User Manual
IoT Using ESP32

by GI Electronic

Contents

1. Introduction 2
2. ESP32 Pinout 3
3. Best Pins to Use 3
4. Software Setup 5
5. Basic Test: Upload a Blink Sketch 7
6. Troubleshooting Common Upload Issues 7
7. Using the Serial Monitor 8
8. LED Control 9
9. Buzzer Control 10
10. OLED Display Interfacing (I2C) 11
11. DHT11 Sensor for Temperature & Humidity 12
12. Single Relay Module Control 14
13. Integrating LED, Buzzer, OLED, DHT11 sensor, and Relay 15
14. Control the LED using the ESP32 Bluetooth App 17
15. ESP32 Web Server (WiFi) 21
16. ESP32 with Blynk IOT 27
17. Blynk Setup in details 32

1. Introduction

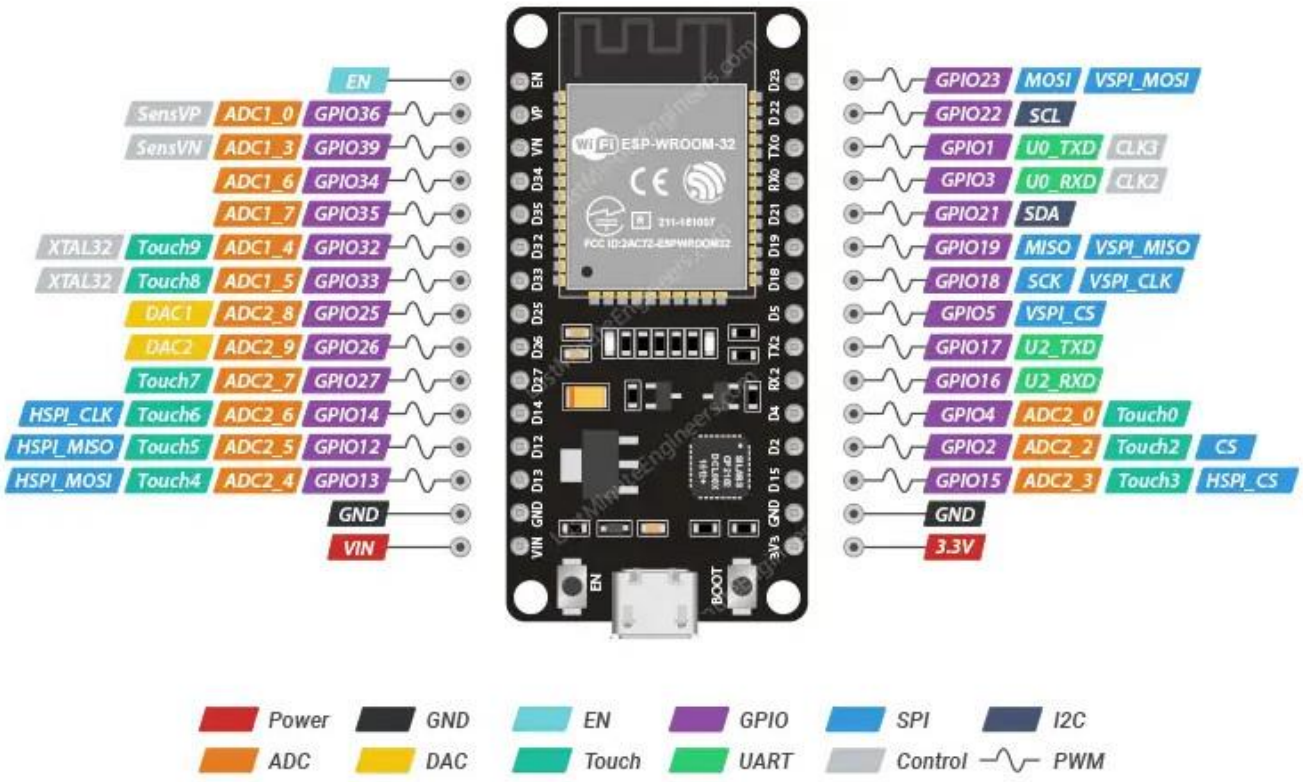
The ESP32 is a powerful, low-cost microcontroller with built-in Wi-Fi and Bluetooth capabilities, developed by Espressif Systems. It's widely used in IoT (Internet of Things) projects, home automation, wearables, and many other applications due to its versatility, power efficiency, and connectivity features.

Some highlights of the ESP32 include:

- **Dual-core processor**: It has two Xtensa LX6 processors, allowing for faster and more efficient multitasking.
- **Memory**: It typically includes 520 KB SRAM and can be paired with additional external RAM.
- **Connectivity**: It supports both Wi-Fi (802.11 b/g/n) and Bluetooth 4.2 (including BLE), making it ideal for wireless communication.
- **I/O options:** With a large number of GPIO pins, PWM, ADC, DAC, and SPI/I2C/UART interfaces, it can connect to various sensors, displays, and other peripherals.
- **Low power modes**: It includes several low-power modes, which makes it ideal for battery-powered applications.

People use ESP32 for projects like home automation, wireless sensors, and remote data logging because it's cost-effective, easy to use, and has a strong online community

2. ESP32 Pinout



3. Best Pins to Use

Although the ESP32 has a lot of pins with various functions, some of them may not be suitable for your projects. The table below shows which pins are safe to use and which pins should be used with caution.

The pins highlighted in green are OK to use. The ones highlighted in yellow are OK to use, but you need to pay attention because they may have unexpected behavior mainly at boot. The pins highlighted in red are not recommended to use as inputs or outputs.

Label	GPIO	Safe to use?	Notes
D0	GPIO0	OK	HIGH at boot must be HIGH during boot and LOW for programming
TX0	GPIO1	X	Tx pin, used for flashing and debugging
D2	GPIO2	OK	must be LOW during boot and also connected to the on-board LED
RX0	GPIO3	X	Rx pin, used for flashing and debugging
D4	GPIO4	OK	
D5	GPIO5	OK	must be HIGH during boot
D6	GPIO6	X	Connected to Flash memory
D7	GPIO7	X	Connected to Flash memory
D8	GPIO8	X	Connected to Flash memory
D9	GPIO9	X	Connected to Flash memory
D10	GPIO10	X	Connected to Flash memory
D11	GPIO11	X	Connected to Flash memory
D12	GPIO12	OK	must be LOW during boot
D13	GPIO13	OK	
D14	GPIO14	OK	
D15	GPIO15	OK	must be HIGH during boot, prevents startup log if pulled LOW
RX2	GPIO16	OK	
TX2	GPIO17	OK	
D18	GPIO18	OK	
D19	GPIO19	OK	
D21	GPIO21	OK	often used as **SDA** (I2C)
D22	GPIO22	OK	often used as **SCL** (I2C)

D23	**GPIO23**	OK	
D25	**GPIO25**	OK	
D26	**GPIO26**	OK	
D27	**GPIO27**	OK	
D32	**GPIO32**	OK	
D33	**GPIO33**	OK	
D34	**GPIO34**	OK	Input only GPIO, cannot be configured as output
D35	**GPIO35**	OK	Input only GPIO, cannot be configured as output
VP	**GPIO36**	OK	Input only GPIO, cannot be configured as output
VN	**GPIO39**	OK	Input only GPIO, cannot be configured as output

4. Software Setup

a) Install Arduino IDE

- Download and install the Arduino IDE from the official Arduino website.
- Install the version compatible with your operating system (Windows, macOS, Linux).

b) Install the ESP32 Board in Arduino IDE

i) Open Arduino IDE

- Open the Arduino IDE on your computer.

ii) Go to Preferences

- In the Arduino IDE, navigate to **File > Preferences** (Windows) or **Arduino > Preferences** (macOS).

iii) Add Board Manager URL

- In the **"Additional Boards Manager URLs"** field, add the following URL:

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

- If there are already other URLs in the box, simply separate them with a comma and add the new URL.

iv) Open Boards Manager

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

v) Install ESP8266 Package

- In the Boards Manager window, type **"ESP32"** into the search bar.
- Find the **ESP32 board package by Espressif Systems** and click **Install**.
- Once installed, close the Boards Manager.

c) Select ESP32 Board in Arduino IDE

i) Go to Tools

- Navigate to **Tools > Board > ESP32 Dev Module**.

Now you've successfully added ESP32 support to the Arduino IDE.

d) Install USB-to-Serial Driver

- **CP210x**: Download VCP(Virtual COM Port) driver and install the driver from here.

Once installed, restart your computer if necessary.

e) Connect ESP32 to Computer

- Use provided **micro-USB cable/type C** to connect your ESP32 to your computer.

f) Select the Correct COM Port

After connecting the ESP32 to your computer:

- Go to **Tools > Port**.
- Select the port where your ESP32 is connected (e.g., **COM3**, **COM4**, etc., on Windows Device Manager or **/dev/cu.usbserial** on macOS).

5. Basic Test: Upload a Blink Sketch

Now that your setup is ready, let's upload a simple **Blink** example to test your ESP32.

a) Open the Blink Example
- Go to **File > Examples > 01.Basics > Blink**.

b) Modify the Pin Number
- The onboard LED on the ESP32 is connected to **GPIO 2**. Change the LED_BUILTIN to **2** in the code:

```
void setup() {
  pinMode(2, OUTPUT);  // Initialize onboard LED pin
}

void loop() {
  digitalWrite(2, HIGH);  // Turn LED on
  delay(1000);            // Wait for a second
  digitalWrite(2, LOW);   // Turn LED off
  delay(1000);            // Wait for a second
}
```

c) Upload the Sketch
- Click on the **Upload** button (the right arrow icon) or go to **Sketch > Upload**.
- You should see a "Compiling..." message at the bottom, followed by an "Uploading..." message.
- After successful upload, the onboard LED on the ESP32 will start blinking.

6. Troubleshooting Common Upload Issues

If the sketch doesn't upload, here are some tips.

- **Error: Failed to connect to ESP32**: Press and hold the **BOOT** button on the ESP32 while clicking the upload button in the Arduino IDE. Release it once uploading starts.
- **Baud Rate Mismatch**: Ensure the correct baud rate is set in the **Tools > Port > Serial Monitor** (usually 115200).
- **Port Not Found**: If you don't see your ESP32's COM port, try a different USB cable or reinstall the drivers.

7. Using the Serial Monitor

To print messages to your computer (useful for debugging):

- Go to **Tools > Serial Monitor.**
- Set the baud rate to **115200** (or the baud rate set in your code).
- Use the Serial.print() or Serial.println() functions in your code to print messages. For example:

```
void setup() {
  Serial.begin(115200);  // Initialize Serial Monitor at 115200 baud rate
  Serial.println("ESP32 is ready!");
}

void loop() {
  Serial.println("Blinking LED");
  digitalWrite(2, HIGH);
  delay(1000);
  digitalWrite(2, LOW);
  delay(1000);
}
```

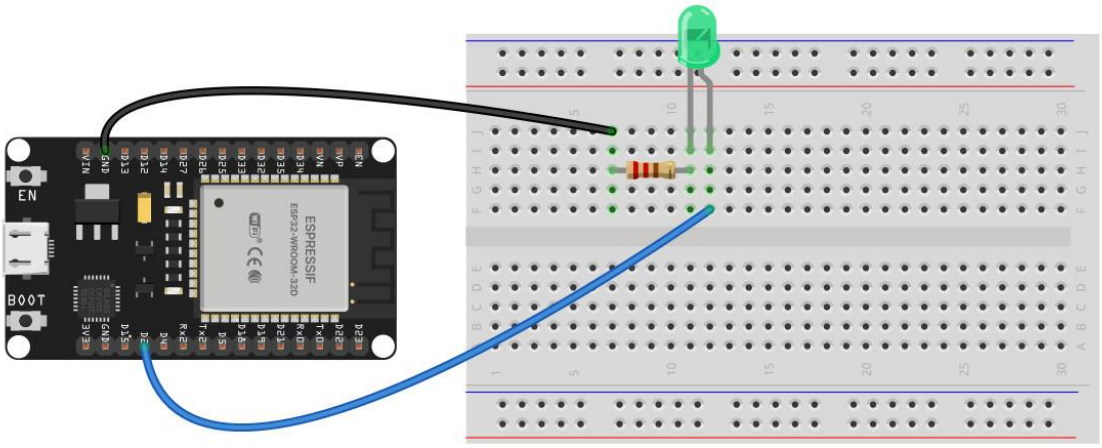
Try upload the code and open the Serial Monitor to see the result.

8. LED Control

Wiring:

- Connect the **positive leg** of the LED to **D2** (GPIO 2) on the ESP32.
- Connect the **negative leg** to the **GND** pin via a **220Ω resistor**.

Diagram:



Code:

```
int ledPin = 2;

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

void loop() {
  digitalWrite(ledPin, HIGH);  // Turn on LED
  delay(1000);  // Wait for 1 second
  digitalWrite(ledPin, LOW);   // Turn off LED
  delay(1000);  // Wait for 1 second
}
```

Explanation:

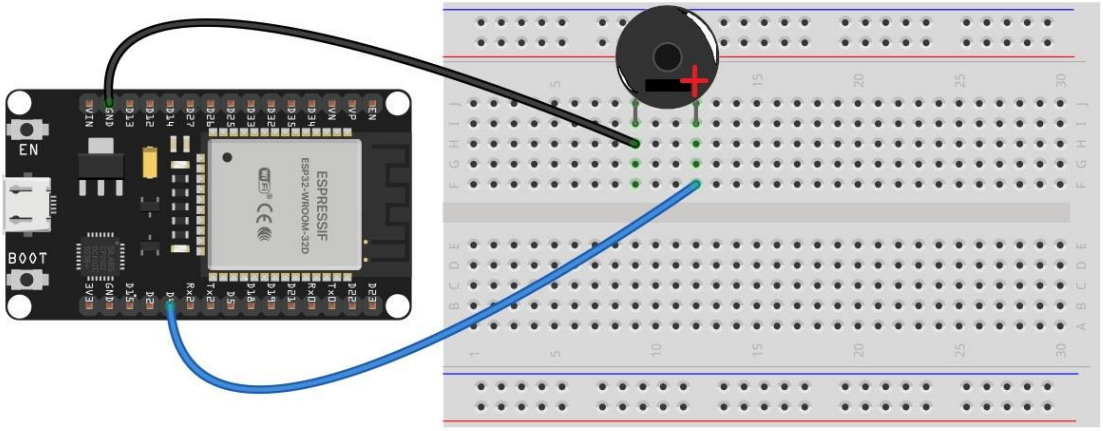
This code will blink an LED connected to the ESP32. The pinMode() function sets the pin as output, and digitalWrite() controls the pin's voltage.

9. Buzzer Control

Wiring:

- Connect the **positive terminal** of the buzzer to **D4** (GPIO 4).
- Connect the **negative terminal** to **GND**.

Diagram:



Code:

```
int buzzerPin = 4;

void setup() {
  pinMode(buzzerPin, OUTPUT);
}

void loop() {
  digitalWrite(buzzerPin, HIGH);  // Buzzer ON
  delay(500);  // Wait 0.5 seconds
  digitalWrite(buzzerPin, LOW);   // Buzzer OFF
  delay(500);  // Wait 0.5 seconds
}
```

Explanation:

This code activates the buzzer, turning it on and off every half second.

10. OLED Display Interfacing (I2C)

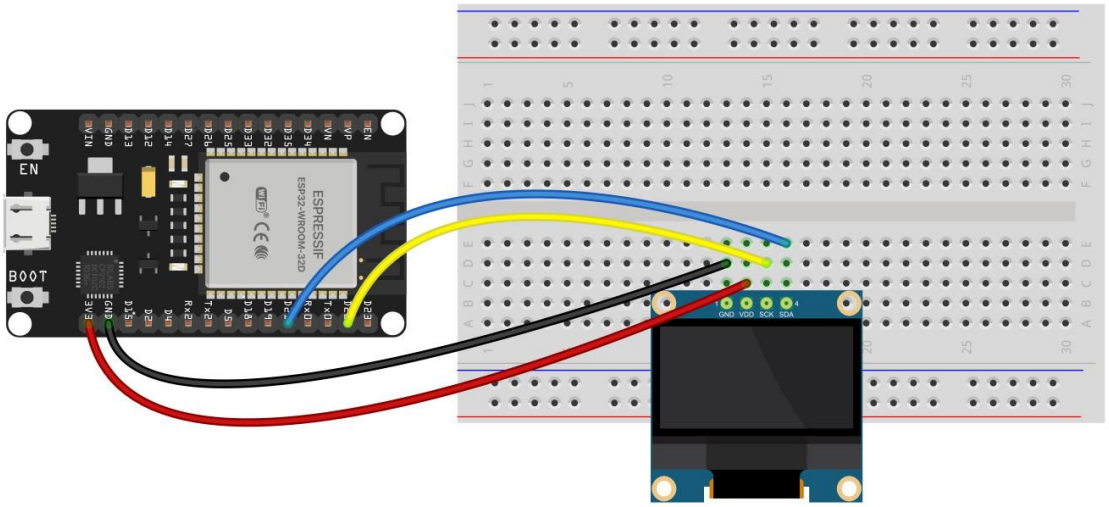
Wiring:

- Connect **VCC** and **GND** of the OLED to **3.3V** and **GND** of the ESP32.
- Connect **SCL/SCK** to **D22** (GPIO22) and **SDA** to **D21** (GPIO21).

Library Required:

- Go to **Tools > Manage Libraries > Library Manager**, search for "**Adafruit SSD1306**" by Adafruit, then click **INSTALL**.

Diagram:



Code:

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

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

void setup() {
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }
  display.clearDisplay();
```

```
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.println("Welcome to IoT World!");
  display.setCursor(0,30);
  display.println("by GI ELECTRONIC");
  display.setCursor(0,40);
  display.println("www.gie.com.my");
  display.display();

}

void loop() {
}
```

Explanation:

This code initializes the OLED display and prints "Welcome to IoT World!" on the screen.

11. DHT11 Sensor for Temperature & Humidity

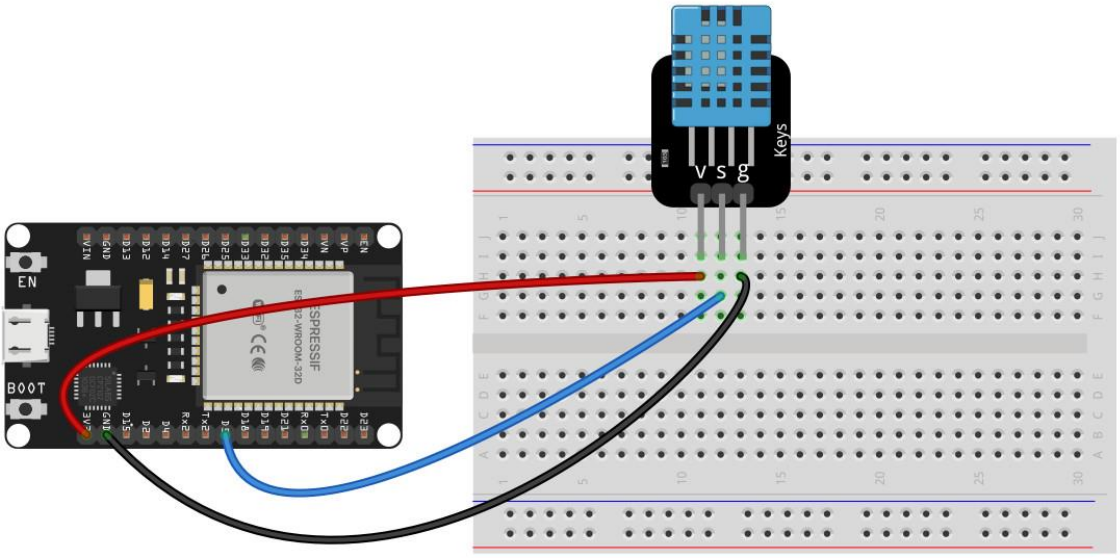
Wiring:

- Connect the **VCC** of DHT11 to ESP32 **3.3V**.
- Connect **GND** to **GND** of ESP32.
- Connect the **Data pin** to **D5** (GPIO 5).

Library Required:

- Go to **Tools > Manage Libraries > Library Manager**, search for "**DHT sensor library**" by Adafruit, then click **INSTALL**.

Diagram:



Code:

```
#include "DHT.h"

#define DHTPIN 5     // Pin connected to DHT11
#define DHTTYPE DHT11 // DHT11 sensor type

DHT dht(DHTPIN, DHTTYPE);

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

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

  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.print(" %\t");
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" *C");

  delay(2000);  // Read every 2 seconds
}
```

Explanation:

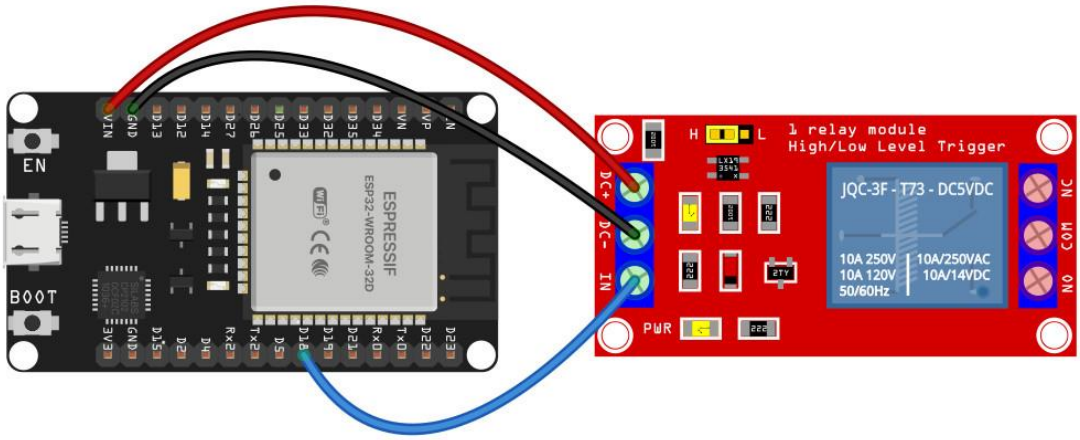
This code reads temperature and humidity from the DHT11 sensor and displays the values in the Serial Monitor.

12. Single Relay Module Control

Wiring:

- Connect **VCC** and **GND** of the relay to **VU** and **GND** on the ESP32.
- Connect the **IN** pin of the relay to **D18** (GPIO18).

Diagram:



Code:

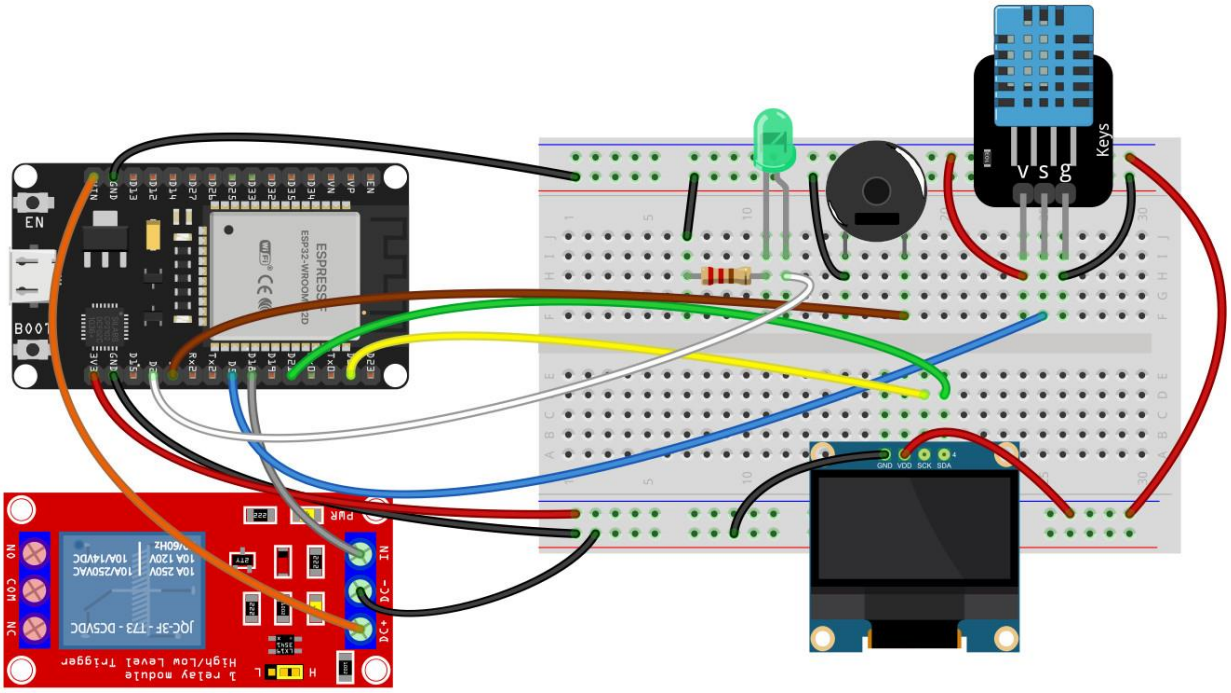
```
int relayPin = 18;

void setup() {
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, LOW);  // Ensure relay starts in OFF position
}

void loop() {
  digitalWrite(relayPin, HIGH);  // Relay ON
  delay(2000);  // Keep ON for 2 seconds
  digitalWrite(relayPin, LOW);   // Relay OFF
  delay(2000);  // Keep OFF for 2 seconds
}
```

Explanation:

This code will control the relay module, turning it on for 2 seconds and off for 2 seconds.

13. Integrating LED, Buzzer, OLED, DHT11 sensor, and Relay

Diagram:



Code:

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

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,-1);

#define DHTPIN 5
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

int ledPin = 2;
```

```
int buzzerPin = 4;
int relayPin = 18;

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

  pinMode(ledPin, OUTPUT);
  pinMode(buzzerPin, OUTPUT);
  pinMode(relayPin, OUTPUT);

  dht.begin();

  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }

  display.clearDisplay();
}

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

  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.println("Temp: " + String(temperature) + " C");
  display.println("Hum: " + String(humidity) + " %");
  display.display();

  digitalWrite(ledPin, HIGH);
  digitalWrite(buzzerPin, HIGH);
  digitalWrite(relayPin, HIGH);

  delay(1000);

  digitalWrite(ledPin, LOW);
  digitalWrite(buzzerPin, LOW);
  digitalWrite(relayPin, LOW);

  delay(1000);
}
```

Explanation:

This code reads temperature and humidity from the DHT11 sensor, displays it on the OLED, blinks the LED, turns on the buzzer, and toggles the relay.

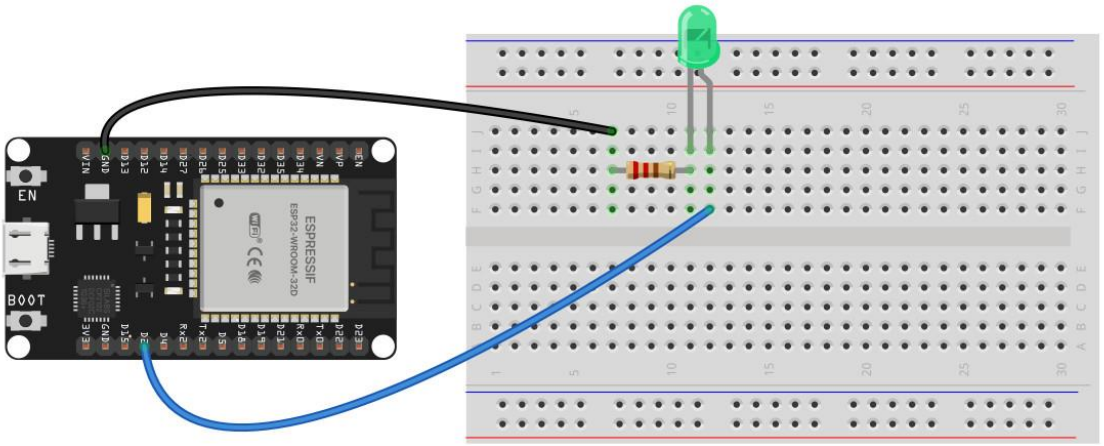
14. Control the LED using the ESP32 Bluetooth App

ESP32 has on-chip Bluetooth and BLE (Bluetooth Low Energy), and it is also referred as classic Bluetooth.

Wiring:

- Connect the **positive leg** of the LED to **D2** (GPIO 2) on the ESP32.
- Connect the **negative leg** to the **GND** pin via a **220Ω resistor**.

Diagram:



Install Required Software:

- **Serial Bluetooth Terminal**: Download the app from the Google Play Store.

Code:

```
/*
 Controlling LED/GPIO using Bluetooth
```

```
 http::://www.electronicwings.com
*/

#include "BluetoothSerial.h"

const char LED= 2;
const char turnON ='a';
const char turnOFF ='b';

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif

BluetoothSerial SerialBT;

void setup() {
 Serial.begin(115200);
 pinMode(LED, OUTPUT);
 SerialBT.begin("ESP32test"); //Bluetooth device name
 Serial.println("The device started, now you can pair it with bluetooth!");
 Serial.println("Now You can TURN ON LED by sending 'a' and TURN OFF by 'b'");
}

void loop() {
 char message;

 if (SerialBT.available()) {
  message=SerialBT.read();
  Serial.write(message);

  if(message==turnON){
   digitalWrite(LED, HIGH);     //Turn LED ON
   Serial.println(" :LED Turned ON");
   SerialBT.println("LED Turned ON");
  }

  else if(message==turnOFF){
   digitalWrite(LED, LOW);       //Turn LED Off
   Serial.println(" :LED Turned OFF");
   SerialBT.println("LED Turned OFF");
  }

  else{
   Serial.println(" :Invalid Input");
   SerialBT.println("Invalid Input");
  }
 }
 delay(20);
```

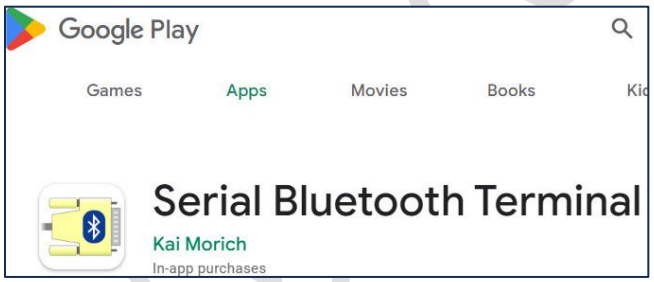
```
}
```

Explanation:

- Once we turn ON the Bluetooth, the code scans for any data available on the Bluetooth serial. if (SerialBT.available())
- Then simply, if the character received is 'a', the Code will Turn ON the LED. Also, it will send the action "LED Turned ON" to the Bluetooth device and computer terminal as well.
- If the Character received is 'b', the Code will Turn OFF the LED.
- If any other character is received, it will be considered invalid.

Testing on ESP32

a. Download and install **Bluetooth Serial Terminal** from Google Play Store.



b. Upload the code in your ESP32 Board

c.

d. Open the Serial Monitor at a baud rate of 115200.

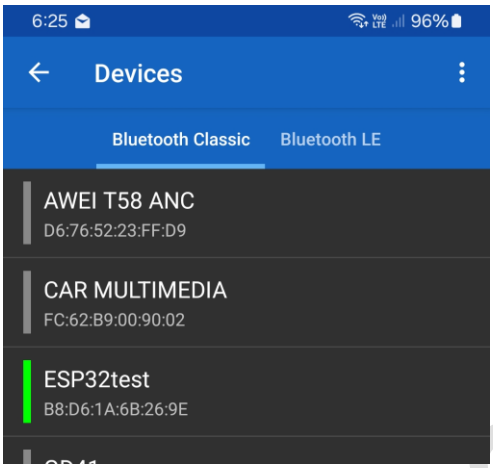
e. Press the ESP32 EN button (reset). The code will start executing and Turns ON the ESP32 Bluetooth Stack.

f. At Android Phone Bluetooth Setting >> search for new device

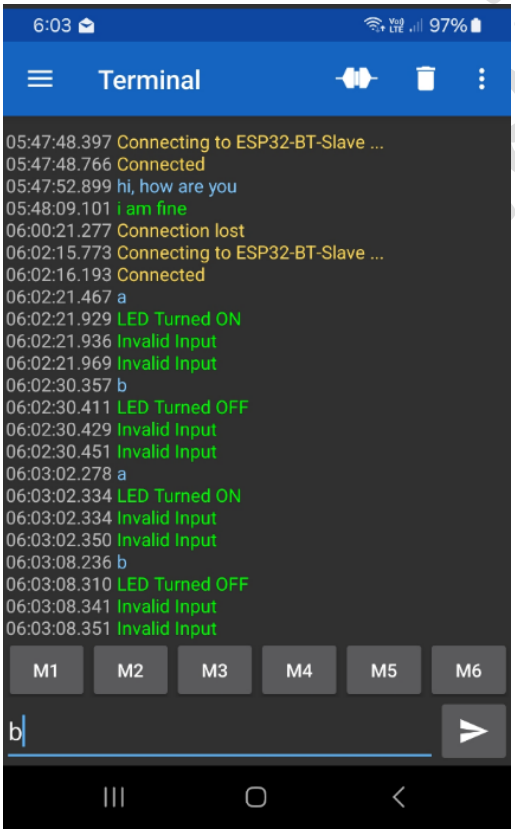
g. You will find the Device with the name "**ESP32test**".

h. Click on it and add it to the pair device.

i. After successfully paring, open the "**Serial Bluetooth Terminal**" App, and connect to the "ESP32test".



j. After connecting with ESP32test, you can type and send the string "**a**" or "**b**" to ESP32 to turn on or off the LED.

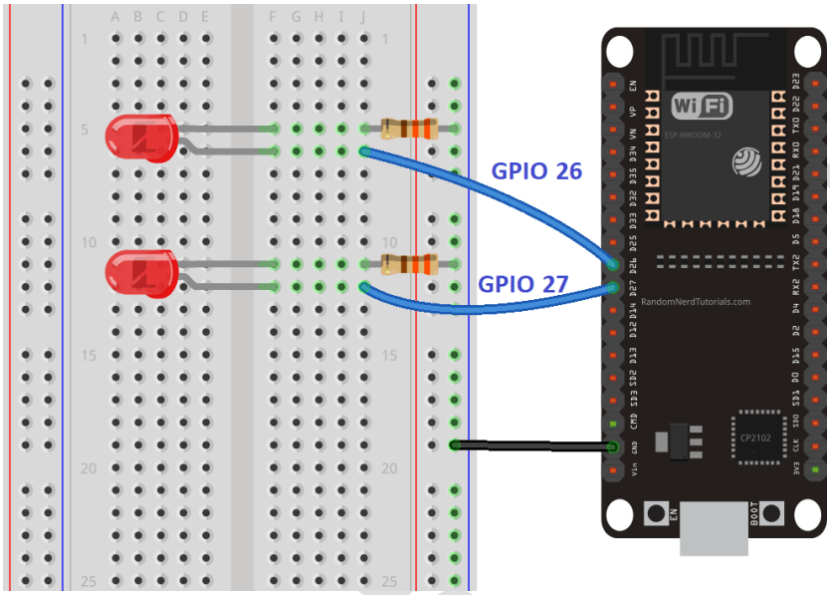


15. ESP32 Web Server (WiFi)

Wiring:

- Connect the **positive leg** of the 1st LED to **D26** (GPIO 26) on the ESP32.
- Connect the **negative leg** of the 1st LED to the **GND** pin via a **220Ω resistor**.
- Connect the **positive leg** of the 2nd LED to **D27** (GPIO 27) on the ESP32.
- Connect the **negative leg of the 2nd LED** to the **GND** pin via a **220Ω resistor**.

Diagram:



Code:

```
/*********
  Rui Santos
  Complete project details at http://randomnerdtutorials.com
*********/

// Load Wi-Fi library
#include <WiFi.h>

// Replace with your network credentials
const char* ssid = "xxx2.4G@unifi";
const char* password = "xxx123";

// Set web server port number to 80
WiFiServer server(80);
```

```
// Variable to store the HTTP request
String header;

// Auxiliar variables to store the current output state
String output26State = "off";
String output27State = "off";

// Assign output variables to GPIO pins
const int output26 = 26;
const int output27 = 27;

// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 2000;

void setup() {
  Serial.begin(115200);
  // Initialize the output variables as outputs
  pinMode(output26, OUTPUT);
  pinMode(output27, OUTPUT);
  // Set outputs to LOW
  digitalWrite(output26, LOW);
  digitalWrite(output27, LOW);

  // Connect to Wi-Fi network with SSID and password
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // Print local IP address and start web server
  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  server.begin();
}

void loop(){
```

```
  WiFiClient client = server.available();   // Listen for incoming clients

  if (client) {                     // If a new client connects,
   currentTime = millis();
   previousTime = currentTime;
   Serial.println("New Client.");        // print a message out in the serial port
   String currentLine = "";             // make a String to hold incoming data from the
client
   while (client.connected() && currentTime - previousTime <= timeoutTime) {  //
loop while the client's connected
     currentTime = millis();
     if (client.available()) {           // if there's bytes to read from the client,
      char c = client.read();            // read a byte, then
      Serial.write(c);                   // print it out the serial monitor
      header += c;
      if (c == '\n') {                   // if the byte is a newline character
       // if the current line is blank, you got two newline characters in a row.
       // that's the end of the client HTTP request, so send a response:
       if (currentLine.length() == 0) {
        // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
        // and a content-type so the client knows what's coming, then a blank line:
        client.println("HTTP/1.1 200 OK");
        client.println("Content-type:text/html");
        client.println("Connection: close");
        client.println();

        // turns the GPIOs on and off
        if (header.indexOf("GET /26/on") >= 0) {
         Serial.println("GPIO 26 on");
         output26State = "on";
         digitalWrite(output26, HIGH);
        } else if (header.indexOf("GET /26/off") >= 0) {
         Serial.println("GPIO 26 off");
         output26State = "off";
         digitalWrite(output26, LOW);
        } else if (header.indexOf("GET /27/on") >= 0) {
         Serial.println("GPIO 27 on");
         output27State = "on";
         digitalWrite(output27, HIGH);
        } else if (header.indexOf("GET /27/off") >= 0) {
         Serial.println("GPIO 27 off");
         output27State = "off";
         digitalWrite(output27, LOW);
        }
```

```
            // Display the HTML web page
            client.println("<!DOCTYPE html><html>");
            client.println("<head><meta name=\"viewport\" content=\"width=device-
width, initial-scale=1\">");
            client.println("<link rel=\"icon\" href=\"data:,\">");
            // CSS to style the on/off buttons
            // Feel free to change the background-color and font-size attributes to fit your
preferences
            client.println("<style>html { font-family: Helvetica; display: inline-block;
margin: 0px auto; text-align: center;}");
            client.println(".button { background-color: #4CAF50; border: none; color:
white; padding: 16px 40px;");
            client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor:
pointer;}");
            client.println(".button2 {background-color: #555555;}</style></head>");

            // Web Page Heading
            client.println("<body><h1>ESP32 Web Server</h1>");

            // Display current state, and ON/OFF buttons for GPIO 26
            client.println("<p>GPIO 26 - State " + output26State + "</p>");
            // If the output26State is off, it displays the ON button
            if (output26State=="off") {
              client.println("<p><a href=\"/26/on\"><button
class=\"button\">ON</button></a></p>");
            } else {
              client.println("<p><a href=\"/26/off\"><button class=\"button
button2\">OFF</button></a></p>");
            }

            // Display current state, and ON/OFF buttons for GPIO 27
            client.println("<p>GPIO 27 - State " + output27State + "</p>");
            // If the output27State is off, it displays the ON button
            if (output27State=="off") {
              client.println("<p><a href=\"/27/on\"><button
class=\"button\">ON</button></a></p>");
            } else {
              client.println("<p><a href=\"/27/off\"><button class=\"button
button2\">OFF</button></a></p>");
            }
            client.println("</body></html>");

            // The HTTP response ends with another blank line
```

```
          client.println();
          // Break out of the while loop
          break;
        } else { // if you got a newline, then clear currentLine
          currentLine = "";
        }
      } else if (c != '\r') {  // if you got anything else but a carriage return character,
        currentLine += c;      // add it to the end of the currentLine
      }
    }
  }
  // Clear the header variable
  header = "";
  // Close the connection
  client.stop();
  Serial.println("Client disconnected.");
  Serial.println("");
 }
}
```

Setting Your Network Credentials:

You need to modify the following lines with your network credentials: SSID and password. The code is well commented on where you should make the changes.

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

Testing on ESP32:

- Upload the code.
- Open the Serial Monitor at a baud rate of 115200.
- Press the ESP32 EN button (reset). The ESP32 connects to Wi-Fi, and outputs the ESP IP address on the Serial Monitor. Copy that IP address, because you need it to access the ESP32 web server.

- Open your browser, paste the ESP32 IP address, and you will see the following page.
- Click the buttons to control the LEDs on/off.

How the Code Works:

- Refer to https://randomnerdtutorials.com/esp32-web-server-arduino-ide/ for details.

16. ESP32 with Blynk IOT

Step-by-step guide on how to use ESP32, an LED, and a DHT11 sensor with Blynk to control the LED and monitor temperature and humidity remotely from a smartphone.

Install Required Software:

- **Blynk App**: Download the Blynk app from the Google Play Store or Apple App Store.
- **Arduino IDE**: Install the Arduino IDE if you don't have it already.

Blynk Setup

a. Create a Blynk Account

- Open the Blynk app on your phone and sign up for a new account.

b. Follow last section 17. Blynk Setup in details to setup the new device.

Wiring:

a. Connect the LED:

- Anode (long leg) of the LED to D2 (GPIO 2)
- Cathode (short leg) to GND via a 220 ohm resistor.

b. Connect the DHT11 Sensor:

- VCC to 3.3V on ESP32.
- GND to GND on ESP32.
- Data pin to D5 (GPIO 5) on ESP32.

Arduino IDE Setup

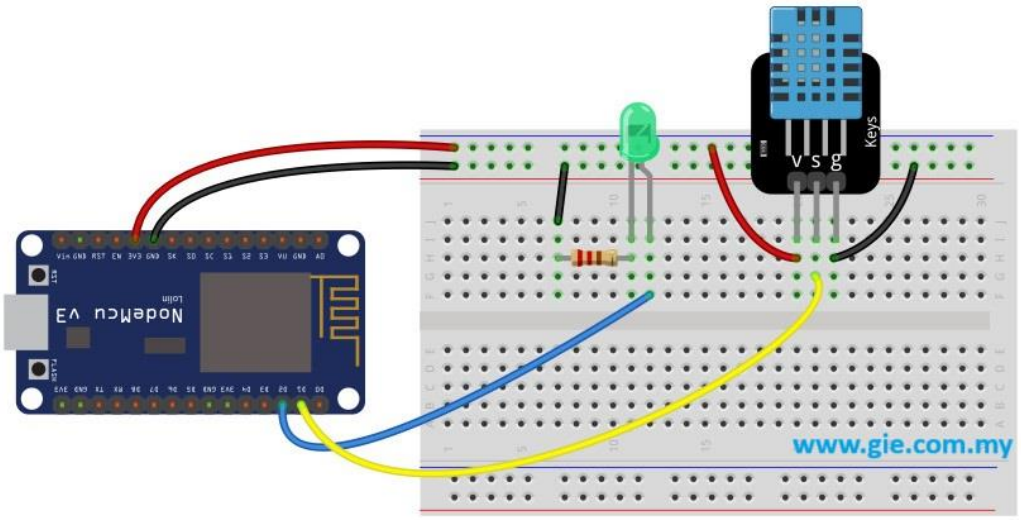
a. Install Blynk Library

- Open the Arduino IDE.
- Go to **Sketch > Include Library > Manage Libraries**.
- Search for **Blynk** and install the **Blynk by Volodymyr Shymanskyy** library.

b. Install DHT Sensor Library (skip this if already installed)

- In the Library Manager, search for DHT and install the DHT sensor library by Adafruit.

Diagram:



Code:

```
#define BLYNK_TEMPLATE_ID "YourTemplateID"
#define BLYNK_DEVICE_NAME "YourDeviceName"
#define BLYNK_AUTH_TOKEN "YourAuthToken"

#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <DHT.h>

// Replace these with your WiFi credentials
char ssid[] = "Your_SSID";
char pass[] = "Your_PASSWORD";

// Blynk Auth Token from the app
char auth[] = BLYNK_AUTH_TOKEN;

// DHT setup
#define DHTPIN 4          // Pin connected to the DHT11 data
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

// LED pin
int ledPin = 2;
```

```
// Blynk virtual pins for widgets
#define LED_VPIN V0
#define TEMP_VPIN V1
#define HUMIDITY_VPIN V2

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

  // Set up LED pin
  pinMode(ledPin, OUTPUT);

  // Start DHT sensor
  dht.begin();

  // Connect to WiFi and Blynk
  Blynk.begin(auth, ssid, pass);
}

void loop() {
  // Blynk.run handles all interactions with the Blynk app
  Blynk.run();

  // Read sensor values
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  // Check if data reading is successful
  if (!isnan(temperature) && !isnan(humidity)) {
    // Send temperature and humidity to Blynk app
    Blynk.virtualWrite(TEMP_VPIN, temperature);
    Blynk.virtualWrite(HUMIDITY_VPIN, humidity);

    Serial.print("Temperature: ");
    Serial.print(temperature);
    Serial.println(" °C");

    Serial.print("Humidity: ");
    Serial.print(humidity);
    Serial.println(" %");
  } else {
    Serial.println("Failed to read from DHT sensor!");
  }
```

```
  delay(2000);  // Read the sensor every 2 seconds
}

// Blynk function to control the LED
BLYNK_WRITE(LED_VPIN) {
  int ledState = param.asInt(); // Get value from the Blynk app
  digitalWrite(ledPin, ledState);
}
```

Explanation:

- Blynk Authentication: The auth[] variable stores your unique authentication token, which you get from the Blynk app.

- Wi-Fi Credentials: Enter your Wi-Fi SSID and password in the ssid[] and pass[] variables.

- Blynk Virtual Pins:
  - **V0**: Used to control the LED (toggle on/off)
  - **V1**: Sends temperature readings.
  - **V2**: Sends humidity readings.

- Blynk Functions:
  - **BLYNK_WRITE(V0)**: Reads the LED toggle state from the Blynk app and controls the physical LED accordingly.

  - **Blynk.virtualWrite()**: Sends the DHT11 sensor values (temperature and humidity) to the app for display.

- Looping and Delays:
  - **Blynk.run()** handles all communication with the app.
  - **Delay(2000)** ensures the DHT11 sensor readings update every 2 seconds.

Upload the Code to ESP32

a. Connect the ESP32 to your computer using a USB cable.

b. In the Arduino IDE, select the correct board and port.
   - Tools > Board > **ESP32 Dev Module**.

- Tools > Port > **COMxx** (or the port your ESP32 is connected to).

c. Upload the code to your ESP32.

d. Open the Serial Monitor to confirm that the device is connecting to WiFi and reading sensor values.

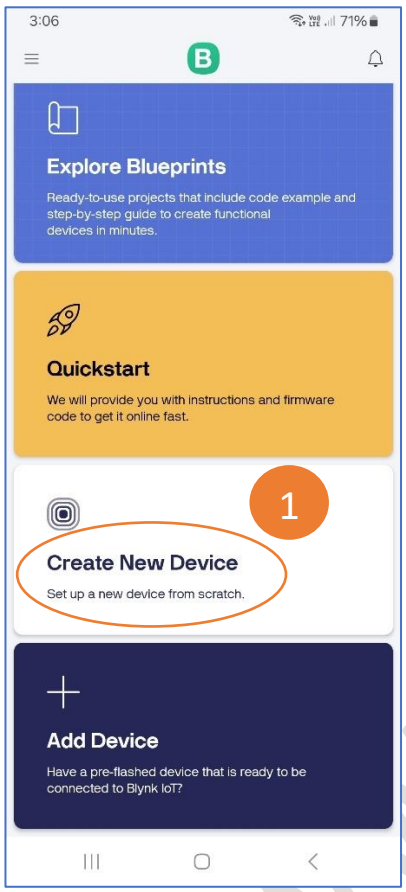
Test the Project

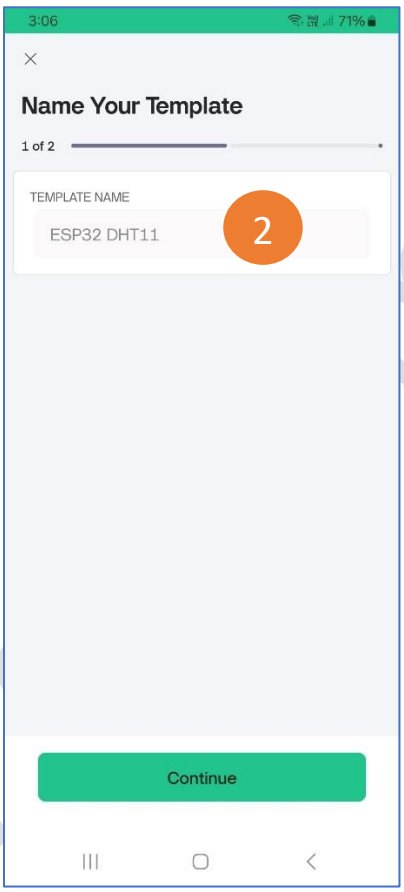
- Open the Blynk app on your smartphone.
- Press the Play button to run the project.
- You should now be able to:
    - Monitor temperature and humidity in real-time.
    - Control the LED by toggling the button in the app.

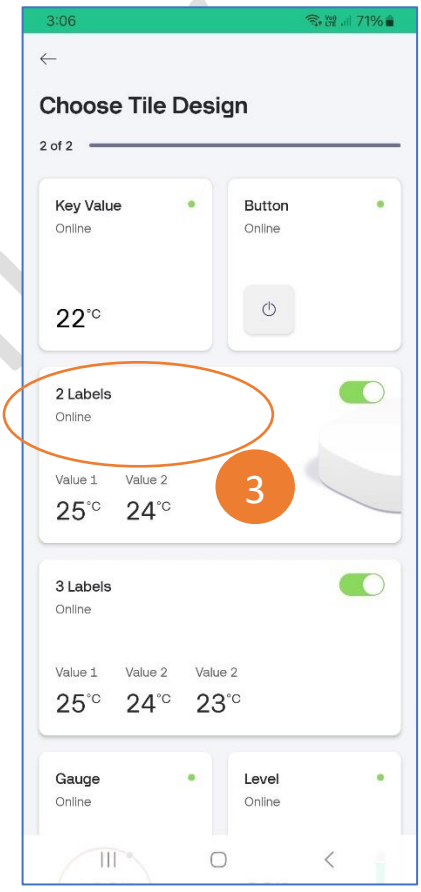
17. Blynk Setup in details

The sequences are **"Create Template"** > **Create Device** base on template created.

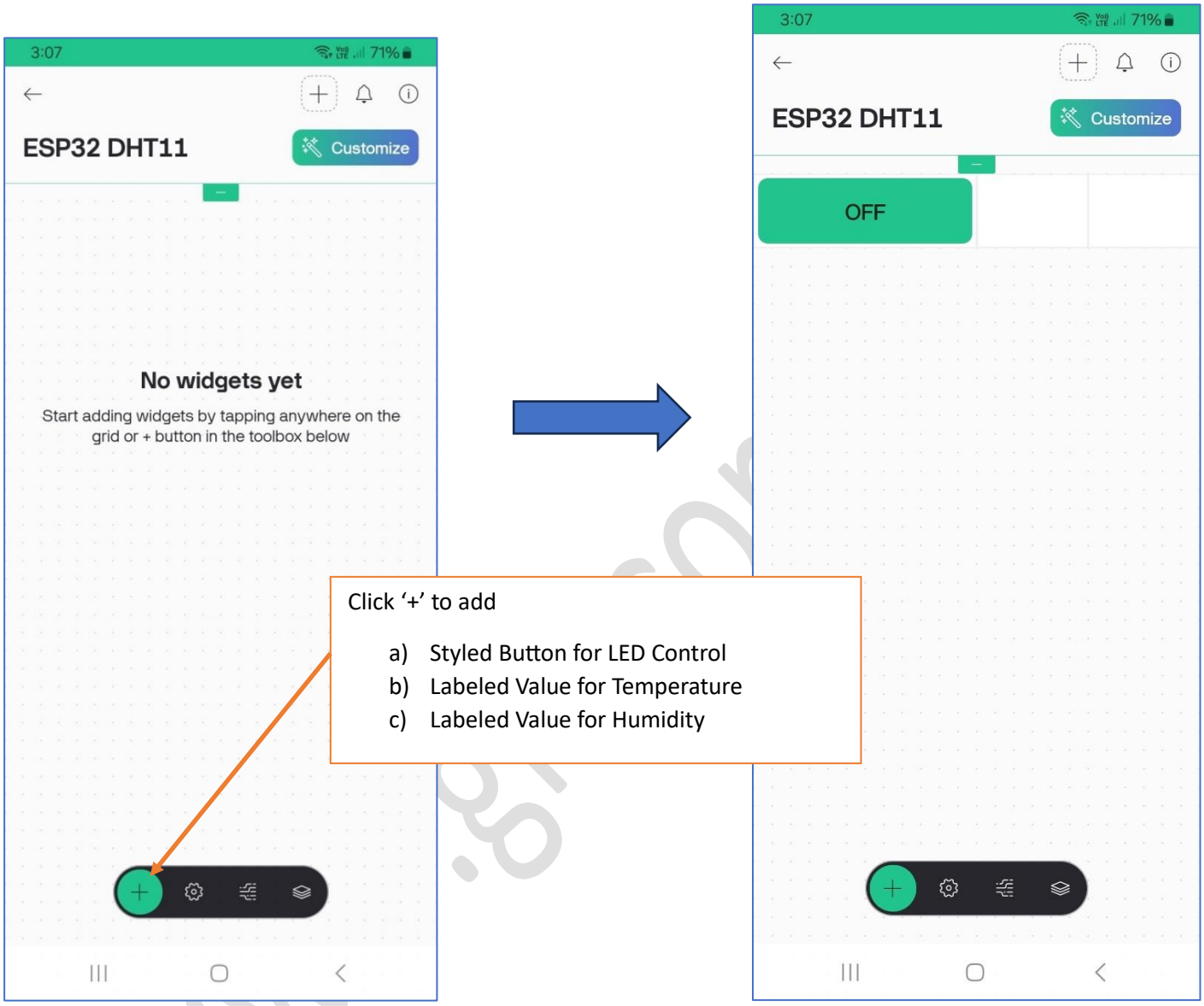
a) Create new Template



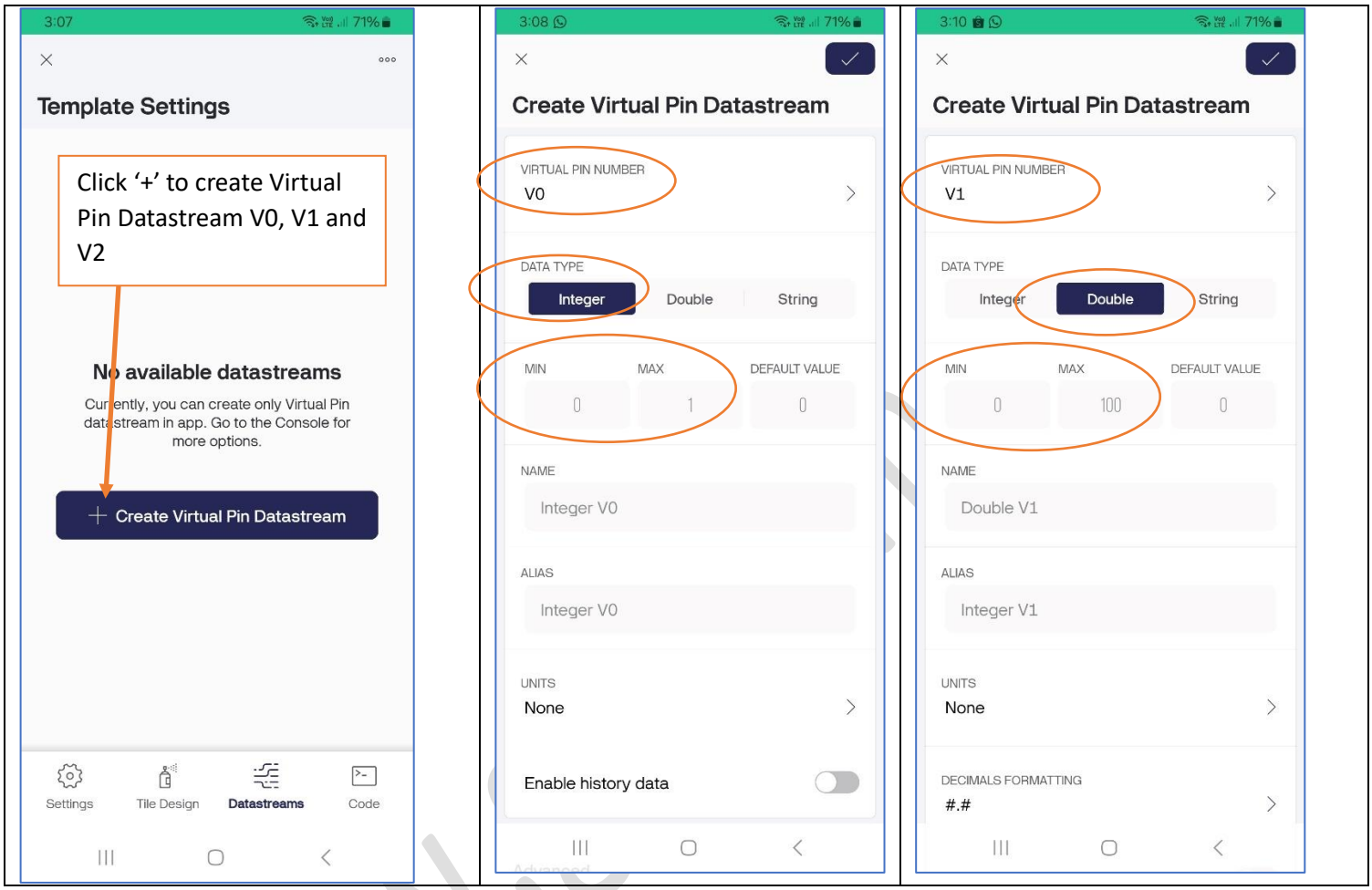


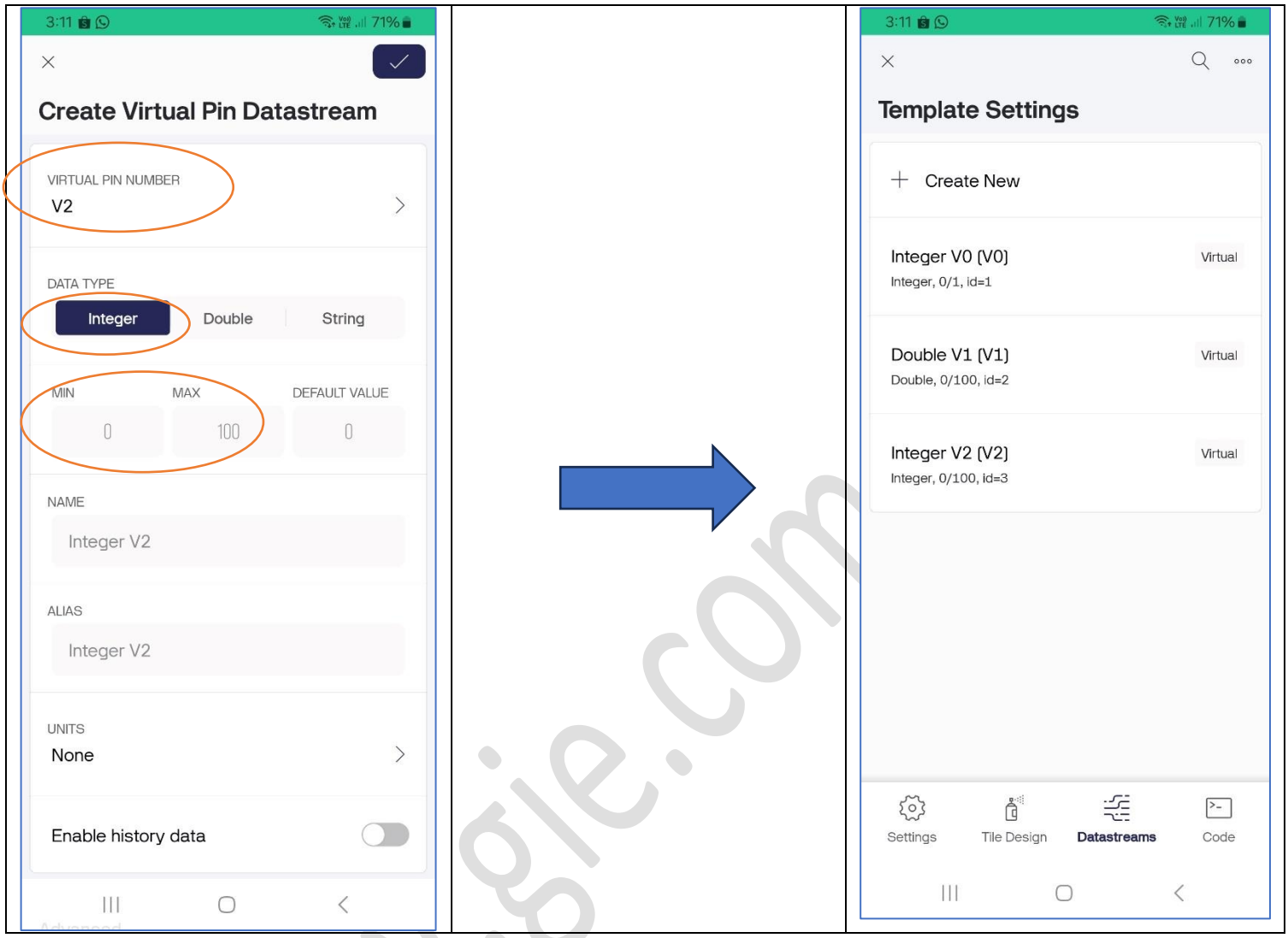


b) Add Widgets

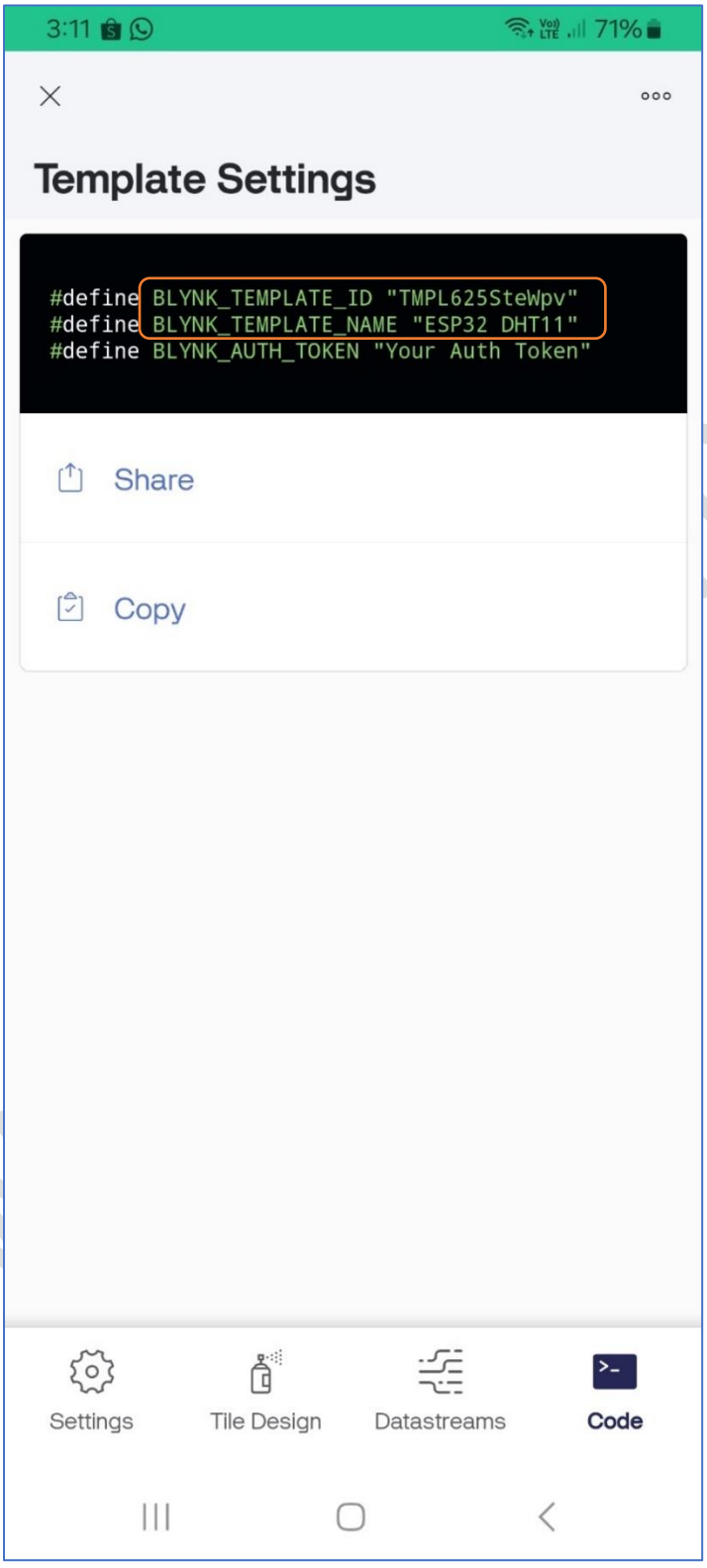


c) Create Virtual Pin Datastream

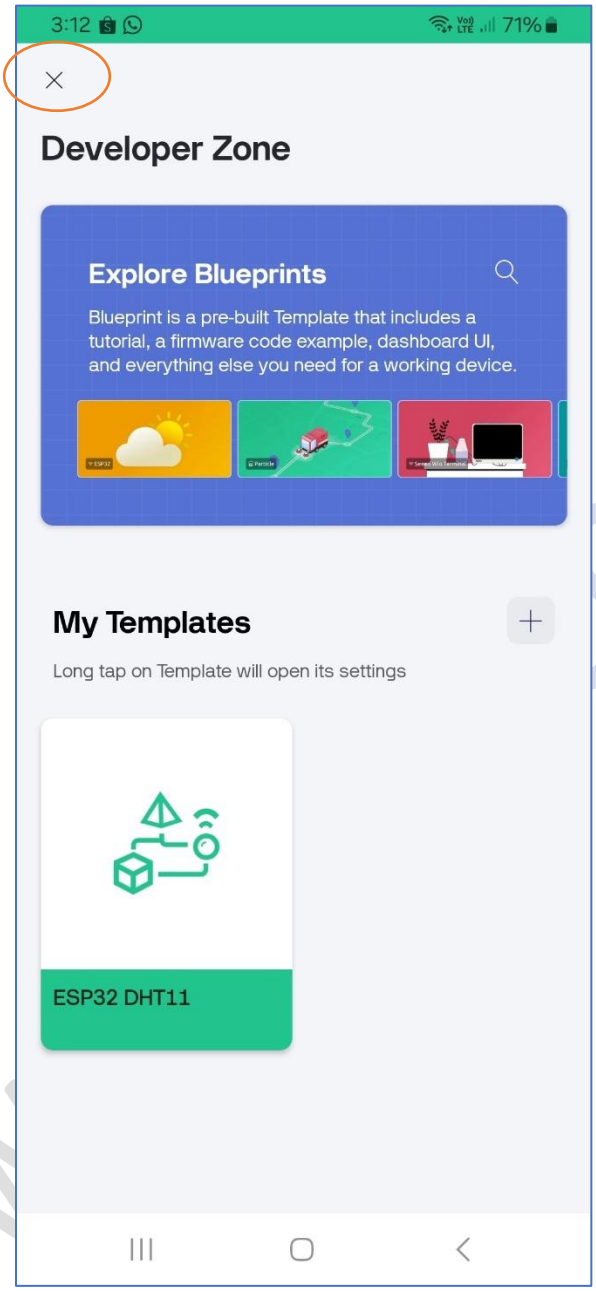



Create Virtual Pin Datastream
VIRTUAL PIN NUMBER
V2
DATA TYPE
Integer
Double
String
MIN
MAX
DEFAULT VALUE
0
100
0
NAME
Integer V2
ALIAS
Integer V2
UNITS
None
Enable history data
Advanced
Template Settings
Create New
Integer V0 (V0)
Integer, 0/1, id=1
Virtual
Double V1 (V1)
Double, 0/100, id=2
Virtual
Integer V2 (V2)
Integer, 0/100, id=3
Virtual
Settings
Tile Design
Datastreams
Code

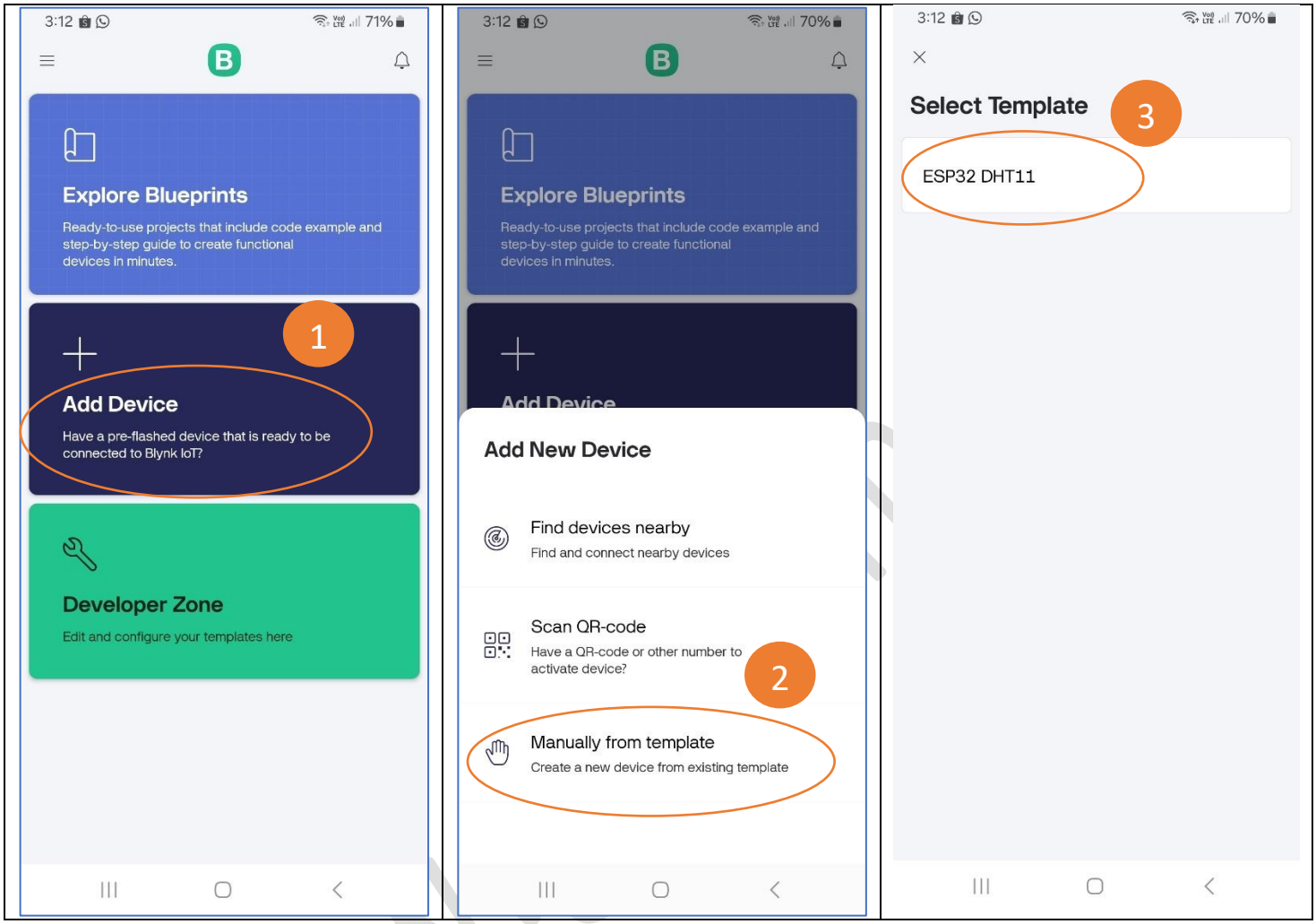
d) Record down **BLYNK_TEMPLATE_ID** and **BLYNK_TEMPLATE NAME**, update these 2 data into code.

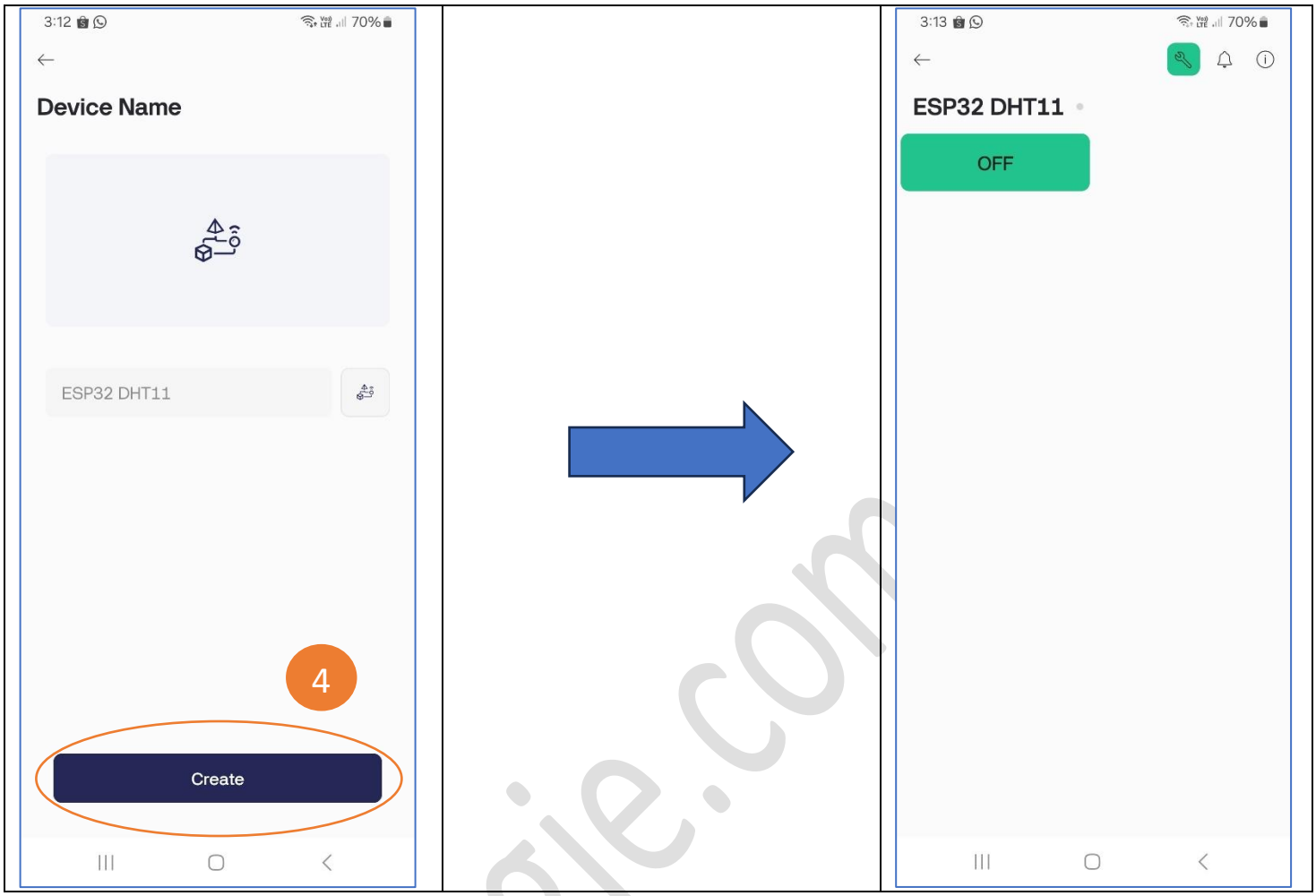


e) Exit from the template setting menu until go into Developer Zone screen. Then click 'X' exit from the Developer Zone.

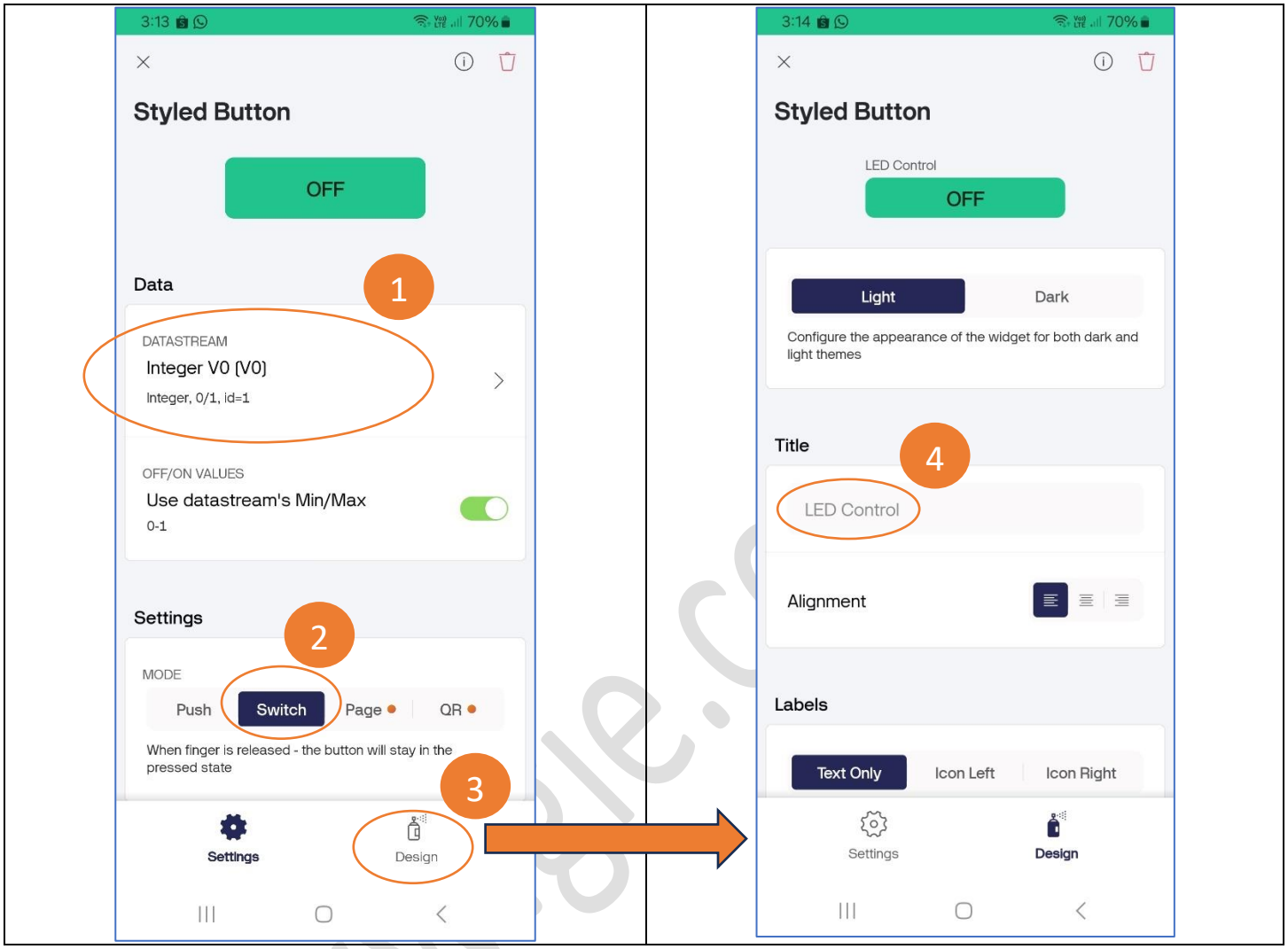


f) Add Device

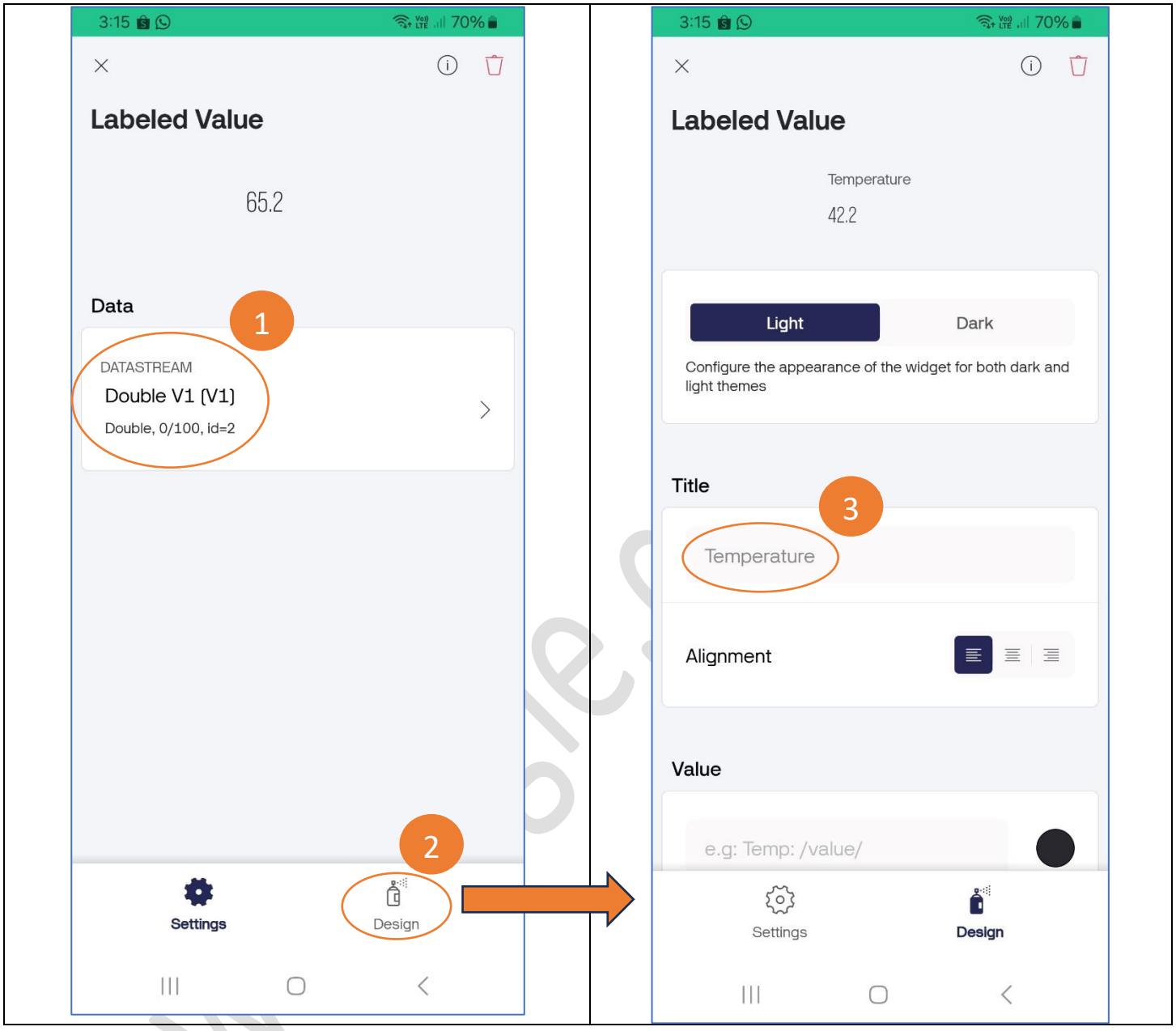

3:12
71%
Explore Blueprints
Ready-to-use projects that include code example and step-by-step guide to create functional devices in minutes.
1
Add Device
Have a pre-flashed device that is ready to be connected to Blynk IoT?
Developer Zone
Edit and configure your templates here
3:12
70%
Explore Blueprints
Ready-to-use projects that include code example and step-by-step guide to create functional devices in minutes.
Add Device
Add New Device
Find devices nearby
Find and connect nearby devices
Scan QR-code
Have a QR-code or other number to activate device?
2
Manually from template
Create a new device from existing template
3:12
70%
Select Template
3
ESP32 DHT11


3:12
70%
Device Name
ESP32 DHT11
4
Create
3:13
70%
ESP32 DHT11
OFF

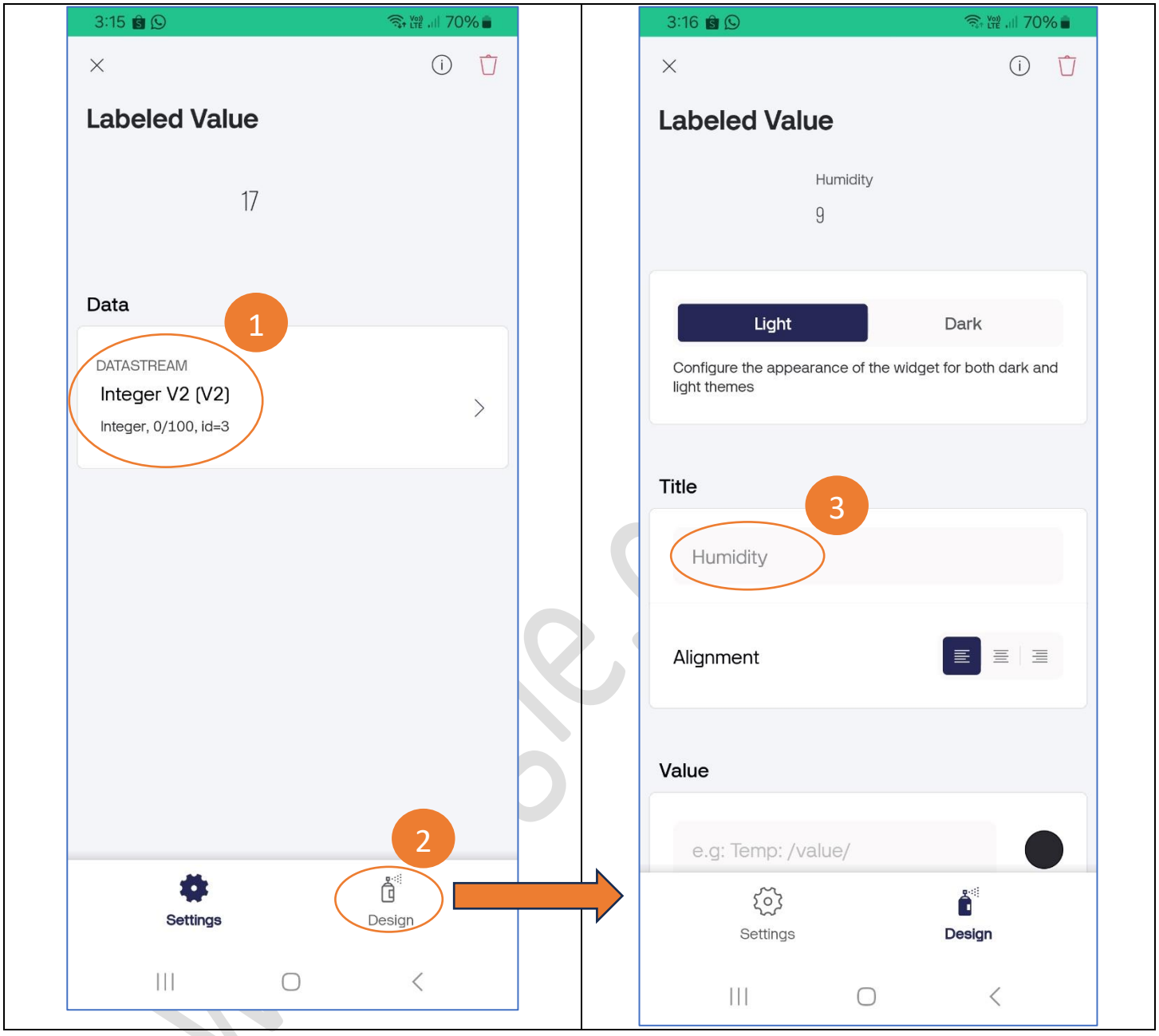
g) Click on Styled Button then configure it as below.



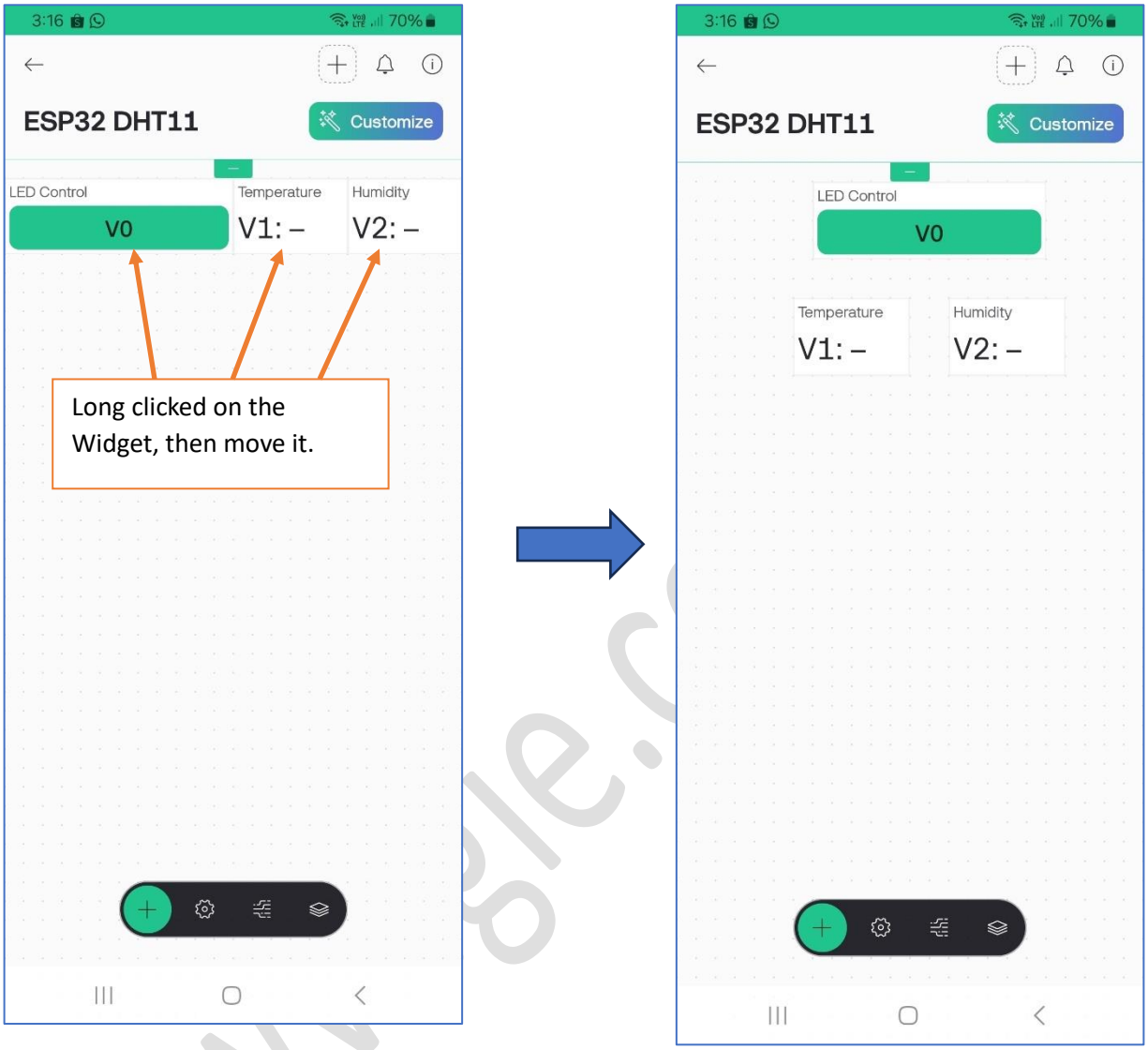
h) Click on Label Display for Temperature then configure it as below.



i) Click on Label Display for Humidity then configure it as below.



j) Arrange Widgets



k) Login to **https://blynk.io/** , then copy the **Auth Token** into your code.