Metro light control system using NodeMCU microcontroller

Objective:

The objective of this project is to pioneer a versatile and economical approach to managing metro train signal systems through a NodeMCU-powered web server infrastructure. By introducing this system, our goal is to offer a dependable and intuitive solution that supersedes conventional manual control methods prevalent in metro train intersections. Through this innovation, we seek to empower operators with remote control capabilities, mitigating the susceptibility to damage associated with physical control mechanisms. This advancement promises to optimize metro train Metro flow, enhance operational flexibility, and expedite emergency responses, thereby elevating safety and efficiency within metro train networks.

Abstract:

In the realm of urban transportation, managing metro train intersections efficiently is paramount to ensure seamless operations and passenger safety. This project presents an innovative adaptation of Metro light control technology, employing NodeMCU microcontrollers to establish a Metro Train Light Control Area Network (MTLCAN).

Departing from traditional manual control systems, the MTLCAN integrates NodeMCU microcontrollers with a web server infrastructure, transforming metro train signal management into a remotely accessible and dynamically adjustable process. Through a user-friendly web interface hosted on the NodeMCU's Wi-Fi network, operators gain real-time control over signal lights governing metro train intersections.

The heart of the system lies in the NodeMCU's ability to coordinate and synchronize signal changes, optimizing Metro flow and minimizing congestion at critical junctions. By harnessing the power of the web server, operators can remotely monitor and adjust signal timings, ensuring swift responses to changing Metro conditions and operational requirements.

This project addresses the challenges posed by conventional manual control systems, offering a robust solution that enhances operational efficiency, minimizes delays, and enhances safety within metro train networks. With its accessibility and adaptability, the MTLCAN stands as a testament to the transformative potential of IoT technology in urban transportation management.

Introduction

This project showcases a practical and cost-effective solution that not only enhances Metro control but also contributes to public safety and convenience. It demonstrates the potential of technology in addressing real-world problems and paves the way for a more efficient and responsive Metro management system.

Hardware / Software Requirements:

Required Hardware

1) NodeMCU ESP8266 12E,
2) 220 ohms
Resistor, 3)LED,
4) Breadboard,
5) Jumper Wire.

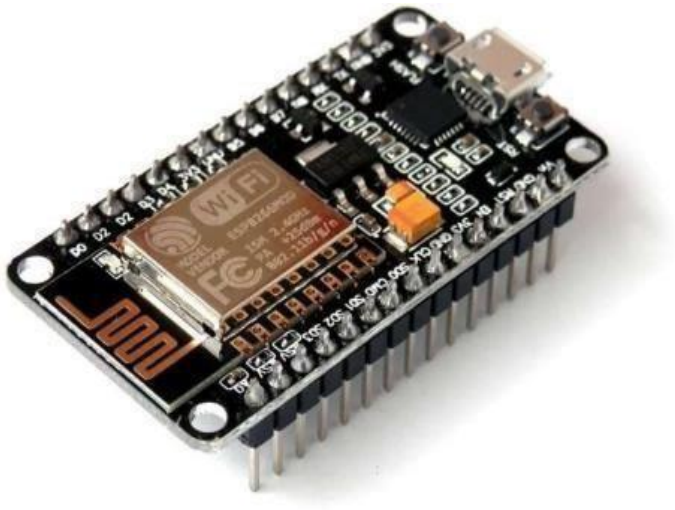
Required Software

Arduino IDE , Internet browser

Hardware Description

Nodemcu ESP8266:-

NodeMCU is an open-source firmware and development kit that helps you to prototype or build IoT products. It includes firmware that runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which is based on the ESP-12 module. The firmware uses the Lua scripting language.



Working Principle:

1. Hardware Setup:

- The project uses a NodeMCU microcontroller, which is an ESP8266-based board with builtin Wi-Fi capability. This board is connected to three LEDs, representing red, green, and yellow Metro lights.

2. Web Server Setup:

- The NodeMCU runs a web server that can be accessed over Wi-Fi. Users can connect to the NodeMCU's Wi-Fi network to control the Metro lights remotely.

3. Remote Control Interface:

- The web server serves a user-friendly web page that provides control options for the three Metro lights (red, green, and yellow).

4. User Interaction:

- Users can access the web page by entering the NodeMCU's IP address in a web browser on their device. This IP address is prominently displayed on the Metro light pole.

5. Metro Light Control:

- Users can control the Metro lights by clicking on buttons provided on the web page. The buttons allow them to turn the lights on or off as needed.

6. Status Updates:

- The web page also displays the current status (on or off) of each Metro light. Users can see the status in real-time.

7. Code Logic:

- The NodeMCU continuously listens for incoming client connections on the specified port (port 80) and handles incoming HTTP requests.

- It parses the incoming HTTP requests to determine which action the user wants to perform (e.g., turning a light on or off).

- It updates the status of the Metro lights and sends the appropriate response back to the user's web browser.

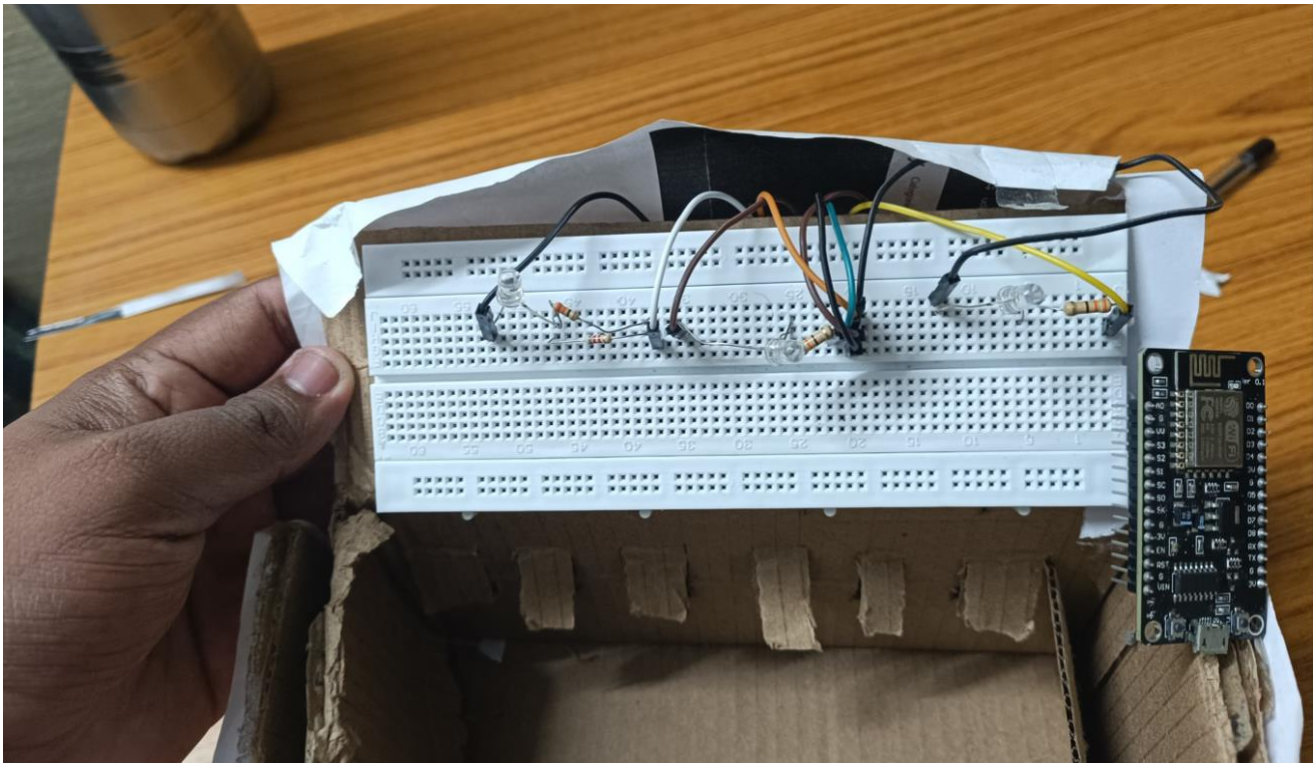
- The code also includes a timeout mechanism to ensure that clients remain connected for a reasonable amount of time.

8. Benefits and Application:

- This system eliminates the need for physical control buttons on the Metro light pole, reducing the risk of damage and malfunctions associated with traditional button-operated systems.

- It allows for remote control of Metro lights, which can be useful in responding to emergencies or adjusting Metro flow patterns based on real-time conditions.

- The prominently displayed IP address on the Metro light pole makes it accessible to anyone with a device connected to the NodeMCU's Wi-Fi network.



By combining hardware and software, this project addresses the objectives and abstract you've described. It provides a practical and cost-effective solution for improving Metro control, enhancing public safety, and ensuring smooth Metro flow through remote management of Metro lights

Approach / Methodology / Programs:

1. Hardware Setup:

- Use a NodeMCU microcontroller board with built-in Wi-Fi capability.
- Connect three LEDs (representing red, green, and yellow Metro lights) to specific GPIO pins on the NodeMCU.

2. Web Server Implementation:

- Implement a web server on the NodeMCU to handle incoming HTTP requests. - The web server should serve a user-friendly web page for Metro light control.

3. User Interface:

- Create an HTML-based user interface that includes buttons for controlling the Metro lights (on/off).
- Display the current status of each Metro light (on or off).

4. User Interaction:

- Users connect to the NodeMCU's Wi-Fi network and access the web page using a web browser.
- Users interact with the web page by clicking the control buttons to change the state of the Metro lights.

5. Code Logic:

- Continuously listen for incoming client connections on a specified port (port 80) using the `WiFiServer` class.
- Parse incoming HTTP requests to determine user actions.
- Update the status of the Metro lights (on or off) based on user inputs. - Send appropriate HTTP responses back to the user's web browser.

6. Error Handling:

- Implement error handling and timeout mechanisms to ensure a smooth user experience and prevent indefinite client connections.

Code Structure and Purpose:

1. Setup Function:

- Initialize serial communication for debugging purposes.
- Set up the GPIO pins for the three LEDs and set their initial state to off (LOW).
- Connect to the Wi-Fi network with the provided SSID and password. - Start the web server on port 80.

2. Loop Function:

- Continuously listen for incoming client connections using the `server.available()` method. - When a new client connects, read and process the incoming HTTP request.

3. HTTP Request Handling:

- Parse the HTTP request to determine the user's action (e.g., turning a Metro light on or off).
- Update the status of the respective Metro light.
- Generate an HTML response to display the current Metro light status and control buttons. - Handle different URLs (e.g., "/2/on" corresponds to turning the red light on).

4. HTML Response:

- Create an HTML response that includes the current status of the Metro lights (on or off) and control buttons.
- Style the web page with CSS to make it visually appealing and user-friendly.

5. Error Handling and Timeout:

- Implement a timeout mechanism to close client connections after a specified period (e.g., 2 seconds) to prevent indefinite connections.
- Handle errors and disconnect clients if necessary.

This code structure and methodology enable users to access a web page served by the NodeMCU, control Metro lights remotely, and view their current status. The code logic ensures responsive and user-friendly Metro light management through a web interface while handling errors and preventing long-term connections for reliability and security.

Program

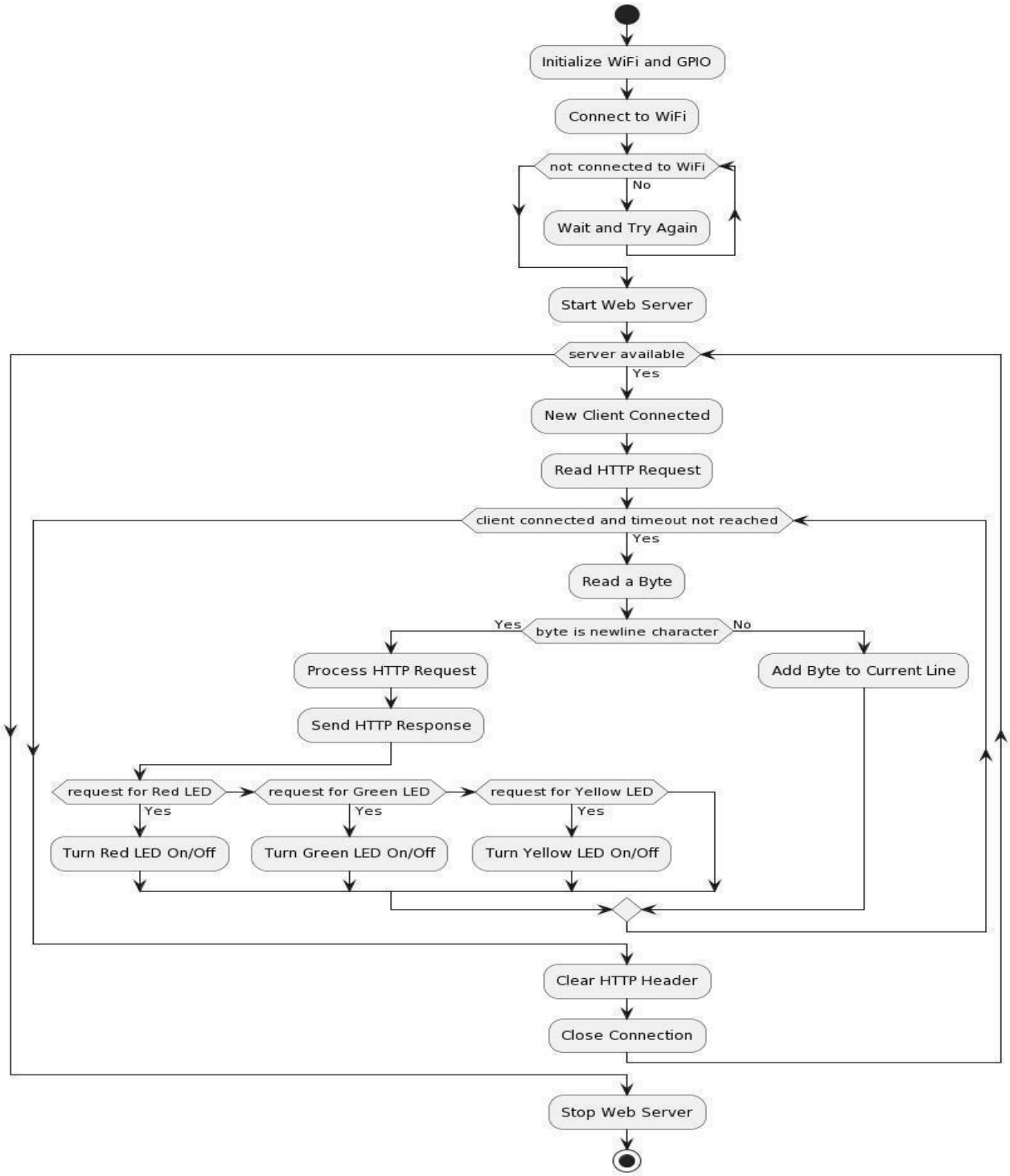
```
#include <ESP8266WiFi.h>

// Enter your wifi network name and Wifi Password const char*
ssid = "YBY";
const char* password = "12345yby";

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

// Variable to store the HTTP
request String header;

// These variables store current output state of LED
String outputRedState = "off";
String outputGreenState = "off";
String outputYellowState = "off";

// Assign output variables to GPIO pins
const int redLED = 2; const int greenLED
= 4;
const int yellowLED = 5;

// 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(redLED, OUTPUT);
pinMode(greenLED, OUTPUT);
pinMode(yellowLED,OUTPUT); // Set
outputs to LOW digitalWrite(redLED,
LOW); digitalWrite(greenLED, LOW);
digitalWrite(yellowLED, 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,
Serial.println("New Client."); // print a message out in the serial port String
currentLine = ""; // make a String to hold incoming data from the client
currentTime = millis(); previousTime = currentTime;
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("Contenttype:text/html");
client.println("Connection: close");
client.println();

// turns the GPIOs on and off
if (header.indexOf("GET /2/on") >= 0) {
Serial.println("RED LED is on");
outputRedState = "on"; digitalWrite(redLED,
HIGH);
} else if (header.indexOf("GET /2/off") >= 0)
{ Serial.println("RED LED is off");
outputRedState = "off"; digitalWrite(redLED,
LOW);
} else if (header.indexOf("GET /4/on") >= 0)
{ Serial.println("Green LED is on");
outputGreenState = "on";
digitalWrite(greenLED, HIGH);
```

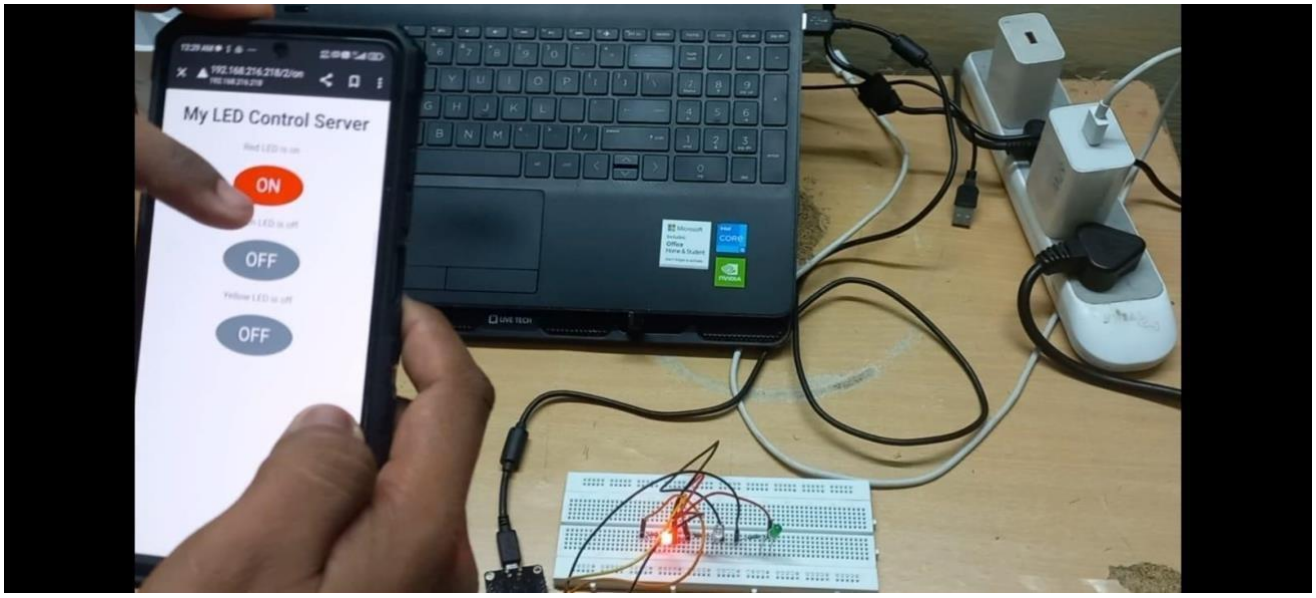
```
} else if (header.indexOf("GET /4/off") >= 0)
{ Serial.println("Green LED is off");
outputGreenState = "off";
digitalWrite(greenLED, LOW);
} else if (header.indexOf("GET /5/on") >= 0)
{ Serial.println("Yellow LED is on");
outputYellowState = "on";
digitalWrite(yellowLED, HIGH);
} else if (header.indexOf("GET /5/off") >= 0) {
Serial.println("Yellow LED is off");
outputYellowState = "off"; digitalWrite(yellowLED,
LOW);
}

// Display the HTML web page client.println("<!DOCTYPE html><html>");
client.println("<head><meta name=\"viewport\" content=\"width=device-width,
        initialscale=1\">"); client.println("<link
rel=\"icon\" href=\"data:,\">");
// CSS to style the on/off buttons
client.println("<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto;
text-align: center;}"); client.println(".buttonRed { background-color: #ff0000; border: none;
color: white; padding:
16px 40px; border-radius: 60%;"); client.println("text-decoration: none; font-size: 30px;
margin: 2px; cursor: pointer;}"); client.println(".buttonGreen { background-color: #00ff00;
border: none; color: white; padding:
16px 40px; border-radius: 60%;"); client.println("text-decoration: none; font-size: 30px;
margin: 2px; cursor: pointer;}"); client.println(".buttonYellow { background-color: #feeb36;
border: none; color: white; padding: 16px 40px; border-radius: 60%;");
client.println("textdecoration: none; font-size: 30px; margin: 2px; cursor: pointer;}");
client.println(".buttonOff { background-color: #77878A; border: none; color: white; padding:
16px 40px; border-radius: 70%;"); client.println("text-decoration: none; font-size:
        30px; margin:          2px; cursor:
pointer;}</style></head>");

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

// Display current state, and ON/OFF buttons for GPIO 2 Red
LED client.println("<p>Red LED is " + outputRedState +
"</p>"); // If the outputRedState is off, it displays the OFF button
if (outputRedState=="off") {
client.println("<p><a href=\"/2/on\"><button                                    class=\"button
buttonOff\">OFF</button></a></p>"); } else
{
```

```
client.println("<p><a href=\"/2/off\"><button class=\"button buttonRed\">ON</button></a></p>"); }


// Display current state, and ON/OFF buttons for GPIO 4 Green LED
client.println("<p>Yellow LED is " + outputGreenState + "</p>");
// If the outputGreenState is off, it displays the OFF button if (outputGreenState =="off") {
client.println("<p><a href=\"/4/on\"><button class=\"button buttonOff\">OFF</button></a></p>");
} else {
client.println("<p><a href=\"/4/off\"><button class=\"button buttonGreen\">ON</button></a></p>");
}
client.println("</body></html>");

// Display current state, and ON/OFF buttons for GPIO 5 Yellow LED client.println("<p>Green LED is " + outputYellowState + "</p>"); // If the outputYellowState is off, it displays the OFF button if (outputYellowState =="off") {
client.println("<p><a href=\"/5/on\"><button class=\"button buttonOff\">OFF</button></a></p>");
} else {
client.println("<p><a href=\"/5/off\"><button class=\"button buttonYellow\">ON</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("");

}
}
```

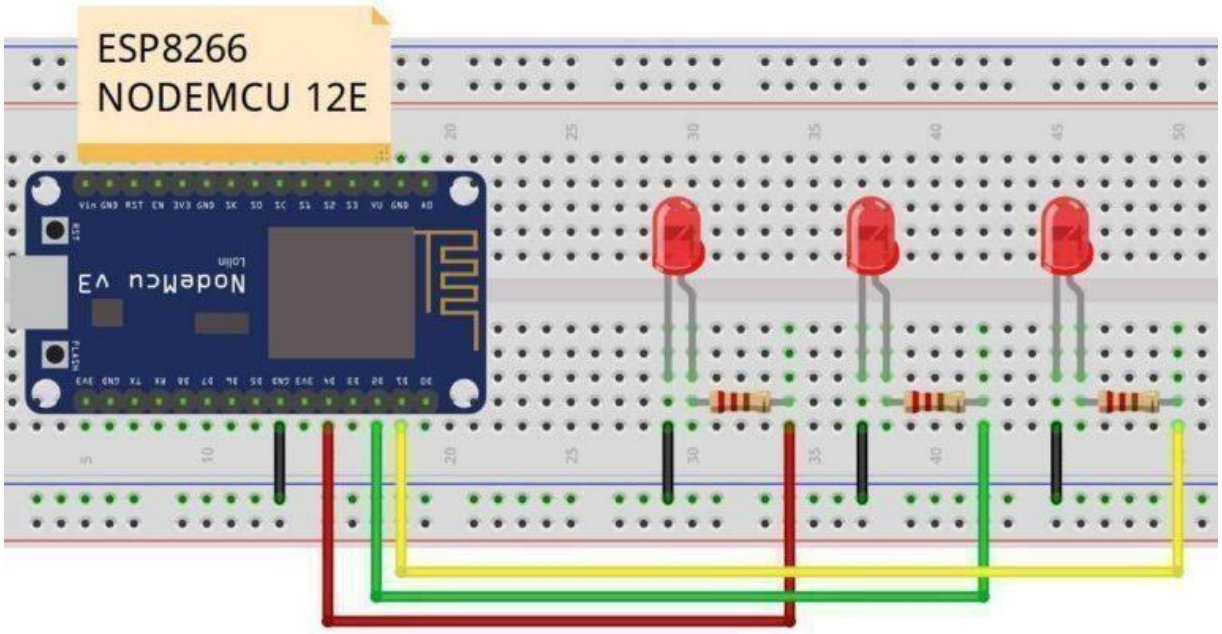
Flowchart


Initialize WiFi and GPIO
Connect to WiFi
not connected to WiFi
No
Wait and Try Again
Start Web Server
server available
Yes
New Client Connected
Read HTTP Request
client connected and timeout not reached
Yes
Read a Byte
Yes
byte is newline character
No
Process HTTP Request
Add Byte to Current Line
Send HTTP Response
request for Red LED
request for Green LED
request for Yellow LED
Yes
Yes
Yes
Turn Red LED On/Off
Turn Green LED On/Off
Turn Yellow LED On/Off
Clear HTTP Header
Close Connection
Stop Web Server

Project Photos:



Schematic Diagram



Conclusion:

In conclusion, the implementation of a NodeMCU-based web server for remote metro light control presents a significant leap forward in efficiency and cost-effectiveness. By facilitating swift responses to emergencies and real-time adjustments to metro flow, this solution contributes directly to public safety and operational fluidity. With its intuitive interface and demonstrated efficacy, this project underscores the transformative impact of technology in addressing pressing urban challenges, signaling its potential for broad application in the management of metro systems worldwide.

References:

Nodemcu :- **https://ieeexplore.ieee.org/document/9297917**

Arduino:- **https://ieeexplore.ieee.org/document/6997578**