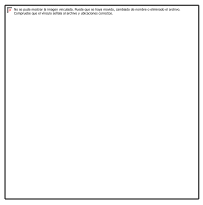

FACULTAD DE INGENIERÍA
CARRERA DE INGENIERÍA DE SISTEMAS

ASIGNATURA:

INTERNET DE LAS COSAS IoT

GUÍA DE LABORATORIO: TEORÍA

PRÁCTICA DE LABORATORIO N.º 3

CONECTIVIDAD A LA NUBE - WIFI

2023 - 2

PRÁCTICA DE LABORATORIO N.º 3
CONECTIVIDAD A LA NUBE - WIFI

1. OBJETIVOS

- Comprensión de los principales comandos AT del módulo
- Comprensión de los principales comandos de la librería WIFI
- Configuración de la placa ESP32 como web server
- Enviar información a un servidor web

2. FUNDAMENTO TEÓRICO

2.1 Módulo ESP8266

EL ESP8266 es un chip que; gracias a la comunidad; sus características y un gran precio ha logrado convertirse en uno de los adaptadores Wifi más populares, y es que este chip alberga gran potencia en un pequeño tamaño. Entre sus principales características tiene:

- Soporta el protocolo 802.11 b/g/n
- **Capacidad para Wi-Fi Direct (P2P), Soft-AP**
- **Tiene integrado el stack del protocolo TCP/IP**
- Tiene un núcleo Diamond Standard Core (LX3) hecho por Tensilica
- Su núcleo de arquitectura RISC 32bits corre a 80Mhz
- 64KBytes de RAM de instrucciones
- 96KBytes de RAM de datos
- Los módulos cuentan con una memoria flash SPI Winbond W25Q40BVNIG

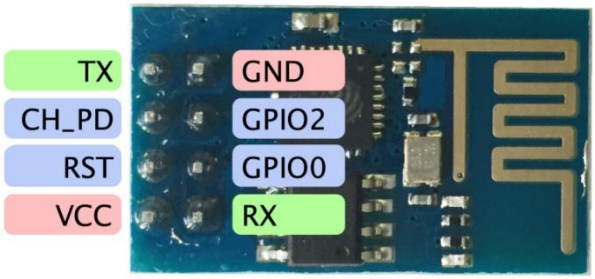

Imagen 1: Módulo ESP8266 y sus pines

El módulo viene inicialmente con el “firmware AT”, con el cual puedes usarlo como un adaptador wifi-serial, usando un microcontrolador externo como un Arduino para controlarlo y comunicarse con la red inalámbrica.

Como cualquier microcontrolador se puede cambiar el firmware del ESP8266 y usar cualquiera de los disponibles, entre los más destacables están, uLUA, microPython, y hasta programas hechos en Arduino.

2.2 Conexión del módulo con un Arduino

Se muestra a continuación el conexionado del módulo con el Arduino Mega:

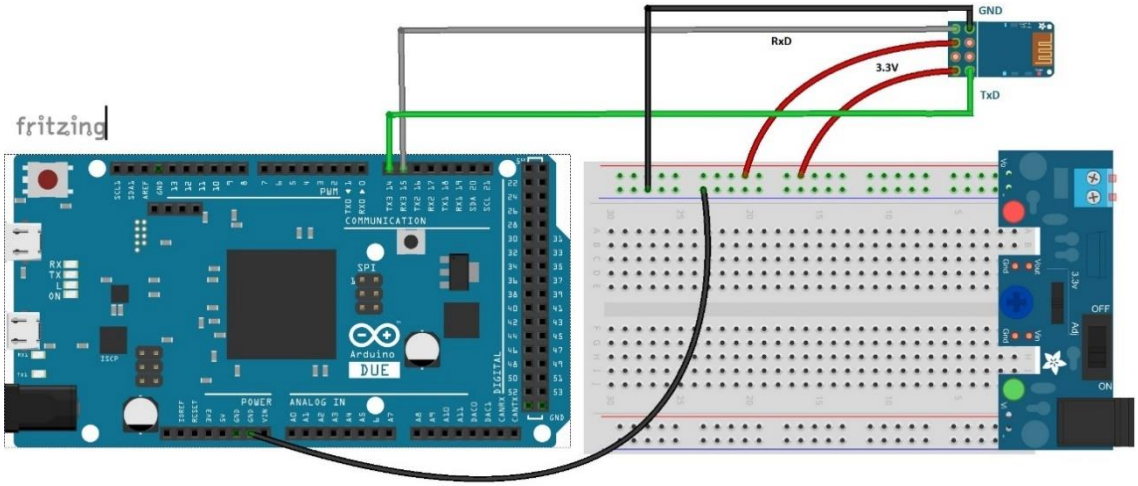

Imagen 2: Conexionado del módulo

- Se debe de tomar en cuenta lo siguiente al usar el módulo:
- El ESP8266 se alimenta con 3.3V, **¡no usar 5V!**
- El ESP8266 NO tiene entradas tolerantes a 5V, así que necesitas convertir el nivel de las entradas si quieres comunicarte con dispositivos de 5V
- Durante los periodos de comunicación inalámbrica el ESP8266 puede necesitar hasta 250mA, por lo que sin una buena fuente de alimentación el ESP8266 se puede reiniciar.
- Se debe de utilizar una fuente de alimentación externa.

2.3 Comandos AT

Al igual que el módulo bluetooth, este módulo posee unos comandos AT especiales para comunicarnos de manera óptima con nuestro módulo. A continuación, se muestran los comandos más utilizados, su función y su respuesta:

Function	***AT Command***	***Response***
Working	AT	OK
Restart	AT+RST	OK [System Ready, Vendor:www.ai-thinker.com]
Firmware version	AT+GMR	AT+GMR 0018000902 OK
List Access Points	AT+CWLAP	AT+CWLAP +CWLAP:(4,"RochefortSurLac",38,"70:62:b8:6f:6d:58",1) +CWLAP:(4,"LiliPad2.4",-83,"f8:7b:8c:1e:7c:6d",1) OK

Join Access Point	AT+CWJAP? AT+CWJAP="SSID","Password"	Query AT+CWJAP? +CWJAP:"RochefortSurLac" OK
Quit Access Point	AT+CWQAP=? AT+CWQAP	Query OK
Get IP Address	AT+CIFSR	AT+CIFSR 192.168.0.105 OK
Set Parameters of Access Point	AT+ CWSAP? AT+ CWSAP= <ssid>,<pwd>,<chl>, <ecn>	Query ssid, pwd chl = channel, ecn = encryption
WiFi Mode	AT+CWMODE? AT+CWMODE=1 AT+CWMODE=2 AT+CWMODE=3	Query STA AP BOTH
Set up TCP or UDP connection	AT+CIPSTART=? (CIPMUX=0) AT+CIPSTART = <type>,<addr>,<port> (CIPMUX=1) AT+CIPSTART= <id><type>,<addr>, <port>	Query id = 0-4, type = TCP/UDP, addr = IP address, port= port
TCP/UDP Connections	AT+ CIPMUX? AT+ CIPMUX=0 AT+ CIPMUX=1	Query Single Multiple
Check join devices' IP	AT+CWLIF	
TCP/IP Connection Status	AT+CIPSTATUS	AT+CIPSTATUS? no this fun
Send TCP/IP data	(CIPMUX=0) AT+CIPSEND=<length>; (CIPMUX=1) AT+CIPSEND= <id>,<length>	
Close TCP / UDP connection	AT+CIPCLOSE=<id> or AT+CIPCLOSE	
Set as server	AT+ CIPSERVER= <mode>[,<port>]	mode 0 to close server mode; mode 1 to open; port = port
Set the server timeout	AT+CIPSTO? AT+CIPSTO=<time>	Query <time>0~28800 in seconds
Baud Rate*	AT+CIOBAUD? Supported: 9600, 19200, 38400, 74880, 115200, 230400, 460800, 921600	Query AT+CIOBAUD? +CIOBAUD:9600 OK
Check IP address	AT+CIFSR	AT+CIFSR 192.168.0.106 OK
Firmware Upgrade (from Cloud)	AT+CIUPDATE	1. +CIPUPDATE:1 found server 2. +CIPUPDATE:2 connect server 3. +CIPUPDATE:3 got edition 4. +CIPUPDATE:4 start update
Received data	+IPD	(CIPMUX=0): + IPD, <len>: (CIPMUX=1): + IPD, <id>, <len>: <data>
Watchdog Enable*	AT+CSYSWDTENABLE	Watchdog, auto restart when program errors occur: enable

Watchdog Disable*	AT+CSYSWDTDISABLE	Watchdog, auto restart when program errors occur: disable

2.4 Librería WIFI en el ESP32

Al instalar la placa ESP32 Devkit 1 en el IDE del Arduino se instalan una serie de librerías que son muy útiles para nuestros proyectos. La librería WIFI nos ahorra mucho tiempo y líneas de código para conectarnos a una red. A continuación, veremos algunos de los comandos más característicos de la librería WIFI:

WiFi.mode(_____)

Con esta función configuramos el modo de funcionamiento del ESP32 como una estación, como acces point o como ambos, para ello usamos la siguiente tabla:

WiFi.mode(WIFI_STA)	Modo estación
WiFi.mode(WIFI_AP)	Modo access point: una estación se puede conectar al ESP32
WiFi.mode(WIFI_STA_AP)	Modo access point y estación

WiFi.softAP(ssid, password);

Con esta función le asignamos un nombre y una contraseña a nuestro ESP32 en modo estación. Este comando no es útil en modo STA. Si queremos que nuestro AP no tenga una contraseña, debemos de escribir NULL como password.

Como adicional tenemos la siguiente instrucción:

WiFi.softAP(const char* ssid, const char* password, int channel, int ssid_hidden, int max_connection)

- ssid: Nombre de nuestro access point – máximo 63 caracteres;
- password: mínimo 8 caracteres; si no se desea contraseña escribir NULL
- channel: número del canal del AP (1-13)
- ssid_hidden: (0 = SSID abierto, 1 = SSID oculto)
- max_connection: máximos clientes conectados en simultáneo (1-4)

WiFi.scanNetworks()

Devuelve el número de todas las redes WIFI cercanas, se recomienda usar una variable entera de la siguiente manera:

int = WiFi.scanNetworks();

WiFi.SSID()

Después de escanear las redes, esta función imprime los nombres de las redes encontradas

WiFi.begin(ssid, password);

Función utilizada para conectarnos a una red en específica, para ello debemos de definir a nuestro ESP32 en modo STA, el ssid y password deben de digitarse de manera manual.

WiFi.status().

Esta función da como respuesta los distintos estados en los que puede estar nuestro nodo en una red WIFI, todos los posibles estados están en la siguiente tabla:

Valor	Constante	Significado
0	WL_IDLE_STATUS	Estado temporal asignado cuando se llama a la función *WiFi.begin()*
1	WL_NO_SSID_AVAIL	El SSID no está disponible
2	WL_SCAN_COMPLETED	La búsqueda de las redes está completa
3	WL_CONNECTED	Se conectó a la red WiFi
4	WL_CONNECT_FAILED	Falla la conexión
5	WL_CONNECTION_LOST	Se ha perdido la conexión
6	WL_DISCONNECTED	Nos hemos desconectado de la red

WiFi.localIP()

Nos devuelve el valor de la IP de la red conectada, para ello se usa la siguiente instrucción: Serial.println(WiFi.localIP());

WiFi.disconnect()

Instrucción para desconectarnos de una red previamente enlazada con éxito

WiFi.reconnect()

Instrucción para reconectarse inmediatamente a una red en la que estuvo conectada en un corto lapso.

2.5 Modos de funcionamiento del ESP32

El microcontrolador tiene la capacidad de configurar sus funciones inalámbricas en dos modos de trabajo: modo estación y modo accespoint

Modo Estación (STA)

Cuando el ESP32 está configurado como una estación Wi-Fi, puede conectarse a otras redes. En este caso, el router asigna una dirección IP única a la placa ESP32. A partir de este punto nos podemos comunicar con el ESP utilizando otros dispositivos (estaciones) que también están conectados a la misma red haciendo referencia a la dirección IP única del ESP32.

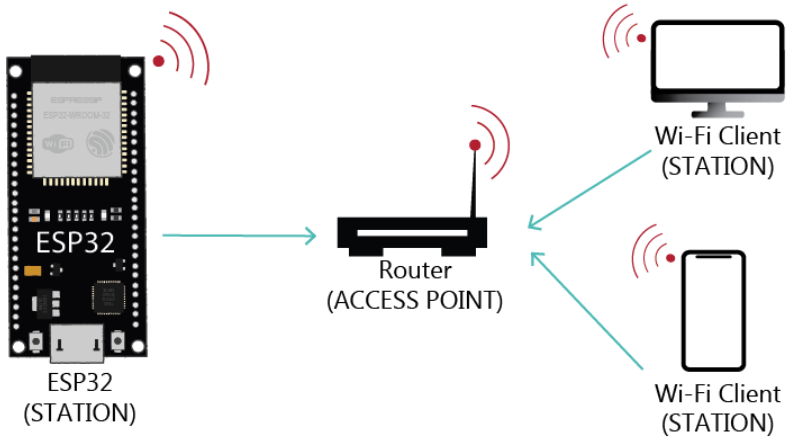

Imagen 3: Funcionamiento modo estación

El router estará conectado a Internet, por lo que podemos solicitar información de Internet utilizando la placa ESP32 como datos de API keys de otros servidores, publicar datos en plataformas en línea, usar iconos e imágenes de Internet o incluir bibliotecas de JavaScript para crear páginas de servidor web.

Modo Access point (AP)

Al configurar la placa ESP32 como un punto de acceso (Access point), puede conectarse usando cualquier dispositivo con capacidades de Wi-Fi sin conectarse a su router. Lo que conseguimos es crear una red Wi-Fi propia y los dispositivos (estaciones) Wi-Fi cercanos pueden conectarse a él, como un dispositivo móvil Smart (smarthphone, Tablet) o computadora. Por lo tanto, no es necesario que esté conectado a un router para controlarlo. Esto también puede ser útil si desea que varios dispositivos ESP32 se comuniquen entre sí sin la necesidad de un router especial.

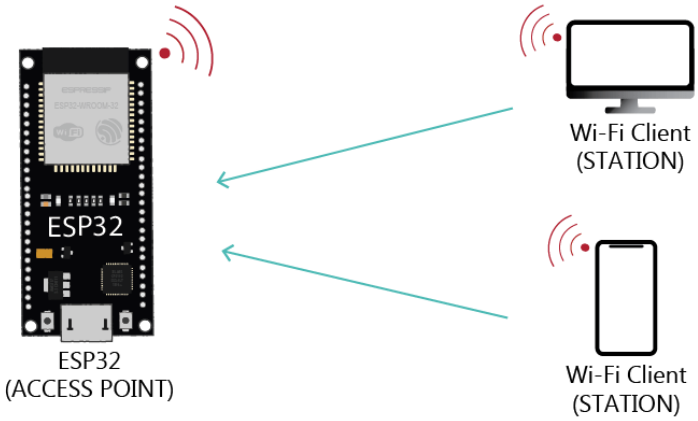

Imagen 4: Funcionamiento modo AP

El módulo viene inicialmente con el "firmware AT", con el cual puedes usarlo como un adaptador wifi-serial, usando un microcontrolador externo como un Arduino para controlarlo y comunicarse con la red inalámbrica.

Como cualquier microcontrolador se puede cambiar el firmware del ESP8266 y usar cualquiera de los disponibles, entre los más destacables están, uLUA, microPython, y hasta programas hechos en Arduino.

2.6 APIs

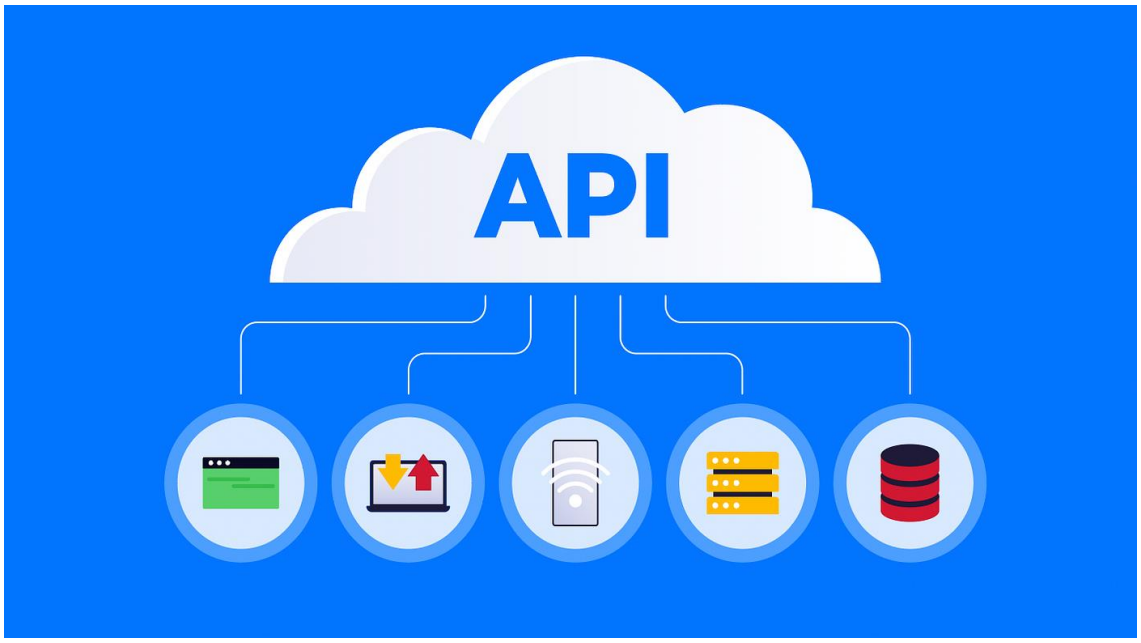

Imagen 5: Funcionamiento de las API

La terminología API proviene de Application Programming Interface (Interfaces de programación de aplicaciones), muy utilizada para el desarrollo de software y nuevos modelos de negocios. Son estas APIs las que permiten a los desarrolladores conectarse al mundo de los objetos y relacionar entre sí los dispositivos que nos rodean: 'enchufar'

nuestro cuerpo a medidores de constantes vitales, domotizar nuestra casa, convertir nuestro vehículo casi en un ser vivo… Las APIs se han convertido en la sangre del software y el llamado Internet de las Cosas no existiría sin ellas. Las APIs incorporan y trabajan con los datos que facilitan la conexión entre objetos.

La API de ThingSpeak siempre trabaja con datos, esa es su gran especialidad. Es una API abierta para el Internet de las Cosas que permite recopilar, almacenar, analizar, visualizar y actuar sobre la información recogida en sensores y dispositivos como aplicaciones web y móviles, redes sociales como Twitter, soluciones de mensajería, VoIP y nube como Twilio, hardware de código abierto como Arduino, Raspberry Pi o BeagleBone (los reyes del Internet de las Cosas y la robótica) o con lenguajes de cálculo computacional como MATLAB. ThingSpeak es una API conocida entre los desarrolladores y dispone ya de una gran comunidad.

ThingSpeak API funciona siempre con canales, los cuales contienen los campos de datos, ubicación y estado. Para empezar a trabajar con esta interfaz es necesario crear un canal, donde se recopilará la información de dispositivos y aplicaciones, datos que posteriormente se pueden analizar y visualizar en gráficos (este es un tutorial bastante completo de cómo crear gráficos con ThingSpeak) y el paso final es operar sobre esa documentación. El proceso con la API siempre es el mismo.

2.7 Servidores MQTT

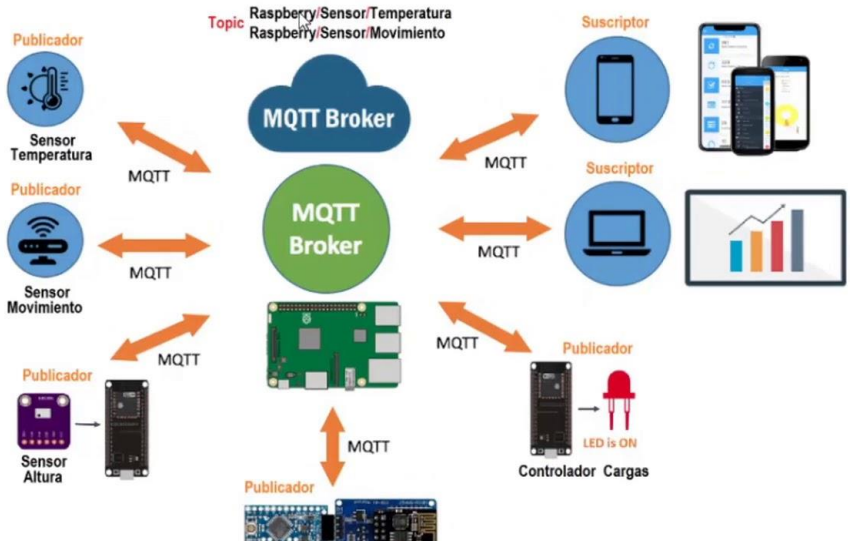

Imagen 6: Conexión MQTT

MQTT son las siglas MQ Telemetry Transport, aunque en primer lugar fue conocido como Message Queing Telemetry Transport. Es un protocolo de comunicación M2M

(machine-to-machine) de tipo message queue. Está basado en la pila TCP/IP como base para la comunicación. En el caso de MQTT cada conexión se mantiene abierta y se "reutiliza" en cada comunicación. Es una diferencia, por ejemplo, a una petición HTTP donde cada transmisión se realiza a través de conexión.

El funcionamiento del MQTT es un servicio de mensajería push con patrón publicador/suscriptor (pub-sub), en este tipo de infraestructuras los clientes se conectan con un servidor central denominado broker. Para filtrar los mensajes que son enviados a cada cliente los mensajes se disponen en topics organizados jerárquicamente. Un cliente puede publicar un mensaje en un determinado topic. Otros clientes pueden suscribirse a este topic, y el broker le hará llegar los mensajes suscritos.

3. SERVIDOR LOCAL

En los siguientes pasos usaremos unos códigos básicos para el correcto funcionamiento del modo de funcionamiento local del ESP32

3.1 Configuración del ESP32 como AP

Empezaremos configurando el ESP32 como un Access point siguiendo una serie de pasos, lo primero será definir el nombre de tu nuevo AP con los siguientes comandos:

```
const char* ssid = "ESP32 AP"; //nombre de la nueva red
const char* password = "123456789"; //contraseña de la red
```

Con estos datos podemos configurar el ESP32 en el Setup con la siguiente instrucción de la librería. Deben de recordar que la librería Wifi.h debe de estar incluida al inicio del código

```
WiFi.softAP(ssid, password);
```

Realizado este paso, debemos de conocer la IP que genera el ESP32 para poder conectarnos de manera directa.

```
IPAddress IP = WiFi.softAPIP();
Serial.print("AP IP address: ");
Serial.println(IP);
```

Con estas líneas ya nos podemos conectar a nuestra red creada desde cualquier dispositivo con conexión wifi digitando la IP generada. El código a usar será ejercicio1.ino

3.2 Configurar el ESP32 webServer para un encendido ON/OFF

Vamos a crea un servidor web independiente con un ESP32 que controla una salida de manera digital (un LED) utilizando el entorno de programación Arduino IDE. El servidor web responde a dispositivos móviles y se puede acceder a él con cualquier dispositivo que sea un navegador en la red local.

Para este caso usaremos el modo STA para poder monitorear la información del ESP desde cualquier punto conectado a la misma red Wifi. El primer paso es definir los parámetros de la red:

```
const char* ssid     = "SSID";
const char* password = "CONTRASEÑA";
```

El siguiente paso será habilitar el puerto 80 del servidor, esto sirve para poder escuchar cualquier petición de un cliente externo, esto funciona en el protocolo HTTP y poder usar el código HTML.

```
WiFiServer server(80);
```

Se recomienda usar unas variables de tiempo para la conexión de un cliente al ESP32, las variables recomendadas son las de tiempo actual, tiempo anterior y tiempo de desconexión con los siguientes comandos:

```
unsigned long currentTime = millis();
unsigned long previousTime = 0;
const long timeoutTime = 2000;
```

Es hora de conectarnos a la red WiFi, para ello hacemos uso de la librería <Wifi.h>. Una vez conectados debemos de conocer la IP asignada al ESP32 para poder ingresar en la misma red Wifi, para ello debemos de usar las siguientes instrucciónes:

```
WiFi.begin(ssid, password);
Serial.println(WiFi.localIP());
```

Para la espera de la entrada de nuevos clientes a nuestra red, debemos de usar el comando:

```
WiFiClient client = server.available()
```

De esta manera estamos esperando una información del puerto serial, en este caso del módulo WiFi del ESP32. Para que el módulo no se quede bloqueado o esperando indefinidamente la llegada de un nuevo cliente, realizaremos un nuevo bucle while con los tiempos configurados anteriormente, el bucle será similar al siguiente:

```
currentTime = millis();
previousTime = currentTime;
while (client.connected() && currentTime - previousTime <= timeoutTime)
```

Dentro del bucle se usará un código HTML para poder visualizar de una mejor manera la web dentro de la IP mostrada. El código se HTML se puede visualizar en el código ejercicio2 que se adjunta a la guía.

3.3 Configurar el ESP32 webServer para una entrada analógica

Explicado el caso anterior, es mucho más fácil colocar un valor de un sensor analógico en una web HTML generada por el ESP32. Tal como se vio en el encendido y apagado de un led, la configuración del ESP32 en modo STA y la conexión a una red existente son las mismas, lo único que se diferencia es como imprimir el valor dentro en el HTML generado.

No debemos de olvidar de usar los pines adecuados del ESP32, no todos valen como entradas analógicas y algunos están reservados.

Entre las líneas de código del HTML dentro del bucle de client, debemos de imprimir el valor de la lectura:

```
client.println(lectura_analógica);
```

Recordamos que existe diversas formas de mostrar los datos usando la programación HTML, con lo cual podemos mostrar la información de diversas maneras. Para poder visualizar mejor la visualización del envío de la información del pin analógico, usaremos el código ejercicio3.ino.

4. SERVIDORES GRATUITOS

En esta guía se utilizarán dos servidores gratuitos, el primero se llama ThingSpeak y el segundo se llama Adafruit IO. El primer servidor usar el protocolo HTTP mientras que el segundo usa el protocolo MQTT

4.1 ThingSpeak

Para el desarrollo de este laboratorio es básico crear una cuenta en dicho servidor, pueden utilizar cualquier correo, de preferencia usar el correo de la universidad para el desarrollo de los proyectos relacionados con la asignatura y la carrera.

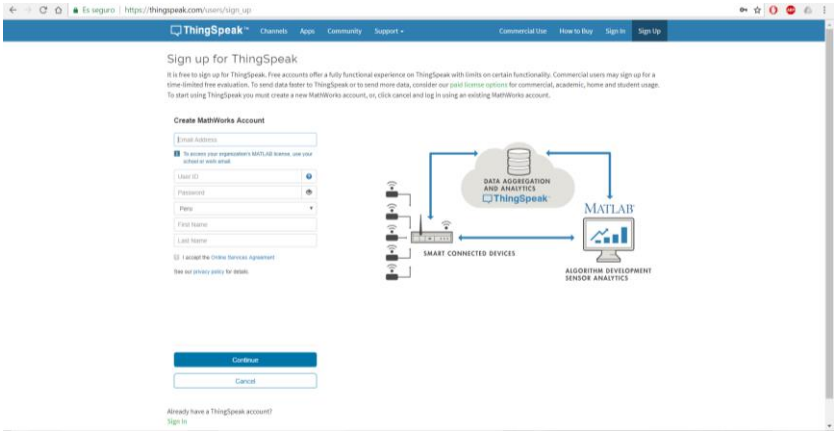

Imagen 7: Interfaz ThingSpeak

Una vez registrados, deberán de crear un nuevo canal y completar los campos requeridos:

- Nombre: PruebaSimple.
- Descripción: Primera prueba con el servidor.
- Activar el Field1 y colocar un nombre, en este caso Sensor analógico

El resultado final debería de ser algo similar a lo siguiente

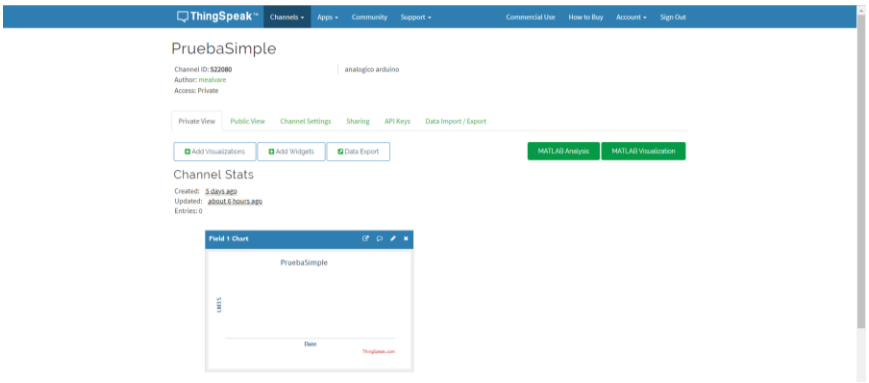

Imagen 8: Visualización de un campo en ThingSpeak

La ventaja de este servidor es la facilidad para cambiar los nombres y configuraciones. Con esta información procedemos a realizar el código Arduino para su comprobación.

4.2 Adafruit IO

Al igual que el Thinkercad, es necesario que se creen una cuenta en dicho servidor utilizando cualquier correo electrónico, de preferencia usar el correo de la universidad para evitar cualquier tipo de SPAM.

Finalizada la etapa de inscripción debemos de crear un nuevo Dashboard con el nombre "Prueba" para empezar a colocar los indicadores gráficos a utilizar en el laboratorio. Entramos en dicho dashboard y debemos de dar click en la tuerca de la derecha y crear un nuevo bloque, tal como se ve en la imagen:

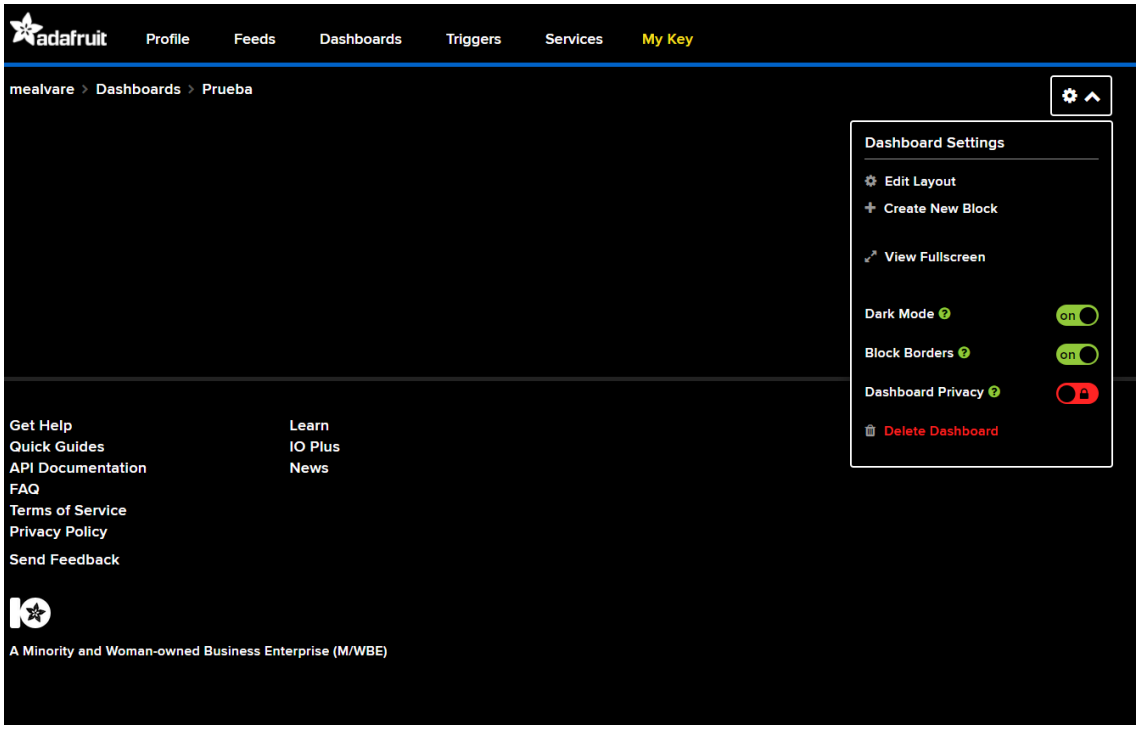

Imagen 9: Visualización del dashboard inicial

Del menú desplegado elegimos el toogle, también conocido como ON/OFF

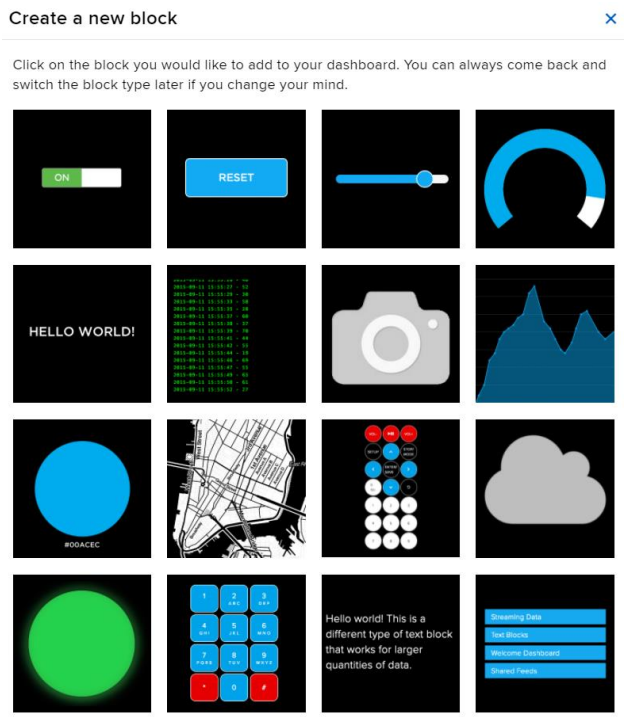

Imagen 10: Bloques para la creación del dashboard

Nos pedirá colocarle un nombre, en este caso se llamará onoff, damos click a siguiente paso para configurar el nombre del botón, tal como se ve en la imagen:

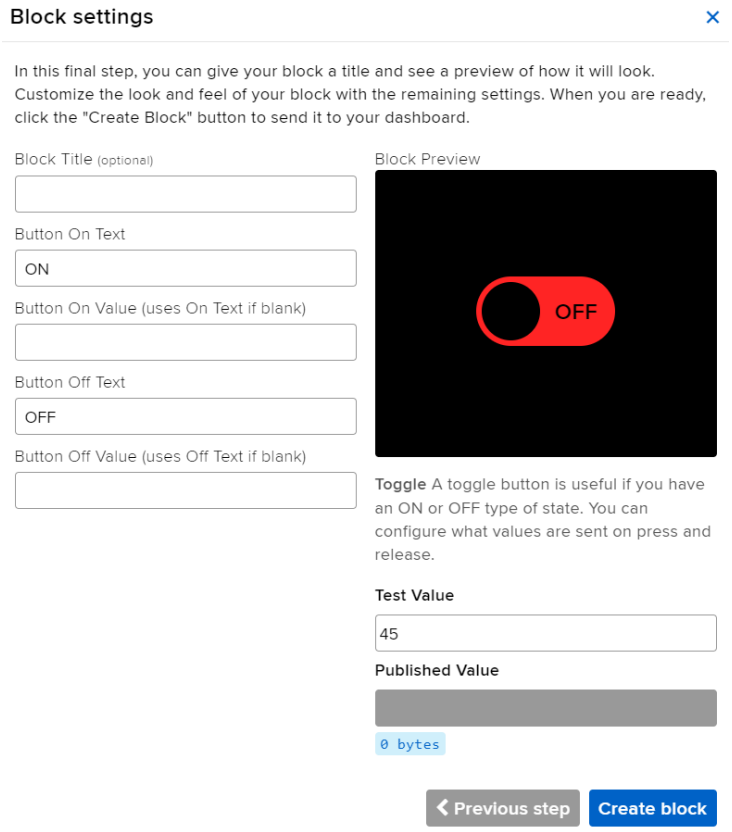

Imagen 11: Creación del botón On Off

Le pondremos como título: “encendido digital” y le damos OK. De esta manera sale la siguiente imagen:

Imagen 12: Visualización del botón On Off

5. Explicación códigos

Esta guía cuenta con un total de 5 códigos, los primeros 3 se basan en la configuración del ESP32 en un entorno local, los otros dos códigos se basan en la conexión del ESP32 con los servidores ThingSpeak y Adafruit IO

5.1 Código ThingSpeak

Para el envío de información a la nube usando el servidor gratuito mediante el protocolo HTTP debemos de usar las siguientes librerías para poder conseguir una correcta conexión a una red WIFI y a un servidor HTTP:

```
#include <WiFi.h>
#include <HTTPClient.h>
```

Tras las definiciones de las librerías, debemos de configurar la red WIFI, tal como el laboratorio anterior crearé unas variables tipo char para definir el nombre de la red y de la contraseña:

```
const char* ssid = "NOMBRE_DE_RED";
const char* password = "PASSWORD_DE_RED";
```

Otra variable global necesaria es la URL de mi servidor web, en este caso el Thingspeak, además de nuestra API key de escritura. No se recomienda usar el API de lectura al tener un desfase muy elevado de tiempo.

```
const char* serverName = "http://api.thingspeak.com/update";
String apiKey = "APIKEYPERSONAL";
```

Dentro del loop, tras realizar todas las acciones de control propio de nuestro sistema programado, crearemos un bucle IF que se mantendrá en ejecución siempre y cuando exista conexión. Dentro de este bucle IF inicializaremos nuestra conexión con el servidor HTTP y definimos el encabezado del contenido con los siguientes comandos:

```
http.begin(serverName);
http.addHeader("Content-Type", "application/x-www-form-urlencoded");
```

Ahora es el momento de enviar toda la data que queramos a los fields creados, en este caso solo tenemos un field y enviaremos una información aleatoria, la cual puede ser traducido perfectamente como una lectura analógica, para ello usamos el siguiente comando:

```
String httpRequestData = "api_key=" + apiKey + "&field1=" + String(random(40));
```

Si quieremos enviar más comandos solo debemos de aumentar la siguiente línea:

```
+ "&fieldX=" + String(valor);
```

Siendo la X el número de field que queremos completar, además recordamos que enviamos una cadena de texto, con lo que el valor será enviado como String

Una vez enviada la data, vamos a crear una variable para ver el código de respuesta del HTTP para asegurarnos que envía correctamente el código, eso lo logramos con las siguientes líneas:

```
int httpResponseCode = http.POST(httpRequestData);
Serial.print("Codigo de Respuesta HTTP: ");
Serial.println(httpResponseCode);//Si es 200 esta OK
```

Finalizamos el envío de la data con el comando:

```
http.end();
```

5.2 Código Adafruit IO

Para este tipo de servidor usaremos otras librerías para poder realizar la conexión MQTT, debemos de instalar las siguientes librerías:

```
#include <WiFi.h>
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"
```

La diferencia de esta librería es la definición de las variables para conseguir la conexión al servidor MQTT, recordamos que usamos el servidor Adafruit ya que es un servidor gratuito que nos permite enviar información sin problemas cada 2,5 segundos. Las variables son las siguientes:

```
#define AIO_SERVER "io.adafruit.com"
#define AIO_SERVERPORT 1883
#define IO_USERNAME  "nombre_de_usuario"
#define IO_KEY       "IO_key_personal"
```

Definidos los parámetros del servidor, debemos de crear el cliente para un servidor MQTT con los siguientes comandos:

```
WiFiClient client;
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, IO_USERNAME, IO_KEY);
```

En el servidor MQTT si podemos leer y escribir información sin problema gracias a ese corto tiempo de envío, además de ser un entorno mucho más amigable de manera visual, para ello usamos los comandos de publicar y subscribir con los respectivos códigos:

```
Adafruit_MQTT_Publish NOMBRE1 = Adafruit_MQTT_Publish(&mqtt, IO_USERNAME"/feeds/FEED_PERSONAL_PUBLICACIÓN");
```

```
Adafruit_MQTT_Subscribe NOMBRE2 = Adafruit_MQTT_Subscribe(&mqtt, IO_USERNAME "/feeds/FEED_PERSONAL_SUBSCRIPCIÓN");
```

La diferencia entre ambos de una manera más sencilla es la siguiente: la función publish sirve para todas las entradas mientras que las Subscribe para todas las salidas. En ambos casos sirven para señales tanto digitales como analógicas.

Para que las funciones de subscripción funcionen adecuadamente, realizaremos una subrutina de comparación de datos del feed con una acción en el ESP32, en este caso veremos primero el encendido y apagado de un actuador, para ser precisos el led interno del controlador. Para ello usamos la siguiente subrutina:

```
void digitalCallback(char *data, uint16_t len) {
Serial.print("Activamos la llamada de retorno digital, el boton se encuentra: ");
Serial.println(data);
String message = String(data);
message.trim();
if (message == "ON") {digitalWrite(led, HIGH);}
if (message == "OFF") {digitalWrite(led, LOW);}
}
```

Pero si lo que queremos es hacer un encendido usando la señal PWM, se creó la subrutina analógica, comparando la data de entrada con la escritura del PWM del led, la subrutina es la siguiente:

```
void analogicaCallback(char *data, uint16_t len) {
  Serial.print("Activamos la llamada de retorno analogico, el valor PWM es: ");
  Serial.println(data);
  String message = String(data);
  message.trim();
  ledcWrite(ledChannel, message.toInt());
}
```

En el Setup del programa realizaremos la conexión de la placa en una red WIFI como se hizo en el laboratorio pasado al configurar la placa en modo STA y de igual manera que se usó en el servidor HTTP. Lo que haremos es habilitar las funciones digitales y analógicas con las siguientes instrucciones para la subscripción digital y analógica respectivamente:

```
encendido.setCallback(digitalCallback);
mqtt.subscribe(&encendido);
pwm.setCallback(analogicaCallback);
mqtt.subscribe(&pwm);
```

No se debe de olvidar de configurar el resto de los parámetros para un correcto funcionamiento, eso incluye los pines del controlador, monitor serial, etc.

Como parte principal vamos a revisar el loop. Vamos a hacer uso de 2 líneas especiales al inicio y final de todo el proceso relacionado al envío y recepción de data al servidor MQTT. Esta función sirve para enviar data al servidor y para poder recibir información, sin estas líneas no funcionará nada:

```
MQTT_connect();
mqtt.processPackets(tiempo);
```

Realizamos la conexión al servidor MQTT y el tiempo que daremos para poder tratar toda la información posible del servidor, se recomienda modificar el tiempo en relación a la cantidad de data a enviar para evitar saturación del servidor, vamos a colocar inicialmente 1 segundo para nuestras pruebas.

Vamos a crear una subrutina para realizar la conexión MQTT, se recomienda no modificar el código para un correcto funcionamiento, para ello usaremos la función .connected() de las librerías iniciales, a continuación toda la subrutina:

```
void MQTT_connect() {
int8_t ret;
    if (mqtt.connected()) {
    return;
    }
Serial.print("Conectando a MQTT... ");
uint8_t retries = 3;
while ((ret = mqtt.connect()) != 0) { // connect devolverá 0 para conectado
Serial.println(mqtt.connectErrorString(ret));
Serial.println("Re-intentando la conexión MQTT en 5 segundos...");
mqtt.disconnect();
delay(5000); // espera 5 segundos
retries--;
    if (retries == 0) {
    // La conexión falla y espera a que se reinicie
    while (1);
    }
}
Serial.println("MQTT Conectado!");
}
```

Para el envío de data desde el ESP32 hacia el servidor MQTT usaremos la función .publish en su respectivo feeds dentro de un bucle if, esto se realiza para poder visualizar si se publicó de manera correcta con la sentencia not del bucle if. A continuación, se muestra un código para entender la publicación:

```
if (! feed.publish(valor)) {
Serial.println(F("Failed"));
} else {
Serial.println(F("OK!"));
}
```

Se puede utilizar un valor random para poder visualizar un valor en caso de no tener un correcto sensor analógico, también se puede usar un valor de un botón o entrada digital. Para esta última opción se recomienda usar un código que solo publique una modificación del valor actual y no estar publicando datos de manera innecesaria, se muestra un código ejemplo para un botón:

```
if (strPulsador != strPulsadorUltimo) { //envia el estado del pulsador solo cuando cambia.
   strPulsadorUltimo = strPulsador;
   strPulsador.toCharArray(charPulsador, 15);
   Serial.print(F("\nSending pulsador val "));
   Serial.print(strPulsador);
   Serial.print("...");
   if (! feed.publish(charPulsador)) {
     Serial.println(F("Failed"));
   } else {
     Serial.println(F("OK!"));
   }
}
```

El código explicado corresponde a las cuatro alternativas que podemos realizar en el servidor desde el ESP32: enviar el valor de un sensor digital, enviar el valor de un sensor analógico, activar una salida digital y activar una salida PWM. Es recomendable usar una red externa de la universidad para poder conectarnos adecuadamente, esta puede ser la red generada por nuestros datos sin WIFI de la universidad.