ESP32 Websocket

8/26/2023

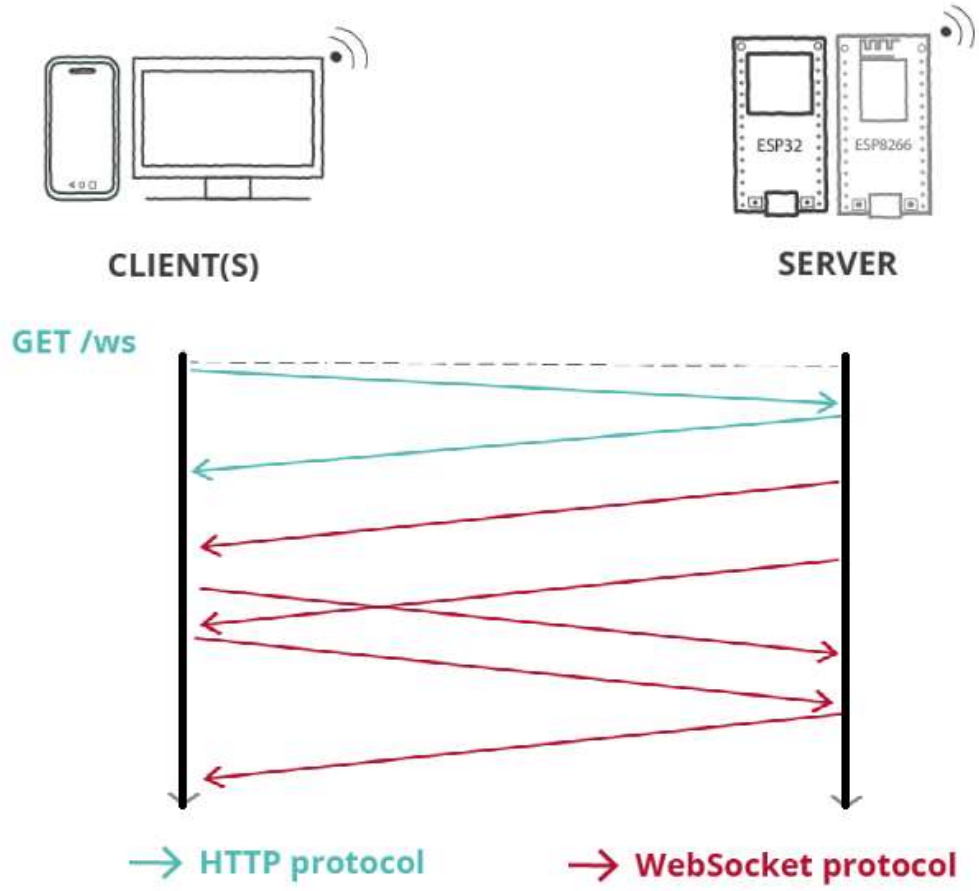
Sangwon Lee

Contents

- ❑ WebSocket
- ❑ Websocket Example
- ❑ Library Requirements
- ❑ Code Analysis
- ❑ Demonstration

- ❑ Reference:
  https://randomnerdtutorials.com/esp32-websocket-server-arduino/

WebSocket



- Websocket is a computer protocol, providing full-duplex channel over a single TCP connection
- To see the updated web page contents, clients should request and read the new web page.
- Sending data from server to client and from client to server is available at any given time.

WebSocket Example 1

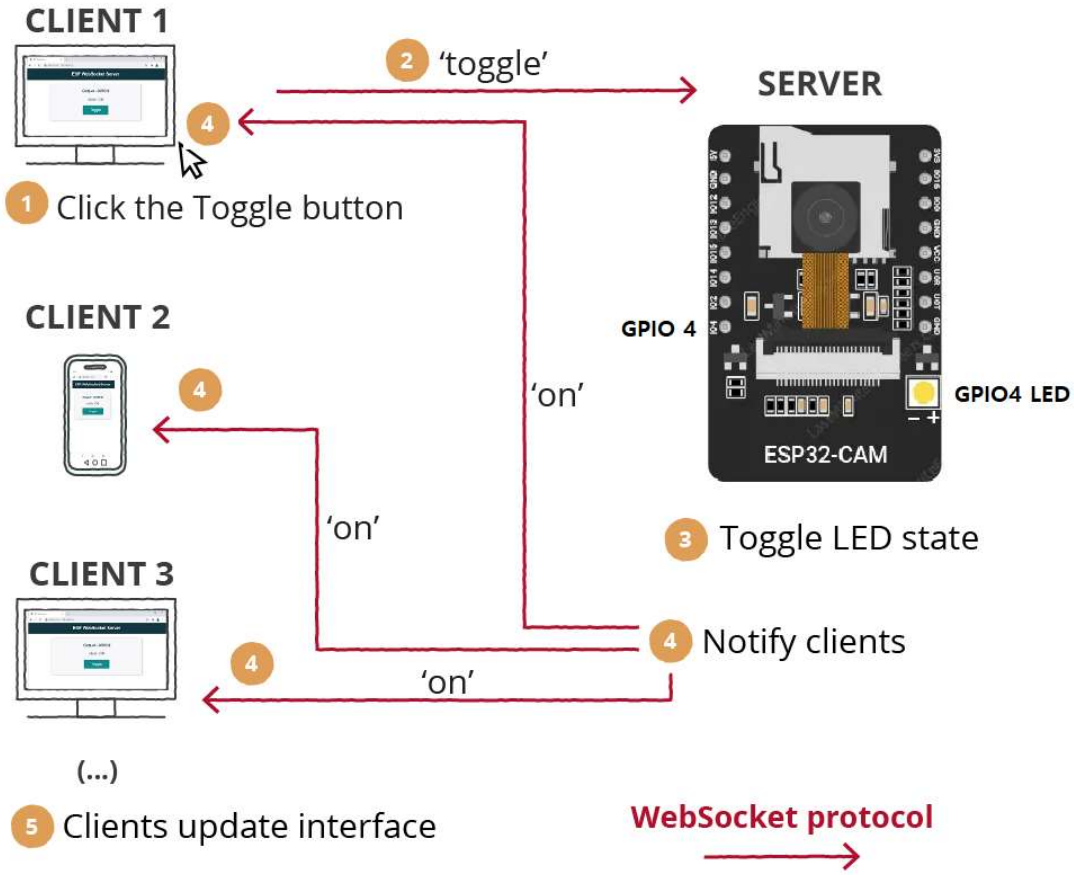
Example Overview

- ESP32-CAM web server for toggling GPIO 4 LED with a web page button.
- GPIO0 pin on ESP32-CAM-MB is for H/W toggling switch.
- GPIO 4 LED on/off status is update automatically in all clients. (maybe for 3 ~ 4 clients)

❑ Reference:
https://randomnerdtutorials.com/esp32-websocket-server-arduino/

WebSocket Example 2

WebSocket Demo Flow



1. Click 'Toggle' Button
2. Client sends data via WebSocket protocol with "toggle" message.
3. ESP32-CAM web server receives this message and toggles LED state.
4. ESP32-CAM web server sends the new LED update state to all clients
5. All clients receive and update the LED state on web page accordingly.

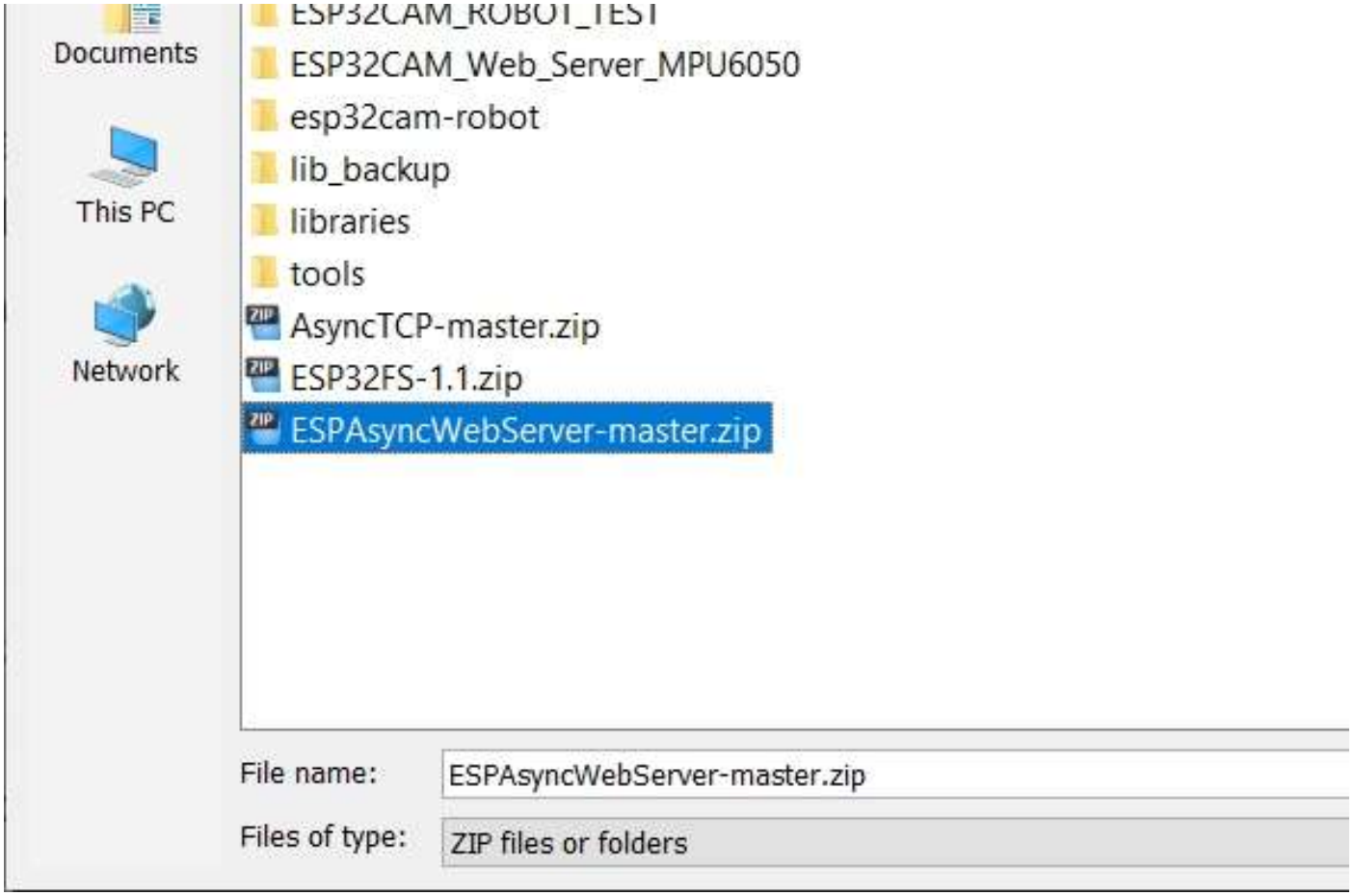
https://randomnerdtutorials.com/esp32-websocket-server-arduino/

Library Requirements

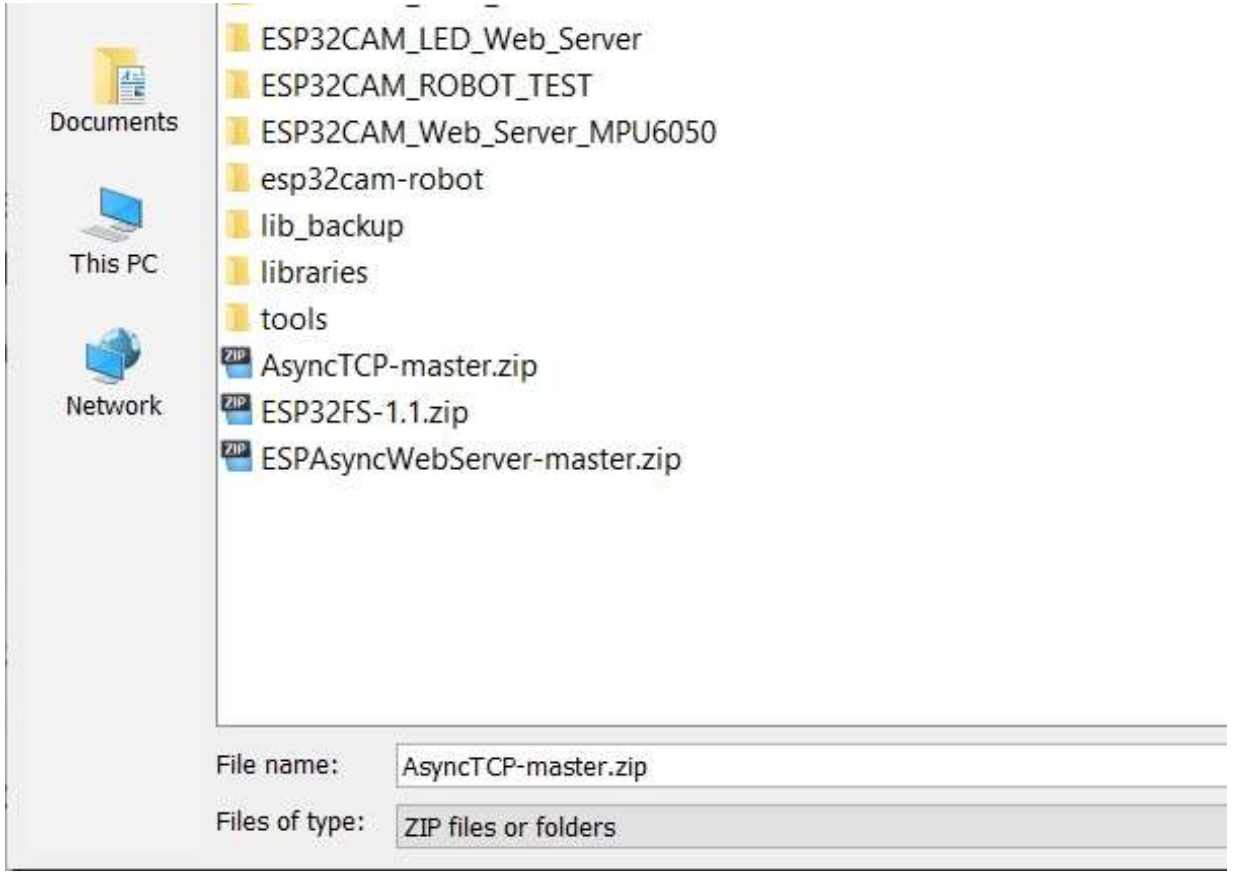
- ESPAsyncWebServer
- AsyncTCP

- Move two .zip files (ESPAsyncWebServer-master.zip , AsyncTCP-master.zip) to ‘Documents>Arduino’ folder
- Go to ‘Sketch > Include Library > Add .zip Library’
- Install two libraries independently.

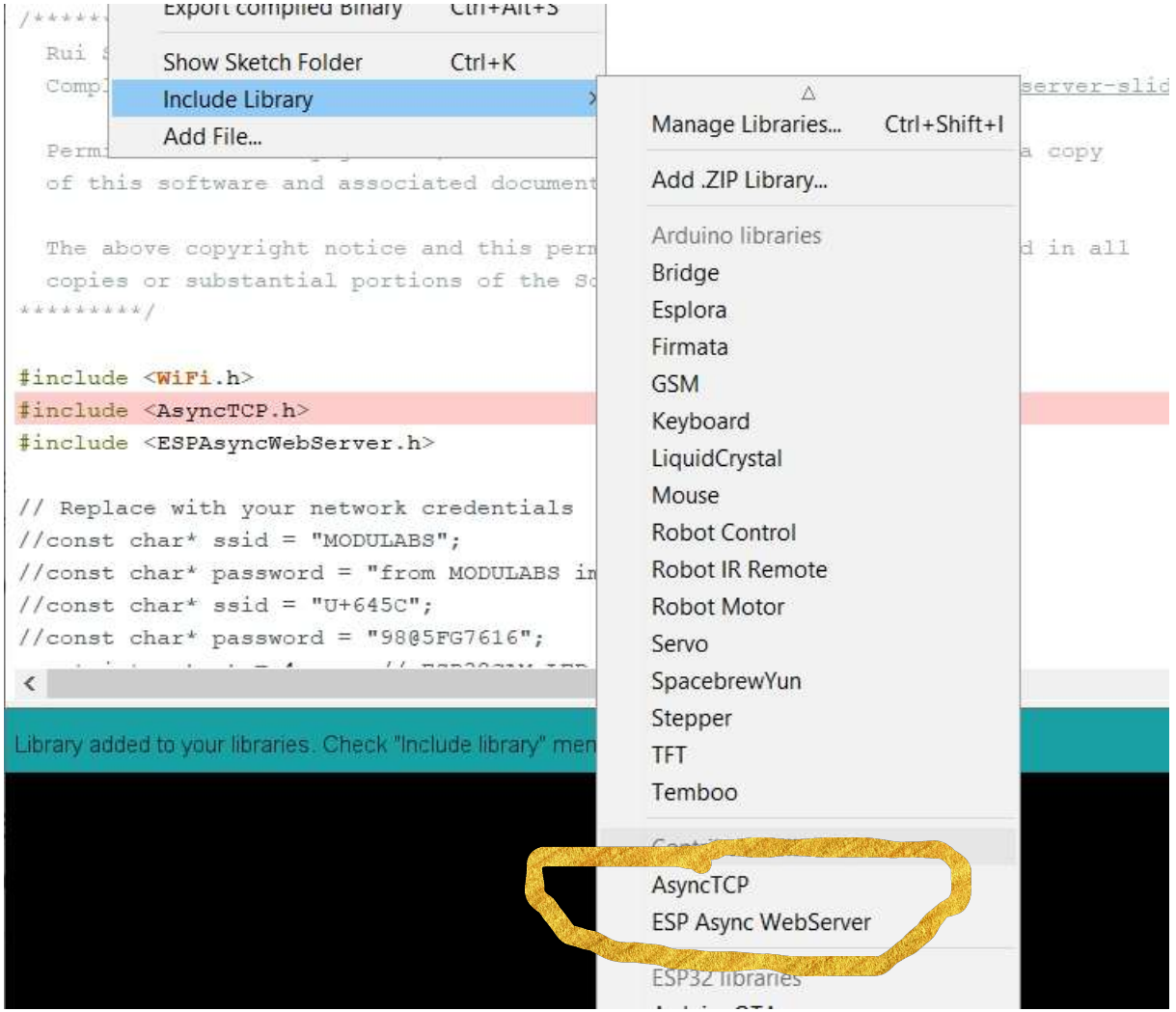
Library Requirements



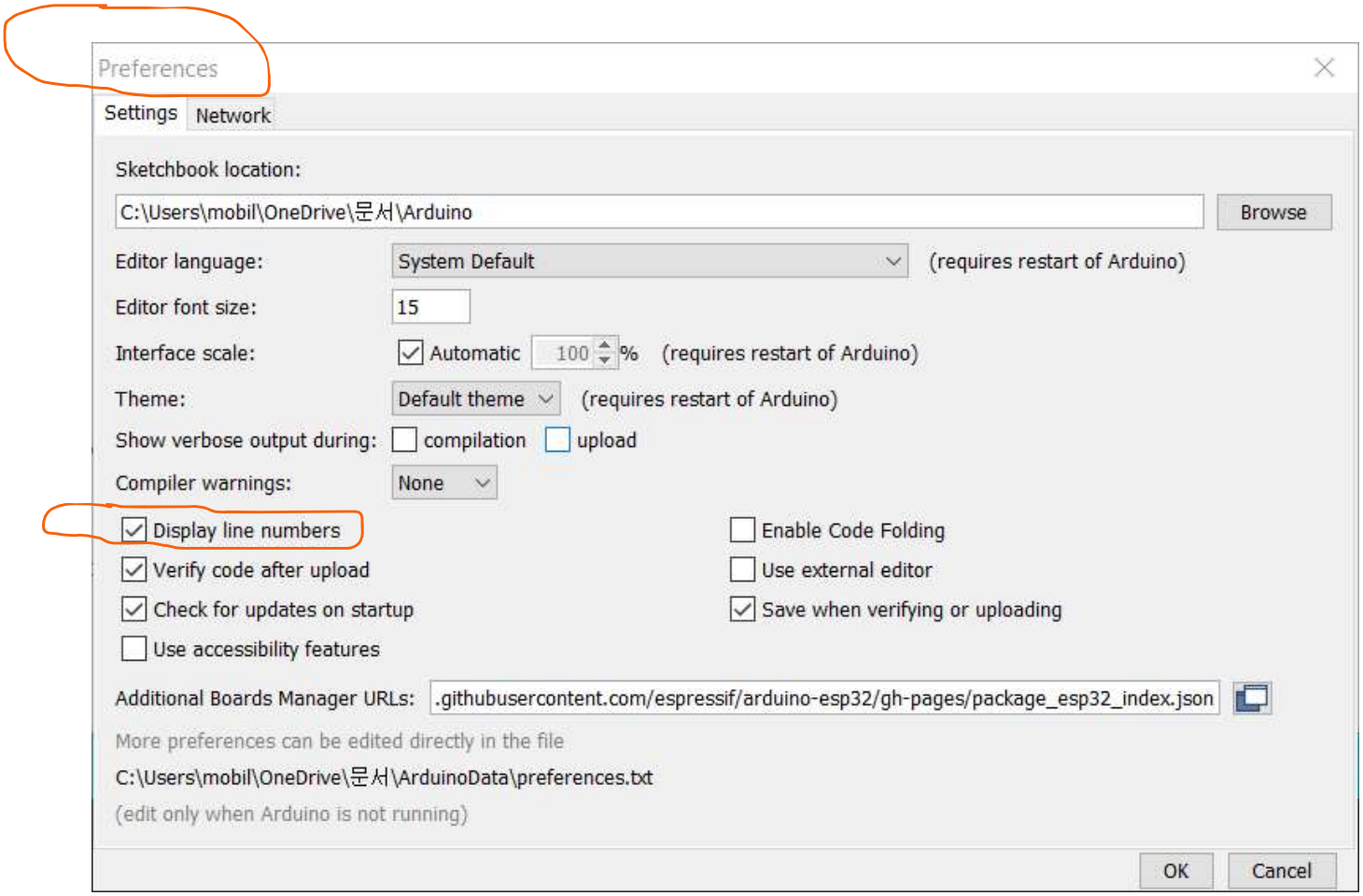
Library Requirements



Library Requirements



Code Analysis – display line numbers



Code Analysis 1

Include Libraries and GPIO Setup

- Line 9 – 11: include Libraries

```
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
```

- Line 17-23: GPIO and PWM setup

```
int freq = 1000;
int ledChannel = 7;  // ledc channel(PWM control unit) number 7
int resolution = 8;  // ledc pwm resolution
int ledPin = 4;

bool ledState = 0;
bool GPIO0_State =0;
```

Code Analysis 2

AsyncWebServer and AsyncWebSocket

- Line 26-27: Create AsyncWebServer and AsyncWebSocket

```
AsyncWebServer server(80);
AsyncWebSocket ws("/ws");      / Create an WebSocket Object – “ws”
```

Code Analysis 3

HTML between <body> and </body>

- Line 101-110: HTML title and span and button

```
 <div class="topnav">
  <h1>ESP WebSocket Server</h1>
</div>
<div class="content">
 <div class="card">
   <h2>Output - GPIO 4</h2>
   <p class="state">state: <span id="state"> %STATE% </span></p>
   <p><button id="button" class="button"> Toggle </button></p>
 </div>
</div>
```

Code Analysis 4

JavaScript – Handling WebSockets

- Line 112-114: Initial setup for WebSocket connection

```
var gateway = `ws://${window.location.hostname}/ws`;
var websocket;
window.addEventListener('load', onLoad);
```

- Line 115-148: functions for WebSocket operation

```
function initWebSocket() { }
function onOpen(event) { }
function onClose(event) { }
function onMessage(event) { }
function onLoad(event) {  }
function initButton() {}
function toggle(){
```

Code Analysis 5

- Line 154-156: notify all clients with a message

```
void notifyClients() {
    ws.textAll(String(ledState));
}
```

- Line 158-167: handling messages from clients via WebSocket protocol

```
void handleWebSocketMessage(void *arg, uint8_t *data, size_t len) {
    AwsFrameInfo *info = (AwsFrameInfo*)arg;
    if (info->final && info->index == 0 && info->len == len && info->opcode == WS_TEXT) {
        data[len] = 0;
        if (strcmp((char*)data, "toggle") == 0) {
            ledState = !ledState;
            notifyClients();
        }
    }
}
```

Code Analysis 6

Configure the WebSocket Server

- Line 169-185: configure the event listener

```
void onEvent(AsyncWebSocket *server, AsyncWebSocketClient *client,
    AwsEventType type, void *arg, uint8_t *data, size_t len) {

        switch (type) {
            case WS_EVT_CONNECT:
            case WS_EVT_DISCONNECT:
            case WS_EVT_DATA:
            case WS_EVT_PONG:
            case WS_EVT_ERROR:
        }
}
```

Code Analysis 7

void setup() {} - line 205 to 239

- Line 213-215: GPIO4 PWM setup

- Line 218: GPIO0 setup as a H/W toggling switch

```
 // configure GPIO4 LED PWM functionalitites
ledcSetup(ledChannel, freq, resolution);
ledcAttachPin(ledPin, ledChannel);
ledcWrite(ledChannel, 0);

// configure GPIO0 for LED toggling
pinMode(0, INPUT);
```

Code Analysis 8

void setup() {} - line 205 to 239

- Line 213-215: GPIO4 PWM setup

- Line 218: GPIO0 setup as a H/W toggling switch

```
 // configure GPIO4 LED PWM functionalitites
ledcSetup(ledChannel, freq, resolution);
ledcAttachPin(ledPin, ledChannel);
ledcWrite(ledChannel, 0);

// configure GPIO0 for LED toggling
pinMode(0, INPUT);
```

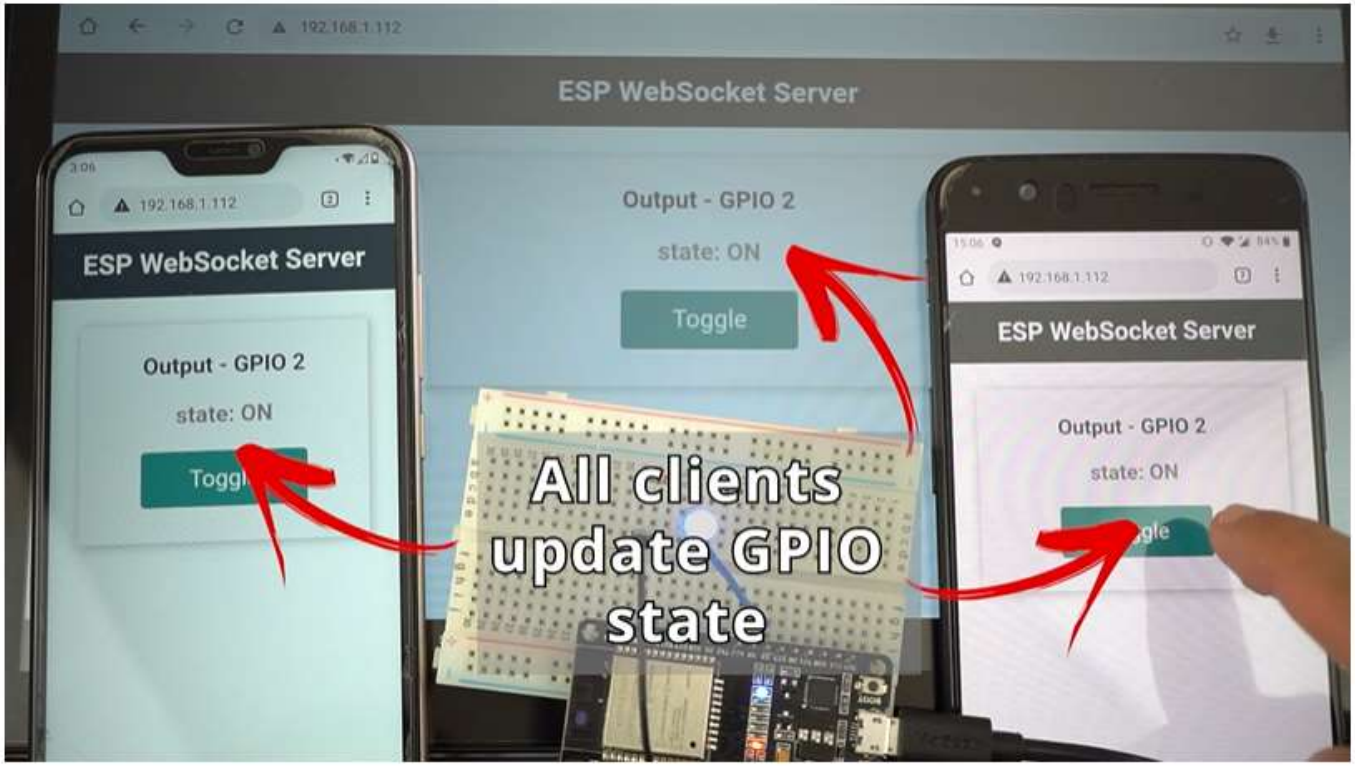
Code Analysis 8

void loop() {} - line 241 to 254

- Line 213-215: GPIO4 PWM setup

```
void loop() {
  ws.cleanupClients();
  ledcWrite(ledChannel, (int)(ledState));
  GPIO0_State = digitalRead(0);
  if(GPIO0_State == 0){          // check the button pressed
    ledState = !ledState;        // toggle 'ledState'
    notifyClients();             // notify all clients of current led state
    delay(300);
  }
}
```

Demonstration



Thank you.