User Manual
IoT Using NodeMCU

by GI Electronic

Contents

1. Introduction

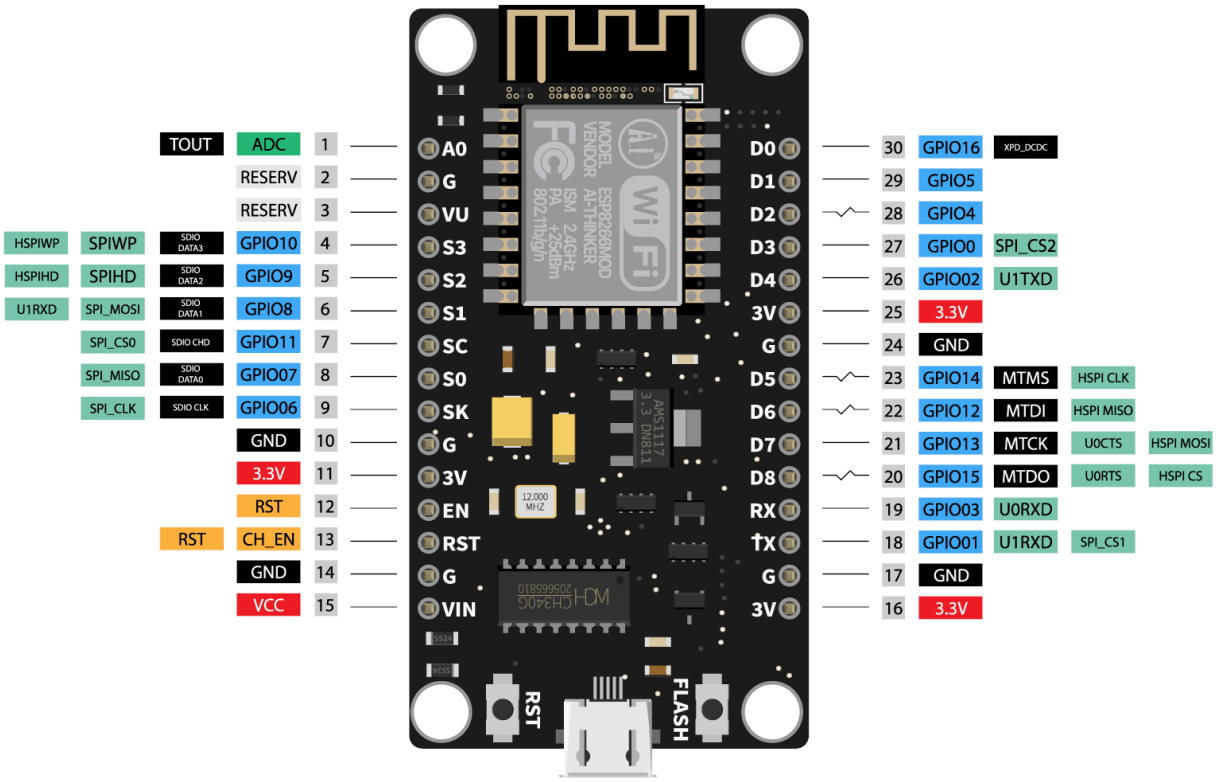
NodeMCU is an open-source firmware and development kit based on the ESP8266 Wi-Fi module. It's popular for building IoT (Internet of Things) projects because it combines a microcontroller (ESP8266) with built-in Wi-Fi capabilities, making it easy to connect devices to the internet.

Here's a quick breakdown of NodeMCU:

- ESP8266: The microcontroller that powers NodeMCU, with built-in Wi-Fi and a powerful CPU for handling IoT tasks.
- LUA scripting language: The NodeMCU firmware originally used the Lua programming language, but most people now program it using the Arduino IDE or MicroPython.
- GPIO pins: NodeMCU provides access to General Purpose Input/Output (GPIO) pins, allowing you to connect sensors, actuators, and other components.
- USB interface: It has a USB-to-serial interface that makes it easy to program and debug using a computer.

People use NodeMCU 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. NodeMCU Pinout



3. Best Pins to Use

One important thing to notice about ESP8266 is that the GPIO number doesn't match the label on the board silkscreen. For example, D0 corresponds to GPIO16 and D1 corresponds to GPIO5.

The following table shows the correspondence between the labels on the silkscreen and the GPIO number as well as what pins are the best to use in your projects, and which ones you need to be cautious.

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	Input	Output	Notes
D0	**GPIO16**	no interrupt	no PWM or I2C support	HIGH at boot used to wake up from deep sleep
D1	**GPIO5**	OK	OK	often used as **SCL** (I2C)
D2	**GPIO4**	OK	OK	often used as **SDA** (I2C)
D3	**GPIO0**	pulled up	OK	connected to FLASH button, boot fails if pulled LOW
D4	**GPIO2**	pulled up	OK	HIGH at boot connected to on-board LED, boot fails if pulled LOW
D5	**GPIO14**	OK	OK	**SPI** (SCLK)
D6	**GPIO12**	OK	OK	**SPI** (MISO)
D7	**GPIO13**	OK	OK	**SPI** (MOSI)
D8	**GPIO15**	pulled to GND	OK	**SPI** (CS) Boot fails if pulled HIGH
RX	**GPIO3**	OK	RX Pin	HIGH at boot
TX	**GPIO1**	TX Pin	OK	HIGH at boot debug output at boot, boot fails if pulled LOW
A0	**ADC0**	Analog Input	X	

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) Add ESP8266 Board Support to Arduino

Since NodeMCU is based on the ESP8266, you need to add support for this 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:

```
http://arduino.esp8266.com/stable/package_esp8266com_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 **ESP8266** into the search bar.
- Find the **ESP8266 by ESP8266 Community** package and click **Install**.
- Once installed, close the Boards Manager.

c) Select NodeMCU Board

i) Go to Tools

- Navigate to **Tools > Board > NodeMCU 1.0 (ESP-12E Module)**.

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

d) Install USB-to-Serial Driver

- **CH340**: Download and install the driver from here.

Once installed, restart your computer if necessary.

e) Connect NodeMCU to Computer

- Use provided **micro-USB cable** to connect your NodeMCU to your computer.

f) Select the Correct COM Port

After connecting the NodeMCU to your computer:

- Go to **Tools > Port**.
- Select the port where your NodeMCU 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 NodeMCU.

a) Open the Blink Example

- Go to **File > Examples > 01.Basics > Blink**.

b) Modify the Pin Number

- The onboard LED on the NodeMCU is connected to **GPIO 2 (D4)**. Change the LED_BUILTIN to **D4** in the code:

```
void setup() {
  pinMode(D4, OUTPUT);  // Initialize onboard LED pin
}

void loop() {
  digitalWrite(D4, HIGH);  // Turn LED on
  delay(1000);             // Wait for a second
  digitalWrite(D4, 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 NodeMCU will start blinking.

6. Troubleshooting Common Upload Issues

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

- **Error: Failed to connect to ESP8266**: Press and hold the **FLASH** button on the NodeMCU 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 NodeMCU'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("NodeMCU is ready!");
}

void loop() {
  Serial.println("Blinking LED");
  digitalWrite(D4, HIGH);
  delay(1000);
  digitalWrite(D4, LOW);
  delay(1000);
}
```

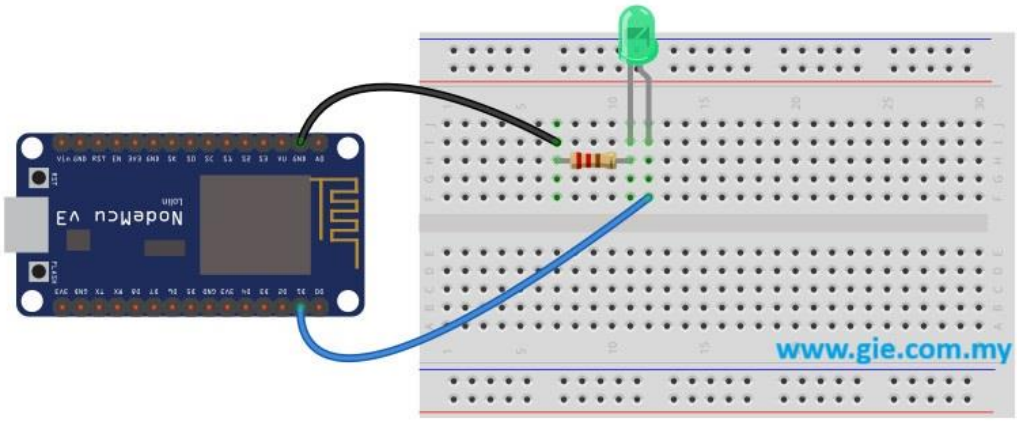
Try upload the code and open the Serial Monitor to see the result.

8. LED Control

Wiring:

- Connect the **positive leg** of the LED to **D1** (GPIO 5) on the NodeMCU.
- Connect the **negative leg** to the **GND** pin via a **220Ω resistor**.

Diagram:



Code:

```
int ledPin = D1;

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:

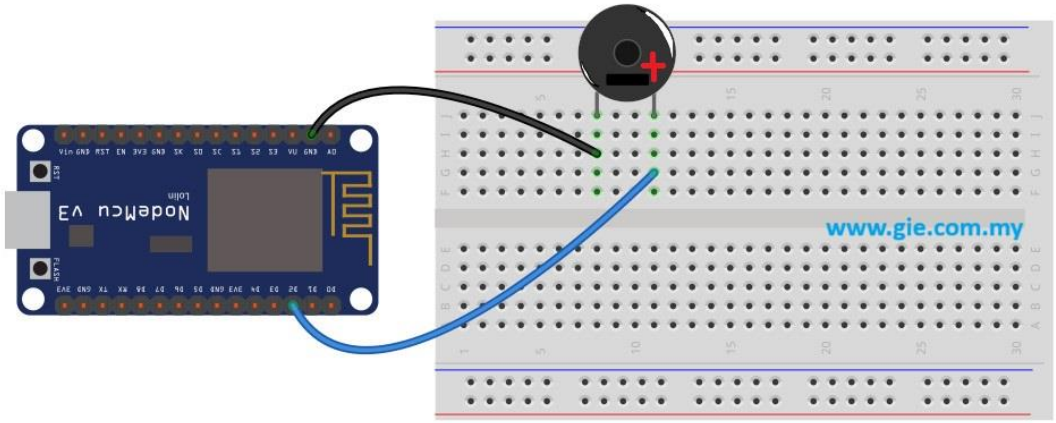
This code will blink an LED connected to the NodeMCU. The pinMode() function sets the pin as output, and digitalWrite() controls the pin's voltage.

9. Buzzer Control

Wiring:

- Connect the **positive terminal** of the buzzer to **D2** (GPIO 4).
- Connect the **negative terminal** to **GND**.

Diagram:



Code:

```
int buzzerPin = D2;

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. LED Display Interfacing (I2C)

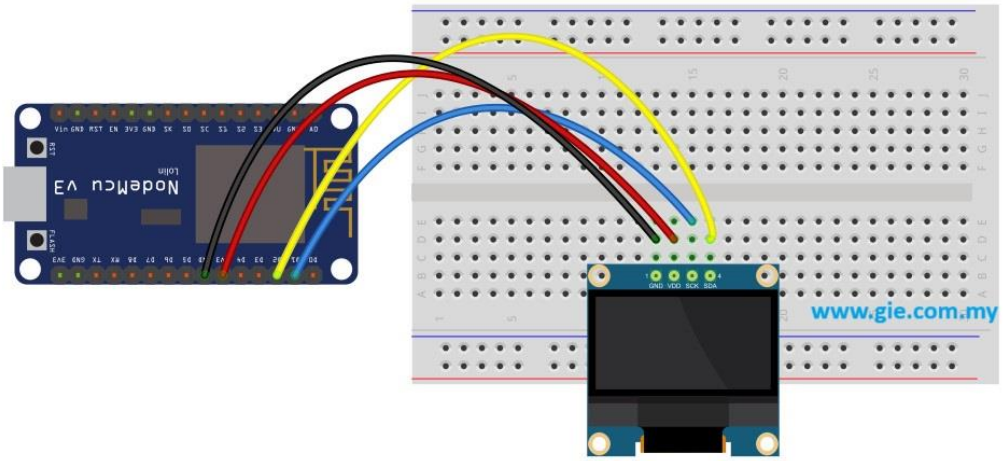
Wiring:

- Connect **VCC** and **GND** of the OLED to **3.3V** and **GND** of the NodeMCU..
- Connect **SCL/SCK** to **D1** (GPIO 5) and **SDA** to **D2** (GPIO 4).

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

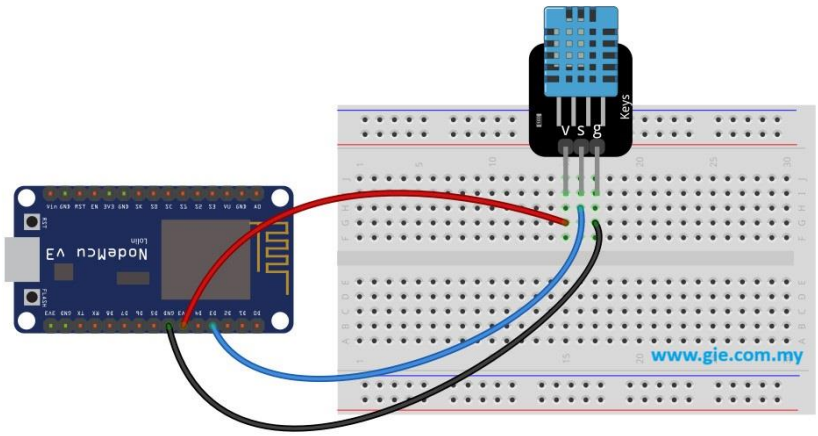
Wiring:

- Connect the **VCC** of DHT11 to **3.3V**.
- Connect **GND** to **GND** of NodeMCU.
- Connect the **Data pin** to **D3** (GPIO 0).

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 D3     // 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:

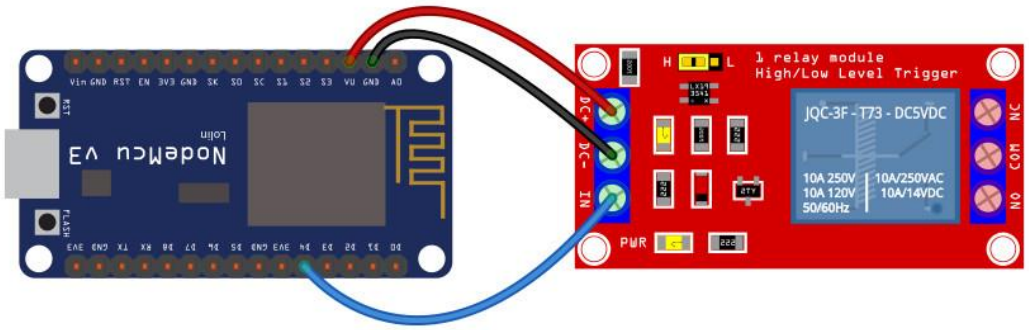
This code reads temperature and humidity from the DHT11 sensor and displays the values in the Serial Monitor.

12. Single Relay Module Control

Wiring:

- Connect **VCC** and **GND** of the relay to **VU** and **GND** on the NodeMCU.
- Connect the **IN** pin of the relay to **D4** (GPIO 2).

Diagram:



Code:

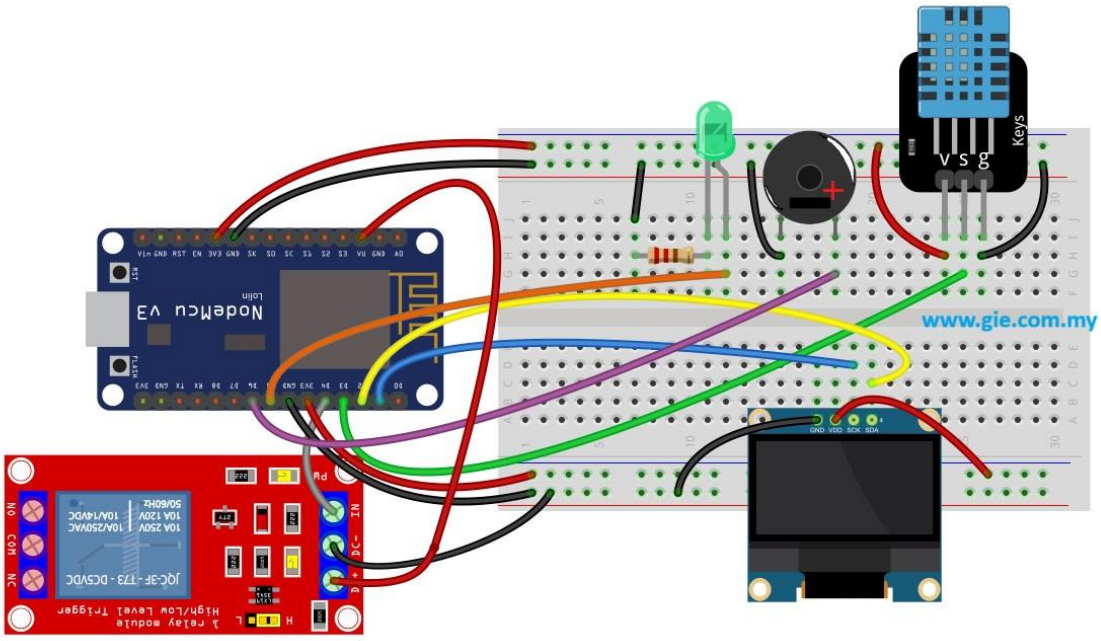
```
int relayPin = D4;

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 D3
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

int ledPin = D5;
int buzzerPin = D6;
int relayPin = D4;

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.

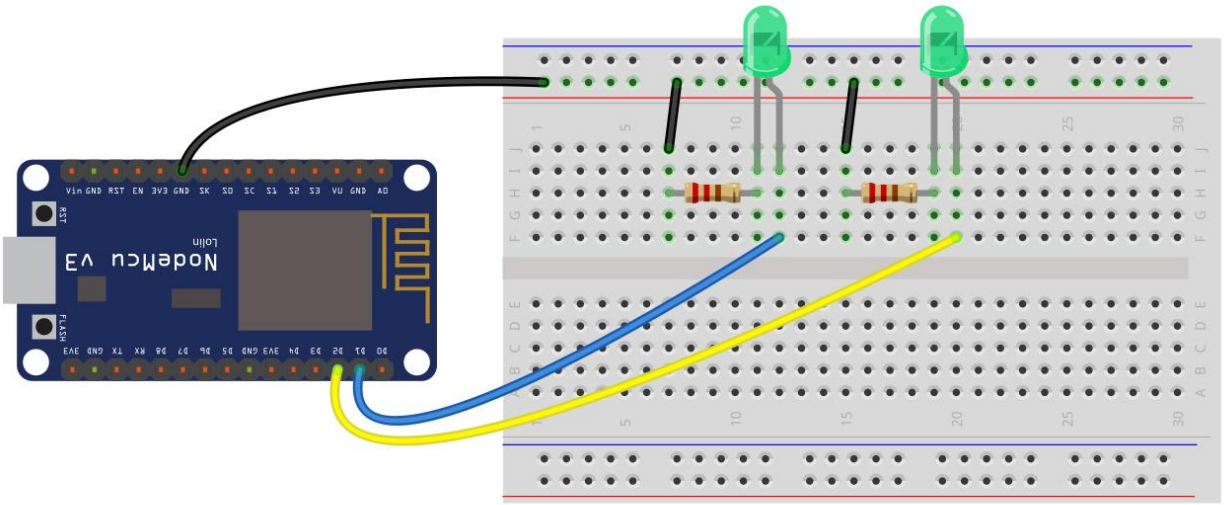
14. NodeMCU Web Server (WiFi)

Wiring:

- Connect the **positive leg** of the 1st LED to **D1** (GPIO 5) on the NodeMCU.
- 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 **D2** (GPIO 4) on the NodeMCU.
- Connect the **negative leg of the 2nd LED** to the **GND** pin via a **220Ω resistor**.

Diagram:



Code:

```
// Load Wi-Fi library
#include <ESP8266WiFi.h>

// Replace with your network credentials
const char* ssid = "xxx_2.4G@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 outputD1State = "off";
String outputD2State = "off";

// Assign output variables to GPIO pins
```

```
const int outputD1 = D1;
const int outputD2 = D2;

// 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(outputD1, OUTPUT);
  pinMode(outputD2, OUTPUT);
  // Set outputs to LOW
  digitalWrite(outputD1, LOW);
  digitalWrite(outputD2, 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 /D1/on") >= 0) {
           Serial.println("D1 on");
           outputD1State = "on";
           digitalWrite(outputD1, HIGH);
         } else if (header.indexOf("GET /D1/off") >= 0) {
           Serial.println("D1 off");
           outputD1State = "off";
           digitalWrite(outputD1, LOW);
         } else if (header.indexOf("GET /D2/on") >= 0) {
           Serial.println("D2 on");
           outputD2State = "on";
           digitalWrite(outputD2, HIGH);
         } else if (header.indexOf("GET /D2/off") >= 0) {
           Serial.println("D2 off");
           outputD2State = "off";
           digitalWrite(outputD2, 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>NodeMCU Web Server</h1>");

            // Display current state, and ON/OFF buttons for GPIO 26
            client.println("<p>D1 - State " + outputD1State + "</p>");
            // If the outputD1State is off, it displays the ON button
            if (outputD1State=="off") {
              client.println("<p><a href=\"/D1/on\"><button
class=\"button\">ON</button></a></p>");
            } else {
              client.println("<p><a href=\"/D1/off\"><button class=\"button
button2\">OFF</button></a></p>");
            }

            // Display current state, and ON/OFF buttons for GPIO 27
            client.println("<p>D2 - State " + outputD2State + "</p>");
            // If the outputD2State is off, it displays the ON button
            if (outputD2State=="off") {
              client.println("<p><a href=\"/D2/on\"><button
class=\"button\">ON</button></a></p>");
            } else {
              client.println("<p><a href=\"/D2/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("");
  }
}
```

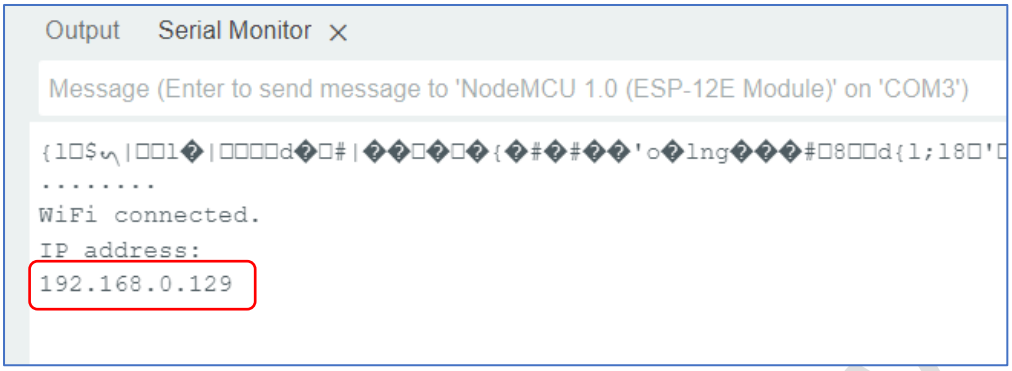
Setting Your Network Credentials:

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";
```

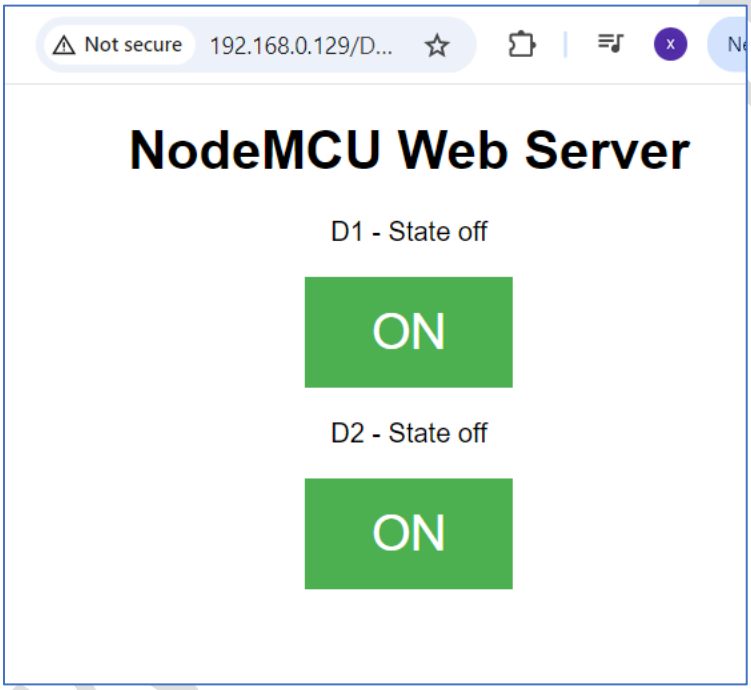
Testing on NodeMCU:

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



- Open your browser, paste the NodeMCU IP address, and you will see the following page.

- Click the buttons to control the LEDs on/off.



15. NodeMCU with Blynk IOT

Step-by-step guide on how to use NodeMCU, 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 the steps at 16. Blynk Setup in details to setup a new device.

Wiring:

a. Connect the LED:

- Anode (long leg) of the LED to D2 (GPIO 4)
- Cathode (short leg) to GND via a 220 ohm resistor.

b. Connect the DHT11 Sensor:

- VCC to 3.3V on NodeMCU.
- GND to GND on NodeMCU.
- Data pin to D1 (GPIO 5) on NodeMCU.

Arduino IDE Setup

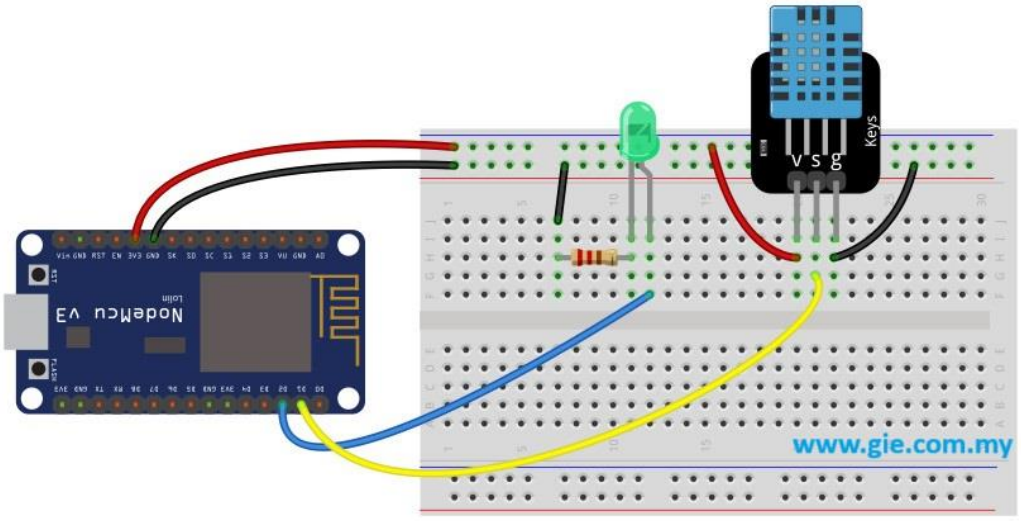
a. Install Blynk Library

- 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 "TMPL625SteWpv"
#define BLYNK_TEMPLATE_NAME "ESP32 DHT11"
#define BLYNK_AUTH_TOKEN "aBgQi1RAOiPqMbeEoMeu92ECy4Exxxx"

// Include necessary libraries
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>
#include "DHT.h"

// DHT11 sensor settings
#define DHTPIN D1     // DHT11 sensor connected to GPIO 5 (D1)
#define DHTTYPE DHT11  // DHT11 sensor type
DHT dht(DHTPIN, DHTTYPE);

// LED connected to D2 (GPIO 4)
int ledPin = D2;

char auth[] = BLYNK_AUTH_TOKEN;
char ssid[] = "xxx2.4G@unifi";  // Your WiFi SSID (name)
char pass[] = "xxx123";  // Your WiFi password

BlynkTimer timer;  // Create a timer object

// Function to send sensor data to Blynk
```

```
void sendSensorData() {
  float humidity = dht.readHumidity();  // Read humidity
  float temperature = dht.readTemperature();  // Read temperature

  // Check if the readings are valid
  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Send temperature and humidity to Blynk
  Blynk.virtualWrite(V1, temperature);  // Display temperature on Virtual Pin V1
  Blynk.virtualWrite(V2, humidity);  // Display humidity on Virtual Pin V2
}

// Blynk function to control LED via virtual pin V0
BLYNK_WRITE(V0) {
  int ledState = param.asInt();  // Get the value from the Blynk app
  digitalWrite(ledPin, ledState);  // Turn the LED ON or OFF
}

void setup() {
  // Begin serial communication
  Serial.begin(115200);

  // Setup Blynk connection
  Blynk.begin(auth, ssid, pass);

  // Setup the LED pin as output
  pinMode(ledPin, OUTPUT);

  // Start the DHT sensor
  dht.begin();

  // Set up a function to run every 2 seconds
  timer.setInterval(2000L, sendSensorData);
}

void loop() {
  Blynk.run();  // Run Blynk
  timer.run();  // Run the timer to send sensor data every 2 seconds
}
```

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.
- LED Control: The BLYNK_WRITE(V0) function allows you to control the LED using a button in the Blynk app (linked to Virtual Pin V0).
- DHT11 Sensor Readings: The sendSensorData() function reads temperature and humidity values and sends them to the app via Virtual Pins V1 and V2.
- Timer: A timer ensures that the temperature and humidity readings are sent to Blynk every 2 seconds.

Upload the Code to NodeMCU

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

b. In the Arduino IDE, select the correct board and port.
   - Tools > Board > NodeMCU 1.0 (ESP-12E Module).
   - Tools > Port > COMxx (or the port your NodeMCU is connected to).

c. Upload the code.

Configure the Blynk App (skip this if already configure)

- **Button Widget:**
  - Select the **Styled Button** widget (created previously)
  - Set the **Pin** to **V0** (for controlling the LED)
  - Set the **Mode** to **Switch**.

- **Temperature Display:**
  - Select the **Labeled Value** widget (created previously).
  - Set the Pin to V1 (for temperature data)

- **Humidity Display:**
  - Select the **Labeled Value** widget (created previously).
  - Set the Pin to V2 (for humidity data)

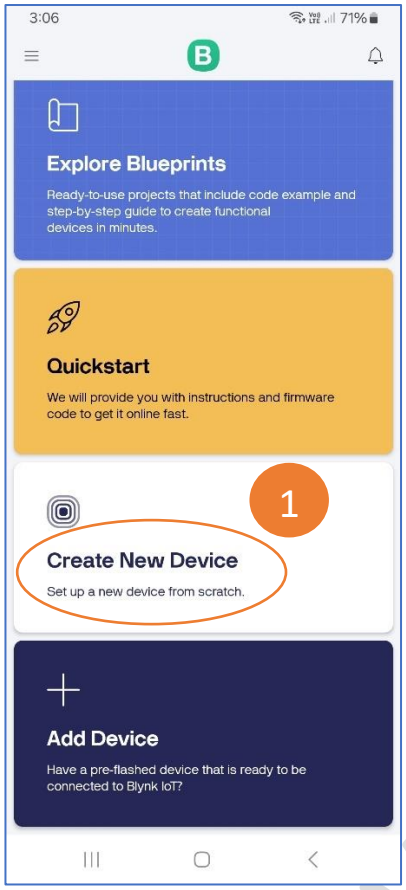
Test the Project

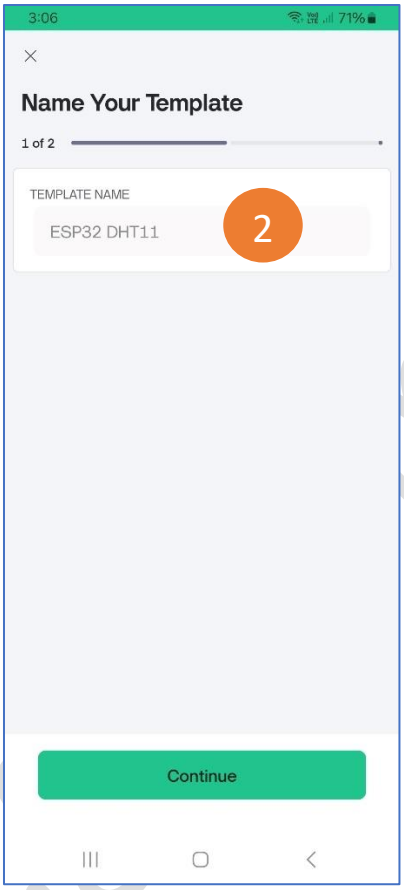
- Open the Blynk app on your smartphone.
- Press the Play button to run the project.
- You should now be able to:
    - Monitor temperature and humidity in real-time.
    - Control the LED by toggling the button in the app.

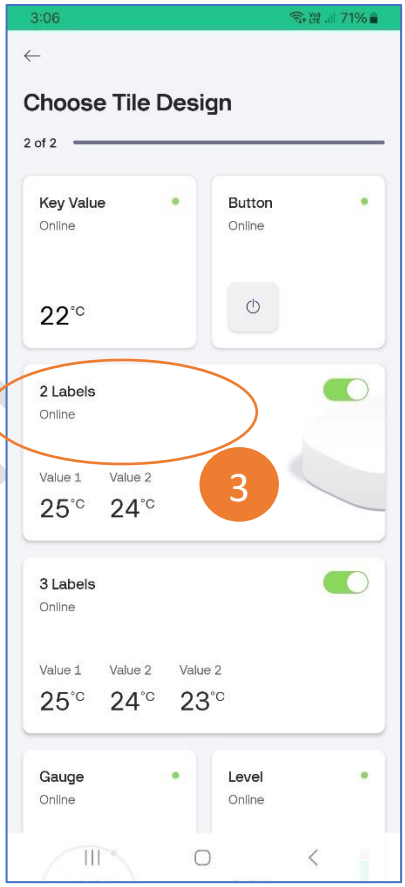
16. Blynk Setup in details

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

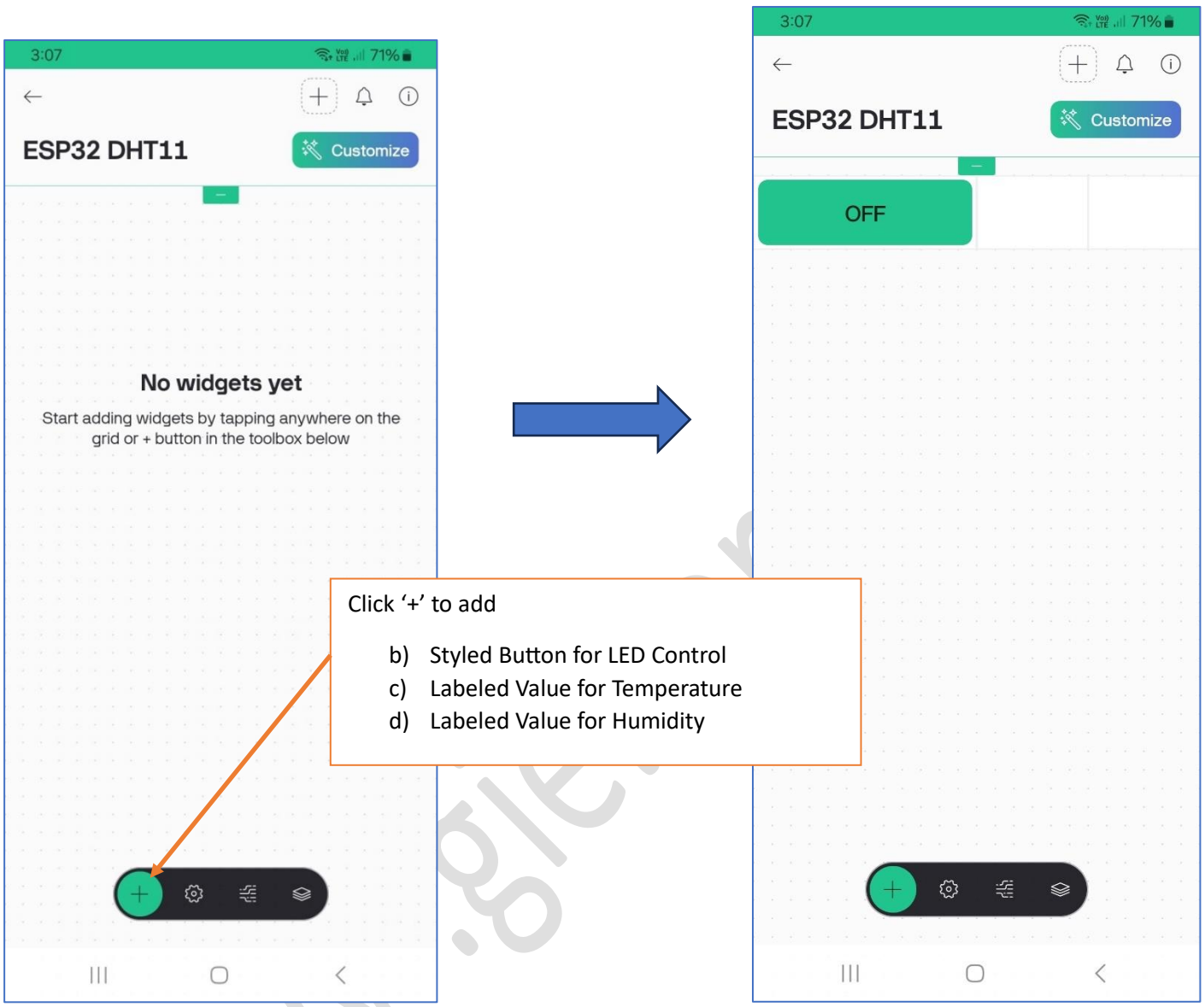
a) Create new Template



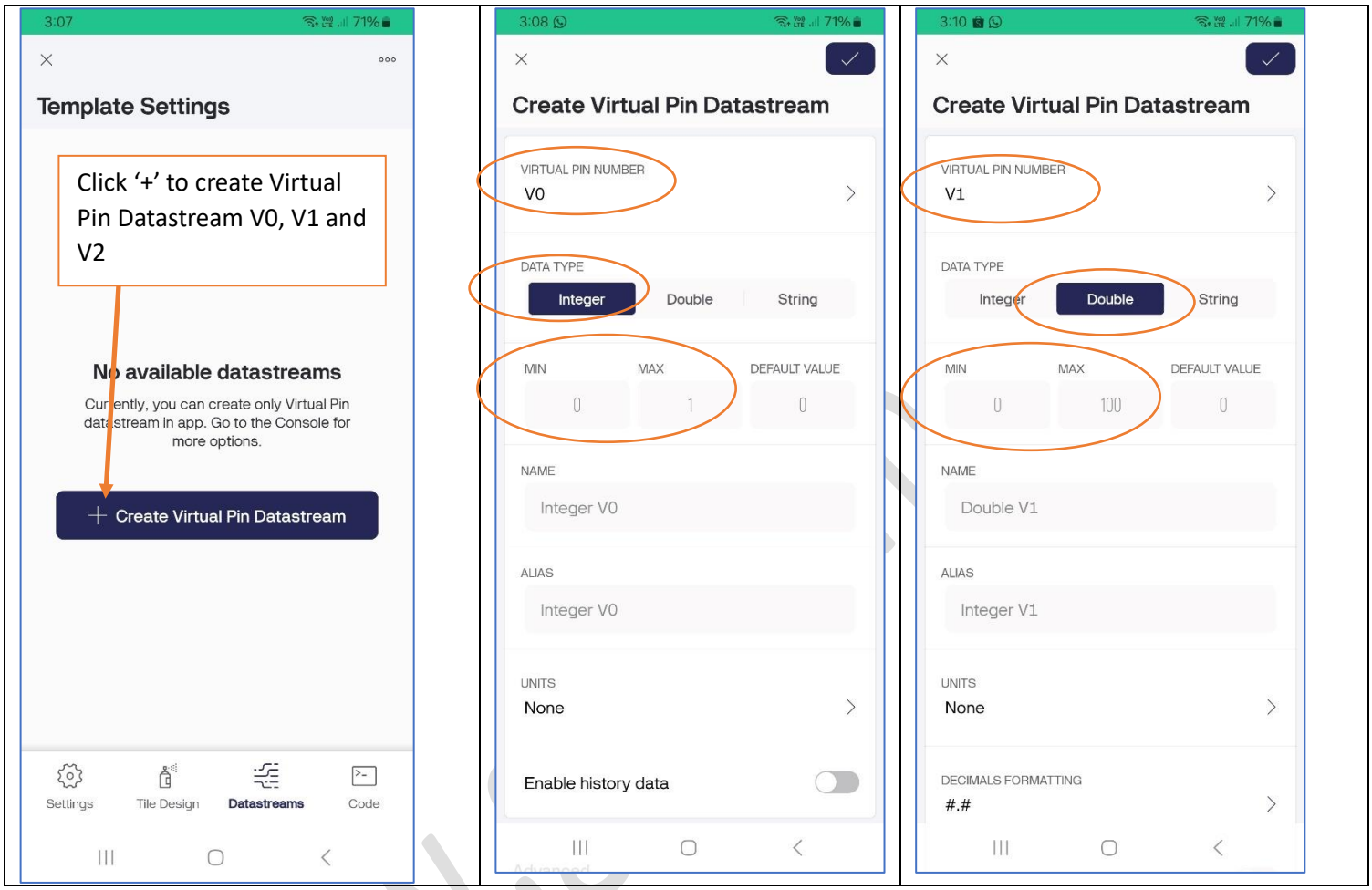


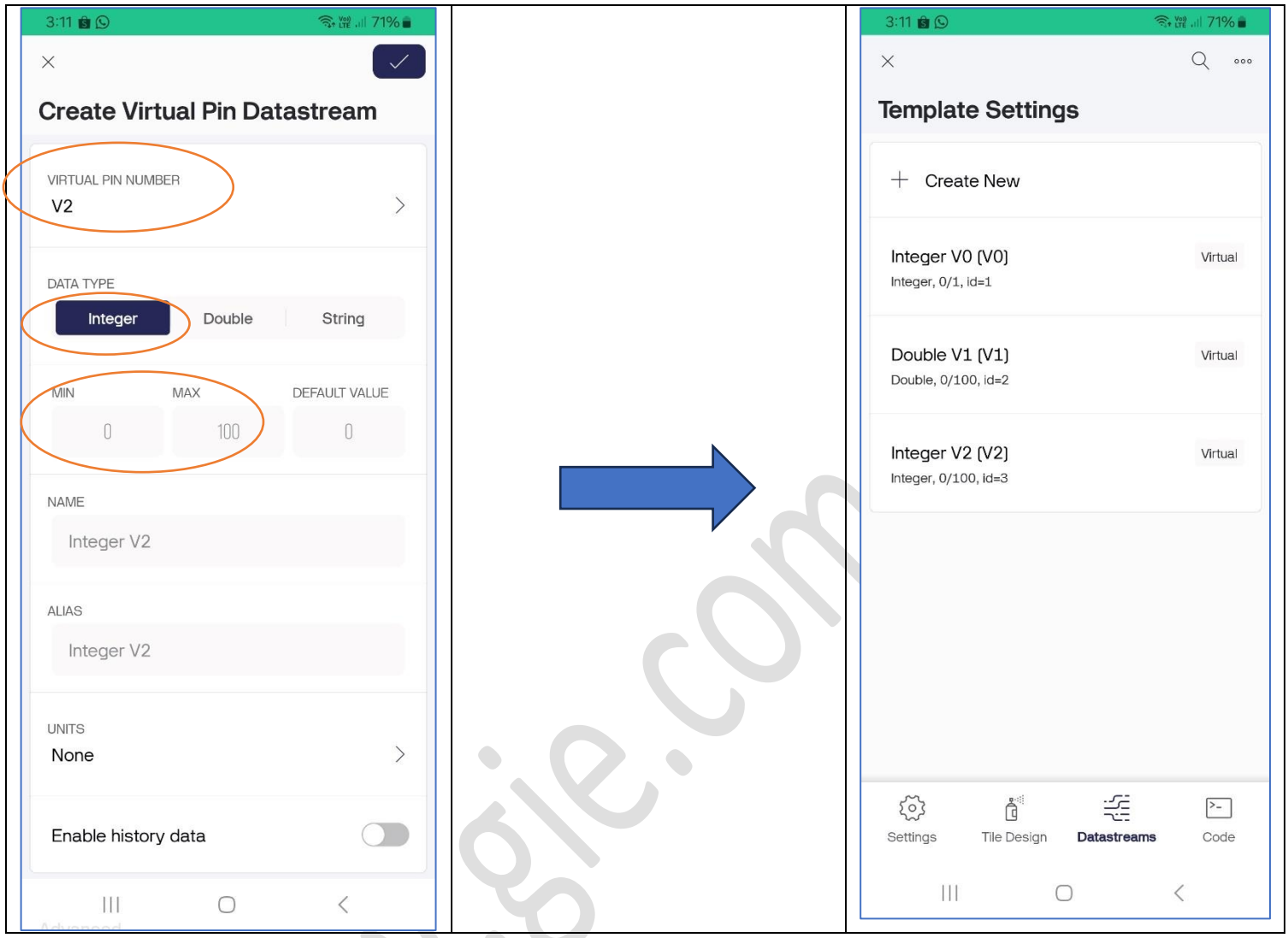


b) Add Widgets

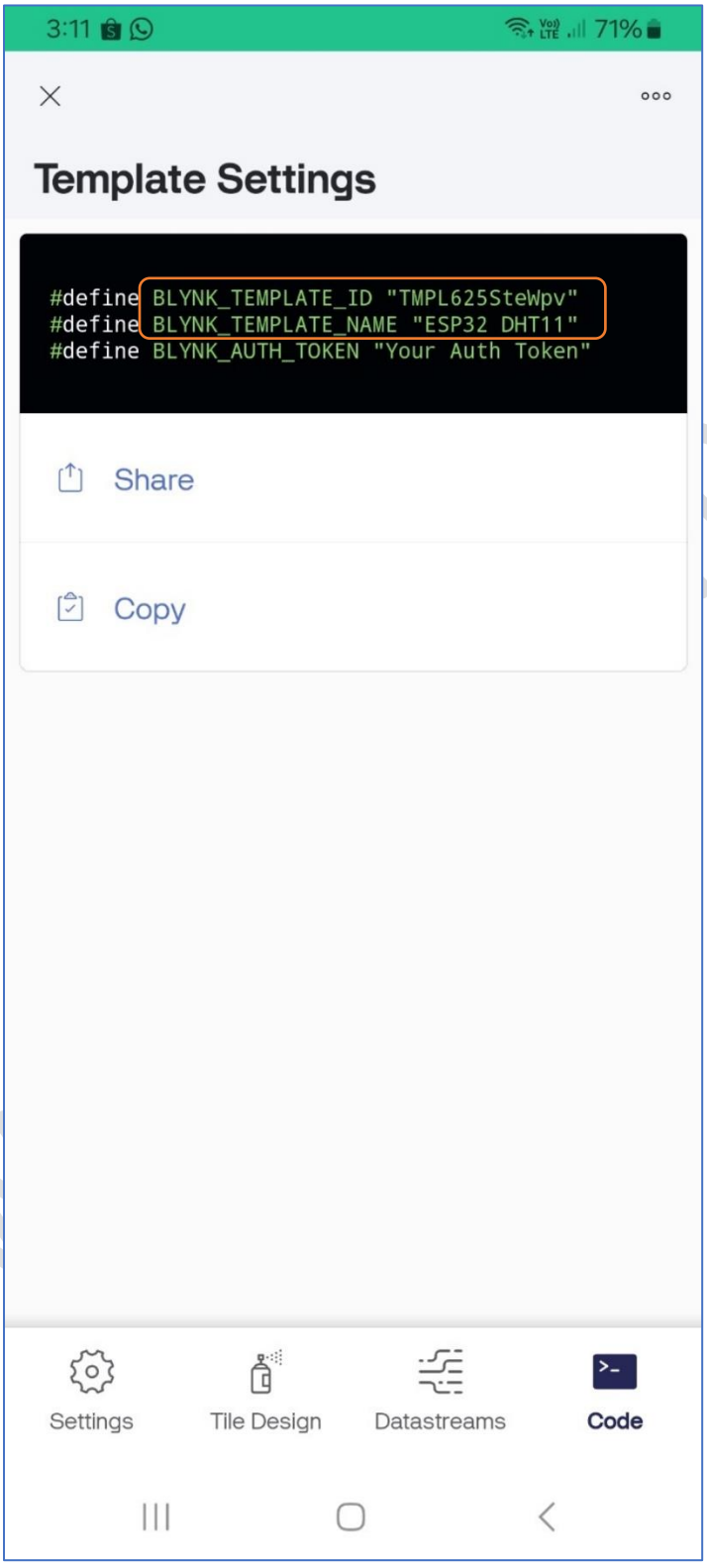

3:07
71%
ESP32 DHT11
Customize
No widgets yet
Start adding widgets by tapping anywhere on the grid or + button in the toolbox below
Click '+' to add
b) Styled Button for LED Control
c) Labeled Value for Temperature
d) Labeled Value for Humidity
3:07
71%
ESP32 DHT11
Customize
OFF

c) Create Virtual Pin Datastream

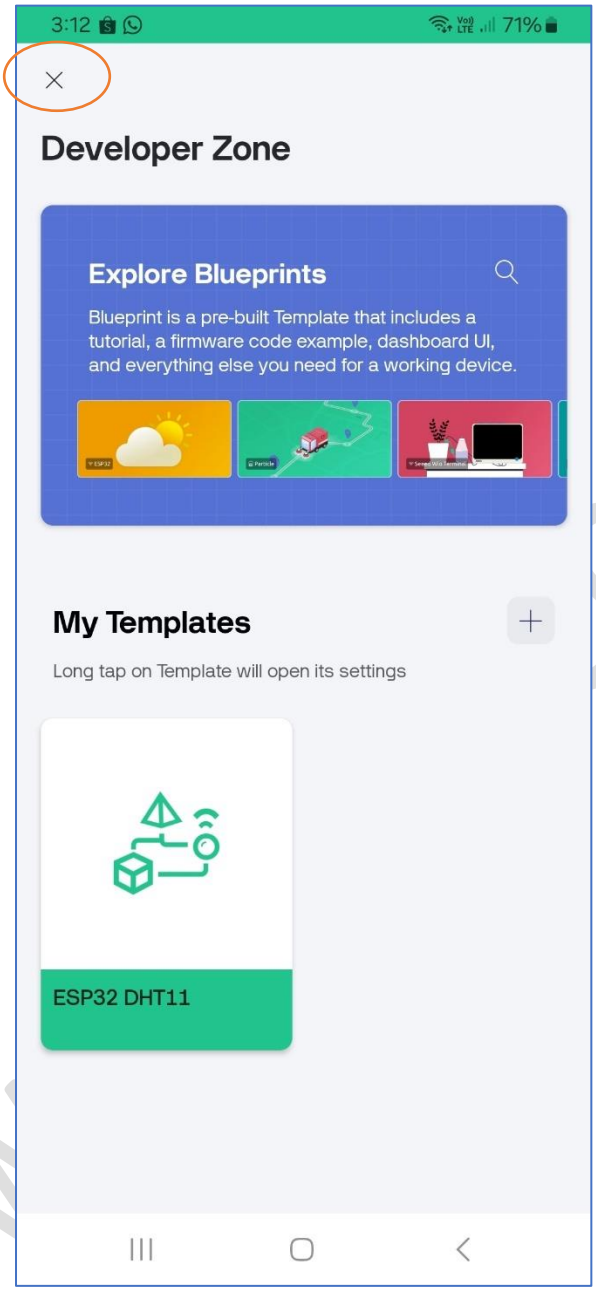



Create Virtual Pin Datastream
VIRTUAL PIN NUMBER
V2
DATA TYPE
Integer
Double
String
MIN
MAX
DEFAULT VALUE
0
100
0
NAME
Integer V2
ALIAS
Integer V2
UNITS
None
Enable history data
Advanced
Template Settings
Create New
Integer V0 (V0)
Integer, 0/1, id=1
Virtual
Double V1 (V1)
Double, 0/100, id=2
Virtual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

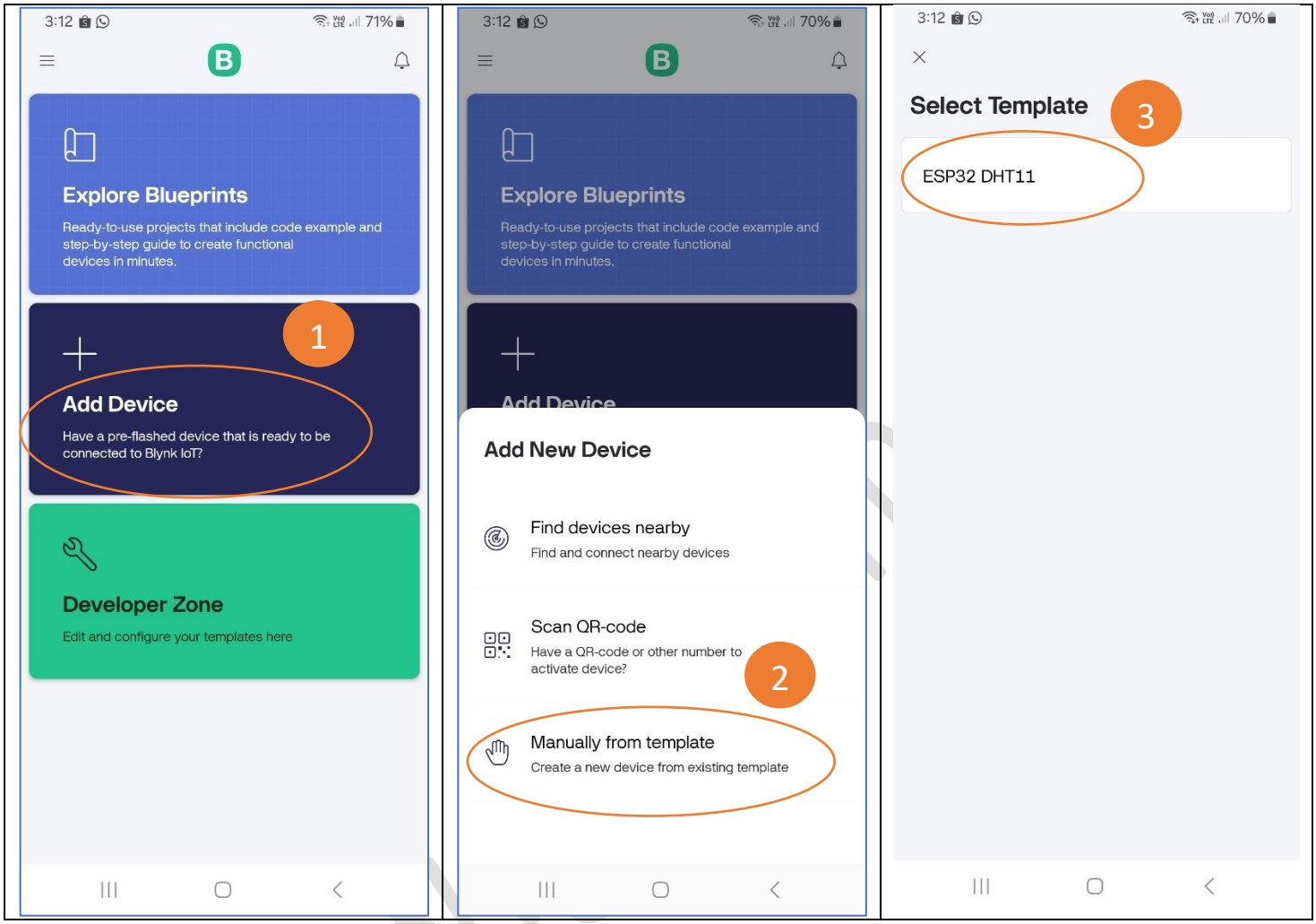
d) Record down **BLYNK_TEMPLATE_ID** and **BLYNK_TEMPLATE NAME**, update these 2 data into code.

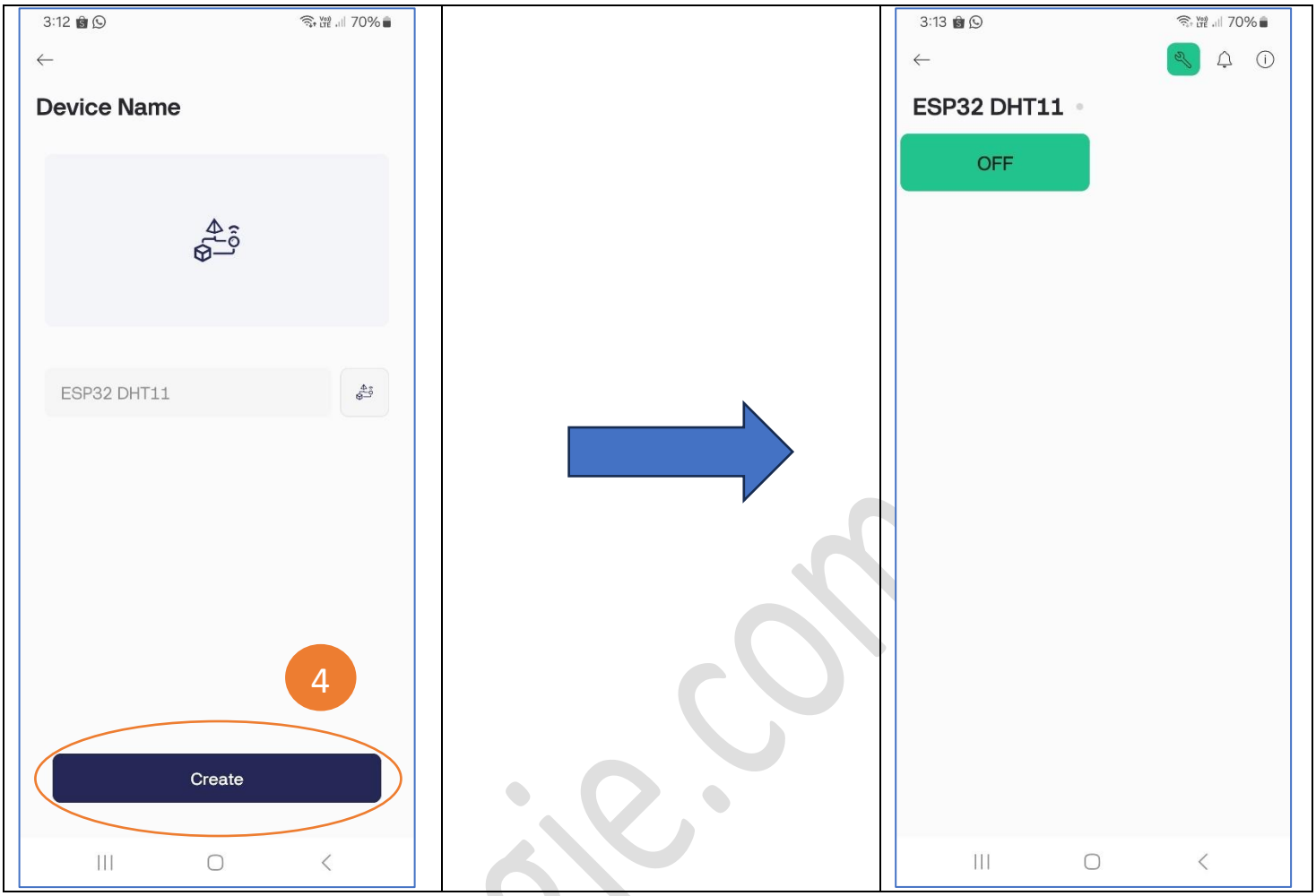


e) Exit from the template setting menu until go into Developer Zone screen. Then click 'X' exit from the Developer Zone.

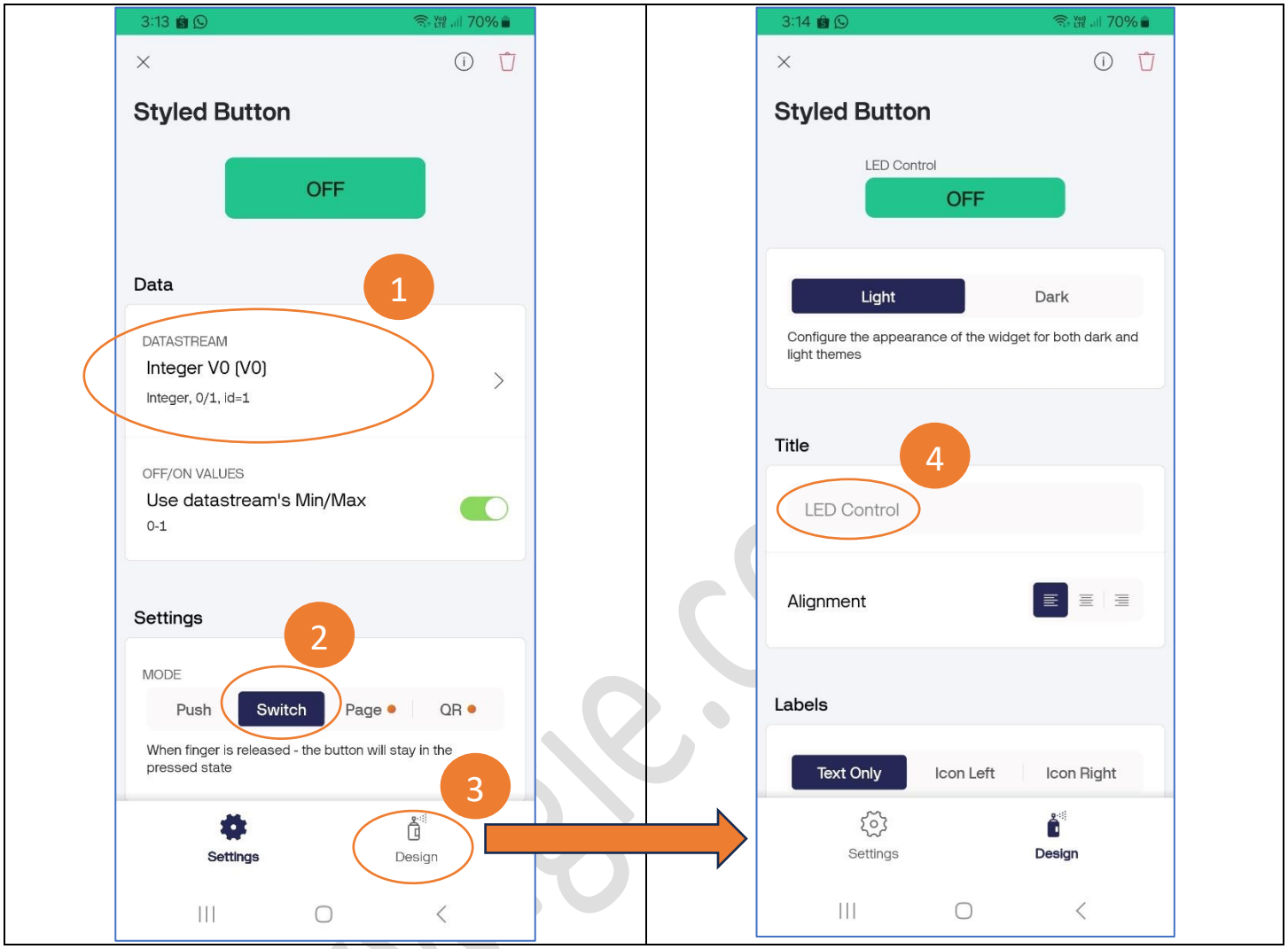


f) Add Device

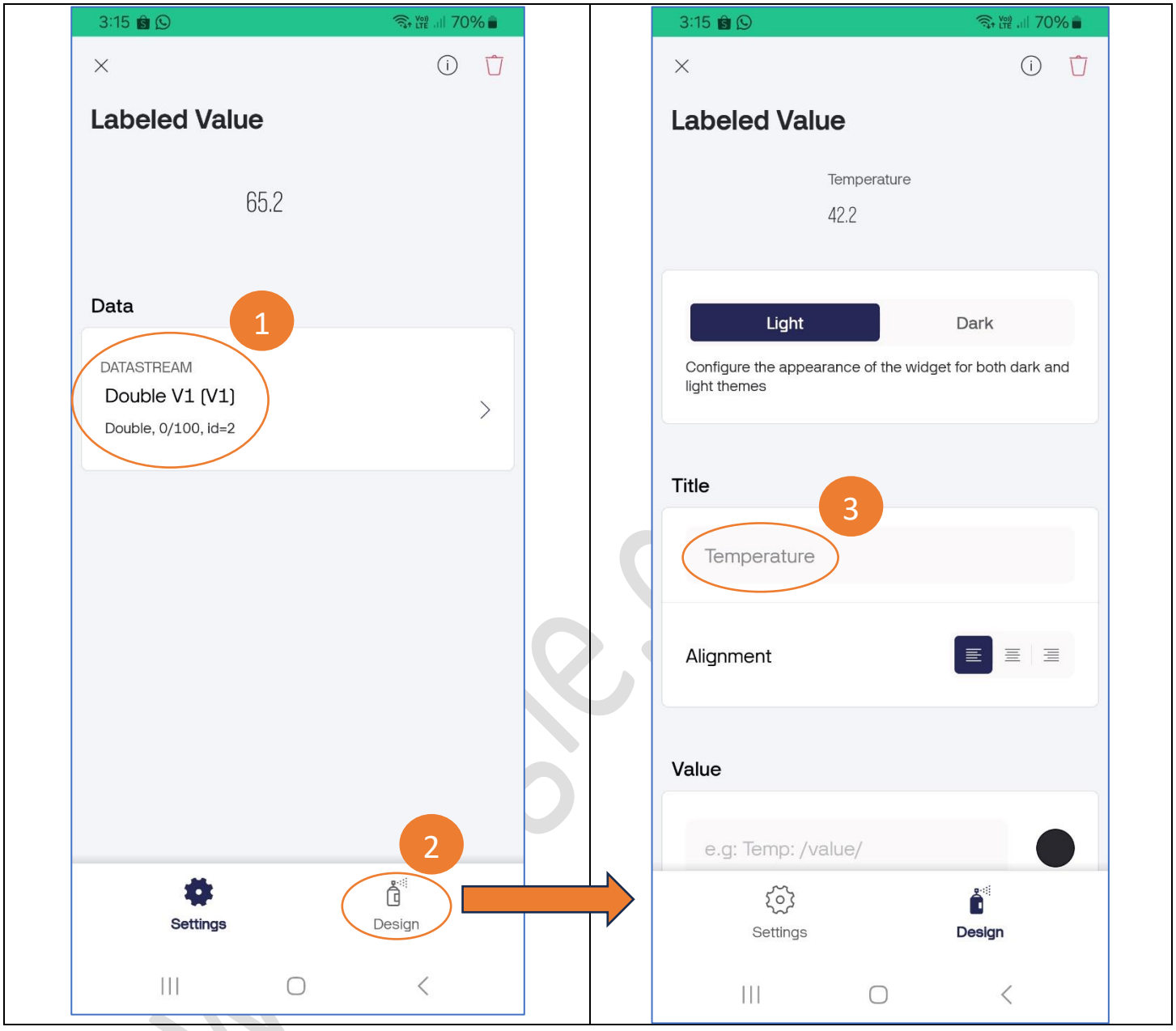

3:12
71%
Explore Blueprints
Ready-to-use projects that include code example and step-by-step guide to create functional devices in minutes.
1
Add Device
Have a pre-flashed device that is ready to be connected to Blynk IoT?
Developer Zone
Edit and configure your templates here
3:12
70%
Explore Blueprints
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

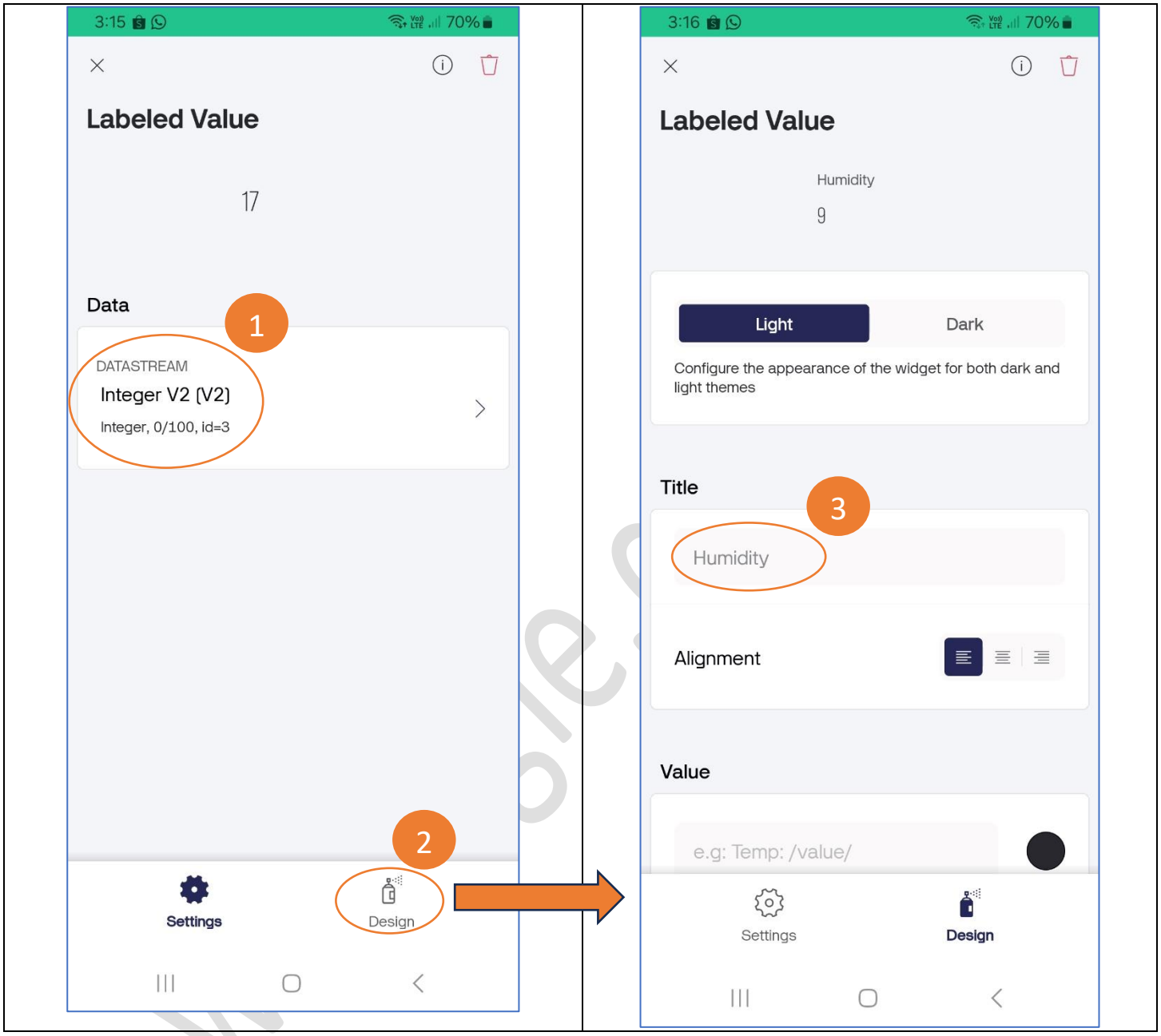
g) Click on Styled Button then configure it as below.



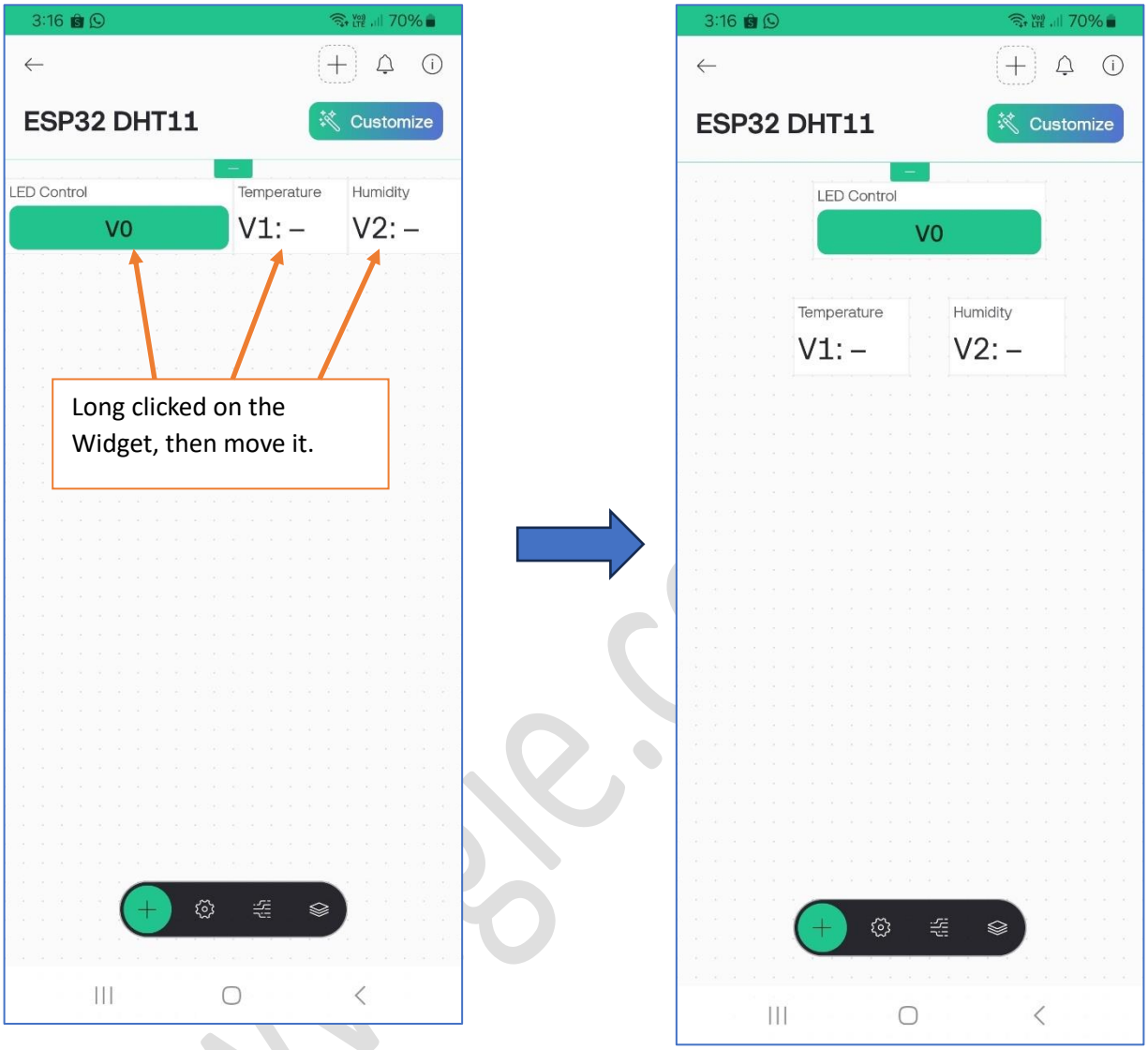
h) Click on Label Display for Temperature then configure it as below.



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.