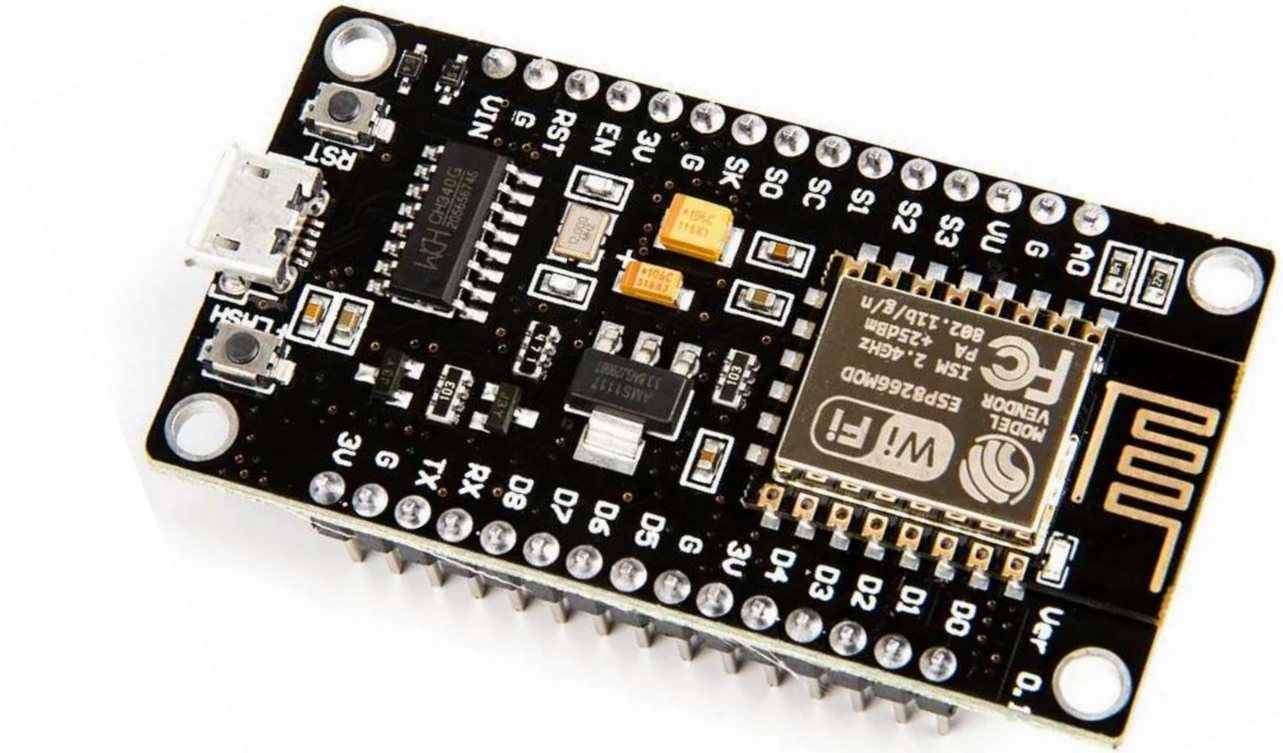
Internet of Things

NodeMCU as Web Server

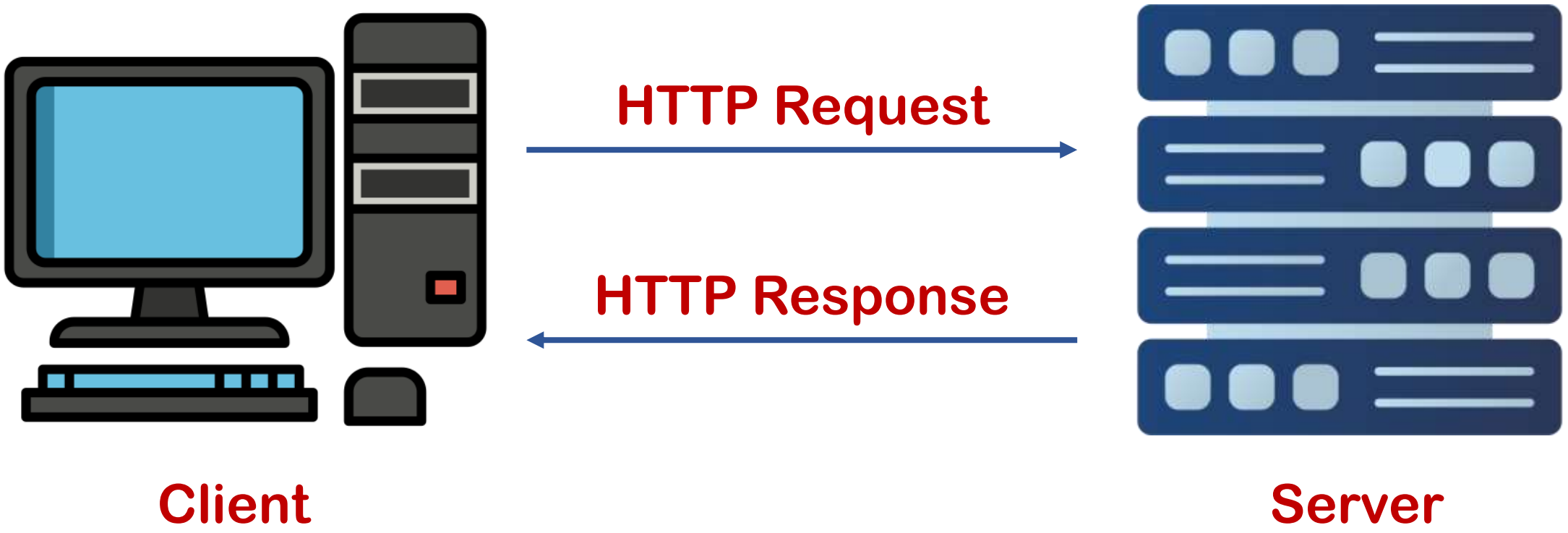
IoT Team, BFCAI

NodeMCU ESP8266

- NodeMCU is a low-cost open-source IoT platform based on the ESP8266 Wi-Fi system on a chip.
- NodeMCU runs on the ESP-12 module.

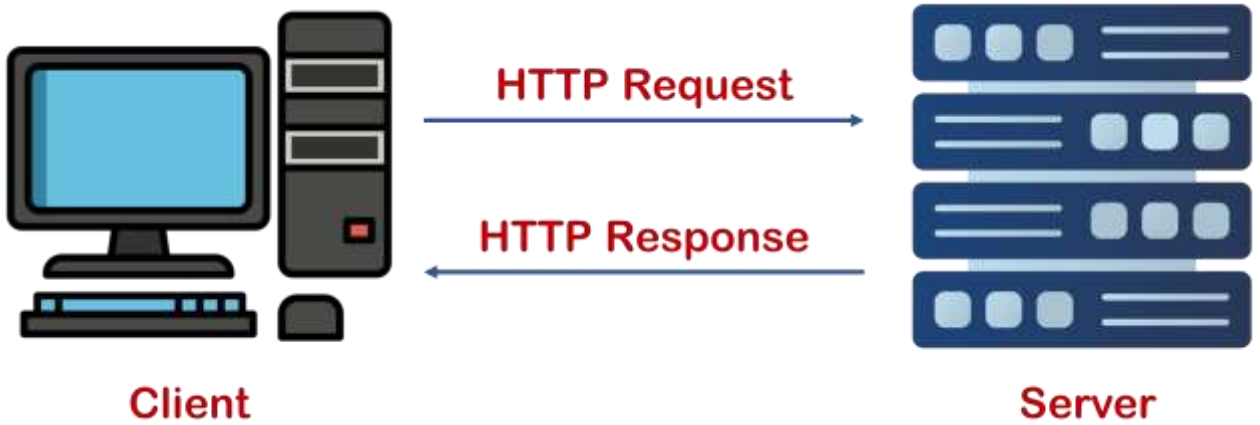


Web Server



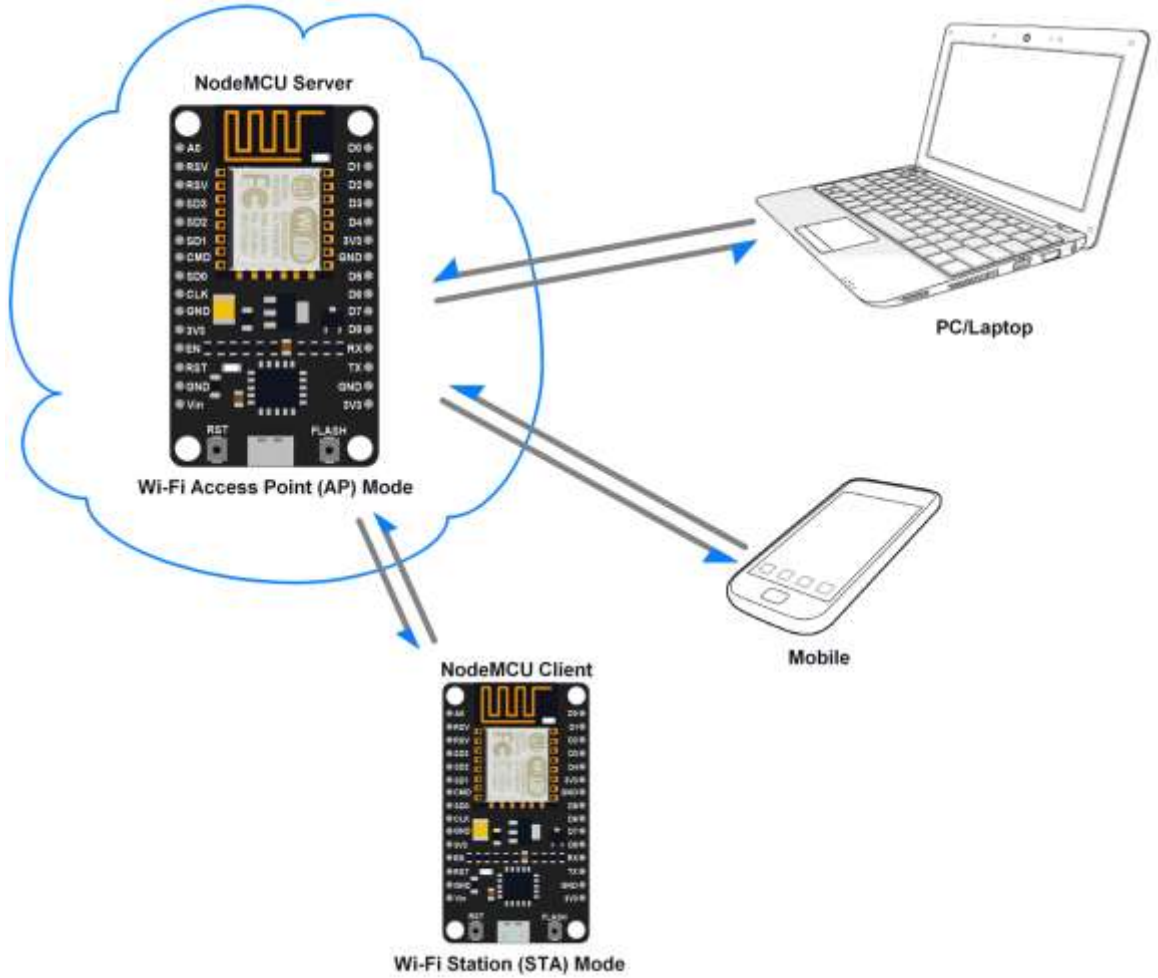
Web Server

- A web server is a place where web pages are stored.
- A web client is just a web browser that we use on our computers.
- A web client and a web server communicate using a special protocol known as Hypertext Transfer Protocol (HTTP).
- In this protocol, a client sending an HTTP request for a specific web page.
- The server then sends back the content of that web page or an error message if it can't find it (like the famous 404 Error).



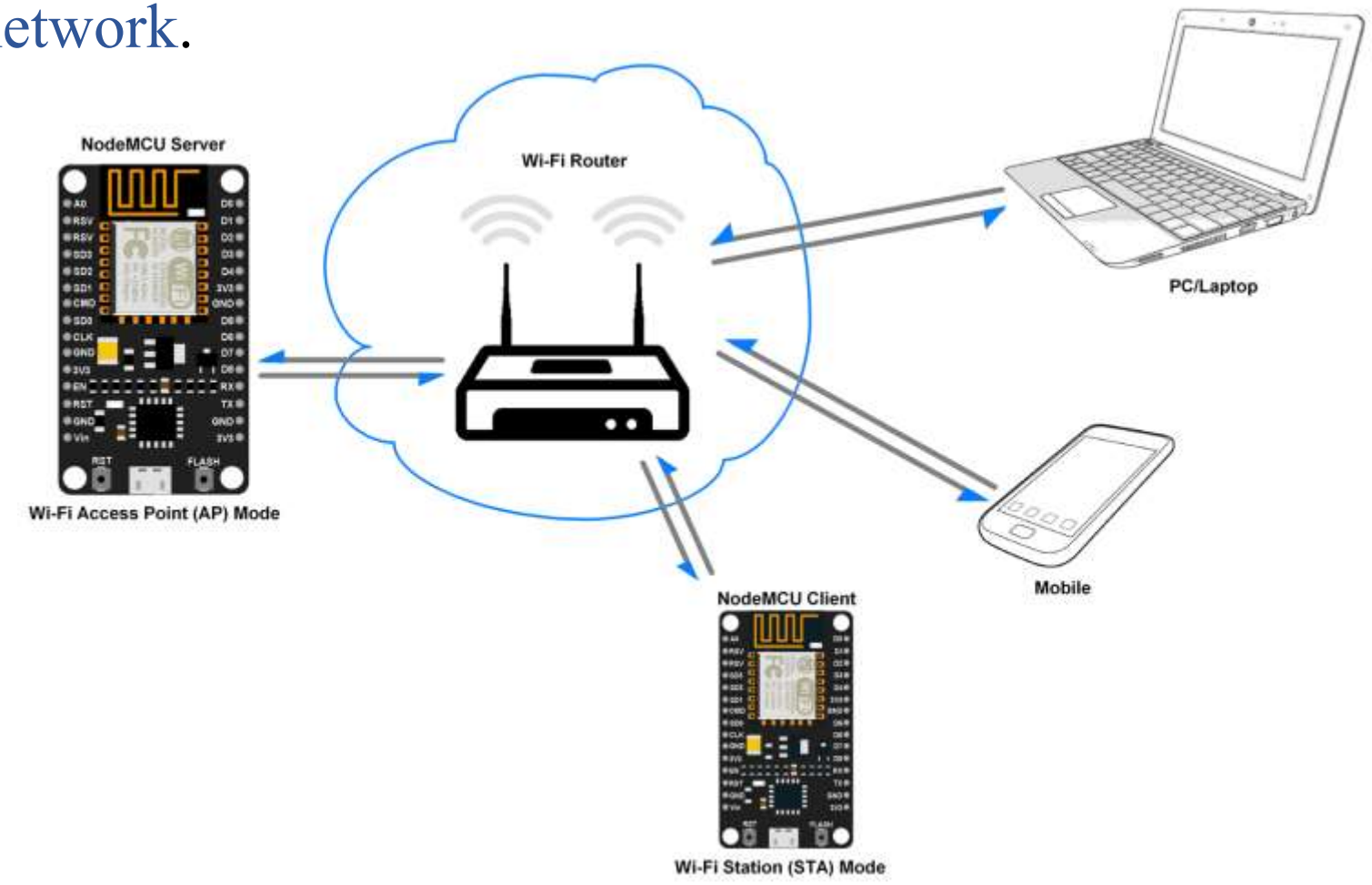
NodeMCU as HTTP Server using Wi-Fi AP Mode

- NodeMCU Wi-Fi has Access Point (AP) mode through which it can create wireless LAN to which any Wi-Fi enabled device can connect.



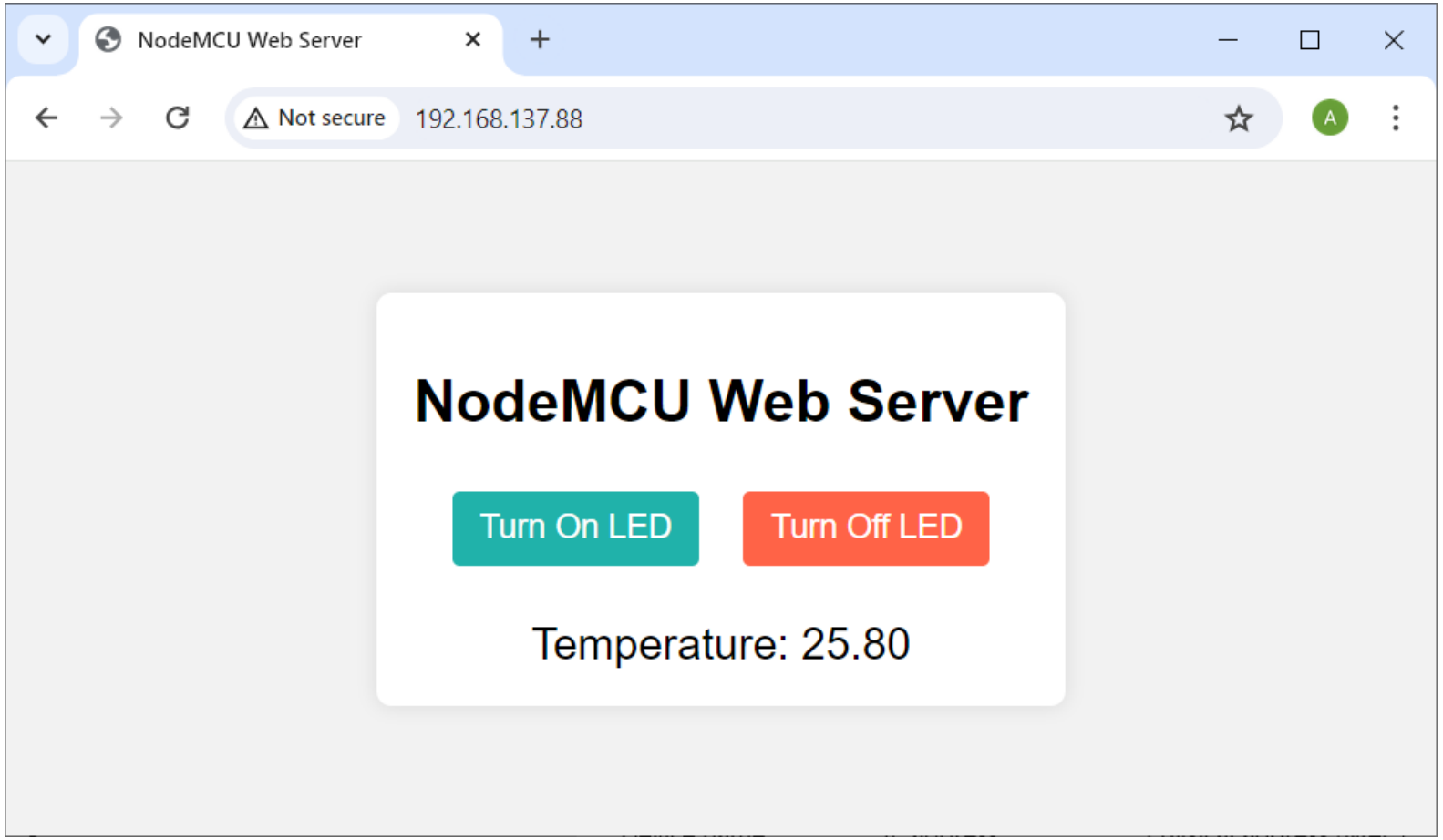
NodeMCU as HTTP Server using Wi-Fi STA Mode

- NodeMCU has Station (STA) mode using which it can connect to the existing Wi-Fi network and can act as a server with an IP address assigned by that network.



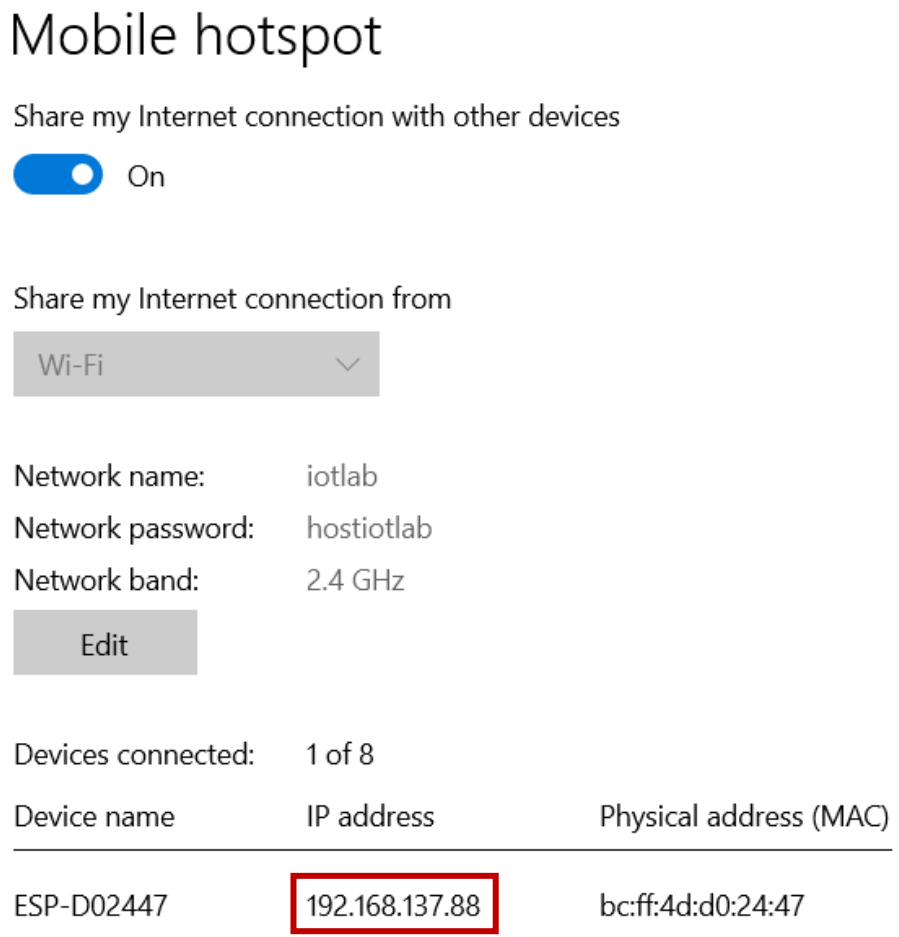
NodeMCU as Web Server

- We want to build a simple webpage to turn on/off an LED and display the temperature value.



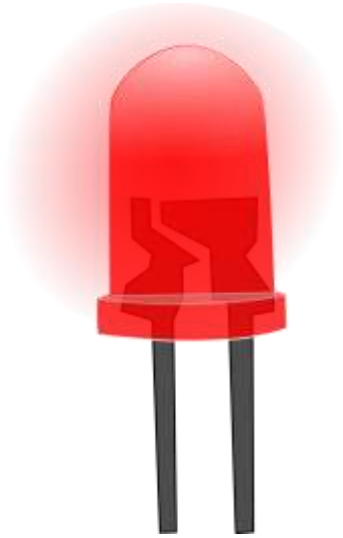
NodeMCU as Web Server: Connecting to Wi-Fi

- When the NodeMCU is connected to a network, it gets an IP address.
- With this IP address, it can act as an HTTP server.



NodeMCU as Web Server: Simple Idea

- When the user enters `192.168.137.88/led/on`, the LED turns on.
- When the user enters `192.168.137.88/led/off`, the LED turns off.



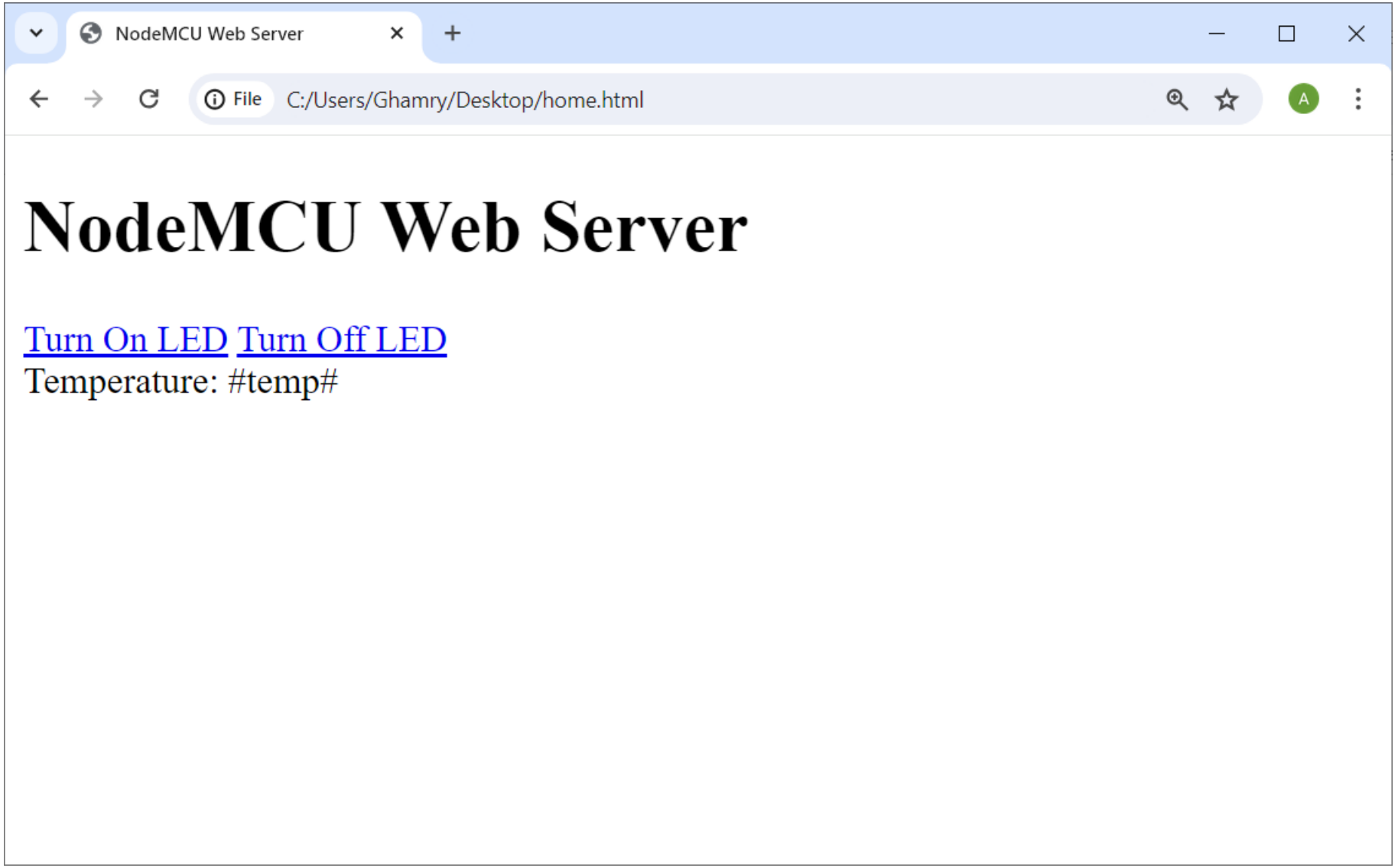
192.168.137.88/led/on

192.168.137.88/led/off

NodeMCU as Web Server: Web Page

```
<!DOCTYPE html>
<html>
<head>
    <title>NodeMCU Web Server</title>
</head>

<body>
    <div>
        <h1>NodeMCU Web Server</h1>
        <a href="/led/on">Turn On LED</a>
        <a href="/led/off">Turn Off LED</a>
        <div>Temperature: #temp#</div>
    </div>
</body>
</html>
```

NodeMCU as Web Server: Web Page



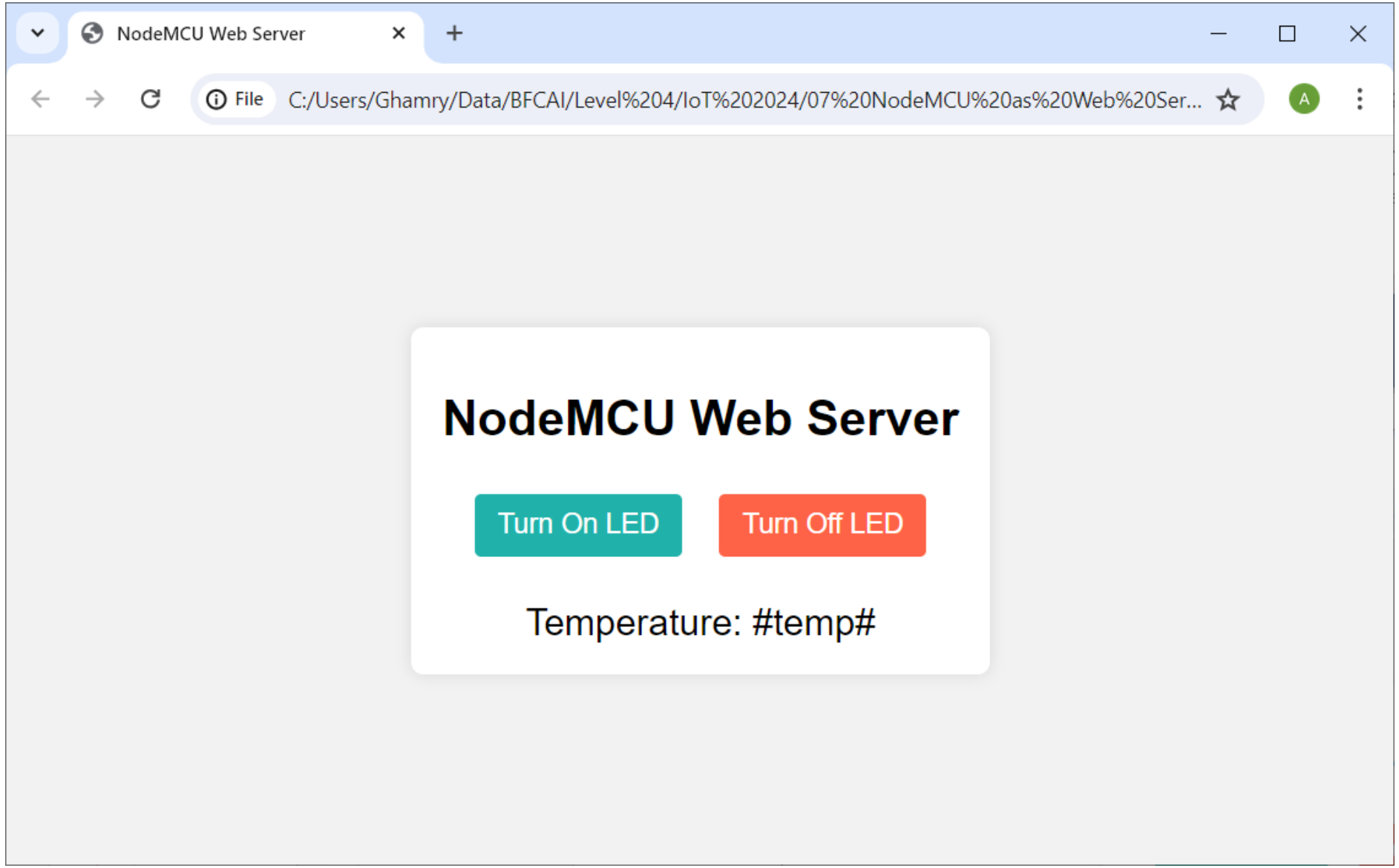
NodeMCU as Web Server: Better Look

```
<!DOCTYPE html>
<html>
<head>
    <title>NodeMCU Web Server</title>
    <style>
        body {
            font-family: Arial;
            background-color: #f2f2f2;
            margin: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
        }
        .container {
            text-align: center;
            padding: 20px;
            border-radius: 8px;
            background-color: #fff;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
        .btn {
            display: inline-block;
            padding: 10px 15px;
            font-size: 18px;
            border: none;
            border-radius: 4px;
            cursor: pointer;
            margin: 10px;
            text-decoration: none;
        }
```

NodeMCU as Web Server: Better Look

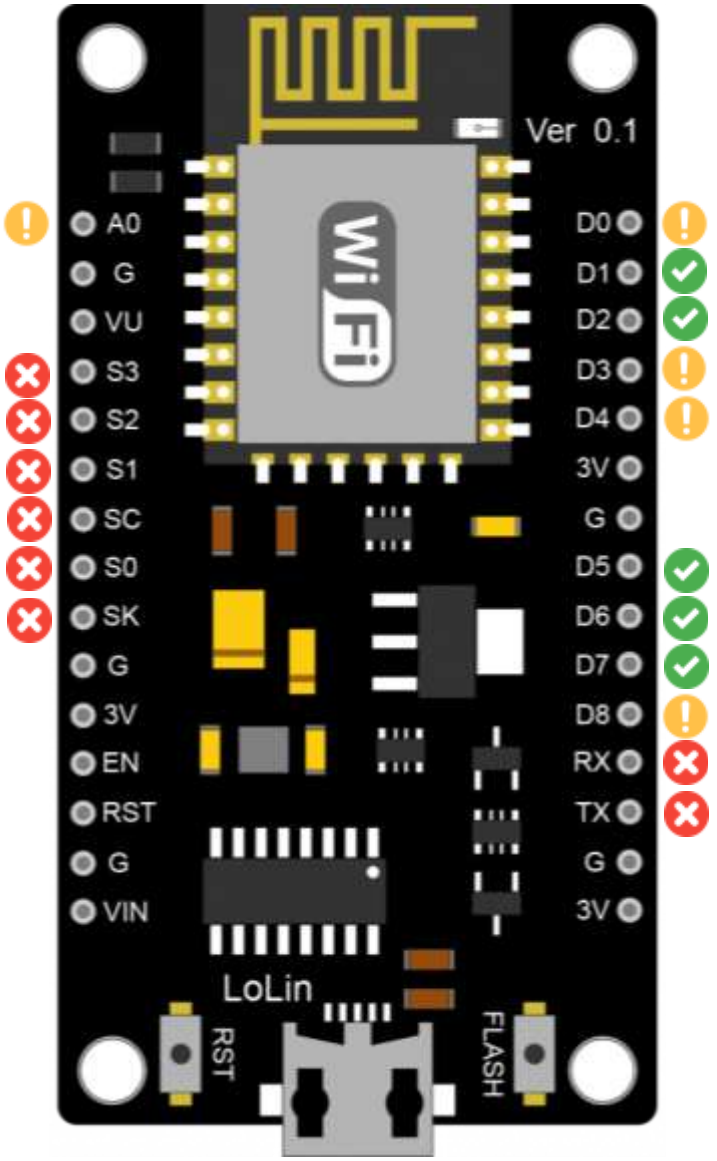
```
        .btn-on {
            background-color: lightseagreen;
            color: white;
        }
        .btn-off {
            background-color: tomato;
            color: white;
        }
        .temp {
            font-size: 24px;
            margin-top: 20px;
        }
    </style>
</head>

<body>
    <div class="container">
        <h1>NodeMCU Web Server</h1>
        <a href="/led/on" class="btn btn-on">Turn On LED</a>
        <a href="/led/off" class="btn btn-off">Turn Off LED</a>
        <div class="temp">Temperature: #temp#</div>
    </div>
</body>
</html>
```

NodeMCU as Web Server: Better Look

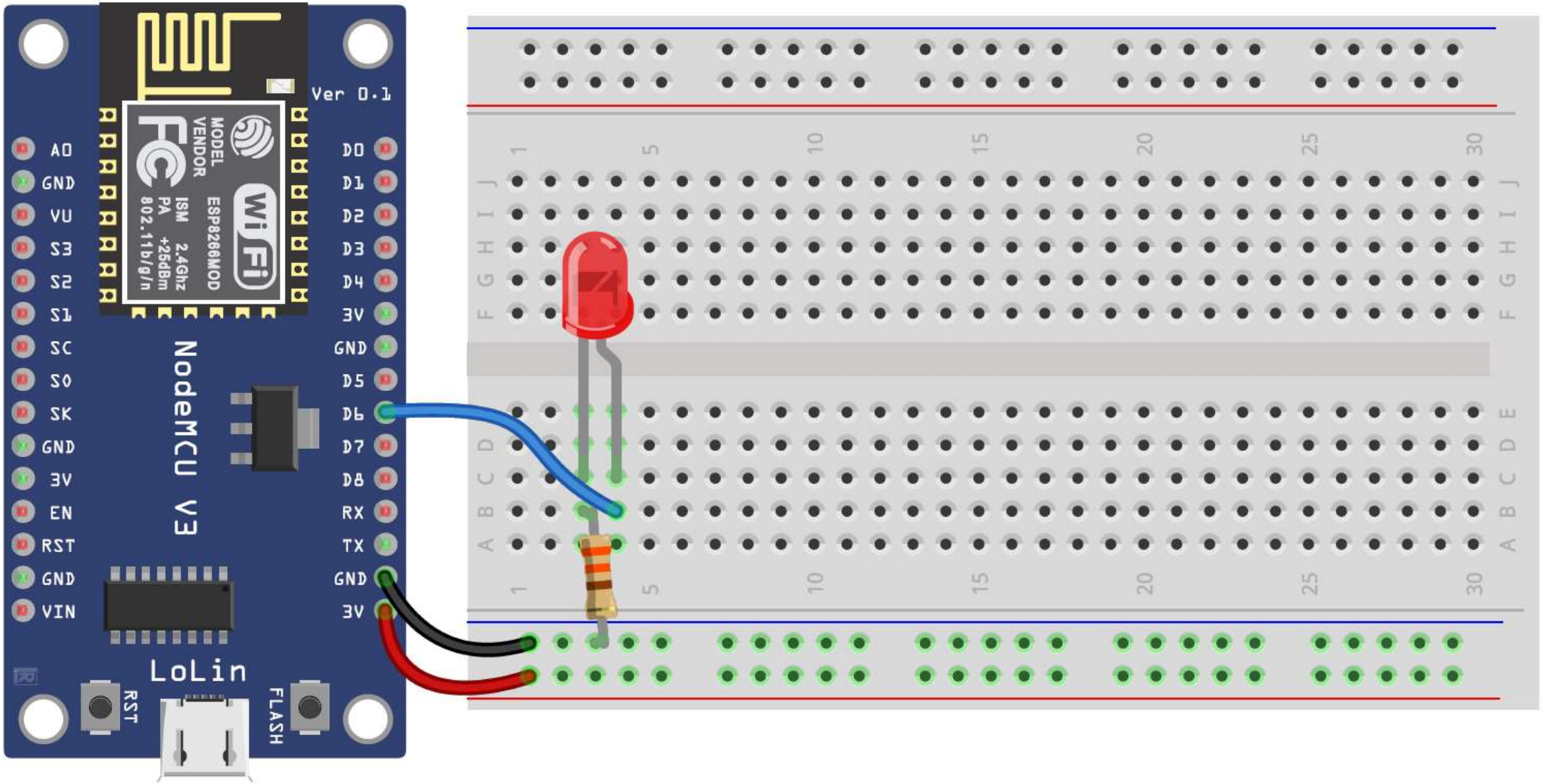


NodeMCU Pinout

PIN	GPIO	Why Not Safe?
D0	GPIO16	HIGH at boot Used to wake up from deep sleep
D1	GPIO5	-
D2	GPIO4	-
D3	GPIO0	Connected to FLASH button Boot fails if pulled LOW
D4	GPIO2	HIGH at boot Boot fails if pulled LOW
D5	GPIO14	-
D6	GPIO12	-
D7	GPIO13	-
D8	GPIO15	Required for boot Boot fails if pulled HIGH



Controlling an LED: Circuit



Controlling an LED: Code Notes

- The `ESP8266WiFi.h` library is for handling Wi-Fi connectivity.

```
#include <ESP8266WiFi.h>
```

- The `ESP8266WebServer.h` library is for creating a web server.

```
#include <ESP8266WebServer.h>
```

- We will define the pin connected to the LED as D6.

```
#define LED_PIN D6
```

- Setting the Wi-Fi network SSID and password.

```
const char* WIFI_SSID = "iotlab";
const char* WIFI_PASS = "hostiotlab";
```

- The variable `ledStatus` is used to keep track of the LED status (1 or 0).

```
bool ledStatus = LOW;
```

Controlling an LED: Code Notes

- An instance of the `ESP8266WebServer` class named `server` is created, listening on port 80 for incoming HTTP requests.

```
ESP8266WebServer server(80);
```

Port	Protocol
20, 21	FTP
22	SSH
23	Telnet
25	SMTP
53	DNS
80	HTTP
443	HTTPS

Controlling an LED: Code Notes

- Serial communication is initiated at a baud rate of 115200 to enable communication with the computer for debugging purposes.

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

- The pin connected to the LED is configured as an output.

```
pinMode(LED_PIN, OUTPUT);
```

- The device attempts to connect to the specified Wi-Fi network using the provided SSID and password.

```
WiFi.begin(WIFI_SSID, WIFI_PASS);
```

- During this process, the program waits until the Wi-Fi connection is established. Once connected, it prints the device's IP address.

Controlling an LED: Code Notes

- HTTP request handlers are defined to specify how the server should respond to different requests, such as turning the LED on or off.

```
server.on("/", handleRoot);
server.on("/led/on", handleLedOn);
server.on("/led/off", handleLedOff);
server.onNotFound(handleNotFound);
```

- Finally, the HTTP server is started, allowing the NodeMCU to handle incoming HTTP requests.

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

Controlling an LED: Code Notes

- In the `loop()` function, the NodeMCU continuously checks for incoming client requests using the `server.handleClient()` function.
- This function is responsible for processing any incoming HTTP requests and generating appropriate responses.

```
void loop() {
  // Handle incoming client requests
  server.handleClient();
}
```

Controlling an LED: Code Notes

- The `handleRoot()` function serves as the HTTP request handler for the root URL ("/").
- The `digitalWrite()` function is used to control the state of the LED pin, setting it either HIGH or LOW to turn the LED on or off.

```
digitalWrite(LED_PIN, ledStatus);
```

- After updating the LED status, an HTML response is sent back to the client using the `server.send()` function.

```
server.send(200, "text/html", getHtml());
```

- The response has a status code of 200, indicating success.
- The HTML content to be sent is obtained by calling the `getHtml()` function, which generates the appropriate HTML.

Controlling an LED: Code Notes

- The `handleLedOn()` function serves as the HTTP request handler for turning the LED on.
- The ledStatus is updated to HIGH, indicating that LED will be turned on.

```
ledStatus = HIGH;
digitalWrite(LED_PIN, ledStatus);
```

- An HTML response is sent back to the client using `server.send()`.

```
server.send(200, "text/html", getHtml());
```

- The response has a status code of 200, indicating success.
- The HTML content to be sent is obtained by calling the `getHtml()` function, which generates the appropriate HTML.

Controlling an LED: Code Notes

- The `handleLedOff()` function serves as the HTTP request handler for turning the LED off.
- The ledStatus is updated to LOW, indicating that LED will be turned off.

```
ledStatus = LOW;
digitalWrite(LED_PIN, ledStatus);
```

- An HTML response is sent back to the client using `server.send()`.

```
server.send(200, "text/html", getHtml());
```

- The response has a status code of 200, indicating success.
- The HTML content to be sent is obtained by calling the `getHtml()` function, which generates the appropriate HTML.

Controlling an LED: Code Notes

- In the `handleNotFound()` function, which is the HTTP request handler for when a requested URL is not found, the server sends a 404 Not Found response to the client.
- This response indicates that the requested resource could not be found on the server.

```
void handleNotFound(){
  // Send a 404 Not Found response
  server.send(404, "text/plain", "Not Found");
}
```

Controlling an LED: Code Notes

- The `getHtml()` function generates and returns the HTML content for the web page served by the NodeMCU web server.
- Clicking `<a>` links triggers the `/led/on` or `/led/off` routes.

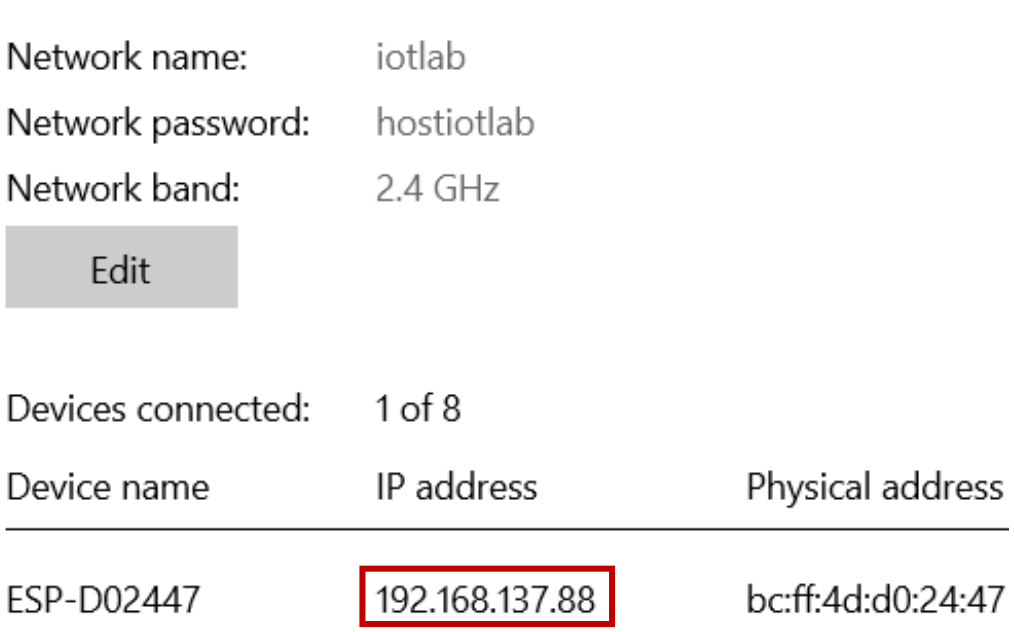
```
<a href="/led/on" class="btn btn-on">Turn On LED</a>
<a href="/led/off" class="btn btn-off">Turn Off LED</a>
```

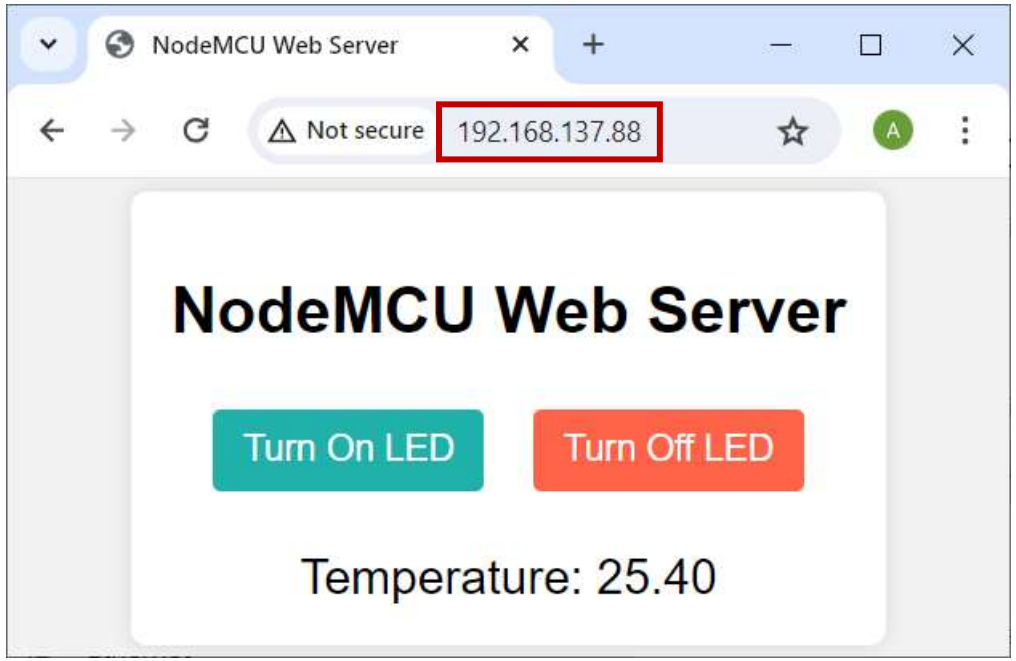
- The temperature value is represented by `#temp#`, which suggests that it's needs to be replaced with the actual temperature value dynamically before sending the HTML response to the client.

```
<div class="temp">Temperature: #temp#</div>
```

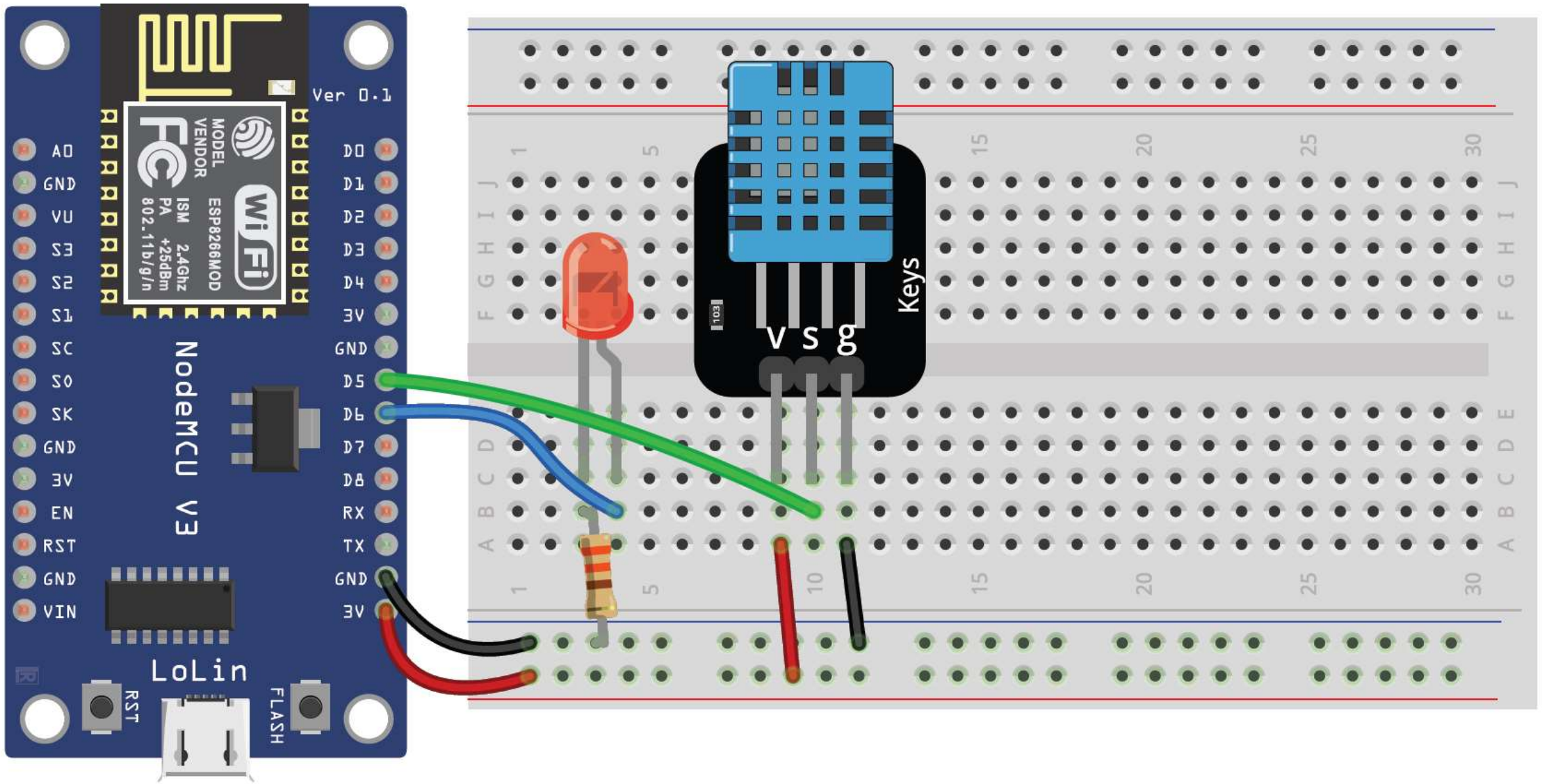
Controlling an LED: Accessing Website

- In Wi-Fi Station (STA) mode, NodeMCU gets IP address from the router (access point).
- If we are also in the same network, then we can directly connect to NodeMCU HTTP server using the IP address only.





Controlling an LED and Reading Temperature: Circuit



Controlling an LED and Reading Temperature: Code Notes

- The directive includes the DHT sensor library, which provides functions for interfacing with DHT sensors.

```
#include "DHT.h"
```

- This line defines the pin connected to the DHT sensor, which is D5.

```
#define DHT_PIN D5
```

- This initializes a DHT11 sensor object named `dht`.

```
DHT dht(DHT_PIN, DHT11);
```

- This declares a variable to store the temperature read from DHT sensor.

```
float temp;
```

- This line starts the DHT sensor, allowing it to begin reading data.

```
dht.begin();
```

Controlling an LED and Reading Temperature: Code Notes

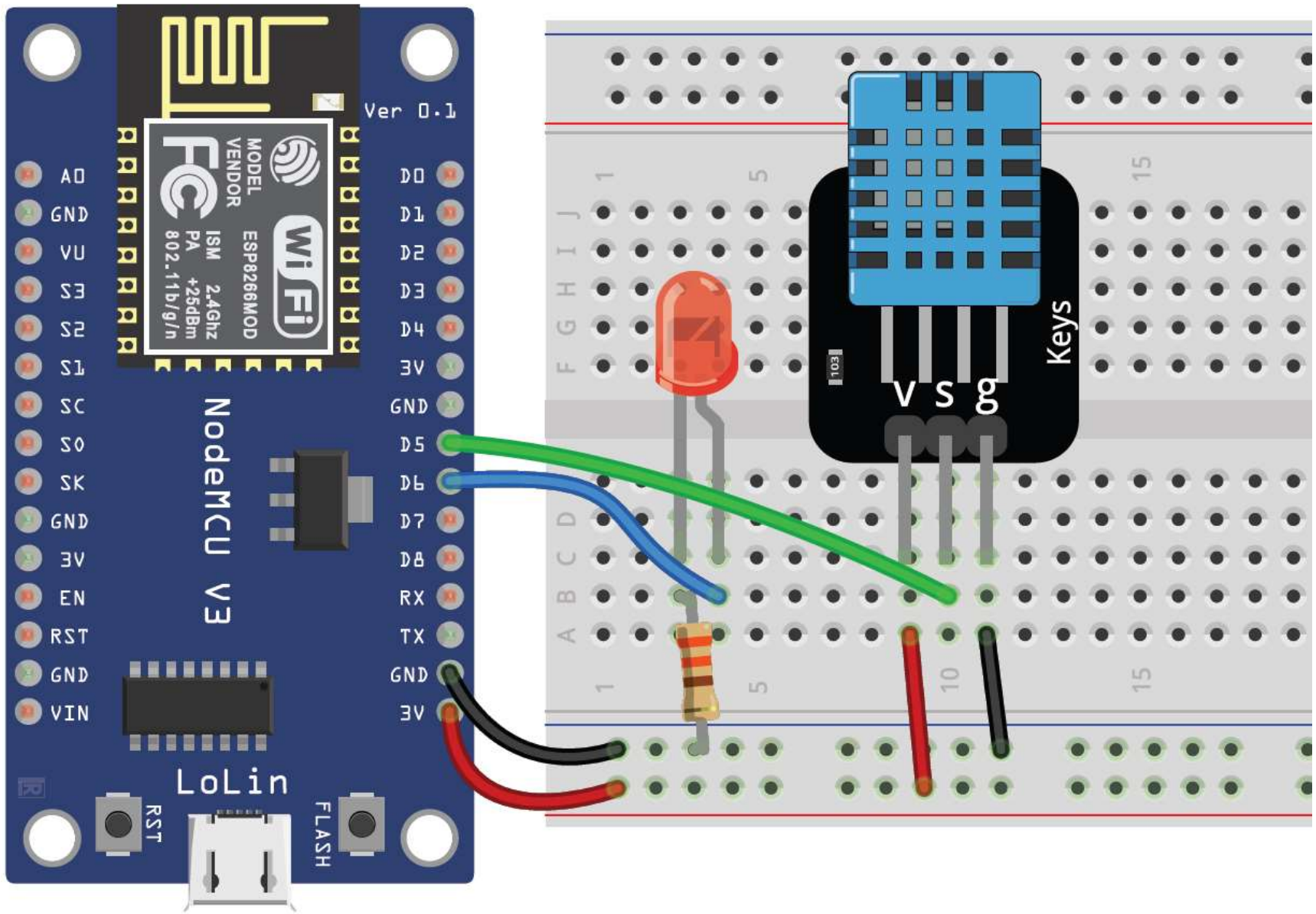
- Inside the handleRoot(), handleLedOn(), and handleLedOff() functions, the temperature is updated.

```
temp = dht.readTemperature();
```

- In the `getHtml()` function, the `#temp#` placeholder in the HTML content is replaced with the actual temperature value.

```
htmlContent.replace("#temp#", String(temp));
```

Controlling an LED, Reading Temperature and Sending to Firebase: Circuit



Controlling an LED, Reading Temperature and Sending to Firebase: Notes

- This directive includes the `FirebaseESP8266` library, which enables integration with Firebase Realtime Database on the ESP8266 platform.

```
#include <FirebaseESP8266.h>
```

- This declares a `FirebaseData` object named `fbdo`, which is used to interact with the Firebase database.

```
FirebaseData fbdo;
```

- This initializes the Firebase connection with the specified host and authentication token.

```
Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
```

Controlling an LED, Reading Temperature and Sending to Firebase: Notes

- This function is responsible for updating the temperature value in the Firebase Realtime Database under the "/temp" path.

```
void updateFirebase(float temp){
  // Set temperature value in the Firebase under the
"/temp" path
  if(Firebase.setFloat(fbdo, "/temp", temp)){
    Serial.print("Temperature: ");
    Serial.println(temp);
  }
  else
    Serial.println(fbdo.errorReason());
}
```

Controlling an LED, Reading Temperature and Sending to Firebase: Notes

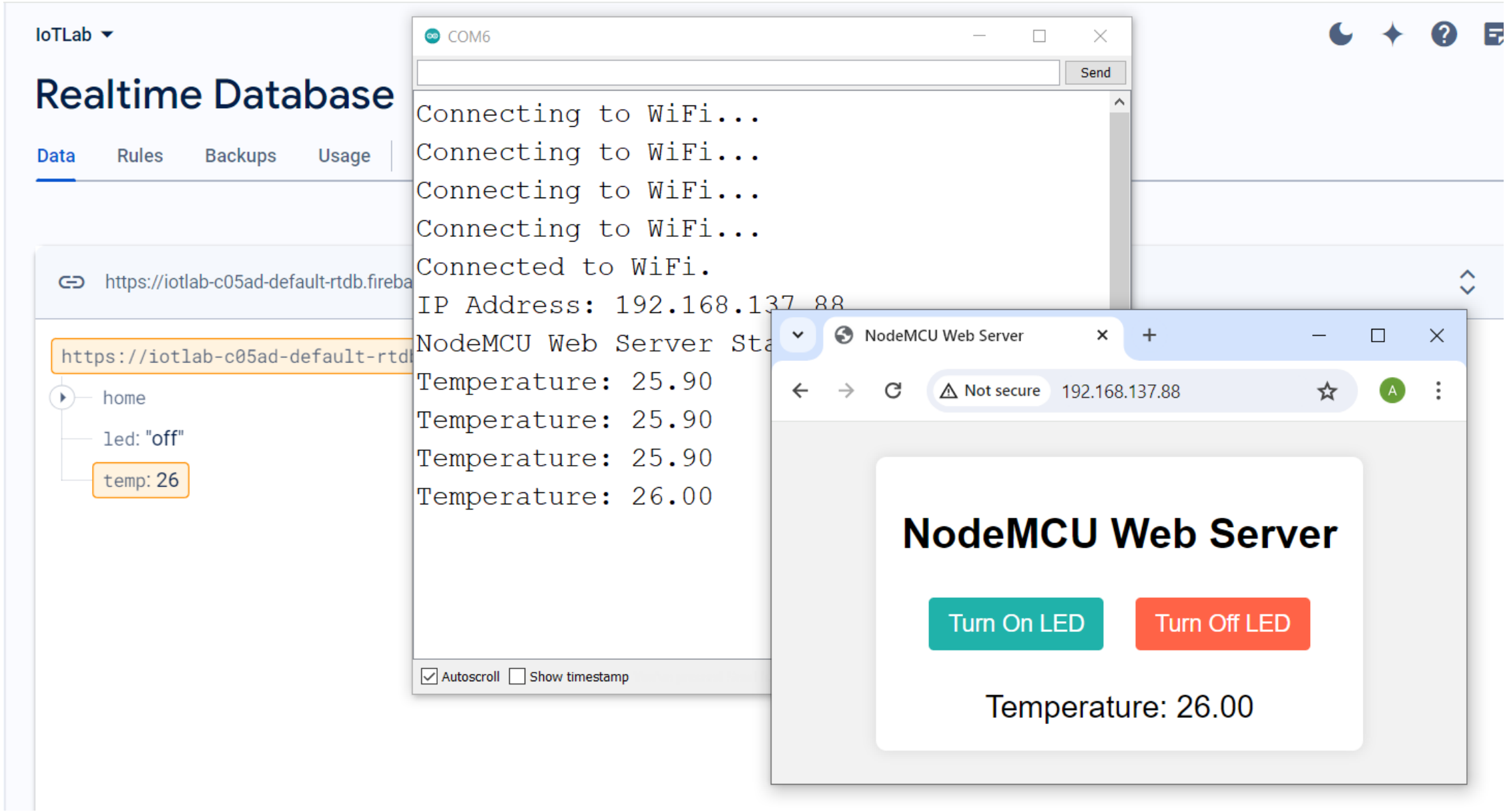
- Inside the `handleRoot()`, `handleLedOn()`, and `handleLedOff()` functions, the `updateFirebase(temp)` function is called to update the temperature value in Firebase each time the LED status is changed.

```
temp = dht.readTemperature();
updateFirebase(temp);
```

- The `getHtml(float temp)` function is modified to include the current temperature value in the HTML content.
- The `#temp#` placeholder in the HTML content is replaced with the actual temperature.

```
htmlContent.replace("#temp#", String(temp));
```

Controlling an LED, Reading Temperature and Sending to Firebase: Output



References

- HTTP Server on NodeMCU with Arduino IDE
- Create A Simple ESP8266 NodeMCU Web Server In Arduino IDE
- Build an ESP8266 Web Server - Code and Schematics (NodeMCU)
- ESP8266 NodeMCU Async Web Server
- ESP8266WebServer - GitHub