhttpd restart
```

En la parte del servidor, se ha implementado la función de subida de archivos. Esta función realiza los siguientes pasos:

- Comprobación de la versión: Leyendo el fichero de texto *versionActual.txt* alojado en la carpeta principal del servidor. Devuelve el valor de la versión a la parte del cliente.

```
$ficheroVer = fopen("actualizaciones/versionActual.txt", "r");//Se abre el fichero
que contiene la version actual del programa
$verActual = fgets($ficheroVer);//Se vuelva la verActual en una variable
fclose($ficheroVer);
```

- Compara el valor de versión leído con el que introduce el usuario mediante el formulario web, y si el que se quiere introducir es mayor, realiza la subida de los ficheros seleccionados mediante dicho formulario.

```
$version = $_REQUEST['ver'];//Versión a subir
if( floatval($version) >= floatval($verActual) )
    #Realizar subida
    .
    .
    .
```

- HTML: Para la parte del cliente, se ha realizado un sencillo e intuitivo formulario web con tres campos: Scripts, Sketch y Versión.

```
<form enctype="multipart/form-data" class="formulario">
  <label>Subir archivo con el script (sólo .tar.gz): </label><br />
  <input name="targz" type="file" id="script" /><br /><br />
  <label>Subir archivo con el sketch (sólo .hex): </label><br />
  <input name="hex" type="file" id="sketch" /><br /><br />
  Versión actual del programa: <input type="number"
name="ver" id="version" step="0.1" value="1.0" min="1.0"><br /><br />
  <input type="button" value="Subir archivo" /><br />
</form>
```

- JavaScript + AJAX: Javascript es un lenguaje enfocado principalmente a la implementación de clientes, y se realizan llamadas a funciones AJAX para que se puedan realizar cambios sin tener que realizar cambios. Las acciones que se realizan con el uso de estas herramientas son:
 - Detectar si se ha seleccionado archivo y comprobación de la extensión de los ficheros: Deben ser tar.gz en el caso del script o .hex en el caso del sketch. Para poder seleccionar el fichero del sketch, debe estar seleccionado previamente el del script:

```
$('#file').change(function()
{
    //obtenemos un array con los datos del archivo
    var file = $("#script")[0].files[0];
    //obtenemos el nombre del archivo
    var fileName = file.name;
    ...
    if(fileName.includes(".tar.gz"))//Si se ha seleccionado un primer archivo de tipo
tar.gz
    {
        //Se comprueb el fichero del sketch
        ...
    }
}
```

- Subida de archivos mediante el método 'POST' de HTTP invocando a función PHP mediante AJAX, así como la obtención del valor de la versión que arroja dicha función.

```
$.ajax({
    url: 'upload.php',
    type: 'POST',
    ...
    success: function(data){ //En data se recibe el valor de la versión última en el
                            //servidor
                            /*Muestra de mensajes éxito/error*/
    ...
}
```

- Mostrar mensajes de éxito,error,información...

```
message = $("<span class='error'> No se ha actualizado. La versi&oacute;n actual es
m&aacute;s reciente. Versi&oacute;n actual: " + data + " Versi&oacute;n a subir:
"+version.value+" </span>");
showMessage(message);
```

- Para la parte de la comprobación e instalación de las nuevas versiones:
 - Perl: El script encargado de la comprobación de la versión y posterior instalación en la placa de los nuevos archivos si esta es más reciente. Es necesario la instalación de *perl* en la placa previamente para poder ejecutarlo:

```
# opkg update
# opkg install perl
```

3.2.3 Implementación: Script y Sketch

1. Script: El script encargado de la actualización consta de varios pasos:

- 1.1. Comparación de la versión actual y de la subida en el servidor:

Para conocer la actual versión instalada en la placa se tiene un fichero con dicha información en la ruta */etc/version.txt*

Se lee dicho fichero y se compara el valor con el que hay en el fichero del servidor *versionActual.txt*, mediante protocolo HTTP (comando *curl*) que contendrá la última versión subida.

Si es mayor que la actual, se seguirá al siguiente paso, sino arrojará mensaje informativo y saldrá del programa.

```
sub comprobacionVersion{
    my $fic1 = '/etc/version.txt';
    open (my $v1,'<',$fic1)
        or die "No se ha podido abrir el fichero '$fic1'!";
    $versionActual = <$v1>;
    #Descargar el fichero con la versión última en el servidor
    system("curl -o ver.tmp 'localhost:80/versiones/actualizaciones/versionActual.txt'
>/dev/null 2>&1 ");
    #Se almacena en un fichero temporal
    my $fic2 = 'ver.tmp';
    #Se abre el fichero temporal con el nº de versión
    open (my $v2,'<',$fic2)
        or die "No se ha podido abrir el fichero '$fic2'!";
    $versionUltima = <$v2>;

    print "Versión actual instalada en Linino: $versionActual";
    print "Versión más reciente del servidor: $versionUltima\n";
    if ($versionUltima >= $versionActual) {
        #Continuar siguiente paso
    }else {
        print "El programa se encuentra ya en su versión más reciente.\n";
    }
}
```

1.2. Creación de directorios y descarga de los ficheros en sus respectivas rutas:

Se crean los ficheros */etc/script* y */etc/sketch/* donde se guardarán los archivos descargados, sobrescribiendo los anteriores. Se descargarán los ficheros subidos , que tendrán el formato *versionX.X/vX.X.tar.gz* y *vX.X.hex* , donde "X.X" representa la versión más reciente disponible en el servidor. Para descargar dichos ficheros puede hacerse por dos métodos (mediante *curl* a la dirección del servidor de la versión específica:

http://arduino.local/versiones/actualizaciones/versionX.X , o bien obteniéndolo de la carpeta */www/versiones/actualizaciones/versionesX.X*, ya que se encuentra en un directorio local de la placa, mediante el comando *cp*. Una vez obtenidos los ficheros, se renombrarán siempre a *sketch.hex* y *script.tar.gz* , y guardarán en las rutas antes citadas. De esta forma, al copiarse a dichas rutas sobrescribirán al anterior y nos aseguraremos de que el único archivo que haya siempre será el último.

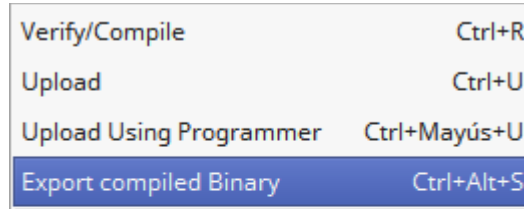
En el caso del fichero de scripts, será necesario descomprimirlo una vez en su carpeta, y posteriormente borrar el archivo, ya que ocupa espacio innecesario.

```
sub creacionFicheros{
    $dirSketch = '/etc/sketch';
    $dirScript = '/etc/script';
    unless(-e $dirSketch or mkdir $dirSketch) {
        die "Error al crear el directorio: $dirSketch\n";
    }
    unless(-e $dirScript or mkdir $dirScript) {
        die "Error al crear el directorio: $dirScript\n";
    }
    return;
}

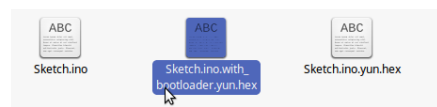
sub descargaFicheros{
    # 1 - Descarga del sketch
    #my $comandoDescarga = "curl -o $dirSketch/sketch.hex
localhost:80/actualizaciones/version".$versionUltima."/v".$versionUltima.".hex > /dev/null 2>&1";
    my $comandoDescarga = "cp /www/versiones/actualizaciones/version".$versionUltima."/v".
$versionUltima.".hex $dirSketch/sketch.hex" ;
    print "Descargando sketch del servidor...";
    system($comandoDescarga);
    print "OK! \n";
    # 2 - Descarga del script
    $comandoDescarga = "curl -o $dirScript/script.tar.gz
localhost:80/versiones/actualizaciones/version".$versionUltima."/v".$versionUltima.".tar.gz > /dev/null
2>&1";
    print "Descargando scripts del servidor...";
    system($comandoDescarga);
    print "OK! \n";
    return;
}
```

1.3. Carga del Sketch:

En esta fase el script se encarga de mezclar el sketch previamente compilado exportado mediante el IDE mediante la opción: **Sketch** → **Exported compiled binary**



EL sketch que debemos subir al servidor será aquel que no tenga bootloader:



Una vez subido al servidor (se verá en un ejemplo cómo) y descargado en el sistema de ficheros de Yún, estará listo para ser mezclado con el bootloader y ser subido al microcontrolador mediante *avrdude*; una de las herramientas para programar memorias on-chip de AVR.

El código de la función que lo realiza es:

```
sub cargarScriptSketch{
    # 1 - Descomprimir archivo de scripts en carpeta de scripts
    print "Descomprimiendo script...";
    system("cd $dirScript && tar -xvf script.tar.gz && rm script.tar.gz > /dev/null
2>&1");
    print "OK!\n";
    # 2 - Carga del sketche en el microcontrolador
    print "Cargando sketch....\n";
    system ("merge-sketch-with-bootloader.lua $dirSketch/sketch.hex > /dev/null
2>&1");
    system ("run-avrdude $dirSketch/sketch.hex ");#> /dev/null 2>&1");
    print "OK! \n";
    return;
}
```


2. Sketch: Como todas las acciones están en el script, el sketch sólo se basa en hacer una llamada al sistema Linux mediante el Bridge a la ejecución de dicho script mediante Perl:

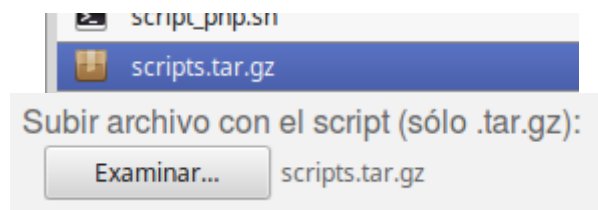
```
void ProyectoDU::actualizacionSincronizacion(){
//Sincronización
proceso.runShellCommand("ntpd -qn -p 0.pool.ntp.org");
//Búsqueda de actualizaciones
proceso.runShellCommand("perl /etc/script/controlVersiones.pl");
}
```

3.2.4 Ejemplo de funcionamiento

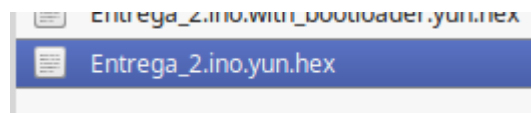
1. Abrir en el navegador la dirección del servidor: <http://arduino.local/versiones>



2. Seleccionar el archivo de scripts (debe ser tar.gz o nos dará error):



3. Seleccionar sketch (debe estar seleccionado el de scripts y ser del formato .hex)



4. Seleccionar la versión a subir (deberá ser superior a la ya instalada o no permitirá la subida):

Versión actual del programa:

Subir archivo

No se ha actualizado. La versión actual es más reciente. Versión actual: 2 Versión a subir: 1.0

Debemos pues subir la versión 2.1 o superior (seleccionamos la 2.2, por ejemplo):

Versión actual del programa:

Subimos los archivos mediante el botón y esperamos a que se carguen. Si la subida ha sido correcta, podremos ver en el Repositorio (opción *Acceso al Repositorio*) que la nueva versión ha sido subida y renombrada con éxito para su posterior descarga automática por parte del sketch.

Parent Directory	-		
version1.0/	2018-01-08 19:08	-	
version1.1/	2018-01-09 17:01	-	
version1.2/	2018-01-09 17:52	-	
version2.2/	2018-01-18 17:02	-	
versionActual.txt	2018-01-18 17:02	3	

Parent Directory	-		
v2.2.hex	2018-01-18 17:02	61K	
v2.2.tar.gz	2018-01-18 17:02	1.3K	

2.2

5. La actualización de la versión a esta 2.2 subida se hará una vez se cumpla el plazo de un día mediante el sketch. Otra opción podría haber sido un proceso en segundo plano comprobando el estado del fichero de la versión del servidor, y en el caso de detectar cambios; realizar la actualización, pero supondría un mayor coste computacional.

3.3 Acceso/Modificación de valores mediante API Rest

3.3.1 Descripción

Se ha decidido que se pueda facilitar el acceso a los datos de la sala en tiempo real de forma remota. Para ello se ha optado por configurar una API Rest que pueda devolver dichos valores de una forma sencilla y transparente para el usuario.

Los valores que se podrán obtener mediante este servicio web serán:

- Temperatura
- Humedad
- Umbrales Mínimo Y Máximo de Temperatura y Humedad

Así mismo, también se deberá poder tener la opción de poder cambiar los límites máximo y mínimo de temperatura y humedad según lo requiera la situación.

En mi caso, he optado por realizar la parte del servidor (la que atiende las peticiones de los clientes) dentro del sketch, y una aplicación web cliente que muestra los datos y permite cambiar los valores ya mencionados.

3.3.2 Herramientas utilizadas

- Para la parte del servidor:
 - Librería *BridgeServer.h* de Arduino: Es una clase que permite realizar llamadas al Yún.
 - Librería *BridgeClient.h*
 - PHP: Para la aplicación web, se utilizará PHP para tomar los valores de las URLs de la API y devolvérselas al lado del cliente.
- Para el cliente:
 - HTML + CSS: Creación de una interfaz intuitiva que muestre los valores y permita el cambio de algunos de ellos.
 - JavaScript + AJAX: Realizar las funciones asociadas a los botones, a los campos numéricos...

3.3.3 Implementación

➤ Servidor:

Se ha utilizado la clase *BridgeServer* para atender las peticiones de los clientes que llamen a la API.

El servidor se encargará de leer cada cliente que conecte con él mediante una de las siguientes URLs:

- <http://arduino.local/arduino/temp> → Devuelve valor de la temperatura
- <http://arduino.local/arduino/hum> → Devuelve valor de la humedad
- [http://arduino.local/arduino/temp/alto\[/valor\]](http://arduino.local/arduino/temp/alto[/valor]) → Devuelve [Modifica] umbral máximo de temperatura [con el valor introducido]
- [http://arduino.local/arduino/temp/baja\[/valor\]](http://arduino.local/arduino/temp/baja[/valor]) → Devuelve [Modifica] umbral mínimo de temperatura [con el valor introducido]
- [http://arduino.local/arduino/hum/alta\[/valor\]](http://arduino.local/arduino/hum/alta[/valor]) → Devuelve [Modifica] umbral máximo de humedad [con el valor introducido]
- [http://arduino.local/arduino/hum/baja\[/valor\]](http://arduino.local/arduino/hum/baja[/valor]) → Devuelve [Modifica] umbral mínimo de humedad [con el valor introducido]

El servidor irá procesando a los clientes por partes (*/temp,/alta...*), y realizando las acciones que procedan para cada petición.

Método principal que escucha las peticiones de los clientes:

```
void ApiRest::atenderClientes(){
  BridgeClient client = server.accept();
  // There is a new client?
  if (client) {
    // Process request
    procesar(client);
    // Close connection and free resources.
    client.stop();
    //Serial.println("Cliente procesado con éxito");
  }
  delay(50); // Poll every 50ms
}
```

Método que procesa la llamada para conocer la temperatura (es igual para la humedad)

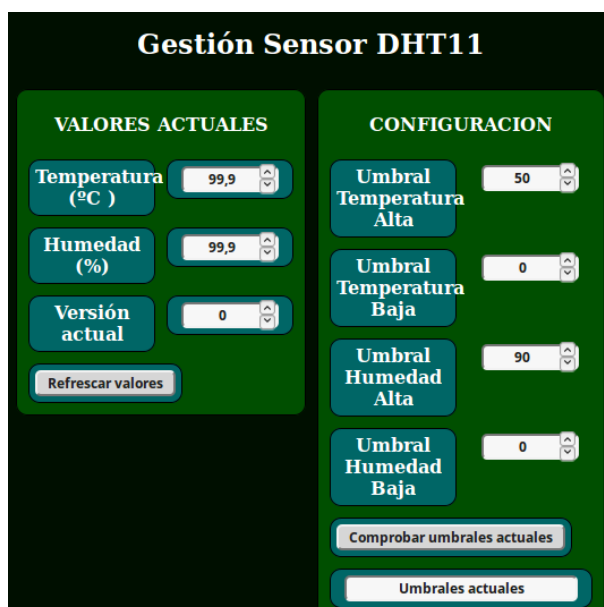
```
void ApiRest::procesarTemperatura(BridgeClient client){
    String command = client.readStringUntil('/');
    if (command.startsWith("alta"))
    {
        procesarTempMAXmin(1,client);
    }
    else if (command.startsWith("baja"))
    {
        procesarTempMAXmin(0,client);
    }
    else client.print(temperatura);
}
//Si es máxima o mínima
void ApiRest::procesarTempMAXmin(int tipo,BridgeClient client){
    String command = client.readStringUntil('/');
    command.trim();
    //client.flush();
    if(command != "")
    {
        if (tipo == 0)
        {
            MIN_TMP = command.toInt();
            client.print("El umbral minimo de la temperatura es ahora:");
            client.print(String(MIN_TMP));
        }
        if (tipo == 1)
        {
            MAX_TMP = command.toInt();
            client.print("El umbral maximo de la temperatura es ahora:");
            client.print(String(MAX_TMP) + "grados C");
        }
    }
    else
    {
        if (tipo == 1) client.println(String(MAX_TMP));
        else client.println(String(MIN_TMP));
    }
}
```

➤ Cliente:

La parte del cliente se basa en la API que levanta el Arduino, ya que no es más que una web que realiza llamadas a las URLs antes enumeradas.

Para su implementación, se han desarrollado dos formularios, uno que simplemente muestra los valores de Temperatura, Humedad y la Versión actual del programa, para lo cuál habrá que refrescarlos de forma manual, de forma que realice las llamadas a la API y los vuelque en los campos de dicho formulario.

El otro formulario, servirá para modificar los valores de los umbrales mediante los campos numéricos. También se podrán conocer los umbrales actuales.



Como ya se ha mencionado antes, el funcionamiento de esta web se basa en llamadas con el método 'POST' realizadas a través de AJAX, a funciones PHP que se encargan de leer los valores de la API y devolverlos al cliente.

Ejemplo de función PHP: *Leer temperatura:*

```
<?php
$url= 'http://localhost:80/arduino/temp';
$content = "";
$file = @fopen($url, 'r');
if($file){
    while(!feof($file)) {
        $content .= @fgets($file, 4096);
    }
    fclose ($file);
}
echo floatval($content);
?>
```

Ejemplo de función JavaScript: *Cambiar umbral mínimo de la temperatura*

```
function cambia_umbralTempBaja(){  
    var valor= document.getElementById("tempBaja_input").value;  
    $.ajax({  
        url: "http://arduino.local:80/arduino/temp/baja/" + valor,  
        type:'POST',  
        success: function(){  
  
        }  
    });  
    mostrarMensaje();//Mensaje de éxito  
}
```

El código de la implementación se adjunta en la entrega, por lo que no será preciso extenderlo en la documentación.

3.4 Envío de información a servidor centralizado

3.4.1 Descripción

Otra funcionalidad de nuestro sistema, será el envío de los datos a un servicio en la nube que pueda almacenarlos y tratarlos, así como representarlos de una forma automática.

Hemos optado por la plataforma Xively, que ofrece de estos servicios de una forma más o menos sencilla de utilizar y que además cuenta con una versión gratuita.

Dicha plataforma nos provee de un Broker, encargado de la gestión de los dispositivos que añadamos (en nuestro caso sólo el Arduino YÚN), así como de la gestión de la información que le enviemos.

La forma elegida para la comunicación de nuestro dispositivo con dicho Broker será el protocolo *MQTT* (Message Queue Telemetry Transport) ; un protocolo sencillo de Publicación/Subscripción muy utilizado en IoT, y que se basa en subscribirse o publicar en un canal o 'topic' los datos deseados y que aquél elemento que esté suscrito a dicho 'topic' podrá recibir.

En nuestro caso, los 'topics' que utilizaremos para comunicarnos con el Broker serán 3:

- **Temperatura**
- **Humedad**
- **Log** : Por el que se enviarán los distintos errores de lectura contemplados (Dispositivo desconectado, temperatura demasiado alta...)

3.4.2 Herramientas utilizadas

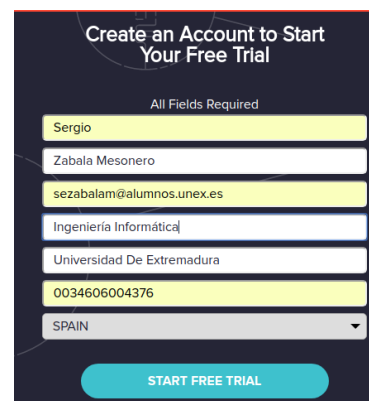
3.4.2.1 Xively IOT

Plataforma que permite conectar dispositivos IoT y que ofrece servicios de gestión de dispositivos, guardando su ubicación, su información , así como comunicarse con dichos dispositivos.

Para poder utilizar esta herramienta es preciso darse de alta para obtener una prueba gratuita.

Ello se realiza en su página web oficial: <https://www.xively.com>

Tendremos que seleccionar la opción *Free Trial* e introducir nuestros datos para obtener las credenciales.



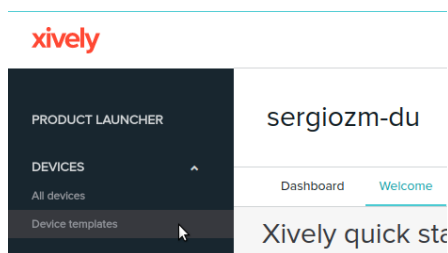
The image shows a web form titled "Create an Account to Start Your Free Trial". It includes a note "All Fields Required". The form fields are: Name (Sergio), Surname (Zabala Mesonero), Email (sezabalam@alumnos.unex.es), Major (Ingeniería Informática), University (Universidad De Extremadura), ID Number (0034606004376), and Country (SPAIN). A "START FREE TRIAL" button is at the bottom.

Una vez registrados, se nos proporciona un subdominio con el nombre que elijamos; en mi caso: **sezabalam-du**

Podremos acceder ya a la plataforma con nuestra identificación.

El siguiente paso será la creación de una plantilla de dispositivo (Device Template), que contendrá los canales por los que queremos comunicarnos con el Broker:

1. Accedemos a *Devices* → *Device Templates*



2. Seleccionamos *Create new Device Template*



3. Le damos un nombre descriptivo



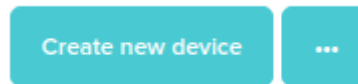
4. Hecho esto, añadimos los tres canales de comunicación, que serán del tipo “Time Series”, con la opción *+Add new channel*



5. Le ponemos el nombre que lo identifique (Temperatura, Humedad y Log)

A screenshot of the 'Add channel' dialog box. The dialog has a title bar with a close button. Inside, there is a section for 'CHANNEL NAME' with a text input field containing 'Temperatura'. Below this, there are two sections: 'CHANNEL TYPE' with a dropdown menu showing 'timeSeries', and 'KINESIS BRIDGE' with a dropdown menu showing 'Do not send'. At the bottom right of the dialog, there are two buttons: 'Cancel' and 'Save Channel'.

6. Una vez hecho esto para los tres canales, ya podremos salvar el Template y crear nuestro dispositivo siguiendo esta plantilla:

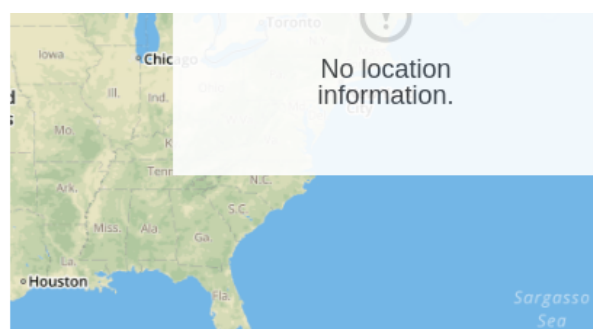
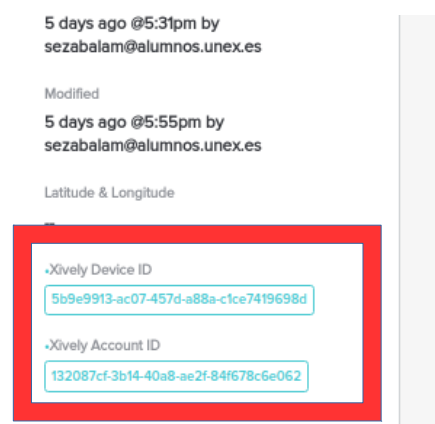


7. Utilizamos la plantilla creada (*Arduino YUN*) y como identificador le asignamos la dirección MAC de la placa:

A screenshot of a web form titled 'CREATE NEW DEVICE' with a close button (X) in the top right corner. The form contains the following elements: a 'Select a template' dropdown menu with 'Arduino YUN' selected; two radio buttons for 'Single' (selected) and 'Batch'; a 'Device's serial number' text input field containing '90:A2:DA:FB:0A:6E'; a link '+ Associate device with a group'; and 'Cancel' and 'Create' buttons at the bottom right.

8. Ya tenemos nuestro dispositivo creado en el Broker, ahora necesitaremos apuntar tres valores para la posterior comunicación con él:

- **Device ID y Account ID**



- **Contraseña**



3.4.2.2 Mosquitto

Una vez configurado el broker, será necesario configurar una aplicación cliente que utilice el protocolo MQTT en nuestro Linino para la publicación y suscripción de datos.

Dicha aplicación será *Mosquitto*, cuya configuración consta de los siguientes pasos:

1. Descarga de los siguientes paquetes a un directorio local de Linino:
 - https://downloads.openwrt.org/barrier_breaker/14.07/ar71xx/generic/packages/libcares_1.10.0-1_ar71xx.ipk
 - https://downloads.openwrt.org/barrier_breaker/14.07/ar71xx/generic/packages/libmosquitto_1.3.5-1_ar71xx.ipk
 - https://downloads.openwrt.org/barrier_breaker/14.07/ar71xx/generic/packages/mosquitto-client_1.3.5-1_ar71xx.ipk
2. Instalación de dichos paquetes:

```
# opkg update
# opkg install $dirPaquetes/libcares_1.10.0-11_ar71xx.ipk
... (Igual para el resto de paquetes)
```

3. Descarga del fichero de certificados CA de la URL https://github.com/xively/xively-client-c/blob/master/res/trusted_RootCA_certs/xi_RootCA_list.pem y almacenado como *certificate.crt* en el Linino
4. Creación de scripts de publicación con los valores obtenidos de Xively (Account ID...)

```
mosquitto_pub \
-p 8883 \
-h broker.xively.com \
-i 132087cf-3b14-40a8-ae2f-84f678c6e062 \
-d \
-u 5b9e9913-ac07-457d-a88a-c1ce7419698d \
-P OG/7r4gVTAopM8DURSzCSc5ZzcAzKzYF9SAllfvoi+E= \
-t xi/blue/v1/132087cf-3b14-40a8-ae2f-84f678c6e062/d/5b9e9913-ac07-457d-a88a-c1ce7419698d/Canal \
-m "Mensaje" \
--cafile "/etc/script/mosquitto/certificate.crt"
```

Diagram illustrating the command arguments for `mosquitto_pub` with labels pointing to specific values:

- Account ID**: Points to `132087cf-3b14-40a8-ae2f-84f678c6e062` (the `-i` argument).
- Device ID**: Points to `5b9e9913-ac07-457d-a88a-c1ce7419698d` (the `-u` argument).
- Password**: Points to `OG/7r4gVTAopM8DURSzCSc5ZzcAzKzYF9SAllfvoi+E=` (the `-P` argument).
- Ruta del certificado**: Points to `"/etc/script/mosquitto/certificate.crt"` (the `--cafile` argument).

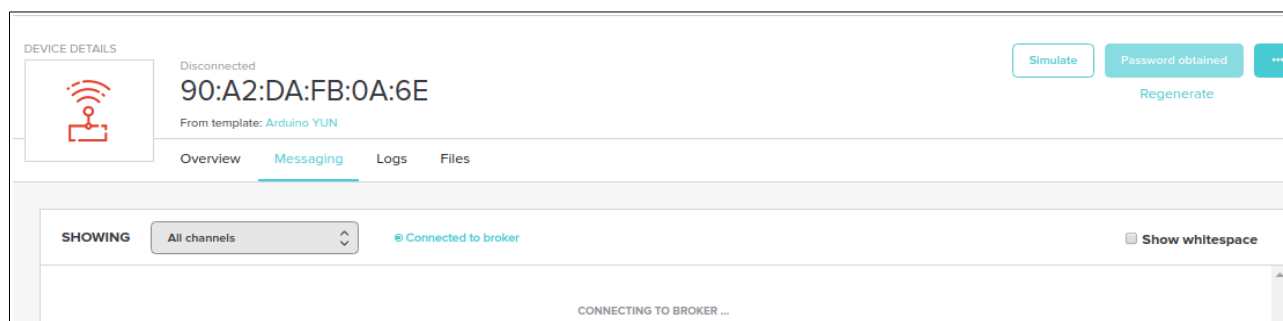
3.4.3 Funcionamiento

Una vez tenemos configurado el cliente MQTT así como el Broker, podemos probar a enviarle mensajes.

Mediante un script *mosquitto_mqtt.sh* con el anterior código, sustituyendo el valor “Canal” por “Temperatura” , y el Mensaje por un parámetro \$1 , podemos enviar mensajes al broker ejecutándolo por terminal mediante la sentencia :

```
# /bin/sh mosquitto_mqtt.sh “,Temp°C,39,”
```

Para visualizar los mensajes enviados debemos acceder a nuestro dispositivo en Xively y acceder al apartado *Messaging*:



Ya podremos ver mensajes como el que hemos enviado según se vayan recibiendo, y filtrando los canales que queremos ver (en nuestro ejemplo, el de temperatura):



3.4.4 Integración en el sketch

Como toda la carga de trabajo se encuentra en el script, desde el sketch no será necesario nada más que realizar una llamada mediante *Process* al linino proporcionándole por parámetro los datos leídos de temperatura , humedad o error.

Para el envío de valores correctos de temperatura y humedad, se ha definido la función *envioABroker()* dentro de la clase *ProyectoDU*, que se invocará cada 2 minutos si la lectura es correcta; y para el envío de errores al canal *Log*, una función denominada *envioError()*, invocada cada vez que haya errores:

```
void ProyectoDU::envioABroker(){
    proceso.runShellCommand("/bin/sh /etc/script/mosquitto/mosquitto_mqtt.sh " +
        String(temperatura) + " " + String(humedad));
}
void ProyectoDU::envioError(){
    proceso.runShellCommand("/bin/sh /etc/script/mosquitto/error_mqtt.sh "+String(mensaje)+"");
}
```

3.5 Envío de E-Mails

3.5.1 Descripción

Para que el usuario del servicio pueda estar informado de la situación de la sala en cada momento y avisarle si ocurre alguna anomalía para solucionarla, se ha configurado un servicio de envío de e-mails a la cuenta de correo del usuario cada vez que ocurra un problema.

Cuando haya habido una lectura errónea, se enviará un correo ; si ésta no es solucionada en una hora, se reenviará otro, y en el momento en que sea solucionada se enviará un correo informando de que la situación vuelve a ser correcta.

3.5.2 Herramientas utilizadas

3.5.2.1 MSMTTP

Se trata de un MTA (Agente de Tranferencia de Correo), que enviará correo a través de nuestra cuenta utilizando el servidor SMTP de *Gmail*.

Su instalación en OpenWRT es sencilla: `#opkg update`
`#opkg install msmt`

Posteriormente será preciso editar su fichero de configuración(/etc/msmtprc)

```
account default
host smtp.gmail.com
port 587
auth on
user sezabalam@alumnos.unex.es
password *****
auto_from off
from sezabalam@alumnos.unex.es
tls on
tls_starttls on
tls_certcheck off
logfile
syslog LOG_MAIL
```

3.5.3 Funcionamiento

Tras la configuración del MTA es sencilla su utilización, simplemente se invoca al comando *sendmail* con la cadena de texto que se quiera enviar:

```
#echo -e 'Subject: Asunto del Mensaje \r\n\r\n Cuerpo del Mensaje' | sendmail  
sezabalam@alumnos.unex.es
```

3.5.4 Integración en el Sketch

Se ha creado una función en la clase *ProyectoDU* llamada *enviarEmail(bool correcto)* que dependiendo del parámetro que se le pase enviará un mensaje de aviso de problemas o de que la situación se ha vuelto correcta.*

```
void ProyectoDU::enviarEmail(bool correcto){  
    if (correcto) proceso.runShellCommand("echo -e 'Subject: Problema solucionado. \r\n\r\n Las  
lecturas vuelven a ser correctas.' | sendmail sezabalam@alumnos.unex.es");  
    else proceso.runShellCommand("echo -e 'Subject: Problema en la lectura: \r\n\r\n Revise el  
dispositivo y los niveles de la sala.' | sendmail sezabalam@alumnos.unex.es");  
}
```

Dicha función será invocada con parámetro *correcto* a 'false' en la primera lectura errónea ,y no enviará otro mensaje de error hasta que pase una hora mínimo.

En cuanto la situación se torne correcta de nuevo se invocara con *correcto* a 'true' una sólo vez hasta que vuelva a haber problemas.

**Inicialmente se parametrizó el envío de mensajes de aviso de error , indicando qué tipo de error era, pero se descartó por problemas al realizar la llamada, mostrando un mensaje genérico.*

ENTREGA 3

3.6 Recepción de datos en smartphone

3.6.1 Descripción

Se pide implementar la posibilidad de recibir los valores de forma actualizada en nuestro móvil mediante un Cliente MQTT para Android.

Se instalará una aplicación y configurará en el móvil para lograr recibir los valores de Temperatura, Humedad y Errores, a través del Broker anteriormente configurado.

3.6.2 Herramientas utilizadas

3.6.2.1 MQTT Dashboard

De entre todas las aplicaciones disponibles para la recepción de datos de dispositivos IoT y su representación, una de las más populares por sus sencillez es *MqTT Dashboard*, disponible gratuitamente en el *Google Play Store*.

A continuación, se explica los pasos a seguir para su instalación y configuración para conectar con el broker y recibir los valores del dispositivo.

1. Descarga e Instalación

Descargar en el *Play Store* la aplicación oficial e instalar la aplicación *IoT MQTT Dashboard*



2. Creación y configuración de la conexión

Añadiendo nueva conexión (botón +) que llamamos *arduinoyun*

Los valores que se introducen son los valores del dispositivo de Xively (obtenidos anteriormente) , así como el nombre del broker de Xively, y el puerto 8883.

Yoigo ... 100 % 20:40

Connection SAVE

Client ID
arduinoyun

Server
broker.xively.com

Port
8883

Username
5b9e9913-ac07-457d-a88a-c1ce7419698

Password
.....

SSL ☒

Key store file
/storage/emulated/0/Downlo... CLEAR

Key store password

Nombre del broker de Xively

Puerto de comunicación

Xively Device ID

Password generada de Xively

Fichero de certificados

El fichero de certidicados será el mismo *certificate.crt*, que se ha renombrado a *.bks*

3. Creación y configuración de los canales

Debemos crear los 3 canales (Temperatura, Humedad y Log), con la opción + de la esquina superior derecha.

La configuración del canal se hará de la siguiente forma:

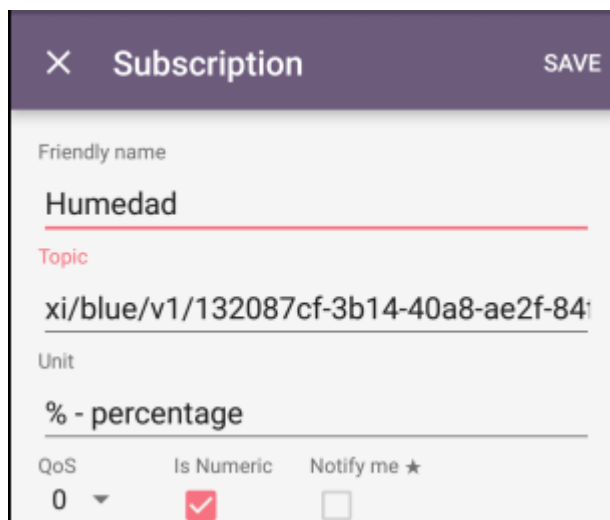
Le asignamos un nombre que lo identifique.

En el Topic, debemos poner la dirección del Topic correspondiente, con el formato:

xi/blue/v1/IdCuenta/d/IdDevice/temperatura

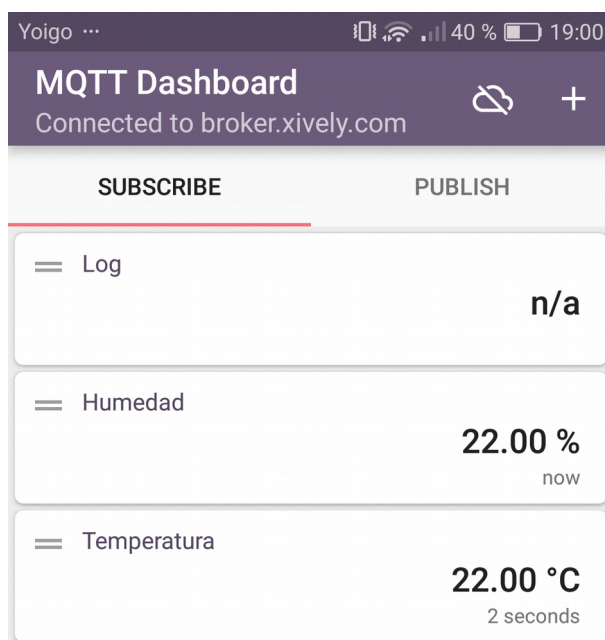
en este caso, el de la Humedad.

Seleccionamos la unidad a representar, y marcamos la casilla 'Is Numeric', para que la aplicación pueda representar los valores de forma automática gráficamente.



3.6.3 Lectura de valores

Para leer los valores, no tenemos más que abrir la aplicación, acceder a la conexión *arduinoyun-du* previamente creada, y, una vez que el sketch esté mandando mensajes al Broker, podemos consultar los valores, que se mostrarán de la siguiente forma:



Estructura del Programa

El programa se ha estructurado de tal forma que se repartan las acciones que realiza para no sobrecargar el sketch.

Las librerías que se necesitan son:

- *Bridge.h*
- *BridgeServer.h* y *BridgeClient.h*
- *LiquidCrystal.h*
- *Process.h*
- *DHT.h*

Aparte de estas librerías ya implementadas , he creado dos nuevas librerías que realicen las funciones del programa:

- *ApiRest.h*: Se encarga de todas las acciones que tienen que ver con el utilizar el Arduino como un servidor utilizando *BridgeClient* y *BridgeServer*, como pueden ser escuchar a los clientes, o el devolver los distintos valores que se piden. Las funciones de esta clase son:

```
void atenderClientes();  
void configurar();  
void procesar(BridgeClient client);  
void procesarTemperatura(BridgeClient client);  
void procesarTempMAXmin(int tipo, BridgeClient client);  
void procesarHumedad(BridgeClient client);  
void procesarHumMAXmin(int tipo, BridgeClient client);  
int umbralTempMax();  
int umbralTempMin();  
int umbralHumMax();  
int umbralHumMin();  
void actualizarTemp(float temp);  
void actualizarHum(float hum);  
void actualizarVersion ( float _ver);
```

- *ProyectoDU.h*: Realiza el grueso de funciones del programa, todos los que tienen que ver con el tratamiento de valores leídos, funciones de la Base de Datos, envío de mensajes al Broker, y el resto de funcionalidades.

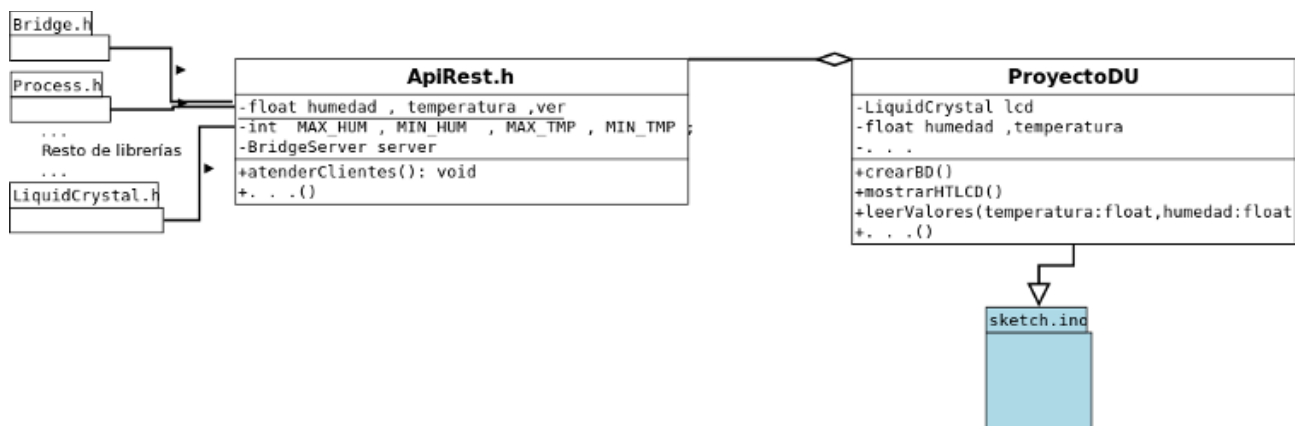
```

ProyectoDU();
void inicializarLCD();
void configuracionInicial();
bool leerValores(float _humedad, float _temperatura);
void controlParpadeos();
void mostrarHTLCD();
void crearBD();
void insertarEnBD();
void escucharClientesAPI();
void actualizacionSincronizacion();
void enviarEmail(bool correcto);
void envioABroker();
void envioError();

```

El sketch realizará las funciones de control de tiempo y realizará llamadas a las funciones de la clase *ProyectoDU*, mediante un objeto de dicha clase.

Un esquema orientativo de la estructura del programa podría ser:



Referencias

Algunas fuentes externas consultadas durante la elaboración del proyecto:

1. <http://panamahitek.com/instalacion-de-php-arduino-yun>
2. <http://foro.seguridadwireless.net/openwrt/envio-de-correos-electronicos-desde-openwrt>
3. <https://karlduino.wordpress.com/2014/01/14/set-clock-on-arduino-yun>
4. <https://www.w3schools.com>

Fe de Errores

El proyecto tiene incompletas algunas funcionalidades debido a la falta de tiempo para completar el proyecto.

Estas limitaciones pueden ser:

- El envío del E-Mail de error no está parametrizado, por lo que no se puede saber la causa del fallo de lectura.
- Se han desactivado todas las funciones de depuración, y por tanto el monitor serie no se usa debido a que lo considero innecesario, ya que se ejecuta mediante WiFi.
- El envío de errores al topic 'Log' sólo se hace cuando se envía el E-Mail, por lo que los mensajes que llegarán a dicho topic serán escasos.

Otras funcionalidades que me hubiese gustado añadir serían:

- Publicación de los umbrales mediante el Cliente MQTT, creando para ello 4 nuevos canales, y que se actualizasen los valores del sketch cada vez que se publique
- Funciones de scroll de la pantalla o de uso de algunos botones, para, por ejemplo, mostrar la fecha y hora.