2024 - 2025

|  |
| --- |
|  |
| Documentatie  Bestuurbare ventilator |
| Quentin Costermans |

Inhoud

[1. Inleiding 3](#_Toc194055798)

[1.1. Gebruikte componenten 3](#_Toc194055799)

[1.2. Functionaliteiten 3](#_Toc194055800)

[2. Schema 4](#_Toc194055801)

[3. Foto 5](#_Toc194055802)

[4. Software 6](#_Toc194055803)

[5. Webapplicatie 14](#_Toc194055804)

[5.1.1. Functionaliteiten: 14](#_Toc194055805)

[5.1.2. Software HTML 15](#_Toc194055806)

[5.1.3. Software CSS 16](#_Toc194055807)

[6. Probleemoplossingen 18](#_Toc194055808)

[6.1. Voedingsprobleem en oplossing 18](#_Toc194055809)

[6.2. Motorvervanging door LED 18](#_Toc194055810)

[7. Bronnen 18](#_Toc194055811)

# Inleiding

Dit document beschrijft de werking en implementatie van een bestuurbare ventilator, waarbij een Superflux LED wordt gebruikt als simulatie voor de motor (had geen component om de motor aan te kunnen sturen). Het systeem maakt gebruik van een ESP32-microcontroller en verschillende sensoren en actuatoren om een op afstand bedienbare ventilator te realiseren.

## Gebruikte componenten

* **ESP32**: De centrale microcontroller die de besturing van het systeem regelt.
* **DHT11**: Een temperatuursensor die temperatuur- en vochtigheidswaarden meet.
* **IR Sensor**: Voor het ontvangen van signalen van de IR-afstandsbediening.
* **IR Afstandsbediening**: Voor het handmatig bedienen van de ventilatorinstellingen.
* **LCD Scherm**: Voor het weergeven van temperatuur, luchtvochtigheid en statusinformatie.
* **6V Voeding**: Voedingsbron voor de servomotoren en andere componenten.
* **Superflux LED**: Ter simulatie van een draaiende ventilator.
* **2x Servomotoren**: Voor de pan/tilt-functionaliteit van de ventilator.

## Functionaliteiten

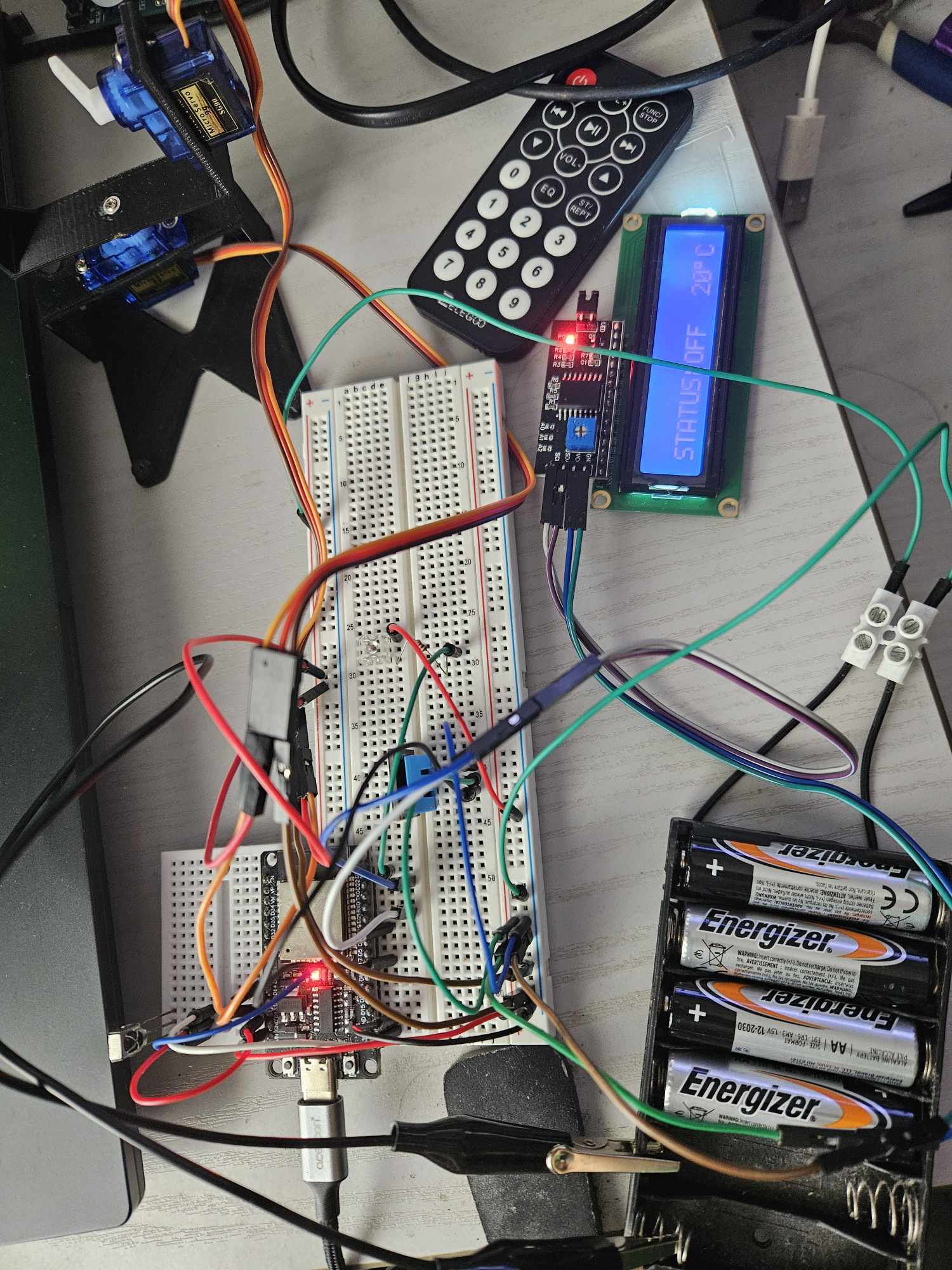
* **Handmatige bediening**: Met een IR-afstandsbediening kan de gebruiker de ventilator in- of uitschakelen en de richting aanpassen.
* **Pan/Tilt-mechanisme**: Twee servomotoren zorgen ervoor dat de ventilator in verschillende richtingen kan bewegen.
* **LCD-weergave**: Live weergave van temperatuur en ventilatorstatus.
* **Lange afstands**-bediening: De webapplicatie zorgt voor bediening via het netwerk.

In de volgende delen wordt uitgelegd hoe de hardware is aangesloten en hoe de software is geïmplementeerd om deze functionaliteiten mogelijk te maken.

# SchemaA diagram of a circuit board AI-generated content may be incorrect.

* **DHT11**: GPIO 17
* **IR Sensor**: GPIO 19
* **LCD Scherm**: SDA GPIO 21 | SCL GPIO 22
* **Superflux LED**: GPIO 23
* **Servomotor PAN**: GPIO 13
* **Servomotor TILT**: GPIO 14

# Foto



# Software

|  |
| --- |
| #include <WiFi.h> |
| #include <ESPAsyncWebServer.h> |
| #include <SPIFFS.h> |
| #include <DIYables\_IRcontroller.h> |
| #include <ESP32Servo.h> |
| #include <LiquidCrystal\_I2C.h> |
| #include <DHT.h> |
|  |
| // Define hardware pins |
| #define DHT11\_PIN 17 // ESP32 GPIO17 connected to DHT11 sensor |
| #define IR\_RECEIVER\_PIN 19 // ESP32 GPIO19 connected to IR receiver |
| #define LED\_PIN 23 // ESP32 GPIO23 for LED control |
|  |
| // Initialize IR controller with debounce time of 200ms |
| DIYables\_IRcontroller\_21 irController(IR\_RECEIVER\_PIN, 200); |
|  |
| // Initialize DHT11 temperature sensor |
| DHT dht11(DHT11\_PIN, DHT11); |
|  |
| // Define servo motor control pins |
| static const int servoPan = 13; |
| static const int servoTilt = 14; |
|  |
| // Servo motor instances |
| Servo servoP; |
| Servo servoT; |
|  |
| // LCD configuration |
| int lcdColumns = 16; |
| int lcdRows = 2; |
|  |
| // Set LCD address and initialize it |
| // If unsure about the address, run an I2C scanner sketch |
| LiquidCrystal\_I2C lcd(0x27, lcdColumns, lcdRows); |
|  |
| // Global variables to track system state |
| bool state = false; // Stores LED on/off state |
| int ServoPPos; // Tracks pan servo position |
| int ServoTPos; // Tracks tilt servo position |
|  |
| // WiFi Credentials (Replace with actual credentials) |
| const char\* ssid = "YOUR SSID"; |
| const char\* password = "YOUR PASSWORD"; |
|  |
| // Create an instance of the web server on port 80 |
| AsyncWebServer server(80); |
|  |
| /\*\* |
| \* @brief Control center function to process commands for light and servo movements. |
| \* @param command String command received (e.g., "ON", "OFF", "UP", "DOWN", etc.) |
| \*/ |
| void controlcenter(String command) { |
|  |
| // Turn light on |
| if (command == "ON") { |
| analogWrite(LED\_PIN, 255); |
| } |
|  |
| // Turn light off |
| if (command == "OFF") { |
| analogWrite(LED\_PIN, 0); |
| } |
|  |
| // Servo tilt up (increments position by 15 degrees) |
| if (command == "UP") { |
| if (ServoTPos < 180) { |
| int newpos = ServoTPos; |
| for (int idx = ServoTPos; idx <= ServoTPos + 15; idx++) { |
| Serial.println(idx); |
| servoT.write(idx); |
| delay(20); |
| newpos = idx; |
| } |
| ServoTPos = newpos; |
| } |
| } |
|  |
| // Servo tilt down |
| if (command == "DOWN") { |
| if (ServoTPos > 0) { |
| int newpos = ServoTPos; |
| for (int idx = ServoTPos; idx >= ServoTPos - 15; idx--) { |
| Serial.println(idx); |
| servoT.write(idx); |
| delay(20); |
| newpos = idx; |
| } |
| ServoTPos = newpos; |
| } |
| } |
|  |
| // Servo pan right |
| if (command == "RIGHT") { |
| if (ServoPPos < 180) { |
| int newpos = ServoPPos; |
| for (int idx = ServoPPos; idx <= ServoPPos + 15; idx++) { |
| Serial.println(idx); |
| servoP.write(idx); |
| delay(20); |
| newpos = idx; |
| } |
| ServoPPos = newpos; |
| } |
| } |
|  |
| // Servo pan left |
| if (command == "LEFT") { |
| if (ServoPPos > 0) { |
| int newpos = ServoPPos; |
| for (int idx = ServoPPos; idx >= ServoPPos - 15; idx--) { |
| Serial.println(idx); |
| servoP.write(idx); |
| delay(20); |
| newpos = idx; |
| } |
| ServoPPos = newpos; |
| } |
| } |
| } |
|  |
| /\*\* |
| \* @brief Displays messages on the LCD screen. |
| \* @param message String message to be displayed. |
| \*/ |
| void writelcd(String message) { |
|  |
| if (message == "ON") { |
| lcd.setCursor(0, 1); |
| lcd.print("STATUS:ON "); |
| } |
| if (message == "OFF") { |
| lcd.setCursor(0, 1); |
| lcd.print("STATUS:OFF"); |
| } |
|  |
| // Display main message |
| lcd.setCursor(0, 0); |
| lcd.print(message); |
| delay(1000); |
| lcd.setCursor(0, 0); |
| lcd.print(" "); // Clear previous message |
| } |
|  |
| /\*\* |
| \* @brief Reads temperature from the DHT11 sensor and displays it on the LCD. |
| \*/ |
| void readtemperature() { |
|  |
| float tempC = dht11.readTemperature(); |
|  |
| // Check if reading is successful |
| if (isnan(tempC)) { |
| Serial.println("Failed to read from DHT11 sensor!"); |
| return; |
| } |
|  |
| if (tempC > 0) { |
| lcd.setCursor(12, 1); |
| String message = String(int(round(tempC))) + (char)223 + "C"; // Format temperature with degree symbol |
| Serial.println(message); |
| lcd.print(message); |
| } |
|  |
| delay(1000); // Delay between readings |
| } |
|  |
| /\*\* |
| \* @brief Setup function to initialize hardware components and WiFi. |
| \*/ |
| void setup() { |
| Serial.begin(115200); |
|  |
| // Initialize SPIFFS for file storage |
| if (!SPIFFS.begin(true)) { |
| Serial.println("SPIFFS initialization failed!"); |
| return; |
| } |
|  |
| // Initialize LCD |
| lcd.init(); |
| lcd.backlight(); |
| lcd.setCursor(0, 1); |
| lcd.print("STATUS:OFF"); |
|  |
| // Initialize IR Controller |
| irController.begin(); |
|  |
| // Initialize servos and set default positions |
| servoP.attach(servoPan); |
| servoT.attach(servoTilt); |
| servoP.write(90); |
| ServoPPos = 90; |
| servoT.write(90); |
| ServoTPos = 90; |
|  |
| // Initialize DHT11 sensor |
| dht11.begin(); |
|  |
| // Initialize LED pin |
| pinMode(LED\_PIN, OUTPUT); |
|  |
| // Connect to WiFi |
| WiFi.begin(ssid, password); |
| Serial.print("Connecting to Wi-Fi..."); |
| while (WiFi.status() != WL\_CONNECTED) { |
| delay(500); |
| Serial.print("."); |
| } |
| Serial.println("\nConnected to Wi-Fi!"); |
| Serial.print("IP Address: "); |
| Serial.println(WiFi.localIP()); |
|  |
| // Define Web Server Routes |
| server.on("/", HTTP\_GET, [](AsyncWebServerRequest \* request) { |
| request->send(SPIFFS, "/index.html", "text/html"); |
| }); |
|  |
| server.on("/style.css", HTTP\_GET, [](AsyncWebServerRequest \* request) { |
| request->send(SPIFFS, "/style.css", "text/css"); |
| }); |
|  |
| server.on("/servo", HTTP\_GET, [](AsyncWebServerRequest \* request) { |
| if (request->hasParam("direction")) { |
| String dir = request->getParam("direction")->value(); |
| controlcenter(dir); |
| request->send(200, "text/plain", "OK"); |
| } |
| }); |
|  |
| server.on("/led", HTTP\_GET, [](AsyncWebServerRequest \* request) { |
| if (request->hasParam("state")) { |
| String ledState = request->getParam("state")->value(); |
| if (ledState == "on") { |
| digitalWrite(LED\_PIN, HIGH); |
| writelcd("ON"); |
| controlcenter("ON"); |
| } else { |
| digitalWrite(LED\_PIN, LOW); |
| writelcd("OFF"); |
| controlcenter("OFF"); |
| } |
| request->send(200, "text/plain", "OK"); |
| } |
| }); |
|  |
| server.on("/temperature", HTTP\_GET, [](AsyncWebServerRequest \* request) { |
| float tempC = dht11.readTemperature(); |
| if (isnan(tempC)) { |
| request->send(500, "text/plain", "Error reading temperature"); |
| } else { |
| request->send(200, "text/plain", String(tempC)); |
| } |
| }); |
|  |
| // Start Web Server |
| server.begin(); |
| } |
|  |
| /\*\* |
| \* @brief Main loop that checks for IR remote commands and updates temperature. |
| \*/ |
| void loop() { |
| Key21 key = irController.getKey(); |
| if (key != Key21::NONE) { |
| switch (key) { |
| case Key21::KEY\_CH\_MINUS: |
| state = !state; |
| writelcd(state ? "ON" : "OFF"); |
| controlcenter(state ? "ON" : "OFF"); |
| break; |
| case Key21::KEY\_CH: |
| writelcd("UP"); |
| controlcenter("UP"); |
| break; |
| case Key21::KEY\_VOL\_PLUS: |
| writelcd("DOWN"); |
| controlcenter("DOWN"); |
| break; |
| case Key21::KEY\_PREV: |
| writelcd("LEFT"); |
| controlcenter("LEFT"); |
| break; |
| case Key21::KEY\_PLAY\_PAUSE: |
| writelcd("RIGHT"); |
| controlcenter("RIGHT"); |
| break; |
| } |
| } |
|  |
| readtemperature(); |
| } |

# Webapplicatie

A screenshot of a computer

AI-generated content may be incorrect.

Deze HTML-pagina biedt een eenvoudige webinterface voor het op afstand bedienen van een ESP32-microcontroller.

### Functionaliteiten:

1. **Temperatuurweergave:**
   * De huidige temperatuur wordt elke 5 seconden opgehaald via /temperature en weergegeven in <span id="temp">.
2. **Servo-besturing:**
   * De knoppen **Up, Down, Left, Right** sturen een HTTP-verzoek naar /servo?direction=<richting> om de servo’s te verplaatsen.
3. **LED-besturing:**
   * De knoppen **Turn LED ON / OFF** sturen een HTTP-verzoek naar /led?state=on of /led?state=off om de LED te schakelen.

Deze pagina maakt gebruik van fetch() om commando’s naar de ESP32-server te sturen en updates binnen te halen.

### Software HTML

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>ESP32 Web Control</title>

    <link rel="stylesheet" href="style.css">

</head>

<body>

    <h1>ESP32 Remote Control</h1>

    <h2>Temperature: <span id="temp">--</span>°C</h2>

    <div class="controls">

        <button onclick="sendCommand('UP')">Up</button><br>

        <button onclick="sendCommand('LEFT')">Left</button>

        <button onclick="sendCommand('RIGHT')">Right</button><br>

        <button onclick="sendCommand('DOWN')">Down</button>

    </div>

    <div class="led">

        <button onclick="toggleLED('on')">Turn LED ON</button>

        <button onclick="toggleLED('off')">Turn LED OFF</button>

    </div>

    <script>

        function sendCommand(direction) {

            fetch(`/servo?direction=${direction}`)

                .then(response => console.log(response));

        }

        function toggleLED(state) {

            fetch(`/led?state=${state}`)

                .then(response => console.log(response));

        }

        function updateTemperature() {

            fetch("/temperature")

                .then(response => response.text())

                .then(temp => document.getElementById("temp").innerText = temp);

        }

        setInterval(updateTemperature, 5000);

    </script>

</body>

</html>

### Software CSS

/\* General Styles \*/

body {

    font-family: Arial, sans-serif;

    text-align: center;

    background-color: #f4f4f4;

    margin: 0;

    padding: 20px;

}

/\* Title \*/

h1 {

    color: #333;

    margin-bottom: 10px;

}

h2 {

    color: #555;

    font-weight: normal;

}

/\* Controls Container \*/

.controls, .led {

    margin: 20px auto;

    padding: 15px;

    background: #fff;

    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);

    border-radius: 8px;

    display: inline-block;

}

/\* Buttons \*/

button {

    background-color: #007bff;

    color: white;

    border: none;

    padding: 12px 18px;

    margin: 5px;

    border-radius: 6px;

    font-size: 16px;

    cursor: pointer;

    transition: background 0.3s ease;

}

button:hover {

    background-color: #0056b3;

}

/\* Control Button Grid \*/

.controls {

    display: grid;

    grid-template-columns: repeat(3, 1fr);

    gap: 10px;

    max-width: 200px;

}

.controls button:nth-child(1) { grid-column: 2; } /\* Up \*/

.controls button:nth-child(2) { grid-column: 1; } /\* Left \*/

.controls button:nth-child(3) { grid-column: 3; } /\* Right \*/

.controls button:nth-child(4) { grid-column: 2; } /\* Down \*/

/\* Responsive Design \*/

@media (max-width: 600px) {

    .controls {

        max-width: 100%;

        grid-template-columns: repeat(2, 1fr);

    }

    button {

        font-size: 14px;

        padding: 10px;

    }

}

# Probleemoplossingen en mogelijke uitbereidingen

## Problemen

### Voedingsprobleem en oplossing

Tijdens de implementatie bleek dat de servomotoren niet voldoende voeding kregen vanuit de ESP32, wat resulteerde in een onbetrouwbare werking. Om dit probleem op te lossen, is een aparte 6V voeding toegevoegd om de servomotoren van voldoende stroom te voorzien. Hierdoor functioneren de motoren nu stabiel en kan het pan/tilt-mechanisme correct werken.

### Motorvervanging door LED

Aanvankelijk was het plan om een DC-motor te gebruiken voor de ventilator, maar dit vereiste een ULN2003-component om de motor correct aan te sturen. Omdat dit component niet beschikbaar was, werd besloten om in plaats daarvan een Superflux LED te gebruiken als simulatie van een draaiende ventilator. Dit zorgt ervoor dat het systeem correct functioneert en de werking van de ventilator visueel kan worden weergegeven.

## Uitbereidingen

Dit systeem kan verder worden uitgebreid met extra functionaliteiten, zoals:

* Ventilator modi: Verschillende snelheidsstanden instellen, bijvoorbeeld een automatische modus die reageert op temperatuur en een handmatige modus waarin de gebruiker de snelheid bepaalt.
* PWM-aansturing: De LED kan worden gedimd om de ventilatorsnelheid na te bootsen.
* Extra sensoren: Integreren van een luchtdruksensor of CO₂-meter om de luchtkwaliteit te monitoren.

# Bronnen

* <https://randomnerdtutorials.com/esp32-servo-motor-web-server-arduino-ide/>
* <https://randomnerdtutorials.com/esp32-esp8266-i2c-lcd-arduino-ide/>
* <https://randomnerdtutorials.com/esp32-dht11-dht22-temperature-humidity-web-server-arduino-ide/>
* <https://randomnerdtutorials.com/esp32-dht11-dht22-temperature-humidity-sensor-arduino-ide/>
* <https://www.instructables.com/One-and-Multiple-Servo-Motor-Control-With-ESP32-De/>
* <https://esp32io.com/tutorials/esp32-ir-remote-control>