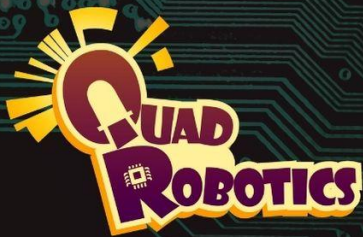


BEST SELLER

ESP32 STARTER KIT

WITH EXPANSION BOARD



A unit of Quad Store

WWW.QUADSTORE.IN

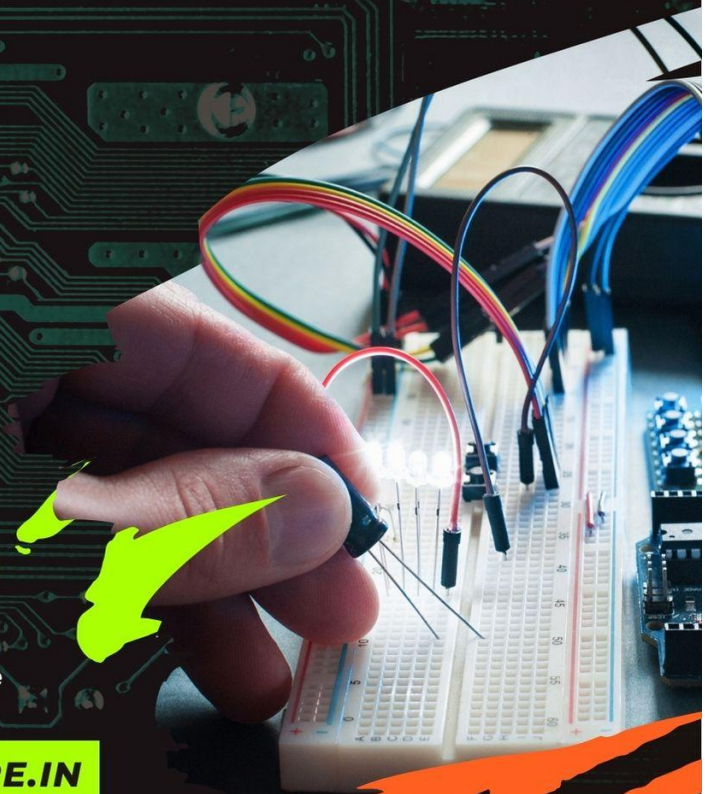


Table of Contents

Introduction:

- About ESP32
- EP32 Pinout and Features Overview
- ESP32 GPIO Pin Usage
- Peripheral Interfaces

Need Help? Ask our AI mentor available 24x7 Support

Initial Setup: How to Use ESP32 Board with Arduino IDE

WARRANTY: Register to claim warranty (within 7 days)

Project 1: Blinking an External LED with ESP32

Project 2: Using an RGB LED Module with ESP32

Project 3: Push Button Control of LED with ESP32

Project 4: LED Brightness Control with Potentiometer and ESP32

Project 5: Using an Active Buzzer with ESP32

Project 6: Play Tones on a Passive Buzzer with ESP32

Project 7: Light-Activated Red LED Using LDR Module

Project 8: Obstacle Detection with IR Sensor and RGB LED

Project 9: Displaying Text on OLED with ESP32

Project 10: Read Temperature & Humidity from DHT11 Sensor

Project 11: Web Server-Based Inbuilt LED Toggle

Project 12: Dual Control of Output via Web and Push Button

Project 13: Web Server to Control 2-Channel Relay

Project 14: Web Server to Display DHT11 Temperature on Mobile

Introduction: About ESP32

ESP32 Overview

The **ESP32** is a powerful, low-cost microcontroller with built-in **Wi-Fi** and **Bluetooth** capabilities, ideal for IoT and embedded applications. Developed by Espressif Systems, it features a **dual-core Tensilica processor**, generous RAM/flash memory, and a rich set of peripherals, making it suitable for real-time tasks, sensor data handling, wireless communication, and automation projects.

Key Features:

- Dual-core 32-bit Xtensa LX6 processor
- Clock speed: Up to 240 MHz
- Built-in Wi-Fi (802.11 b/g/n) and Bluetooth (Classic & BLE)
- 520 KB SRAM, external flash support
- Rich peripheral interfaces: ADC, DAC, PWM, I2C, SPI, UART, etc.

Common ESP32 Pin Details (based on DevKit V1):

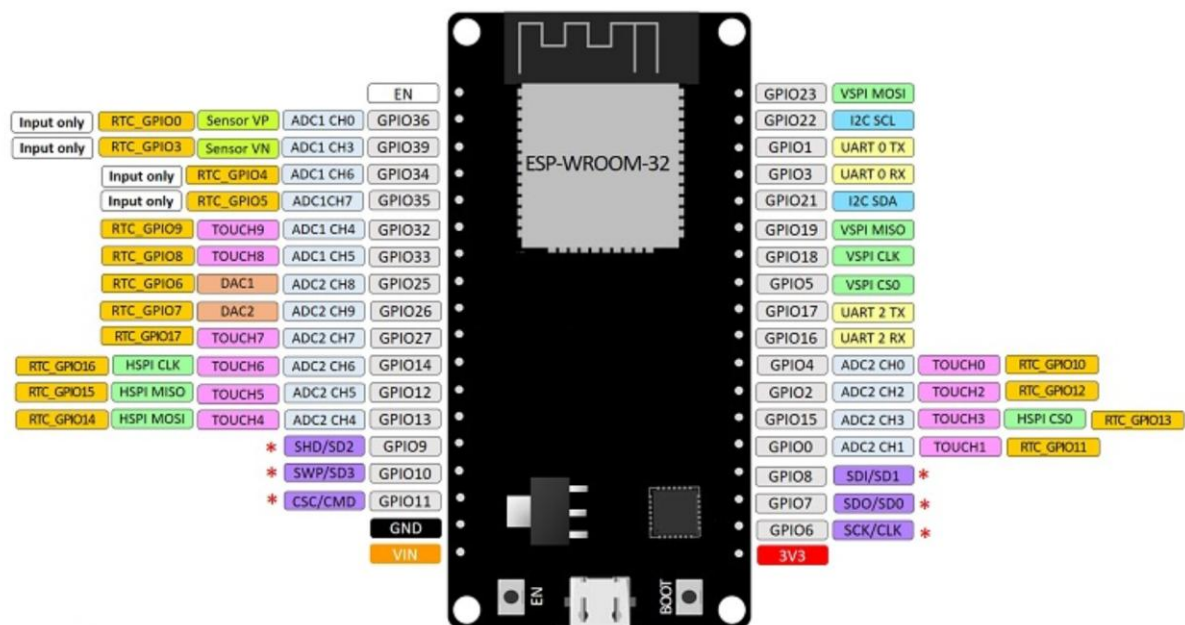
Pin	Function
GPIO 0	Used for boot mode selection
GPIO 1, 3	UART TX/RX (used for serial communication)
GPIO 2	General I/O, often used for onboard LED
GPIO 4, 5	General I/O (can be used for I2C, PWM, etc.)
GPIO 12–15	SPI (can also be used as GPIOs)
GPIO 21, 22	Default I2C SDA and SCL
GPIO 32–39	ADC (Analog Input)
EN	Reset (Enable pin to restart the board)
VIN (or 5V)	Power input (5V from USB)
3.3V	Regulated output (use to power sensors)
GND	Ground

 **Note:** Some pins (e.g., GPIOs 6–11) are used for internal flash memory and **should not be used** for I/O in most development boards.

ESP32 Pinout & Features Overview:

The **ESP32** is a powerful microcontroller developed by **Espressif Systems**, widely used in IoT applications for its integrated **Wi-Fi**, **Bluetooth**, and a rich set of **peripherals**. Thanks to its **flexible GPIO multiplexing**, most functions can be assigned to multiple pins via software.

ESP32 DEVKIT V1 – DOIT version with 36 GPIOs



Key Peripheral Features:

- 18 ADC (Analog-to-Digital Converter) channels
- 3 SPI (Serial Peripheral Interface) interfaces
- 3 UART (Universal Asynchronous Receiver/Transmitter) interfaces
- 2 I2C (Inter-Integrated Circuit) interfaces
- 16 PWM (Pulse Width Modulation) output channels
- 2 DAC (Digital-to-Analog Converter) channels
- 2 I2S (Integrated Inter-IC Sound) interfaces
- 10 Capacitive touch GPIOs

ESP32 GPIO Pin Usage

GPIO	Input	Output	Notes
0	Yes	Yes	Strapping pin; must be LOW to flash
1 (TX)	No	Yes	Debug output at boot
2	Yes	Yes	On-board LED; must be LOW or floating to flash
3 (RX)	Yes	No	HIGH at boot
4–5	Yes	Yes	General-purpose I/O
12–15	Yes	Yes	Strapping pins; caution during boot
16–19	Yes	Yes	Safe to use
21–23	Yes	Yes	Default I2C/SPI pins
25–27	Yes	Yes	DAC available on 25 & 26
32–33	Yes	Yes	Also capacitive touch capable
34–39	Yes	No	Input-only; no pull-up/pull-down resistors

Special Pin Considerations

⚠ Do Not Use These GPIOs

- **GPIO 6–11:** Internally connected to SPI flash, not usable for I/O.

● Input-Only Pins

- **GPIOs 34, 35, 36, 39:** Use only for reading input signals (no output capability, no internal pull resistors).

Analog-to-Digital Converter (ADC)

- 18 ADC input channels (12-bit resolution: 0–4095 range for 0V–3.3V).
- **ADC1 (8 channels):** GPIOs 32–39
- **ADC2 (10 channels):** GPIOs 0, 2, 4, 12–15, 25–27 (*Note: ADC2 conflicts with Wi-Fi use.*)

Digital-to-Analog Converter (DAC)

- 2 DAC outputs:
 - **DAC1 → GPIO25**
 - **DAC2 → GPIO26**
-

PWM (Pulse Width Modulation)

- 16 independent channels.
 - All output-capable GPIOs (except 34–39) can generate PWM signals.
 - Adjustable frequency, duty cycle, and resolution in code.
-

I2C Communication

- Two I2C interfaces, assignable to any GPIO.
 - **Default (Arduino IDE):**
 - SDA → GPIO21
 - SCL → GPIO22
-

SPI Communication

SPI Bus	MOSI	MISO	SCK	CS
VSPI	GPIO23	GPIO19	GPIO18	GPIO5
HSPI	GPIO13	GPIO12	GPIO14	GPIO15

Capacitive Touch Pins

- Detect human touch on GPIOs:
 - T0–T9 → GPIOs 4, 0, 2, 15, 13, 12, 14, 27, 33, 32
-

RTC GPIOs (Deep Sleep Support)

Used for waking from deep sleep:

- GPIOs 0, 2, 4, 12–15, 25–27, 32–36, 39
-

Strapping Pins (Boot Mode Control)

Special boot behavior; avoid external components here unless required:

Pin	Boot Function
GPIO0	Must be LOW to flash
GPIO2	Must be LOW or floating at boot
GPIO5	Should be HIGH at boot
GPIO12	Must be LOW at boot
GPIO15	Should be HIGH at boot

Enable (EN) Pin

- Used to **reset/restart** the ESP32. Pulled up internally; pulling it LOW disables the 3.3V regulator.

Need Help? Ask our Smart AI Mentor:

 **NEED HELP?**

Meet QuadBot AI Assistant

Your Smart Robotics Learning Assistant

Stuck on a project? Facing an error?
Have a question about robotics or coding?
QuadBot AI Assistant is here to help you anytime!



Hi!
I'm QuadBot.
How can I help you today?

**24/7 AI SUPPORT**

QuadBot can help you with:

-  Fixing coding errors
-  Circuit connection guidance
-  Arduino IDE & ESP32 setup help
-  Sensor troubleshooting
-  Robotics project ideas
-  Understanding electronics concepts
-  Step-by-step explanations for beginners
-  AI-powered learning support for students

GET HELP INSTANTLY!

Scan the QR code or click the link below to chat with **QuadBot AI Assistant**.



Scan me!

OR CLICK THE LINK BELOW

 <https://tinyurl.com/QuadBot-ai>

EXAMPLE QUESTIONS YOU CAN ASK

“ Why is my Arduino not connecting? ”

“ How do I connect the ultrasonic sensor? ”

“ Can you explain this code for beginners? ”

“ My motor is not rotating. What should I check? ”

“ Give me project ideas using ESP32. ”



**Learn Faster.
Build Smarter.
With QuadBot!**

QuadBot AI Assistant is designed especially for students, beginners, teachers, and hobbyists to make robotics learning **easier, faster and more exciting!**



 Trusted Support

 24/7 Availability

 Made for Learners

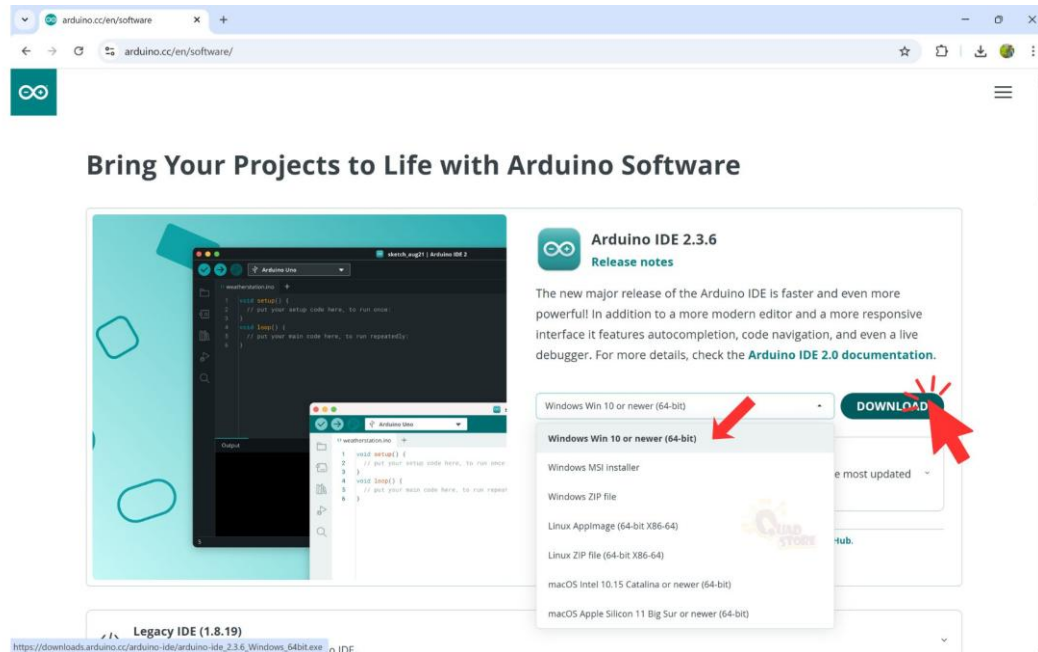
 Here to Help You Succeed!

Initial Setup: How to Use ESP32 Board with Arduino IDE

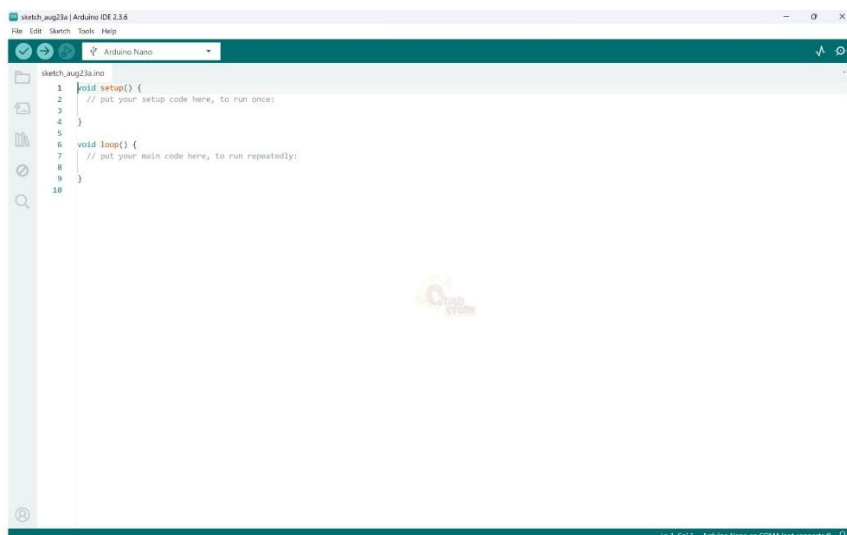
✅ Step 1: Install the Arduino IDE

1.1. Visit the official Arduino website: <https://www.arduino.cc/en/software>

1.2. Download and install the Arduino IDE for Windows, Mac, or Linux based on your hardware



1.3. Open the IDE by clicking on the desktop icon after installation. IDE should open as shown.



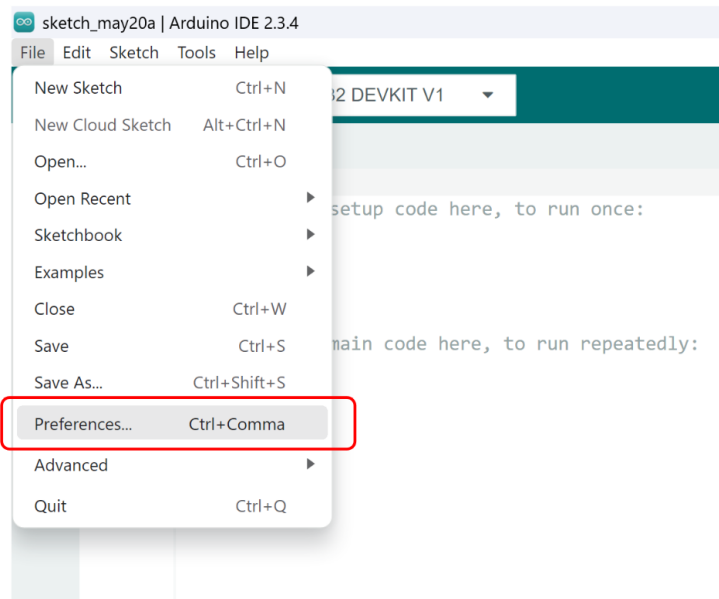
💡 Recommended: Use Arduino IDE version 1.8.x or higher for best ESP32 compatibility.

✅ Step 2: Install ESP32 Board Package

The **Arduino IDE does not support ESP32 out of the box**. You need to install the ESP32 board definitions.

♦ 2.1 Open Arduino IDE

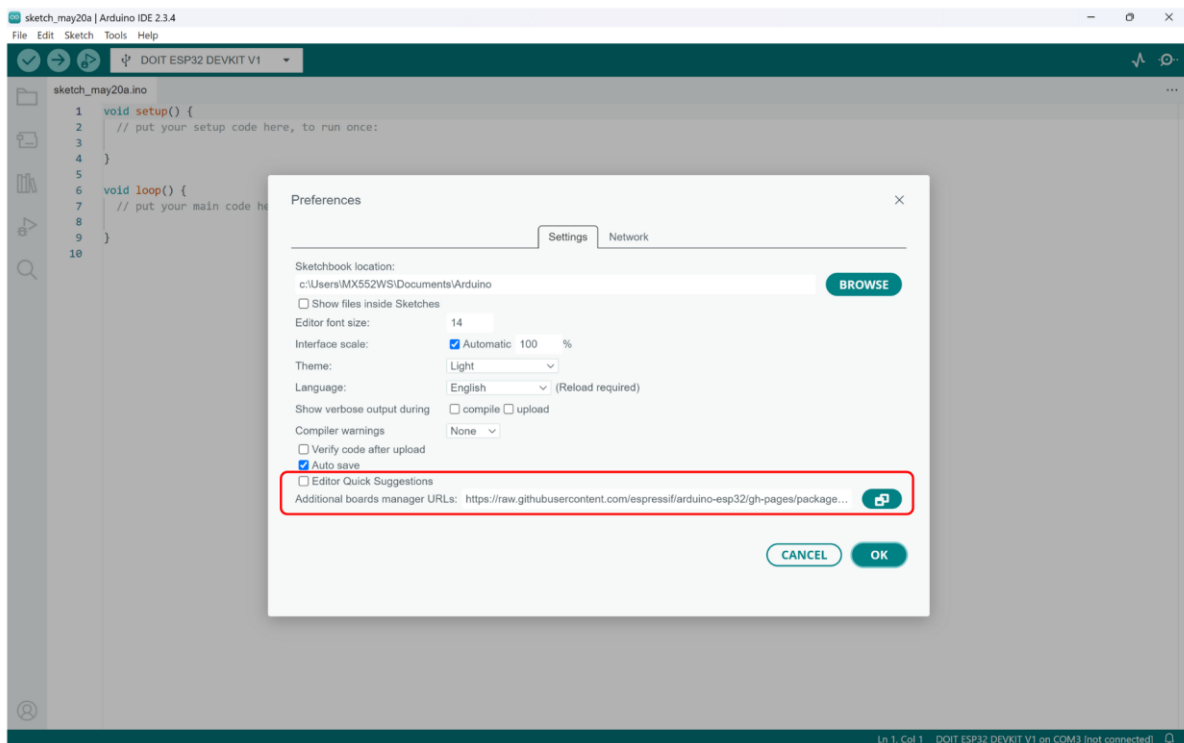
Go to **File > Preferences**



♦ 2.2 Add Board Manager URL

In the "Additional Board Manager URLs" field, paste the following link:

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

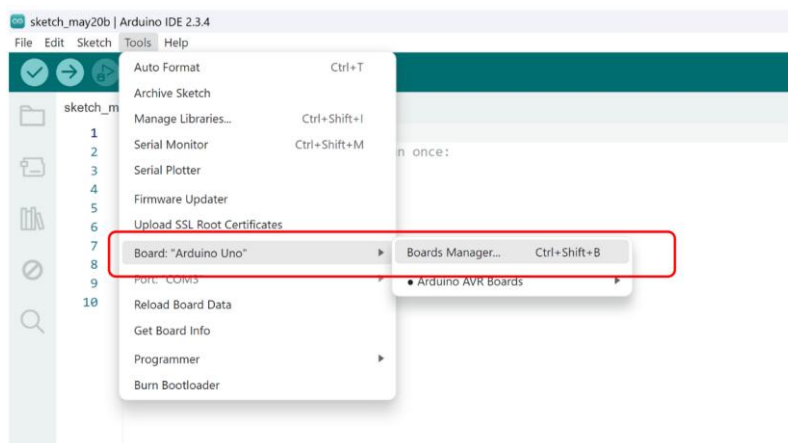


⚠ If you already have a URL there, separate this one with a comma.

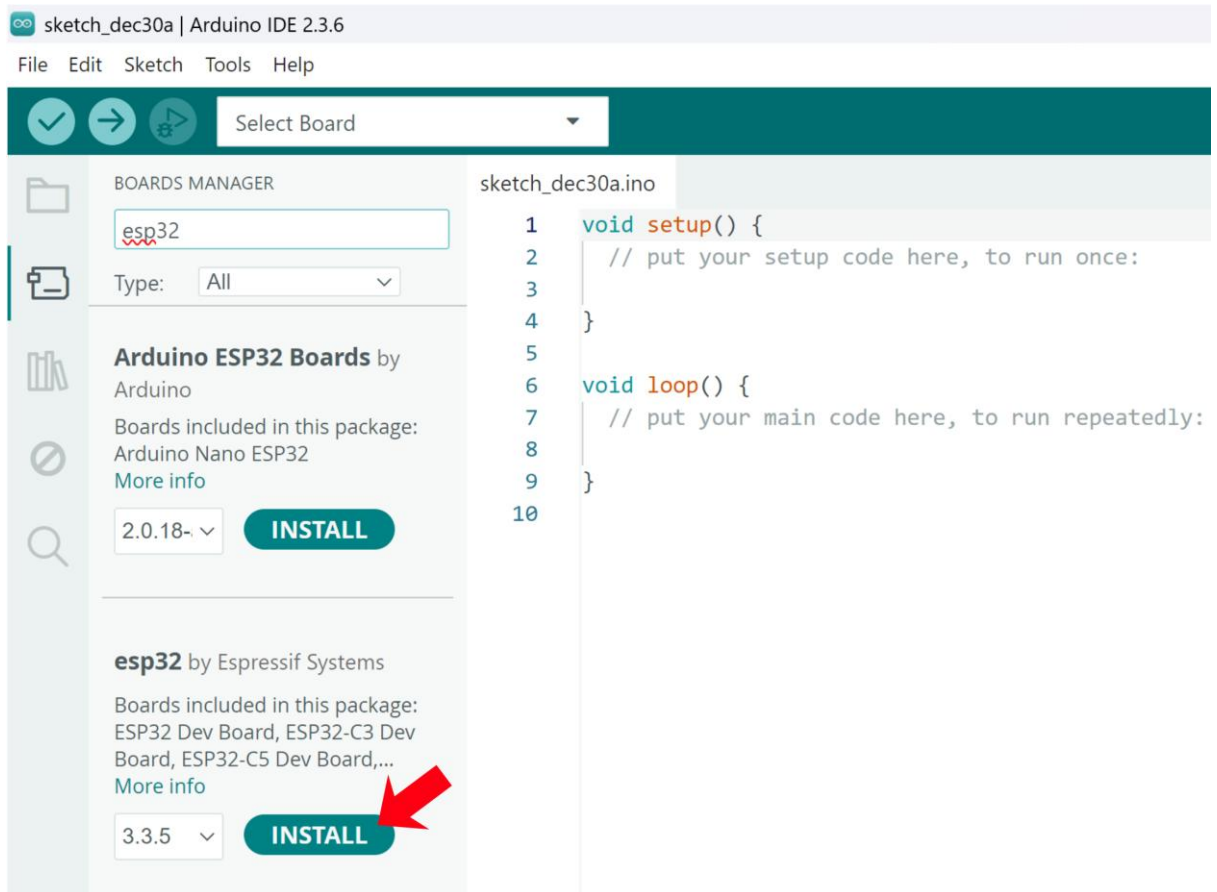
Click OK.

✅ Step 3: Install the ESP32 Boards

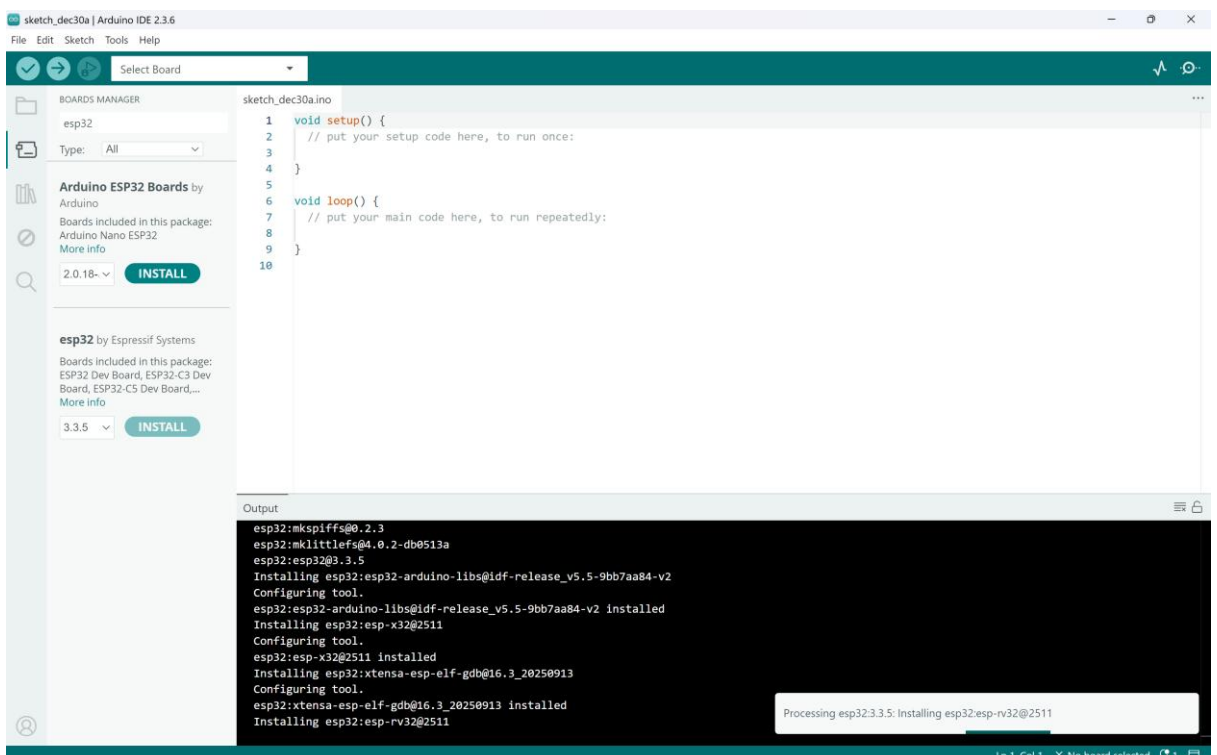
1. Go to **Tools > Board > Boards Manager**.



2. In the search bar, type **ESP32**. Locate **"ESP32 by Espressif Systems"** and click **install**. Install the latest version that is available.



Installation should progress

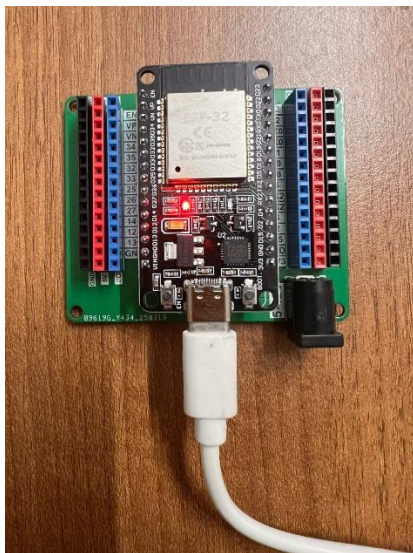


- Wait for the installation to complete.

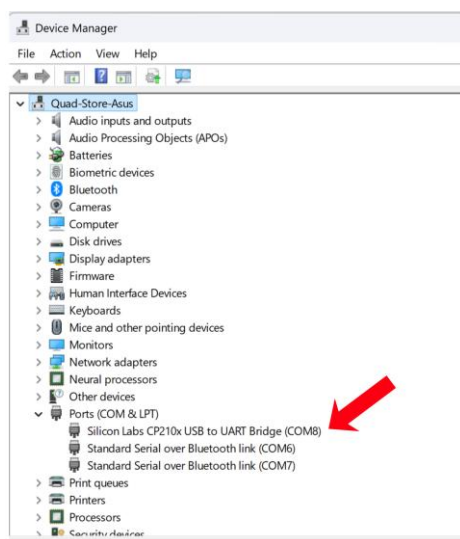
```
Output
esp32:openocd-esp32@v0.12.0-esp32-20250707 installed
Installing esp32:esptool_py@5.1.0
Configuring tool.
esp32:esptool_py@5.1.0 installed
Installing esp32:mksppiffs@0.2.3
Configuring tool.
esp32:mksppiffs@0.2.3 installed
Installing esp32:mklittlefs@4.0.2-db0513a
Configuring tool.
esp32:mklittlefs@4.0.2-db0513a installed
Installing platform esp32:esp32@3.3.5
Configuring platform.
Platform esp32:esp32@3.3.5 installed
```

✅ Step 4: Connect the ESP32 Board to Your PC

- Use the USB cable to connect your ESP32 board to your laptop or computer.



- Open Device Manager (Windows) or System Information (Mac) to find the COM port assigned to your board.



NOTE: If the COM port does not appear in the Device Manager or Arduino IDE, you may need to install the **CP210x USB to UART Bridge VCP drivers** to enable communication.

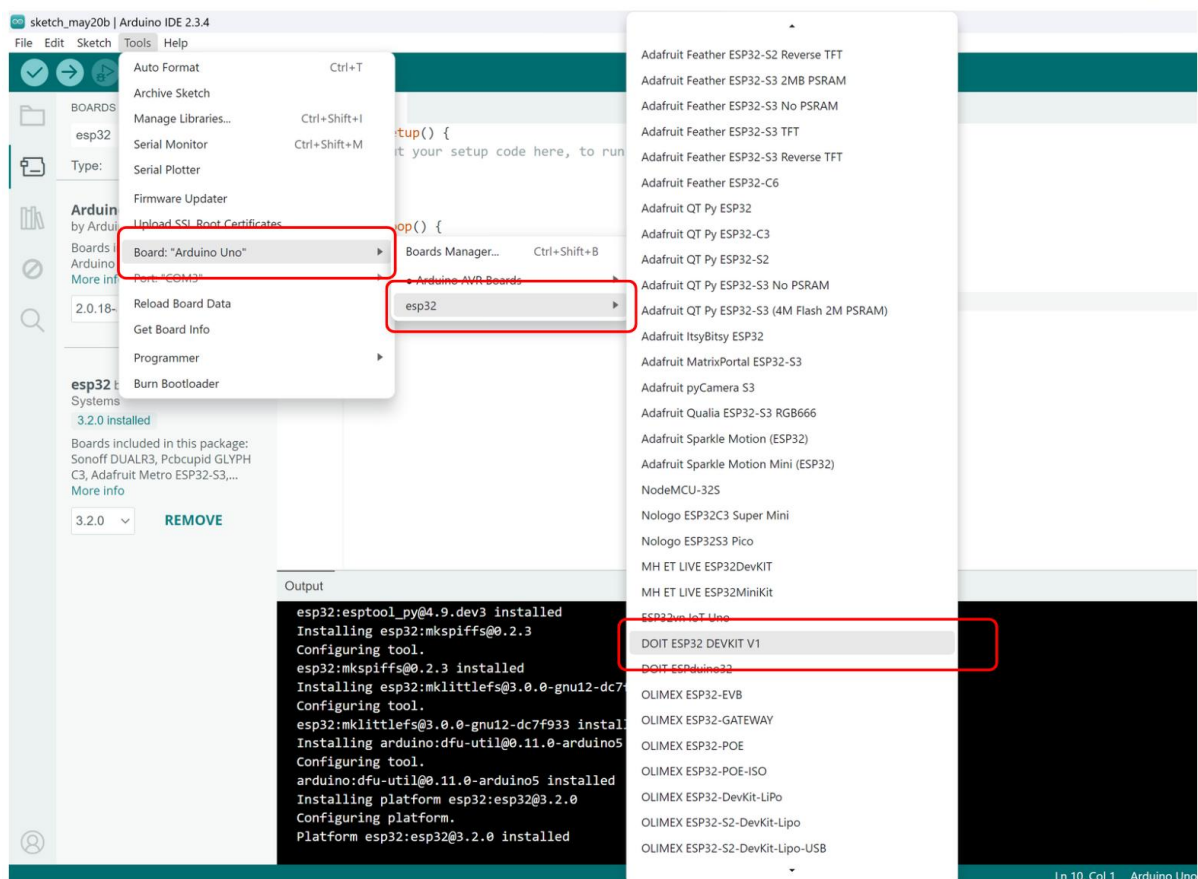
You can download and install the driver from the below link

<https://www.silabs.com/developer-tools/usb-to-uart-bridge-vcp-drivers?tab=downloads>

✓ Step 5: Select the ESP32 Board in Arduino IDE

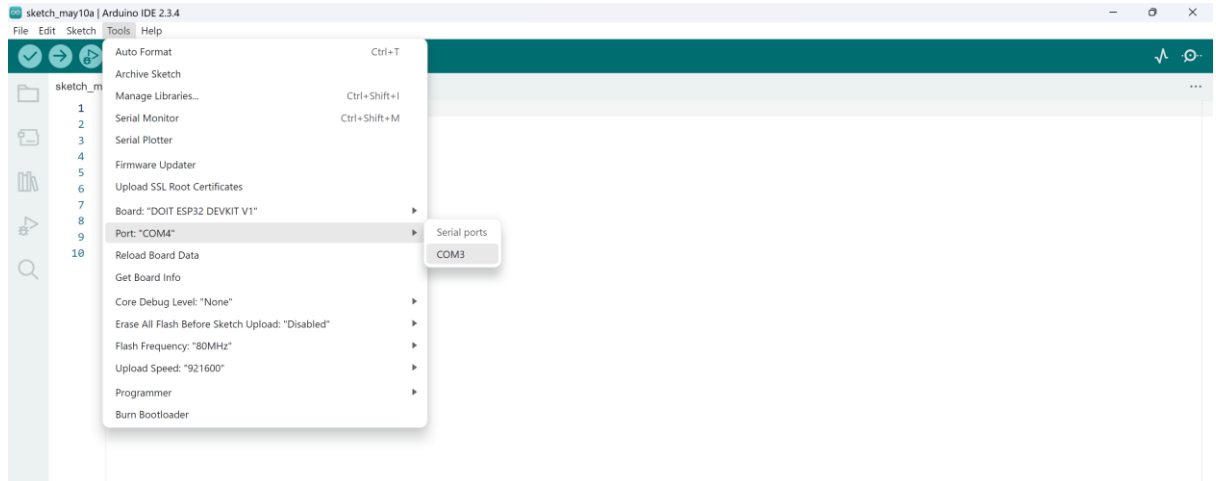
1. Go to **Tools > Board**.
2. Scroll down and choose your board — for example:

✓ DOIT ESP32 DEVKIT V1



✓ Step 6: Select the Correct Port

1. Go to **Tools > Port**.
2. Choose the port corresponding to your ESP32 board (e.g., COM3, COM4, etc.).



💡 **NOTE:** If the COM port does not appear in the Device Manager or Arduino IDE, you may need to install the **CP210x USB to UART Bridge VCP drivers** to enable communication.

You can download and install the driver from the below link as per your operating system.

<https://www.silabs.com/developer-tools/usb-to-uart-bridge-vcp-drivers?tab=downloads>

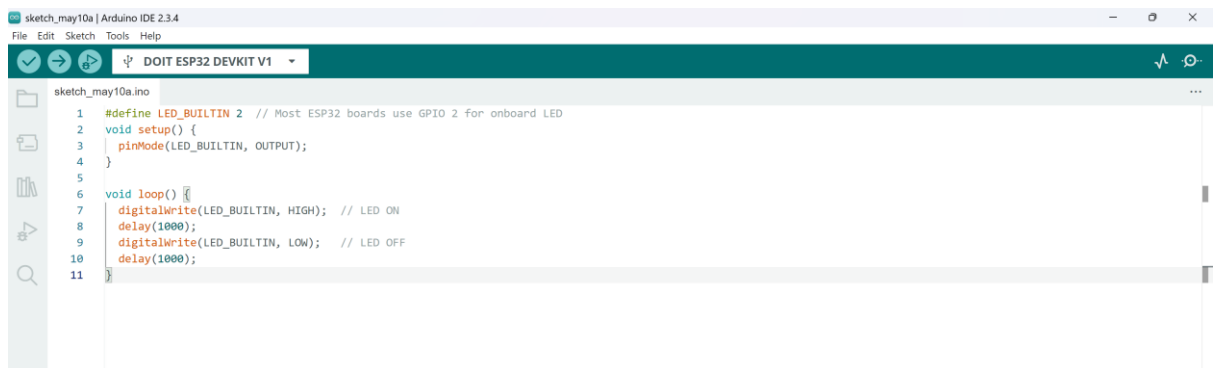
✓ Step 7: Upload a Sample Program (Blink LED)

Copy this code:

```
#define LED_BUILTIN 2 // Most ESP32 boards use GPIO 2 for onboard LED
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

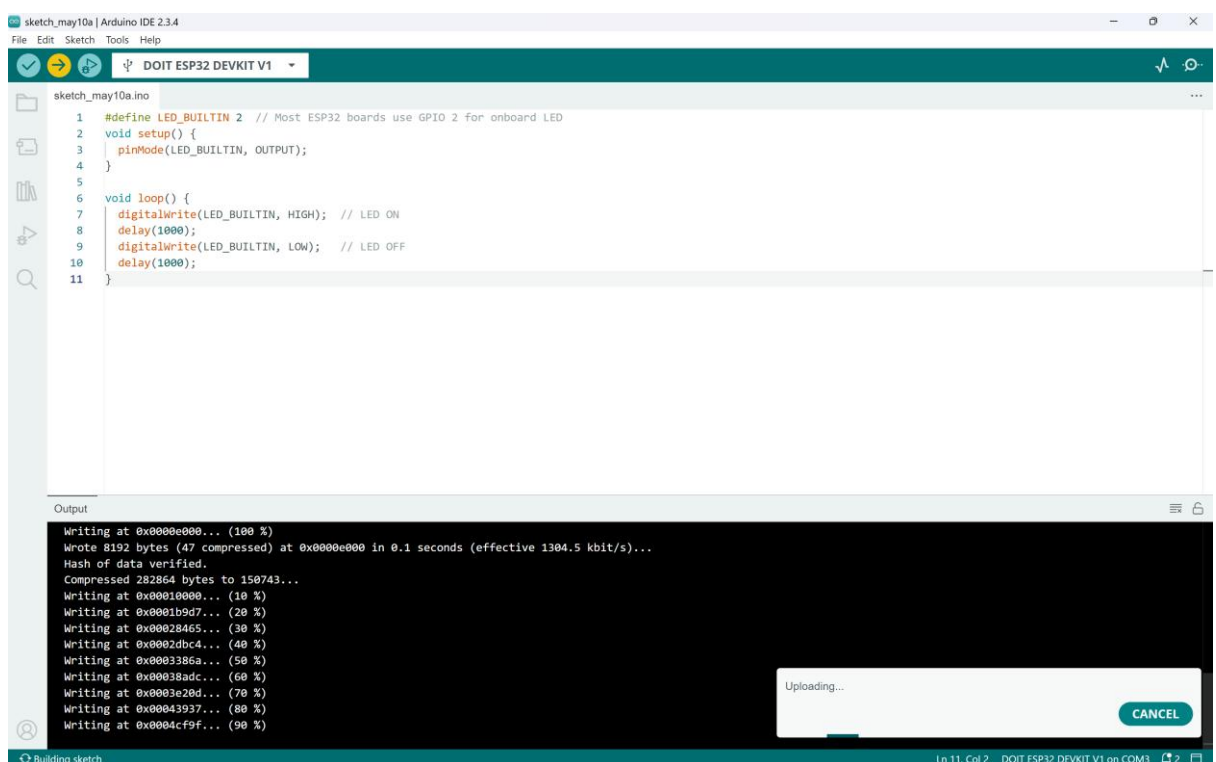
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // LED ON
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW); // LED OFF
  delay(1000);
}
```

1. Paste the code in the Arduino IDE.



```
sketch_may10a.ino
1 #define LED_BUILTIN 2 // Most ESP32 boards use GPIO 2 for onboard LED
2 void setup() {
3   pinMode(LED_BUILTIN, OUTPUT);
4 }
5
6 void loop() {
7   digitalWrite(LED_BUILTIN, HIGH); // LED ON
8   delay(1000);
9   digitalWrite(LED_BUILTIN, LOW); // LED OFF
10  delay(1000);
11 }
```

2. Click the Upload (→) button. Code should be uploaded successfully.



```
sketch_may10a.ino
1 #define LED_BUILTIN 2 // Most ESP32 boards use GPIO 2 for onboard LED
2 void setup() {
3   pinMode(LED_BUILTIN, OUTPUT);
4 }
5
6 void loop() {
7   digitalWrite(LED_BUILTIN, HIGH); // LED ON
8   delay(1000);
9   digitalWrite(LED_BUILTIN, LOW); // LED OFF
10  delay(1000);
11 }
```

```
Output
Writing at 0x00000000... (100 %)
Wrote 8192 bytes (47 compressed) at 0x00000000 in 0.1 seconds (effective 1304.5 kbit/s)...
Hash of data verified.
Compressed 282864 bytes to 150743...
Writing at 0x00010000... (10 %)
Writing at 0x0001b0d7... (20 %)
Writing at 0x00028465... (30 %)
Writing at 0x0002dbc4... (40 %)
Writing at 0x0003386a... (50 %)
Writing at 0x00038ad4... (60 %)
Writing at 0x0003e20d... (70 %)
Writing at 0x00043937... (80 %)
Writing at 0x0004cf9f... (90 %)
Writing at 0x00053d71... (100 %)
Wrote 282864 bytes (150743 compressed) at 0x00010000 in 2.8 seconds (effective 821.6 kbit/s)...
Hash of data verified.
Leaving...
Hard resetting via RTS pin...

Uploading... [Progress bar] [CANCEL]
```

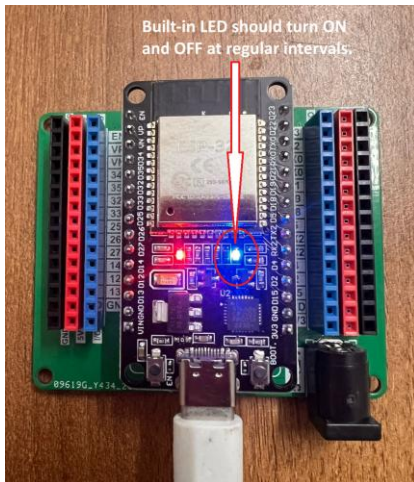
3. Or open the code “Project0_BlinkingLed.ino” directly from the CODE folder where you downloaded the user guide and click upload button.

If You See "Connecting..." Error:

- Hold the BOOT button on the ESP32 until it starts uploading.
- Release the button when uploading begins. Code should upload successfully.

Step 8: View Output

1. Once uploaded, the onboard LED should start blinking (1 second ON, 1 second OFF).



Common Issues & Fixes

Problem	Solution
"Failed to connect" error	Hold BOOT button while uploading
No COM port shown	Try another USB cable or driver (CP2102/CH340)
Board not blinking	Try GPIO 2, or check if your board uses GPIO 5
Upload stuck on "Connecting..."	Press and hold BOOT, or tap it once quickly

Ask Our Smart AI Mentor - QuadBot

Scan the QR code or click the picture link & ask our **Smart AI Mentor** for instant guidance!

Example Questions You Can Ask QuadBot:

- 🗨 My ESP32 board is not detected in Arduino IDE. What should I do?
- 🗨 Why is the COM port not showing for my ESP32 board?
- 🗨 Arduino IDE says “No board detected” for ESP32. How can I fix it?
- 🗨 Which board should I select for ESP32 Type-C board in Arduino IDE?
- 🗨 How do I install ESP32 board support in Arduino IDE?
- 🗨 How do I add the ESP32 board manager URL in Arduino IDE?
- 🗨 Why am I getting an upload error while uploading code to ESP32?
- 🗨 My ESP32 powers ON but code is not uploading. Why?
- 🗨 What should I do if Arduino IDE shows “Failed uploading” error?
- 🗨 Why does my ESP32 show “Timed out waiting for packet header”?
- 🗨 Should I press the BOOT button while uploading code to ESP32?
- 🗨 Why is my ESP32 not appearing in Device Manager?
- 🗨 How do I check if the USB driver is installed correctly?
- 🗨 Which USB driver is needed for my ESP32 board?
- 🗨 Which USB cable should I use for ESP32 Type-C board?
- 🗨 How do I connect ESP32 to Wi-Fi using Arduino IDE?
- 🗨 Why is my ESP32 Wi-Fi not connecting?
- 🗨 How do I use Bluetooth with ESP32?
- 🗨 Can ESP32 be used for IoT projects?
- 🗨 Can you guide me step-by-step to connect, program, and upload code to my ESP32 board?





WARRANTY – Register your product for warranty

Did you know your ESP32 Type-C board comes with a standard 3-month warranty?

Register your product on our website within 7 days of purchase to activate it.



Want to extend it to 6 months — absolutely free?

We're confident in the quality of our product, so we're offering you a chance to **double your warranty at no extra cost**. "Just provide a honest short feedback about the product and claim it."

Click the link below to complete your registration and enjoy extended coverage.

<https://quadstore.in/warranty/>

NOTE: Warranty is applicable only on ESP32 board. No other components are covered under warranty.

Project-1: Blinking an External LED with ESP32

In this project, you will connect an external LED to an ESP32 board and blink it at 1-second intervals using Arduino code. This helps you understand how to control external components with ESP32's digital GPIO pins.

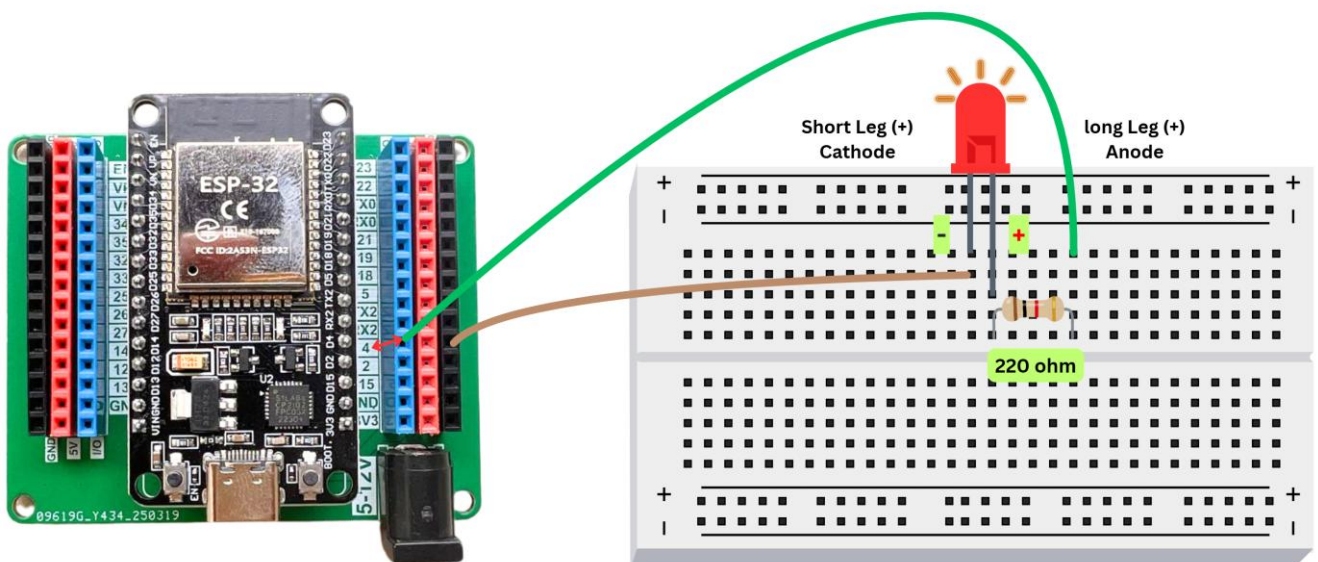
🔧 Components Required:

- ✓ ESP32 Dev Board × 1
- ✓ LED × 1
- ✓ Breadboard × 1
- ✓ Jumper Wires
- ✓ USB Cable × 1

⚙️ Circuit Connections:

Component	Connection Method	ESP32 Board
LED (Anode, Long Leg +)	Through 220Ω resistor	Pin 4
LED (Cathode, Short Leg -)	Direct Connection	GND

💡 Use 220Ω resistor in series with the LED to limit current and prevent burning it out.



ESP32 Arduino Code: Blinking external LED

```
#define LED_PIN 4 // GPIO 4 connected to external LED

void setup() {
  pinMode(LED_PIN, OUTPUT); // Set GPIO 4 as output
}

void loop() {
  digitalWrite(LED_PIN, HIGH); // Turn the LED ON
  delay(1000);                // Wait 1 second
  digitalWrite(LED_PIN, LOW);  // Turn the LED OFF
  delay(1000);                // Wait 1 second
}
```

Steps to Upload and Run the Code:

1. Connect your ESP32 to your computer using a USB cable.
2. Open Arduino IDE.
3. Go to Tools > Board, and select DOIT ESP32 DEVKIT V1 (or your board).
4. Go to Tools > Port, and select the correct COM port.
5. Paste the code into the IDE (**or**) open the program
 "Project1_BlinkingExternalLed.ino" directly from CODE folder.
6. Click Upload (→ icon).
7. The external LED on your breadboard should now blink on and off every second.

Expected Output:

The external LED will:

- Turn ON for 1 second
- Turn OFF for 1 second
- Repeat this cycle endlessly

 You should see a clear blinking LED on your breadboard controlled by GPIO 4.

Project-2: Using an RGB LED Module with ESP32

This project demonstrates how to control an RGB LED module using an ESP32. The RGB module has built-in resistors and separate input pins for Red, Green, and Blue channels. By sending PWM signals to the module's pins, you can mix colors and create effects like fading or cycling.

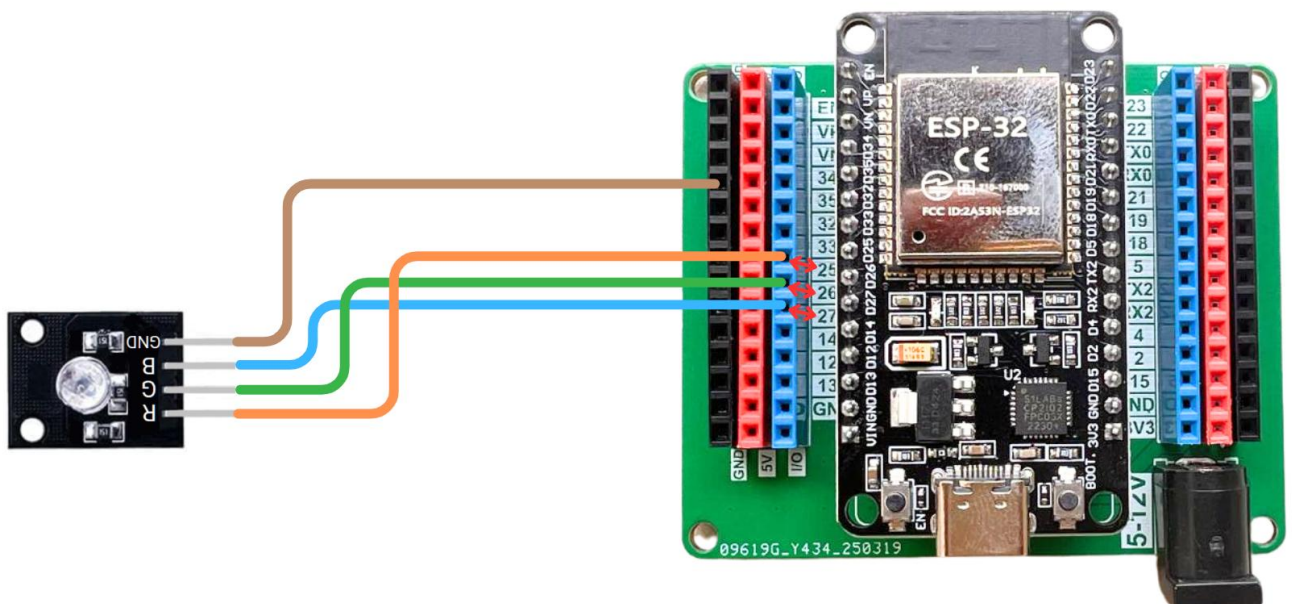
🔧 Components Required:

- ✓ ESP32 Dev Board × 1
- ✓ RGB LED Module × 1
- ✓ Breadboard × 1
- ✓ Jumper Wires
- ✓ USB Cable × 1

⚙️ Circuit Connections:

Component - RGB Led	Connection Method	ESP32 Board
Red (R)	Direct Connection	Pin 25
Green (G)	Direct Connection	Pin 26
Blue (B)	Direct Connection	Pin 27
GND	Direct Connection	GND

💡 RGB modules come with onboard resistors, so no external resistor is required.



ESP32 Arduino Code: Blinking RGB Colors on Module

```
// GPIO pins connected to RGB module
#define RED_PIN    25
#define GREEN_PIN  26
#define BLUE_PIN   27

void setup() {
  pinMode(RED_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);
}

void loop() {
  // Red
  analogWrite(RED_PIN, 255);
  analogWrite(GREEN_PIN, 0);
  analogWrite(BLUE_PIN, 0);
  delay(1000);

  // Green
  analogWrite(RED_PIN, 0);
  analogWrite(GREEN_PIN, 255);
  analogWrite(BLUE_PIN, 0);
  delay(1000);

  // Blue
  analogWrite(RED_PIN, 0);
  analogWrite(GREEN_PIN, 0);
  analogWrite(BLUE_PIN, 255);
  delay(1000);

  // White (all channels on)
  analogWrite(RED_PIN, 255);
  analogWrite(GREEN_PIN, 255);
  analogWrite(BLUE_PIN, 255);
  delay(1000);

  // Off
  analogWrite(RED_PIN, 0);
  analogWrite(GREEN_PIN, 0);
  analogWrite(BLUE_PIN, 0);
  delay(1000);
}
```

Steps to Upload and Run the Code:

1. Connect your ESP32 to your computer using a USB cable.
2. Open Arduino IDE.
3. Go to Tools > Board, and select DOIT ESP32 DEVKIT V1 (or your board).
4. Go to Tools > Port, and select the correct COM port.
5. Paste the code into the IDE (**or**) open the program “**Project2_RGBLed.ino**” directly from CODE folder.
6. Click Upload (→ icon).

Expected Output:

The RGB LED module will:

- Blink Red for 1 second
 - Blink Green for 1 second
 - Blink Blue for 1 second
 - Blink White for 1 second
 - Turn Off for 1 second
- ...and repeat.

 You can create other colors by mixing R, G, and B values (e.g., purple, yellow, cyan).

Project-3: Push Button Control of LED with ESP32

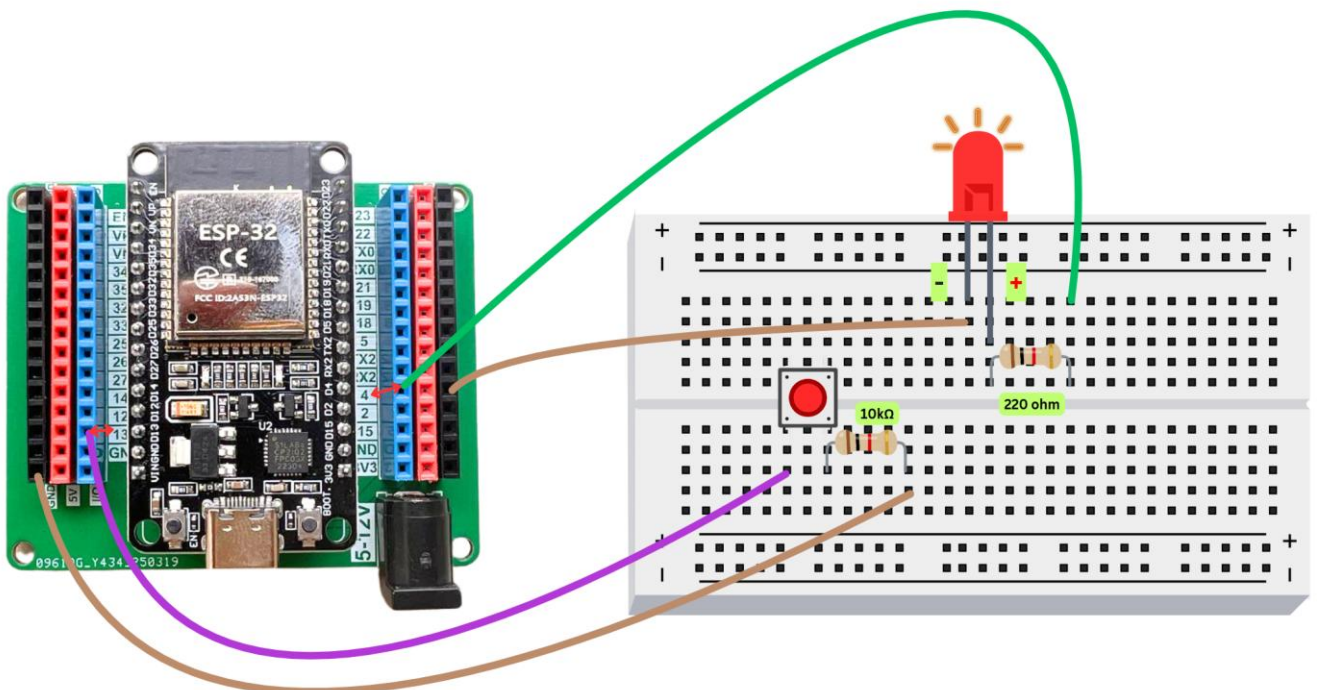
In this beginner-friendly project, you'll learn how to connect a push button to an ESP32 and use it to control a 5mm external LED. When the push button is pressed, the LED turns ON. When released, the LED turns OFF.

🔧 Components Required:

- ✓ ESP32 Dev Board × 1
- ✓ Breadboard × 1
- ✓ 5mm LED × 1
- ✓ Resistor (220Ω for LED) × 1
- ✓ Push Button × 1
- ✓ Resistor (10kΩ) × 1
- ✓ Jumper Wires × 6
- ✓ USB Cable × 1

⚙️ Circuit Connections:

Component	Connection Method	ESP32 Board
LED (Anode, Long Leg +)	Through 220Ω resistor	Pin 4
LED (Cathode, Short Leg -)	Direct Connection	GND
Push Button - One leg	Direct Connection	Pin 13
Push Button - Other leg	Through 10KΩ resistor	GND



ESP32 Arduino Code: LED ON When Button Pressed

```
#define LED_PIN 4
#define BUTTON_PIN 13

void setup() {
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP); // Use internal pull-up
  Serial.begin(9600);
}

void loop() {
  int buttonState = digitalRead(BUTTON_PIN);

  Serial.print("Button State: ");
  Serial.println(buttonState); // LOW when pressed, HIGH when
  released

  if (buttonState == LOW) {           // Button is pressed
    digitalWrite(LED_PIN, HIGH);      // Turn LED ON
  } else {
    digitalWrite(LED_PIN, LOW);       // Turn LED OFF
  }

  delay(100);
}
```

Steps to Run the Project:

1. Build the circuit on a breadboard as described above.
2. Connect your ESP32 to your computer.
3. Paste the code into the IDE (**or**) open the program “**Project3_PushButtonLED.ino**” directly from CODE folder.
4. Select the correct board (DOIT ESP32 DEVKIT V1) and COM port.
5. Click Upload.
6. Press the button and see the LED light up!

Expected Output:

- When button is not pressed → LED is OFF
- When button is pressed → LED is ON

Project-4: LED Brightness Control with Potentiometer and ESP32

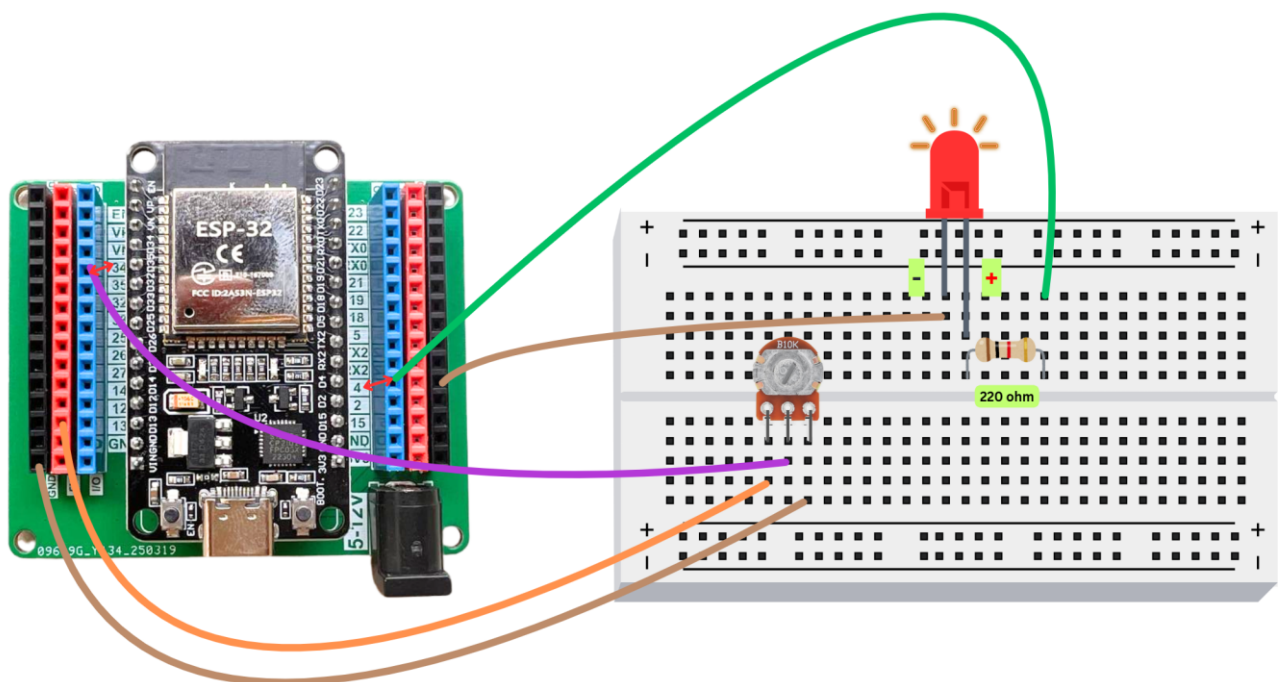
In this project, you'll use a **10k Ω potentiometer** to control the brightness of a **5mm LED** connected to the ESP32. The potentiometer's analog value (0–4095) will be mapped to a PWM value (0–255) and used to control the LED brightness.

🔌 Components Required:

- ✓ ESP32 Dev Board × 1
- ✓ Breadboard × 1
- ✓ 5mm LED × 1
- ✓ Resistor (220 Ω) × 1
- ✓ 10k Ω Potentiometer × 1
- ✓ Jumper Wires × 6
- ✓ USB Cable (ESP32) × 1

⚙️ Circuit Connections:

POTENTIOMETER		
Component	Connection Method	ESP32 Board
Potentiometer Left Pin	Direct Connection	5V
Potentiometer Middle Pin	Direct Connection	Pin 34
Potentiometer Right Right	Direct Connection	GND
LED		
Component	Connection Method	ESP32 Board
LED (Anode, Long Leg +)	Through 220Ω resistor	Pin 4
LED (Cathode, Short Leg -)	Direct Connection	GND



ESP32 Arduino Code: LED Brightness Control.

```
#define LED_PIN    4        // GPIO pin connected to LED
#define POT_PIN    34       // GPIO pin connected to potentiometer

const int PWM_CHANNEL = 0;
const int PWM_FREQ = 5000;
const int PWM_RESOLUTION = 8; // 8-bit resolution (0-255)

void setup() {
    Serial.begin(115200);

    // NEW ESP32 3.x PWM API
    ledcAttach(LED_PIN, PWM_FREQ, PWM_RESOLUTION);
}

void loop() {
    int potValue = analogRead(POT_PIN);           // 0-4095
    int brightness = map(potValue, 0, 4095, 255, 0); // reverse
    mapping

    ledcWrite(LED_PIN, brightness);               // NEW API

    Serial.print("Pot: ");
    Serial.print(potValue);
    Serial.print(" → Brightness: ");
    Serial.println(brightness);

    delay(10);
}
```

Steps to Upload and Run:

1. Wire the potentiometer and LED as shown.
2. Open Arduino IDE and paste the code **or** open the program
“**Project4_BrightnessControlLED.ino**” from CODE folder.
3. Select your board (**DOIT ESP32 DEVKIT V1**) and correct **COM port**.
4. Click **Upload**.
5. Open **Serial Monitor** (baud: 115200).
6. Rotate the potentiometer and observe LED brightness change in real-time.

✓ **Expected Output:**

- As you rotate the **potentiometer**, the LED **brightness increases or decreases** smoothly.
- Serial Monitor shows raw ADC value and mapped brightness (0–255).

Project-5: Using an Active Buzzer with ESP32

This project demonstrates how to connect and control an **active buzzer module** using an ESP32. An active buzzer has an internal oscillator and only needs a HIGH or LOW signal to turn ON or OFF. We'll use the ESP32 to turn the buzzer ON and OFF at 1-second intervals.

🔧 Components Required:

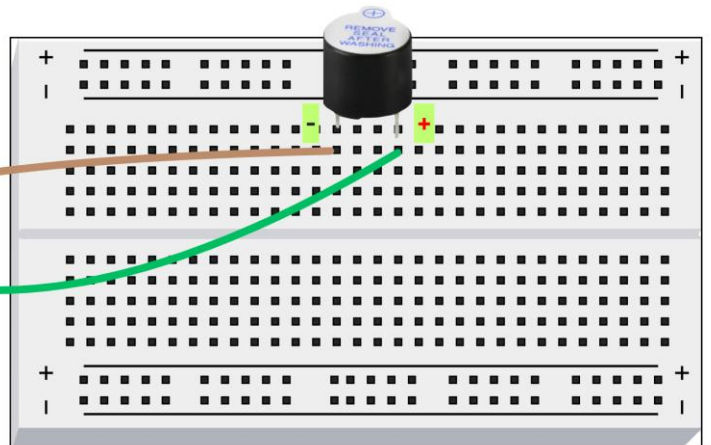
- ✓ ESP32 Dev Board × 1
- ✓ Active Buzzer Module (sticker on top) × 1
- ✓ Breadboard × 1
- ✓ Jumper Wires × 3
- ✓ USB Cable × 1

⚙️ Circuit Connections:

Component	Connection Method	ESP32 Board
Active Buzzer VCC (+)	Direct Connection	Pin 4
Active Buzzer GND (-)	Direct Connection	GND



Active Buzzer: Buzzer with sticker (remove it and use)



Arduino Code to Beep the Active Buzzer:

```
#define BUZZER_PIN 4 // GPIO pin connected to buzzer

void setup() {
  pinMode(BUZZER_PIN, OUTPUT); // Set buzzer pin as output
}

void loop() {
  digitalWrite(BUZZER_PIN, HIGH); // Turn buzzer ON
  delay(1000);                     // Wait 1 second
  digitalWrite(BUZZER_PIN, LOW);  // Turn buzzer OFF
  delay(1000);                     // Wait 1 second
}
```

Steps to Upload and Run the Project:

1. Wire the buzzer as shown above.
2. Open **Arduino IDE**.
3. Select **DOIT ESP32 DEVKIT V1** as your board.
4. Select the correct **COM port**.
5. Paste the code into the IDE (or) open the program “**Project5_ActiveBuzzer.ino**” directly from CODE folder.
6. click **Upload**.
7. Listen for the **buzzer beeping every second**.

Expected Output:

The active buzzer will:

- **Beep ON for 1 second**
- **Beep OFF for 1 second**
- Repeats in a loop

 You will hear a regular beeping pattern.

Project-6: Play Tones on a Passive Buzzer with ESP32

This project demonstrates how to use a **passive buzzer module** with an ESP32 to generate different tones. Unlike an active buzzer (which just beeps ON/OFF), a passive buzzer can produce various sounds and frequencies using PWM, making it ideal for playing simple melodies or alarms.

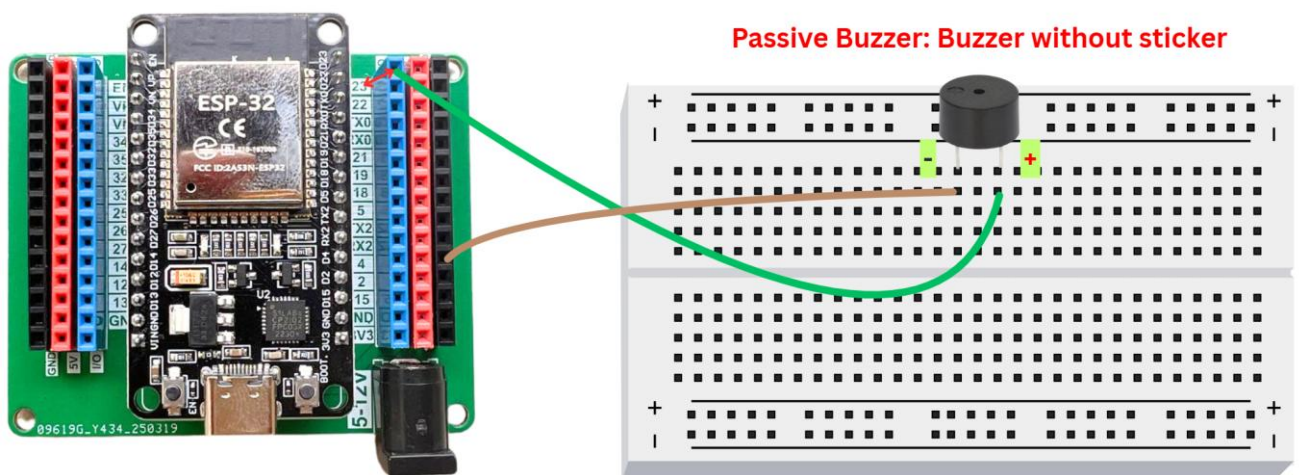
🔧 Components Required:

- ✓ ESP32 Dev Board × 1
- ✓ Passive Buzzer Module × 1
- ✓ Breadboard × 1
- ✓ Jumper Wires × 3
- ✓ USB Cable × 1

⚙️ Circuit Connections:

Component	Connection Method	ESP32 Board
Passive Buzzer VCC (+)	Direct Connection	Pin 23
Passive Buzzer GND (-)	Direct Connection	GND

💡 Passive buzzers must be controlled with a **PWM signal** to produce sound (we'll use `ledcWriteTone()` for ESP32).



Arduino Code to Play a Simple Musical Tone:

```
#define BUZZER_PIN      23
#define BUZZER_CHANNEL 0

// Notes in frequency (Hz)
int melody[] = {
  262, 294, 330, 349, 392, 440, 494, 523 // C4, D4, E4, F4, G4, A4,
  B4, C5
};

// Duration of each note (in ms)
int noteDurations[] = {
  400, 400, 400, 400, 400, 400, 400, 800
};

void setup() {
  ledcSetup(BUZZER_CHANNEL, 2000, 8);          // Init PWM (frequency
  here is placeholder)
  ledcAttachPin(BUZZER_PIN, BUZZER_CHANNEL);
}

void loop() {
  for (int i = 0; i < 8; i++) {
    int note = melody[i];
    int duration = noteDurations[i];

    ledcWriteTone(BUZZER_CHANNEL, note);        // Play the note
    delay(duration);                            // Hold it for note
    duration
    ledcWriteTone(BUZZER_CHANNEL, 0);           // Pause between notes
    delay(50);                                  // Short pause between
    notes
  }

  delay(2000); // Wait before repeating the melody
}
```

Steps to Upload and Run the Project:

1. Connect the passive buzzer to **GPIO 23**, **GND**, and **3.3V**.
2. Open **Arduino IDE**.
3. Select **DOIT ESP32 DEVKIT V1** as your board.
4. Choose the correct **COM port**.

5. Paste the code into the IDE (**or**) open the program **Project6_PassiveBuzzer.ino** directly from CODE folder.
6. The buzzer will produce a **1kHz tone for 1 second**, then be silent for 1 second, repeatedly.

 Expected Output:

This plays a simple **ascending scale** (C4 to C5). You can easily customize it with different notes or durations.

 You can change the tone frequency (e.g., 500, 2000) in `ledcWriteTone()` to produce different sounds.

Project-7: Light-Activated Red LED Using ESP32 and LDR Module

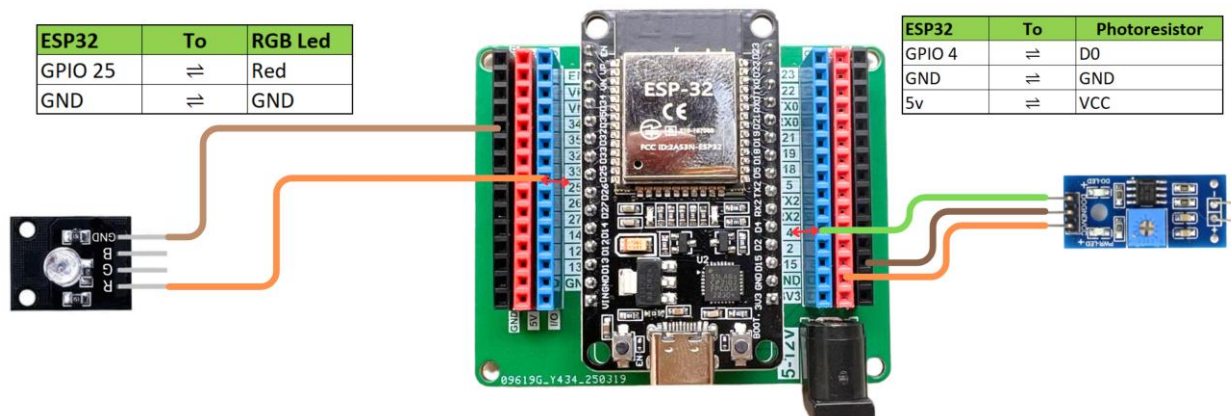
This project uses an **LDR (photoresistor) module** to detect ambient light levels. When it gets dark (light intensity drops below a threshold), the **ESP32** turns ON the **red light** of an **RGB LED module** to act as an automatic night indicator.

Components Required:

- ✓ ESP32 Dev Board × 1
- ✓ Photoresistor (LDR) Module × 1
- ✓ RGB LED Module (Common Cathode) × 1
- ✓ Breadboard × 1
- ✓ Jumper Wires × 6

Circuit Diagram & Connections:

RGB Led		
Component	Connection Method	ESP32 Board
RED (R)	Direct Connection	Pin 25
GND	Direct Connection	GND
GREEN (G) , BLUE (B)	NOT USED	
PHOTORESISTOR		
Component	Connection Method	ESP32 Board
Photoresistor (DO)	Direct Connection	Pin 4
Photoresistor GND	Direct Connection	GND
Photoresistor VCC	Direct Connection	5v



Arduino Code: Light Detection to Control Red LED

```
#define LDR_PIN 4      // Analog pin connected to LDR module output
#define RED_PIN 25     // PWM pin connected to Red pin of RGB LED

const int PWM_CHANNEL = 0;
const int PWM_FREQ = 5000;
const int PWM_RESOLUTION = 8; // 8-bit resolution: 0-255

int threshold = 2000; // Light threshold (0-4095). Below = dark.

void setup() {
    Serial.begin(115200);

    // Configure PWM for RED LED
    ledcSetup(PWM_CHANNEL, PWM_FREQ, PWM_RESOLUTION);
    ledcAttachPin(RED_PIN, PWM_CHANNEL);
}

void loop() {
    int lightLevel = analogRead(LDR_PIN); // Read ambient light (0-4095)

    Serial.print("Light Level: ");
    Serial.println(lightLevel);

    if (lightLevel < threshold) {
        // It's bright → turn OFF red LED
        ledcWrite(PWM_CHANNEL, 0);
    } else {
        // It's dark → turn ON red LED
        ledcWrite(PWM_CHANNEL, 255);
    }

    delay(500); // Check every 0.5 seconds
}
```

Steps to Upload and Run:

1. Connect LDR and RGB modules as shown.
8. Open Arduino IDE, Paste the code into the IDE (**or**) open the program **Project7_LDR.ino** directly from **CODE** folder.
2. Select **DOIT ESP32 DEVKIT V1** and your **COM port**.
3. Upload the code.
4. Cover the LDR — the **red LED turns ON**.
Expose it to light — the **red LED turns OFF**.

Expected Output:

- In **bright light**: red LED is **OFF**.
- In **darkness**: red LED is **ON**.
- Serial Monitor prints current **light levels** continuously.

Project-8: Obstacle Detection with IR Sensor and RGB LED Using ESP32

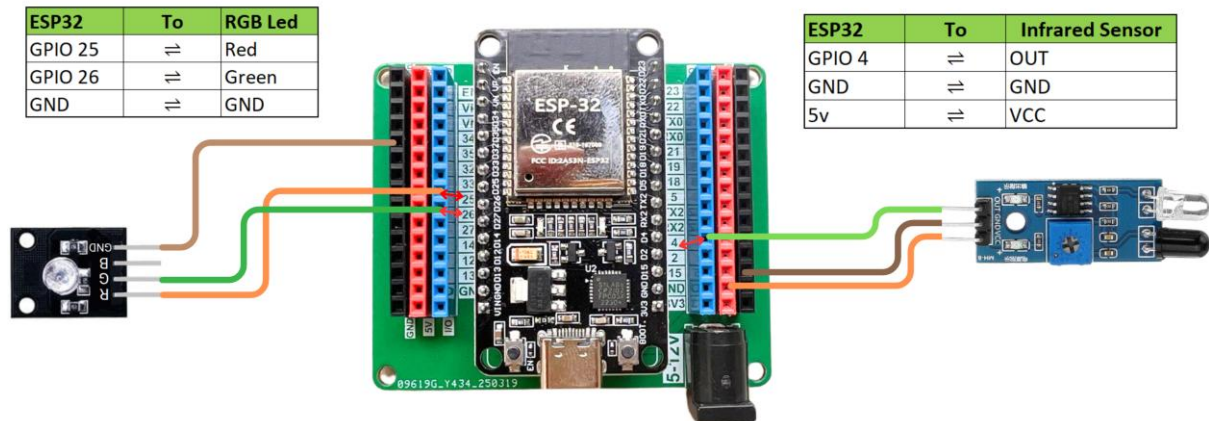
This project demonstrates how to detect obstacles using an **IR sensor module** and visually indicate the result using an **RGB LED module**. The ESP32 reads the digital signal from the IR sensor and turns ON the **red LED** if an obstacle is nearby, or the **green LED** if the path is clear.

Components Required:

- ✓ ESP32 Dev Board × 1
- ✓ Infrared (IR) Sensor Module × 1
- ✓ RGB LED Module (Common Cathode) × 1
- ✓ Breadboard × 1
- ✓ Jumper Wires × 6
- ✓ USB Cable × 1

Circuit Diagram & Connections:

RGB Led		
Component	Connection Method	ESP32 Board
RED (R)	Direct Connection	Pin 25
GREEN (G)	Direct Connection	Pin 26
BLUE (B)	NOT USED	
GND	Direct Connection	GND
IR - Infrared Sensor		
Component	Connection Method	ESP32 Board
IR Sensor (OUT)	Direct Connection	Pin 4
IR Sensor GND	Direct Connection	GND
IR Sensor VCC	Direct Connection	5v



Arduino Code: IR Obstacle Detection with RGB LED

```
#define IR_SENSOR_PIN 4    // Digital input from IR sensor
#define RED_PIN        25  // Red channel of RGB LED
#define GREEN_PIN      26  // Green channel of RGB LED

// PWM setup
const int PWM_FREQ = 5000;
const int PWM_RES  = 8;   // 8-bit (0-255)

void setup() {
    Serial.begin(115200);

    pinMode(IR_SENSOR_PIN, INPUT);

    // ESP32 core 3.x PWM attach
    ledcAttach(RED_PIN, PWM_FREQ, PWM_RES);
    ledcAttach(GREEN_PIN, PWM_FREQ, PWM_RES);
}

void loop() {
    int irValue = digitalRead(IR_SENSOR_PIN);

    Serial.print("IR Sensor: ");
    Serial.println(irValue); // LOW = obstacle detected

    if (irValue == LOW) {
        // Obstacle detected → Red ON, Green OFF
        ledcWrite(RED_PIN, 255);
        ledcWrite(GREEN_PIN, 0);
    } else {
        // No obstacle → Green ON, Red OFF
        ledcWrite(RED_PIN, 0);
        ledcWrite(GREEN_PIN, 255);
    }

    delay(100);
}
```

▶ Steps to Upload and Run:

1. Wire the IR sensor and RGB LED module as shown.
2. Paste the code into the IDE (**or**) open the program **Project8_InfraredSensor.ino** directly from CODE folder.
3. Select your ESP32 board and COM port.
4. Click **Upload**.
5. Place your hand or object in front of the IR sensor:
 - **Obstacle near → Red LED ON**
 - **No obstacle → Green LED ON**

✔ Expected Output:

Condition	LED Status
Obstacle detected (IR LOW)	Red ON, Green OFF
No obstacle (IR HIGH)	Green ON, Red OFF

- The ESP32 will automatically switch LED color based on sensor readings.

Note: You can also add a buzzer to alert when an obstacle is detected.

Project-9: Displaying Text on OLED with ESP32

This project demonstrates how to interface a **0.96-inch OLED display** (based on the **SSD1306 driver**) with an **ESP32** via I2C. The text **“Quad Robotics”** will be shown on the screen. This is useful for displaying sensor readings, menus, or status messages in embedded projects.

🔧 Components Required:

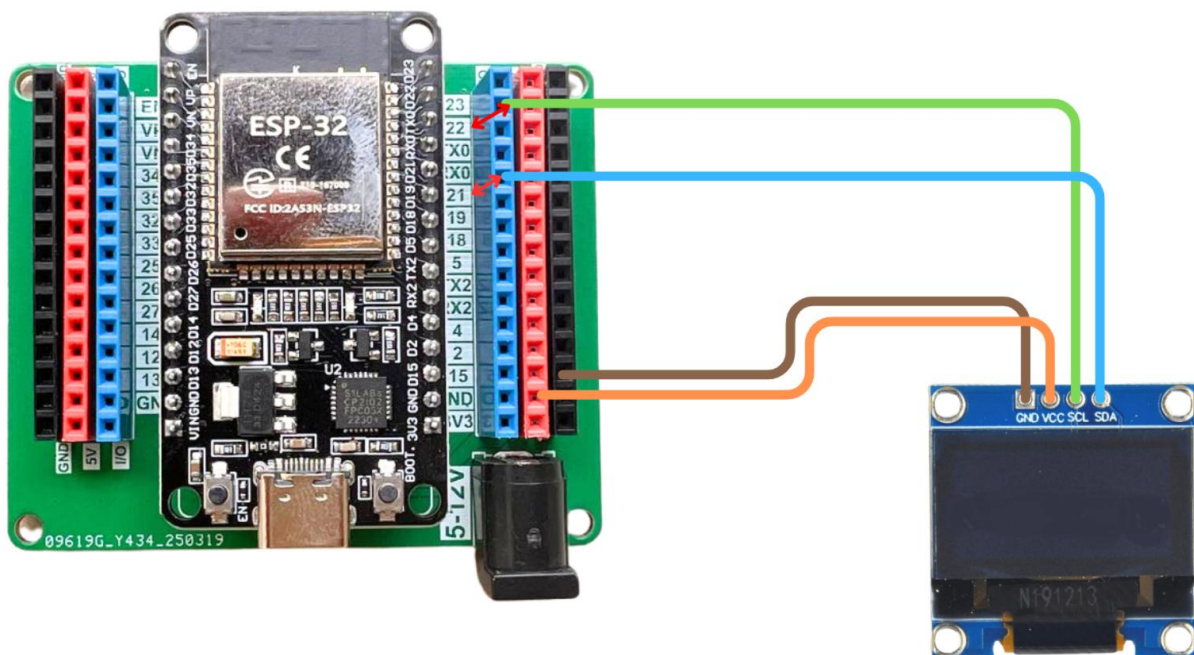
- ✓ ESP32 Dev Board × 1
- ✓ 0.96" OLED Display (SSD1306, I2C) × 1
- ✓ Breadboard × 1
- ✓ Jumper Wires × 4
- ✓ USB Cable × 1

🌿 Circuit Diagram & Connections (I2C OLED):

OLED Pin	Connection Method	ESP32 Board
VCC	Direct Connection	5v
GND	Direct Connection	GND
SCL	Direct Connection	Pin 22
SDA	Direct Connection	Pin 21

💡 These are the default I2C pins for most ESP32 Dev boards.

You can change them in the code if needed.



Library Installation:

Before uploading the code:

1. Go to **Sketch > Include Library > Manage Libraries**.
2. Search for and install:
 - **Adafruit SSD1306**
 - **Adafruit GFX Library**

Arduino Code to Display "Quad Robotics":

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

// Declaration for SSD1306 display connected via I2C
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C // Default I2C address for most OLEDs

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

void setup() {
  // Initialize Serial (optional)
  Serial.begin(115200);

  // Initialize OLED
  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    while (true); // Stop if failed
  }

  display.clearDisplay();

  // Set text properties
  display.setTextSize(2); // Text size (1-3 recommended)
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 20); // x = 0, y = 20

  display.print("Quad");
  display.setCursor(0, 45);
  display.print("Robotics");

  display.display(); // Push to screen
}

void loop() {
  // Nothing here - display shows static text
}
```

Steps to Upload and Run the Project:

1. Connect the OLED display as described above.
2. Install the required libraries.
3. Paste the code into the IDE (**or**) open the program **Project9_OLED.ino** directly from CODE folder.
4. Select your ESP32 board and COM port.
5. Click **Upload**.
6. The OLED should display: Quad Robotics

Expected Output:

- The OLED screen will show “**Quad Robotics**” in two lines.
- Text will appear in **white color** (monochrome) on the black background.

Project-10: Read Temperature and Humidity from DHT11 Sensor Using ESP32

This project shows how to read **temperature and humidity** data using the **DHT11 sensor module** and an ESP32. The data will be printed on the **Serial Monitor**, and you can also take actions like turning on an LED if temperature/humidity crosses a threshold.

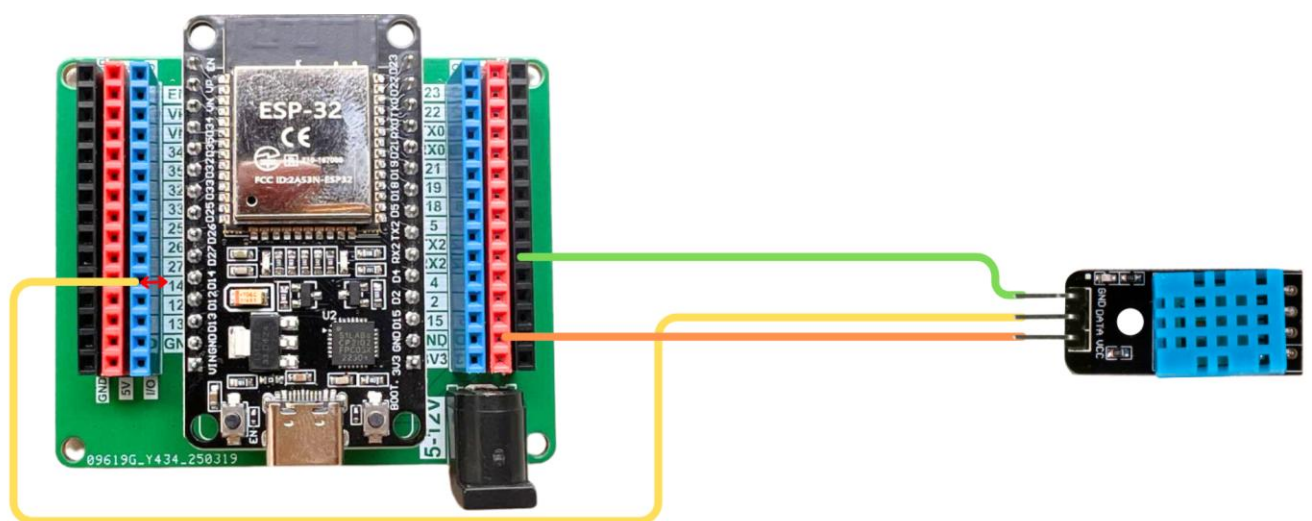
🔧 Components Required:

- ✓ ESP32 Dev Board × 1
- ✓ DHT11 Sensor Module × 1
- ✓ Breadboard × 1
- ✓ Jumper Wires × 3
- ✓ USB Cable × 1

⚙️ Circuit Diagram & Connections:

DHT 11 Sensor	Connection Method	ESP32 Board
DATA	Direct Connection	Pin 14
GND	Direct Connection	GND
VCC	Direct Connection	5v

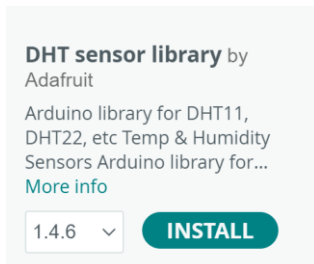
- ◆ Many DHT11 modules include a pull-up resistor, so **no external resistor** is needed.



Library Installation:

Before uploading the code:

1. Go to **Sketch > Include Library > Manage Libraries**.
2. Search for **DHT sensor library by Adafruit** and install it.



3. It may also ask you to install **Adafruit Unified Sensor** — install that too.

Arduino Code for Reading DHT11 with ESP32:

```
#include <DHT.h>

#define DHTPIN 14      // GPIO connected to DHT11 OUT pin
#define DHTTYPE DHT11 // DHT11 sensor

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(115200);
  dht.begin();
  Serial.println("DHT11 Sensor Initialization Complete");
}

void loop() {
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature(); // Celsius

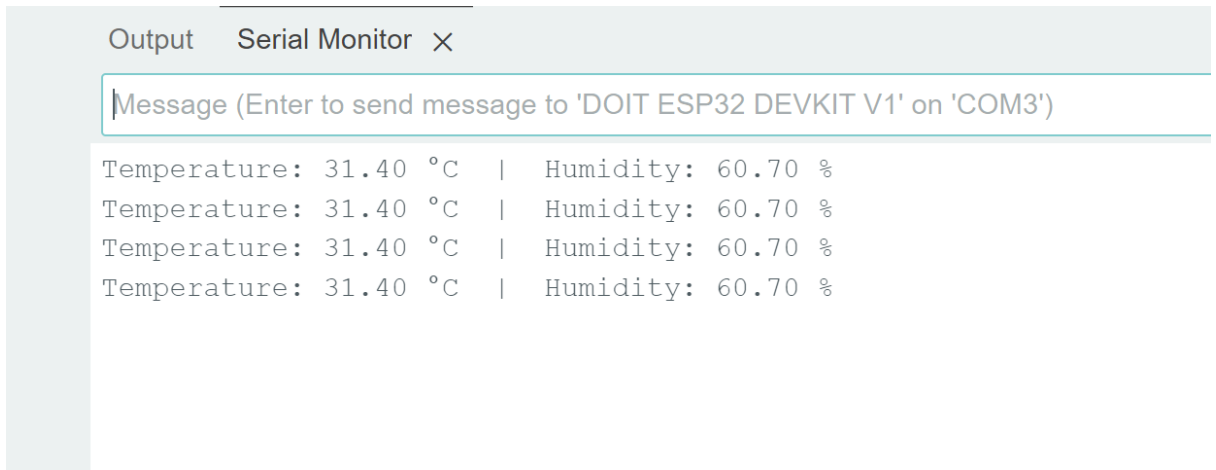
  // Check if reads failed
  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT11 sensor!");
    delay(2000);
    return;
  }

  // Print readings
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C | Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");

  delay(2000); // Wait 2 seconds before next reading
}
```

▶ Steps to Run the Project:

1. Wire the DHT11 sensor to ESP32 as described.
2. Install the required libraries.
3. Paste the code into the IDE (**or**) open the program **Project10_DHT11.ino** directly from CODE folder.
4. Select your board (**DOIT ESP32 DEVKIT V1**) and COM port.
5. Click **Upload**.
6. Open the **Serial Monitor (115200 baud)**.
7. Observe temperature and humidity readings every 2 seconds.



The screenshot shows the Arduino IDE Serial Monitor window. The title bar includes 'Output', 'Serial Monitor', and a close button. Below the title bar is a text input field with the placeholder text 'Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM3')'. The main area of the window displays four lines of output data, each showing temperature and humidity readings separated by a vertical bar.

```
Temperature: 31.40 °C | Humidity: 60.70 %  
Temperature: 31.40 °C | Humidity: 60.70 %  
Temperature: 31.40 °C | Humidity: 60.70 %  
Temperature: 31.40 °C | Humidity: 60.70 %
```

✅ Expected Output:

DHT11 Sensor should display your current temperature and humidity.

Temperature: 26.00 °C | Humidity: 55.00 %

Temperature: 26.10 °C | Humidity: 54.80 %

🧩 Optional Expansion Ideas:

- Trigger an **LED or buzzer** if temperature > 30°C
- Display data on an **OLED or LCD** screen
- Send readings to a web dashboard via Wi-Fi (IoT)

Project-11: Web Server-Based Inbuilt LED Toggle Using ESP32

This project turns the **ESP32** into a **Wi-Fi web server**, accessible from any smartphone or PC on the same network. The web page shows a **toggle switch** that turns the ESP32's **inbuilt LED ON or OFF** in real-time using **AJAX**, ensuring **fast response and no page reloads**.

Components Required:

- ✓ ESP32 Dev Board × 1
- ✓ USB Cable × 1
- ✓ Wi-Fi Network (You need to know your **WiFi Network Username & Password**)
- ✓ Smartphone / PC × 1

💡 No external components are needed — uses the **inbuilt LED (GPIO 2)**.

Wi-Fi Setup:

IMPORTANT: Before uploading the code, replace:

```
const char* ssid = "YOUR_SSID";
```

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

with your actual **Wi-Fi credentials**. "YOUR_SSID" should be replaced with your Wifi Username and "YOUR_PASSWORD" should be replaced with your Wifi password.

NOTE: both your SSID and Password should be within Double-Quotes “ ”.

ESP32 Arduino Code:

```
#include <WiFi.h>

const char* ssid = " YOUR_SSID";          // Replace with your WiFi Username
const char* password = " YOUR_PASSWORD";  // Replace with your WiFi password

WiFiServer server(80);
bool ledState = false;

void setup() {
  Serial.begin(115200);
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);

  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(300);
  }
```

```

    Serial.print(".");
}

Serial.println("\nConnected. IP:");
Serial.println(WiFi.localIP());

server.begin();
}

void loop() {
    WiFiClient client = server.available();
    if (!client) return;

    String request = "";
    while (client.connected()) {
        if (client.available()) {
            char c = client.read();
            request += c;

            if (c == '\n') {
                // Handle toggle requests
                if (request.indexOf("GET /toggle?state=1") != -1) {
                    ledState = true;
                    digitalWrite(LED_BUILTIN, HIGH);
                }
                if (request.indexOf("GET /toggle?state=0") != -1) {
                    ledState = false;
                    digitalWrite(LED_BUILTIN, LOW);
                }

                // Send HTML page
                if (request.indexOf("GET /toggle") == -1) {
                    client.println("HTTP/1.1 200 OK");
                    client.println("Content-Type: text/html");
                    client.println("Connection: close");
                    client.println();
                    client.println("<!DOCTYPE html><html><head><meta name='viewport'");
content='width=device-width, initial-scale=1'>");
                    client.println("<title>ESP32 Toggle LED</title>");
                    client.println("<style>");
                    client.println("body{font-family:sans-serif;text-align:center;padding-");
top:40px;}");
                    client.println("h1{margin-bottom:5px;} p{margin-top:0;color:gray;}");
                    client.println(".switch{position:relative;display:inline-");
block;width:60px;height:34px;}");
                    client.println(".switch input{display:none;}");
                    client.println(".slider{position:absolute;cursor:pointer;top:0;left:0;ri");
ght:0;bottom:0;background-color:#ccc;transition:.4s;}");
                    client.println(".slider:before{position:absolute;content:'';height:26px;");
width:26px;left:4px;bottom:4px;background-color:white;transition:.4s;}");
                    client.println("input:checked + .slider{background-color:#2196F3;}");
                    client.println("input:checked +");
.slider:before{transform:translateX(26px);}");
                    client.println(".slider.round{border-radius:34px;}");
                    client.println(".slider.round:before{border-radius:50%;}");
                    client.println("</style>");
                    client.println("</head><body>");

                    client.println("<h1>QUAD ROBOTICS</h1>");

```

```

        client.println("<p>A unit of Quad Store</p>");
        client.println("<h2>Smart Web Controlled LED</h2>");

        client.println("<div style='display:inline-block;'>");
        client.print("<label class='switch'><input type='checkbox' "
onchange='toggleLED(this)' ">");
        if (ledState) client.print("checked");
        client.println("><span class='slider round'></span></label>");
        client.println("<div style='display:flex; justify-content:space-between;
margin-top:5px;'>");
        client.println("<span style='width:60px; text-align:left;'>OFF</span>");
        client.println("<span style='width:60px; text-align:right;'>ON</span>");
        client.println("</div></div>");

        client.println("<script>function toggleLED(element){");
        client.println("var xhr=new XMLHttpRequest();");
        client.println("xhr.open('GET','/toggle?state='+element.checked?1:0),tr
ue);");
        client.println("xhr.send();");
        client.println("}</script>");
        client.println("</body></html>");
    } else {
        client.println("HTTP/1.1 200 OK");
        client.println("Content-Type: text/plain");
        client.println("Connection: close");
        client.println();
        client.println("OK");
    }
    break;
}
}
}

delay(1);
client.stop();
}

```

▶ Steps to Run the Project:

1. Open Arduino IDE and connect your ESP32.
2. Paste the code into the IDE (**or**) open the program **Project11_Webserver_LedToggle.ino** directly from CODE folder.
3. Set your Wi-Fi credentials in the CODE.
4. Select:
 - **Board:** DOIT ESP32 DEVKIT V1
 - **Port:** COMx (as shown)
5. Click **Upload**.

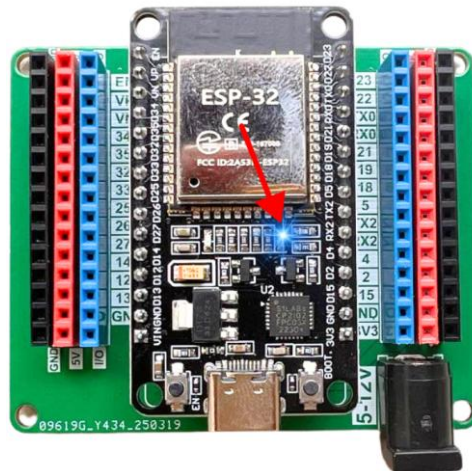
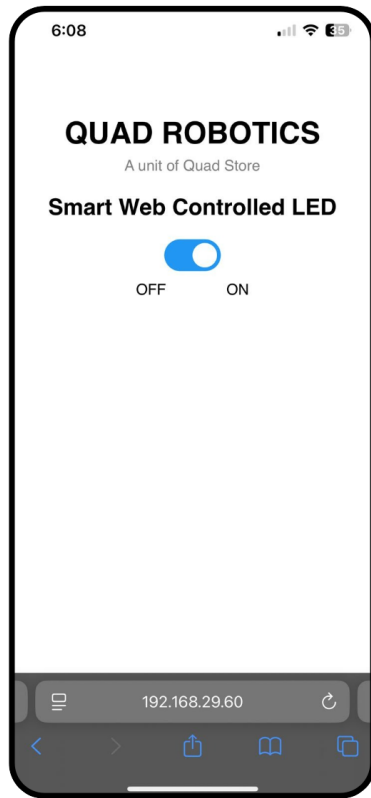
6. Open **Serial Monitor** at 115200 baud and note the **IP address**. **If it does not show the IP address then press the RESET Button in the ESP32 once.**



7. Enter the IP address in a **mobile/desktop browser** on the **same Wi-Fi**. **NOTE: Your mobile and ESP32 should be connected to same WIFI network.**



8. Toggle the switch to control the **inbuilt LED** instantly.



✓ **Expected Output:**

Toggle Switch LED Status

Switched **ON** LED turns **ON**

Switched **OFF** LED turns **OFF**

Project-12: Dual Control of ESP32 Output via Web Server & Push Button

✓ Objective

Build a system using **ESP32** that allows:

- Controlling an output (LED or relay) through a **web server**
- Toggling the same output via a **physical push button**
- **Real-time status updates** on the web interface, regardless of the control method used

📦 Components Required

- ✓ ESP32 Dev Board × 1
- ✓ LED × 1
- ✓ 220Ω Resistor × 1
- ✓ Push Button × 1
- ✓ 10kΩ Resistor × 1 (optional)
- ✓ Breadboard × 1
- ✓ Jumper Wires × As needed
- ✓ Micro USB Cable × 1

🔌 Circuit Diagram

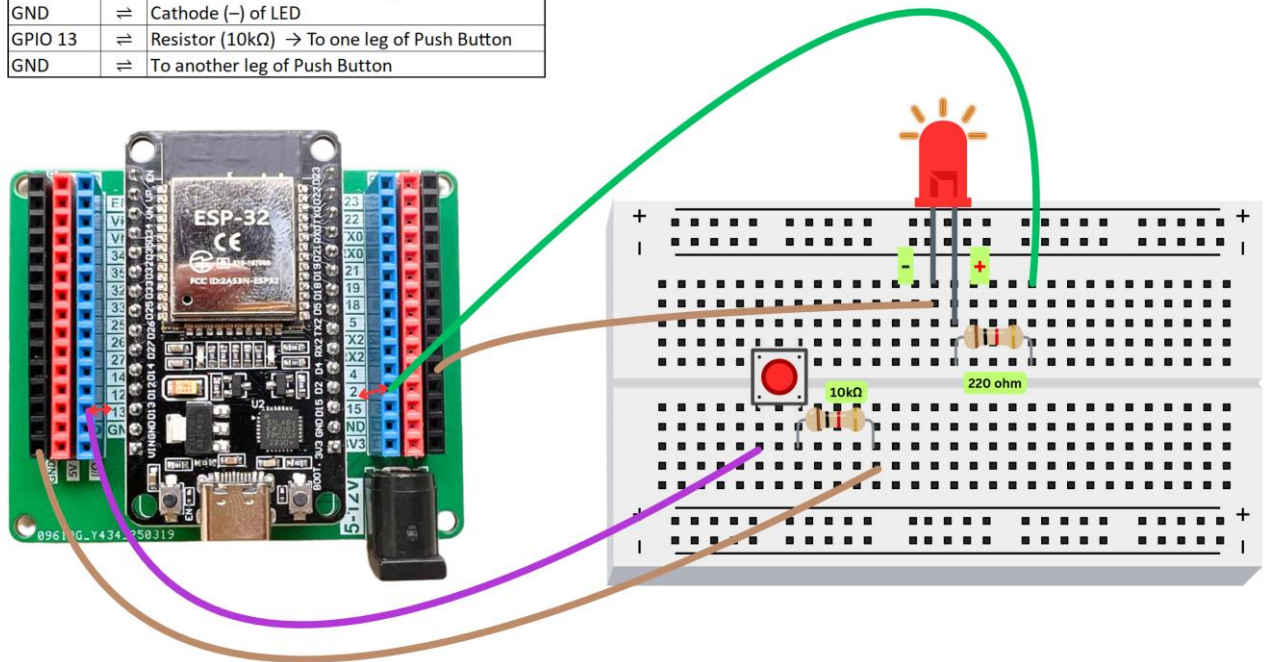
1. LED:

- LED +ve → GPIO 2 via 220Ω resistor
- LED -ve → GND

2. Button:

- One side → GPIO 13
- Other side → GND
(Use `INPUT_PULLUP` in code, no external resistor needed)

ESP32	To	Breadboard
GPIO 2	⇒	Resistor (220ohm) → Anode (+) of LED
GND	⇒	Cathode (−) of LED
GPIO 13	⇒	Resistor (10kΩ) → To one leg of Push Button
GND	⇒	To another leg of Push Button



Web Server Behavior

- Hosts a webpage showing current output state (ON/OFF)
- Button on page toggles output
- Physical push button toggles same output
- State is synced using AJAX polling every second

Wi-Fi Setup:

IMPORTANT: Before uploading the code, replace:

```
const char* ssid = "YOUR_SSID";
```

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

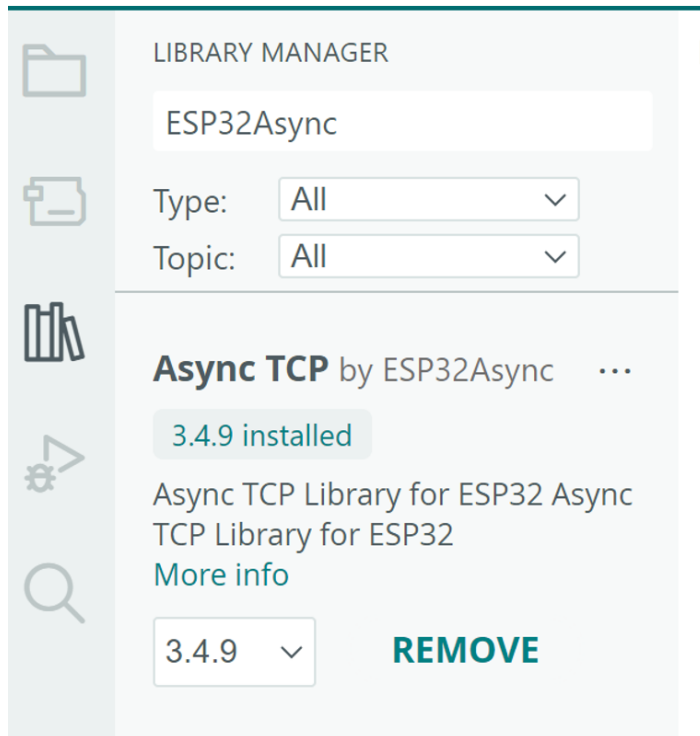
with your actual **Wi-Fi credentials**.

NOTE: both your SSID and Password should be within Double-Quotes “ ”.

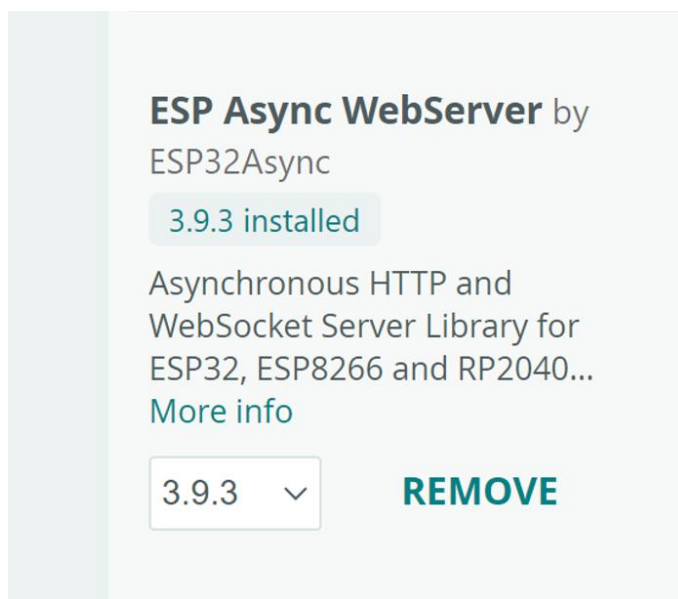
 Prerequisites: (MUST INSTALL)

Install the following Two libraries via Arduino Library Manager

- **AsyncTCP** by ESP32Async:



- **ESPAsyncWebServer** by ESP32Async:



✓ ESP32 Arduino Code:

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

// ===== WiFi Credentials =====
const char* ssid      = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

// ===== GPIO Pins =====
static const uint8_t OUTPUT_PIN = 2;    // LED / Relay
static const uint8_t BUTTON_PIN = 13;   // Push button (INPUT_PULLUP)

// ===== Globals =====
AsyncWebServer server(80);

bool outputState = false;
bool lastButtonState = HIGH;

unsigned long lastDebounceTime = 0;
const unsigned long debounceDelay = 50;

// ===== Web Page =====
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE html>
<html>
<head>
  <title>QUAD ROBOTICS</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      background-color: #f4f4f4;
      padding: 30px;
    }
    h1 { color: #2c3e50; margin-bottom: 0; }
    h4 { color: #7f8c8d; margin-top: 5px; font-weight: normal; }
    .state { font-size: 2em; margin: 30px 0 20px; }
    .toggle-button {
      font-size: 1.5em;
      padding: 12px 25px;
      border: none;
      border-radius: 8px;
      cursor: pointer;
      color: white;
    }
    .on { background-color: #27ae60; }
    .off { background-color: #c0392b; }
  </style>
</head>
<body>
```

```

<h1>QUAD ROBOTICS</h1>
<h4>A unit of Quad Store</h4>

<div class="state">
  Status: <span id="outputState">...</span>
</div>

<button id="toggleBtn" class="toggle-button" onclick="toggleOutput()">
  Loading...
</button>

<script>
  function updateUI(state) {
    const btn = document.getElementById("toggleBtn");
    btn.innerText = state === "ON" ? "Turn OFF" : "Turn ON";
    btn.className = "toggle-button " + (state === "ON" ? "on" : "off");
    document.getElementById("outputState").innerText = state;
  }

  function fetchState() {
    fetch("/status")
      .then(res => res.text())
      .then(updateUI);
  }

  function toggleOutput() {
    fetch("/toggle")
      .then(res => res.text())
      .then(updateUI);
  }

  setInterval(fetchState, 1000);
  window.onload = fetchState;
</script>
</body>
</html>
)rawliteral";

// ===== Setup =====
void setup() {
  Serial.begin(115200);

  pinMode(OUTPUT_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  digitalWrite(OUTPUT_PIN, LOW);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
}

```

```

Serial.println("\nConnected!");
Serial.print("IP Address: ");
Serial.println(WiFi.localIP());

// ----- Web Routes -----
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/html", index_html);
});

server.on("/toggle", HTTP_GET, [](AsyncWebServerRequest *request) {
    outputState = !outputState;
    digitalWrite(OUTPUT_PIN, outputState ? HIGH : LOW);
    request->send(200, "text/plain", outputState ? "ON" : "OFF");
});

server.on("/status", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send(200, "text/plain", outputState ? "ON" : "OFF");
});

server.begin();
}

// ===== Loop =====
void loop() {
    bool reading = digitalRead(BUTTON_PIN);

    if (reading == LOW &&
        lastButtonState == HIGH &&
        (millis() - lastDebounceTime) > debounceDelay) {

        lastDebounceTime = millis();
        outputState = !outputState;
        digitalWrite(OUTPUT_PIN, outputState ? HIGH : LOW);

        Serial.println(outputState ? "Button -> ON" : "Button -> OFF");
    }

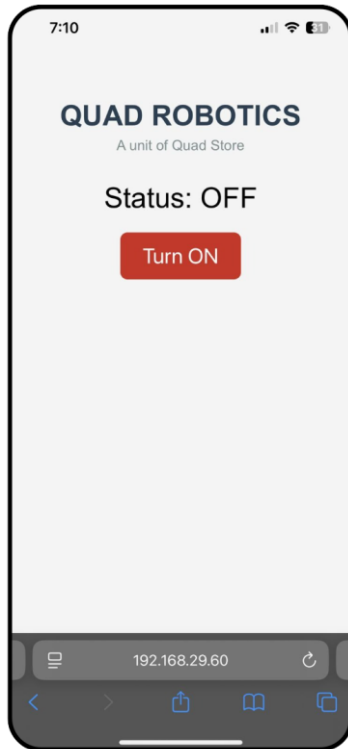
    lastButtonState = reading;
}

```

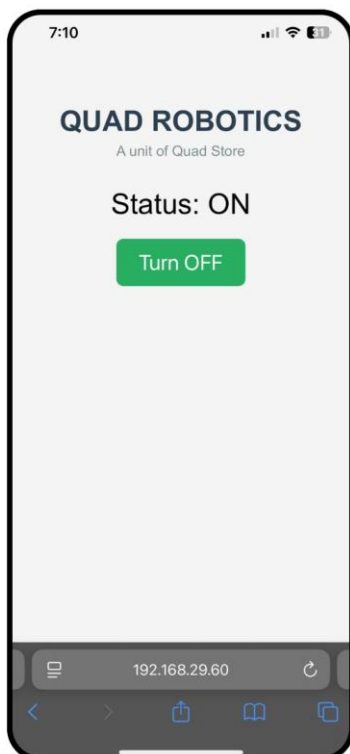
How to Use:

1. Replace YOUR_SSID and YOUR_PASSWORD with your Wi-Fi details.
2. Paste the code into the IDE (or) open the program **Project12_DualControlLED.ino** directly from CODE folder.
3. Upload the code to the ESP32 using the Arduino IDE.

4. Open Serial Monitor to find ESP32's IP address.
5. Access that IP from your phone/computer browser.
6. Use the button on the webpage or press the physical button to toggle output.



7. Web UI updates in real-time with any change.



Project-13: ESP32 Web Server to Control 2-Channel Relay

✓ Objective

Build a web interface hosted on the ESP32 to:

- Control two relays independently over Wi-Fi
- Display real-time ON/OFF status of each relay
- Allow easy toggling via buttons on a web page

📦 Components Required

- ✓ ESP32 Dev Board × 1
- ✓ 2-Channel Relay Module × 1
- ✓ Jumper Wires × As needed
- ✓ Breadboard - Optional
- ✓ USB Cable × 1

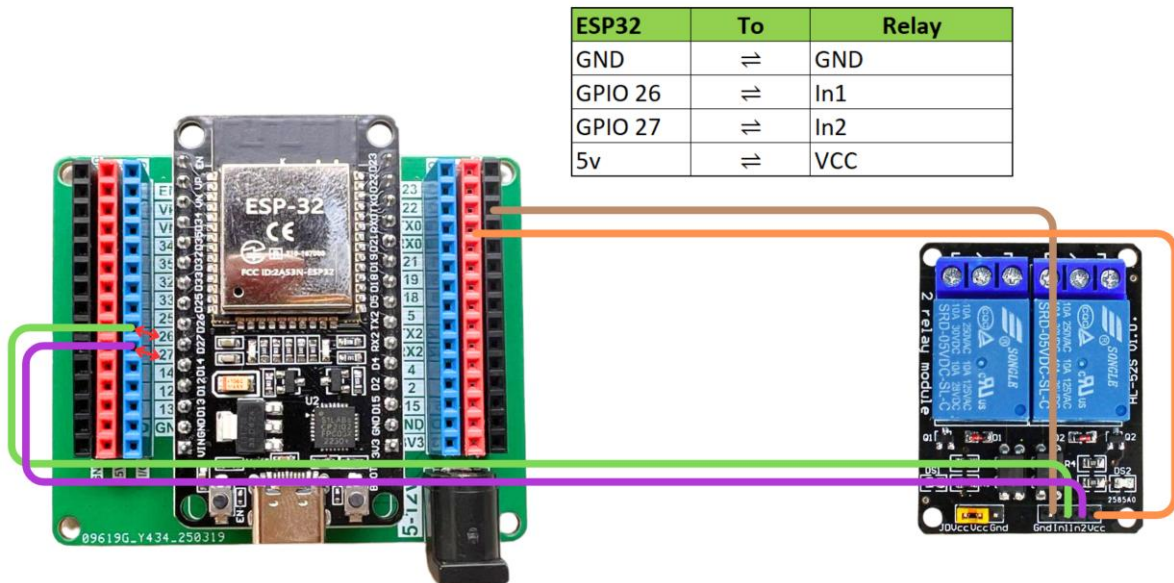
⚠️ Relay Power & Caution

- Relays typically use **IN1/IN2** to control the switches.
- Make sure the **relay module has an opto-isolated circuit** if you're working with AC.
- The ESP32 outputs **3.3V logic**. Most relay modules accept this, but if not, a level shifter may be needed.

🔌 Relay Wiring to ESP32

Relay	Connection Method	ESP32 Board
IN 1	Direct Connection	Pin 26
IN 2	Direct Connection	Pin 27
GND	Direct Connection	GND
VCC	Direct Connection	5v

Note: Relay is **active LOW**. LOW = ON, HIGH = OFF.



Web Server Features

- Hosts a simple HTML page with two buttons
- Buttons toggle each relay independently
- Displays current ON/OFF status for both

Wi-Fi Setup:

IMPORTANT: Before uploading the code, replace:

```
const char* ssid = "YOUR_SSID";
```

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

with your actual **Wi-Fi credentials**.

NOTE: both your SSID and Password should be within Double-Quotes “ ”.

ESP32 Arduino Code

```
#include <WiFi.h>
#include <WebServer.h>

// Wi-Fi credentials
const char* ssid = "YOUR_SSID"; // Replace with your wifi user name
const char* password = "YOUR_PASSWORD"; // replace with your wifi password

// Relay GPIOs
```

```

#define RELAY1 26
#define RELAY2 27

WebServer server(80);

// Track state of each relay
bool relay1State = false;
bool relay2State = false;

// HTML Page
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE html>
<html>
<head>
  <title>QUAD ROBOTICS</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      background-color: #f9f9f9;
      padding: 20px;
    }
    h1 {
      color: #2c3e50;
      margin-bottom: 5px;
    }
    h4 {
      color: #7f8c8d;
      margin-top: 0;
      font-weight: normal;
    }
    .status {
      font-size: 1.4em;
      margin: 20px 0 10px;
    }
    button {
      font-size: 1.2em;
      padding: 10px 25px;
      border: none;
      border-radius: 6px;
      background-color: #3498db;
      color: white;
      cursor: pointer;
      margin-bottom: 20px;
    }
    button:hover {
      background-color: #2980b9;
    }
  </style>
</head>
<body>
  <h1>QUAD ROBOTICS</h1>
  <h4>A unit of Quad Store</h4>

```

```

<div class="status">Relay 1 is <span id="r1">%STATE1%</span></div>
<button onclick="toggleRelay(1)">Toggle Relay 1</button>

<div class="status">Relay 2 is <span id="r2">%STATE2%</span></div>
<button onclick="toggleRelay(2)">Toggle Relay 2</button>

<script>
  function toggleRelay(relay) {
    fetch("/toggle?relay=" + relay)
      .then(response => response.text())
      .then(data => {
        let state = JSON.parse(data);
        document.getElementById("r1").innerText = state.relay1;
        document.getElementById("r2").innerText = state.relay2;
      });
  }
</script>
</body>
</html>
)rawliteral";

// Replace placeholder with current state
String processor(const String& var) {
  if (var == "STATE1") return relay1State ? "ON" : "OFF";
  if (var == "STATE2") return relay2State ? "ON" : "OFF";
  return "";
}

void setup() {
  Serial.begin(115200);
  pinMode(RELAY1, OUTPUT);
  pinMode(RELAY2, OUTPUT);
  digitalWrite(RELAY1, HIGH); // relay OFF
  digitalWrite(RELAY2, HIGH);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  Serial.print("Connecting");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConnected to WiFi!");
  Serial.println(WiFi.localIP());

  // Serve HTML page
  server.on("/", HTTP_GET, []() {
    String html = index_html;
    html.replace("%STATE1%", relay1State ? "ON" : "OFF");
    html.replace("%STATE2%", relay2State ? "ON" : "OFF");
    server.send(200, "text/html", html);
  });
}

```

```

// Handle toggle
server.on("/toggle", HTTP_GET, []() {
  if (server.hasArg("relay")) {
    int relay = server.arg("relay").toInt();
    if (relay == 1) {
      relay1State = !relay1State;
      digitalWrite(RELAY1, relay1State ? LOW : HIGH);
    } else if (relay == 2) {
      relay2State = !relay2State;
      digitalWrite(RELAY2, relay2State ? LOW : HIGH);
    }
  }
  String response = "{\"relay1\": \"" + String(relay1State ? "ON" : "OFF") + "\", \"relay2\": \"" + String(relay2State ? "ON" : "OFF") + "\"}";
  server.send(200, "application/json", response);
});

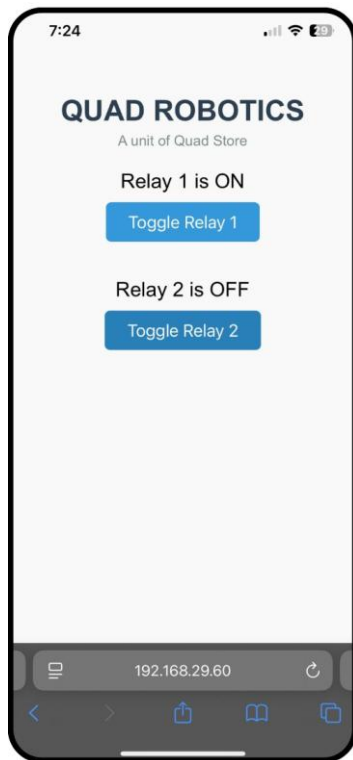
server.begin();
}

void loop() {
  server.handleClient();
}

```

✓ How to Use

1. Replace YOUR_SSID and YOUR_PASSWORD with your Wi-Fi credentials.
2. Paste the code into the IDE (**or**) open the program **Project13_Relay.ino** directly from CODE folder.
3. Upload the code to your ESP32 using Arduino IDE.
4. Open the Serial Monitor to get the IP address.
5. Open the IP in any browser connected to the same network.
6. Toggle Relay 1 and Relay 2 from the web interface.



7. The status updates immediately after each click.

Notes

- The relays are **active LOW**, so LOW turns them **ON**.
- You can modify this for more relays by adding more GPIOs and HTML buttons.
- Ideal for home automation, controlling fans, lights, or devices remotely.

Project-14: ESP32 Web Server to Display DHT11 Temperature on Mobile

Objective

Use an ESP32 and DHT11 sensor to:

- Read and display **temperature** (and optionally humidity)
- Host a web page accessible from a **mobile browser**
- Auto-refresh temperature values every few seconds using **AJAX**

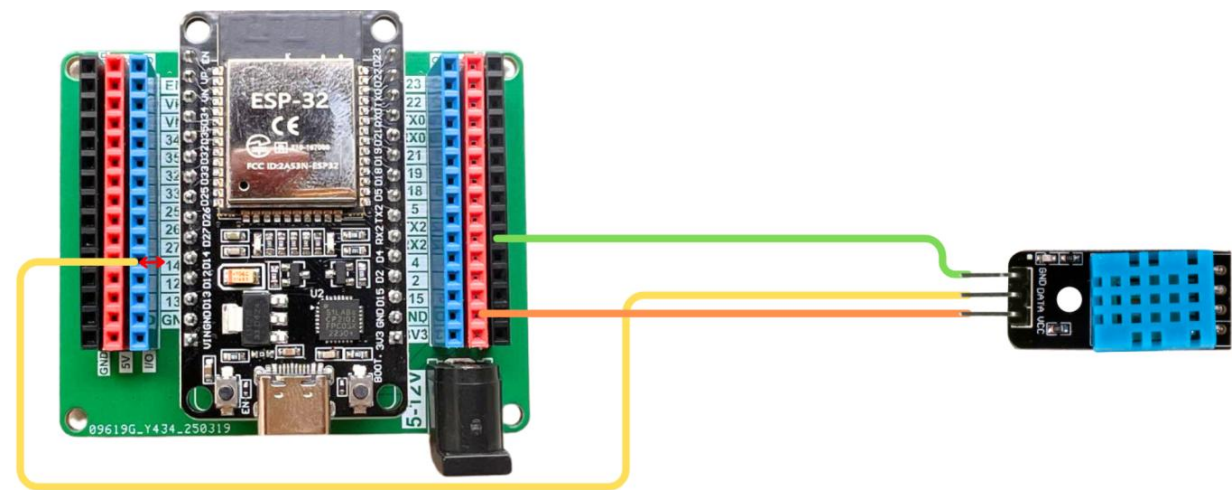
Components Required

- ✓ ESP32 Dev Board × 1
- ✓ DHT11 Sensor × 1
- ✓ 10kΩ Resistor × 1
- ✓ Breadboard × 1
- ✓ Jumper Wires × As needed
- ✓ USB Cable × 1

Circuit Connections

DHT 11 Sensor	Connection Method	ESP32 Board
DATA	Direct Connection	Pin 14
GND	Direct Connection	GND
VCC	Direct Connection	5v

(Optional: place a 10kΩ pull-up resistor between DATA and VCC)



Libraries Required

Install via Arduino Library Manager:

- DHT sensor library by Adafruit
- Adafruit Unified Sensor
- WiFi.h (comes with ESP32 core)

Web Server Features

- Displays temperature (and humidity if needed)
- Automatically updates every 5 seconds
- Mobile-responsive layout

Wi-Fi Setup:

IMPORTANT: Before uploading the code, replace:

```
const char* ssid = "YOUR_SSID";
```

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

with your actual **Wi-Fi credentials**.

NOTE: both your SSID and Password should be within Double-Quotes “ ”.

ESP32 Arduino Code

```
#include <WiFi.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>

// Wi-Fi Credentials
const char* ssid = "YOUR_SSID"; // Replace with your wifi user name
const char* password = "YOUR_PASSWORD"; // replace with your wifi password

// DHT11 settings
#define DHTPIN 14 // GPIO where DHT11 is connected
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
```

```

// Create server on port 80
WiFiServer server(80);

// Global temperature variable
float temperature = 0.0;

void setup() {
  Serial.begin(115200);
  dht.begin();

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500); Serial.print(".");
  }
  Serial.println("\nConnected! IP address: " + WiFi.localIP().toString());

  server.begin();
}

void loop() {
  WiFiClient client = server.available();

  if (client) {
    Serial.println("Client connected");
    while (client.connected()) {
      if (client.available()) {
        String request = client.readStringUntil('\r');
        Serial.println(request);
        client.readStringUntil('\n');

        if (request.indexOf("/temperature") != -1) {
          temperature = dht.readTemperature(); // °C
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/plain");
          client.println("Connection: close");
          client.println();
          client.println(temperature);
        } else {
          temperature = dht.readTemperature();
          String html = "<!DOCTYPE html><html><head><title>ESP32 Temp</title>";
          html += "<meta name='viewport' content='width=device-width, initial-";
          scale=1'>";
          html += "<style>body{font-family:Arial;text-align:center;} h1{margin-";
          top:30px;font-size:2.5em;} h2{color:gray;font-size:1.2em;} #temp{font-";
          size:2.2em;margin-top:20px;}</style>";
          html +=
            "<script>setInterval(()=>{fetch('/temperature').then(r=>r.text()).then(t=>{documen";
            t.getElementById('temp').innerHTML=t+'&deg;C'})},5000);</script>";
          html += "</head><body>";
          html += "<h1>QUAD ROBOTICS</h1>";
          html += "<h2>A unit of Quad Store</h2>";
          html += "<h3>Room Temperature</h3>";
          html += "<p id='temp'>" + String(temperature) + "&deg;C</p>";
          html += "</body></html>";

          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
        }
      }
    }
  }
}

```

```

        client.println("Connection: close");
        client.println();
        client.println(html);
    }
    break;
}
}
delay(1);
client.stop();
Serial.println("Client disconnected");
}
}

```

How to Use on Mobile

1. Replace YOUR_SSID and YOUR_PASSWORD in the code.
2. Upload the program **Project14_WebDHT11.ino** to the ESP32 via Arduino IDE.
3. Open Serial Monitor — note the IP address.
4. Open the IP in your **mobile browser**.
5. The page shows **real-time temperature** and updates every 5 seconds.

