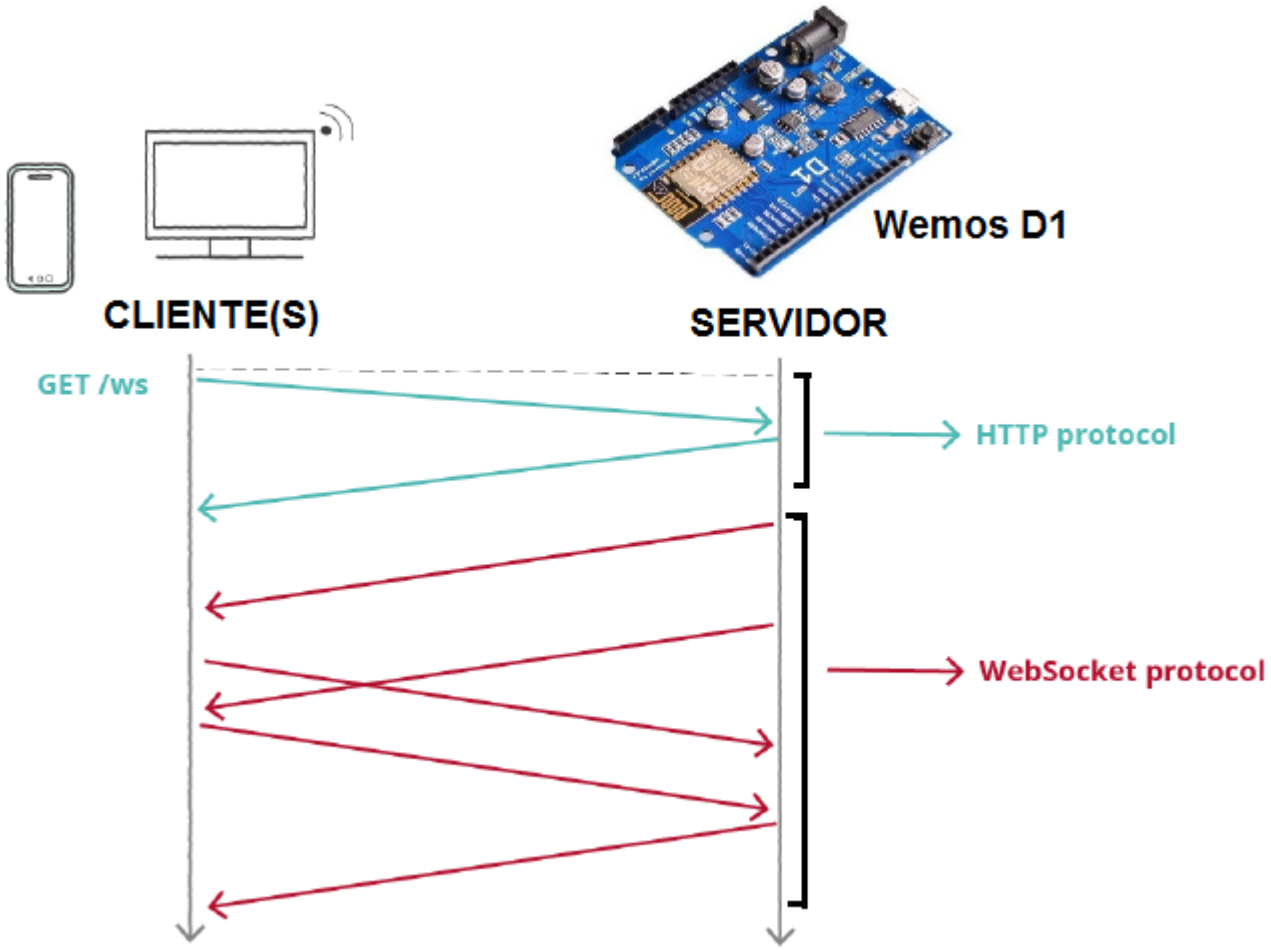
SERVIDOR WEB PARA CONTROL DE SALIDA MEDIANTE WEBSOCKET.

Este programa los realizamos previamente, en esta ocasión lo realizaremos mediante sockets. En el navegador de los clientes conectados puede observarse el cambio de estado que alguno de los clientes realice.

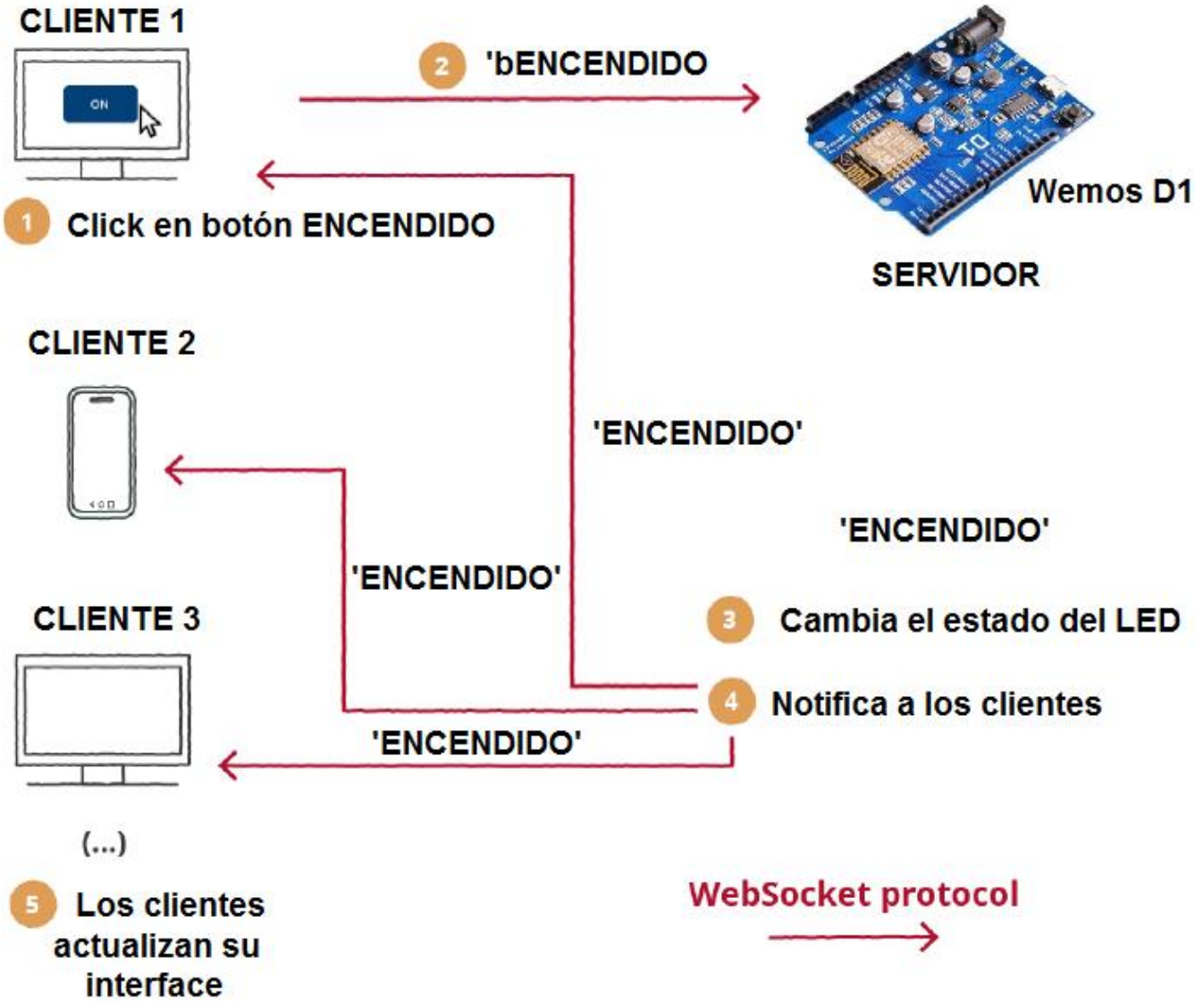
Un WebSocket es una conexión persistente entre un cliente y un servidor que permite comunicación bidireccional entre ambas partes mediante una conexión TCP. Esto significa que puede enviar datos del cliente al servidor y del servidor al cliente en cualquier momento.



El cliente establece una conexión WebSocket con el servidor a través de un proceso conocido como protocolo de enlace de WebSocket. El reconocimiento comienza con un HTTP solicitud / respuesta, lo que permite a los servidores manejar conexiones HTTP, así como Conexiones WebSocket en el mismo puerto. Una vez establecida la conexión, el cliente y el servidor pueden enviar datos de WebSocket en modo dúplex completo.

Al ejecutar el servidor debe aparecer la siguiente interface gráfica:

El siguiente diagrama ilustra lo que sucede cuando se pulsa el botón encendido:



LISTA DE MATERIALES:
1 Wemos D1 R1
1 Cable USB a USB micro.

ARCHIVO main.cpp

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <LittleFS.h>

const char* ssid = "MEGACABLE-F821E";
const char* password = "eWR237Ps";
// crea el objeto AsyncWebServer en el puerto 80
AsyncWebServer server(80);
/* La biblioteca ESPAsyncWebServer incluye un complemento WebSocket que facilita el
manejar las conexiones WebSocket. Crear un objeto AsyncWebSocket llamado ws para
manejar las conexiones en la ruta /ws  */
AsyncWebSocket ws("/ws");
// Establece el LED 2 de la tarjeta Wemos D1
const int ledPin = 2;
// Almacena el estado de LED
bool EstadoDeLED = 0;
// Inicializa LittleFS
void initFS() {
 if (!LittleFS.begin()) {
 Serial.println("Un error ha ocurrido mientras se montaba LittleFS");
 }
 Serial.println("LittleFS montado exitosamente ");
}
// Inicializa Wi Fi
void initWiFi() {
 WiFi.mode(WIFI_STA);
 WiFi.begin(ssid, password);
 Serial.print("Conectando a Wi Fi ..");
 while (WiFi.status() != WL_CONNECTED) {
 Serial.print('.');
 delay(1000);
 }
 Serial.println(WiFi.localIP());
}
// Reemplaza marcadores de posición con el valor del estado del LED
String processor(const String& var){
/* cuando la página web es solicitada, el ESP8266 verifica si el texto HTML
tiene marcadores de posición, se lee el estado actual del GPIO con
digitalRead(ledPin) se establece el valor de la variable de acuerdo al
estado en que se encentre el LED) */
```

```
 if(var == "ESTADO"){
 if(digitalRead(ledPin)){
 EstadoDeLED = 0;
 return "ENCENDIDO";
 }
 else{
 EstadoDeLED = 1;
 return "APAGADO";
 }
 }
 return String();
}
void notifyClients(String estado) {
 ws.textAll(estado);  // notifica el estado a todos los clientes
}  /* les envía el mismo mensaje a todos los clientes que estén
 conectados, les hace el envío al mismo tiempo
 La clase AsyncWebSocket provee el método .textAll para estar
 enviando el mismo mensaje a todos los clientes al mismo tiempo */
void handleWebSocketMessage(void *arg, uint8_t *data, size_t len)
{ /* La función handleWebSocketMessage () es una función de devolución
de llamada que se ejecutará siempre que recibamos nuevos datos de los
clientes a través del protocolo WebSocket.*/
 AwsFrameInfo *info = (AwsFrameInfo*)arg;
 if (info->final && info->index == 0 && info->len == len && info->opcode == WS_TEXT) {
 data[len] = 0;
 /*Si recibimos el mensaje "bENCENDIDO", se establece el valor de la variable
 EstadoDeLED en 0 y se notifica a todos los clientes llamando a la función
 notifyClients ().*/
 if (strcmp((char*)data, "bENCENDIDO") == 0) {
 EstadoDeLED = 0;
 notifyClients("ENCENDIDO"); //Notifica a todos los clientes
 }
  /*Si recibimos el mensaje "bAPAGADO", se establece el valor de la variable
 EstadoDeLED en 1 y se notifica a todos los clientes llamando a la función
 notifyClients ().*/
 if (strcmp((char*)data, "bAPAGADO") == 0) {
 EstadoDeLED = 1;
 notifyClients("APAGADO"); //Notifica a todos los clientes
 }
 }
}
/* Ahora necesitamos configurar un detector de eventos para manejar los
diferentes pasos asincrónicos pasos del protocolo WebSocket.
Este controlador de eventos se puede implementar definiendo
onEvent () de la siguiente manera: */
void onEvent(AsyncWebSocket *server, AsyncWebSocketClient *client, AwsEventType type,
 void *arg, uint8_t *data, size_t len) {
 // El argumento de tipo representa el evento que ocurre. Puede tomar los siguientes valores:
 switch (type) {
```

```
 case WS_EVT_CONNECT: // cuando un cliente ha iniciado sesión
 Serial.printf("WebSocket client #%u connected from %s\n", client->id(), client-
>remoteIP().toString().c_str());
 break;
 case WS_EVT_DISCONNECT: // cuando un cliente se desconecta
 Serial.printf("WebSocket client #%u disconnected\n", client->id());
 break;
 case WS_EVT_DATA: // cuando se recibe un paquete de datos del cliente
 handleWebSocketMessage(arg, data, len);
 break;
 case WS_EVT_PONG: // en respuesta a una solicitud de ping
 /* Ping es un comando o una herramienta de diagnóstico que permite hacer una verificación
  del estado de una determinada conexión o host local.
Es un acrónimo para Packet Internet Groper, lo que literalmente significa
 “buscador de paquetes en redes”. Se trata de un comando que permite verificar el estado de
 una conexión para determinar si una dirección IP específica o host es accesible desde la red o no. */
 case WS_EVT_ERROR: // cuando se recibe un error del cliente.
 break;
 }
}
void initWebSocket() { // inicializa el protocolo WebSocket
 ws.onEvent(onEvent);
 server.addHandler(&ws);
}
void setup() {
 pinMode(ledPin, OUTPUT); // Se prepara ledPn como SALIDA
 Serial.begin(115200);
 initFS(); // se inicializa Littlefs
 initWiFi(); // inicializa Wi Fi
 initWebSocket(); // inicializa el protocolo Websocket
 /* Las siguientes líneas manejan lo que sucede cuando recibe una
 solicitud en la raíz (/) URL (dirección IP de ESP8266).
 Cuando recibe esa solicitud, envía el texto HTML guardado en el archivo
 index.html a construir la página web. También necesita pasar la función
  del procesador como argumento para que reemplace todos los marcadores
  de posición con los valores correctos. */
 server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
 request->send(LittleFS, "/index.html", "text/html",false, processor);
 });
/* El primer argumento de la función send () es el sistema de archivos donde
se guardan los archivos, en este caso se guarda en SPIFFS (o LittleFS en
el caso del ESP8266).
El segundo El argumento es la ruta donde se encuentra el archivo.
 El tercer argumento se refiere a la tipo de contenido (texto HTML).
 El tercer argumento significa descarga = falso.
 Finalmente, el último argumento es la función del procesador. */

 /* Cuando el archivo HTML se cargue en su navegador, solicitará el CSS,
Archivos JavaScript y favicon. Estos son archivos estáticos guardados en
```

```
el mismo directorio (SPIFFS o LittleFS). Entonces, podemos simplemente
agregar la siguiente línea para servir archivos en un directorio cuando
sea solicitado por la URL raíz. Servirá los archivos CSS y favicon
automáticamente (también se agrega una imagen en este ejemplo) */
 server.serveStatic("/", LittleFS, "/");
 // Inicia servidor
 server.begin();
}
void loop() {
 ws.cleanupClients(); /* Los navegadores a veces no cierran correctamente
 la conexión WebSocket, incluso cuando se llama a la función close () en
 JavaScript. Esto eventualmente agotará los recursos del servidor web y hará
 que el servidor se bloquee. Llamar periódicamente a la función
 cleanupClients () desde el bucle principal () limita el número de clientes
 cerrando el cliente más antiguo cuando se ha superado el número máximo
 de clientes. Esto se puede llamar en cada ciclo, sin embargo, si desea
 usar menos energía, entonces es suficiente llamar con tan poca frecuencia
 como una vez por segundo. Si se incluye el valor 50 entre paréntesis podemos
ampliar la cantidad de clientes que perciben los cambios en la página web
del servidor embebido*/
 digitalWrite(ledPin, EstadoDeLED); //escribe el estado del LED en ledPin
}
```

ARCHIVO HTML

```
<!DOCTYPE html>
<html>
<head>
 <title>Botones Web Socket</title>
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="icon" type="image/png" href="favicon.png">
 <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/al
l.css" integrity="sha384-
fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr" crossorigin="an
onymous">
 <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
 <div class="marquesina">
 <h1>SERVIDOR WEB WEMOS D1 EN MODO SOCKETS</h1>
 </div>
 <div class="contenido">
 <div class="cuadricula-tarjeta">
 <div class="tarjeta">
 <p class="titulo-tarjeta"><i class="fas fa-lightbulb"></i> GPIO 2</p>
 <p>
 <button class="boton-encendido" id="bENCENDIDO">ENCENDIDO</button>
 <button class="boton-apagado" id="bAPAGADO">APAGADO</button>
 </p>
```

```
 <p class="estado">Estado: <span id="estado">%ESTADO%</span></p>
 </div>
 </div>
 </div>
 <script src="script.js"></script>
 <IMG SRC="wemosD1.png" ALIGN=MIDDLE>
</body>
</html>
```

ARCHIVO style.css

```
html {
 font-family: Arial, Helvetica, sans-serif;
 text-align: center;
}
h1 {
 font-size: 1.8rem;
 color: white;
}
.marquesina {
 overflow: hidden;
 background-color: #1538a8;
}
body {
 margin: 0;
}
.contenido {
 padding: 50px;
}
.cuadricula-tarjeta {
 max-width: 800px;
  margin: 0 auto;
 display: grid;
 grid-gap: 2rem;
 grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
}
.tarjeta {
 background-color: white;
 box-shadow: 2px 2px 12px 1px rgba(140,140,140,.5);
}
.titulo-tarjeta {
 font-size: 1.2rem;
 font-weight: bold;
 color: #034078
}
.estado {
 font-size: 1.2rem;
 color: #1282A2;
}
```

```
button {
 border: none;
 color: #FEFCFB;
 padding: 15px 32px;
 text-align: center;
 text-decoration: none;
 font-size: 16px;
 width: 200px;
 border-radius: 4px;
 transition-duration: 0.4s;
}
.boton-encendido
/* color del botón cuando NO se está apuntando con mouse */
{
 background-color: #034078;
}
.boton-encendido:hover
/* color a que cambia cuando se le apunta con el mouse */
{
 background-color: #1282A2;
}
.boton-apagado
/* color del botón cuando NO se está apuntando con mouse */
{
 background-color: #858585;
}
.boton-apagado:hover
/* color a que cambia cuando se le apunta con el mouse */
{
 background-color: #252524;
}
```

ARCHIVO script.js

```
/* Este es responsable de inicializar una conexión WebSocket con el servidor,
solicita al servidor el valor actual del control deslizante tan pronto como
la interfaz web está completamente cargada en el navegador y maneja el intercambio
de datos a través del protocolo WebSocket. */

// La compuerta es el punto de entrada a la interfaz de WebSocket.
var compuerta = `ws://${window.location.hostname}/ws`;
/* window.location.hostname obtiene la dirección de la página actual
 (la IP del servidor web )  */
var miWebSocket; //  Crear una nueva variable global llamada miWebSocket.
/*  Se agrega un detector de eventos que llamará a la función onload
cuando se cargue la página web. */
window.addEventListener('load', onload); // onload se ejecuta cuando el document HTML carga
function initButton()
```

```
/* La función initButton () obtiene los botones por sus
identificadores ('bENCENDIDO' y 'bAPAGADO') y agrega un detector de eventos
de tipo 'clic'. */
{
 /* Esto significa que cuando hace clic en el botón ON, se llama a la
 función cambiaENCENDIDO
 Cuándo hace clic en el botón OFF, se llama a la función cambiaAPAGADO  */

 document.getElementById('bENCENDIDO').addEventListener('click', cambiaENCENDIDO);
 document.getElementById('bAPAGADO').addEventListener('click', cambiaAPAGADO);
}
function cambiaENCENDIDO(event)
/* La función cambiaENCENDIDO () envía un mensaje usando la conexión WebSocket
con el texto 'bENCENDIDO' */
{
 miWebSocket.send('bENCENDIDO');
}
function cambiaAPAGADO(event)
/* La función cambiaAPAGADO () envía un mensaje usando la conexión WebSocket con
el texto 'bAPAGADO'. */
{
 miWebSocket.send('bAPAGADO');
}
 /* Entonces, el ESP8266 (microcontrolador de la Wemos D1, debería manejar
    lo que sucede cuando recibe estos Mensajes: enciende o apaga el LED y
    notifica a todos los clientes. */
```

ARCHIVO platformio.ini

Después de haber añadido las libraries necesarias al proyecto, el archivo de configuración queda de la siguiente manera:

```
[env:d1]
platform = espressif8266
board = d1
framework = arduino
monitor_speed = 115200
lib_deps =
        ESP Async WebServer
        ottowinter/ESPAsyncTCP-esphome@^1.2.3
        ottowinter/ESPAsyncWebServer-esphome@^1.2.7
board_build.filesystem = littlefs
```

PRUEBA:

Si aparece un mensaje similar al siguiente desconecta y conecta el cable USB y repetir los pasos de construir imagen de archivos de sistema, Actualizar imagen de archivos de sistema y grabar en el hardware Wemos D1.

```
  File "c:\users\lenovo\.platformio\penv\lib\site-packages\serial\__init__.py", line 90, in serial_for_url
    instance.open()
  File "c:\users\lenovo\.platformio\penv\lib\site-packages\serial\serialwin32.py", line 64, in open
    raise SerialException("could not open port {!r}: {!r}".format(self.portstr, ctypes.WinError()))
serial.serialutil.SerialException: could not open port 'COM4': PermissionError(13, 'Acceso denegado.', None, 5)
*/
```

Al teclear la IP que proporciona el monitor de platformio, en el navegador debe aparecer lo siguiente:

Con los botones debe encenderse y apagarse el LED construido en la tarjeta Wemos D1.