Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

# Red de sensores - Internet de las cosas

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor**: Enrique González Daza

**Tutores**: Juan Luis Posadas-Yagüe

Jose Luis Poza Luján

2014/2015

# Agradecimientos

Gracias a mi familia y amigos, por su paciencia y apoyo en todo momento, a toda la comunidad de Software y Hardware Libre por compartir sus conocimientos con el resto del mundo y a mis tutores el Doctor Juan Luis Posadas Yagüe y el Doctor Jose Luis Poza Luján, por la ayuda y apoyo en la realización de este proyecto.

# Resumen

Este proyecto consiste en el estudio y realización de una red de sensores conectados de forma permanente a internet, los cuales nos ofrecerán una serie de servicios como puede ser lecturas de diversos sensores ofreciéndonos dichos datos para poder procesarlos de la manera deseada o permitirnos el uso de actuadores para poder interactuar de forma remota con el mundo real.

Esto sería muy útil, por ejemplo en entornos industriales donde podríamos disponer de datos referentes a la temperatura, humedad de una nave o de seguridad, donde obtendríamos datos sobre sensores de presencia, sonidos y otro uso podría ser el de la domótica donde nos podrían ofrecer un sin fin de servicios utilizando el concepto de internet de las cosas (IoT por sus siglas en inglés) haciendo referencia a la interconexión de objetos cotidianos con internet.

Se estudiarán a lo largo del proyecto diferentes herramientas tanto software como hardware disponibles para poder llevarlo a cabo de la manera más simple y clara posible y se realizará una implementación de ejemplo.

Para su realización se hará uso de la plataforma Arduino, de la placa wifi de bajo coste esp8266, de diversos sensores y actuadores, de servicios web REST y del ordenador de placa reducida Raspberry Pi.

**Palabras clave:** Red, sensores, internet de las cosas, esp8266, Arduino, Raspberry Pi, servicios web, REST.

# Tabla de contenidos

# Tabla de ilustraciones

# 1. Introducción

## 1.1. Presentación

En los últimos tiempos han aparecido diversas plataformas de desarrollo tanto hardware como software con un coste muy reducido y con gran documentación y una comunidad muy activa que permiten la realización de proyectos de diverso tipo que hace tiempo serían impensables su realización a pequeña escala.

Tanto la plataforma Arduino como Raspberry Pi disponen de una gran y activa comunidad donde se puede encontrar mucha documentación y ayuda del resto de usuarios, facilitando de esta forma la realización de los proyectos deseados.

De la misma forma el módulo wifi esp8266 también ha generado una gran comunidad en especial por su bajo coste comparado con el resto de alternativas existentes.

A modo de ejemplo se realizará una implementación lo más sencilla posible pero funcional, pensando en su posible posterior ampliación con nuevas características y funciones o a su unión con otros proyectos más específicos.

## 1.2. Motivación

Con el avance actual de la tecnología, es cada vez más posible conseguir que objetos cotidianos estén conectados de forma permanente a internet sin necesidad de la interacción humana y realizando tareas u ofreciendo servicios a un usuario si así se les solicita.

Si al concepto anterior le unimos el modelo de computación distribuida, conseguiríamos una red independiente de objetos conectados a internet que interactúan entre ellos para poder obtener un sistema muy confiable, tolerante a fallos y escalable, el cual nos puede ofrecer una gran cantidad de servicios.

## 1.3. Objetivos

A partir de lo anterior, se plantean como objetivos del proyecto, los siguientes:

- Estudiar e implementar los nodos que formaran parte de la red, de forma que estos nodos leerán una serie de sensores y ofrecerán los resultados a los usuarios, de la misma forma permitirán que los usuarios hagan uso de actuadores.
- Implementar la comunicación con Internet para poder ofrecer los datos obtenidos de los sensores.
- Implementar la interfaz del usuario para poder obtener dichos resultados.

## 1.4. Descripción del documento

El presente documento está organizado por apartados:

- **Apartado 1:** Es el actual apartado donde se realiza una introducción al proyecto.
- **Apartado 2:** Una descripción y estudio de la plataforma Arduino.
- **Apartado 3:** Se describirán y se probarán distintos sensores y actuadores para su inclusión en los nodos.
- **Apartado 4:** Descripción y estudio del módulo esp8266 junto con Arduino.
- **Apartado 5:** Se estudiará y se configurará el miniordenador Raspberry Pi para su uso en el proyecto.
- **Apartado 6:** Estudio e implementación de un servicio web mediante REST.
- **Apartado 7:** Se explicará la implementación utilizando los componentes anteriormente estudiados.
- **Apartado 8:** En este apartado se expondrán las conclusiones del proyecto así como posibles ampliaciones y mejoras realizables en un futuro.

# 2. Arduino

## 2.1. Introducción


*Ilustración 1: Logotipo de Arduino (www.arduino.cc)*

Arduino es una plataforma de electrónica abierta con software y hardware libre y flexible para poder realizar prototipos y desarrollos de forma sencilla y fácil de utilizar.

Se inició en el año 2005 como un proyecto educativo en el instituto IVREA, en Ivrea (Italia), donde uno de sus fundadores Massimo Banzi, daba clases. El nombre viene del Bar di Re Arduino donde MaSSimo Banzi pasaba algunas horas. En su creación también contribuyeron tanto estudiantes como Hernando Barragán (Colombia) y profesores como David Cuartielles (España).

El instituto se vio obligado a cerrar en 2005 y se decidió abrir todo el proyecto a la comunidad para que no se pierda el proyecto y que pudiera evolucionar y mejorar gracias a la colaboración de muchísima gente a nivel mundial.

Gracias al modelo abierto de Arduino han surgido una infinidad de versiones y clones del mismo, ofreciendo un gran abanico de opciones y prestaciones disponibles, incluso pudiendo desarrollar nuestro propio modelo de Arduino para adaptarlo a nuestras necesidades

Se puede decir que Arduino es en realidad está formada por 3 cosas:

- Una placa hardware.
- Un software de desarrollo.
- Un lenguaje de programación.

### 2.1.1. Una placa hardware

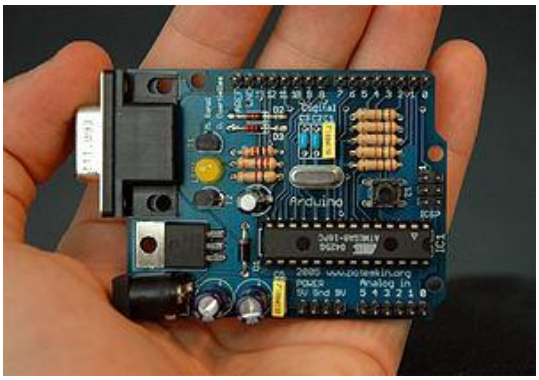

*Ilustración 2: Placa Arduino RS232*

Es una placa de circuito impreso que incorpora un microcontrolador y los pines o puertos necesarios para su utilización, ya sean pines Entrada/Salida para su uso con los sensores como de comunicación RS232 o USB e incluso los de alimentación para su correcto funcionamiento.

El diseño hardware de la placa estaba inspirado en un principio en otra placa de Hardware libre existente, la placa Wiring (http://wiring.org.co)

Los microcontroladores que utilizan las placas arduino son modelos estandar del fabricante Atmel, los cuales vienen preprogramados con un bootloader o gestor de arranque para que podamos utilizarlos de forma sencilla con la plataforma Arduino.

Cada modelo de Arduino dispone de unas características diferentes, ya que pueden tener distintos microcontroladores y distintas características según el uso para el que fueron diseñados.

### 2.1.2. Un software de desarrollo

*Ilustración 3: IDE Arduino*

Es un entorno de desarrollo gratuito, libre y multiplataforma, funciona en entornos Windows, Linux y MacOs, en él se especificará la placa que estamos utilizando, el puerto al que la hemos conectado y se podrán añadir las librerías que deseemos utilizar en nuestro proyecto.

Con este software podemos escribir, compilar y programar la placa Arduino conectando la placa a nuestro ordenador mediante un cable usb que a su vez nos permite alimentar la placa si no queremos utilizar alimentación externa.

También dispone de un monitor serie para poder comunicar nuestro ordenador y la placa con nuestro programa de una forma sencilla.

A pesar de funcionar correctamente, no es un editor de código muy completo y no dispone de características consideradas básicas por muchos desarrolladores como el autocompletado entre otras y por ello permite la opción de poder utilizar nuestro editor preferido (CodeBlocks, Notepad++, Sublime, etc) y utilizar el IDE solo para cargar el programa a la placa.

### 2.1.3. Un lenguaje de programación

Es un lenguaje de programación libre basado en otro lenguaje libre ya existente Processing (https://processing.org), el IDE de Arduino también está basado en él ya que era el lenguaje que utilizaban en el instituto Ivrea.

El lenguaje de Arduino tiene una gran similitud con C++, compartiendo las mismas estructuras de control, sintaxis, operadores, etc. Por lo que si se conoce dicho lenguaje es muy fácil adaptarse a él.

## 2.2. Versiones de Arduino

En este apartado se enumeraran algunos de los modelos oficiales de la placa Arduino así como sus características, los modelos no oficiales y clones son muchísimo, muchos de ellos con características muy destacables y normalmente con un precio más reducido.

**Arduino UNO Rev3**

Placa basada en el microcontrolador ATmega328, se puede decir que es la placa base de la plataforma arduino. Dispone de 14 pines digitales y 6 analógicos

*Ilustración 4: Arduino UNO Rev3*

**Arduino YÚN**

Esta placa dispone de conexión WIFI, se basa en el microcontrolador Atmega32u$ y en el chip Atheros AR9331, dispone de conector microsd y soporta una versión de la distribución Linux OpenWrt.


*Ilustración 5: Arduino YÚN*

**Arduino Nano 3.x**

A pesar de tener un tamaño mucho menor dispone de las mismas características del Arduino UNO, pero utilizando un conectror miniUSB y sin conector para alimentación, al utilizar pines macho se puede insertar perfectamente en una protoboard facilitando las pruebas.

Dado su pequeño tamaño y compatibilidad con el Arduino Uno, será esta versión la que utilice para los Nodos del proyecto.

*Ilustración 6: Arduino Nano 3.x*

**Arduino DUE**

El Arduino DUE es la evolución del Arduino Mega. Es la placa más potente de todas a la vez que nos proporciona una grandísima cantidad de pines, 54 de E/S, por si nuestro proyecto los necesitara y los anteriores modelos no fueran suficiente, el microcontrolador de esta placa dispone de registros de 32 bits, al contrario que el resto de microcontroladores utilizados en Arduino, los cuales tienen registros de 8 bits.


*Ilustración 7: Arduino DUE*

**Tabla comparativa:**

| | UNO | YUN | NANO | DUE |
|---|---|---|---|---|
| **Microcontrolador** | ATmega328 | ATmega32u4 | ATmega328 | AT91SAM3X8E |
| **Voltaje** | 5V | 5V | 5 V | 3.3V |
| **Voltaje Vin** | 7-12V | 5V | 7-12 V | 7-12V |
| **Voltaje (limite)** | 6-20V | 5V | 6-20 V | 6-20V |
| **Pines E/S Digitales** | 14 | 20 | 14 | 54 |
| **Pines PWM** | 6 | 7 | 6 | 12 |
| **Pines E analógicos** | 6 | 12 | 8 | 12 |
| **Corriente pin E/S** | 40 mA | 40 mA | 40 mA | 130 mA |
| **Corriente pin 3.3V** | 50 mA | 50 mA | 40 mA | 800 mA |
| **Memoria Flash** | 32 KB | 32 kB | 32 KB | 512 KB |
| **Tamaño Bootloader** | 0.5 KB | 4 kB | 2 kB | - |
| **SRAM** | 2 KB | 2.5 kB | ATmega328 | 96 KB |
| **EEPROM** | 1 KB | 1 kB | 1 KB | - |
| **Velocidad reloj** | 16 MHz | 16 MHz | 16 MHz | 84 MHz |
| **Alto** | 68.6 mm | 53 mm | 45 mm | 101.52 mm |
| **Ancho** | 53.4 mm | 73 mm | 18 mm | 53.3 mm |
| **Peso** | 25 g | 32 g | 5 g | 36 g |

*Tabla 1: Revisión de características de los Arduinos más empleados*

## 2.3. Hardware

El estudio del Hardware se realizará de la placa Arduino UNO dado que es la Básica y compatible en número de pines y microcontrolador con la placa nano, la placa UNO utiliza un formato DIP (Dual In-line Package) mientras la nano utiliza SMD (Surface Mount Device)

Esta placa hace uso del microcontrolador ATmega328P, la “P” significa que dispone de la tecnología picopower que permite reducir el consumo, tiene una arquitectura AVR desarrollada por Atmel y competencia de otras arquitecturas como puede ser PIC de Microchip

El Pinout completo tanto de la placa UnoR3 como del ATmega328:

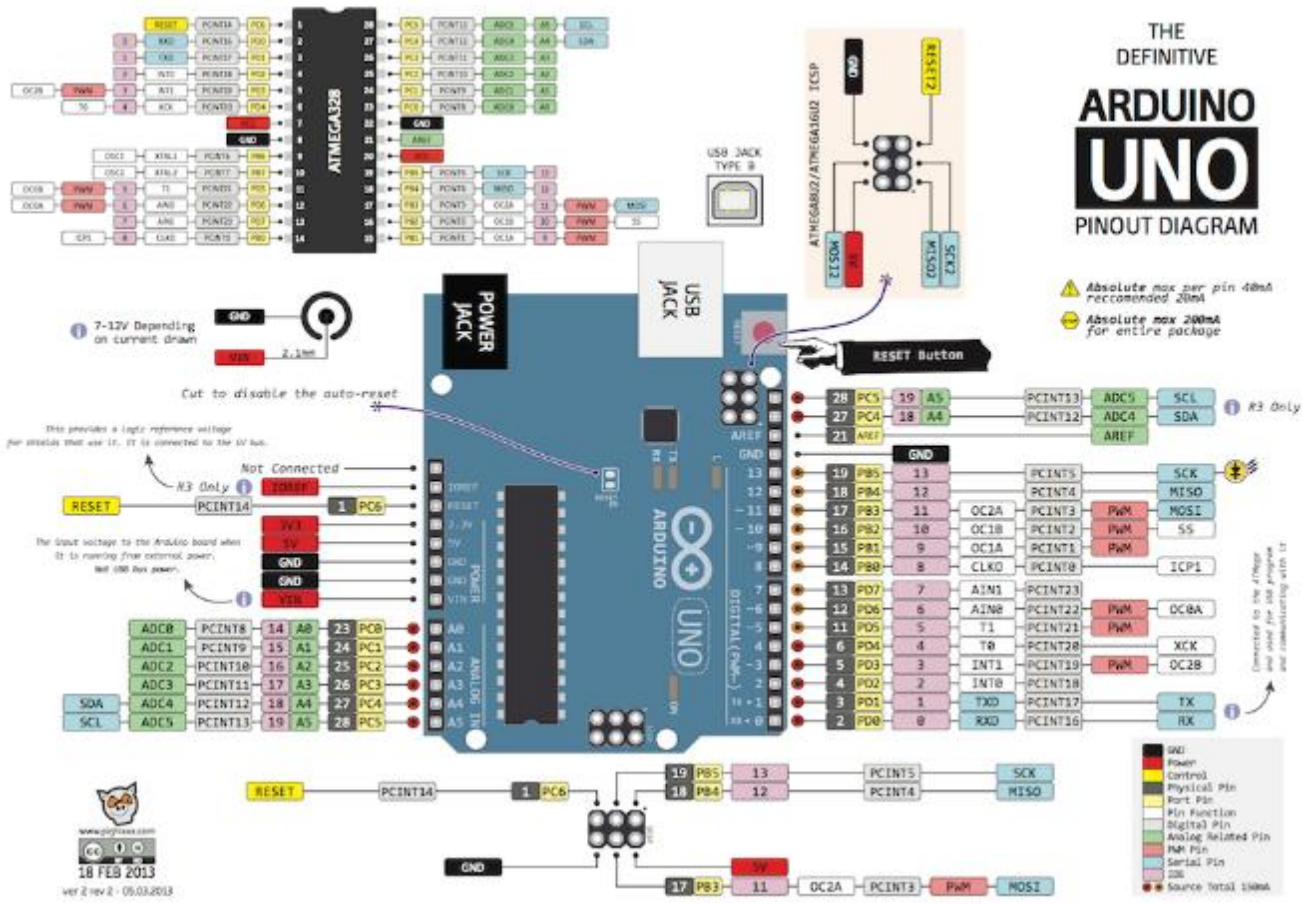


*Tabla 2: Diagrama de pines Arduino UNO*

### 2.3.1. Memorias

**Memoria Flash**: Es una memoria persistente en la cual se carga de forma permanente el programa que se va a ejecutar, hasta que sea sustituido por otro. Dispondremos de 32KB, pero hay que tener en cuenta que no se pueden utilizar por completo ya que 512 bytes son utilizados por el "bootloader block" para cargar el "bootloader".

**Memoria SRAM**: Esta es una memoria volátil donde se almacenarán los datos y variables que el programa utiliza en tiempo de ejecución. Su valor será eliminado cuando se pierda la alimentación o se reinicie la placa. Disponemos de 2KB.

**Memoria EEPROM**: Es una memoria persistente donde podemos almacenar los datos que deseamos conservar aunque se pierda la alimentación de la placa y así poder utilizar en siguientes ejecuciones. Se dispone de 1KB.

Se puede ampliar la cantidad de memoria tanto SRAM como EEPROM conectando módulos de las mismas y comunicándolas con el microcontrolador mediante algún protocolo de comunicación, como pueda ser $I^2C$ por ejemplo. En el caso de la memoria persistente, también se pueden incluir otro tipo de memoria como pueda ser una tarjeta SD (Secure Digital) y mediante el correspondiente módulo comunicarla con el microcontrolador.

### 2.3.2. Protocolos de comunicación

En la placa Arduino UNO disponemos de dos protocolos de comunicación:

**I2C** (Inter-Integrated Circuit): Es un estándar de comunicación, se caracteriza por utilizar solo dos líneas para realizar la comunicación, una línea SDA para transferir los datos y otra línea SCL mediante la cual se envía una señal de reloj para coordinar la comunicación, también hacen falta don líneas más la de alimentación y el común pero se presuponen en el circuito. El pin de la señal SDA es el 27 y el de la señal SCL el 28.

**SPI** (Serial Peripheral Interface): También es un estándar, un maestro se puede comunicar con varios esclavos, éste protocolo necesita 4 líneas para realizar la comunicación. SCK es la señal de reloj. SS una línea diferente para cada esclavo con la que se decide que esclavo activar MOSI y MISO son líneas de datos que son compartidas por todos los esclavos al igual que SCK, solo hará uso de ellas el seleccionado por la línea SS. Los pines correspondientes son SS el 16, MOSI el 17, MISO el 18 y el SDK es el 19, para controlar más esclavos solo hay que utilizar cualquier pin digital como SS, uno para cada esclavo.

### 2.3.3. Alimentación

El voltaje de funcionamiento de la placa es de 5v DC, la placa se puede alimentar de varias maneras:

- Conectando la placa mediante el conector de tipo Jack de 2,1mm a una fuente externa de entre 7v y 12v, aunque en teoría estaría preparada para soportar tensiones de entre 6v y 20v. La polaridad tiene que ser positivo en el centro. El regulador que dispone la placa transformará el voltaje a los 5v necesarios para su funcionamiento.

- También se puede conectar la placa al USB del ordenador, la placa se alimentará de las 5v que proporciona el USB, nos dará 500mA, hay que tener en cuenta que el USB puede no ser capaz de proporcionar la corriente necesaria para el circuito, por ejemplo hay que alimentar el módulo esp8266 de forma externa ya que no es capaz de proporcionar la corriente necesaria para su uso.

- Pines de alimentación de la placa:
    - **GND**: Pin común conectado a tierra.
    - **Vin**: Hace la misma función que el pin central del conector Jack pudiendo alimentar la placa de la misma manera que con dicho conector.
    - **5v**: Se puede utilizar para alimentar la placa mediante una fuente regulada de 5v o también si se alimenta la placa por otro de los medios

disponibles ofrecer 5v regulados con 40mA para conectar cualquier elemento.

- **3.3v**: Este pin ofrece un voltaje regulado de 3.3v y 50mA sea cual sea el método de alimentación de la placa

### 2.3.4. Comunicación USB-Serie

Para la comunicación con el ordenador el Arduino UNO R3 hace uso de otro microcontrolador, el ATmega16U2 para que realice la conversión de USB a TTL-UART serie que incorpora el ATmega328P, de esta forma se nos crea una especie de puerto serie virtual para la comunicación USB-Arduino.

Los pines digitales 0 (RX) y 1 (TX) se utilizan también para realizar la comunicación serie por lo que no podrán ser utilizados como E/S digitales si se está realizando comunicación serie.

En modelos anteriores de Arduino se utilizaba el integrado FT232RL para realizar dicha conversión. En algunos clones del modelo nano, en especial modelos chinos, se incorpora el integrado CH340G para realizar dicha función, este modelo no me ha dado ningún problema utilizando el IDE desde Linux pero en Windows no reconoce el dispositivo por lo que hay que instalar los drivers necesarios para su correcto funcionamiento:

http://wch.cn/download/CH340PCB_ZIP.html

### 2.3.5. Entradas y salidas digitales

Se dispone de 14 entradas y salidas digitales (pines del 0 al 13), funcionan a 5v y pueden trabajar con un máximo de 40mA, también disponen estos pines de una resistencia tipo "pull-up" interna pero que está desconectada, hay que activarla en el código del programa.

### 2.3.6. Entradas analógicas

Disponemos de 6 entradas analógicas (pines del A0 al A5) que son capaces de recibir voltajes entre 0v y 5v, estos pines van a un conversor analógico digital en la placa de 6 canales y 10 bits de resolución por canal por lo que dispondremos de una precisión de $2^{10}$ lo que es lo mismo 1024 que corresponderán 0 a 0v y 1023 a 5v.

### 2.3.7. Salidas analógicas PWM

La placa Arduino no dispone de salidas analógicas propiamente dichas pero lo simula utilizando unas cuantas salidas digitales, estas salidas están etiquetadas con PWM (Pulse Width Modulation) y son las que corresponden a los pines 3, 5, 6, 9, 10 y 11.

Esto se consigue emitiendo una señal de pulsos cuadrada con una frecuencia constante, sobre 490Hz, al variar la duración de los pulsos se varía también la tensión promedio resultante, a pulsos más cortos, menor tensión promedio y viceversa.

Los pines PWM tienen una resolución de 8 bits por lo que con valor 0 tendríamos 0v y con valor 255 los 5v deseados.

También hay que tener los pines PWM están controlados por 3 temporizadores diferentes:

- Pines 3 y 11 controlados por Timer1
- Pines 5 y 6 controlados por Timer2
- Pines 9 y 10 controlados por Timer3

Hay que tener en cuenta los Timers ya que en ocasiones distintas librerías hacen uso del mismo Timer produciendo efectos no deseados en el funcionamiento del circuito, como me ocurrió al utilizar el servo y la librería SoftwareSerial.h, se solucionó utilizando la librería PWMServo.h ( http://arduiniana.org/libraries/pwmservo/ ) dado que utiliza otro Timer distinto.

### 2.3.8. Interrupciones

Se puede hacer uso de interrupciones hardware con Arduino UNO, para ello se utilizan los pines 2 y 3 pero también se pueden utilizar el resto de pines digitales usando interrupcions por software mediante el uso de librerías como por ejemplo:

https://github.com/GreyGnome/EnableInterrupt

### 2.3.9. Pines varios

- AEREF: Voltaje de referencia externo para aumentar la precisión de las entradas analógicas.
- IOREF: indica el voltaje al que trabajan los pines de entrada/salida
- RESET: Si se pone este pin a BAJO, el microcontrolador se reiniciara de la misma forma que si se pulsara el botón de RESET.

## 2.4. Lenguaje de programación

El lenguaje de programación de Arduino, como se ha indicado anteriormente es una versión simplificada de C/C++, los programas de Arduino de denominan Sketch tiene extensión ".ino" y una vez compilados se convierten a código binario AVR.

El programa se verifica y se carga en el Arduino utilizando los botones del IDE:

**Características:**

- Es un lenguaje case-sensitive (diferencia entre mayúsculas y minúsculas)
- Todas las instrucciones terminan con un punto y coma.
- Un programa o sketch de Arduino se compone de tres secciones:
    - **Sección de declaraciones**

      En esta sección se declararán las variables globales del programa y también se incluirán las librerías y ficheros que utilicemos en el mismo.

    - **Sección "void setup() { … }"**

      Esta sección solo se ejecutará una vez cuando se alimente la placa o cuando se reinicie la misma. Se ejecutarán una sola vez todas las instrucciones dentro de la función setup() estas instrucciones incluirán las inicializaciones de las variables y librerías necesarias, así como también se establecerán el estado de los Pins de la placa.

    - **Sección "void loop() { … }"**

      Justo tras terminar la sección setup() se ejecutarán las instrucciones contenidas dentro de la función loop() de forma continua e infinitas veces hasta que la placa sea desconectada o reseteada.

El lenguaje de programación de la plataforma Arduino es sencillo y prácticamente igual a C/C++ por lo que cualquiera que tenga conocimientos de programación no tendrá excesivas dificultades en empezar a desarrollar programas para esta plataforma, se ha incluido un anexo con la estructura de dicho lenguaje para su mayo comprensión.

# 3. Sensores y Actuadores

## 3.1. Introducción

En esta sección se estudiarán y se implementarán los programas y circuitos necesarios para el correcto uso de una serie de sensores y actuadores, de forma que más tarde se puedan integrar en los nodos de nuestra red de sensores, se intentará que los circuitos y programas comiencen siendo sencillos y se incremente su complejidad, conforme aumente el conocimiento y la práctica con Arduino y los sensores mismos. Tanto el código de los programas como los circuitos en formato '.fzz' se incluirán en los archivos adjuntos.

La plataforma Arduino y su comunidad nos proporciona una gran cantidad de ejemplos y documentación para poder controlar dichos sensores, el código correspondiente a los siguientes ejemplos se puede encontrar en el Anexo.

Para representar los circuitos de forma sencilla y reproducible se utilizará el programa 'Fritzing' que nos permite diseñar los circuitos de una forma visual e intuitiva. Fritzing es una iniciativa de sortware libre creada bajo los principios de Processing y Arduino, también es una herramienta multiplataforma y dispone de una gran comunidad de usuarios que ayuda a su evolución y mejora continua.

*Ilustración 8: Logotipo Fritzing*

## 3.2. Led

Este puede ser el circuito y programa más simple, consiste en conectar un led a una salida digital y cambiar el estado de dicha salida para que se encienda o se apague dicho led. Hay que recordar incluir una resistencia o de otra forma podríamos dañar la placa Arduino.

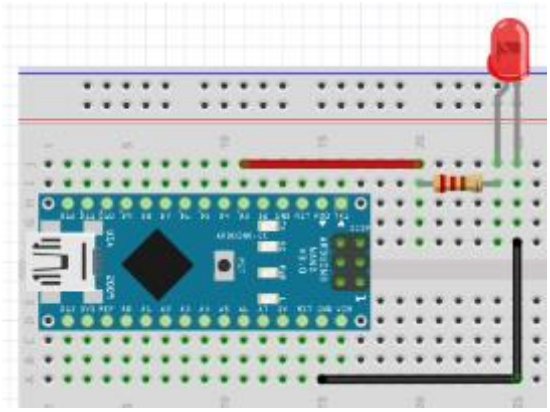

*Ilustración 9: Circuito led*

7

### 3.3. Buzzer

Este es un circuito similar al anterior pero en este caso vamos a hacer uso de las salidas analógicas, es decir de las salidas PWM, para hacer vibrar a cierta frecuencia el buzzer piezoeléctrico y así conseguir un sonido audible.

De esta sencilla forma podemos disponer de una alarma para nuestros nodos.

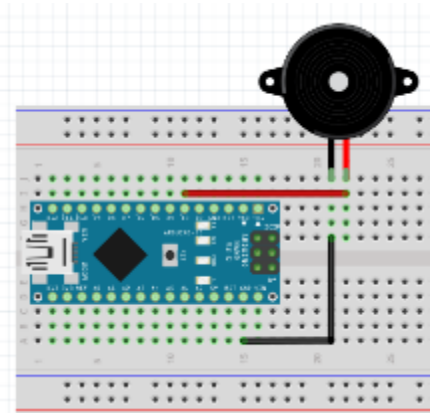

*Ilustración 10: Circuito buzzer*

## 3.4. Sensor de movimiento

Para este circuito se utilizará un sensor de movimiento PIR (Passive Infrared), este sensor dispone de componente que detecta los cambios de radiación infraroja recibida como pueda ser el calor que generamos los seres humanos o los animales, el circuito de este módulo activa uno de los pines a 5v en el momento que detecte un cambio.

Este módulo hay que alimentarlo a 5v por lo que podemos utilizar los pines del mismo arduino para realizar esto, también dispone de 2 potenciómetros para calibrar su sensibilidad.

En el programa mostraremos por el puerto serie un mensaje alertando del movimiento detectado, así hacemos uso de él, es una buena forma para hacer debug a nuestro programa, para poder verlo hay que seleccionar en el IDE de Arduino ‘Herramientas > Monitor Serie’, hay que comprobar que la velocidad del puerto corresponda a la que hemos definido en el programa o lo se recibirán los datos correctos.

Uniendo este programa al anterior del buzzer dispondríamos de una alarma casera.

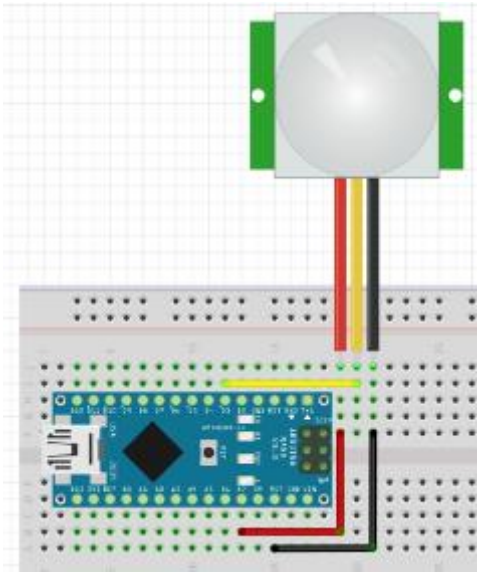

*Ilustración 11: Circuito sensor de movimiento*

### 3.5. Servo

El siguiente elemento a probar será un servo, el servo necesita de las señales PWM para poder cambiar el ángulo del brazo, entre 0º y 180º, este elemento que nos brinda una gran cantidad de posibilidades para actuar en el mundo real, podríamos por ejemplo mover una cerradura o la orientación de una cámara.

El circuito es muy sencillo como los anteriores, se alimenta el servo a 5v y mediante la señal PWM del pin 3 se controla, esta vez en lugar de escribir el Arduino en el puerto serie, seremos nosotros los que escribiremos en dicho puerto el ángulo que deseamos que tenga el brazo del servo, se hará uso de la librería servo.h.

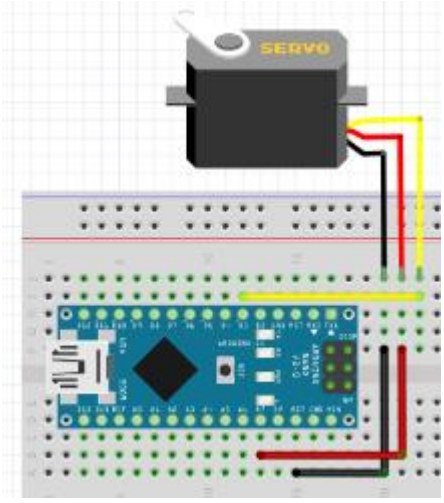


*Ilustración 12: Circuito servo*

### 3.6. Relé

Otro componente de gran utilidad para actuar con el mundo real es el relé el cual nos permite tener control de otros circuitos de Voltaje superior al que admite Arduino como puede ser el control de las luces de nuestra casa, podríamos utilizarlo de interruptor automático si lo unimos al módulo de sensor de movimiento por ejemplo.

El relé se alimenta también a 5v, y se controla con una sola señal en este caso conectada al pin 3, dispone de un conector común y otros dos conectores que son normalmente abierto y normalmente cerrado respectivamente, cuando cambiemos el valor del pin 3 se cambiarán el valor de estas 2 salidas

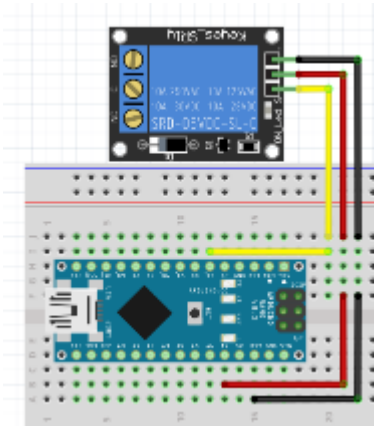

*Ilustración 13: Circuito relé*

### 3.7. Temperatura y humedad

Existen sensores específicos para conocer la temperatura y otros para conocer la humedad respectivamente, en este caso se utilizará un sensor que nos proporcionará estos dos valores él solo, este sensor es el DHT11, es un sensor muy barato con el que podemos medir la temperatura entre 0º y 50º C con una precisión de ±2°C y la humedad entre 20% y 80% con una precisión de 5%

También se alimenta a 5v este sensor, el pin de señal será el 3 que se conecta al 2 del sensor y hace falta una resistencia pull up de 10k entre el pin 3 y el pin 2 del sensor

Se utilizará la librería de Adafruit para esta familia de sensores:

https://github.com/adafruit/DHT-sensor-library

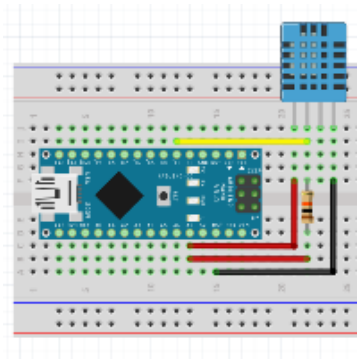

*Ilustración 14: Circuito temperatura y humedad*

# 4. Esp8266

## 4.1. Introducción

Para poder conectar las placas Arduino a Internet existen muchas posibilidades, incluidas placas oficiales de Arduino para realizar esta tarea, tanto por wifi como por cable Rj45, con un funcionamiento muy correcto como por ejemplo las siguientes dos placas:

*Ilustración 15: Arduino Ethernet Shield*

*Ilustración 16: Arduino Wifi Shield*

El único pero que pueden tener estos módulos es que tienen un precio un poco elevado 69€ la placa wifi y 29€ la placa Ethernet, en este apartado de estudiará y se utilizará un módulo de reciente aparición y un bajísimo precio, el esp8266 que con un precio cercano a los 3€ es una opción a tener muy en cuenta.

*Ilustración 17: Esp8266 ESP-01*

Hay distintas versiones del módulo pero se utilizará en este proyecto la versión ESP-01 dado que es la que se dispone.

El módulo incluye todo lo necesario para conectarse por WIFI mediante comandos AT utilizando el puerto serie, en verdad es una placa que funciona de forma independiente del Arduino pero mediante comandos AT podemos utilizarla como si el propio Arduino estuviera conectado al WIFI.

### 4.2. Características técnicas

- Protocolos soportados: 802.11 b/g/n
- Wi-Fi Direct (P2p), Soft Access Point
- Pila TCP/IP integrado
- PLL, reguladores y unidades de manejo de energía integrados
- Potencia de salida: +19.5dBm en modo 802.11b
- Sensor de temperatura integrado
- Consumo en modo de baja energía: <10 uA
- Procesador integrado de 32 bits, puede ser utilizado como procesador de aplicaciones
- Wi-Fi 2.4 GHz, soporta WPA/WPA2
- Tamaño ultra reducido (11.5mm x 11.5mm)
- Conversor analógico a digital de 10-bit
- Soporta variedad de antenas
- SDIO 2.0, SPI, UART, I2C
- Encendido y transmisión de datos en menos de 2ms
- Rango de operación -40C° ~ 125C°

### 4.3. Comandos AT

Los comandos utilizables con la placa Esp8266 son:

| Orden AT | Descripcion |
|---|---|
| **AT** | Verifica si funciona devolviendo 'OK' |
| **AT+GMA** | Información firmware |
| **AT+RST** | Reinicia el módulo |
| **ATE1 , ATE0** | Activa o desactiva el eco de los comando AT |
| **AT+GSLP** | Modo reposo |
| **AT+CWMODE** | Modo de funcionamiento. (1)estación, (2)puntoAcceso, (3)mixto. |
| **AT+CIOBAUD** | Velocidad comunicación en baudios |
| **AT+CSYSWDTENABLE** | Activa Watchdog |
| **AT+CSYSWDTDISABLE** | Desactiva Watchdog |
| **AT+CIPSTAMAC** | Establecer o consultar MAC del Punto de acceso |
| **AT+CIPAPAMAC** | Establecer o consultar MAC de la estación |
| **AT+CWLAP** | Consultar redes disponibles y establecer conexión |

| | |
|---|---|
| **AT+CWQAP** | Desconectar punto de acceso |
| **AT+CWSAP** | Configura punto de acceso |
| **AT+CWDHCP** | Activa/Desactiva DHCP en el punto de acceso |
| **AT+CIPSTA** | Establece IP de la estación |
| **AT+CIPAP** | Establece IP del punto de acceso |
| **AT+CWLIF** | IPs conectadas al punto de acceso |
| **AT+CIFSR** | Ip actual del módulo |
| **AT+CIPMUX** | Modo de conexión (0) simple, (1) múltiple |
| **AT+CIPSERVER** | Activa/Desactiva el modo servidor |
| **AT+CIPSTO** | Configura timeout |
| **AT+CIPSTART** | Inicia una conexión |
| **AT+CIPSTATUS** | Muestra estado de una conexión |
| **AT+CIPCLOSE** | Cierra una conexión |
| **AT+CIPSEND** | Prepara el envío de datos |
| **AT+CIPUPDATE** | Se usa para actualizar el módulo |

*Tabla 3: Comandos AT Esp8266*

Hay que tener en cuenta que dependiendo de la versión de firmware del módulo, los comandos AT del mismo pueden variar. Hay muchas versiones de firmware, incluso hay alguna para poder utilizar el lenguaje Lua y así facilitar su uso (https://nodelua.org/). Pero en este Proyecto se utilizará el firmware que viene en el módulo ya preinstalado.

### 4.4. Circuito

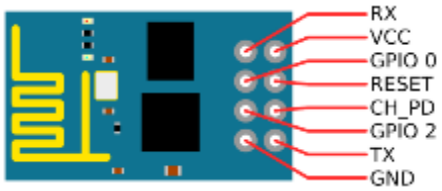


*Ilustración 18: Pines Esp8266*

Para realizar circuito básico para poder utilizar el módulo hay que tener cuidado con la alimentación del módulo ya que es a 3,3v y hay que utilizar una alimentación externa al Arduino dado que éste no parece ser suficiente para alimentar el módulo, por lo que tendremos que unir ambas GNDs para el correcto funcionamiento del mismo.

Para la comunicación con el módulo se utilizará la librería SoftWareSerial.h para poder utilizar otros pines como puerto serie, es este caso utilizaremos los pines 2 y 3 del Arduino y de esta forma poder seguir utilizando el puerto serie del Arduino con nuestro ordenador, que utiliza los pines 0 y 1.

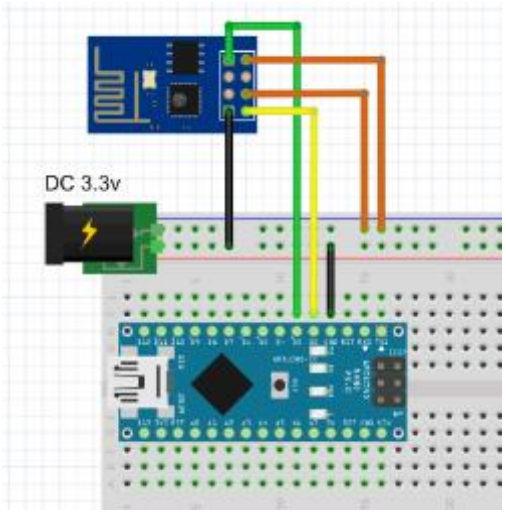


*Ilustración 19: Circuito Esp8266 Básico*

## 4.5. Pruebas

Con el programa *basicoEsp8166.ino* leemos los caracteres del puerto serie del ordenador y lo escribimos en el del módulo esp8266 y viceversa. Cargamos el programa y abrimos el Monitor Serie.

### 4.5.1.Conexión básica:

Comprobamos que funciona el circuito y el módulo escribiendo:

**AT**

Lo que nos responderá:

**OK**

Esto es que se comunican de forma correcta, si aparecieran caracteres extraños u otra cosa probaremos a seleccionar otra velocidad en el Monitor Serie.

**AT+GMR**

Nos devolverá la versión del firmware en mi caso:

**0018000902-AI03**

Escribimos:

**AT+CIOBAUD?**

Y nos devuelve la velocidad de comunicación del puerto, se puede cambiar con AT+CIOBAUD=9600 por ejemplo, ahora escribimos:

**AT+CWMODE?**

Que devuelve el modo en el que trabaja el esp8266, para cambiar el modo de funcionamiento escribiremos AT+CWMODE=1 por ejemplo. Seguimos escribiendo comandos:

**AT+CWLAP**

Nos devolverá una lista de redes que están dentro del alcance del esp8266 con el siguiente formato:

+CWLAP: `ecn,ssid,rssi,mac`

Donde:

- ecn: 0 OPEN , 1 WEP , 2 WPA_PSK , 3 WPA2_PSK , 4 WPA_WPA2_PSK
- ssid: string, SSID del punto de acceso
- rssi: fuerza de la señal
- mac: string, dirección MAC

Para conectarnos a una de las redes listadas anteriormente escribiremos:

**AT+CWJAP="SSID","password"**

y con:

**AT+CIFSR**

Nos devolverá la IP que nos ha sido asignada.

Una vez que nos hemos conectado a una red el módulo por defecto guarda la última conexión y a no ser que indiquemos lo contrario tratará de conectarse de forma automática otra vez a la última red a la que se conectó.

### 4.5.2. Probando modo servidor

Asignamos el modo deseado

**AT+CWMODE=1**

Comprobamos que se realizó correctamente:

**AT+CWMODE?**

Permitiremos conexiones múltiples:

**AT+CIPMUX=1**

Y activaremos el modo servidor en el puerto 80:

**AT+CIPSERVER=1,80**

desde un navegador del ordenador escribimos la IP que nos ha sido asignada y el puerto:

**192.168.1.128:80**

Con lo que obtenemos la petición del navegador por el Monitor Serie:

**Link**
**+IPD,0,308:GET / HTTP/1.1**
**Host: 192.168.1.128**
**User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0**
**Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8**
**Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3**
**Accept-Encoding: gzip, deflate**
**Connection: keep-alive**
**OK**

Y le mandamos la respuesta:

**AT+CIPSEND=0,13**

Le vamos a mandar 13 caracteres por el canal 0, en el monitor serie aparecerá el símbolo > esperando esos 13 caracteres que le hemos indicado, escribimos por ejemplo "Hola Mundo!!!" y tras esto cerramos el canal:

**AT+CIPCLOSE=0**

En el navegador nos aparecerá la respuesta:

**Hola Mundo!!!**

### 4.5.3. Conectando con thingspeak.com


*Ilustración 20: Logotipo ThingSpeak*

ThingSpeak es una plataforma web para el internet de las cosas, en la cual podremos almacenar los datos que consigamos con nuestros sensores como pueda ser la temperatura de una forma muy sencilla y poder procesar esos datos mediante gráficas por ejemplo.

Es Software libre y su API para procesar peticiones HTTP está disponible en un repositorio de GitHub ( https://github.com/iobridge/thingspeak )

Los pasos para utilizarlo consisten en:

1. Nos creamos una cuenta.
2. Creamos un canal, por ejemplo Temperatura.
3. Añadimos un field al canal.
4. Nos darán un Write API Key del canal que utilizaremos para identificar dicho canal, en nuestro caso nos dio: 5G87A0JO4U5WCS2S

Ahora utilizaremos el módulo esp8266 para actualizar el valor del field que hemos creado antes por lo que seguiremos utilizando el circuito anterior y el Monitor Serie como hasta ahora:

Establecemos una conexión con la IP de ThingSpeak mediante el puerto 80:

**AT+CIPSTART=4,"TCP","184.106.153.149",80**

Nos responderá:

**OK**

**Linked**

Indicamos el número de caracteres que vamos a enviar:

**AT+CIPSEND=4,43**

Y escribimos cuando aparezca el símbolo >

**GET /update?key=5G87A0JO4U5WCS2S&field1=5**

Nos responderá:

**SEND OK**

**+IPD,4,1:5**

**OK**

**OK**

**Unlink**

Ahora solo nos falta cerrar la conexión:

**AT+CIPCLOSE**

# 5. Raspberry Pi

## 5.1. Introducción

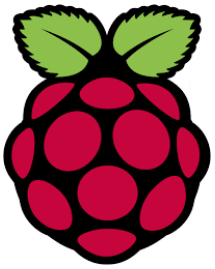

*Ilustración 21: Logotipo Raspberry Pi*

La Raspberry Pi es un ordenador con el tamaño de una tarjeta de crédito, también llamado placa computadora (SBC) y de muy bajo coste (sobre los 35€), es un proyecto que comenzó en 2006, en la universidad de Cambridge para fomentar la enseñanza de las ciencias computacionales en los colegios, en 2012 se comenzó a fabricar y comercializar en serie.

El sistema operativo que utiliza es Linux y dispone de una gran cantidad de distribuciones para ella, gracias a esto un usuario puede utilizarla como si de un PC con Linux se tratara, también dispone de una comunidad de usuarios y desarrolladores que crece sin parar, ya que aparte de su objetivo educacional, la Raspberry Pi también dispone de grandes características multimedia, en especial los últimos modelos, convirtiéndola en una plataforma de entretenimiento muy poderosa y a un precio muy competitivo.

Dispone de entradas y salidas digitales por lo que también se pueden llevar a cabo los proyectos anteriores (Capítulo 3) que no hagan uso de las entradas y salidas analógicas.

## 5.2. Características

| | Raspberry Pi Model B+ | Raspberry Pi 2 Model B |
|---|---|---|
| | | |
| **SoC** | BCM2835 | BCM2836 |
| **Fabricante** | Broadcom | Broadcom |
| **CPU** | ARM1176JZF-S | ARM Cortex-A7 |
| **Instrucciones** | ARMv6 | ARMv7 |
| **Cores** | Single-core | Quad-core |
| **Velocidad** | 700MHz | 900MHz |
| **RAM** | 512MB | 1GB |
| **Almacenamiento** | MicroSD slot | MicroSD slot |
| **GPU** | 250MHz Broadcom VideoCore IV | 250MHz Broadcom VideoCore IV |

| | | |
|---|---|---|
| **Conexiones** | HDMI<br>4x USB2 ports<br>10/100 Ethernet<br>40 GPIO pins<br>MIPI camera connector<br>MIPI display DSI<br>Vídeo compuesto (PAL y NTSC) vía 3.5 mm TRRS jack compartido con audio estéreo | HDMI<br>4x USB2 ports<br>10/100 Ethernet<br>40 GPIO pins<br>MIPI camera connector<br>MIPI display DSI<br>Vídeo compuesto (PAL y NTSC) vía 3.5 mm TRRS jack compartido con audio estéreo |
| | Largo: 8.6cm<br>Ancho: 5.7cm<br>Peso: 45g | Largo: 8.6cm<br>Ancho: 5.7cm<br>Peso: 45g |
| **Alimentación** | 5 V a 2A micro USB | 5 V a 2A micro USB |
| **Precio** | 30€ | 40€ |

*Tabla 4: Características de las últimas  Raspberry Pi*

*Ilustración 22: Raspberry Pi 2*

Desde que comenzó la fabricación de esta placa se han sucedido distintas revisiones de la misma mejorando con cada una de ellas las características o prestaciones de su antecesora de su antecesora sin aumentar en gran medida su precio, se pueden observar las características de las 2 últimas versiones en la anterior tabla.

### 5.3. Distribuciones

Para Raspberry PI existen una gran cantidad de distribuciones, cada una con sus propias características y peculiaridades como por ejemplo OpenElec que está principalmente pensada para convertir nuestra Raspberry Pi en un completo centro multimedia. En su mayoría son adaptaciones de las distribuciones que podemos disfrutar en PC pero optimizadas para su uso en esta plataforma.

Dado que el sistema de ficheros se guarda en una tarjeta SD, es muy fácil tener varias distribuciones en distintas tarjetas para poder cambiar entre ellas de una manera simple, según el uso que queramos dar a la tarjeta en cada momento.

Incluso se dispone de una versión de Windows10 para su uso con la última versión de la placa la Raspberry Pi 2. Esta versión está pensada para su uso en el internet de las cosas.

La distribución que cuenta con el apoyo oficial y se puede decir que es la más estable y optimizada a nivel general el Raspbian OS que se basa en la distribución Debian Wheezy (Debian 7.0), esta será la que utilice en este proyecto.

*Ilustración 23: Distribuciones Raspberry Pi*

## 5.4. Instalación

La instalación la realizaré en una Raspberry Pi 2, para la conexión a la red en lugar de utilizar el conector RJ45 que trae la propia tarjeta se utilizará una tarjeta wifi-usb para facilitar el acceso y posterior colocación de la Raspberry PI. Para poder instalar la distribución necesitaremos una tarjeta microSD de como mínimo 4GB de capacidad.

Los pasos para instalar la distribución Raspbian OS son los siguientes:

1. Descargamos la ultima imagen oficial de Raspbian:
   http://downloads.raspberrypi.org/raspbian_latest

2. Copiamos la imagen a la sd:
   ```
   dd if=/ruta/descarga/zip/2015-05-05-raspbian-wheezy.img
   of=/dev/sdc bs=1M
   ```

   Si utilizamos un sistema Windows podemos utilizar por ejemplo el programa Win32DiskImager (http://sourceforge.net/projects/win32diskimager/) para realizar dicha copia

- Conectamos la placa a un monitor por HDMI, un teclado, la tarjeta wifi-usb, insertamos la tarjeta microSD y conectamos la alimentación.

- Acto seguido nos aparecerá un menú

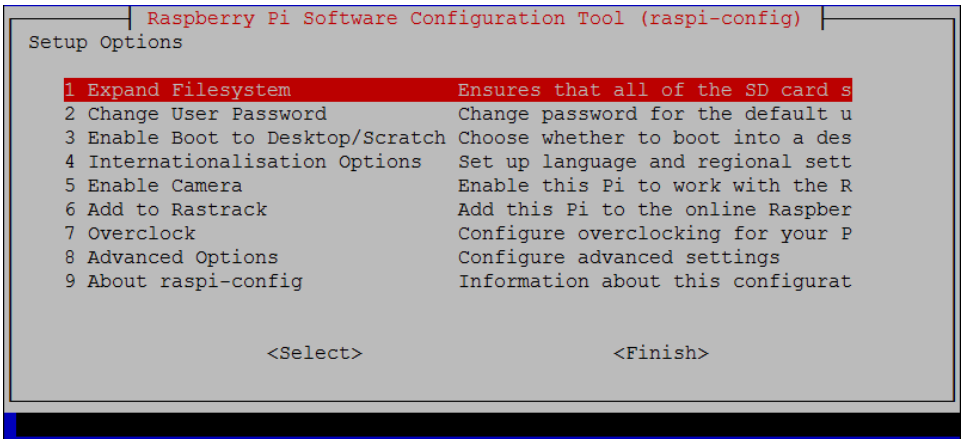


*Ilustración 24: Menú configuración Raspberry Pi*

- En el menú que aparece seleccionamos:

1.Expand Filesystem
Para que el sistema de ficheros utilice todo el espacio disponible de nuestra tarjeta microSD.

2. Change User Password
Se introduce la contraseña del usuario "pi" en este caso como contraseña introduciremos "passwordpi".

4. Internationalisation Options
Son las opciones para poder cambiar el idioma al sistema:
4.1 Change Locale
seleccionamos es_ES.UTF8 UTF-8
y seleccionamos otra vez es_ES.UTF8
4.2 Change Timezone
Seleccionamos la zona horaria:
Europe
Madrid
4.3 Change Keyboard Layout
Seleccionar tipo y distribución de teclado:
Generic 105-key (Intl) PC
Spanish
Spanish
Por defecto
No compose key
Yes (para reiniciar las X con Ctr+Alt+Del)
8.Advanced Options
8.2 Hostname
Para cambiar el nombre al server.
8.3 Memory Split
Si se quiere asignar más memoria a la GPU.
8.4 SSH
Para habilitar el servidor SSH (Secure SHell)

Para conectarnos a nuestra red wifi se edita el fichero:

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

y añadiremos al final del mismo:

```
network={
    ssid="NombreDeLaRed"
    psk="PasswordDeLaRed"
}
```

La IP nos la asignará el router, en mi caso se asignara la IP 192.168.0.254

**NOTA:** Hay que tener precaución y utilizar una fuente de alimentación de calidad con la placa. Se recomienda como mínimo que tenga 1A de corriente por seguridad, ya que de no ser así podría no funcionar correctamente e incluso se corromper la tarjeta SD.

### 5.5. Resumen

Gracias a esta placa y a estos sencillos pasos disponemos en este momento de un ordenador con una distribución Linux plenamente funcional a la cual podemos acceder incluso remotamente mediante SSH, y ya no necesitamos tener conectada la placa a ningún monitor ni tan siquiera un teclado.

# 6. Servicios Web REST

## 6.1. Introducción

REST (Representational State Transfer) o Transferencia de Estado Representacional es un estilo de programación de servicios web centrado en los recursos y en sus representaciones sin las abstracciones de los protocolos basados en patrones de intercambio de mensajes, como por ejemplo SOAP.

Una de las características es que es un protocolo cliente/servidor sin estado, toda información necesaria tiene que estar en la propia consulta http

El acceso a los recursos se hace mediante los métodos estándar de http:

- PUT – Modificar un recurso
- GET – Leer un recurso
- POST – Añadir un recurso
- DELETE – Borrar un recurso

Los recursos se identifican mediante identificadores de recursos uniformes URI (Uniform Resource Identifier) y un recurso puede tener diferentes representaciones (texto, xml, json…)

## 6.2. Flask


*Ilustración 25: Logotipo Flask*

Flask es un microframework desarrollado por Armin Ronacher que nos permite crear de una forma muy sencilla aplicaciones web utilizando el lenguaje de programación Python con muy pocas líneas de código.

Dada la sencillez de su construcción y la gran cantidad de documentación así como su agilidad de desarrollo será el entorno utilizado para realizar nuestra api REST.

Características:

- Servidor de desarrollo y depuración
- Soporte para pruebas unitarias integradas
- Despachador de solicitudes RESTful
- Uso de templates Jinja2
- Soporte para Cookies de seguridad (sesiones del lado del cliente)
- 100% compatible con WSGI 1.0
- Basado en Unicode
- Documentación amplia

### 6.2.1 Instalación

Su instalación es muy sencilla, abriremos una sesión en nuestra Raspberry PI y ejecutaremos en el terminal:

```
$ sudo pip install flask
```

"Pip" es un instalador de paquetes para Python ( https://pip.pypa.io/en/stable )

Para comprobar que lo hemos instalado correctamente escribiremos un pequeño programa:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hola Mundo!!!

if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

Lo guardaremos con el nombre "hola.py" y lo ejecutaremos con la orden:

```
$ python hello.py
```

Nos aparecerá el mensaje:

* Running on http://0.0.0.0:5000/

Si por ejemplo desde el ordenador ahora ponemos desde el navegador la dirección de nuestro servidor y el puerto indicado, en nuestro caso será http://192.168.1.254:5000, tras esto nos aparecerá el mensaje "Hola Mundo!!!"

De momento todo parece funcionar de forma correcta.

### 6.2.2. API REST lecturas

Lo primero que tendremos que definir que nuestro modelo de datos para las lecturas, siguiendo la premisa de implementarlo de la forma más sencilla posible cada una de las lecturas dispondrá de los siguientes campos:

| Nombre campo | Tipo de dato | Descripción |
|---|---|---|
| **id** | entero | Id del sensor |
| **fecha** | timestamp | Fecha de la creación o de la última modificación |
| **tipo** | texto | Descripción del sensor o actuador |
| **valor** | float | Valor actual del sensor |
| **idnodo** | entero | Id del nodo al que pertenece dicho sensor |

*Tabla 5: Modelo de datos a utilizar.*

Con este formato podemos acceder a cada sensor que tengamos en nuestra red de Nodos, sin necesidad de conocer a que nodo pertenece, si quisiéramos conocer los datos de todos los sensores de los que dispone un nodo, también sería sencillo ya que los podríamos identificar fácilmente mediante el campo 'idnodo'.

Los métodos HTTP de los que dispondrán nuestros recursos serán:

| Método HTTP | URI | Acción |
|---|---|---|
| **GET** | http://192.168.1.254/lecturas | Leerá todas las lecturas |
| **GET** | http://192.168.1.254/lecturas /id_lectura | Leerá la lectura con id=id_lectura |
| **PUT** | http://192.168.1.254/lecturas /id_lectura | Modificará la lectura con id=id_lectura |
| **POST** | http://192.168.1.254/lecturas /id_lectura | Añadirá una lectura con id=id_lectura |
| **DELETE** | http://192.168.1.254/lecturas /id_lectura | Eliminará la lectura con id=id_lectura |
| **GET** | http://192.168.1.254/lecturasnodo/id_nodo | Leerá todas las lecturas del nodo id_nodo |

*Tabla 6: Servicios a implementar.*

La información se representará en formato json para que pueda ser fácilmente accesible y manipulable esa información por cualquier futura aplicación, por ejemplo:

Al realizar un GET a http://192.168.1.254/lecturas /1

Obtendríamos como resultado:

```
{
  "lectura1": [
    {
      "fecha": "Sun, 10 Aug 2015 00:51:21 GMT",
      "id": 1,
      "idnodo": 1,
      "tipo": "temperatura",
      "valor": 30.0
    }
  ]
}
```

La implementación de los servicios REST se realizará en el apartado siguiente.

# 7. Implementación

## 7.1. Introducción

Hasta ahora hemos estudiado, comprobado y definido el funcionamiento de los componentes que forman parte de nuestro proyecto, en este apartado completaremos su finalización uniendo cada una de esas partes para que interactúen entre ellas.

Como se ha podido ver el sistema se puede dividir en dos partes, compuestas a su vez en:

- Servidor:
    - Raspberry Pi
    - Raspbian OS
    - Flask
- Nodos:
    - Arduino
    - Sensores y actuadores
    - Esp8266

Para evitar conflictos con las direcciones IP, serán a signadas por el router según la mac de cada adaptador de red, siempre será la misma a cada componente del sistema:

- Router: 192.168.1.1
- Servidor: 192.168.1.254
- Nodo1: 192.168.1.130
- Nodo2: 192.168.1.131
- NodoX: 192.168.1.(129 + X)

También tendremos que seleccionar como se almacenarán los datos en nuestro servidor para poder acceder más tarde a ellos, como siempre continuaremos con la premisa de realizar dicha tarea de la forma más sencilla posible.

Por último se implementará un Nodo plenamente funcional, con varios sensores o actuadores y un pequeño programa a modo de cliente para que podamos comprobar el correcto funcionamiento del sistema.

## 7.2. Servidor


*Ilustración 26: Logotipos Servidor*

Como ya se ha adelantado en el punto anterior tenemos que poder almacenar los datos de los sensores que nos envíen los nodos, en este caso vamos a hacer uso de una base de datos para lo cual haremos uso de MySQL ( https://www.mysql.com/ ), se ha seleccionado este sistema de gestión de base de datos porque es uno de los sistemas más extendidos y está muy documentado a la vez de que dispone de herramientas tales como por ejemplo PHPMyAdmin (https://www.phpmyadmin.net/), la cual nos facilita mucho la gestión de la base de datos, pero podríamos haber utilizado cualquier otro sistema de base de datos.

### 7.2.1. MySQL

Junto a MySQL también vamos a instalar la herramienta phpMyAdmin para que nos ayude a la hora de trabajar con la base de datos. Para instalar esta última tendremos que instalar también el servidor web Apache.

Comenzamos con la instalación de los paquetes necesarios en nuestro servidor (Raspberry Pi), hay que recordar que nos podemos conectar también de forma remota mediante ssh al servidor, ahora ejecutaremos los siguientes comandos:

Primero actualizaremos el sistema:

```
$ sudo apt-get update
$ sudo apt-get dist-upgrade
$ sudo apt-get install raspberrypi-ui-mods
```

Continuaremos instalando Apache y PHP:

```
$ sudo apt-get install apache2 php5 libapache2-mod-php5
$ sudo /etc/init.d/apache2 restart
```

En este punto ya tenemos funcionando Apache y PHP, se puede comprobar accediendo a la dirección desde el navegador:

```
http://192.168.1.254/
```

Ahora instalaremos MySQL y phpmyadmin ejecutando los comandos:

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

```
$ sudo apt-get install php5-mysql libapache2-mod-auth-mysql
$ sudo apt-get install phpmyadmin
```

Nos preguntará durante la instalación por el password para el usuario root de MySQL en nuestro caso introduciremos como password “passmysql531” por ejemplo

Por último nos falta editar el fichero /etcapache2/apache2.conf y al final del mismo insertar la siguiente línea:

```
Include /etc/phpmyadmin/apache.conf
```

Ahora tenemos que crear nuestra base de datos que nombraremos “pruebaiot”, dada la simplicidad del sistema, contendrá dos tablas:

- datosmedidas : Esta tabla contendrá un registro por cada sensor que esté en funcionamiento, es donde se encontrarán la últimas lectura de cada sensor.
- datosmedidaslog : Esta tabla será un histórico de la anterior, es decir, se almacenarán todas las lecturas realizadas para así por ejemplo elaborar un histórico de cualquier sensor. La única diferencia con respecto a ‘datosmedidas’ es que dispondrá de su propio campo ‘id’ y que el campo ‘iddatos’ contendrá la ‘id’ que correspondía en ‘datosmedidas’

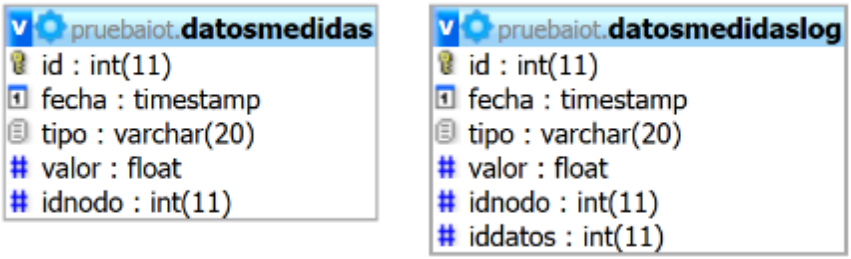


*Ilustración 27: Tablas de la base de datos*

Las consultas SQL utilizadas para crear las tablas son las siguientes:

```
CREATE TABLE IF NOT EXISTS `datosmedidas` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `fecha` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  `tipo` varchar(20) NOT NULL,
  `valor` float NOT NULL,
  `idnodo` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 ;

CREATE TABLE IF NOT EXISTS `datosmedidaslog` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `fecha` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  `tipo` varchar(20) NOT NULL,
  `valor` float NOT NULL,
  `idnodo` int(11) NOT NULL,
```

```
  `iddatos` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 ;
     .
```

### 7.2.2. Servicios REST

En este apartado terminaremos de definir los servicios REST a utilizar y su implementación final.

Dado que se utilizará una segunda tabla para guardar el histórico de las lecturas se añadirán 2 servicios REST más para hacer uso de los datos incluidos en esta tabla.

Por último se detectó que existen ciertos problemas a la hora de realizar PUTs utilizando el módulo esp8266 con el firmware original y recomiendan actualizar los datos mediante GETs y pasando los datos como argumento en la misma petición, por lo que se creará también ese servicio.

Dado lo anterior nuestros servicios REST quedarán finalmente:

| Método HTTP | URI | Acción |
|---|---|---|
| **GET** | http://192.168.1.254/lecturas | Leerá todas las lecturas |
| **GET** | http://192.168.1.254/lecturas /id_lectura | Leerá la lectura con id=id_lectura |
| **PUT** | http://192.168.1.254/lecturas /id_lectura | Modificará la lectura con id=id_lectura |
| **POST** | http://192.168.1.254/lecturas /id_lectura | Añadirá una lectura con id=id_lectura |
| **DELETE** | http://192.168.1.254/lecturas /id_lectura | Eliminará la lectura con id=id_lectura |
| **GET** | http://192.168.1.254/lecturasnodo/id_nodo | Leerá todas las lecturas del nodo id_nodo |
| **GET** | http://192.168.1.254/historicolecturas | Leerá todo el histórico de las lecturas |
| **GET** | http://192.168.1.254/historicolecturas /id_lectura | Leerá todo el histórico de la lectura con id=id_lectura |
| **GET** | http://192.168.1.254/cambialecturas /id_lectura | Modificará la lectura con id=id_lectura |

*Tabla 7: Servicios REST implementados finalmente*

Los datos leídos seguirán devolviéndose en formato json.

Los servicios REST que necesitamos los crearemos en un fichero llamado ‘appiot.py’ el cual lo pondremos dentro de la ruta ‘/home/pi/flask/appiot.py’

Se utilizará la librería ‘MySQLdb’ de Python para realizar las consultas a la base de datos, es necesario instalar los paquetes necesarios ejecutando en un terminal:

```
$ sudo apt-get install python-dev libmysqlclient-dev
$ sudo pip install MySQL-python
```

Para cada servicio que queremos implementar se definirá una función en Python, el código completo se puede ver en el fichero ‘appiot.py’, también se definirán 2 funciones para facilitar las consultas a la base de datos.

Las cabeceras de las funciones implementadas son las siguientes:

```
# ejecuta una query en la base de datos
def run_query(query=''):
# ejecuta una query en la base de datos y el resultado lo
# devuelve como json
def run_query_json(query=''):
# devuelve todas la lecturas de un nodo
@app.route('/lecturasnodo/<int:nodo_id>', methods=['GET'])
def get_lecturasnodo(nodo_id):

# devuelve todas las lecturas
@app.route('/lecturas', methods=['GET'])
def get_lecturas():
# devuelve la lectura indicada
@app.route('/lecturas/<int:lectura_id>', methods=['GET'])
def get_lectura(lectura_id):
# devuelve el historico de las lecturas
@app.route('/historicolecturas', methods=['GET'])
def get_historicolecturas():
# devuelve el historico de una lectura indicada
@app.route('/historicolecturas/<int:lectura_id>',
methods=['GET'])
def get_historicolectura(lectura_id):
#insertar un nueva lectura
@app.route('/lecturas', methods=['POST'])
def create_lectura():
#borrar una lectura de la BD
@app.route('/lecturas/<int:lectura_id>', methods=['DELETE'])
def delete_lectura(lectura_id):
#modifica una lectura de la BD
@app.route('/lecturas/<int:lectura_id>', methods=['PUT'])
def update_lectura(lectura_id):
#modifica una lectura de la BD utilizando GET
@app.route('/cambialectura', methods=['GET'])
def cambia_lectura():
```

Para comprobar el correcto funcionamiento de los servicios se puede hacer uso de ellos utilizando por ejemplo la herramienta curl (http://curl.haxx.se/ ) de la siguiente manera:

```
//obtener todas las lecturas
curl -X GET http://192.168.1.254:5000/lecturas

//obtener una lectura
curl -X GET http://192.168.1.254:5000/lecturas/1

//insertar una lectura
curl -i -H "Content-Type: application/json" -X POST -d '{"id":1,
"tipo":"temperatura", "valor":69.0, "idnodo":1}'
http://192.168.1.254:5000/lecturas

//borrar una lectura
curl -i -H "Content-Type: application/json" -X DELETE
http://192.168.1.254:5000/lecturas/1

//modificar una lectura
curl -i -H "Content-Type: application/json" -X PUT -d
'{"tipo":"temperatura", "valor":11, "idnodo":1}'
http://192.168.1.141:5000/lecturas/1
```

```
//obtener histórico de todas las lecturas
curl -X GET http://192.168.1.254:5000/historicolecturas

//obtener histórico de una lectura
curl -X GET http://192.168.1.254:5000/ historicolecturas /1


//obtener las lecturas de un nodo
curl -X GET http://192.168.1.254:5000/lecturasnodo/1

//para modificar una lectura mediante GET se puede hacer directamente
desde un navegador accediendo por ejemplo a la dirección:
http://192.168.1.254:5000/cambialectura?lectura=1&tipo=temperatura&val
or=10&idnodo=1
```

Ahora solo nos queda conseguir que se ejecute el programa al inicio del sistema de forma automática, esto lo conseguimos de forma sencilla editando el fichero /etc/rc.local y añadiendo la siguiente línea al final del mismo, justo antes de 'exit(0)':

```
python2.7 /home/pi/flask/appiot.py
```

Con estos pasos ya disponemos de nuestro propio servicio web REST.

## 7.3. Nodos

Como parte final del proyecto se implementará uno de los nodos que dispondrá de una serie de una serie de sensores y actuadores a modo de ejemplo y la programación de la placa Arduino y para poder controlar el módulo esp8266 y poder conectarse a la red, también se implementará un pequeño programa cliente para poder controlar el circuito de forma directa.

### 7.3.1. Circuito

El circuito que se va implementar es la unión de los circuitos vistos en el capítulo dedicado a los sensores. Se utilizará el sensor DHT11 conectado al pin 8, un servo conectado al pin 10, un led conectado al pin 13 (en verdad se utilizará el que ya trae de serie la placa arduino conectado a dicho pin) y un pin bicolor (rojo y verde), conectado a los pines 11 y 12 respectivamente, este led bicolor nos indicará el estado de la conexión a la red, cuando estemos conectados tendrá color verde y cuando no lo estemos, su color será rojo.

El pin 4 se conectará con una resistencia de 220 Ω al pin reset del módulo esp8266 para poder reiniciar el módulo por hardware si fuera necesario.

Para que el nodo funcione de forma correcta, necesitamos dos fuentes de tensión regulada, una de 5v para alimentar al Arduino y al resto de componentes y otra de 3,3v para alimentar al módulo esp8266, esto lo podemos conseguir utilizando reguladores como pueden ser la familia de reguladores AMS1117 (http://www.advanced-monolithic.com/pdf/ds1117.pdf ) que nos ofrece variedad de reguladores para distintos voltajes.

Siguiendo con la línea de implementar un circuito lo más simple y sencillo posible se encontró un módulo para poder alimentar los circuitos directamente en la protoboard y a un bajísimo precio, por lo que se decidió utilizar dicho módulo para tal fin. El módulo en cuestión es el "YwRobot Breadboard Power Supply" (www.petervis.com/Raspberry_PI/Breadboard_Power_Supply/YwRobot_Breadboard_Power_Supply.html), este módulo utiliza dos reguladores de la familia AMS1117 para ofrecernos los 3,3v y los 5v que necesitamos a la vez que nos proporciona conector de alimentación, interruptor y led de estado.


*Ilustración 28: Módulo YwRobot Breadboard Power Supply*

También se ha incluido un jumper para aislar al Arduino de los 5v del regulador por si fuera necesario cuando tenemos al Arduino conectado por USB.

Ahora utilizando una placa de baquelita perforada, soldador, estaño y un poco de paciencia podemos implementar nuestro circuito:

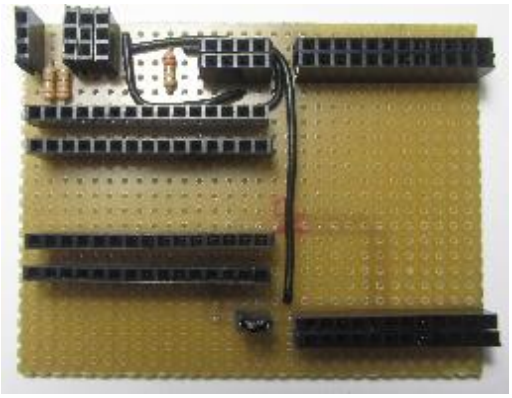

*Ilustración 29: Vista superior del circuito sin elementos*

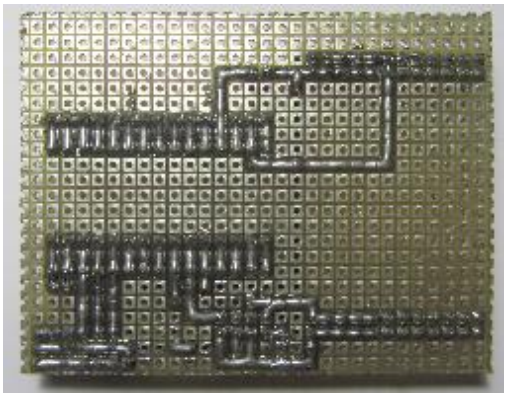

*Ilustración 30: Vista inferior del circuito sin elementos*

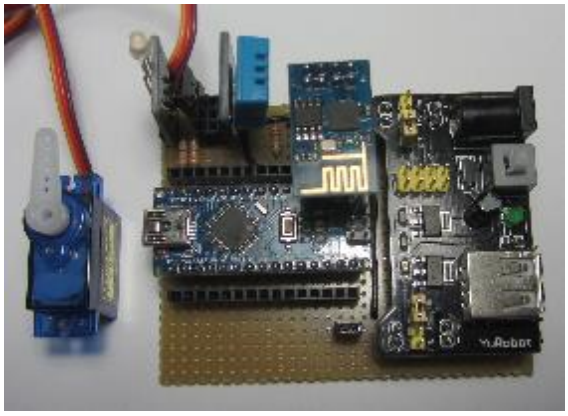

*Ilustración 31: Vista superior del circuito con elementos*

### 7.3.2. Programación

El programa correspondiente al Arduino consistirá en conseguir que el módulo esp8266 se conecte correctamente a la red y esperar a que se le soliciten los valores de los sensores o se quiera modificar el estado de los actuadores.

Cuando se reciba una solicitud, se actuará de la forma deseada, por ejemplo si se le solicita la temperatura, devolverá el valor que tenga el sensor de temperatura y también actualizará el valor de la misma en la base de datos, para ello utilizará el servicio REST que se ha creado anteriormente para tal fin,

| Método HTTP | URI | Acción |
|---|---|---|
| **GET** | http://192.168.1.254/cambialecturas /id_lectura | Modificará la lectura con id=id_lectura |

El programa resultado es el fichero '**finalEsp8266.ino**' en el cual se pueden observar las 5 funciones que lo componen:

- **void setup()** : En esta función se inicializan todas las variables y pines necesarios del Arduino, se definen las comunicaciones serie utilizadas y se conecta a la red llamando a la función 'connectWiFi()'

- **void loop()**: En esta función se está comprobando si hay datos en el puerto serie y en caso afirmativo, se evalúan por si contienen la palabra clave que identifica a las opciones que el nodo posee, en este caso, las palabras clave serán:
    - **temperatura**: nos están solicitando la temperatura del sensor.
    - **humedad**: nos están solicitando la humedad del sensor.
    - **ledON**: indica que encendamos el led
    - **ledOFF**: indica que apaguemos el led
    - **servoPos**: indica que movamos el servo

  Una vez realizada la operación solicitada formará la cadena de respuesta correspondiente y llamará a las funciones 'enviarRespuesta(String respuesta)' para responder a quien hizo la petición con dicha respuesta y a 'enviarRespuestaREST(int lectura, String tipo, float valor, int idnodo)' para actualizar los valores en la base de datos, en este caso 'lectura' se refiere al id del sensor, 'tipo' la descripción del sensor, 'valor' el valor actual del sensor, 'idnodo' el id del nodo, en este caso 'idnodo' valdrá 1.

- **boolean connectWiFi()**: Esta función es la encargada de realizar las operaciones necesarias para que el módulo esp8266 se conecte de forma correcta a la red, de no ser así devolverá 'false'.

- **void enviarRespuesta(String respuesta)** : Esta es la función mediante la cual se envían los datos de respuesta al programa que nos ha realizado la petición

- **void enviarRespuestaREST(int lectura, String tipo, float valor, int idnodo)**: En esta última función se establecerá una conexión con nuestro servidor y se realizará la petición GET deseada actualizando los valores de la base de datos.

Se ha realizado de esta forma para tener un mayor control de lo que se escribe en la base de datos, pero se podría hacer fácilmente que escribiera esos valores de forma automática cada cierto tiempo, solo tendríamos que hacer un temporizador que llame a la función 'enviarRespuestaREST( )' con el intervalo de tiempo deseado.

Por último solo nos falta implementar el programa que realizará las peticiones a nuestro nodo, será un sencillo programa en Python que nos mostrará un pequeño menú con las opciones disponibles en nuestro nodo.

Cuando seleccionemos una opción abrirá una conexión con el nodo y le enviará la palabra clave correspondiente a la opción del menú elegida. En caso de querer mover el servo nos pedirá los grados en que hay que moverlo y los enviará junto con la palabra clave, el nodo ya procesará esta cadena y actuará en consecuencia.

Por último esperará a recibir una respuesta del nodo, cuando la reciba, cerrará la conexión y mostrará la respuesta por pantalla, acto seguido mostrará nuevamente el menú.

Este programa se encuentra en el fichero '**menu.py**' y para ejecutar dicho programa solo hay que escribir en un terminal:

```
$ python menu.py
```

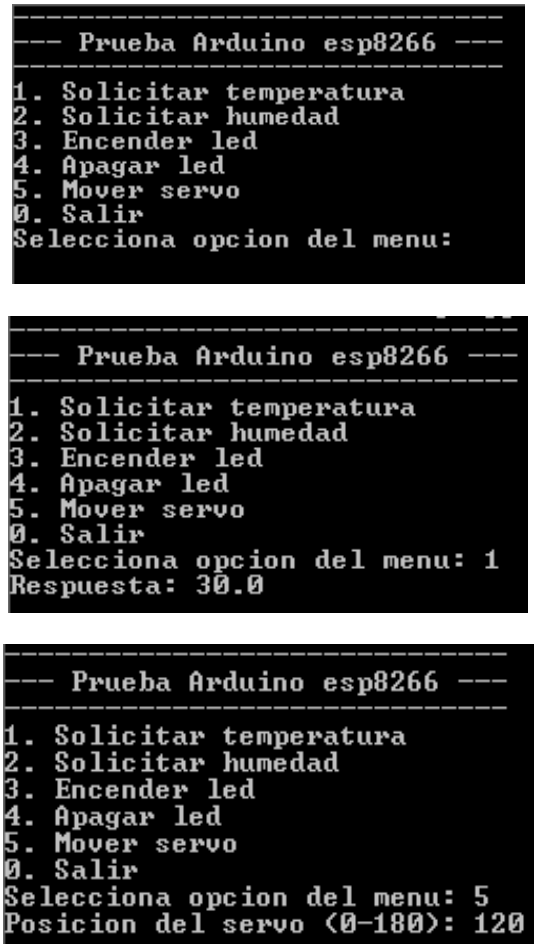


*Ilustración 32: Capturas programa menú*

# 8. Conclusiones

Como se puede observar a lo largo del proyecto se ha logrado un sistema sencillo, pero funcional, modular y fácilmente ampliable dado que la mayoría de los sensores y actuadores tiene un funcionamiento muy similar, por poner un ejemplo, sería trivial el sustituir el led del nodo desarrollado por un relé.

Las posibilidades que nos brindan los servicios web REST son muy grandes, y gracias a la sencillez de las herramientas utilizadas se pueden adaptar estos servicios sin una elevada complicación a distintos proyectos.

Desarrollar con Arduino ha sido muy gratificante al ver como poco a poco se conseguía mejoras con los circuitos, los sensores y las posibilidades que nos brindan, tanto es así que adquirí un kit de sensores y actuadores ( kit 37 in 1 ) para continuar probando con otros modelos.

La utilización de Raspberry Pi también ha sido muy agradable y sencilla ya que prácticamente se comporta como un PC con Linux, por lo que si se tiene cierta experiencia, no debería haber ningún problema en su uso.

Por el contrario he tenido bastantes problemas utilizando el módulo esp8266 dado la escasez de documentación oficial y ciertos problemas que según parece deberían solucionarse al utilizar otro firmware. Aunque gracias a la comunidad de usuarios de este módulo se consiguió solucionar la mayoría.

Como futuras ampliaciones se proponen:

- Realizar una interfaz de usuario con más opciones y más usable utilizando por ejemplo, páginas webs al estilo de ThingSpeak.
- Mejorar el tratamiento de los errores por medio de códigos que muestren el error producido y mejorando la estabilidad en el tratamiento del mismo.
- Mejorar la forma de asignar las direcciones IP y los nombres a los nodos
- Probar otras versiones del firmware del módulo esp8266.
- Con una batería recargable, el nodo podría ser completamente inalámbrico.
- Mejorar la implementación del circuito para dar un aspecto más profesional e incluso se podría realizar una carcasa para contener al nodo con todos sus componentes.

# 9. Bibliografía

Bibliografía


- Arduino: aplicaciones en robótica, mecatrónica e ingenierías - Fernando Reyes Cortes; Jaime Cid Monjaraz – Marcombo – ISBN: 9788426722041
- ARDUINO Curso práctico de formación  Óscar Torrente Artero – rclibros – ISBN: 978-84-940725-0-5
- Introducción a Arduino - Massimo Banzi - Anaya - ISBN: 9788441531772
- Raspberry Pi 200 ejercicios prácticos – Simon Monk - ISBN: 9788441536289
- Flask Web Development - Miguel Grinberg - Orreilly - ISBN: 978-1-4493-7262-0
- Python 3 al descubierto - Arturo Fernandez Montoro - ARTURO FERNANDEZ MONTORO - RC LIBROS - ISBN: 9788493945046
- Introduccion a la programación con Python y C – Andrés Marzal, Isabel Gracia – Publicaciones Universitat Jaume I - ISBN: 978-84-692-5869-9


Internet


- Página oficial Arduino
  Fecha consulta Marzo 2015
  https://www.arduino.cc/

- Página oficial Flask
  Fecha consulta Marzo 2015
  http://flask.pocoo.org/

- Página oficial Fritzing
  Fecha consulta Marzo 2015
  http://fritzing.org

- Página oficial ThingSpeak
  Fecha consulta Marzo 2015
  https://thingspeak.com/

- Página oficial Raspberry PI
  Fecha consulta Marzo 2015
  https://www.raspberrypi.org

- Tutoriales Arduino
  Fecha consulta Abril 2015
  http://www.prometec.net/

- Internet de las cosas
  Fecha consulta Junio 2015
  http://www.internetdelascosas.cl

- ESP8266 Community Forum
  Fecha Consulta Julio 2015
  http://www.esp8266.com/

- Manual programación Arduino
  Fecha consulta Junio 2015
  http://arduinobot.pbworks.com/f/Manual+Programacion+Arduino.pdf

- Flask Web Development
  Fecha consulta Junio 2015
  http://blog.miguelgrinberg.com/

- Introducción Arduino
  Fecha Consulta Julio 2015
  http://recursos.cepindalo.es/course/view.php?id=35

- Análisis comparativo de las placas Arduino
  Fecha consulta Mayo 2015
  http://comohacer.eu/analisis-comparativo-placas-arduino-oficiales-compatibles/

- Alselectro esp8266
  Fecha Consulta Agosto 2015
  https://alselectro.wordpress.com/

- Arduino, esp8266
  Fecha Consulta Agosto 2015
  http://electronut.in

- ChildOfCode
  Fecha Consulta Agosto 2015
  http://childofcode.com/

- Arduino, ESP8266
  Fecha Consulta Julio 2015
  http://www.jopapa.me/

- Electrónica, Arduino, ESP8266
  Fecha Consulta Agosto 2015
  http://polaridad.es/

- Sensor movimiento PIR
  Fecha Consulta Agosto 2015
  http://www.omniblug.com/index.php/blog

- Arduino, Raspberry Pi
  Fecha Consulta Agosto 2015
  http://www.diverteka.com/

- Raspberry Pi configuración
  Fecha Consulta Mayo 2015
  http://bricolabs.cc/wiki

- Raspberry Pi configuración
  Fecha Consulta Junio 2015
  http://raspberryparatorpes.net

- Resumen Arduino
  Fecha Consulta Agosto 2015
  https://github.com/antonio-m/Acordeon-arduino/blob/master/Acordeon%20Arduino.pdf?raw=true

- Raspberry Pi, Arduino
  Fecha Consulta Agosto 2015
  https://geekytheory.com/

- Documentación Python
  Fecha Consulta Mayo 2015
  https://docs.python.org

# ANEXO: Código de las aplicaciones

Se incluye el código implementado:

## basicoEsp8266.ino

```
#include <SoftwareSerial.h>
SoftwareSerial SerialEsp8266(2, 3); // TX/RX

void setup()
  {  Serial.begin(9600);
     SerialEsp8266.begin(9600);
  }

void loop()
  {  String B= "." ;
     if (SerialEsp8266.available())
         { char c = SerialEsp8266.read() ;
           Serial.print(c);
         }
     if (Serial.available())
         {  char c = Serial.read();
            SerialEsp8266.print(c);
         }
   }
```

## led.ino

```
 void setup() {
  // inicializa el pin 3 como salida
  pinMode(3, OUTPUT);
}

void loop() {
  digitalWrite(3, HIGH);   // pone el pin 3 en HIGH (5v) para
encender el led
  delay(1000);             // esperamos 1s
  digitalWrite(3, LOW);    // pone el pin 3 en LOW (0v) para
apagar el led
  delay(1000);             // esperamos 1s
}
```

## buzzer.ino

```
void setup() {
  // inicializa el pin 3 como salida
  pinMode(3, OUTPUT);
}

void loop() {
  analogWrite(3, 25);   // pone el pin 3 con valor analogico 25
  delay(500);           // esperamos 0,5s
```

```
  analogWrite(3, 100);  // pone el pin 3 con valor analogico 100
  delay(500);           // esperamos 0,5s
}
```

## rele.ino

```
void setup() {
  // inicializa el pin 3 como salida
  pinMode(3, OUTPUT);
}

void loop() {
  digitalWrite(3, HIGH);   // pone el pin 3 en HIGH (5v) para
activar el rele
  delay(1000);             // esperamos 1s
  digitalWrite(3, LOW);    // pone el pin 3 en LOW (0v) para
apagar el rele
  delay(1000);             // esperamos 1s
}
```

## servo.ino

```
#include <Servo.h>

int val = 0; //Variable de entrada del Serial
Servo servo; //Creamos un objeto Servo

void setup() {
  Serial.begin(9600); //Iniciamos el serial
  servo.attach(3);    //Conectamos el servo al pin PWM 3
}

void loop() {
  if (Serial.available() > 0) { //Detecta si hay alguna entrada
por serial
    val = Serial.parseInt();
    if (val != 0) {
      servo.write(val); //Mueve el servo a la posicion entrada
(excepto si es 0)
    }
  }
  delay(500);
}
```

## temperatura_humedad.ino

```
#include "DHT.h"

#define DHTPIN 3     // what pin we're connected to
#define DHTTYPE DHT11   // DHT 11

DHT dht(DHTPIN, DHTTYPE);

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

```
  dht.begin();
}

void loop() {
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  // Se comprueba que la lectura es correcta
  // isnan(var) devuelve 1 si var no es un número
  if (isnan(t) || isnan(h)) { // Se comprueba que la lectura es
correcta
    Serial.println("Error al leer DHT11");
  } else {
    Serial.print("Humedad: ");
    Serial.print(h);
    Serial.print(" %\t");
    Serial.print("Temperatura: ");
    Serial.print(t);
    Serial.println(" C");
  }
}
```

## finalEsp8266.ino

```
#include <SoftwareSerial.h>
#include <PWMServo.h>
SoftwareSerial SerialEsp8266(2, 3); // RX, TX

#include "DHT.h"        // libreria utilizada
#define DHTPIN 8        // pin del sensor DHT11
#define DHTTYPE DHT11   // DHT11 modelo del sensor
DHT dht(DHTPIN, DHTTYPE);

#define SSID "MIRED" //El ssid de tu router
#define PASS "passmired" //La clave Wifi

#define MAX_STRING_LEN 20

PWMServo servo; //Creamos un objeto Servo

// los pines para cada led
int ledWifiRojo=11;
int ledWifiVerde=12;
int led = 13;

int idnodo=1;

void setup() {
      //Conectamos el servo al pin 10
      servo.attach(10);

      //pin del led
      pinMode(led, OUTPUT);
      //pin del led rojo wifi
      pinMode(ledWifiRojo, OUTPUT);
      digitalWrite(11, HIGH);
      //pin del led verde wifi
      pinMode(ledWifiVerde, OUTPUT);
    digitalWrite(12, LOW);
```

```
//pin reset HW del esp8266
pinMode(4, OUTPUT);
digitalWrite(4, HIGH);

// serie para conectar con el modulo esp8266
SerialEsp8266.begin(19200);
// serie para conectar con el modulo esp8266
Serial.begin(19200);

// reiniciamos el modulo esp8266
SerialEsp8266.println("AT+RST");
delay(1000);

boolean listo=false;

while(!listo){
      if(SerialEsp8266.find("ready")){
            listo=true;
            Serial.println("Modulo preparado");

            // conectandonos al wifi
            // hasta que no se conecte no hace otra cosa
            boolean conectado=false;
            while(!conectado){
                  conectado=connectWiFi();
            }

            //Aceptar multiples conexiones
            SerialEsp8266.println("AT+CIPMUX=1");
            delay(1000);

            // mostramos la IP asignada
            Serial.print("IP:");
            SerialEsp8266.println("AT+CIFSR");
            delay(100);
            while ( SerialEsp8266.available() ) {
                  Serial.write(SerialEsp8266.read());
            }

            // Actua como Server en puerto 80
            Serial.println("Comenzando Server TCP");
            SerialEsp8266.println("AT+CIPSERVER=1,80");

            //encendemos el led rojo
            digitalWrite(ledWifiRojo, LOW);
            digitalWrite(ledWifiVerde, HIGH);

      }else{
            // si no responde lo reseteamos por hardware
            Serial.println("Modulo no responde");
            digitalWrite(4, LOW);
            delay(500);
            digitalWrite(4, HIGH);
            delay(1000);
            SerialEsp8266.println("AT+RST");
            delay(1000);
      }

}

Serial.println("Esperando conexion...");
```

```
	// el sensor DHT11
	dht.begin();
}


void loop() {

	// Si recive datos por el puerto del esp8266
	if (SerialEsp8266.available()>0){
		String S;
		String respuesta;
		String cadena;
		// lee los datos hasta que encuentre '\n'
		S=SerialEsp8266.readStringUntil('\n');

		// Evalua los datos recividos y actua en consecuencia

		if(S.indexOf("temperatura")>0) {
			float t = dht.readTemperature();
			// convierto los datos a cadena
			char buf[16];
			String strTemp = dtostrf(t, 4, 1, buf);
			respuesta="Temperatura = ";
			respuesta += strTemp;
			enviarRespuesta(respuesta);
			enviarRespuestaREST(1,"temperatura",t,idnodo);
		}

		if(S.indexOf("humedad")>0) {
			float h = dht.readHumidity();
			// convierto los datos a cadena
			char buf[16];
			String strHumi = dtostrf(h, 4, 1, buf);
			respuesta="Humedad = ";
			respuesta += strHumi;
			enviarRespuesta(respuesta);
			enviarRespuestaREST(2,"humedad",h,idnodo);
		}

		if(S.indexOf("ledON")>0) {
			digitalWrite(led, HIGH);
			respuesta="Led encendido";
			enviarRespuesta(respuesta);
			enviarRespuestaREST(3,"led",1,idnodo);
		}

		if(S.indexOf("ledOFF")>0) {
			digitalWrite(led, LOW);
			respuesta="Led apagado";
			enviarRespuesta(respuesta);
			enviarRespuestaREST(3,"led",0,idnodo);
		}

		if(S.indexOf("servoPos")>0) {
			int ini,fin;

			ini=S.indexOf("<servo>")+7;
			fin=S.indexOf("</servo>");
			String S2=S.substring(ini,fin);
			Serial.println(S2);
```

```
                  int s=S2.toInt();
                  int sAux=s;
                  // el servo que utilizo solo gira hasta 165
grados
                  // pero en la respuesta indico los grados que
me solicitaron
                  if(s>165){
                        s=165;
                  }
                  servo.write(s);
                  respuesta="servo movido";
                  enviarRespuesta(respuesta);
                  enviarRespuestaREST(4,"servo",sAux,idnodo);
            }
      }
}

boolean connectWiFi(){
      // Modo de funcionamiento 1
      SerialEsp8266.println("AT+CWMODE=1");

      //se prepara la cadena para realizar la conexion
      String cmd="AT+CWJAP=\"";
      cmd+=SSID;
      cmd+="\",\"";
      cmd+=PASS;
      cmd+="\"";
      Serial.println("Intentando conectar a:");

      // se intenta conectar
      Serial.println(cmd);
      SerialEsp8266.println(cmd);
      delay(100);

      // Comprueba si ha tenido respuesta OK a la peticion
      if(SerialEsp8266.find("OK")){
            Serial.println("Esp8266 conectado a WIFI");
      }else{
            Serial.println("Error al conectar Esp8266 a WIFI");
      }

      //  Comprueba que tenemos una IP de la red
      SerialEsp8266.println("AT+CIFSR");
      if(SerialEsp8266.find("192.")){
            Serial.println("Si tiene IP");
            return true;
      }else{
            Serial.println("No tiene IP");
            return false;
      }
}

// envia la respuesta
void enviarRespuesta(String respuesta){
      Serial.println("Enviado a esp8266:");
      Serial.println(respuesta);
      // a la conexion le dare el canal 0
      int connectionId =0;
      // se forma la cadena de respuesta
      String cipSend = "AT+CIPSEND=";
      cipSend += connectionId;
```

```
      cipSend += ",";

      // tamaño de los datos a enviar
      cipSend +=respuesta.length();
      cipSend +="\r\n";
      cipSend +=respuesta;

      // se envian los datos
      Serial.println("cipSend:");
      Serial.println(cipSend);
      SerialEsp8266.print(cipSend);
      delay(1000);

      // se cierra la conexion
      respuesta="AT+CIPCLOSE=";
      respuesta+=connectionId;
      respuesta+="\r\n";
      Serial.println("Respuesta:");
      Serial.println(respuesta);
      SerialEsp8266.print(respuesta);
      delay(1000);
}


void enviarRespuestaREST(int lectura, String tipo, float valor,
int idnodo){

      // conexion TCP a nuestro server
      String cmd = "AT+CIPSTART=4,\"TCP\",\"";
      cmd += "192.168.1.254"; // ip del servidor
      cmd += "\",5000";
      SerialEsp8266.println(cmd);
      Serial.println(cmd);

      // Si no se ha realizado correctamente la conexion
      if(!SerialEsp8266.find("OK")){
            Serial.println("AT+CIPSTART error");
            return;
      }

      //convierto el float en String y quito los espacios en
blanco
      char buffer[20];
      String sValor = dtostrf(valor, 10, 2, buffer);
      sValor.trim();

      // se prepara el GET
      String getStr = "GET /cambialectura?lectura=";
      getStr += String(lectura);
      getStr +="&tipo=";
      getStr += tipo;
      getStr +="&valor=";
      getStr += sValor;
      getStr +="&idnodo=";
      getStr += String(idnodo);
      getStr += "\r\n";

      // tamaño de los datos a enviar
      cmd = "AT+CIPSEND=4,";
      cmd += String(getStr.length());
```

```
      // se realiza la peticion
      Serial.println(cmd);
      delay(1000);
      SerialEsp8266.println(cmd);
      delay(2000);

      // si no se ha realizado la peticion bien termina
      if(SerialEsp8266.find(">")){
            Serial.println("Encontrado >");
      }else{
            Serial.println("No Encontrado >");
            SerialEsp8266.println("AT+CIPCLOSE=4");
            return;
      }

      Serial.println(getStr);
      SerialEsp8266.println(getStr);
      delay(1000);

      // se cierra la conexion
      Serial.println("AT+CIPCLOSE=4");
      delay(1000);
      SerialEsp8266.println("AT+CIPCLOSE=4");
      delay(1000);

}
```

## appiot.py

```
#!flask/bin/python
from flask import Flask, jsonify, abort, make_response, request,
url_for

import MySQLdb

app = Flask(__name__, static_url_path = "")

# datos de la la Base de Datos a utilizar
DB_HOST = '192.168.1.254'
DB_USER = 'pi'
DB_PASS = '12345678'
DB_NAME = 'pruebaiot'

# ejecuta una query en la base de datos
def run_query(query=''):
    datos = [DB_HOST, DB_USER, DB_PASS, DB_NAME]
    conn = MySQLdb.connect(*datos) # Conectar a la base de datos
    cursor = conn.cursor()         # Crear un cursor
    cursor.execute(query)          # Ejecutar una consulta
    if query.upper().startswith('SELECT'):
        data = cursor.fetchall()   # Trae los resultados de un
select
    else:
        conn.commit()              # Hace efectiva la escritura
de datos
        data = None
    cursor.close()                 # Cerrar el cursor
    conn.close()                   # Cerrar la conexion
    return data
```

```
# ejecuta una query en la base de datos y el resultado lo
devuelve como json
def run_query_json(query=''):
    datos = [DB_HOST, DB_USER, DB_PASS, DB_NAME]
    conn = MySQLdb.connect(*datos) # Conectar a la base de datos
    cursor = conn.cursor()         # Crear un cursor
    cursor.execute(query)          # Ejecutar una consulta
    # al resultado se le da formato json
    rows=cursor.fetchall()
    columns = [desc[0] for desc in cursor.description]
    result = []
    for row in rows:
        row = dict(zip(columns, row))
        result.append(row)
    cursor.close()                 # Cerrar el cursor
    conn.close()                   # Cerrar la conexion
    return result

# devuelve todas la lecturas de un nodo
@app.route('/lecturasnodo/<int:nodo_id>', methods=['GET'])
def get_lecturasnodo(nodo_id):
    #realizar consulta
    query = "SELECT * FROM datosmedidas WHERE idnodo = '%d'" %
nodo_id
    result = run_query_json(query)
    if len(result) == 0:
        abort(404)
    nodo='nodo'+str(nodo_id)
    return jsonify({nodo: result})

# devuelve todas las lecturas
@app.route('/lecturas', methods=['GET'])
def get_lecturas():
    #realizar consulta
    query = "SELECT * FROM datosmedidas"
    result = run_query_json(query)
    return jsonify({'lecturas': result})

# devuelve la lectura indicada
@app.route('/lecturas/<int:lectura_id>', methods=['GET'])
def get_lectura(lectura_id):
    #realizar consulta
    query = "SELECT * FROM datosmedidas WHERE id = '%d'" %
lectura_id
    result = run_query_json(query)
    if len(result) == 0:
        abort(404)
    lectura='lectura'+str(lectura_id)
    return jsonify({lectura: result})

# devuelve el historico de las lecturas
@app.route('/historicolecturas', methods=['GET'])
def get_historicolecturas():
    #realizar consulta
    query = "SELECT * FROM datosmedidaslog"
    result = run_query_json(query)
    return jsonify({'lecturas': result})

# devuelve el historico de una lectura indicada
@app.route('/historicolecturas/<int:lectura_id>',
methods=['GET'])
```

```
def get_historicolectura(lectura_id):
    #realizar consulta
    query = "SELECT * FROM datosmedidaslog WHERE iddatos = '%d'"
% lectura_id
    result = run_query_json(query)
    if len(result) == 0:
        abort(404)
    lectura='lectura'+str(lectura_id)
    return jsonify({lectura: result})

#insertar un nueva lectura
@app.route('/lecturas', methods=['POST'])
def create_lectura():
      #compruebo que exista un json en la peticion
    if not request.json:
        abort(400)
    #los datos a insertar
    id1 = request.json.get('id')
    tipo = request.json.get('tipo')
    valor = request.json.get('valor')
    idnodo = request.json.get('idnodo')
    query = "INSERT INTO datosmedidas VALUES
('%d',CURRENT_TIMESTAMP,'%s','%f','%d')" % (id1,tipo, valor,
idnodo)
    run_query(query)
    query = "INSERT INTO datosmedidaslog VALUES
(NULL,CURRENT_TIMESTAMP,'%s','%f','%d','%d')" % (tipo, valor,
idnodo, id1)
    run_query(query)
    return jsonify({'result': True})

#borrar una lectura de la BD
@app.route('/lecturas/<int:lectura_id>', methods=['DELETE'])
def delete_lectura(lectura_id):
    #compruebo que exista esa id
    query = "SELECT * FROM datosmedidas WHERE id = '%d'" %
lectura_id
    result = run_query_json(query)
    if len(result) == 0:
        abort(404)
    query = "DELETE FROM datosmedidas WHERE id = '%s'" %
lectura_id
    run_query(query)
    return jsonify({'result': True})

@app.route('/lecturas/<int:lectura_id>', methods=['PUT'])
def update_lectura(lectura_id):
    #compruebo que exista esa id
    query = "SELECT * FROM datosmedidas WHERE id = '%d'" %
lectura_id
    result = run_query_json(query)
    if len(result) == 0:
        abort(404)
    #compruebo que exista un json en la peticion
    if not request.json:
        abort(400)
    #los datos a insertar
    id1 = lectura_id
    tipo = request.json.get('tipo')
    valor = request.json.get('valor')
    idnodo = request.json.get('idnodo')
```

```
    query = "UPDATE datosmedidas SET tipo='%s', valor='%f',
idnodo='%d' WHERE id='%d'" % (tipo, valor, idnodo, id1)
    run_query(query)
    query = "INSERT INTO datosmedidaslog VALUES
(NULL,CURRENT_TIMESTAMP,'%s','%f','%d','%d')" % (tipo, valor,
idnodo, id1)
    run_query(query)
    return jsonify({'result': True})

@app.route('/cambialectura', methods=['GET'])
def cambia_lectura():
    id1 = request.args.get('lectura')
    #compruebo que exista esa id
    query = "SELECT * FROM datosmedidas WHERE id = '%d'" %
int(id1)
    result = run_query_json(query)
    if len(result) == 0:
        abort(404)
    #los datos a insertar
    tipo = request.args.get('tipo')
    valor = request.args.get('valor')
    idnodo = request.args.get('idnodo')
    query = "UPDATE datosmedidas SET tipo='%s', valor='%f',
idnodo='%d' WHERE id='%d'" % (tipo, float(valor), int(idnodo),
int(id1))
    run_query(query)
    query = "INSERT INTO datosmedidaslog VALUES
(NULL,CURRENT_TIMESTAMP,'%s','%f','%d','%d')" % (tipo,
float(valor), int(idnodo), int(id1))
    run_query(query)
    return jsonify({'result': True})

if __name__ == '__main__':
    app.run(host='0.0.0.0',debug=True)
```

## menu.py

```
import socket
import sys

HOST, PORT = "192.168.1.130", 80

menu_item = 0
namelist = []
while 1:
    # Creamos un socket TCP
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print("------------------------------")
    print("--- Prueba Arduino esp8266 ---")
    print("------------------------------")
    print("1. Solicitar temperatura")
    print("2. Solicitar humedad")
    print("3. Encender led")
    print("4. Apagar led")
    print("5. Mover servo")
    print("0. Salir")
    menu_item = int(input("Selecciona opcion del menu: "))
    if menu_item == 1:
        cadena = "temperatura"
    elif menu_item == 2:
```

```
    cadena = "humedad"
elif menu_item == 3:
    cadena = "ledON"
elif menu_item == 4:
    cadena = "ledOFF"
elif menu_item == 5:
    grados = input("Posicion del servo (0-180): ")
    cadena = "servoPos<servo>"+grados+"</servo>"
elif menu_item == 0:
    print("Adios")
    sys.exit(1)
try:
    # Conectamos con el servidor y enviamos los datos
    sock.connect((HOST, PORT))
    sock.sendall(bytes(cadena + "\n", "utf-8"))

    # Recivimos los datos del servido y cerramos el socket
    recibido = str(sock.recv(1024), "utf-8")
finally:
    sock.close()
    print("Respuesta: {}".format(recibido))
```

# ANEXO: Estructura lenguaje Arduino

## Estructuras de control

- if *(comparador si-entonces)*
- if...else *(comparador si...sino)*
- for *(bucle con contador)*
- switch case *(comparador múltiple)*
- while *(bucle por comparación booleana)*
- do... while *(bucle por comparación booleana)*
- break *(salida de bloque de código)*
- continue *(continuación en bloque de código)*
- return *(devuelve valor a programa)*

## Sintaxis

- ; (punto y coma)
- {} (llaves)
- // (comentarios en una línea)
- /* */ (comentarios en múltiples líneas)

## Operadores Aritméticos

- = (asignación)
- + (suma)
- - (resta)
- * (multiplicación)
- / (división)
- % (modulo)

## Operadores Comparativos

- == (igual a)
- != (distinto de)
- < (menor que)
- > (mayor que)

- <= (menor o igual que)
- >= (mayor o igual que)

## Operadores Booleanos

- && (y)
- || (o)
- ! (negación)

## Operadores compuestos

- ++ (incremento)
- -- (decremento)
- += (suma compuesta)
- -= (resta compuesta)
- *= (multiplicación compuesta)
- /= (división compuesta)

## Constantes

- HIGH | LOW
- INPUT | OUTPUT
- true | false
- Constantes Numéricas

## Tipos de Datos

- boolean *(1,0,true,false)*
- char *(carácter ej. 'a' o de -128 a 127)*
- byte *(de 0 a 255 )*
- int *(entero de -32768 a 32767 )*
- unsigned int *(entero sin signo de 0 a 65535)*
- long *(entero 32b, de -2147483648 a 2147483647)*
- unsigned long *(entero 32b sin signo de 0 a 4294967295)*
- float *(en coma flotante de -3.4028235E +38 a 3.4028235E +38)*
- double *(en coma flotante de 32b de -3.4028235E +38 a 3.4028235E +38)*
- string *(cadena de caracteres)*

- array *(cadena)*
- void *(vacío)*

### E/S Digitales

- pinMode(pin,[INPUT, OUTPUT])
- digitalWrite(pin, valor)
- digitalRead(pin)

### E/S Analógicas

- analogRead(pin)
- analogWrite(pin, valor) - *PWM (modulación por ancho de pulso)*

### E/S Avanzadas

- tone(pin, freqhz)
- tone(pin, freqhz, duración_ms)
- noTone(pin)
- shiftOut(pinDatos, pinReloj,[MSBFIRST,LSBFIRST], valor)
- pulseIn(pin, [HIGH, LOW])

### Tiempo

- millis() - Se desborda el contador en 50 dias
- micros() – Se desborda el contador en 70 minutos
- delay(ms)
- delayMicroseconds(us)

### Matemáticas

- min(x,y) *(mínimo)*
- max(x,y) *(máximo)*
- abs(x) *(valor absoluto)*
- constrain(x, valMin, valMax) *(limita)*
- map(val, deBAJO, deALTO, aBAJO, aALTO ) *(cambia valor de rango)*
- pow(base, exponente) *(eleva a un número)*
- sq(x) *(eleva al cuadrado)*
- sqrt(x) *(raíz cuadrada)*

**Trigonometría**

- sin(x) *(seno)*
- cos(x) *(coseno)*
- tan(x) *(tangente)*

**Números Aleatorios**

- randomSeed(semilla) – long o int
- random(max)
- random(min, max)

## Comunicación Serial

- begin(velocidad) – abre puerto serie y establece velocidad
- end() – desactiva puerto serie
- available() – rebela si hay datos en el Puerto
- read() – lee un único carácter del buffer
- flush() – vacía el buffer
- print(datos) - imprime los datos en el puerto serie
- println(datos) - imprime los datos en el puerto serie con retorno de carro
- write(bytes) – escribe caracteres en el puerto serie