WebSocket with the ESP32: the LED that follows the slider

A slider on the web page sets the LED in real time, and every connected phone sees it move. With ESPAsyncWebServer.

Online guide, always up to date

By **Ada Intelisano** · informaticait.it technical team

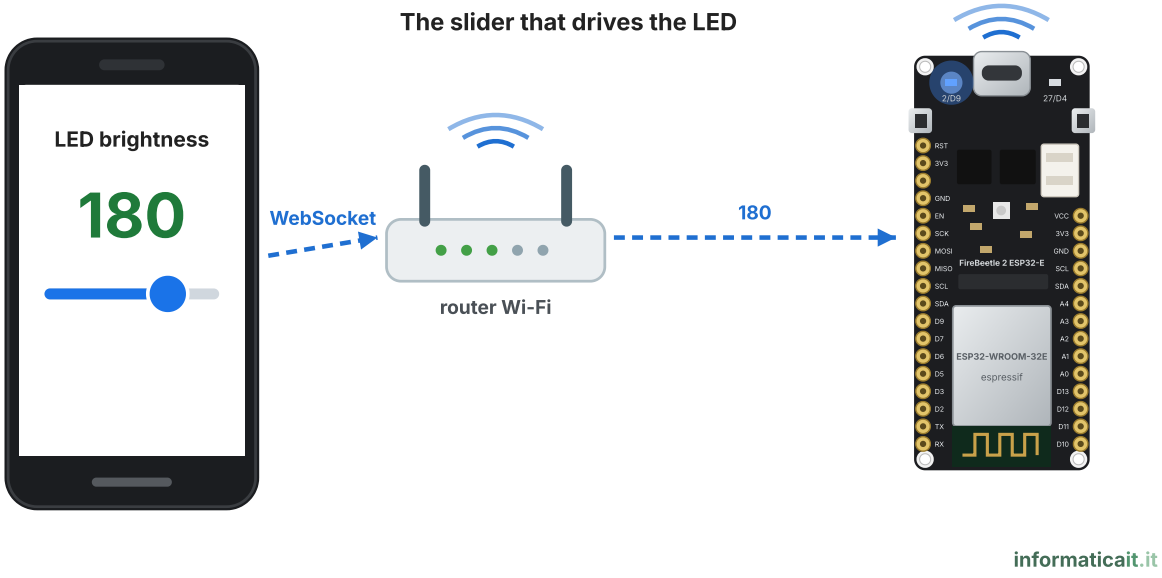
IN SHORT

- The WebSocket stays open: browser and board write to each other whenever they want.
- AsyncTCP and ESPAsyncWebServer libraries from the ESP32Async group.
- `textAll` updates every connected browser.
- `ws.cleanupClients()` in the loop, always.

Time: 45 minutes | Level: next steps | Before: guide no. 56

In the weather station from guide no. 56 the browser asked for the data every 5 seconds. To control something you need more: move the slider on your phone and the LED changes **immediately**; and if another phone is connected, it sees its slider move by itself. That is the job of **WebSockets**.

1 How it works



New word: WebSocket
A channel that stays open between the browser and the board: both can write whenever they want, without a new request each time. The browser sends the slider value; the board sends it back to **everyone** connected.

2 The libraries

You need **AsyncTCP** and **ESPAsyncWebServer**, now maintained by the ESP32Async group: they work with ESP32 packages 2.x and 3.x. In PlatformIO the two `lib_deps` lines are enough; in Arduino IDE install them from the Library Manager, searching for the exact names.

3 The sketch

No wires to connect: we use the LED already on the board (GPIO 2). Write your network name and password. The page is inside the sketch, as in guide no. 56.

```
// A slider on the web page sets the LED brightness. With WebSockets the value arrives at once
// and all the connected phones see the slider move together.
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>

const char* NETWORK = "your-network-name";
const char* PASSWORD = "your-network-password";
const int LED = 2;
int brightness = 0;                  // from 0 to 255

AsyncWebServer server(80);
AsyncWebSocket ws("/ws");

const char PAGE[] PROGMEM = R"rawliteral(
<!DOCTYPE html><html lang="en"><head><meta charset="utf-8"><meta name="viewport"
content="width=device-width, initial-scale=1">
<title>LED</title><style>body{font-family:sans-serif;text-
align:center;margin:24px}input{width:90%}b{font-size:40px;color:#1f7a3a}</style>
</head><body><h1>LED brightness</h1><b id="v">0</b><br><br>
<input type="range" id="c" min="0" max="255" value="0">
<script>
const ws = new WebSocket('ws://' + location.host + '/ws');
const c = document.getElementById('c'), v = document.getElementById('v');
c.oninput = () => ws.send(c.value);
ws.onmessage = e => { c.value = e.data; v.textContent = e.data; };
</script></body></html>
)rawliteral";

void wsEvent(AsyncWebSocket* s, AsyncWebSocketClient* client, AwsEventType type, void* arg,
uint8_t* data, size_t length) {
  if (type == WS_EVT_CONNECT) {
    client->text(String(brightness));              // whoever arrives sees the current value
right away
  } else if (type == WS_EVT_DATA) {
    AwsFrameInfo* info = (AwsFrameInfo*)arg;
    if (info->final && info->index == 0 && info->len == length && info->opcode == WS_TEXT) {
// a complete text message
      String text;
```

```
      for (size_t i = 0; i < length; i++) text += (char)data[i];
      brightness = constrain(text.toInt(), 0, 255);
      analogWrite(LED, brightness);
      ws.textAll(String(brightness));              // tells everyone connected
    }
  }
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(NETWORK, PASSWORD);
  while (WiFi.status() != WL_CONNECTED) delay(500);
  Serial.print("Open http://");
  Serial.println(WiFi.localIP());
  ws.onEvent(wsEvent);
  server.addHandler(&ws);
  server.on("/", HTTP_GET, [](AsyncWebServerRequest* r) { r->send(200, "text/html", PAGE); });
  server.begin();
}

void loop() {
  ws.cleanupClients();     // closes the connections the browser left half-open
  delay(10);
}
```

New word: event
The `wsEvent` function is called by the library when something happens: a browser connects (`WS_EVT_CONNECT`) or a message arrives (`WS_EVT_DATA`). We never call it ourselves: we "register" it with `ws.onEvent(wsEvent)`.

4 Try it

✓ What you should see
Open the address printed in the Serial Monitor from your phone and from your PC at the same time. Move the slider on the phone: the LED changes brightness and the slider on the PC moves by itself.

If it doesn't work

- The page opens but the slider does nothing: the browser cannot open the WebSocket; reload the page.
- Compile errors in AsyncTCP: you installed an old copy of the library; keep only the ESP32Async ones.
- After a while the board freezes: `ws.cleanupClients()` is missing from the loop.

5 Two things to know

- The “asynchronous” server works on its own: the `loop` stays free for other things. That is why you do not put long waits inside `wsEvent`.
- The library accepts only a few browsers at a time (8 at most, by default): enough for a home.
- The WebSocket here is not protected: anyone on your network can open the page and move the slider.

6 Challenge yourself

Try it yourself

- Instead of the LED, the servo from guide no. 41: the slider goes from 0 to 180 degrees.
- Three sliders for the RGB LED from guide no. 46.
- Have the board send the BME280 temperature to everyone every 2 seconds, with `ws.textAll()`.

✓ What you learned

- a WebSocket is a channel open in both directions;
- with `textAll` the board updates every connected browser;
- the asynchronous server leaves the `loop` free.

Sources: ESP32Async: ESPAsyncWebServer (WebSocket documentation and examples) and AsyncTCP · MDN Web Docs, WebSocket API · DFRobot, FireBeetle 2 ESP32-E wiki; DFRobot blog on asynchronous web servers · informaticait guides no. 41, 46 and 56