ESP32 Web Server: Control Stepper Motor (WebSocket)

In this guide you'll learn how to create a web server with the ESP32 that displays a web page to control a stepper motor. The web page allows you to insert the number of steps and select clockwise or counterclockwise direction. Additionally, it also shows whether the motor is currently spinning or if it is stopped. The communication between the client and the server is achieved via WebSocket protocol. All clients are updated with the current motor state.

This tutorial is the second part of this article ***ESP32 Web Server: Control Stepper Motor (HTML Form)***, but it can also be followed as a standalone tutorial.

To better understand how this project works, you can take a look at the following tutorials:

- ESP32 with Stepper Motor (28BYJ-48 and ULN2003 Motor Driver)
- ESP32 Web Server: Control Stepper Motor (HTML Form)
- ESP32 WebSocket Server: Control Outputs (Arduino IDE)

Table of Contents

Prerequisites

Before proceeding with the tutorial, make sure you check the following prerequisites.

1) Parts Required

To follow this tutorial, you need the following parts:

- 28BYJ-48 Stepper Motor + ULN2003 Motor Driver
- ESP32 (read Best ESP32 Development Boards)
- Jumper Wires
- 5V Power Supply

You can use the preceding links or go directly to MakerAdvisor.com/tools to find all the parts for your projects at the best price!

2) Arduino IDE and ESP32 Boards Add-on

We'll program the ESP32 using Arduino IDE. So, you must have the ESP32 add-on installed. Follow the next tutorial if you haven't already:

- Installing ESP32 Board in Arduino IDE (Windows, Mac OS X, Linux)

If you want to use VS Code with the PlatformIO extension, follow the next tutorial instead to learn how to program the ESP32:

- Getting Started with VS Code and PlatformIO IDE for ESP32 and ESP8266 (Windows, Mac OS X, Linux Ubuntu)

3) Filesystem Uploader Plugin

To upload the HTML, CSS, and JavaScript files needed to build this project to the ESP32 filesystem (LittleFS), we'll use a plugin for Arduino IDE: **LittleFS Filesystem uploader**. Follow the next tutorial to install the filesystem uploader plugin if you haven't already:

- Arduino IDE 2: Install ESP32 LittleFS Uploader (Upload Files to the Filesystem)

If you're using VS Code with the PlatformIO extension, read the following tutorial to learn how to upload files to the filesystem:

- ESP32 with VS Code and PlatformIO: Upload Files to LittleFS Filesystem

4) Libraries

To build this project, you need to install the following libraries:

- ESPAsyncWebServer (.zip folder)
- AsyncTCP (.zip folder)

The ESPAsyncWebServer and AsynTCP libraries aren't available to install through the Arduino Library Manager. You need to click on the previous links to download the library files. Then, in your Arduino IDE, go to **Sketch**> **Include Library** > **Add .zip Library** and select the libraries you've just downloaded.

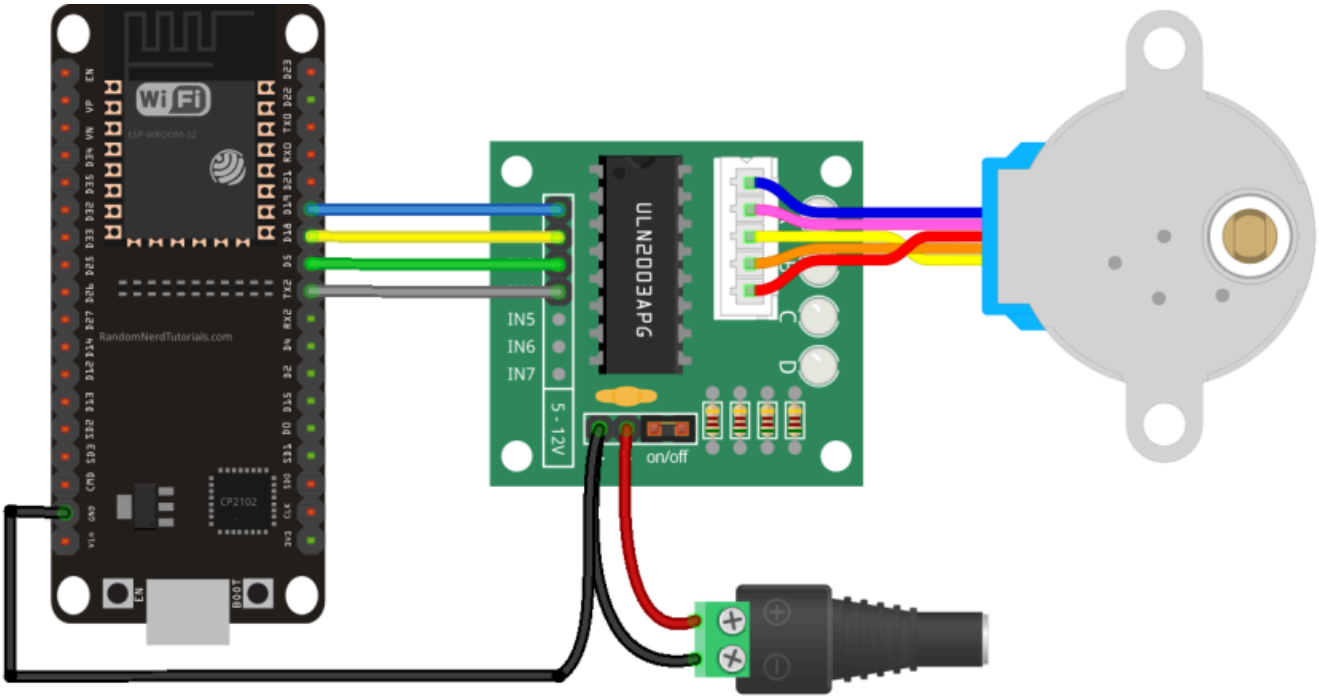
Installing Libraries (VS Code + PlatformIO)

If you're programming the ESP32 using PlatformIO, you should add the following lines to the `platformio.ini` file to include the libraries and set the default filesystem to LittleFS (also change the Serial Monitor speed to 115200):

```
monitor_speed=115200
lib_deps = ESP Async WebServer
    arduino-libraries/Stepper @ ^1.1.3
board_build.filesystem = littlefs
```

5) Schematic Diagram

The following schematic diagram shows the connections between the stepper motor and the ESP32.



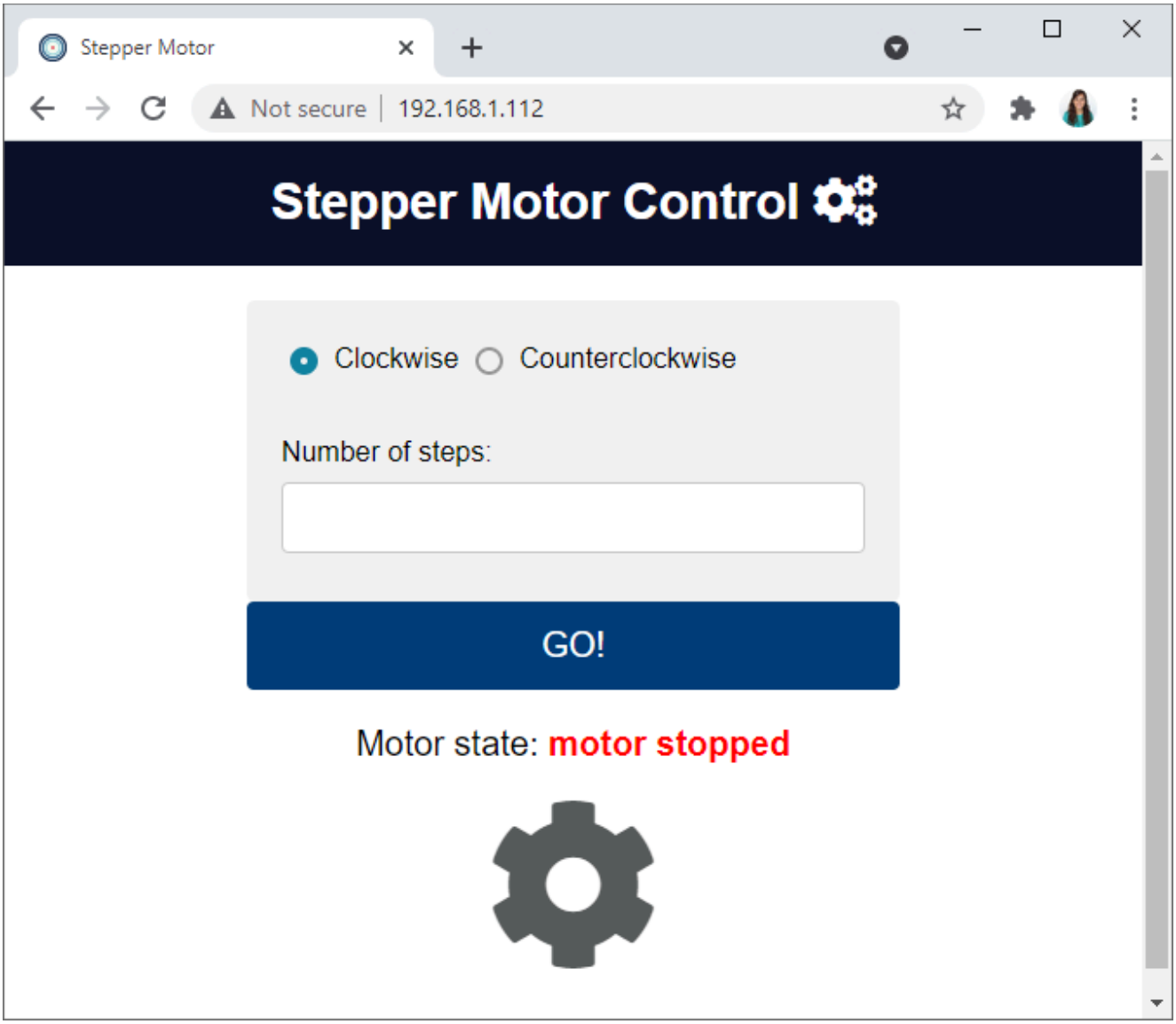
Note: you should power the ULN2003APG motor driver using an external 5V power supply.

Motor Driver	ESP32
IN1	GPIO 19
IN2	GPIO 18
IN3	GPIO 5
IN4	GPIO 17

Project Overview

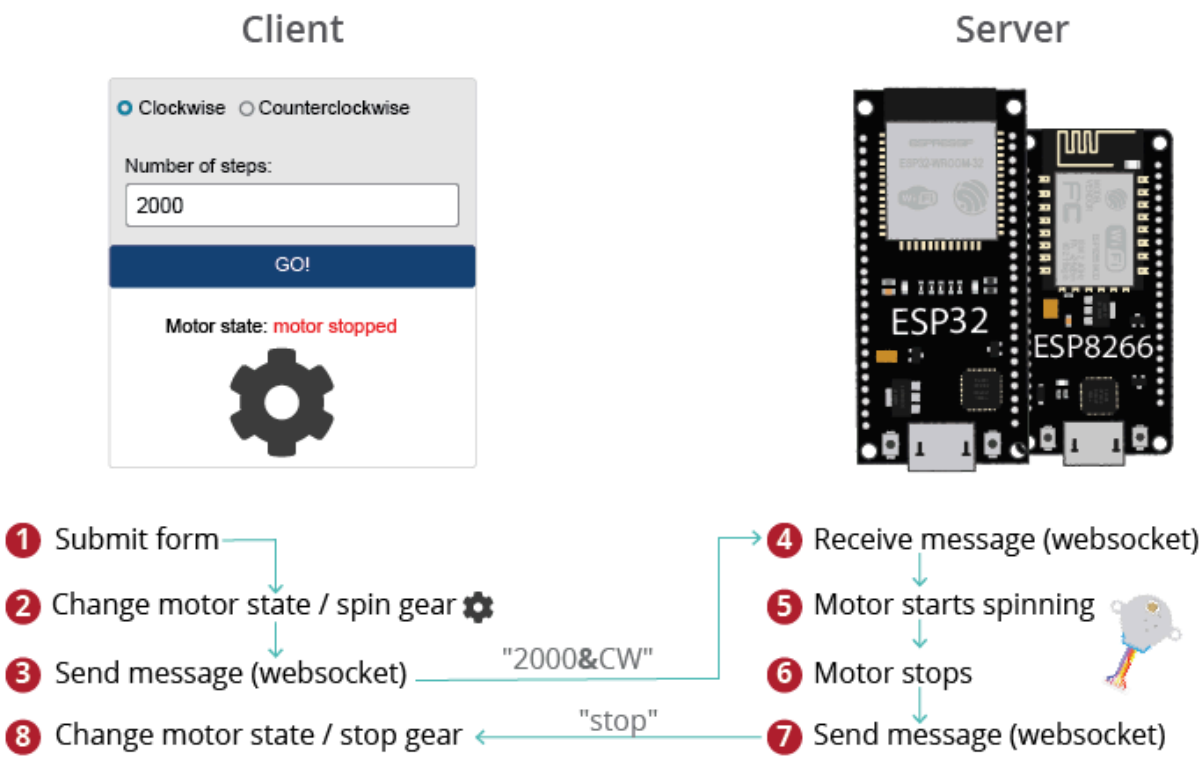
The following video shows a quick demonstration of what you'll achieve by the end of this tutorial.

The following image shows the web page you'll build for this project.



- The web page shows a form where you can enter the number of steps you want the motor to move and select the direction: clockwise or counterclockwise.

- It also shows the motor state: *motor spinning* or *motor stopped*. Additionally, there's a gear icon that spins as long as the motor is spinning. The gear spins clockwise or counterclockwise accordingly to the chosen direction.



- The server and the client communicate using WebSocket protocol.
- When you click on the **GO!** button, it calls a Javascript function that sends a message via WebSocket protocol with all the information: steps and direction (**3**). The message is in the following format:

```
steps&direction
```

So, if you submit 2000 steps and clockwise direction, it will send the following message:

```
2000&CW
```

- At the same time, it will change the motor state on the web page, and the gear will start spinning in the proper direction (**2**).
- Then, the server receives the message (**4**) and spins the motor accordingly (**5**).
- When the motor stops spinning (**6**), the ESP will send a message to the client(s), also via WebSocket protocol, informing that the motor has stopped

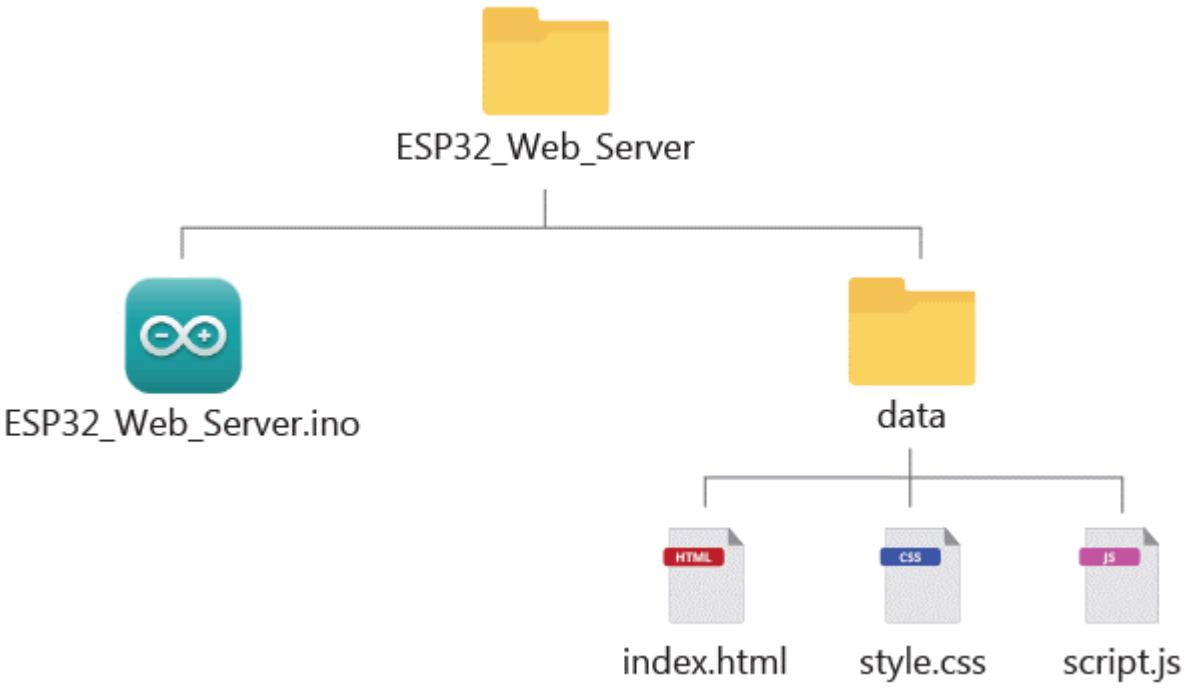
(**7**).

- The client(s) receive this message and update the motor state on the web page (**8**).

Organizing your Files

The files you want to upload to the ESP filesystem should be placed in a folder called `data` under the project folder. We'll move three files to that folder:

- `index.html` to build the web page;
- `style.css` to style the web page;
- `script.js` to handle websocket communication and start/stop the gear animation.



You should save the HTML, CSS, and JavaScript files inside a folder called `data` inside the Arduino sketch folder, as shown in the previous diagram. We'll upload those files to the ESP32 filesystem (LittleFS).

You can download all project files:

- **Download All the Arduino Project Files**

HTML File

Create a file called `index.html` with the following content:

```
<!DOCTYPE html>
<html>
<head>
  <title>Stepper Motor</title>
  <meta name="viewport" content="width=device-width, initial-sc
  <link rel="stylesheet" type="text/css" href="style.css">
  <link rel="icon" type="image/png" href="favicon.png">
  <link rel="stylesheet" href="https://use.fontawesome.com/rele
</head>
<body>
  <div class="topnav">
    <h1>Stepper Motor Control <i class="fas fa-cogs"></i></h1>
  </div>
  <div class="content">
        <form>
          <input type="radio" id="CW" name="direction" value="C
          <label for="CW">Clockwise</label>
          <input type="radio" id="CCW" name="direction" value="
          <label for="CW">Counterclockwise</label><br><br><br>
          <label for="steps">Number of steps:</label>
          <input type="number" id="steps" name="steps">
        </form>
        <button onclick="submitForm()">GO!</button>
        <p>Motor state: <span id="motor-state">Stopped</span></
        <p><i id="gear" class="fas fa-cog"></i> </p>


  </div>
```

View raw code

This HTML file is very similar to the one used in the **previous tutorial**. You can *click here* for a complete explanation of the HTML file.

We've added ids to the HTML elements we want to manipulate using JavaScript—the radio buttons and the input field:

- clockwise radio button: `id="CW"`
- counterclowise radio button: `id="CCW"`
- steps input field: `id="steps"`

```
<input type="radio" id="CW" name="direction" value="CW" checked>
<label for="CW">Clockwise</label>
<input type="radio" id="CCW" name="direction" value="CCW">
<label for="CW">Counterclockwise</label><br><br><br>
<label for="steps">Number of steps:</label>
<input type="number" id="steps" name="steps">
```

We want to send the form results to the server (ESP32) via WebSocket protocol. So, we've added a button, that when clicked (`onclick` event) calls the `submitForm()` user-defined javascript function that sends the results to the server as you'll see later in the JavaScript section.

```
<button onclick="submitForm()">GO!</button>
```

Additionally, we also added a paragraph to display the motor state. We've added a `<span>` tag with the `motor-state` id so that we're able to manipulate the text between the `<span>` tags using Javascript.

```
<p>Motor state: <span id="motor-state">Stopped</span></p>
```

Finally, there's a paragraph displaying a gear with the `id="gear"` . We need this id to make the gear move.

```
<p><i id="gear" class="fas fa-cog"></i> </p>
```

Don't forget that you need to reference the JavaScript file (`scrip.js`) in the HTML file as follows:

```
<script src="script.js"></script>
```

CSS File

Create a file called `style.css` with the following content:

```
html {
  font-family: Arial, Helvetica, sans-serif;
}

h1 {
  font-size: 1.8rem;
  color: white;
}

p{
  font-size: 20px;
  text-align: center;
}

.topnav {
  overflow: hidden;
  background-color: #0A1128;
  text-align: center;
}

body {
  margin: 0;
}

.content {
  padding: 20px;
  max-width: max-content;
```

View raw code

We already covered how the CSS for the HTML form works. You can *click here* for a detailed explanation. Let's take a look at the relevant parts for this tutorial.

We format the motor state text font-weight (`bold`) and color (`red`). To refer to a specific id in CSS, use `#` followed by the id (`#motor-state`).

```
#motor-state{
  font-weight: bold;
  color: red;
}
```

The following line formats the gear icon color and size—remember that its id is `gear` , so we refer to it with `#gear` :

```
#gear{
  font-size:100px;
  color:#2d3031cb;
}
```

Then, we format two classes `spin` and `spin-back` that are not attributed to any HTML element yet. We'll attribute the `spin` and `spin-back` classes to the gear using JavaScript when the motor starts moving.

These classes use the `animation` property to rotate the gear. To learn more about how the `animation` property works, we recommend taking a look at this quick tutorial.

```
.spin {
  -webkit-animation:spin 4s linear infinite;
  -moz-animation:spin 4s linear infinite;
  animation:spin 4s linear infinite;
}

.spin-back {
  -webkit-animation:spin-back 4s linear infinite;
```

```
  -moz-animation:spin-back 4s linear infinite;
  animation:spin-back 4s linear infinite;
}

@-moz-keyframes spin { 100% { -moz-transform: rotate(360deg); }
@-webkit-keyframes spin { 100% { -webkit-transform: rotate(360de
@keyframes spin { 100% { -webkit-transform: rotate(360deg); tran

@-moz-keyframes spin-back { 100% { -moz-transform: rotate(-360de
@-webkit-keyframes spin-back { 100% { -webkit-transform: rotate(
@keyframes spin-back { 100% { -webkit-transform: rotate(-360deg)
```

JavaScript File

Create a file called `script.js` with the following content:

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

function onload(event) {
    initWebSocket();
}

function initWebSocket() {
    console.log('Trying to open a WebSocket connection…');
    websocket = new WebSocket(gateway);
    websocket.onopen = onOpen;
    websocket.onclose = onClose;
    websocket.onmessage = onMessage;
}

function onOpen(event) {
    console.log('Connection opened');
}
```

```
function onClose(event) {
    console.log('Connection closed');
    setTimeout(initWebSocket, 2000);
}

function submitForm(){
```

View raw code

Let's see how the JavaScript for this project works.

The `gateway` is the entry point to the WebSocket interface. `window.location.hostname` gets the current page address (the web server IP address)

```
var gateway = `ws://${window.location.hostname}/ws`;
```

Create a new global variable called `websocket`.

```
var websocket;
```

Create another global variable called `direction` that will hold the motor's current direction: clockwise, counterclowise or stopped.

```
var direction;
```

Add an event listener that will call the `onload` function when the web page loads.

```
window.addEventListener('load',  onload);
```

The `onload()` function calls the `initWebSocket()` function to initialize a WebSocket connection with the server.

```
function onload(event) {
  initWebSocket();
}
```

The `initWebSocket()` function initializes a WebSocket connection on the gateway defined earlier. We also assign several callback functions that will be triggered when the WebSocket connection is opened, closed or when a message is received.

```
function initWebSocket() {
  console.log('Trying to open a WebSocket connection…');
  websocket = new WebSocket(gateway);
  websocket.onopen = onOpen;
  websocket.onclose = onClose;
  websocket.onmessage = onMessage;
}
```

When the connection is opened, print a message in the console for debugging purposes.

```
function onOpen(event) {
  console.log('Connection opened');
}
```

If for some reason the web socket connection is closed, call the `initWebSocket()` function again after 2000 milliseconds (2 seconds).

```
function onClose(event) {
  console.log('Connection closed');
  setTimeout(initWebSocket, 2000);
}
```

Finally, we need to handle what happens when the form is submitted and when the client receives a new message (`onMessage` event).

When the form is submitted, the `submitForm()` function is called:

```
function submitForm(){
```

We start by getting which radio button is selected. We save the value of the selected radio button in the `direction` variable.

```
const rbs = document.querySelectorAll('input[name="direction"]')
var direction;
for (const rb of rbs) {
  if (rb.checked) {
    direction = rb.value;
    break;
  }
}
```

Then, we change the motor state text to `motor spinning…` and its color to `blue` . We refer to that HTML element by its id `motor-state` .

```
document.getElementById("motor-state").innerHTML = "motor spinni
document.getElementById("motor-state").style.color = "blue";
```

Then, we check whether we've selected clockwise or counterclockwise direction to spin the gear in the right direction. To do that, we add the class `spin` or `spin-back` to the element with the `gear` id.

```
if (direction=="CW"){
  document.getElementById("gear").classList.add("spin");
}
else{
```

```
    document.getElementById("gear").classList.add("spin-back");
  }
```

We get the number of steps inserted and save it in the `steps` variable.

```
  var steps = document.getElementById("steps").value;
```

Then, we finally send a message via WebSocket protocol to the server (ESP32) with the number of steps and direction separated by a `&`.

```
  websocket.send(steps+"&"+direction);
```

The server (your ESP board) will send a message when it is time to change the motor state. When that happens, we save the message in the `direction` variable.

We check the content of the message and change the motor state and gear animation accordingly.

```
  function onMessage(event) {
    console.log(event.data);
    direction = event.data;
    if (direction=="stop"){
      document.getElementById("motor-state").innerHTML = "motor st
      document.getElementById("motor-state").style.color = "red";
      document.getElementById("gear").classList.remove("spin", "sp
    }
    else if(direction=="CW" || direction=="CCW"){
      document.getElementById("motor-state").innerHTML = "motor sp
      document.getElementById("motor-state").style.color = "blue";
      if (direction=="CW"){
        document.getElementById("gear").classList.add("spin");
      }
      else{
        document.getElementById("gear").classList.add("spin-back")
```

```
    }
  }
}
```

Arduino Sketch

Before uploading, you can use the following link to:

- **Download All the Arduino Project Files**

Copy the following code to the Arduino IDE. Insert your network credentials and it will work straight away.

```
/*
  Rui Santos & Sara Santos - Random Nerd Tutorials
  Complete project details at https://RandomNerdTutorials.com/s
  Permission is hereby granted, free of charge, to any person c
  The above copyright notice and this permission notice shall b
*/
#include <Arduino.h>
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include "LittleFS.h"
#include <Stepper.h>

const int stepsPerRevolution = 2048;  // change this to fit the
#define IN1 19
#define IN2 18
#define IN3 5
#define IN4 17
Stepper myStepper(stepsPerRevolution, IN1, IN3, IN2, IN4);

String message = "";

// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
```

```
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

View raw code

The Arduino sketch is very similar to the previous tutorial, but it handles the client-server communication using WebSocket protocol. Let's see how it works or skip to the demonstration section.

Include Libraries

First, include the required libraries. The `WiFi`, `AsyncTCP`, and `ESPAsyncWebServer` to create the web server, the LittleFS library to use the ESP32 filesystem, and the `Stepper` library to control the stepper motor.

```
#include <Arduino.h>
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include "LittleFS.h"
#include <Stepper.h>
```

Stepper Motor Pins and Steps per Revolution

Define the steps per revolution of your stepper motor—in our case, it's 2048:

```
const int stepsPerRevolution = 2048;  // change this to fit the
```

Define the motor input pins. In this example, we're connecting to GPIOs 19, 18, 5, and 17, but you can use any other suitable GPIOs.

```
#define IN1 19
#define IN2 18
#define IN3 5
#define IN4 17
```

Initialize an instance of the stepper library called `myStepper`. Pass as arguments the steps per revolution and the input pins. In the case of the 28BYJ-48 stepper motor, the order of the pins is `IN1`, `IN3`, `IN2`, `IN4`—it might be different for your motor.

```
Stepper myStepper(stepsPerRevolution, IN1, IN3, IN2, IN4);
```

Network Credentials

Insert your network credentials in the following lines.

```
// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

AsyncWebServer and AsyncWebSocket

Create an `AsyncWebServer` object called `server` on port 80.

```
AsyncWebServer server(80);
```

The `ESPAsyncWebServer` library includes a WebSocket plugin that makes it easy to handle WebSocket connections. Create an `AsyncWebSocket` object called `ws` to handle the connections on the `/ws` path.

```
AsyncWebSocket ws("/ws");
```

Initializing Variables

The following variables will save the direction and number of steps received via WebSocket protocol. When the program first starts, the motor is stopped.

```
String direction ="stop";
```

```
String steps;
```

The `newRequest` variable will be used to check whether a new request occurred. Then, in the `loop()`, we'll spin the motor when a new request is received—when the `newRequest` variable is `true`.

```
bool newRequest = false;
```

initLittleFS()

The `initLittleFS()` function initializes the ESP32 Filesystem.

```
void initLittleFS() {
  if (!LittleFS.begin(true)) {
    Serial.println("An error has occurred while mounting LittleF
  }
  else{
    Serial.println("LittleFS mounted successfully");
  }
}
```

initWiFi()

The `initWiFi()` function initializes WiFi.

```
// Initialize WiFi
void initWiFi() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi ..");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print('.');
    delay(1000);
  }
```

```
  Serial.println(WiFi.localIP());
}
```

Handling WebSockets – Server

Previously, you've seen how to handle the WebSocket connection on the client side (browser). Now, let's take a look on how to handle it on the server side.

Notify All Clients

The `notifyClients()` function notifies all clients with a message containing whatever you pass as a argument. In this case, we'll want to notify all clients of the current motor state whenever there's a change.

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

The `AsyncWebSocket` class provides a `textAll()` method for sending the same message to all clients that are connected to the server at the same time.

Handle WebSocket Messages

The `handleWebSocketMessage()` function is a callback function that will run whenever we receive new data from the 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 && inf
    data[len] = 0;
    message = (char*)data;
    steps = message.substring(0, message.indexOf("&"));
    direction = message.substring(message.indexOf("&")+1, messag
    Serial.print("steps");
    Serial.println(steps);
    Serial.print("direction");
    Serial.println(direction);
```

```
      notifyClients(direction);
      newRequest = true;
    }
  }
```

We split the message to get the number of steps and direction.

```
message = (char*)data;
steps = message.substring(0, message.indexOf("&"));
direction = message.substring(message.indexOf("&")+1, message.le
```

Then, we notify all clients of the motor direction so that all clients change the motor state on the web interface.

```
notifyClients(direction);
```

Finally, set the `newRequest` variable to `true`, so that the motors starts spinning in the `loop()`.

```
newRequest = true;
```

Configure the WebSocket server

Now we need to configure an event listener to handle the different asynchronous steps of the WebSocket protocol. This event handler can be implemented by defining the `onEvent()` as follows:

```
void onEvent(AsyncWebSocket *server, AsyncWebSocketClient *clien
  switch (type) {
    case WS_EVT_CONNECT:
      Serial.printf("WebSocket client #%u connected from %s\n",
      //Notify client of motor current state when it first conne
```

```
        notifyClients(direction);
        break;
      case WS_EVT_DISCONNECT:
        Serial.printf("WebSocket client #%u disconnected\n", clien
        break;
      case WS_EVT_DATA:
          handleWebSocketMessage(arg, data, len);
          break;
      case WS_EVT_PONG:
      case WS_EVT_ERROR:
       break;
    }
}
```

The `type` argument represents the event that occurs. It can take the following values:

- `WS_EVT_CONNECT` when a client has logged in;
- `WS_EVT_DISCONNECT` when a client has logged out;
- `WS_EVT_DATA` when a data packet is received from the client;
- `WS_EVT_PONG` in response to a ping request;
- `WS_EVT_ERROR` when an error is received from the client.

There's a section to notify any client of the current motor state when it first connects:

```
case WS_EVT_CONNECT:
  Serial.printf("WebSocket client #%u connected from %s\n", clie
  //Notify client of motor current state when it first connects
  notifyClients(direction);
  break;
```

Initialize WebSocket

Finally, the `initWebSocket()` function initializes the WebSocket protocol.

```
void initWebSocket() {
  ws.onEvent(onEvent);
  server.addHandler(&ws);
}
```

setup()

In the `setup()`, initialize the Serial Monitor.

```
Serial.begin(115200);
```

Call the `initWiFi()` function to initialize WiFi.

```
initWiFi();
```

Call the `initLittleFS()` function to initialize the filesystem.

```
initWebSocket();
```

And set the stepper motor speed in rpm.

```
myStepper.setSpeed(5);
```

Handle requests

Then, handle the web server. When you receive a request on the root (`/`) URL— this is when you access the ESP IP address— send the HTML text to build the web page:

```
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send(200, "text/html", index_html);
```

```
});
```

When the HTML file loads on your browser, it will make a request for the CSS and JavaScript files. These are static files saved on the same directory (LittleFS). So, we can simply add the following line to serve files in a directory when requested by the root URL. It will serve the CSS and JavaScript files automatically.

```
server.serveStatic("/", LittleFS, "/");
```

Finally, start the server.

```
server.begin();
```

loop()

Let's take a look at the `loop()` section.

If the `newRequest` variable is `true`, we'll check what's the spinning direction: `CW` or `CCW`. If it is `CW`, we move the motor the number of steps saved in the `steps` variable using the `step()` method on the `myStepper` object. To move the motor counterclockwise, we just need to pass the number of steps but with a minus (`-`) sign.

```
if (direction == "CW"){
  // Spin the stepper clockwise direction
  myStepper.step(steps.toInt());
}
else{
  // Spin the stepper counterclockwise direction
  myStepper.step(-steps.toInt());
}
```

After spinning the motor, set the `newRequest` variable to `false`, so that it can detect new requests again.

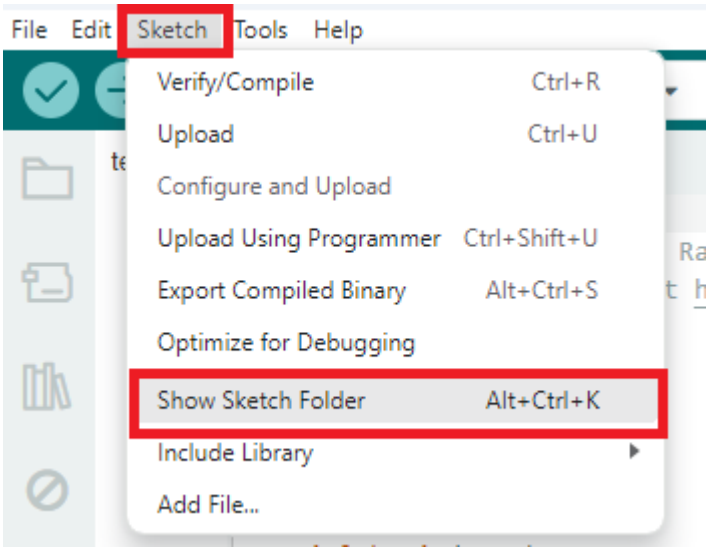
```
  newRequest = false;
```

Additionally, notify all clients that the motor has stopped.

```
  notifyClients("stop");
```

Upload Code and Files

After inserting your network credentials, save the code. Go to **Sketch** > **Show Sketch Folder**, and create a folder called **data**.



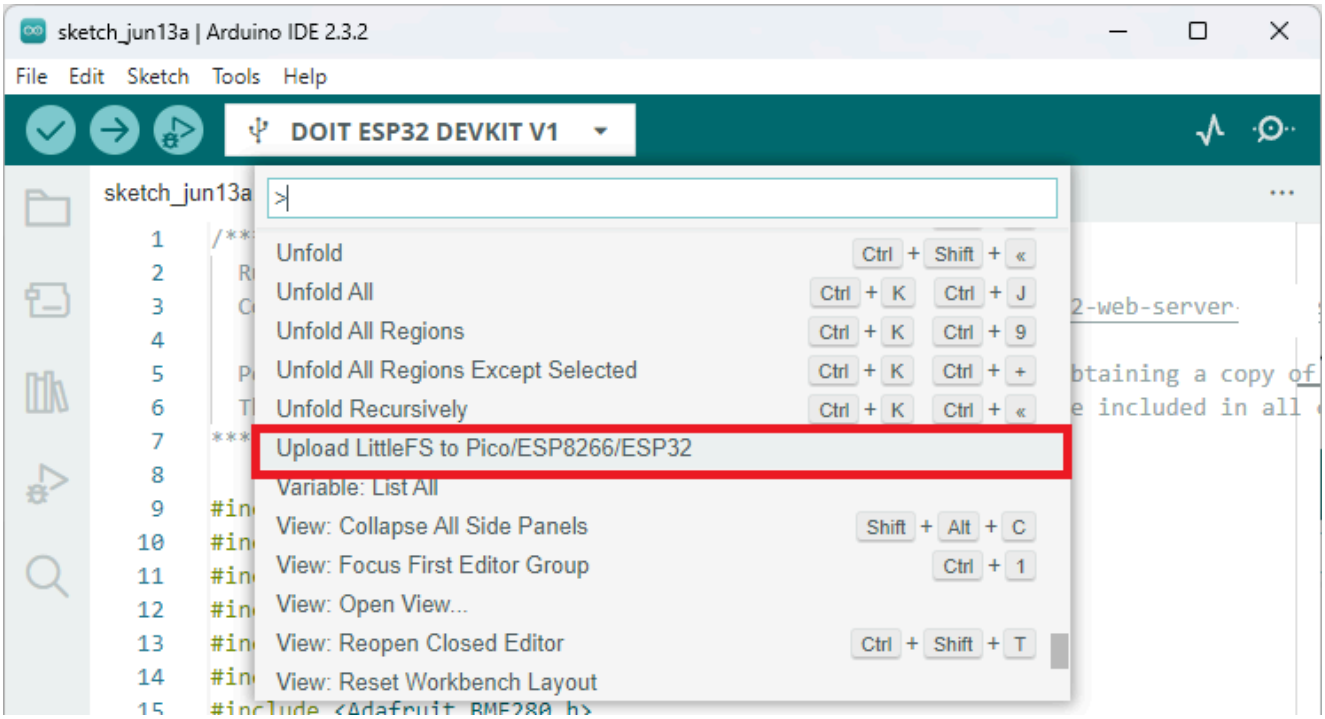
Inside that folder, you should save the HTML, CSS, and JavaScript files.

Then, upload the code to your ESP32 board. Make sure you have the right board and COM port selected. Also, make sure you've added your network credentials.

After uploading the code, you need to upload the files.

Press [**Ctrl**] + [**Shift**] + [**P**] on Windows or [⌘] + [**Shift**] + [**P**] on MacOS to open the command palette. Search for the **Upload LittleFS to Pico/ESP8266/ESP32** command and click on it.

If you don’t have this option is because you didn’t install the filesystem uploader plugin. Check this tutorial.

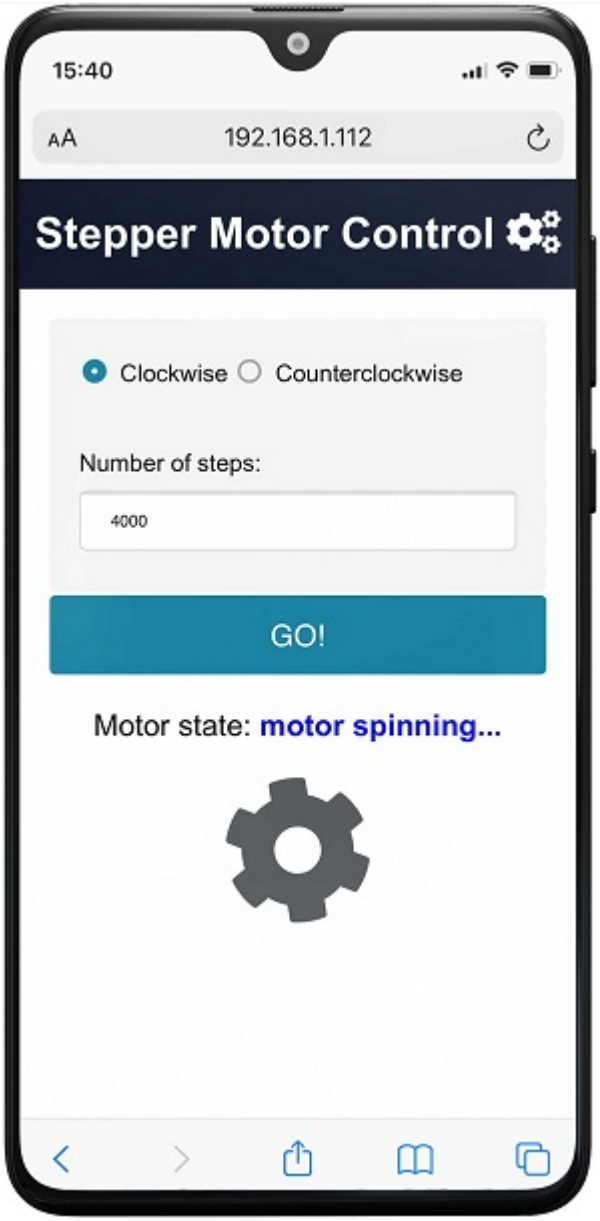


> **Important:** make sure the Serial Monitor is closed before uploading to the filesystem. Otherwise, the upload will fail.

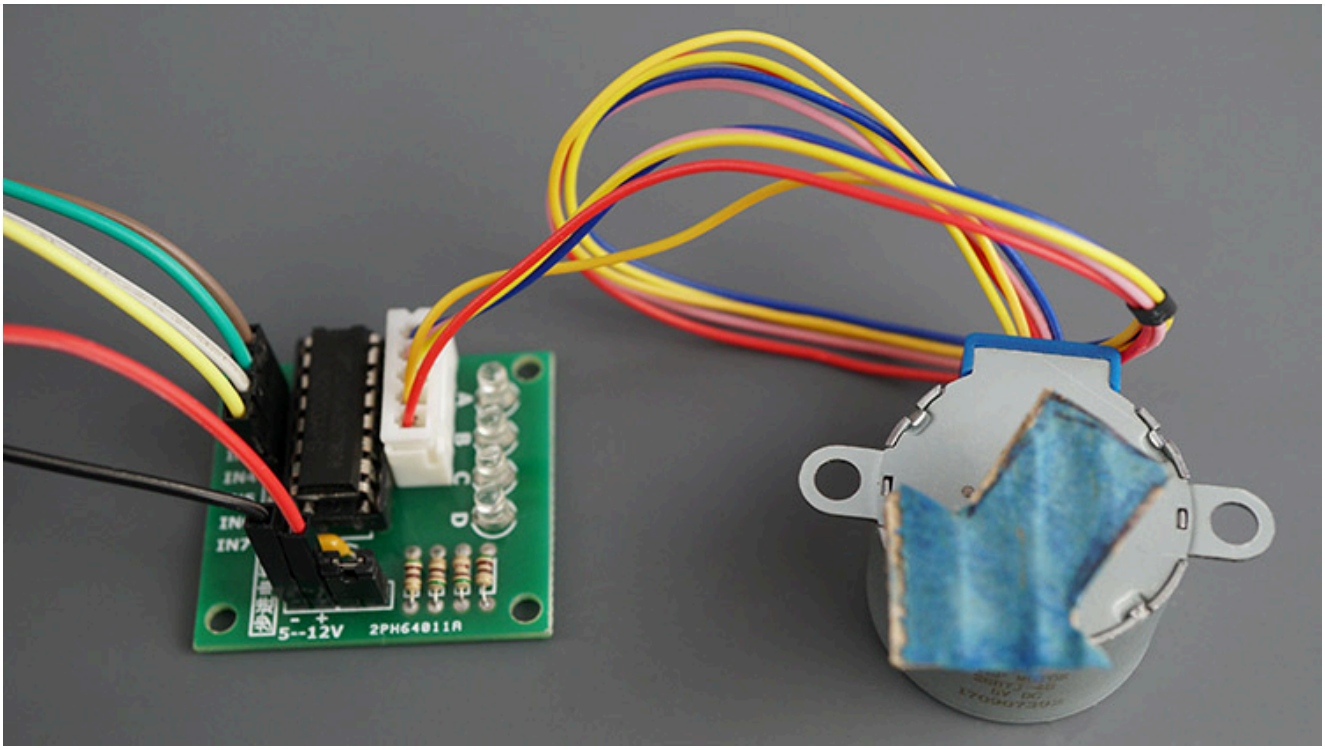
When everything is successfully uploaded, open the Serial Monitor at a baud rate of 115200. Press the ESP32 EN/RST button, and it should print the ESP32 IP address.

Demonstration

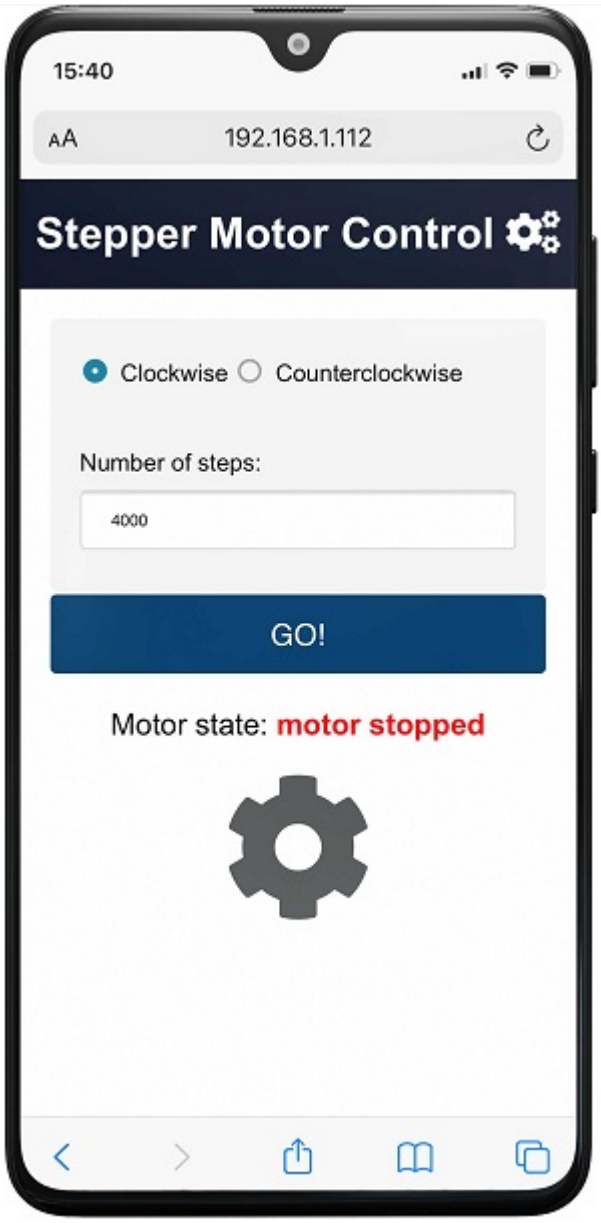
Open a web browser or multiple web browser windows on your local network and you’ll access the web page to control the motor. Submit the form to control the motor.



The gear on the web page starts spinning in the right direction and the motor starts working.



When it stops, the gear on the web page and the motor state change accordingly.



Notice that if you have multiple clients connected, all clients update the motor state almost instantaneously.

Watch the video below for a live demonstration.

Wrapping Up

In this tutorial, you've learned how to control a stepper motor using a web server built with the ESP32. The web server provides a web page to control the stepper motor using a form whose results are sent to the ESP32 via WebSocket protocol.

This is part 3 of a series of tutorials about controlling a stepper motor using a web server. You can follow Part 1 and 2 at the following link:

- Control Stepper Motor with ESP32 Web Server (HTML Form)

If you want to learn more about HTML, CSS, JavaScript, and client-server communication protocols to build your ESP32 and ESP8266 web servers from scratch as we've done in this tutorial, make sure you take a look at our eBook:

- Build Web Servers with ESP32 and ESP8266

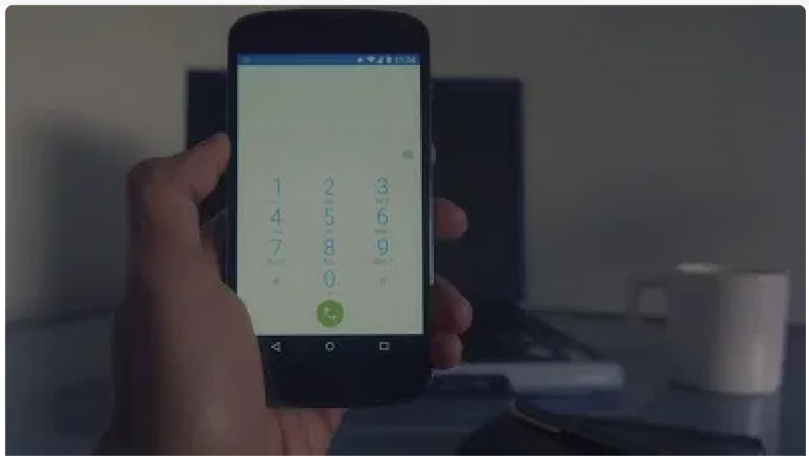
We hope you find this tutorial useful.

Thanks for reading.

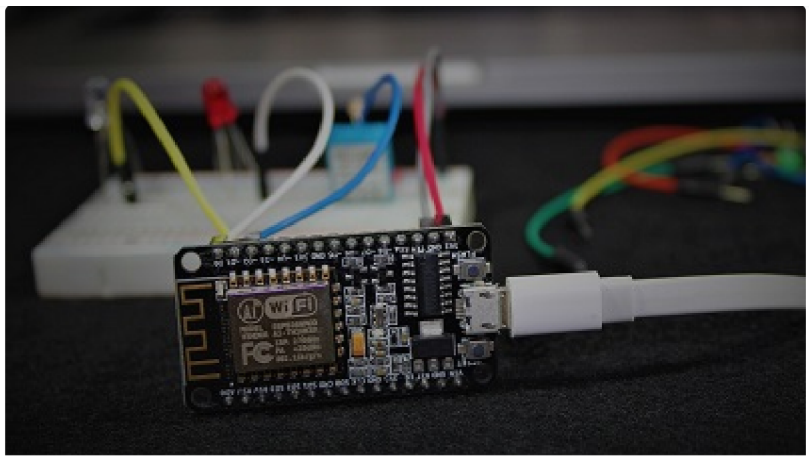
SMART HOME with Raspberry Pi, ESP32, ESP8266 [eBook]

Learn how to build a home automation system and we'll cover the following main subjects: Node-RED, Node-RED Dashboard, Raspberry Pi, ESP32, ESP8266, MQTT, and InfluxDB database **DOWNLOAD »**

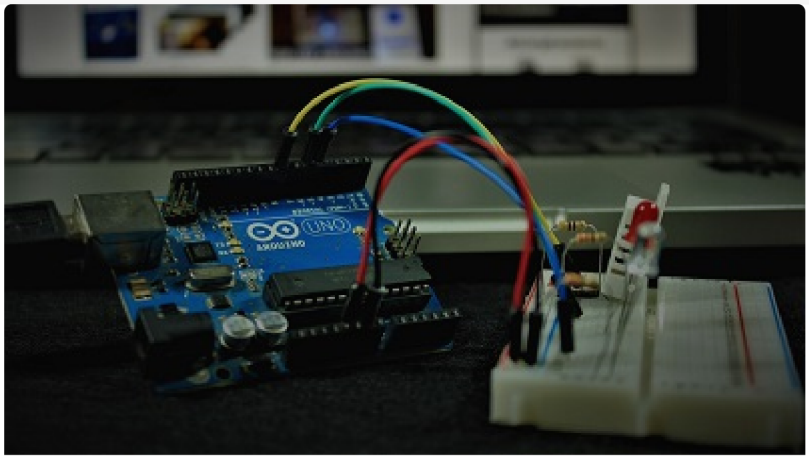
Recommended Resources



Build a Home Automation System from Scratch » With Raspberry Pi, ESP8266, Arduino, and Node-RED.



Home Automation using ESP8266 eBook and video course » Build IoT and home automation projects.



Arduino Step-by-Step Projects » Build 25 Arduino projects with our course, even with no prior experience!

What to Read Next…

ESP32 Remote-Controlled Wi-Fi Car Robot (Arduino IDE)

ESP32 Deep Sleep with Arduino IDE and Wake Up Sources

ESP-NOW with ESP8266: Receive Data from Multiple Boards (many-to-one)

Enjoyed this project? Stay updated by subscribing our newsletter!

Your Email Address

SUBSCRIBE

21 thoughts on “ESP32 Web Server: Control Stepper Motor (WebSocket)”

Bert

October 16, 2021 at 6:07 am

Can you help me ?

ESP32_Stepper_Motor_Websocket:141:6: error: ‘class AsyncWebSocket’ has no member named ‘cleanupClients’

ws.cleanupClients();

Reply

Sara Santos
October 16, 2021 at 10:32 am

Hi.
What's the ESP32 boards version that you have? Did you install all the required libraries?
The code compiles just fine for me. I'm using ESP32 boards version 1.0.6.
Go to Tools > Board > Boards Manager search for "ESP32" and check the version you have installed.
Regards,
Sara

Reply

Old man Bert
October 16, 2021 at 6:10 pm

Thank you for your response.
I am working with Arduino ide version 1.8.12.
I also use ESP32 boards version 1.0.6
The esp32 is a wroom.
The Esp8266 I used as a test is a 12F version.
With both sketches I kept getting the same error message.
Strangely enough, both sketches work fine when I use the ws.cleanupClients(); rule disable.

Reply

Srečko
October 19, 2021 at 11:49 am

Hi there. Your tutorials are priceless. Thanks for all the good work you do.

Reply

Alex
January 16, 2022 at 6:37 am

I'm joining to you. This is the best learning resource that I know of.

Reply

Sara Santos
January 16, 2022 at 5:10 pm

Thank you so much for your support.
Regards,
Sara

Reply

Alex
January 16, 2022 at 6:34 am

Great tutorial! Can I to use my own local gif animation instead of a gear?

Reply

Sara Santos
January 16, 2022 at 5:10 pm

Hi.
Yes.
If it is stored somewhere on the internet, you just need to use its URL.
Regards,
Sara

Reply

Alex
January 17, 2022 at 8:32 am

I think it is necessary to corrective the link in the file index.html ?

This link leads to the css file. Do I need to adjust the path to the media file? For example

It doesn’t work, or I’m doing something wrong?

Reply

Sara Santos
January 18, 2022 at 7:12 pm

I'm sorry, but I didn't understand your question.
Can you try to reformulate?
Regards,
Sara

Reply

Randy S
April 15, 2022 at 11:43 pm

Enjoyed your presentation.
I am using web sockets to control a stepper motor also. I am trying to implement an emergency stop button, however while the stepper motor move function is running. The web socket will not process any new request until the stepper motor stops. Did you the same issue?

Reply

Sara Santos
April 17, 2022 at 10:37 am

Hi.
Because the Stepper.h is not an asynchronous library, it won't do anything else until the motor has stopped spinning. So, if you try to make new requests while the motor is spinning, it will not work.
You can try using another library that doesn't block the code, for example, the AccelStepper:
https://www.airspayce.com/mikem/arduino/AccelStepper/
You can also find a discussion about that here:
https://groups.google.com/g/accelstepper/c/wusHVDxhufw?pli=1
We use that library with the ESP8266 if you want to take a look:

https://randomnerdtutorials.com/stepper-motor-esp8266-websocket/
I hope this helps.
Regards,
Sara

Reply

Pascal I.
June 17, 2022 at 4:08 pm

Hello,

I tried your project. It works but sometimes I have this message in the serial monitor :

CWsteps10
directionCW
CWWebSocket client #5 disconnected
Guru Meditation Error: Core 1 panic'ed (LoadProhibited). Exception was unhandled.
Core 1 register dump:
PC : 0x40154ac3 PS : 0x00060e30 A0 : 0x800d4956 A1 : 0x3ffb1ee0
A2 : 0x3ffb1f2c A3 : 0x00000001 A4 : 0x00000003 A5 : 0x3ffb1e00
A6 : 0x00000000 A7 : 0x3ffb0060 A8 : 0x00060023 A9 : 0x3ffb8058
A10 : 0x00000000 A11 : 0x00000000 A12 : 0x8008dd84 A13 : 0x3ffd0a20
A14 : 0x00000000 A15 : 0x3ffb0060 SAR : 0x0000000a EXCCAUSE: 0x0000001c
EXCVADDR: 0x00000001 LBEG : 0x4008b74c LEND : 0x4008b768
LCOUNT : 0xffffffff

ELF file SHA256: 0000000000000000

Backtrace: 0x40154ac3:0x3ffb1ee0 0x400d4953:0x3ffb1f00
0x400d4a6f:0x3ffb1f20 0x400d4ab5:0x3ffb1f60 0x400d10d6:0x3ffb1f80
0x400de2b5:0x3ffb1fb0 0x4008db72:0x3ffb1fd0

Rebooting…
ets Jun 8 2016 00:22:57

rst:0xc (SW_CPU_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x0
0
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:10944
load:0x40080400,len:6360
entry 0x400806b4
E (96) psram: PSRAM ID read error: 0xffffffff
Connecting to WiFi …..192.168.1.92
SPIFFS mounted successfully
WebSocket client #1 connected from 192.168.1.55

My ESP is an ESP32MCU-Node

What is the problem ?

Regards

Reply

David
August 10, 2022 at 11:45 am

Hi,
excellent tutorials as always. Keep up the good work.

Regarding stepper motors – my stepper motor has the following 6 wires: blue, pink, yellow, orange, red, red.

Is there a way to figure which wires go to which connections?
Thanks
David

Reply

Sara Santos
August 12, 2022 at 10:18 am

Hi.
What's the reference for your stepper motor?
Regards,
Sara

Reply

hatta
March 20, 2023 at 8:11 pm

Hi,
excellent tutorial as always. Keep up the good work.
regarding the stepper motor control, can this be controlled from anywhere, I got the assignment to be able to control it from anywhere, thank you

Reply

Ernster Klein
March 29, 2024 at 7:29 pm

Dear RANDOM NERD TUTORIALS!

Can you please tell me what version of the Arduino IDE I need to have to make your program work?

Is there a chance that it can work with newest Arduino IDE 2.3.2 too?

Reply

Sara Santos
March 30, 2024 at 10:29 am

Hi.
It should work with any version.
But, for uploading the filesystem image, you need to use Arduino 1.8.x
Regards,
Sara

Reply

Marco
April 15, 2024 at 4:56 pm

Thank you for the tutorial!
I just have one question, my motor stops shortly after starting and only starts again when the set cycles are about to end. What could it be?

Regards
Marco

Reply

Sara Santos
April 15, 2024 at 8:53 pm

Hi.
Check the power supply.
Regards,
Sara

Reply

Marius
September 26, 2024 at 7:47 pm

Hello Sara,
I have a problem with ESPAsyncWebServer by lacamera:

I get an Error with ESPAsyncWebServer 3.1.2 and also with 3.1.0 .
I tried Arduino IDE 1.8.19, 2.3.0, 2.3.2 and 2.3.3 on different PC's.

Are you using the same ESPAsyncWebServer?
if not, how can I get another make?

Reply

Leave a Comment

Name *

Email *

Website

Notify me of follow-up comments by email.

Notify me of new posts by email.

Post Comment

Learn ESP32 with Arduino IDE (2nd Edition) Course » Complete guide to program the ESP32 with Arduino IDE!

SMART HOME with Raspberry Pi, ESP32, and ESP8266 » learn how to build a complete home automation system.

Learn Raspberry Pi Pico/Pico W with MicroPython » The complete getting started guide to get the most out of the the Raspberry Pi Pico/Pico W (RP2040) microcontroller board using MicroPython programming language.

🔥 **Learn LVGL: Build GUIs for ESP32 Projects »** Learn how to build Graphical User Interfaces (GUIs) for ESP32 Projects using LVGL (Light Versatile Graphics Library) with the Arduino IDE.