Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Internet de las cosas. Sistema electrónico de control basado en Arduino

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Irigoyen Gallego, Rodrigo

Tutor: Busquets Mataix, José Vicente

Curso 2017-2018

Resumen

Este proyecto busca desarrollar un sistema de control domótico, es decir, una estructura IoT basada en gestionar y controlar remotamente los elementos de un hogar, tales como: luces, temperatura, electrodomésticos, etc. Vamos a desarrollar un sistema completo, empezando por implementar los mecanismos para obtener la información, continuando por crear un servidor para almacenarla y para finalizar desarrollar las herramientas necesarias para interactuar con él.

Basándonos en tecnologías como NodeMCU gestionaremos sensores, actuadores y desarrollaremos una comunicación con una Raspberry Pi. Esta será la que ejerza la función de servidor a través del servidor web Apache y montará una base de datos MySQL. La interactuación con el sistema se llevará a cabo remotamente a través de dos herramientas independientes entre ellas: una aplicación móvil para sistemas Android y una página web alojada en nuestro servidor.

Con estos elementos creamos una estructura autosuficiente que nos permitirá desde cualquier lugar ver el estado del hogar e interactuar con él en tiempo real gracias a las soluciones implementadas.

Palabras clave: Raspberry, sensor, actuador, domótica, comunicación, IoT, BD, MySQL, NodeMCU, servidor, remoto, Android, tiempo real, página web.

Abstract

This project aims to develop a complete system of home automation. A home automation is an IoT structure based in the management and remote control of home elements as: lights, temperature, electrical appliances, etc.

We work with technologies as NodeMCU to control sensors and actuators to aim the communication with a Raspberry Pi that works as a server through the web server Apache and making a MySQL database. The interaction with the system will work remotely through two different tools that are independent between them: A Smartphone application for Android systems and a web page located in our server.

With all these elements we create a self-working structure that allows us to see which the situation of our home in real time is even if we are not at home. All of this is possible thanks to the information proportioned by implemented solutions.

Keywords: Raspberry, sensor, actuator, NodeMCU, home automation, communication, IoT, server, DB, Android, web page, MySQL, remote, real time.

Tabla de contenidos

1. Introducción

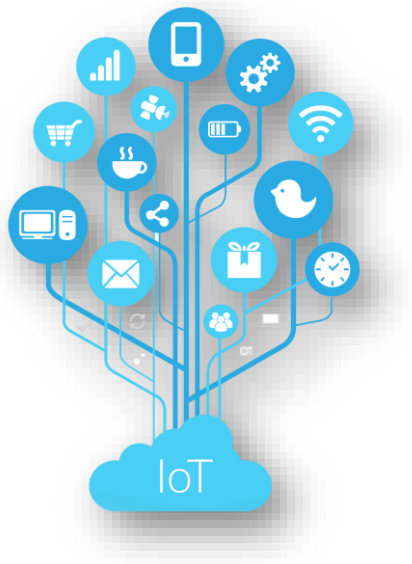

Ilustración 1: Internet de las cosas

Internet de las cosas (IoT) es un concepto que se refiere a la interconexión digital de objetos cotidianos con internet. El objetivo es hacer que todos estos dispositivos se comuniquen entre sí y, por consiguiente, sean más inteligentes e independientes. IoT tiene miles de aplicaciones, en nuestro caso nos vamos a centrar en el campo de la domótica, es decir, sistemas capaces de automatizar una vivienda, aportando servicios de gestión energética, comunicación, seguridad, etc.

La motivación de elegir este proyecto fue por el interés personal en la placa de Arduino y en hacer algo que fuese tangible, trabajando con electrónica y conseguir unir eso a otros segmentos más puramente informáticos. También la posibilidad de un proyecto que pueda continuarse después de terminar el TFG como puede ser en este caso implementándolo en casa.

El proyecto que nos ocupa se centrará en crear todos los componentes que forman un sistema domótico. Los diferentes campos del proyecto se pueden dividir en cuatro grandes grupos: electrónica, servidor, comunicación y dispositivos.

En el campo de la electrónica encontramos todo lo relativo a sensores, actuadores y NodeMCU. Utilizaremos sensores de humedad, temperatura,

luminosidad y actuadores para obtener el estado de la vivienda y hacer alguna modificación, por ejemplo, encender luces. Con la placa NodeMCU gestionamos los sensores y manejamos los actuadores. Y usaremos esta placa para enviar esta información a la base de datos.

El servidor lo vamos a montar sobre una Raspberry Pi 3, que es un computador de placa reducida de bajo coste. Usaremos Apache para montar el servidor en la placa, la base de datos la montaremos con MySQL y la gestionaremos con la herramienta web libre phpMyAdmin para administración de bases de datos.

El tercer campo que tenemos que resolver es la comunicación. Los datos se envían a través de wifi con la placa NodeMCU a la Raspberry y para poder comunicarnos con el servidor (ya que estará en una red privada) nos ayudaremos del servicio DNS No-IP. La comunicación sensores-servidor-Android-página web hará uso del protocolo HTTP que nos permitirá hacer peticiones fácilmente entre “hosts”.

Finalmente hay que poder mostrar el estado de la vivienda e interactuar con esta, para lo cual vamos a crear una aplicación en Android y una página web alojada en el servidor apache para este fin, que se comunicará con la BD.

2. Tecnologías

En este apartado analizamos las tecnologías y componentes de las que haremos uso en nuestro proyecto.

2.1 NodeMCU (ESP8266-12)

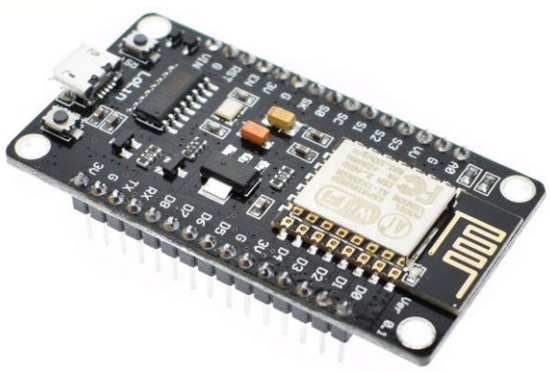

Ilustración 2: Modulo NodeMCU (ESP8266-12)

NodeMCU es una placa de desarrollo totalmente abierta, a nivel de software y de hardware. Al igual que ocurre con Arduino, en NodeMCU todo está dispuesto para facilitar la programación de un microcontrolador o MCU (del inglés “Microcontroller Unit”).

Se compone de un SOC (Sistema en Chip) ESP8266. Básicamente consiste en un chip que tiene todo integrado para funcionar de forma autónoma.

NodeMCU es un módulo formado por un ESP8266-12 con la electrónica necesaria para facilitar su uso, como la alimentación con micro-USB que se conecta al ordenador y con la que podemos transmitir más fácilmente los programas.

Esta placa de desarrollo se integra con el IDE de Arduino permitiéndonos programarlo de la misma manera y con la misma filosofía que este.

La razón de usar NodeMCU en vez de un ESP8266 simple ha sido la facilidad de conexión con el ordenador y la alimentación, siendo con el ESP8266 necesaria para su programación usar comandos AT o usar un conector FTDI para poder conectarlo y subir programas mediante el IDE de Arduino.

Otras opciones para el desarrollo del proyecto eran usar una placa Arduino Mega y usar el ESP8266 como esclavo, algo que al final fue descartado porque el chip wifi es capaz de hacer él las tareas de Arduino y acababa siendo innecesario.

En el mercado podemos encontrar otras opciones como son la Raspberry Zero, Arduino MKR1000. La primera muy potente, con Raspbian de SO, aunque un consumo más elevado y de precio más caro, y el Arduino más parecido al NodeMCU, con wifi, mismo IDE, aunque con un precio más caro, procesador más lento y menos memoria ROM.

Características técnicas del NodeMCU:

CPU	RISC 32 bits 80MHz/160MHz
Alimentación	5V
Pines digitales E/S	16 GPIO
Vel. Reloj	16MHz
Conectividad Wifi	2.4GHz 802.11b/g/n
ROM	4MB
RAM	128Bytes

2.2 Sensores y actuadores

Un sistema domótico se basa en la obtener información del medio y para mostrarla y poder modificarla. Los sensores y actuadores son piezas imprescindibles en los que se basan los sistemas domóticos e IoT.

2.2.1 Módulo de temperatura y humedad (DHT 11)

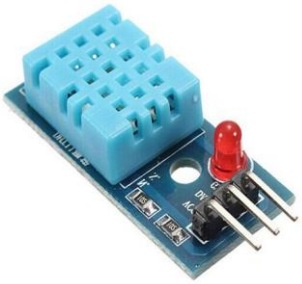

Ilustración 3: Módulo DHT11

El DHT11 es un sensor de temperatura y humedad muy económico y muy usado que proporciona una salida de datos digital. Entre sus ventajas podemos mencionar el bajo coste y el despliegue de datos digitales. Esto supone una gran ventaja frente a los sensores del tipo análogo, como el LM335, por ejemplo, en los cuales las fluctuaciones en el voltaje alteran la lectura de datos.

El DHT11 es un sensor de media precisión (para más precisión usar por ejemplo el DHT22), por lo que solo lee enteros, no obtendremos valores con decimales, por lo que hay que tener en cuenta la precisión que necesitemos en las mediciones a la hora de elegir este chip.

Se compone de 3 pines: VCC (5V), GND y datos y para tratar los datos recibidos en Arduino haremos uso de la librería “DHT sensor library” que nos permitirá tratar los datos de forma sencilla.

2.2.2 Módulo con fotorresistencia (KY-018)

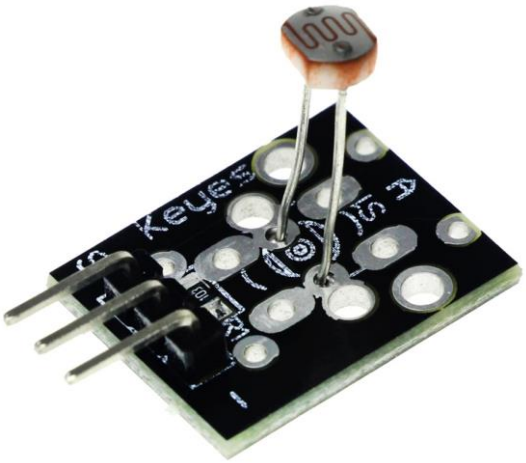

Ilustración 4: Módulo KY-018

El módulo KY-018 tiene una fotorresistencia, la cual es una resistencia variable dependiente de la cantidad de luz en su entorno. En la oscuridad, su resistencia es muy alta, a veces hasta 1MΩ, pero cuando el sensor se expone a la luz, la resistencia se reduce drásticamente, incluso a unos pocos ohm, dependiendo de la luz.

Los valores de su resistencia, sensibilidad, coeficiente de temperatura y su curva de voltaje-corriente dependen directamente de la cantidad de luz que recibe el sensor.

Este sensor es ampliamente utilizado en cámaras, lámparas de jardín y calle, detectores, relojes, luces automáticas y un sinfín de aplicaciones.

Se compone de 3 pines: VCC (5V), GND, datos que envía una señal analógica que es un numero entero que indica la intensidad de la luz.

2.2.3 Led + Botón

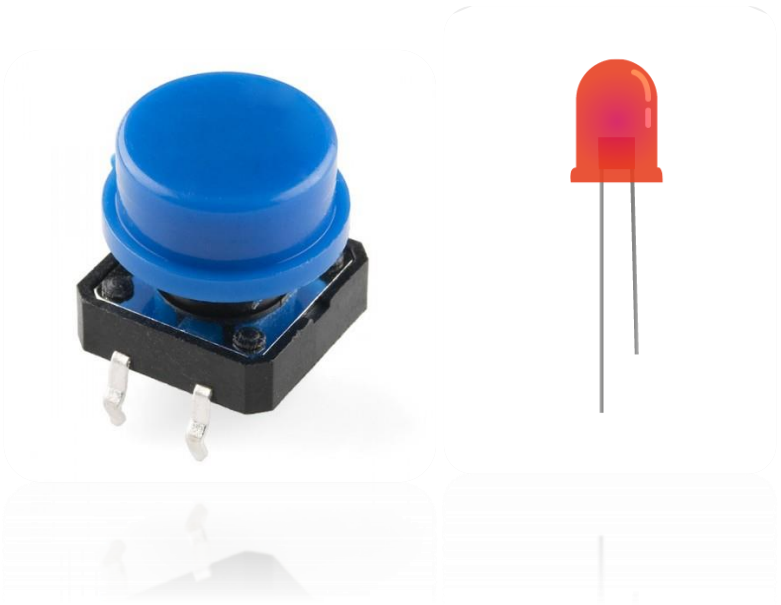

Ilustración 5: Botón (izq.) + led (der.)

Un led es un diodo emisor de luz de unión p-n. Si se aplica una tensión adecuada a los terminales, los electrones se recombinan con los huecos en la región de la unión p-n del dispositivo, liberando energía en forma de fotones. Este efecto se denomina electroluminiscencia, y el color de la luz generada viene determinado por la anchura de la banda prohibida del semiconductor.

Un botón o pulsador en Arduino es un interruptor, al pulsarlo hace contacto y permite pasar la corriente de un lado al otro. Luego controlaremos los estados en reposo (cuando no está pulsado) con una resistencia “pull-down” para poner en reposo el estado en “LOW”.

Con estos dos elementos vamos a crear un montaje que nos permitirá encender y apagar un led manualmente (simulando con esto el funcionamiento de un interruptor de una luz de la vivienda). Esto se verá representado en el sistema domótico y podremos modificarlo también remotamente, ejemplificando así el control dual (remoto y presencial) de un elemento de un sistema domótico.

2.3 Raspberry Pi

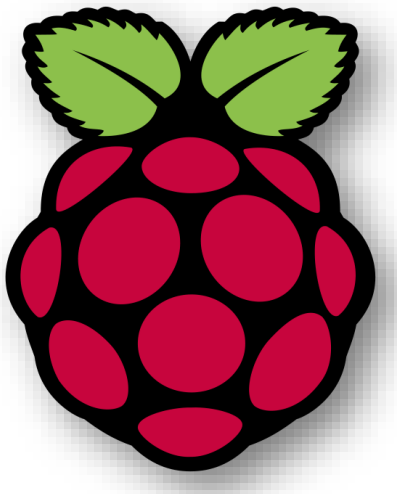

Ilustración 6: Raspberry Pi logo

Raspberry Pi es un computador de placa reducida (SBC) de bajo costo desarrollado en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas. Es un dispositivo muy versátil (pines E/S, Bluetooth, wifi, ethernet, HDMI, USB…), a un precio muy asequible (<40€), un tamaño muy reducido (85.60mm × 53.98mm) y una gran potencia para su tamaño. Puede usarse como un PC con su propio sistema operativo, interfaz gráfica y programas.

Se ha elegido Raspberry para el proyecto entre otras cosas por su bajo consumo, su bajo precio y su gran potencia y transporte en un tamaño reducido. A esto hay que sumarle la gran comunidad detrás del proyecto y cantidad de información disponible para desarrollar en ella.

Hay otras alternativas en el mercado como Orange Pi, muy parecida y que puede usar hasta las mismas imágenes oficiales de Raspberry y otras más potentes como ODROID-XU4 con procesador de 8 núcleos y 2 GB de RAM, aunque su precio también es bastante mayor llegando a los 125€.

Raspberry no es hardware libre, es un producto con propiedad registrada, manteniendo el control de la plataforma, pero permitiendo su uso libre tanto a nivel educativo como particular. A nivel software si es código abierto y de sistema operativo utiliza una versión adaptada de Debian: Raspbian, aunque permite usar otros sistemas operativos, incluido una versión de Windows 10.

Se va a montar en la Raspberry un servidor con la ayuda del servidor HTTP Apache. El servidor se encargará de albergar la base de datos y gestionar las peticiones de la aplicación Android y de la placa NodeMCU.

Como el servidor estará alojado dentro de una red privada tendremos que hacer uso del servicio DNS No-IP para poder redireccionar las peticiones a nuestro servidor.

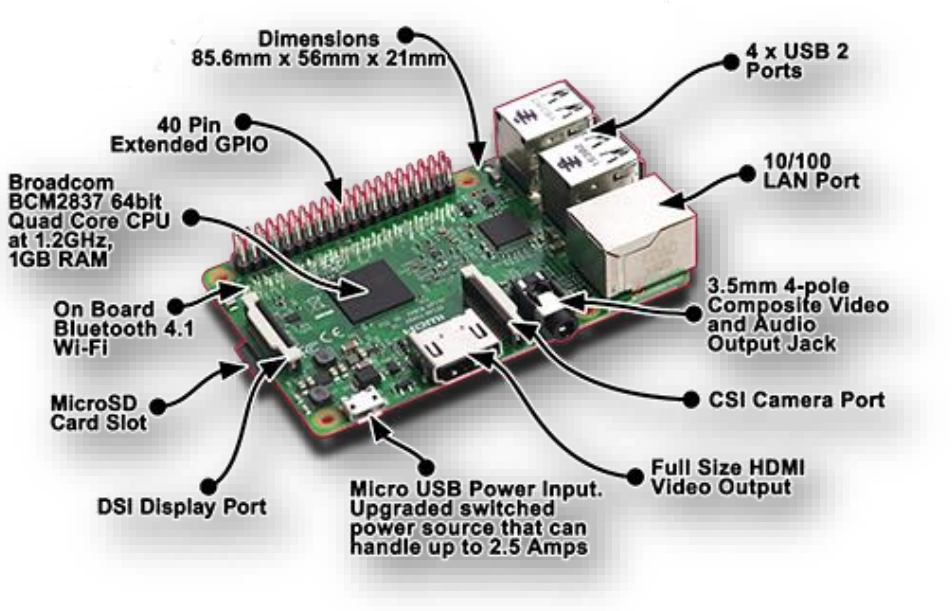

Ilustración 7: Raspberry Pi 3 B

Especificaciones técnicas:

CPU	1.2GHz 64-bit quad-core ARMv8
Juego instrucciones	RISC 32 bits
Alimentación	5V
Memoria SDRAM	1GB
Conectividad Red	Wifi 802.11n, Bluetooth, Ethernet
Almacenamiento	Con MicroSD
SO	GNU/Linux
Pines E/S	17 GPIO
Conexiones	HDMI, 4 USB 2.0,

2.3.1 Base de Datos

Ilustración 8: MySQL logo

MySQL es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual GPL/Licencia comercial por Oracle y está considerada como la base datos código libre más popular del mundo, y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web.

Al contrario de proyectos como Apache, donde el software es desarrollado por una comunidad pública y los derechos de autor del código están en poder del autor individual, MySQL es patrocinado por una empresa privada, que posee el copyright de la mayor parte del código.

2.3.1.1 phpMyAdmin

Ilustración 9: phpMyAdmin logo

Esta herramienta escrita en PHP con la intención de manejar la administración de MySQL a través de páginas web, utilizando Internet. Actualmente puede

crear y eliminar bases de datos, crear, eliminar y alterar tablas, borrar, editar y añadir campos, ejecutar cualquier sentencia SQL, administrar claves en campos, administrar privilegios, exportar datos en varios formatos y está disponible en 72 idiomas. Se encuentra disponible bajo la licencia GPL Versión 2.

Usaremos phpMyAdmin para facilitar la gestión de la base de datos, aunque no es imprescindible, pudiendo gestionar la BD directamente desde terminal creando tablas, columnas, etc.

2.3.2 Servidor Apache

Ilustración 10: Apache logo

El servidor HTTP Apache es un servidor web HTTP de código abierto, para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.12 y la noción de sitio virtual.

Es el servidor HTTP más usado. Jugó un papel fundamental en el desarrollo de la “World Wide Web” y alcanzó su máxima cuota de mercado en 2005 siendo el servidor empleado en el 70% de los sitios web en el mundo

Nos permitirá convertir nuestra Raspberry en un servidor, que pueda responder a peticiones HTTP y que albergará la página web con la que interactuaremos con el sistema.

2.4 Android

Ilustración 11: Android logo

Android es un sistema operativo basado en el núcleo Linux. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes, tabletas y también para relojes inteligentes, televisores y automóviles.

Actualmente el sistema operativo para móviles más usado a nivel mundial y con el 93.9% de cuota de mercado español en 2016 según un análisis de *"*Kantar Wordlpanel*"*.

Vamos a diseñar una aplicación Android en la que poder mostrar todos los datos provenientes del funcionamiento del sistema domótico y donde podamos interactuar con él. Podremos apagar una luz, mostrar temperatura, etc. Para su desarrollo usaremos el IDE de AndroidStudio que nos facilitara su programación.

2.5 Comunicación

Ilustración 12: Comunicación

En el apartado de la comunicación vamos a englobar aquellos aspectos que toman parte en proporcionar los mecanismos necesarios para que se intercambien datos entre los diferentes sistemas: Arduino, Raspberry, Android.

2.5.1 Servicio DNS No-IP

No-IP es una compañía que ofrece un servicio de DNS dinámica para servicios de pago y gratuitos. La función más usada del sistema de nombres de dominio (DNS) es la de relacionar una dirección IP a un nombre, por ejemplo, 216.58.210.163 al nombre de www.google.es.

El problema que se presenta y se tiene que solventar es que el servidor estará ubicado en dentro de una red privada, por lo cual no se puede acceder a él. Para ello se hará uso de No-IP. Hay que registrar la dirección pública del rúter y el puerto por el cual accederá y luego abrimos ese puerto en el rúter (“*port forwarding”*). El servicio permitirá elegir un nombre de dominio para acceder. A partir de aquí cada vez que queramos se acceda al servidor se usará el nombre del dominio proporcionado y este nos redirigirá.

2.5.2 PHP y HTTP

Ilustración 13: Icono HTTP y PHP

El envío de datos entre los diferentes nodos del sistema (Raspberry, Web, NodeMCU) se realizará utilizando HTTP y PHP.

HTTP es el protocolo de comunicación que permite las transferencias de información en la red. HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxis) para comunicarse. HTTP es un protocolo sin estado, es decir, no guarda ninguna información sobre conexiones anteriores. Para esto se usan las cookies, que es información que un servidor puede almacenar en el sistema cliente

PHP es un lenguaje de código abierto muy popular especialmente adecuado para el desarrollo web y que puede ser incrustado en HTML. PHP está centrado en la programación de scripts del lado del servidor,

Lo que distingue a PHP de algo del lado del cliente como JavaScript es que el código es ejecutado en el servidor, generando HTML y enviándolo al cliente. El cliente recibirá el resultado de ejecutar el script, aunque no se sabrá el código subyacente que era.

Se va a utilizar PHP para acceder y hacer las consultas a la BD en el servidor y desde la página web y NodeMCU transmitirá los datos usando HTTP a la Raspberry3

2.6 Herramientas de ayuda al desarrollo

Se va a usar la Raspberry Pi como servidor, ello implica estar mucho tiempo encendida por lo que el consumo es muy importante. Por esa razón no vamos a conectar periféricos como pantalla, teclado ni ratón, reduciendo así el consumo de este.

Para interactuar con ella entonces necesitaremos conectarnos remotamente, para lo que nos ayudaremos del protocolo SSH. De esta forma instalaremos, configuraremos y nos conectaremos en general con el servidor.

La Raspberry viene con SSH instalado, pero hay que ir a configuración y activarlo. Una vez tenemos el servidor SSH instalado tenemos que instalar el cliente en el ordenador desde el que queramos conectarnos:

Desde dispositivos Linux:

```
// Instalación

$ sudo apt-get install ssh

// Conexión con el servidor por (por defecto puerto 22)

$ sudo ssh pi@rodrigodomotica.ddns.net

// nos pedirá la contraseña y ya estaremos conectados
```

Desde dispositivos Windows:

En los ordenadores con Windows para llevar a cabo la conexión SSH hemos hecho uso de la herramienta MobaXterm que nos permite abrir varias sesiones y otras utilidades. También tendrás que introducir nombre, contraseña y dirección.

3. Desarrollo (implementación)

3.1 Estudio del proyecto

El sistema muestra tres partes diferenciables: electrónica (con toda la parte de sensores, actuadores y chip wifi), servidor (formado por Apache, MySQL y la Raspberry) y los dispositivos (Android y página web).

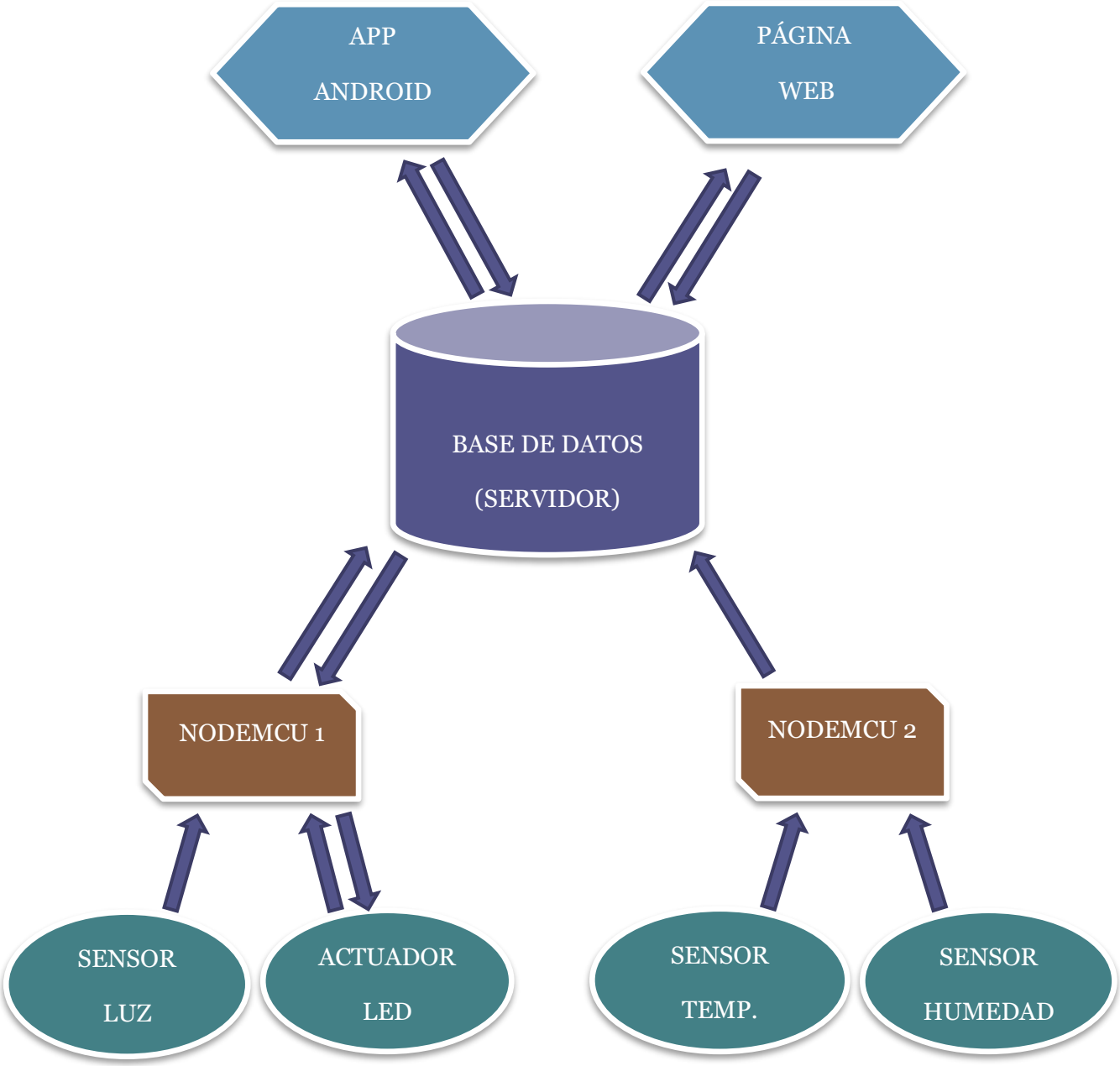

Ilustración 14: Esquema de la estructura del sistema

En el esquema superior podemos ver la estructura del proyecto y el flujo de datos que se produce entre los diferentes elementos del sistema.

Tenemos un servidor que hace de nodo central que funciona de puente entre los dos lados. No hay comunicación directa entre las placas y los sensores y la aplicación Android ni la página web. Esta independencia entre componentes hace el sistema más escalable facilitando el añadir nuevos dispositivos, ya que no habría que reprogramar ningún NodeMCU, sensor o actuador.

En base a este razonamiento vamos a exponer la implementación del sistema dividiéndolo en tres temas: los NodeMCU y sensores primero, el servidor en segundo lugar, y finalizaremos con los dispositivos de interactuación con el sistema.

3.2 Implementación

3.2.1 Electrónica

La parte electrónica, formada por los sensores y actuadores y el NodeMCU que obtienen los datos del sistema los transmiten y ejecutan sus órdenes.

Los principales componentes hardware que montamos son un sensor DHT11 de temperatura y humedad, un sensor KY-018 de iluminación, un botón, un led, y dos NodeMCU.

Para controlar los componentes vamos a usar los NodeMCU. Estos gracias a un “plugin” se puede programar como si fuera un Arduino y usando su IDE. Como los NodeMCU se conectan al ordenador por micro USB no requieren de conector “FTDI” para cargar los programas.

Para poder usar el IDE de Arduino con los NodeMCU necesitamos primero configurarlo adecuadamente. Para ello debemos ir a “Archivo/Preferencias/gestor URLs Adicionales de Tarjetas” y ahí poner la URL de la placa que queremos cargar (la URL la encontramos en https://github.com/esp8266/Arduino).

Con esto ya tenemos la placa en el IDE, ahora cuando queramos compilar y cargar el programa tenemos que elegir la configuración correspondiente a esta placa. Para ello vamos a “Herramientas/Placa/Gestor de tarjeras” y elegimos la placa NodeMCU 1.0 (ESP 12E module).

Con esto ya estamos preparados para programar nuestros NodeMCU.

Los siguientes apartados se pueden dividir en dos partes diferenciadas, por un lado, la conexión de los componentes electrónicos que vamos a usar y por otro la programación de estos con el IDE de Arduino.

3.2.1.1 Modulo DHT11

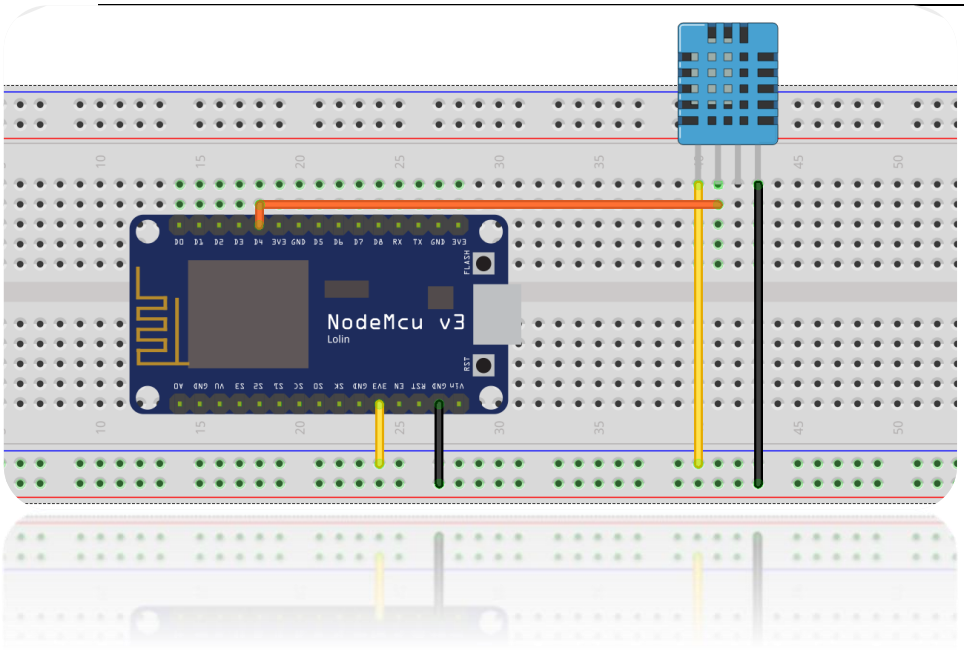

Ilustración 15: Imagen de la conexión del DHT11 con NodeMCU

Conexión:

Como ya hemos expuesto antes, el DHT11 es un sensor de temperatura y humedad de precisión media. El que tenemos nosotros es la versión del DHT11 con PCB que tiene 3 pines y trae ya una resistencia “pull-up” de 5kΩ integrada.

La conexión la haremos como se puede ver en la ilustración superior, el pin de la derecha se conectará GND, el de la izquierda se conecta a VCC de 3,3V para alimentarse y el pin central, que es el que proporciona los datos, se conecta a un pin digital configurado como “INPUT” en el NodeMCU.

Programación:

Empezamos cargando la librería DHT necesaria (Adafruit v1.2.3). Hay otra versión más nueva no funciona correctamente. El siguiente paso es indicar cuál será el pin del que tiene que leer los datos del sensor, en este caso el 2 (también se puede identificar como D4) y se carga DHT con ese pin.

```
#include <DHT.h>

int pinDht=2;
DHT dht(pinDht,DHT11);
```

El proceso de coger la información y mostrarla se hace en el método "getDataFromSensor". En el leeremos la temperatura y la humedad, comprobaremos que no recibimos valores fuera de lo esperado debido a algún fallo de lectura del sensor (cuando se da algún problema de estos se reciben valores exageradamente grandes y como la temperatura y humedad no son parámetros con subidas o bajadas drásticas es fácil establecer unos valores esperados).

```
void getDataFromSensor()
{
    int auxTemp= dht.readTemperature();
    int auxHum=dht.readHumidity();

    //CONTROL DE VALORES ERRONEOS
    if((abs(auxTemp) < abs(temp)*5) && (abs(auxHum) < abs(hum)*5))
    {
      temp=auxTemp;
      hum=auxHum;
    }
}
```

En el método "setup" iniciamos "dht" y el serial (para mostrar la información por pantalla).

```
void setup()
{
   dht.begin();
   Serial.begin(9600);
}
```

En el método “loop” que se repite constantemente lo que hacemos es simplemente llamar al método “getDataFromSensor”. Le hemos introducido un “delay” para que coja datos cada 30 segundos ya que estamos hablando de valores que cambian poco con el tiempo y lo hacen lentamente por lo que estar constantemente consultando el estado solo significa un consumo extra por parte del sistema y no tendríamos ventajas en el funcionamiento general de este.

```
void loop()
{
   //Coge informacion del sensor
   getDataFromSensor();

   //Lee dos veces por minuto
   delay(30000);
}
```

3.2.1.1 Modulo KY-018

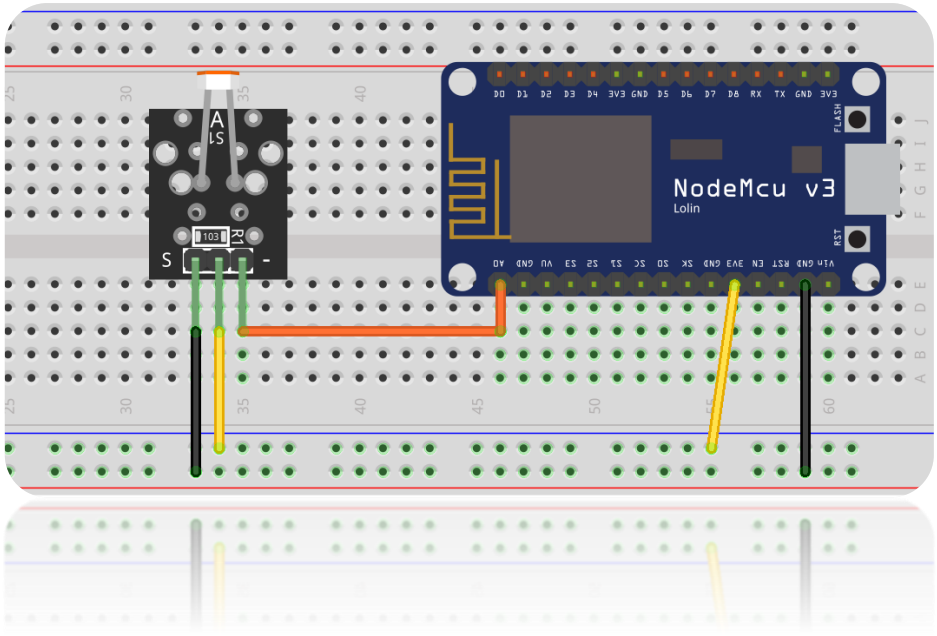

Ilustración 16: Imagen de la conexión del KY-18

Conexión:

El siguiente componente que vamos a conectar es el sensor KY-18. Este sensor lumínico al igual que el DHT11 tiene 3 pines que conectaremos de la siguiente manera: el de la izquierda va conectado al GND, el central se conecta a un VCC de 3,3V y el de la derecha es el pin de datos que conectaremos a la placa.

La salida de este sensor es analógica por lo que hay que conectarla al pin A0 del NodeMCU que es el único pin analógico de la placa.

Programación:

Empezamos definiendo el pin por el que recibirá los datos la placa. En este caso al ser analógica la entrada solo puede ser por el pin A0 ya que no tiene más entradas analógicas.

```
int pinLuz=A0;//sensor luz
```

Para recoger los datos hemos creado un método “getDataFromSensor” que se encarga de hacer la lectura de los datos.

```
void getDataFromSensor()
{
    luz=analogRead(pinLuz);
    Serial.print("  Luz:  ");
    Serial.print(luz,DEC);
 }
```

En el “loop” del programa solo necesitamos la llamada a este método. A diferencia del lector de temperatura y humedad, en este caso los valores cambian totalmente de un momento a otro, por lo que es necesario para poder reflejar el estado real del sistema que se hagan comprobaciones continuamente. Por esa razón en este “loop” no tenemos un “delay”.

```
void loop()
{
   //Coge informacion del sensor
   getDataFromSensor();
}
```

3.2.1.1 Led y Botón

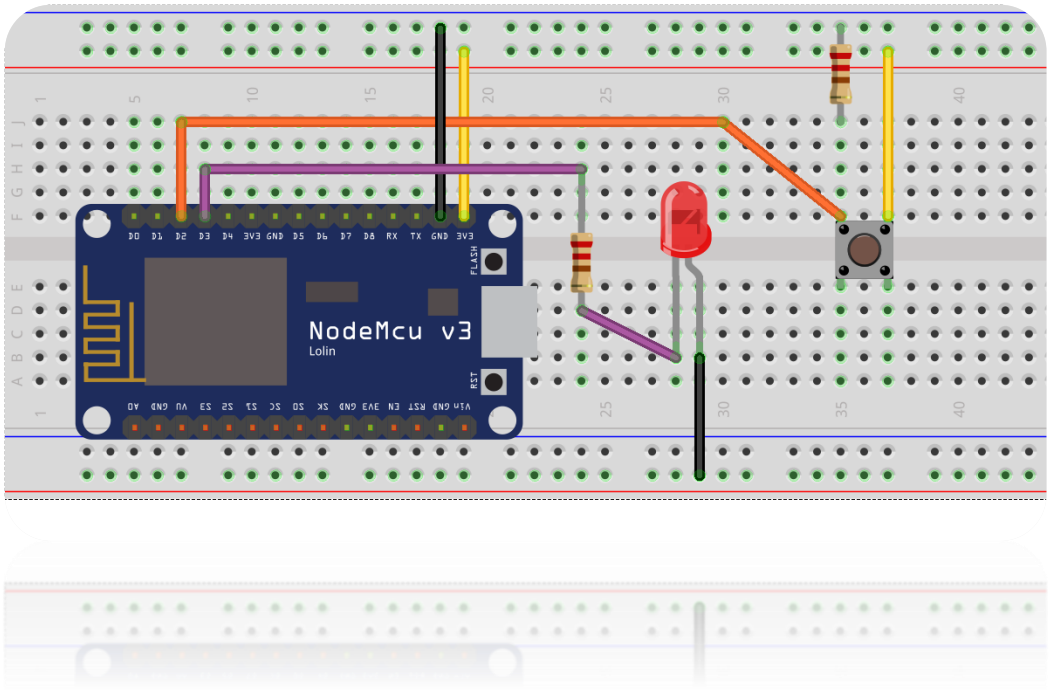

Ilustración 17: Imagen de la conexión del led y el botón

Conexión:

En este caso vamos a montar un led que se enciende y apaga apretando un botón.

El led dispone de dos patas, el ánodo y el cátodo. El cátodo está representado por la pata más corta y esta es la que se conecta a tierra, mientras que el ánodo es por la que entra la corriente. La diferencia de potencial que requieren los leds varía en función del color del led. El led amarillo requiere una diferencia de potencial de unos 2V. Para que no se queme el led hay que usar una resistencia entre la placa y el led, así que le hemos puesto una de 220 ohmios.

El botón es un interruptor que al presionarlo deja pasar la corriente y conectando eso a la placa podemos saber que se ha presionado. El botón tiene dos patas, una la conectamos a VCC y la otra la conectamos a la placa que recibirá la señal. Le hemos montado una resistencia “pull-down” a la pata que conectada a la placa para que mientras este en reposo (no pulsado) la entrada lógica sea cero (LOW).

Programación:

El programa funciona mediante interrupciones hardware. En vez de estar en el "loop" controlando si ha sido pulsado o no (con el retardo que eso conllevaría ya que tiene que ejecutar todo el código hasta que vuelva a comprobarlo) establecemos una interrupción hardware que cuando sea pulsado el botón parara la ejecución del programa y ejecutara el método que crearemos para gestionar este evento. Una vez ejecutado la interrupción vuelve la ejecución del código donde estaba.

El primer paso para establecer una interrupción hardware es configurarla. Las interrupciones funcionan indicando el pin en el que tiene que escuchar el cambio de estado, el método que llamara en caso de ejecutarse y el comportamiento del estado (si pasa de "HIGH" a "LOW", o de "LOW" a "HIGH", o mientras se mantenga en "LOW" …). En el "setup" establecemos un "attachInterrupt" indicando el pin del que tiene que escuchar el estado (el pin del botón), y en nuestro caso cuando pase de "LOW" a "HIGH" queremos que se lance el método por lo que lo pondremos en "RISING".

```
void setup()
{
   pinMode(led, OUTPUT);
   pinMode(button, INPUT);
   attachInterrupt(button, manejadorLed, RISING);
}
```

Una vez lanzada la interrupción se ejecuta el método "manejadorLed" que leerá el estado del led, y escribirá el estado contrario al que tuviera.

El botón tiene un comportamiento errático llamado "rebote" que lo que hace es que en vez de una pulsación del botón leer varias pulsaciones seguidas produciendo un comportamiento incorrecto. Para evitar esto hemos definido un intervalo mínimo de tiempo que tiene que pasar para poder volver a pulsarse el botón. Se ha establecido 250 milisegundos como el tiempo mínimo entre dos pulsaciones ya que una persona real no puede pulsar más rápido que eso y así evitamos las pulsaciones incorrectas del botón.

```
void manejadorLed(){
  if(millis()>t+250)
  {
    estado=digitalRead(led);
    if(estado==LOW)
    {
      digitalWrite(led,HIGH);
    }
    else
```

```
  {
    digitalWrite(led,LOW);
  }
 }
 t=millis();
}
```

3.2.1.1 Comunicación Wifi

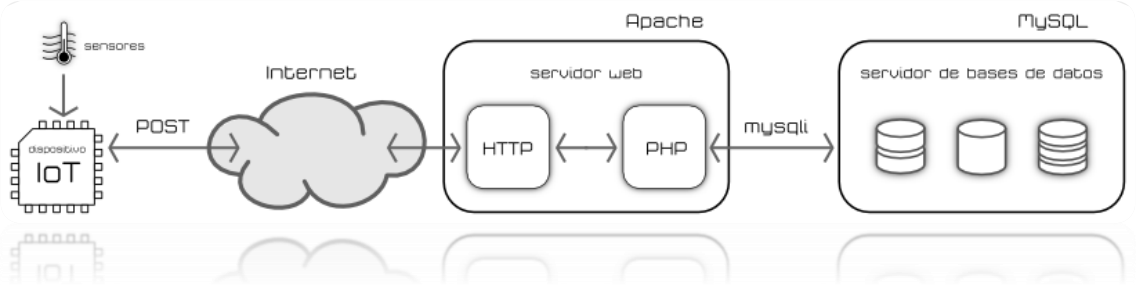

Ilustración 18: Conexión de los sensores con la base de datos

Una vez que ya tenemos implementado todos los sensores y actuadores tenemos que transmitir los datos obtenidos al servidor montado en la Raspberry.

Nos vamos a encontrar con dos situaciones diferentes: en un caso tendremos un comportamiento de cliente con los sensores de temperatura, humedad, luz y led que solo van a transmitir datos al servidor. Y en el caso del led funcionaremos también como servidor, ya que necesitamos recibir los cambios y mostrarlos.

Como Cliente:

Para poder conectarnos con el servidor y poder enviar datos primero necesitaremos conectarnos a una red wifi a la que tengamos acceso. Una vez hecho esto ya podremos comunicarnos mediante peticiones HTTP con el servidor.

Para conectarnos a la red wifi y realizar las peticiones HTTP vamos a servirnos de una librería disponible para ESP8266 que es “ESP8266WiFi” que nos proporcionara las herramientas necesarias para realizar nuestro trabajo.

Para llevar a cabo la conexión vamos a empezar por definir las variables para conectarnos al wifi y el nombre del host donde está el servidor.

```
#include <ESP8266WiFi.h>

const char* ssid = "MOVISTAR_3089";
const char* password = "***************";
const char* host = "rodrigodomotica.ddns.net";
```

En el “setup” vamos a conectarnos al wifi haciendo uso de las variables definidas antes y esperaremos hasta que este realizada la conexión para empezar el “loop”.

```
void setup()
{
   WiFi.begin(ssid, password);
   while (WiFi.status() != WL_CONNECTED) {
   delay(500);
   Serial.print(".");
   }
}
```

En el método “loop” crearemos un cliente y abriremos una conexión TCP. Una vez hecho esto crearemos la URL de la petición poniendo las variables y sus valores. Le vamos a pasar una variable “pass” con la contraseña necesaria para poder ejecutar el PHP y así hacerlo un poco más seguro.

```
void loop()
{
   // Usamos WiFiClient para crear conexiones TCP
   WiFiClient client;
   const int httpPort = 80;
   if (!client.connect(host, httpPort)) {
   Serial.println("connection failed");
   return;
   }

   //Coge informacion del sensor y lo muestra por serial
   getDataFromSensor();

  // Creamos la url para la peticion
  String url = "/subirToDB.php";
  String key = "?pass=1234";
  String dato1 = "&Temperatura=";
  String dato2 = "&Humedad=";
```

Por último, realizaremos el “GET” y esperaremos la respuesta del servidor.

```
 // Esto manda la petición al servidor
  client.print(String("GET ") + url + key + dato1 + temp + dato2 + hum +
dato3 + luz
               +" HTTP/1.1\r\n" + "Host: " + host
               + "\r\n" + "Connection: close\r\n\r\n");
  unsigned long timeout = millis();
  while (client.available() == 0) {
    if (millis() - timeout > 5000) {
      Serial.println(">>> Client Timeout !");
      client.stop();
      return;
     }
  }

  // Lee todas las líneas de la respuesta
  while (client.available()) {
  String line = client.readStringUntil('\r');
  Serial.print(line);
  }
}
```

El “GET” lo que va a hacer es lanzar el script PHP “subirToDB.php” (esto lo hace el NodeMCU del DHT11, el otro llama al script “subirToDBLuz”) y pasarlo los valores de las variables para que este sea el que se conecte con la base de datos y la actualice.

En el PHP “subirToDB.php” con “$_GET” cogemos los valores transmitidos en la URL, formamos la petición a la base de datos y realizamos la petición insertándolos de esta forma en la tabla adecuada. Esto esta explicado más extensamente en el apartado que explica la comunicación de la página web con el servidor.

```
<?php
$password="1234";
if( $_GET['pass'] == $password)
{
  require("parametrosConfDB.php");

  //Conexion con la base de datos
  $con = mysqli_connect($dbhost, $dbuser, $dbpass, $dbname);

  //obtenemos los datos de la url y con el formato para query apropiado
  $Temperatura = mysqli_real_escape_string($con, $_GET['Temperatura']);
  $Humedad = mysqli_real_escape_string($con, $_GET['Humedad']);

  // E insertamos los valores en la tabla
```

```
  $query = "INSERT INTO valores(Temperatura, Humedad)
            VALUES('$Temperatura','$Humedad')";

  // Ejecutamos la instruccion
  mysqli_query($con, $query);
  mysqli_close($con);
}
else
{
  echo "Acceso bloqueado. Necesitas la contraseña para acceder a la base
de datos.";
}
?>
```

Como Servidor:

Para conectarnos al NodeMCU como servidor primero debemos abrir un puerto libre (nosotros abrimos el 5000) para que se redireccione el tráfico que va dirigido a él a través del rúter como hemos hecho con la Raspberry Pi y se explica más detalladamente en el apartado de NO-IP.

Al igual que para el cliente, lo primero es conectarse a la red wifi y vamos a crear un “WifiServer” y le indicaremos el puerto en el que escuchara las peticiones y lo iniciamos en el “setup”.

```
//Puerto donde oirá el NodeMCU
WiFiServer server(5000);

void setup()
{
  // Start the server
  server.begin();
}
```

El método “getDataClient” va a ser llamado en cada iteración del “loop” para comprobar si hay algún cliente conectado, y si lo está, recibir los datos que este le transmita.

```
int getDataClient()
{
    // Comprueba si hay algún cliente conectado
  WiFiClient client = server.available();
  if (!client) {
    return -1;
  }

  // Espera hasta que el cliente envíe algún dato
  Serial.println("new client");
  while(!client.available()){
```

```
    delay(1);
  }

  // Lee la primera línea de la petición
  String req = client.readStringUntil('\r');
  client.flush();
```

Ahora comprobamos la URL y si acaba en “/gpio/0” o “/gpio/1”. Esto será lo que nos indique si nos están mandando apagar o encender el led ya que no estamos mandando variables en la URL para controlar el led, sino que usamos solo la dirección de la petición.

```
  // comprobamos coincidencia de la peticion
  int val;
  if (req.indexOf("/gpio/0") != -1)
    val = 0;
  else if (req.indexOf("/gpio/1") != -1)
    val = 1;
  else {
    Serial.println("invalid request");
    client.stop();
    return -1;
  }
  client.flush();
  return val;
}
```

Ya finalmente cada iteración del “loop” llamamos a la función que acabamos de explicar, y si nos devuelve un valor 0 ponemos la salida del pin que está conectado al led a “LED” y si es 1 lo ponemos a “HIGH”.

```
void loop()
{

   //int estadoLed = digitalRead(led);
   int clientLed = getDataClient();
   if(clientLed != estado && clientLed != -1)
   {
    digitalWrite(led, clientLed);
    estado = clientLed;
   }
}
```

3.2.2 Servidor

El servidor, montado sobre una Raspberry Pi 3B va a ser el eje sobre el que giren los demás componentes del proyecto. Reciben datos del NodeMCU, de la página web y de la aplicación Android y proporciona datos a la página web y la aplicación.

Sobre la implementación del servidor se va a separar en dos partes, el montaje de los componentes y la configuración (LAMP) y el funcionamiento propiamente dicho que este se verá en el apartado de comunicación.

3.2.2.1 Configurar NO-IP

Como el servidor está alojado en una red privada no puede accederse desde fuera de la red por lo que no funcionaría desde fuera de casa. Para ello lo que hay que hacer es hacer redirección de puertos (“port forwarding”), es decir, accedemos al rúter de la red en la que está el servidor y abrimos uno de los puertos que estén libres y le indicamos que todo el tráfico que valla a ese puerto lo envíe a la dirección IP de nuestro servidor. De esta forma si te conectas a la dirección pública del rúter con el puerto indicado te enviara a nuestro servidor.

Para acceder a nuestro rúter ponemos su dirección en el navegador: 192.168.1.1 y así accedemos.

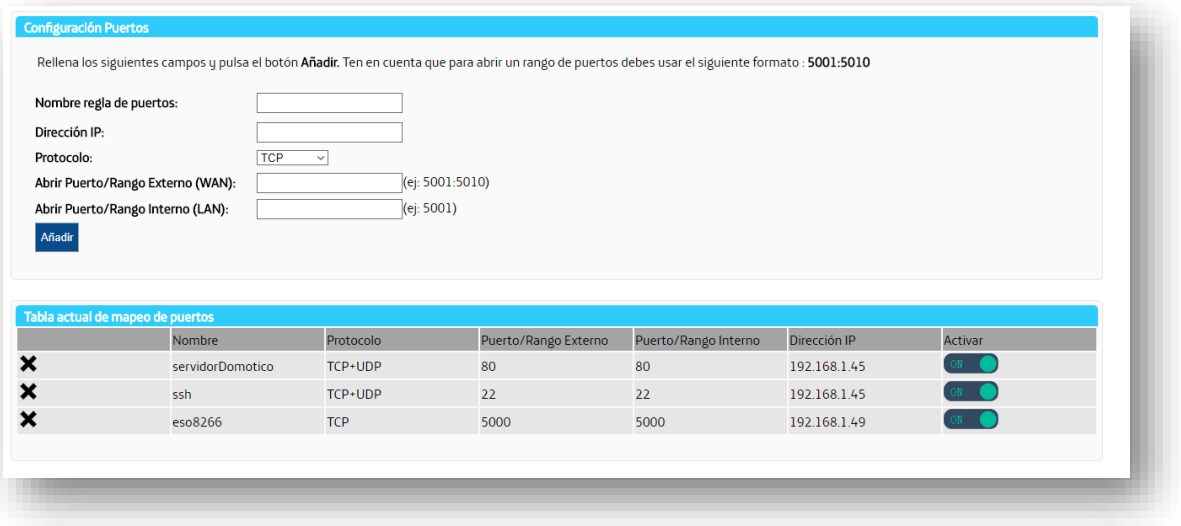

Ilustración 19: Imagen de los puertos del router

Como se observa en la captura los puertos 80, 22, 5000 están abiertos para el servidor, la conexión SSH y la conexión con el ESP8266 respectivamente.

Para facilitar el acceso al servidor queremos poder acceder a él a través de un nombre de dominio y no tener que introducir la dirección IP y el puerto, además de que controle automáticamente si cambia la dirección IP del servidor y no tener que modificar la IP manualmente usaremos un servicio NO-IP dinámico.

En este registramos la dirección IP publica de nuestro rúter y el puerto asignado (en el “port forwarding” anteriormente realizado) a nuestro servidor e indicamos el nombre de dominio: rodrigodomotica.ddns.net. Ahora instalamos el cliente del servicio NO-IP en la Raspberry Pi que se encargara de gestionar los cambios de IP del servidor y actualizarse.

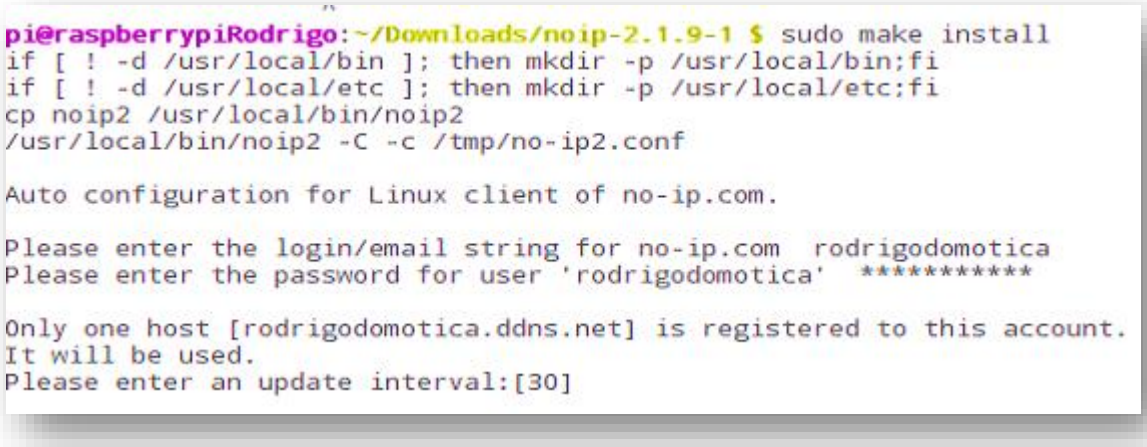

Ilustración 20: Imagen de la configuración NO-IP

Con esta configuración ya se puede acceder al servidor escribiendo en el navegador: rodrigodomotica.ddns.net

3.2.2.2 Configurar LAMP

LAMP es el acrónimo usado para describir un sistema de infraestructura de internet que usa las siguientes herramientas:

-Linux, en nuestro caso Raspbian que es un sistema operativo ligero basado en Debian para Raspberry Pi.

-Apache, el servicio web.

-MySQL (también MariaDB), es el gestor de bases de datos.

-PHP (también Python o Perl), son los lenguajes de programación.

La combinación de estas tecnologías es usada principalmente para definir la infraestructura de un servidor web, utilizando un paradigma de programación para el desarrollo.

A pesar de que en origen estos programas de código abierto no han sido específicamente diseñado para trabajar entre sí, la combinación se popularizó debido a su bajo coste de adquisición y ubicuidad de sus componentes

Empezaremos instalando el servidor Apache: Lo primero siempre actualizar repositorios e instalar actualizaciones.

```
$ sudo apt-get update

$ sudo apt-get upgrade

$ sudo apt-get install apache2
```

Si el servidor web se ha instalado correctamente al acceder a rodrigodomotica.ddns.net te saldrá una página diciendo “It Works”.

El siguiente paso es instala PHP5 y cURL (para peticiones HTTP):

```
$ sudo apt-get install php5 libapache2-mod-php5

$ sudo apt-get install php5-curl
```

Finalmente instalaremos la base de datos MySQL y phpMyAdmin:

```
$ sudo apt-get install php5-mysql mysql-server mysql-client

//Te pedirá que determines la contraseña raíz para MySQL

$ sudo apt-get install phpMyAdmin

// Te pedirá que pongas una contraseña (tiene que ser la misma // que la de MySQL
```

Ahora hay que vincular phpMyAdmin con el servidor web Apache:

```
$ sudo nano /etc/apache2/apache2.conf
```

Al final de archivo hay que incluir:

```
Include /etc/phpMyAdmin/apache.conf

// reiniciamos Apache
```

Ya tenemos instalado y funcional nuestro sistema LAMP

3.2.2.3 Creación Base de Datos

Para crear la base de datos no lo vamos a hacer directamente en MySQL, sino que vamos a aprovechar como ya hemos comentado, la herramienta phpMyAdmin, con la que crearemos la base de datos, las tablas y su estructura.

Como hemos comentado anteriormente vamos a tener dos módulos NodeMCU que se encargaran de gestionar cada uno unos sensores y actuadores determinados. Como cada uno envía datos independientemente del otro se ha decidido por facilidad de implementación y gestión que cada uno opere sobre una tabla diferente.

Dicho lo anterior vamos a implementar una base de datos con dos tablas, “valores” que se encargara de almacenar los datos del sensor de temperatura y humedad que enviara un NodeMCU. Y “valoresLuz” que almacenará los valores del sensor lumínico y del actuador led enviados por el otro modulo wifi.

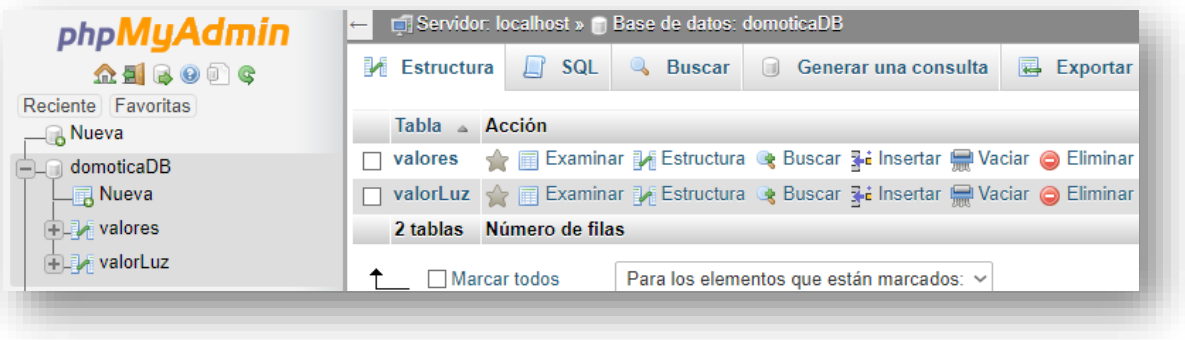

Ilustración 21: Imagen de la base de datos domoticaDB

Tenemos dos tablas creadas, “valores” y “valorLuz” en la que cada NodeMCU va a guardar datos de forma independiente. Ahora hay que crear los campos que van a formar estas tablas.

La tabla “valores” estará compuesta por cuatro campos, los campos “ID”, “Tiempo”, “Temperatura” y “Humedad”.

- ID: Clave primaria se autoincrementará con cada nueva fila insertada. Campo del tipo “int”.
- Tiempo: Guarda la fecha y hora del momento en el que se añada la fila. Campo del tipo “CURRENT_TIMESTAMP”.

- Temperatura: Guarda la temperatura del sensor DHT11. Campo del tipo “int”.
- Humedad: Guarda la temperatura del sensor DHT11. Campo del tipo “int”.

Ilustración 22: Imagen de la tabla “valores”

La tabla “valorLuz” estará compuesta igualmente por cuatro campos, los campos “ID”, “Tiempo”, “Luz” y “Led”.

- ID: Clave primaria se autoincrementará con cada nueva fila insertada. Campo del tipo “int”.
- Tiempo: Guarda la fecha y hora del momento en el que se añada la fila. Campo del tipo “CURRENT_TIMESTAMP”.
- Luz: Guarda el estado 0 ó 1 según detecte luz o no el sensor lumínico. Campo del tipo “int”.
- Led: Guarda el estado 0 ó 1 según el led este encendido o apagado. Campo del tipo “int”.

Ilustración 23: Imagen de la tabla “valorLuz”

3.2.3 Dispositivos

Los dispositivos de este proyecto son los sistemas que funcionan de herramienta al usuario para interactuar con el sistema domótico. Este proyecto va a implementar una página web y una aplicación Android.

3.2.3.1 Página Web

Para mostrar los datos hemos creado una página web albergada en la Raspberry a la cual se accede desde el navegador escribiendo: rodrigodomotica.ddns.net o también con rododrigodomotica.ddns.net/index.php.

La página esta creada sin utilizar “frameworks”, ni plantillas y usando las siguientes tecnologías;

- HTML para escribir la página en sí.
- CSS3 para crear el diseño de la página.
- PHP para la comunicación con el servidor, obtener los datos, enviar datos y mostrarlos en la página.
- jQuery para crear algunos efectos en la página.

El resultado final de la página es el siguiente:

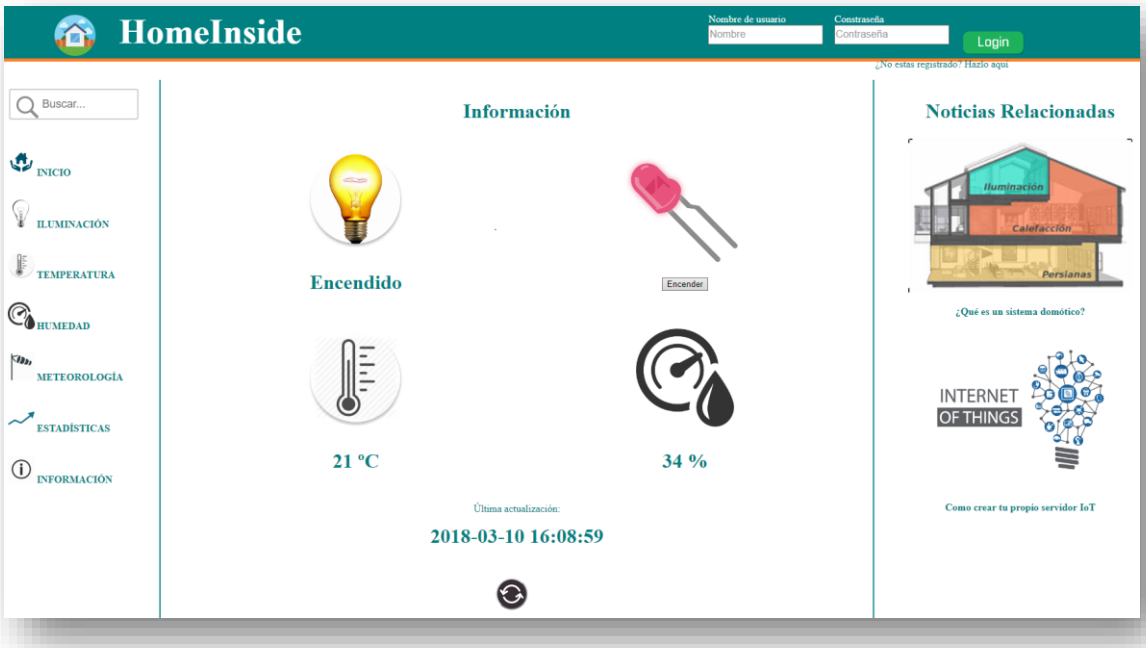

Ilustración 24: Imagen de la página web

Como podemos ver en la ilustración superior tenemos seis elementos en la zona central de la página, una bombilla, un led, un termómetro, un medidor de humedad, la fecha y un botón de actualizar. Estos seis elementos realizan alguna función a través de código PHP que se carga en el “index.php”, como es mostrar datos y realizar acciones.

Al lanzar la página se lanza un script PHP para acceder a la base de datos y escribirlos en la página web. Este script consulta las dos tablas de la base de datos, obtiene la información y la guarda en variables globales de forma que pueda ser accesible desde cualquier parte del HTML. Este script se lanza cada vez que se llama a “index.php” como al recargar la página o cuando apretamos el botón del led.

```
<?php
    include("consultaDB.php");
    consultaDB($dbhost,$dbuser,$dbpass,$dbname);
?>
```

Esto lanza la función “consultaDB” con los parámetros para poder iniciar conexión con MySQL.

Primero creamos las variables globales donde guardar los datos que serán accesibles desde todo el HTML, creamos la conexión con la base de datos y formamos la petición SQL.

```
<?php
    include('parametrosConfDB.php');
    include('crearConexion.php');
    include("consulta.php");
    include('curl');
    function consultaDB($dbhost,$dbuser,$dbpass,$dbname)
    {
        global $temp, $humedad, $time, $luz, $led;

        $enlace=conectar($dbhost,$dbuser,$dbpass,$dbname);
        $consultaSQL="SELECT Temperatura, Humedad, Tiempo FROM valores
                    WHERE ID=(Select MAX(ID) From valores)";
        $consultaSQLLuz="SELECT Luz, Led  FROM valorLuz
                    WHERE ID=(Select MAX(ID) From valorLuz)";
```

A continuación, con la conexión creada y la petición formada hacemos la consulta y guardamos los datos obtenidos en las variables.

```
$data=consulta($enlace,$consultaSQL);
$dataLuz=consulta($enlace,$consultaSQLLuz);

$temp=$data["Temperatura"];
$humedad=$data["Humedad"];
$time=$data["Tiempo"];
$luz=$dataLuz["Luz"];
$led = $dataLuz["Led"];
```

Y por último si “index.php” ha sido lanzado desde el botón para que encienda o apague el led, nos habrá mandado una variable “valorLed” con “true”, si ha sido así entonces creamos la URL oportuna para la petición HTTP al NodeMCU. Si el led estaba encendido mandamos un 0 y si estaba apagado un 1 para que cambie de estado.

```
if($_POST["valorLed"])
{
    if($led == 1)
    {
        $led =0;
        $url = "http://rodrigodomotica.ddns.net:5000/gpio/0";
    }
    else
    {
        $url = "http://rodrigodomotica.ddns.net:5000/gpio/1";
        $led =1;
    }
```

Con la URL formada ahora hacemos dos cosas, guardamos el nuevo estado en la base de datos, y usando cURL hacemos la petición HTTP al NodeMCU.

```
        $consultaLed = "INSERT INTO valorLuz(Luz, Led)
                        VALUES('$luz','$led')";
        consulta($enlace,$consultaLed);

        $ch = curl_init("$url");
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, false);
        curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "GET");
        curl_exec($ch);
    }
    //Cierra la conexión
    mysqli_close($enlace);
}
```

En el script anterior se llama a dos funciones que son “conectar” y consulta, el primero se encarga de iniciar la conexión con la base de datos y devolver los errores en caso de que no se haya podido conectar.

```
<?php
    function conectar($dbhost,$dbuser,$dbpass,$dbname)
    {
        if (!($enlace=mysqli_connect($dbhost,$dbuser,$dbpass,$dbname)))
        {
            echo "Error: No se pudo conectar a MySQL." . PHP_EOL;
            echo "error de depuración: " . mysqli_connect_errno() .
                        PHP_EOL;
            echo "error de depuración: " . mysqli_connect_error() .
                        PHP_EOL;
            exit;
        }
        return $enlace;
    }
?>
```

Y consulta se encarga de hacer la petición a la base de datos y sacar las filas de datos de la respuesta.

```
<?php
    function consulta($enlace,$consultaSQL)
    {
        if($resultado= mysqli_query($enlace,$consultaSQL))
        {
            while($fila=mysqli_fetch_array($resultado,MYSQLI_ASSOC))
            {
                $data=$fila;
            }
        }
        return $data;
    }
?>
```

La actualización de los datos en pantalla se realiza a través de código PHP incrustado en el HTML como podemos observar en el caso del botón de encender led:

```
<div class="datos">
      <img class ="iconosDatosLed" src="led.png">
      <form class="btnActualizarLuz" action="index.php" method="post">
            <input type="hidden" name="valorLed" value=true />
            <?php
                  if($led)
```

```
                {
                        echo "<p><input type='submit' value='Apagar'
                            /></p>";
                }
                else echo "<p><input type='submit' value='Encender'
                            /></p>";
            ?>
        </form>
</div>
```

Aquí se ve como se escriben los datos obtenidos de humedad en la página web mediante PHP.

```
 <div class="datos">
        <img class="iconosDatos" src="humedad.png">
        <?php
              echo "<p class= 'valores'><strong>" .$humedad ."
                    %</strong></p>";
        ?>
</div>
```

Para crear algunos efectos en la página y hacerla un poco más profesional hemos hecho uso de jQuery, que es una biblioteca para JavaScript que nos ayuda a interactuar con los documentos HTML.

Hemos hecho que el texto se subraye al pasar el ratón por encima:

```
$(document).ready(function(){
            $(".enlacesNoSubrallados").mouseover(function () {
             $(this).css('text-decoration', 'underline');
    })
})

$(document).ready(function(){
            $(".enlacesNoSubrallados").mouseleave(function () {
             $(this).css('text-decoration', 'none');
    })
})
```

También que cambie de color el texto al poner el ratón encima:

```
$(document).ready(function(){
            $(".menu").mouseleave(function () {
             $(this).css('text-decoration', 'none');
             $(this).css('color', '#008080');
    })
   })
```

```
$(document).ready(function(){
            $(".menu").mouseover(function () {
             $(this).css('text-decoration', 'underline');
             $(this).css('color', '#ff8000');
    })
})
```

No se ha mostrado todo el código jQuery, PHP ni HTML, así como no se ha mostrado el código CSS con el que hemos realizado el diseño de la página. Todo este código está disponible en el Anexo.

3.2.3.2 Aplicación Android

La otra manera que tenemos de mostrar la información del sistema domótico es mediante una aplicación Android. La aplicación va a realizar las mismas acciones que se pueden hacer desde la página web, pero además va a conectar con un sistema de meteorología para tener la información meteorológica de la zona que quieras.

Para desarrollar la aplicación vamos a hacer uso del IDE que proporciona Google AndroidStudio. La aplicación se estructura en seis actividades, una pantalla principal desde donde tienes acceso a todas las demás y con una barra lateral desde la que también puedes moverte por las distintas opciones. Otras tres actividades donde se muestran los datos de los sensores y el control del led como son “Iluminación”, “Temperatura” y “Humedad”. Otra actividad con información meteorológica, otra de configuración y finalmente una con información sobre el desarrollador.

Actividad Principal:

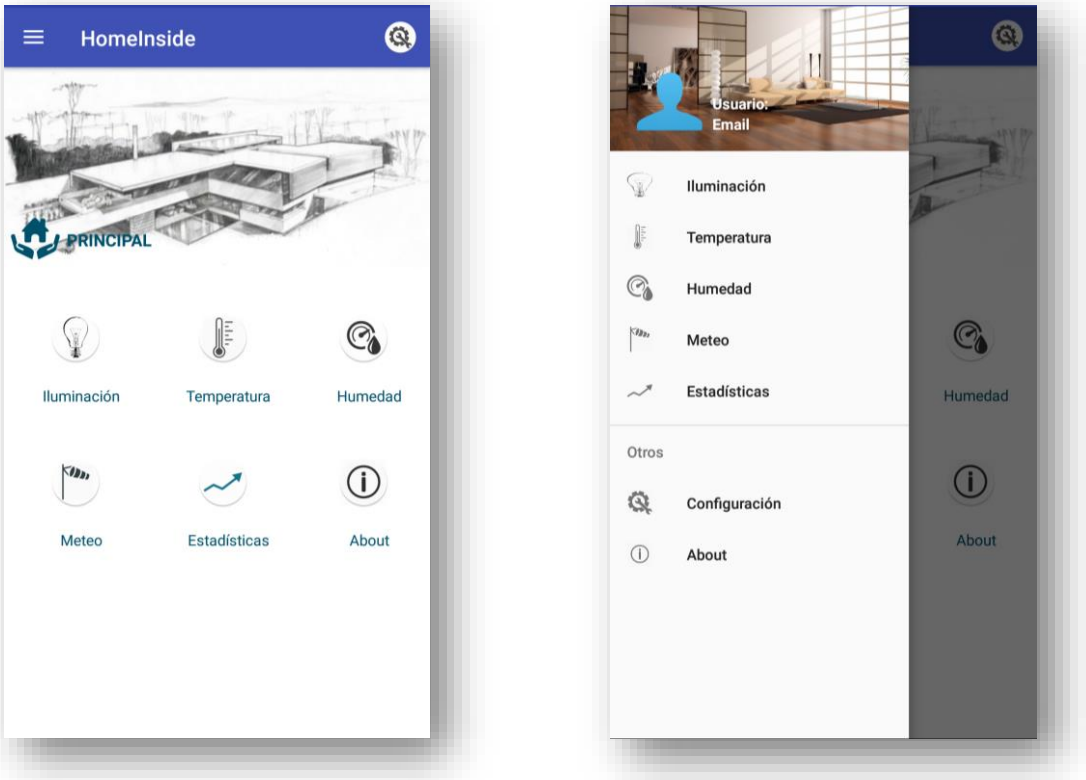

Ilustración 25 y 26: Imagen de la actividad Principal

Esta actividad es la primera que vemos después de la pantalla de carga de la aplicación, y no muestra información propiamente dicha, sino que te permite elegir qué información quiere ver y te manda a esa actividad.

Para ello tenemos un método “buttonOnClicked” que se lanzara cuando se pinche sobre alguna de las “ImageView”. Este método sacara el “ID” del elemento que creado el evento y lanzara la actividad correspondiente.

```
public void buttonOnClicked(View v){
    int id= v.getId();
    Intent intent;
    switch (id){
        case R.id.button_humedad:
            intent= new Intent(this,HumedadActivity.class);
            startActivity(intent);
            break;
        case R.id.button_iluminacion:
            intent= new Intent(this,IluminacionActivity.class);
            startActivity(intent);
            break;
        case R.id.button_estadisticas:
            intent= new Intent(this,EstadisticasActivity.class);
            startActivity(intent);
            break;
        case  R.id.button_about:
            intent=new Intent(this,AboutActivity.class);
            startActivity(intent);
            break;
        case  R.id.button_meteo:
            intent=new Intent(this,WeatherActivity.class);
            startActivity(intent);
            break;

        case R.id.button_temperatura:
            intent = new Intent(this,TemperaturaActivity.class);
            startActivity(intent);
            break;
    }
}
```

Actividad Temperatura y Humedad:

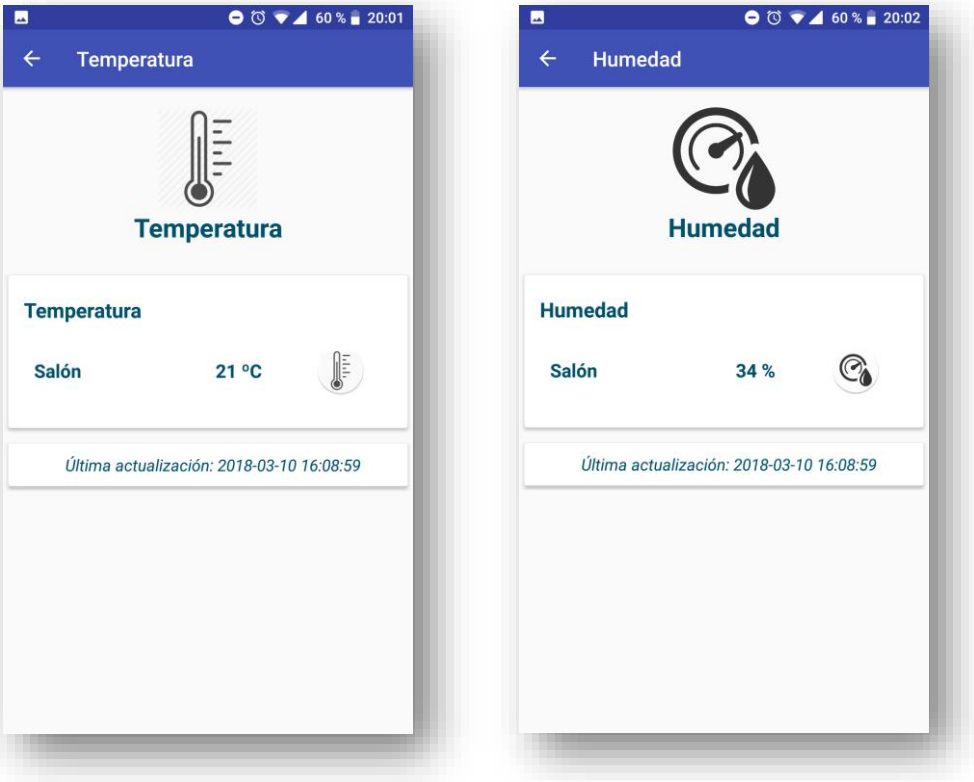

Ilustración 27 e 28: Imagen de la actividad Temperatura y Humedad

Estas dos actividades y la de iluminación (exceptuando el control del led) tienen el mismo funcionamiento por lo cual las vamos a explicar juntas.

Al iniciarse las actividades entran en el método “onCreate” donde lo primero que hace es guardar un estado anteriormente guardado si lo hubiese para cargarlo y cargar el “layout” de la interfaz.

Seguidamente realiza la instrucción “getDataSensor.execute” (“getDataSensor” es un objeto de la clase “GetDataSensor” previamente creado) con la URL a la que queremos acceder (en nuestro caso es un PHP que hay en el servidor llamado “getAdroid.php”) y le pasamos la variable “pass” con valor 1234 para que pueda ejecutar el script ya que le hemos puesto contraseña como medio de seguridad.

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_temperatura);

      getDataSensor.delegate=this;
      getDataSensor.execute("http://rodrigodomotica.ddns.net/getAndroid.p
      hp?pass=1234");
}
```

Al ejecutar esa instrucción llama al método “doInBackground” de la clase “GetDataSensor” que hereda de “AsyncTask<String, Void, String>”. Este método se lanza en un hilo nuevo, volviendo una vez que ha terminado al hilo principal. Una vez terminado este método se lanza el método “onPostExecute” y se le pasa el resultado de la operación.

```
@Override
protected String doInBackground(String... urls) {

    try {
        URL url = new URL(urls[0]);
        InputStream is = null;
        int len = 500;

        String contentAsString;
        HttpURLConnection connection = (HttpURLConnection)
                                    url.openConnection();
        connection.setRequestMethod("GET");
        connection.setDoInput(true);

        if (connection.getResponseCode() == HttpURLConnection.HTTP_OK) {
            int response = connection.getResponseCode();

            is = connection.getInputStream();
            contentAsString = readStream(is, len);

            Log.d("stringFinal", "the response is: " + contentAsString);
            return contentAsString;
        }
        connection.disconnect();
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return "";
```

Y finalmente nosotros haciendo uso de la interfaz creada por nosotros “AsyncResponse” llevamos el resultado a nuestra clase.

```
@Override
protected void onPostExecute(String result)
{
    delegate.processFinish(result);
}
```

En el método “processFinish” de nuestra clase “HumedadActivity” recibimos la respuesta en formato JSON. Del JSON sacamos los valores y los asignamos a los “TextView” de la Actividad.

```
public void processFinish(String output)
{
      TextView temperatura = (TextView)
                        findViewById(R.id.temperaturaMostrar);
      TextView ultActualizacion = (TextView)
                        findViewById(R.id.tempUltimaActualizacion);

      try {
            JSONObject json = new JSONObject(output);
            temperatura.setText(json.getString("Temperatura")+" ºC");
            ultActualizacion.setText("Última actualización:
                              "+json.getString("TiempoTemp"));

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

Cuando se hace la peticion HTTP en el “doInBackground” lo que hace es llamar a un PHP que hay en el servidor, el cual accede a la base de datos y los devuelve. El script obtiene todos los datos de las dos tablas, luego crea un array con ellos y este lo pasa a JSON para enviarlo como respuesta. Esta respuesta la recibe el metodo “doInBackground” la pasa a formato “InputStream” y luego a String que es lo que devuelve.

```
<?php
  include('parametrosConfDB.php');
  include('crearConexion.php');
  include("consulta.php");

  $password="1234";
  if( $_GET['pass'] == $password)
  {
      $enlace=conectar($dbhost,$dbuser,$dbpass,$dbname);
```

```
      $consultaSQL="SELECT Temperatura, Humedad, Tiempo FROM valores
            WHERE ID=(Select MAX(ID) From valores)";
      $consultaSQLLuz="SELECT Luz, Led, Tiempo  FROM valorLuz
            WHERE ID=(Select MAX(ID) From valorLuz)";

      $data=consulta($enlace,$consultaSQL);
      $dataLuz=consulta($enlace,$consultaSQLLuz);
      $arr =
array("Luz"=>$dataLuz["Luz"],"Led"=>$dataLuz["Led"],"Tiempo"=>$dataLuz["T
iempo"],

"Temperatura"=>$data["Temperatura"],"Humedad"=>$data["Humedad"],"TiempoTe
mp"=>$data["Tiempo"]);
      echo json_encode($arr);
  }
?>
```

Este funcionamiento es el mismo para obtener los datos de las otras actividades “Temperatura” y “Iluminación” aunque en el caso de “Iluminación” tiene una funcionalidad más, que es la de encender y apagar el led.

Actividad Iluminación:

Ilustración 29: Imagen de la actividad Iluminación

La Actividad Iluminación tiene el mismo funcionamiento para obtener los datos del servidor como el estado del sensor lumínico y la fecha en que se guardaron los valores. Pero tiene una funcionalidad extra que es la de encender y apagar el led.

Cuando haces click en el “ImagenView” del led lanza el metodo “onClickBombilla”. Este metodo lo que hace es ejecutar la instrucción “execute(URL)” pero esta vez de la clase “PostDataSensor”. La URL llama a un recurso PHP del servidor (“postLuzAndroid.php”) y le pasa una variable “Led” con el valor que tiene que cambiar.

```
public void onClickBombilla(View v)3
{
	ImageView image=(ImageView) v;
	PostDataSensor postDataSensor = new PostDataSensor();

	if(datos.isLed())
```

```
            {
            image.setImageDrawable(getResources().getDrawable(R.mipmap.
                  bombilla_apagada_redonda));
            datos.setLed(false);

            postDataSensor.execute("http://rodrigodomotica.ddns.net/postL
            uzAndroid.php?pass=1234&Led=0");
      }else
      {
            image.setImageDrawable(getResources().getDrawable(R.mipmap.
                        bombilla_encendida_redonda));
            datos.setLed(true);

            postDataSensor.execute("http://rodrigodomotica.ddns.net/postL
            uzAndroid.php?pass=1234&Led=1");
      }
}
```

El archivo que llamamos desde Android para encender y apagar el led es el siguiente. En el podemos observar el mismo funcionamiento que cuando encendiamos o apagabamos el led desde la pagina web. Comprobamos el valor que hay en la base de datos, luego guardamos el nuevo valor del led y finalmente usamos cURL para mandar al NodeMCU el nuevo estado.

```
Include('…')
$password="1234";
if( $_GET['pass'] == $password)
{
    $led = $_GET['Led'];
    $enlace=conectar($dbhost,$dbuser,$dbpass,$dbname);

    $consultaLuz = "SELECT Luz  FROM valorLuz
                    WHERE ID=(Select MAX(ID) From valorLuz)";
    $dataLuz=consulta($enlace,$consultaLuz);
    $luz = $dataLuz["Luz"];

    $consultaLed = "INSERT INTO valorLuz(Luz, Led)
                     VALUES('$luz','$led')";
    $data=consulta($enlace,$consultaLed);
    mysqli_close($enlace);

    $url = "http://rodrigodomotica.ddns.net:5000/gpio/{$led}";
    $ch = curl_init("$url");
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, false);
    curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "GET");
    curl_exec($ch);
    curl_close($ch);
```

Actividad Meteorología:

Ilustración 30: Imagen de la actividad Meteorología

Esta actividad se conecta a openweathermap.org que ofrece un servicio meteorológico, así que a través de su API podemos crear una URL y hacer una petición HTTP obteniendo así los datos del tiempo que del lugar que queramos. Un servicio muy útil y relacionado con un sistema domótico ya que saber el clima nos permite tomar decisiones como subir la calefacción, etc.

Su funcionamiento es parecido al que hemos llevado a cabo en las actividades anteriores para obtener datos mediante peticiones HTTP. La principal diferencia está en crear la URL, ya que esta debe seguir el formato especificado por la API para hacer la consulta.

Una vez conformada la URL solo hay que llamar al método “doInBackground” y obtenemos la respuesta en forma de JSON.

```
String ciudad=this.parent.getUbicacion();

Uri.Builder uriBuilder = new Uri.Builder();
uriBuilder.scheme("http");
uriBuilder.authority("api.openweathermap.org");
uriBuilder.appendPath("data");
uriBuilder.appendPath("2.5");
uriBuilder.appendPath("weather");
uriBuilder.appendQueryParameter("q", ciudad);
uriBuilder.appendQueryParameter("APPID",
          "a4d47e1f013caa3f3718785b5c8b9dab");
URL url = new URL(uriBuilder.build().toString());
```

Una vez conformada la URL y ya obtenido la respuesta la tratamos para obtener los valores. Como se puede observar cambiando el valor de la variable “ciudad” obtenemos los datos de la ciudad que queramos. Esto se podrá cambiar en la actividad “Configuración”

```
JSONObject result = new JSONObject(lectura);
String temp = result.getJSONObject("main").getString("temp");
String humidity =
          result.getJSONObject("main").getString("humidity");
String viento = result.getJSONObject("wind").getString("speed");
JSONArray array = result.getJSONArray("weather");
String desc = array.getJSONObject(0).getString("description");
```

Actividad Configuración:

Ilustración 31: Imagen de la actividad Configuración

Finalmente, la última actividad funcional que tenemos es la de “Configuración”, en ella podemos introducir los datos de usuario y cambiar la ubicación.

La opción de identificarse no tiene funcionalidad realmente, simplemente te permite meter un nombre y contraseña ya registrados en la aplicación y una vez identificado aparece esa información en la barra lateral de la actividad principal. Y la opción de “Idioma” no está implementada.

La parte que sirve de esta actividad es la de “Ubicación” que nos permite introducir una ciudad y el sistema de meteorología sacara los datos de esa ciudad.

Para guarda estos datos de forma persistente y no tengas que configurar ubicaciones y usuarios continuamente, hace uso de las “SharedPreferences”, estas permiten almacenar persistentemente duplas <identificador, valor> y ser cargadas desde cualquier parte de la aplicación.

Para guardar los datos de una nueva ubicación se llama al método “onClickUbicacion”.

```
public void onClickUbicacion(View view){
    EditText et=(EditText) findViewById(R.id.editarUbicacion);
    SharedPreferences
      preferences=PreferenceManager.getDefaultSharedPreferences(this);
    SharedPreferences.Editor editor=preferences.edit();
    String hola=et.getText().toString();

    if(!hola.equals("")){
        editor.putString("ubicacion",et.getText().toString());
        editor.apply();

      if(hola.equals(preferences.getString("ubicacion","aaasdfafddf")))
            Toast.makeText(ConfiguracionActivity.this,R.string.
                  correcto_cambioUbicacion,Toast.LENGTH_SHORT).show();

        else Toast.makeText(ConfiguracionActivity.this,
            R.string.error_cambioUbicacion,Toast.LENGTH_SHORT).show();
    }
    else Toast.makeText(ConfiguracionActivity.this,"Ubicación no
                  válida",Toast.LENGTH_SHORT).show();
}
```

4. Presupuesto

En este apartado se especifican los costes asociados al proyecto. Los costes se encuentran desglosados en tres apartados: mano de obra, material y licencias de software.

Mano de obra:

En esta tabla se muestran los costes por hora y por tipo de trabajo realizados.

Puesto	Horas	Coste por hora (€/h)	Coste total (€)
Ingeniero	576	10.5	6048
Supervisor	10	20	200
Subtotal			6248

Tabla 1: Mano de obra

Material:

En este apartado se muestran las herramientas que hemos usado para desarrollar el proyecto.

Material para la elaboración del estudio	Concepto	Cantidad	Coste total (€)
Asus S550cb	Ordenador portátil	1	900
Documentación	Libros, etc.	1	50
Subtotal			950

Tabla 2: Material para la elaboración del estudio

En la siguiente tabla se desglosa los componentes que han sido necesarios para implementar el sistema domótico.

Material para la implementación del sistema	Concepto	Cantidad	Coste unitario (€/u)	Coste total (€)
Raspberry Pi 3B	Computador	1	40	40
NodeMCU	Placa de desarrollo	2	5	10
DHT11	Sensor	1	2.5	2.5
KY-18	Sensor	1	2	2
Multímetro	Instrumento eléctrico	1	15	15
Electrónica	Componentes varios: led, cables, etc.	1	20	20
Subtotal				89.5

Tabla 3: Material para la implementación del proyecto

Licencias de software:

En este apartado se muestran las licencias que hemos tenido que adquirir para poder desarrollar el proyecto.

Licencias de software	Cantidad	Coste unitario (€/u)	Coste total (€)
Microsoft Office 2016	1	149	149
Subtotal			149

Tabla 4: Licencias de software

Presupuesto final:

Tipo de coste	Coste total (€)
Mano de obra	6248
Material para la elaboración del estudio	950
Material para la implementación del sistema	89.5
Licencias de software	149
Total	7436.5

Tabla 5: Coste final

5. Conclusión y mejoras

Por último, para concluir este proyecto vamos a realizar un análisis del trabajo realizado para conseguir llegar hasta este punto, y las posibles mejoras que se podrían implantar o cambios que ahora con perspectiva se vean más claros.

Hemos realizado un sistema domótico, empezando por el servidor, pasando por control de sensores, hasta hacer una página web y una aplicación Android. El objetivo ha sido crear todo el sistema de la forma más independiente posible, desarrollando todas las funcionales nosotros mismos, sin usar ningún servicio de terceros como pueden ser “frameworks” para páginas web como por ejemplo “WordPress”, “Symfony” para diseñar el “back-end” o algún servidor en la nube para sistemas IoT.

Esto ha significado que hemos tocado multitud de tecnologías, gran parte de ellas vistas por primera vez y que hemos tenido que aprender, por nombrar algunas: PHP, Android, ESP8266, HTML, CSS, jQuery, C, Raspberry, redes, Apache, MySQL, HTTP, etc.

Lo que ha significado ser un proyecto increíblemente enriquecedor y con un valor formativo elevadísimo. Mientras ibas desarrollando algún punto concreto ibas profundizando en esa tecnología y una vez acababas esa parte, la dejabas y empezabas de cero con la siguiente parte, documentándote como hacerla y que tecnología era más apropiada. Esta búsqueda del método más apropiado de desarrollar el proyecto nos ha llevado a usar tecnologías hasta momentos avanzados del proyecto que al final han sido sustituidos y no aparecen en este proyecto. Alguna de estas son desde chips ESP8266-01 que alimentábamos con una fuente ATX de un ordenador (finalmente ambas fueron sustituidas por las placas NodeMCU), hasta Arduino, con el que desarrollamos al principio toda la parte de electrónica y que al final también lo sustituimos por las NodeMCU.

A lo largo del proyecto ha habido tecnologías que queríamos implementar y al final no ha sido posible por falta de tiempo o porque significaría modificar gran parte de este.

Estas mejoras que se podrían aplicar al proyecto son utilizar el protocolo MQTT (que es un protocolo planeado para IoT, con un consumo muy reducido y posibilidad de suscribirse a temas…) para realizar la comunicación entre los componentes. Otra mejora muy importante (sobre todo en sistemas conectados a la red) es la seguridad. Este es un tema muy amplio, ya que hay muchos campos a segurizar: registro de usuarios, encriptación de contraseñas de

usuarios, protocolos seguros como HTTPS, en vez de permitir el acceso directo a la base de datos, desarrollar una API para obtener los datos.

Si fuéramos a darle un uso real al sistema y lo mantuviéramos alimentado mediante baterías, una opción para mejorar su autonomía seria usar los chips ESP8266 en vez de la placa NodeMCU ya que el consumo de estos últimos es un poco mayor.

Elegimos los componentes que íbamos a usar en el sistema domótico intentando que fuesen lo más representativos posible. Una vez que tenemos funcionando 3 sensores y un actuador ya se pude ampliar sin ningún problema, no hay ninguna diferencia respecto a dificultad ni a tener que desarrollar nuevas funcionalidades el meter n componentes más. El led lo montamos para mostrar el funcionamiento de un actuador y por la facilidad que tiene a la hora de transportarlo para la defensa del TFG, aunque una idea mejor es usar un relé como interruptor.

Por lo que, para finalizar, este proyecto toca una cantidad asombrosa de tecnologías lo que se traduce en un gran aprendizaje por parte del estudiante. También es un proyecto con muchas posibilidades y que se podría seguir aumentando indefinidamente, como hemos mostrado con las mejoras implementables, así que hemos intentado asegurarnos de que el proyecto es suficientemente representativo de la variedad de posibilidades que hay.

6. Bibliografía

[1] Como montar LAMP en una Raspberry Pi: https://www.1and1.es/digitalguide/servidores/configuracion/como-configurar-un-servidor-web-raspberry-pi-con-lamp/

[2] Página web oficial de Raspberry PI, información, instalación de sus sistemas operativos: https://www.Raspberrypi.org/

[3] Página web oficial de Arduino, información, IDE, productos: https://www.arduino.cc/

[4] Tutorial sobre NodeMCU: https://programarfacil.com/podcast/nodemcu-tutorial-paso-a-paso/

[5] NO-IP: https://www.noip.com/

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

[7] Api de CSS: https://www.w3schools.com/css/

[8] Api jQuery: http://api.jquery.com/

[9] Tutoriales sobre Arduino, ESP8266, NodeMCU y componentes electrónicos: https://www.prometec.net/indice-tutoriales/

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

[11] Wikipedia: Enciclopedia digital con información de cualquier tema: https://es.wikipedia.org/wiki/Wikipedia:Portada

[12] Stackoverflow: foro sobre prácticamente cualquier tema de informática: https://stackoverflow.com/

[13] Api de Android: https://developer.android.com/index.html

[14] NodeMCU, “Página oficial”. Disponible en: http://nodemcu.com/index_en.html.

[15] Llorens Agost, Marisa, Gómez Adrián, Jon Ander, Galiano Ronda, Isabel (2016), “Empezar a programar usando Java”, Académica

[16] Jesús Tomás Gironés (2017), “El gran libro de Android 6ª Ed”, Marcombo

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

7. Anexos

7.1 Programa NodeMCU (DHT11)

Programa que se ejecuta en uno de los NodeMCU. Se encarga de obtener la temperatura y humedad y transmitirla al servidor.

```
#include <DHT.h>
#include <ESP8266WiFi.h>

//info conectar sensores
int pinDht=2;

DHT dht(pinDht,DHT11);//cargar dht

//LOS VALORES DE LOS SENSORES QUE ENVIAREMOS
//A LA BASE DE DATOS
int temp=-100;
int hum=-100;

//info para conectarse wifi
const char* ssid = "MOVISTAR_3089";
const char* password = "SSyjZLbBJ9q6tvJnzebv";
const char* host = "rodrigodomotica.ddns.net";

void getDataFromSensor()
{
    //luz=analogRead(pinLuz);
    int auxTemp= dht.readTemperature();
    int auxHum=dht.readHumidity();

    //CONTROL DE VALORES ERRONEOS
    if((abs(auxTemp) < abs(temp)*5) && (abs(auxHum) < abs(hum)*5))
    {
      temp=auxTemp;
      hum=auxHum;
    }
}

void setup()
{
   dht.begin();
   Serial.begin(9600);

   //INICIA LA CONEXION Y ESPERA HASTA QUE ESTE REALIZADA
```

```
   WiFi.begin(ssid, password);
   while (WiFi.status() != WL_CONNECTED) {
   delay(500);
   Serial.print(".");
   }
}

void loop()
{
   Serial.print("connecting to ");
   Serial.println(host);

   // Use WiFiClient class to create TCP connections
   WiFiClient client;
   const int httpPort = 80;
   if (!client.connect(host, httpPort)) {
   Serial.println("connection failed");
   return;
   }

   //Coge informacion del sensor y lo muestra por serial
   getDataFromSensor();

  // We now create a URL for the request
  String url = "/subirToDB.php";
  String key = "?pass=1234";
  String dato1 = "&Temperatura=";
  String dato2 = "&Humedad=";

  // This will send the request to the server
  client.print(String("GET ") + url + key + dato1 + temp + dato2 + hum
               +" HTTP/1.1\r\n" + "Host: " + host
               + "\r\n" + "Connection: close\r\n\r\n");
  unsigned long timeout = millis();
  while (client.available() == 0) {
    if (millis() - timeout > 5000) {
      Serial.println(">>> Client Timeout !");
      client.stop();
      return;
     }
  }
  // Read all the lines of the reply from server and print them to Serial
  while (client.available()) {
  String line = client.readStringUntil('\r');
  Serial.print(line);
  }
  delay(30000);             //Lee dos veces por minuto
}
```

7.2 Programa NodeMCU (Luz, Led y Botón)

Programa que se ejecuta en el otro de los NodeMCU. Se encarga de obtener la iluminación y el control del led. Ejerce de cliente y de servidor.

```
#include <ESP8266WiFi.h>

//info conectar sensores
int pinLuz=A0;//sensor luz
int led = D1;
int button = D2;
volatile int estado = LOW;
volatile int clientLed;
volatile int t=0;//para ver si el boton tiene efecto rebote

//LOS VALORES DE LOS SENSORES QUE ENVIAREMOS
//A LA BASE DE DATOS
int luz = -100;

//info para conectarse wifi
const char* ssid = "MOVISTAR_3089";
const char* password = "SSyjZLbBJ9q6tvJnzebv";
const char* host = "rodrigodomotica.ddns.net";

//Puerto donde oira el esp
WiFiServer server(5000);

void getDataFromSensor()
{
    luz=analogRead(pinLuz);
 }

void manejadorLed(){
  if(millis()>t+250)
  {
    //cont++;
    estado=digitalRead(led);
    if(estado==LOW)
    {
      digitalWrite(led,HIGH);
      clientLed = estado = HIGH;
    }
    else
    {
      digitalWrite(led,LOW);
      clientLed = estado = LOW;
```

```
    }
   }
   t=millis();
  }
void setup()
{
   //Cuando caiga(falling), flanco de bajada de la señal
   pinMode(led, OUTPUT);
   digitalWrite(led, estado);
   pinMode(button, INPUT);
   attachInterrupt(button, manejadorLed, RISING);

   Serial.begin(9600);
   //INICIA LA CONEXION Y ESPERA HASTA QUE ESTE REALIZADA
   WiFi.begin(ssid, password);
   while (WiFi.status() != WL_CONNECTED) {
   delay(500);
   Serial.print(".");
   }

  // Start the server
  server.begin();
  Serial.println("Server started");
}

int getDataClient()
{
    // Check if a client has connected
  WiFiClient client = server.available();
  if (!client) {
    return -1;
  }

  // Wait until the client sends some data
  Serial.println("new client");
  while(!client.available()){
    delay(1);
  }

  // Read the first line of the request
  String req = client.readStringUntil('\r');
  client.flush();

  // Match the request
  int val;
  if (req.indexOf("/gpio/0") != -1)
    val = 0;
  else if (req.indexOf("/gpio/1") != -1)
    val = 1;
```

```
  else {
    Serial.println("invalid request");
    client.stop();
    return -1;
  }

  client.flush();
  return val;
}

void loop()
{
   //int estadoLed = digitalRead(led);
   int clientLed = getDataClient();
   if(clientLed != estado && clientLed != -1)
   {
    digitalWrite(led, clientLed);
    estado = clientLed;
   }

   // Use WiFiClient class to create TCP connections
   WiFiClient client;
   const int httpPort = 80;
   if (!client.connect(host, httpPort)) {
   Serial.println("connection failed");
   return;
   }

   //Coge informacion del sensor y lo muestra por serial
   getDataFromSensor();

  // Creamos url para peticion
  String url = "/subirToDBLuz.php";
  String key = "?pass=1234";
  String dato1 = "&Luz=";
  String dato2 = "&Led=";

  client.print(String("GET ") + url + key + dato1 + luz + dato2 + estado
               +" HTTP/1.1\r\n" + "Host: " + host
               + "\r\n" + "Connection: close\r\n\r\n");
  unsigned long timeout = millis();
  while (client.available() == 0) {
    if (millis() - timeout > 5000) {
      Serial.println(">>> Client Timeout !");
      client.stop();
      return;
     }
  }
  // Read all the lines of the reply from server and print them to Serial
```

```
  while (client.available()) {
  String line = client.readStringUntil('\r');
  }
}
```

7.3 Subir datos del NodeMCU 1 a la BD (subirToDB.php)

Programa que se ejecuta en el servidor. Coge los datos del NodeMCU de temperatura y humedad y los guarda en la base de datos.

```
<?php
$password="1234";
if( $_GET['pass'] == $password)
{
  require("parametrosConfDB.php");
  //Conexion con la base de datos
  $con = mysqli_connect($dbhost, $dbuser, $dbpass, $dbname);
  //obtenemos los datos de la url y con el formato para query apropiado
  $Temperatura = mysqli_real_escape_string($con, $_GET['Temperatura']);
  $Humedad = mysqli_real_escape_string($con, $_GET['Humedad']);
  // EInsertamos los valores en la tabla
  $query = "INSERT INTO valores(Temperatura, Humedad)
            VALUES('$Temperatura','$Humedad')";
  // Ejecutamos la instruccion
  mysqli_query($con, $query);
  mysqli_close($con);
  */
}
else
{
  echo "Acceso bloqueado. Necesitas la contraseña para acceder a la base
de datos.";
}
?>
```

7.4 Subir datos del NodeMCU 2 a la BD (controlLed.php)

Programa que se ejecuta en el servidor. Coge los datos del NodeMCU de iluminación y led y los guarda en la base de datos.

```
<?php
$password="1234";
if( $_GET['pass'] == $password)
```

```
{
  require("parametrosConfDB.php");
  //Conexion con la base de datos
  $con = mysqli_connect($dbhost, $dbuser, $dbpass, $dbname);
  //obtenemos los datos de la url y con el formato para query apropiado
  $led = mysqli_real_escape_string($con, $_GET['Led']);
  $luz = mysqli_real_escape_string($con, $_GET['Luz']);
  // EInsertamos los valores en la tabla
  $query = "INSERT INTO valorLuz(Luz, Led) VALUES('$luz','$led')";
  // Ejecutamos la instruccion
  mysqli_query($con, $query);
  mysqli_close($con);

}
else
{
  echo "Acceso bloqueado. Necesitas la contraseña para acceder a la base
de datos.";
}
?>
```

7.5 Crear conexión MySQL (crearConexion.php)

Función para crear una conexión MySQL.

```
<?php

    function conectar($dbhost,$dbuser,$dbpass,$dbname)
    {
        if (!($enlace=mysqli_connect($dbhost,$dbuser,$dbpass,$dbname)))
        {
            echo "Error: No se pudo conectar a MySQL." . PHP_EOL;
            echo "error de depuración: " . mysqli_connect_errno() .
                                PHP_EOL;
            echo "error de depuración: " . mysqli_connect_error() .
                                PHP_EOL;
            exit;
        }
        return $enlace;
    }
?>
```

7.6 Petición (Query) MySQL (consulta.php)

Función para realizar una petición MySQL.

```
<?php
    function consulta($enlace,$consultaSQL)
    {
        if($resultado= mysqli_query($enlace,$consultaSQL))
        {
            while($fila=mysqli_fetch_array($resultado,MYSQLI_ASSOC))
            {
                $data=$fila;
        }
        return $data;
    }
?>
```

7.7 Programa principal Pag. Web (consultaDB.php)

Script que carga todos los datos en la página web. Se ejecuta siempre al llamar a “index.php”.

```
<?php
    include('parametrosConfDB.php');
    include('crearConexion.php');
    include("consulta.php");
    include('curl');
    function consultaDB($dbhost,$dbuser,$dbpass,$dbname)
    {
        global $temp, $humedad, $time, $luz, $led;
        $enlace=conectar($dbhost,$dbuser,$dbpass,$dbname);
        $consultaSQL="SELECT Temperatura, Humedad, Tiempo FROM valores
                    WHERE ID=(Select MAX(ID) From valores)";
        $consultaSQLLuz="SELECT Luz, Led  FROM valorLuz
                    WHERE ID=(Select MAX(ID) From valorLuz)";

        $data=consulta($enlace,$consultaSQL);
        $dataLuz=consulta($enlace,$consultaSQLLuz);

        $temp=$data["Temperatura"];
        $humedad=$data["Humedad"];
        $time=$data["Tiempo"];
        $luz=$dataLuz["Luz"];
        $led = $dataLuz["Led"];

        if($_POST["valorLed"])
        {
            if($led == 1)
```

```
            {
                $led =0;
                $url = "http://rodrigodomotica.ddns.net:5000/gpio/0";
            }
            else
            {
                $url = "http://rodrigodomotica.ddns.net:5000/gpio/1";
                $led =1;
            }
            $consultaLed = "INSERT INTO valorLuz(Luz, Led)
                        VALUES('$luz','$led')";
            consulta($enlace,$consultaLed);

            $ch = curl_init("$url");
            curl_setopt($ch, CURLOPT_RETURNTRANSFER, false);
            curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "GET");
            curl_exec($ch);
            curl_close($ch);
        }
        //Cierra la conexión
        mysqli_close($enlace);
    }
    //consultaDB($dbhost,$dbuser,$dbpass,$dbname);
?>
```

7.8 Operación “GET” Android (getAndroid.php)

Programa que llama la aplicación Android para obtener los datos de la base de datos.

```
<?php
  include('parametrosConfDB.php');
  include('crearConexion.php');
  include("consulta.php");

  $password="1234";
  if( $_GET['pass'] == $password)
  {
      $enlace=conectar($dbhost,$dbuser,$dbpass,$dbname);
      $consultaSQL="SELECT Temperatura, Humedad, Tiempo FROM valores
            WHERE ID=(Select MAX(ID) From valores)";
      $consultaSQLLuz="SELECT Luz, Led, Tiempo  FROM valorLuz
            WHERE ID=(Select MAX(ID) From valorLuz)";

      $data=consulta($enlace,$consultaSQL);
      $dataLuz=consulta($enlace,$consultaSQLLuz);
```

```
      $arr =
array("Luz"=>$dataLuz["Luz"],"Led"=>$dataLuz["Led"],"Tiempo"=>$dataLuz["T
iempo"],

"Temperatura"=>$data["Temperatura"],"Humedad"=>$data["Humedad"],"TiempoTe
mp"=>$data["Tiempo"]);
      echo json_encode($arr);
  }
?>
```

7.9 Operación “Post” Android (postLuzAndroid.php)

Programa que llama la aplicación Android para cambiar el estado del led.

```
<?php
    include('parametrosConfDB.php');
    include('crearConexion.php');
    include("consulta.php");
    include('curl');

    $password="1234";
    if( $_GET['pass'] == $password)
    {
        $led = $_GET['Led'];

        $enlace=conectar($dbhost,$dbuser,$dbpass,$dbname);

        $consultaLuz = "SELECT Luz  FROM valorLuz
                        WHERE ID=(Select MAX(ID) From valorLuz)";
        $dataLuz=consulta($enlace,$consultaLuz);
        $luz = $dataLuz["Luz"];

        $consultaLed = "INSERT INTO valorLuz(Luz, Led)
VALUES('$luz','$led')";
        $data=consulta($enlace,$consultaLed);

        //Cierra la conexión
        mysqli_close($enlace);

        $url = "http://rodrigodomotica.ddns.net:5000/gpio/{$led}";
        $ch = curl_init("$url");
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, false);
        curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "GET");
        curl_exec($ch);
        curl_close($ch);
    }
?>
```

7.10 Android clase GetDataSensor.java

Clase Android para pedir los datos del servidor.

```
public class GetDataSensor extends AsyncTask<String, Void, String>
{
    public AsyncResponse delegate = null;

    //este es el unico metodo que funciona en otro hilo,
    // se ejecuta y devuelve a onPostExecute
    //ver estado -> onProgressUpdate
    @Override
    protected String doInBackground(String... urls) {

        try {
            // Creates a new URL from the URI
            URL url = new URL(urls[0]);
            InputStream is = null;
            int len = 500;
            String contentAsString;
            // Get sa connection to the web service
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            connection.setRequestMethod("GET");
            connection.setDoInput(true);

            // Process the response if it was successful (HTTP_OK = 200)
            if (connection.getResponseCode() ==
HttpURLConnection.HTTP_OK) {
                int response = connection.getResponseCode();
                Log.d("REsponse", "the response is: " + response);
                is = connection.getInputStream();

                //convert InputStream into string
                contentAsString = readStream(is, len);

                Log.d("stringFinal", "the response is: " +
                        contentAsString);
                return contentAsString;
            }
            // Close the connection
            connection.disconnect();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return "";
```

```
    }

    @Override
    protected void onPostExecute(String result)
    {
        delegate.processFinish(result);
    }

    public String readStream(InputStream stream, int maxReadSize)
            throws IOException, UnsupportedEncodingException {
        Reader reader = null;
        reader = new InputStreamReader(stream, "UTF-8");
        char[] rawBuffer = new char[maxReadSize];
        int readSize;
        StringBuffer buffer = new StringBuffer();
        while (((readSize = reader.read(rawBuffer)) != -1) && maxReadSize
> 0) {
            if (readSize > maxReadSize) {
                readSize = maxReadSize;
            }
            buffer.append(rawBuffer, 0, readSize);
            maxReadSize -= readSize;
        }
        return buffer.toString();
    }
```

7.11 Android clase PostDataSensor.java

Clase Android para para enviar los datos del led.

```
public class PostDataSensor extends AsyncTask<String, Void, String>
{
    @Override
    protected String doInBackground(String... urls)
    {
        try
        {
            // Creates a new URL from the URI
            URL url = new URL(urls[0]);

            // Get a connection to the web service
            HttpURLConnection connection = (HttpURLConnection)
                  url.openConnection();
            connection.setRequestMethod("GET");
            connection.setDoInput(true);

            // Process the response if it was successful (HTTP_OK = 200)
```

```
            if (connection.getResponseCode() ==
HttpURLConnection.HTTP_OK) {
                InputStreamReader isr = new
                  InputStreamReader(connection.getInputStream());
                Log.i("responde connection", connection.getResponseCode()
                                + "");
            }
            // Close the connection
            connection.disconnect();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        // Return the received Quotation object
        return "OK";
    }

}
```

7.12 Android clase IluminacionActivity.java

Clase Android para que controla la actividad Iluminación.

```
public class IluminacionActivity extends AppCompatActivity implements
AsyncResponse {

    protected GetDataSensor getDataSensor = new GetDataSensor();
    protected Datos datos = new Datos();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_iluminacion);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);// activa
el boton 'atras' en la barra superior

        getDataSensor.delegate=this;


getDataSensor.execute("http://rodrigodomotica.ddns.net/getAndroid.php?pas
s=1234");

    }

    public void processFinish(String output)
    {
```

```
        TextView ultActualizacion = (TextView)
findViewById(R.id.luzUltimaActualizacion);

        try {
            JSONObject json = new JSONObject(output);

            if (json.getString("Led").equals("1")) datos.setLed(true);
            else datos.setLed(false);

            if (json.getString("Luz").equals("1")) datos.setLuz(true);
            else datos.setLuz(false);

            ultActualizacion.setText("Última actualización:
"+json.getString("Tiempo"));

        } catch (JSONException e) {
            e.printStackTrace();
        }
        inicializarValores(datos.isLed(),datos.isLuz(),
datos.getFechaLuz());

    }

    public void inicializarValores(boolean led, boolean luz, String
fecha)
    {
        ImageView ledIV = (ImageView) findViewById(R.id.IVled);
        ImageView luzIV = (ImageView) findViewById(R.id.IVluz);

        if (led)
ledIV.setImageDrawable(getResources().getDrawable(R.mipmap.bombilla_encen
dida_redonda));
        else
ledIV.setImageDrawable(getResources().getDrawable(R.mipmap.bombilla_apaga
da_redonda));

        if (luz)
luzIV.setImageDrawable(getResources().getDrawable(R.mipmap.bombilla_encen
dida_redonda));
        else
luzIV.setImageDrawable(getResources().getDrawable(R.mipmap.bombilla_apaga
da_redonda));
    }

    public void onClickBombilla(View v){

        ImageView image=(ImageView) v;
        PostDataSensor postDataSensor = new PostDataSensor();
```

```
        if(datos.isLed())
        {

image.setImageDrawable(getResources().getDrawable(R.mipmap.bombilla_apaga
da_redonda));
            datos.setLed(false);

postDataSensor.execute("http://rodrigodomotica.ddns.net/postLuzAndroid.ph
p?pass=1234&Led=0");
        }else
        {

image.setImageDrawable(getResources().getDrawable(R.mipmap.bombilla_encen
dida_redonda));
            datos.setLed(true);

postDataSensor.execute("http://rodrigodomotica.ddns.net/postLuzAndroid.ph
p?pass=1234&Led=1");

        }
    }
}
```

7.13 Android clase TemperaturaActivity.java

Clase Android para que controla la actividad Temperatura (Es prácticamente igual que la de Humedad por lo que no será incluida).

```
public class TemperaturaActivity extends AppCompatActivity  implements
AsyncResponse{

    protected GetDataSensor getDataSensor = new GetDataSensor();

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_temperatura);

        getDataSensor.delegate=this;
      getDataSensor.execute("http://rodrigodomotica.ddns.net/getAndroid.p
                  hp?pass=1234");
    }

    public void processFinish(String output)
    {
        TextView temperatura = (TextView)
                  findViewById(R.id.temperaturaMostrar);
```

```
        TextView ultActualizacion = (TextView)
                findViewById(R.id.tempUltimaActualizacion);

        try {
            JSONObject json = new JSONObject(output);

            temperatura.setText(json.getString("Temperatura")+" ºC");

            ultActualizacion.setText("Última actualización:
                    "+json.getString("TiempoTemp"));

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

7.14 Android clase GetWeatherAsyncTask.java

Clase Android para realizar la consulta asíncrona al servicio meteorológico.

```
public class GetWeatherAsyncTask extends AsyncTask<Void, Void, Weather> {

    private WeatherActivity parent = null;

    public void setParent(WeatherActivity parent) {
        this.parent = parent;
    }

    Weather wobject = new Weather();

    @Override
    protected Weather doInBackground(Void... params) {

        String ciudad=this.parent.getUbicacion();
        Uri.Builder uriBuilder = new Uri.Builder();

        uriBuilder.scheme("http");
        uriBuilder.authority("api.openweathermap.org");
        uriBuilder.appendPath("data");
        uriBuilder.appendPath("2.5");
        uriBuilder.appendPath("weather");
        uriBuilder.appendQueryParameter("q", ciudad);
        uriBuilder.appendQueryParameter("APPID",
                "a4d47e1f013caa3f3718785b5c8b9dab");

        Log.d("url", uriBuilder.toString());
```

```
try {
    // Creates a new URL from the URI
    URL url = new URL(uriBuilder.build().toString());

    // Get a connection to the web service
    HttpURLConnection connection = (HttpURLConnection)
              url.openConnection();
    connection.setRequestMethod("GET");
    connection.setDoInput(true);

    // Process the response if it was successful (HTTP_OK = 200)
    if (connection.getResponseCode() == HttpURLConnection.HTTP_OK) {

        InputStreamReader isr = new
              InputStreamReader(connection.getInputStream());
        Log.i("responde connection",
              connection.getResponseCode()+"");

        BufferedReader br = new BufferedReader(isr);
        String lectura = br.readLine();
        Log.i("LECTURA", lectura);

    try{

        JSONObject result = new JSONObject(lectura);
        String temp = result.getJSONObject("main").getString("temp");
        String humidity =
              result.getJSONObject("main").getString("humidity");
        String viento =
              result.getJSONObject("wind").getString("speed");
        JSONArray array = result.getJSONArray("weather");
        String desc =
              array.getJSONObject(0).getString("description");

        Double gradosK = Double.parseDouble(temp);
        Double gradosC = (gradosK-273.15);
        wobject.setTemperatura(round(gradosC,2));
        wobject.setHumedad(Double.parseDouble(humidity));
        wobject.setViento(Double.parseDouble(viento));
        wobject.setPrecipitaciones(desc);

} catch(Exception e){e.printStackTrace();}

        // Close the input channel
        isr.close();
    }
```

```
            // Close the connection
            connection.disconnect();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        // Return the received Quotation object
        return wobject;
    }

        @Override
        protected void onPostExecute(Weather param) {
            // Quotation received
            if (param != null) {
                this.parent.gotWeather(param.getTemperatura(),
param.getHumedad(),param.getPrecipitaciones(),param.getViento());
            }
            // No response received
            else {
                this.parent.resetWeather();
            }
            super.onPostExecute(param);
        }

        public static double round(double value, int places) {
            if (places < 0) throw new IllegalArgumentException();

            BigDecimal bd = new BigDecimal(value);
            bd = bd.setScale(places, RoundingMode.HALF_UP);
            return bd.doubleValue();
        }
    }
```