



## Práctica: Web Socket y ESP32 básico

### Objetivo:

- Emplear una variante distinta al ESP32-CAM
- Conocer y emplear un web socket con el fin de poder ampliar las opciones de uso de un microcontrolador

### Introducción

Dentro de la familia de chips ESP32” de ESPRESSIF encontramos por un lado al ESP32-CAM que es el que previamente hemos utilizado, sin embargo, no solamente es la única versión que existe, en este caso en concreto se empleará la versión **ESP32 Development Board**, la cual se caracteriza principalmente por tener más pines de entrada y salida, sin embargo, una de las mayores desventajas que presenta es que no tiene una cámara integrada dentro del mismo chip.

Por otro lado, una de las tecnologías a emplear en esta práctica es un Web Socket que proporcionará un canal de comunicación bidireccional sobre socket con protocolo de comunicación TCP, este tipo de socket está diseñado para ser implementado en navegadores y servidores web, es decir, sirve de manera general para cualquier aplicación que emplea la arquitectura cliente/servidor.

### Descripción:

Lo primero que debemos realizar y entender es el patrón de diseño denominado como “Cliente - Servidor”, el cual es uno de los más utilizados comercialmente, este patrón de diseño se caracteriza principalmente de comprender de manera general 2 integrantes:

El primero denominado **Servidor** que será la computadora encargada de desplegar todo el Backend o parte de procesamiento de datos, por otro lado, tenemos el **Cliente** que es la computadora o dispositivo que solamente consume los datos o información que despliega el servidor, a continuación, se muestra cómo sería este patrón de diseño empleando el ESP32

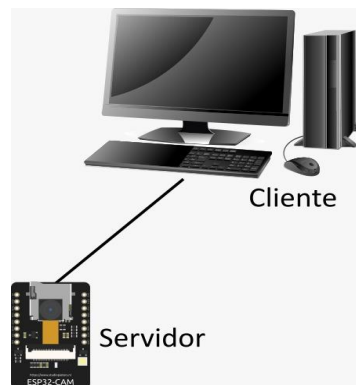


Figura 1. Arquitectura y patrón de diseño Cliente - Servidor



Entendiendo que comprende e integra la arquitectura o patrón de diseño Cliente Servidor, ahora podemos abordar lo que es un WebSocket, que será lo que se desarrollará en la presente práctica.

### Tabla de entradas y salidas:

A continuación, se muestra una imagen con las entradas y salidas del ESP32-CAM:

Tabla 1. Puertos a utilizar del ESP-32

Puerto	Tipo	Descripción
GPIO 26	Output	Se empleará para mandar el estado del led 1
GPIO 27	Output	Se empleará para mandar el estado del led 2

NOTA: En el caso del ESP32 tarjeta de desarrollo o “Development Board” directamente ya tenemos integrado una conexión USB para poder descargar programas, por lo que no es necesario utilizar puertos explícitos de nuestro microcontrolador.

### Diagrama de conexiones:

A continuación, se muestra el diagrama de conexiones, además de nuestro microcontrolador haremos uso de 2 resistencias y dos leds para poder prenderlos o apagarlos a través del web socket empleando el Wifi integrado en el ESP32

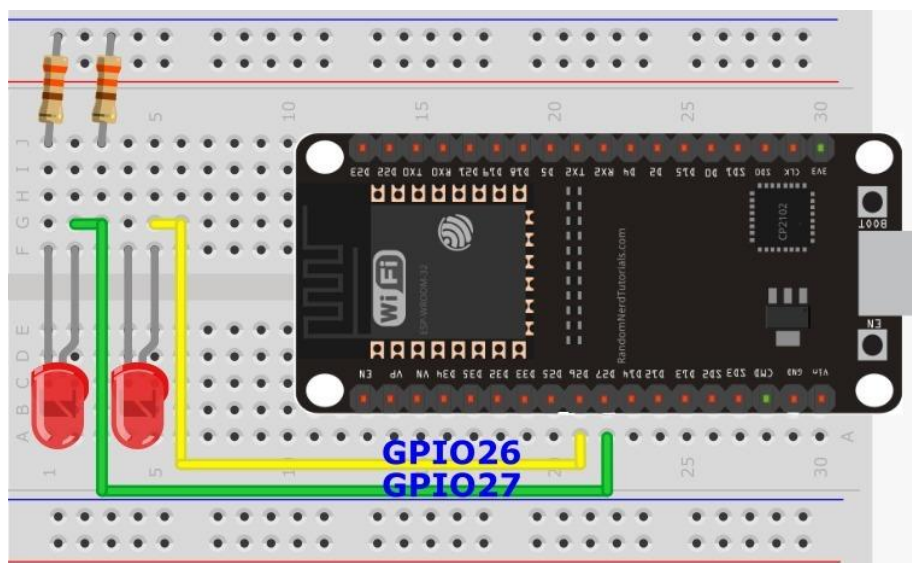


Figura 2. Esquema de conexiones entre los LEDs y el ESP32

### Listado del programa y descripción:

A continuación, se muestra paso a paso cada una de las partes que conforman al código encargado del funcionamiento de esta práctica, cabe destacar que para conocer la información enviada a nuestro microcontrolador haremos uso de una interfaz web desarrollada mediante código en HTML y CSS básico:



### 1. Definición de puertos y configuración de estos

```
#include <WiFi.h>

// Replace with network credentials
const char* ssid = "murryFly";
const char* password = "contrasena";

// Web server port number to 80
WiFiServer server(80);

// Variable to store the HTTP request
String header;

// Auxiliar variables to store the current output state
String output26State = "off";
String output27State = "off";

// Assign output variables to GPIO pins
const int output26 = 26;
const int output27 = 27;

// Current time
unsigned long currentTime = millis();

// Previous time
unsigned long previousTime = 0;

// Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 2000;
```

Lo primero que debemos hacer es importar la biblioteca **wifi.h** la cual la encontraremos dentro del paquete de bibliotecas previamente descargados en la práctica 1 de configuración de ambiente. Cabe destacar que en caso de haber trabajado previamente con Arduino es necesario desinstalar o renombrar la biblioteca **wifi.h** de Arduino para no tener problemas con duplicidad de bibliotecas o entidades.

Posteriormente declarar cadenas implícitas con los datos de nuestra respectiva conexión wifi, en este caso el nombre de mi wifi o SSID es murryFly, además y parte muy importante del web socket declarar explícitamente el puerto 80 que es el encargado en cualquier computadora de poder hacer la conexión remota a un server en una computadora.

Por último, agregaremos las correspondientes variables encargadas de hacer explícitos los valores que se mandarán a través de los puertos de nuestro microcontrolador.



## 2. Función setup (Configuración del proyecto y sus respectivas variables)

```
void setup() {  
    Serial.begin(115200);  
  
    // Initialize the output variables as outputs  
    pinMode(output26, OUTPUT);  
    pinMode(output27, OUTPUT);  
  
    // Set outputs to LOW  
    digitalWrite(output26, LOW);  
    digitalWrite(output27, LOW);  
  
    // Connect to Wi-Fi network with SSID and password  
    Serial.print("Connecting to ");  
    Serial.println(ssid);  
    WiFi.begin(ssid, password);  
  
    while (WiFi.status() != WL_CONNECTED) {  
        delay(500);  
        Serial.print(".");  
    }  
  
    // Print local IP address and start web server  
    Serial.println("");  
    Serial.println("WiFi connected.");  
    Serial.println("IP address: ");  
    Serial.println(WiFi.localIP());  
    server.begin();  
}
```

Como anteriormente se ha realizado, necesitamos hacer explícita cierta información de si ha sido exitoso o no el cargado de nuestro programado, es por ello que a través de esta función es como mandaremos datos al puerto serial de información relevante para su posterior uso del web socket.

Concretamente podemos observar que se mandará la dirección IP en el monitor serial una vez compilado y cargado el programa en nuestro microcontrolador.

## 3. Función loop (Web Socket y desarrollo web básico)

Debido a que esta práctica está pensada solamente en emplear un web socket como medio de comunicación entre un cliente y un servidor, directamente el desarrollo web se hará en el código en C en nuestro Arduino IDE, sin embargo, es necesario mencionar que existen otras formas de hacer el desarrollo del Front End o de la interfaz de usuario sin hacerlo directamente en código en C.



```
void loop() {
    WiFiClient client = server.available(); // Listen for incoming clients

    if (client) { // If a new client connects,
        currentTime = millis();
        previousTime = currentTime;

        Serial.println("New Client."); // print a message out in the serial port
        String currentLine = ""; // make a String to hold incoming data from the client

        while (client.connected() && currentTime - previousTime <= timeoutTime) { // loop while the c
            currentTime = millis();
            if (client.available()) { // if there's bytes to read from the client,
                char c = client.read(); // read a byte, then
                Serial.write(c); // print it out the serial monitor
                header += c;
                if (c == '\n') { // if the byte is a newline character
                    // if the current line is blank, you got two newline characters in a row.
                    // that's the end of the client HTTP request, so send a response:
                    if (currentLine.length() == 0) {
                        // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
                        // and a content-type so the client knows what's coming, then a blank line:
                        client.println("HTTP/1.1 200 OK");
                        client.println("Content-type:text/html");
                        client.println("Connection: close");
                        client.println();

                        // turns the GPIOs on and off
                        if (header.indexOf("GET /26/on") >= 0) {
                            Serial.println("GPIO 26 on");
                            output26State = "on";
                            digitalWrite(output26, HIGH);
                        } else if (header.indexOf("GET /26/off") >= 0) {
                            Serial.println("GPIO 26 off");
                            output26State = "off";
                            digitalWrite(output26, LOW);
                        } else if (header.indexOf("GET /27/on") >= 0) {
                            Serial.println("GPIO 27 on");
                            output27State = "on";
                            digitalWrite(output27, HIGH);
                        } else if (header.indexOf("GET /27/off") >= 0) {
                            Serial.println("GPIO 27 off");
                            output27State = "off";
                            digitalWrite(output27, LOW);
                        }
                    }
                }
            }
        }
    }
}
```

Lo primero que haremos es declarar uno de los objetos o estructuras de datos incluido en la biblioteca wifi.h, específicamente haremos una instancia de un servidor web, el cual una vez instanciado estará siempre pendiente de las peticiones que le hagamos como cliente.

Posterior a ello, las siguientes líneas de código se encargan de hacer la obtención de los estados mandados por el cliente a través de la interfaz web, además recordemos que al ser la función



loop, es decir aquella que siempre estará corriendo en nuestro microcontrolador, estas peticiones realizadas por el cliente siempre serán escuchadas por el servidor.

Por último, debido a que el cliente debe ver no solamente reflejado sus cambios físicamente sino también en la misma interfaz gráfica, a continuación, se muestra la parte de la función loop encargada, primero de mostrar la interfaz web, además de realizar los cambios realizados por el mismo cliente al determinar los estados de los leds:

```
// Display the HTML web page
client.println("<!DOCTYPE html><html>");
client.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">");
client.println("<link rel=\"icon\" href=\"data:,\">");

// CSS to style the on/off buttons
// Feel free to change the background-color and font-size attributes to fit your preferences
client.println("<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}");
client.println(".button { background-color: #0101fe; border: none; color: white; padding: 16px 40px;");
client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer;}");
client.println(".button2 {background-color: #555555;}</style></head>");

// Web Page Heading
client.println("<body><h1>ESP-32 Web Server </h1>");

// Display current state, and ON/OFF buttons for GPIO 26
client.println("<p>GPIO 26 - State " + output26State + "</p>");
// If the output26State is off, it displays the ON button

if (output26State=="off") {
  client.println("<p><a href=\"/26/on\"><button class=\"button\">ON</button></a></p>");
} else {
  client.println("<p><a href=\"/26/off\"><button class=\"button button2\">OFF</button></a></p>");
}

// Display current state, and ON/OFF buttons for GPIO 27
client.println("<p>GPIO 27 - State " + output27State + "</p>");
// If the output27State is off, it displays the ON button
if (output27State=="off") {
  client.println("<p><a href=\"/27/on\"><button class=\"button\">ON</button></a></p>");
} else {
  client.println("<p><a href=\"/27/off\"><button class=\"button button2\">OFF</button></a></p>");
}
client.println("</body></html>");

// The HTTP response ends with another blank line
client.println();
// Break out of the while loop
break;

else { // if you got a newline, then clear currentLine
  currentLine = "";
```

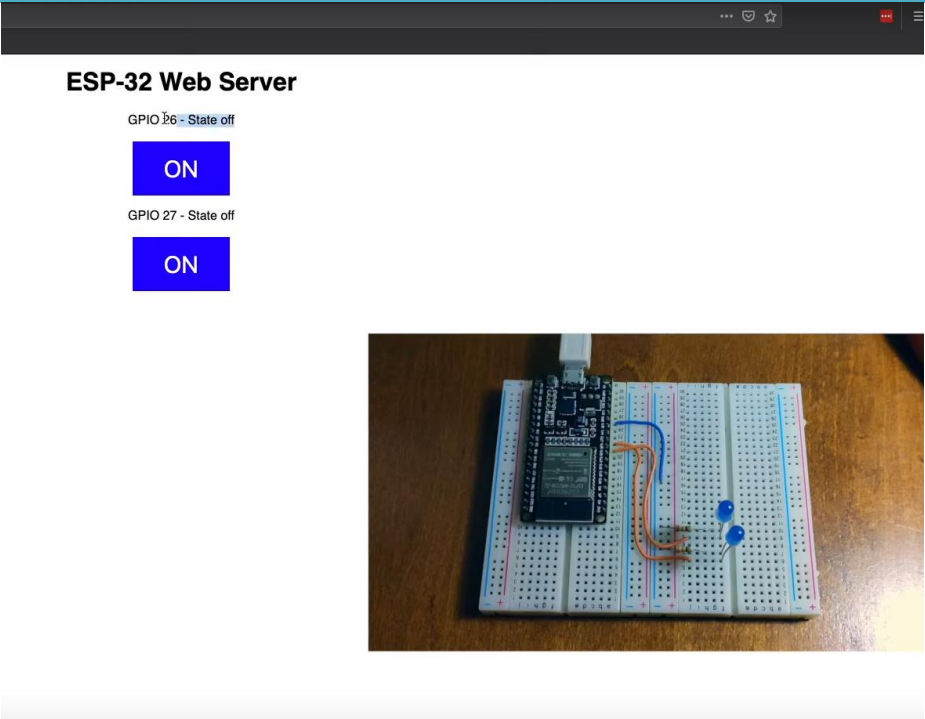
Como se puede observar, a pesar de que la parte externa de cada objeto y método está escrita en c, los argumentos que se mandan al servidor son código HTML y CSS que es la forma en que cualquier página web funciona.

#### 4. Resultado y manejo de la interfaz



Como bien se menciona en el anterior punto, la interfaz de usuario hace uso de 2 botones para el determinar el estado de cada uno de los leds. A continuación, algunas fotografías de los estados/resultados obtenidos mediante el web socket:

Tabla 1. Resultados obtenidos tras usar el web socket para prender los leds

Estados / resultados	Imagen descriptiva
<b>Ambos leds apagados</b>	





Un led prendido  
y otro apagado

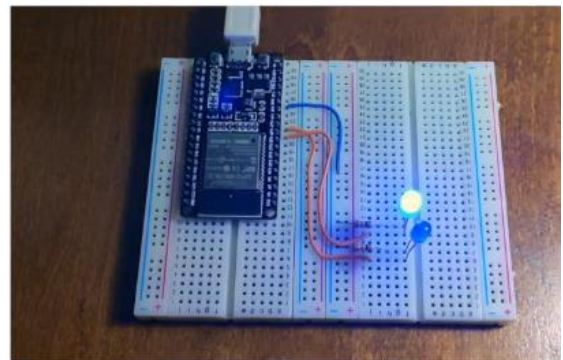
### ESP-32 Web Server

GPIO 26 - State on

OFF

GPIO 27 - State off

ON



### Notas y reglas de funcionamiento

En el caso del módulo de desarrollo ESP32 a diferencia del ESP32-CAM no hay necesidad de desconectar ningún cable entre pines, solamente con oprimir el botón de reset confirmamos el programa en memoria.

### Recursos extras:

Video de funcionamiento de la presente práctica:

<https://youtu.be/aVfBxol9CI8>