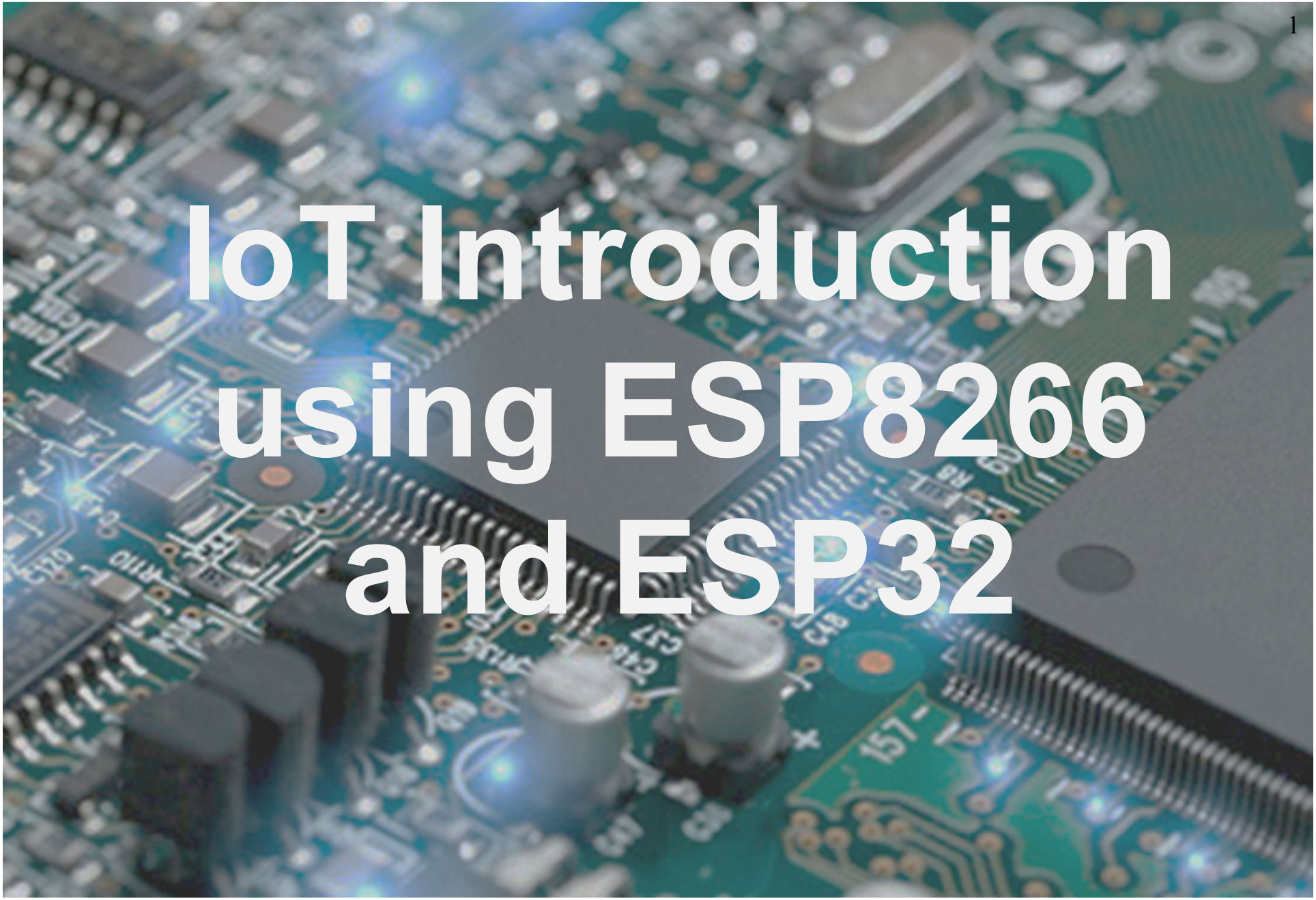
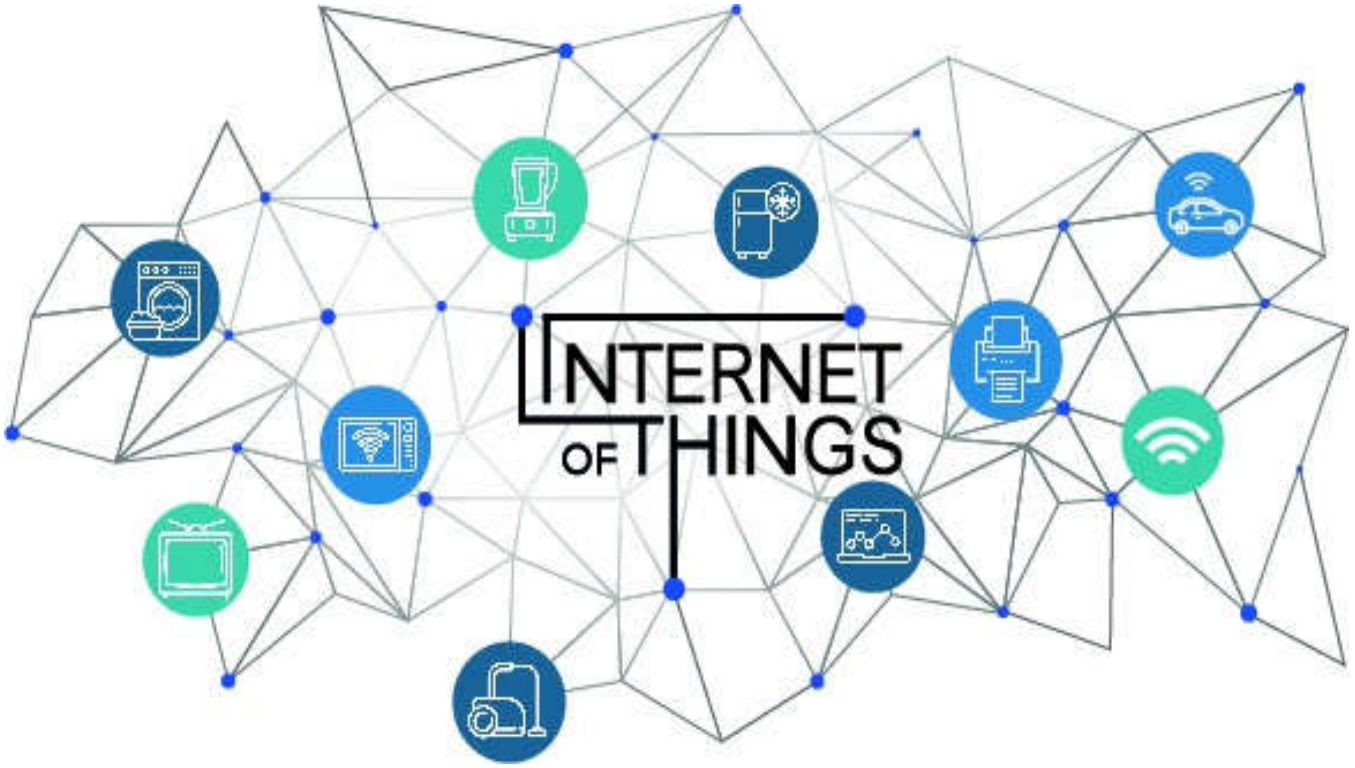


IoT Introduction using ESP8266 and ESP32

Internet of Things Introduction



IoT Introduction using ESP8266 and ESP32

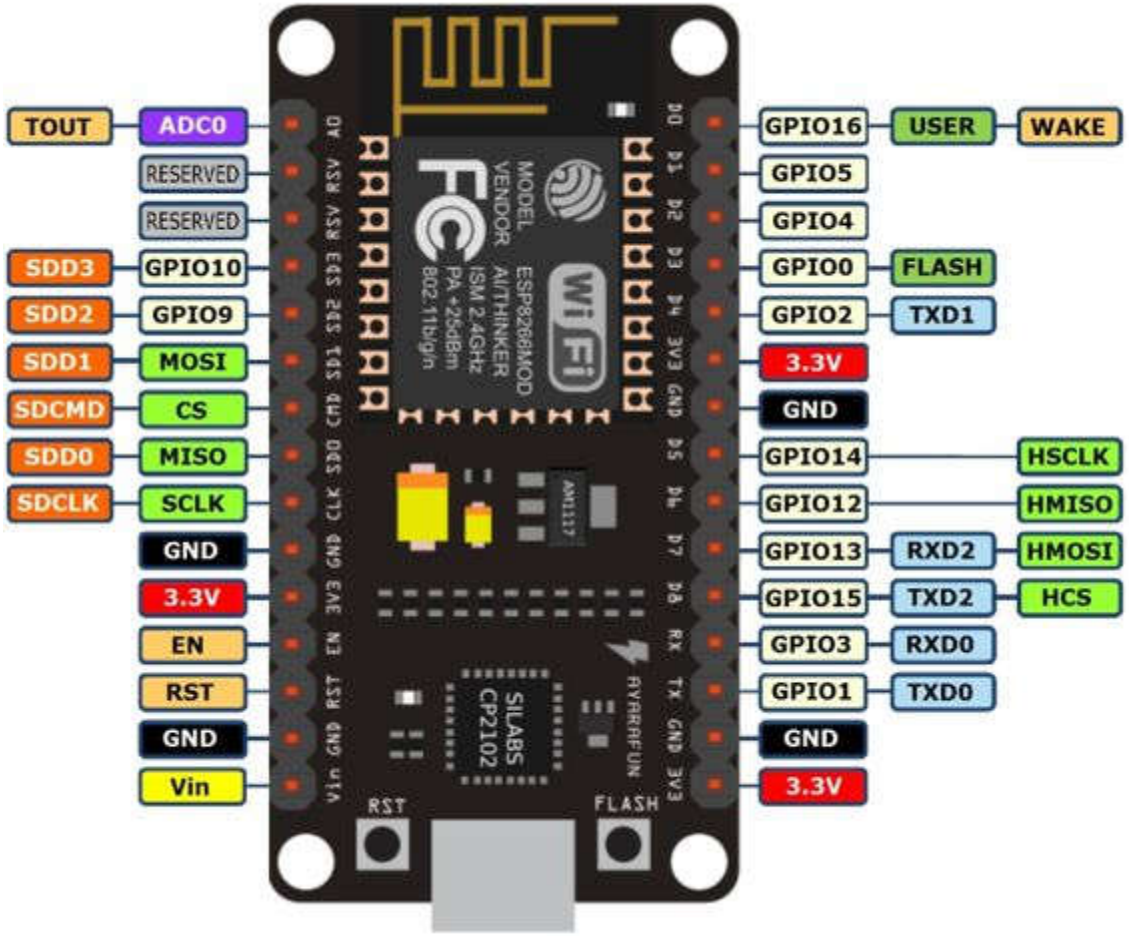
Contents

- Objectives
- Wi-Fi Module Technologies
- Internet of Things
- Beyond Arduino

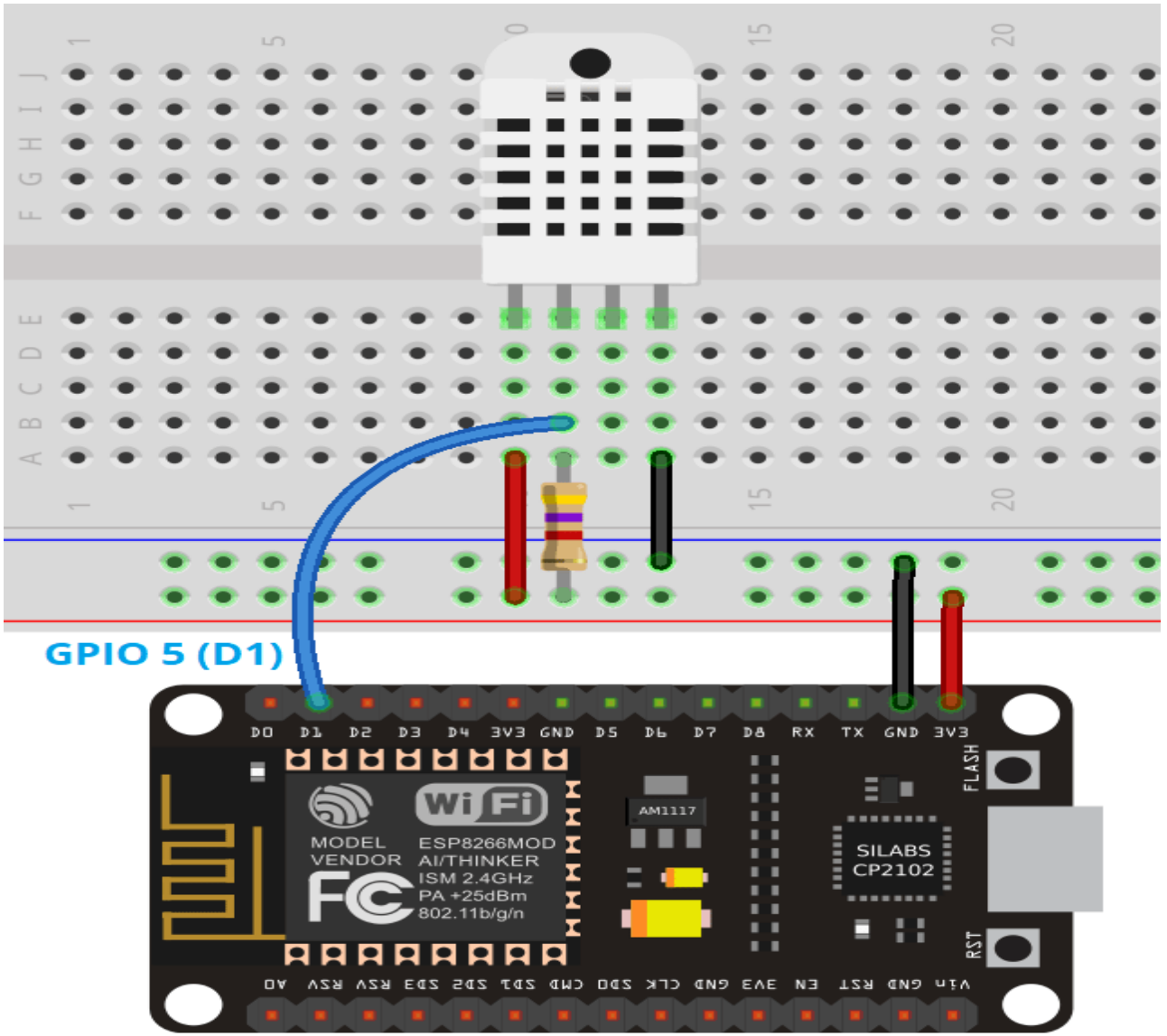
Objectives

- In this tutorial, you will learn:
  - To familiarize different Wi-Fi technologies can be used using ESP8266 and ESP32.
  - To know about Internet of Things .
  - To understand basic Arduino Programming using ESP8266 and ESP32.

IoT using ESP8266 WiFi Module



IoT using ESP8266 WiFi Module



IoT using ESP8266 WiFi Module

```
//NodeMCUDHT11.ino
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <Hash.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>

// Replace with your network credentials
const char* ssid = "SSID";
const char* password = "PASSWORD";

#define DHTPIN 5     // Digital pin D1 connected to the DHT sensor

// Uncomment the type of sensor in use:
#define DHTTYPE    DHT11     // DHT 11
//#define DHTTYPE    DHT22     // DHT 22 (AM2302)
//#define DHTTYPE    DHT21     // DHT 21 (AM2301)

DHT dht(DHTPIN, DHTTYPE);

// current temperature & humidity, updated in loop()
float t = 0.0;
float h = 0.0;

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

// Generally, you should use "unsigned long" for variables that hold time
// The value will quickly become too large for an int to store
unsigned long previousMillis = 0;    // will store last time DHT was updated

// Updates DHT readings every 10 seconds
const long interval = 10000;
```


IoT using ESP8266 WiFi Module

```
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css"
integrity="sha384-fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr"
crossorigin="anonymous">
  <style>
    html {
     font-family: Arial;
     display: inline-block;
     margin: 0px auto;
     text-align: center;
    }
    h2 { font-size: 3.0rem; }
    p { font-size: 3.0rem; }
    .units { font-size: 1.2rem; }
    .dht-labels{
      font-size: 1.5rem;
      vertical-align:middle;
      padding-bottom: 15px;
    }
  </style>
</head>
<body>
  <h2>ESP8266 DHT11 WebServer</h2>
  <p>
    <i class="fas fa-thermometer-half" style="color:#059e8a;"></i>
    <span class="dht-labels">Temperature</span>
    <span id="temperature">%TEMPERATURE%</span>
    <sup class="units">&deg;C</sup>
  </p>
  <p>
    <i class="fas fa-tint" style="color:#00add6;"></i>
    <span class="dht-labels">Humidity</span>
```

IoT using ESP8266 WiFi Module

```
<span id="humidity">%HUMIDITY%</span>
    <sup class="units">%</sup>
  </p>
  <p>

</body>
<script>
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("temperature").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/temperature", true);
  xhttp.send();
}, 10000 ) ;

setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("humidity").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/humidity", true);
  xhttp.send();
}, 10000 ) ;
</script>
</html>)rawliteral";
```

IoT using ESP8266 WiFi Module

```
// Replaces placeholder with DHT values
String processor(const String& var){
  //Serial.println(var);
  if(var == "TEMPERATURE"){
    return String(t);
  }
  else if(var == "HUMIDITY"){
    return String(h);
  }
  return String();
}

void setup(){
  // Serial port for debugging purposes
  Serial.begin(115200);
  dht.begin();

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  Serial.println("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println(".");
  }

  // Print ESP8266 Local IP Address
  Serial.println(WiFi.localIP());

  // Route for root / web page
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html, processor);
  });
  server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", String(t).c_str());
  });
```


IoT using ESP8266 WiFi Module


```
server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", String(h).c_str());
  });

  // Start server
  server.begin();
}

void loop(){
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    // save the last time you updated the DHT values
    previousMillis = currentMillis;
    // Read temperature as Celsius (the default)
    float newT = dht.readTemperature();
    // Read temperature as Fahrenheit (isFahrenheit = true)
    //float newT = dht.readTemperature(true);
    // if temperature read failed, don't change t value
    if (isnan(newT)) {
      Serial.println("Failed to read from DHT sensor!");
    }
    else {
      t = newT;
      Serial.println(t);
    }
    // Read Humidity
    float newH = dht.readHumidity();
    // if humidity read failed, don't change h value
    if (isnan(newH)) {
      Serial.println("Failed to read from DHT sensor!");
    }
    else {
      h = newH;
      Serial.println(h);
    }
  }
}
```

IoT using ESP8266 WiFi Module

```
/*********
  IECEP RIZAL BME WEBSERVER
  BME280Webserver
*********/

// Load Wi-Fi library
#include <ESP8266WiFi.h>
#include <Wire.h>
#include <Adafruit_BME280.h>
#include <Adafruit_Sensor.h>

//uncomment the following lines if you're using SPI
/*#include <SPI.h>
#define BME_SCK 14
#define BME_MISO 12
#define BME_MOSI 13
#define BME_CS 15*/

#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BME280 bme; // I2C
//Adafruit_BME280 bme(BME_CS); // hardware SPI
//Adafruit_BME280 bme(BME_CS, BME_MOSI, BME_MISO, BME_SCK); // software SPI

// Replace with your network credentials
const char* ssid     = “SSID";
const char* password = “PASSWORD";

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

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


IoT using ESP8266 WiFi Module

```
void setup() {
  Serial.begin(115200);
  bool status;

  // default settings
  // (you can also pass in a Wire library object like &Wire2)
  //status = bme.begin();
  if (!bme.begin(0x76)) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
  }

  // 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
```

IoT using ESP8266 WiFi Module

```
while (client.connected()) {            // loop while the client's connected
      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();

            // 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 table
            client.println("<style>body { text-align: center; font-family: \"Trebuchet MS\",
Arial;}");
            client.println("table { border-collapse: collapse; width:35%; margin-left:auto;
margin-right:auto; }");
            client.println("th { padding: 12px; background-color: #0043af; color: white; }");
            client.println("tr { border: 1px solid #ddd; padding: 12px; }");
            client.println("tr:hover { background-color: #bcbcbc; }");
            client.println("td { border: none; padding: 12px; }");
            client.println(".sensor { color:white; font-weight: bold; background-color: #bcbcbc;
padding: 1px; }");
```

IoT using ESP8266 WiFi Module

```
// Web Page Heading
            client.println("</style></head><body><h1>ESP8266 with BME280</h1>");
            client.println("<table><tr><th>MEASUREMENT</th><th>VALUE</th></tr>");
            client.println("<tr><td>Temp. Celsius</td><td><span class=\"sensor\">");
            client.println(bme.readTemperature());
            client.println(" *C</span></td></tr>");
            client.println("<tr><td>Temp. Fahrenheit</td><td><span class=\"sensor\">");
            client.println(1.8 * bme.readTemperature() + 32);
            client.println(" *F</span></td></tr>");
            client.println("<tr><td>Pressure</td><td><span class=\"sensor\">");
            client.println(bme.readPressure() / 100.0F);
            client.println(" hPa</span></td></tr>");
            client.println("<tr><td>Approx. Altitude</td><td><span class=\"sensor\">");
            client.println(bme.readAltitude(SEALEVELPRESSURE_HPA));
            client.println(" m</span></td></tr>");
            client.println("<tr><td>Humidity</td><td><span class=\"sensor\">");
            client.println(bme.readHumidity());
            client.println(" %</span></td></tr>");
            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
        }
      }
    }
```


IoT using ESP8266 WiFi Module

```
// Clear the header variable
    header = "";
    // Close the connection
    client.stop();
    Serial.println("Client disconnected.");
    Serial.println("");
  }
}
```

IoT using ESP8266 WiFi Module

```
/*********
  IECEP RIZAL
  RelayWebserverESP8266.ino
*********/

// Import required libraries
#include "ESP8266WiFi.h"
#include "ESPAsyncWebServer.h"

// Set to true to define Relay as Normally Open (NO)
#define RELAY_NO    true

// Set number of relays
#define NUM_RELAYS  4

// Assign each GPIO to a relay
int relayGPIOs[NUM_RELAYS] = {14, 2, 0, 16};

// Replace with your network credentials
const char* ssid = “SSID";
const char* password = “PASSWORD";

const char* PARAM_INPUT_1 = "relay";
const char* PARAM_INPUT_2 = "state";

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <style>
    html {font-family: Arial; display: inline-block; text-align: center;}
    h2 {font-size: 3.0rem;}
```

IoT using ESP8266 WiFi Module

```
p {font-size: 3.0rem;}
    body {max-width: 600px; margin:0px auto; padding-bottom: 25px;}
    .switch {position: relative; display: inline-block; width: 120px; height: 68px}
    .switch input {display: none}
    .slider {position: absolute; top: 0; left: 0; right: 0; bottom: 0; background-color: #ccc;
border-radius: 34px}
    .slider:before {position: absolute; content: ""; height: 52px; width: 52px; left: 8px;
bottom: 8px; background-color: #fff; -webkit-transition: .4s; transition: .4s; border-radius:
68px}
    input:checked+.slider {background-color: #2196F3}
    input:checked+.slider:before {-webkit-transform: translateX(52px); -ms-transform:
translateX(52px); transform: translateX(52px)}
  </style>
</head>
<body>
  <h2>ESP Web Server</h2>
  %BUTTONPLACEHOLDER%
<script>function toggleCheckbox(element) {
  var xhr = new XMLHttpRequest();
  if(element.checked){ xhr.open("GET", "/update?relay="+element.id+"&state=1", true); }
  else { xhr.open("GET", "/update?relay="+element.id+"&state=0", true); }
  xhr.send();
}</script>
</body>
</html>
)rawliteral";

// Replaces placeholder with button section in your web page
String processor(const String& var){
  //Serial.println(var);
  if(var == "BUTTONPLACEHOLDER"){
    String buttons ="";
    for(int i=1; i<=NUM_RELAYS; i++){
      String relayStateValue = relayState(i);
      buttons+= "<h4>Relay #" + String(i) + " - GPIO " + relayGPIOs[i-1] + "</h4><label
```

IoT using ESP8266 WiFi Module

```
class=\"switch\"><input type=\"checkbox\" onchange=\"toggleCheckbox(this)\" id=\"" + String(i) +
"\" "+ relayStateValue +"><span class=\"slider\"></span></label>";
    }
    return buttons;
  }
  return String();
}

String relayState(int numRelay){
  if(RELAY_NO){
    if(digitalRead(relayGPIOs[numRelay-1])){
      return "";
    }
    else {
      return "checked";
    }
  }
  else {
    if(digitalRead(relayGPIOs[numRelay-1])){
      return "checked";
    }
    else {
      return "";
    }
  }
  return "";
}

void setup(){
  // Serial port for debugging purposes
  Serial.begin(115200);

  // Set all relays to off when the program starts - if set to Normally Open (NO), the relay is
off when you set the relay to HIGH
  for(int i=1; i<=NUM_RELAYS; i++){
```

IoT using ESP8266 WiFi Module

```
pinMode(relayGPIOs[i-1], OUTPUT);
    if(RELAY_NO){
      digitalWrite(relayGPIOs[i-1], HIGH);
    }
    else{
      digitalWrite(relayGPIOs[i-1], LOW);
    }
  }

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
  }

  // Print ESP8266 Local IP Address
  Serial.println(WiFi.localIP());

  // Route for root / web page
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html, processor);
  });

  // Send a GET request to <ESP_IP>/update?relay=<inputMessage>&state=<inputMessage2>
  server.on("/update", HTTP_GET, [] (AsyncWebServerRequest *request) {
    String inputMessage;
    String inputParam;
    String inputMessage2;
    String inputParam2;
    // GET input1 value on <ESP_IP>/update?relay=<inputMessage>
    if (request->hasParam(PARAM_INPUT_1) & request->hasParam(PARAM_INPUT_2)) {
      inputMessage = request->getParam(PARAM_INPUT_1)->value();
      inputParam = PARAM_INPUT_1;
      inputMessage2 = request->getParam(PARAM_INPUT_2)->value();
```


IoT using ESP8266 WiFi Module

```
inputParam2 = PARAM_INPUT_2;
      if(RELAY_NO){
        Serial.print("NO ");
        digitalWrite(relayGPIOs[inputMessage.toInt()-1], !inputMessage2.toInt());
      }
      else{
        Serial.print("NC ");
        digitalWrite(relayGPIOs[inputMessage.toInt()-1], inputMessage2.toInt());
      }
    }
    else {
      inputMessage = "No message sent";
      inputParam = "none";
    }
    Serial.println(inputMessage + inputMessage2);
    request->send(200, "text/plain", "OK");
  });
  // Start server
  server.begin();
}

void loop() {

}
```

IoT using ESP8266 WiFi Module

```
// WiFi Weather Station

// Required libraries
#include <ESP8266WiFi.h>

// Libraries
#include "DHT.h"

// Pin
#define DHTPIN 4

// Use DHT11 sensor
#define DHTTYPE DHT11

// Initialize DHT sensor
DHT dht(DHTPIN, DHTTYPE, 15);

// WiFi parameters
const char* ssid = “SSID";
const char* password = “PASSWORD";

// Create an instance of the server
WiFiServer server(80);

// Pin
int output_pin = 5;

void setup() {

  // Start Serial
  Serial.begin(115200);
  delay(10);
```


IoT using ESP8266 WiFi Module

```
// Prepare GPIO5
  pinMode(output_pin, OUTPUT);
  digitalWrite(output_pin, 0);

  // Init DHT
  dht.begin();

  // Connect to WiFi network
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

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

  // Start the server
  server.begin();
  Serial.println("Server started");

  // Print the IP address
  Serial.println(WiFi.localIP());
}

void loop() {

  // Check if a client has connected
  WiFiClient client = server.available();
  if (!client) {
    return;
```

IoT using ESP8266 WiFi Module

```
// Wait until the client sends some data
  Serial.println("new client");
  while(!client.available()){
    delay(1);
  }

  // Reading temperature and humidity
  float h = dht.readHumidity();
  // Read temperature as Celsius
  float t = dht.readTemperature();

  // Read the first line of the request
 // String req = client.readStringUntil('\r');
 // Serial.println(req);
//  client.flush();

// Read the first line of the request
  String req = client.readStringUntil('\r');
  Serial.println(req);
  client.flush();

  // Match the request
  if (req.indexOf("/on") != -1){
    digitalWrite(output_pin, 1);
  }
  else if (req.indexOf("/off") != -1) {
    digitalWrite(output_pin, 0);
  }

  client.flush();
```

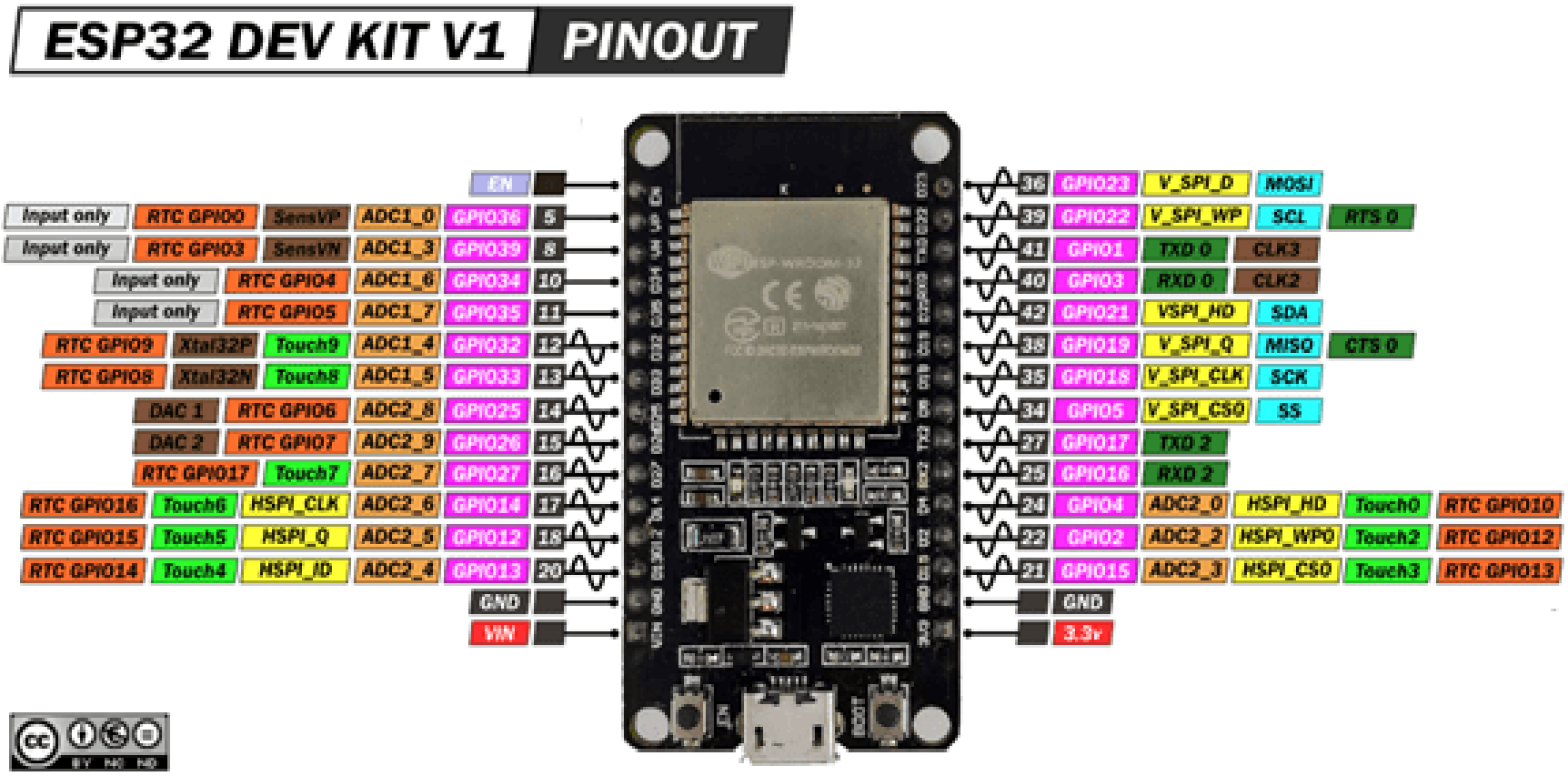
IoT using ESP8266 WiFi Module

```
// Prepare the response
  String s = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n";
  s += "<head>";
  s += "<meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">";
  s += "<meta http-equiv=\"refresh\" content=\"60\" />";
  s += "<script src=\"https://code.jquery.com/jquery-2.1.3.min.js\"></script>";
  s += "<link rel=\"stylesheet\"
href=\"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/css/bootstrap.min.css\">";
  s += "<style>body{font-size: 24px;} .voffset {margin-top: 30px;}</style>";
  s += "</head>";

  s += "<h1>Sample IoT Application</h1>";
  s += "<h2>ESP8266 Controlled Lamp</h2>";
  s += "<div class=\"row\">";
  s += "<div class=\"col-md-2\"><input class=\"btn btn-block btn-lg btn-primary\"
type=\"button\" value=\"On\" onclick=\"on()\"></div>";
  s += "<div class=\"col-md-2\"><input class=\"btn btn-block btn-lg btn-danger\" type=\"button\"
value=\"Off\" onclick=\"off()\"></div>";
  s += "</div></div>";
  s += "<script>function on() {$.get(\"/on\");}</script>";
  s += "<script>function off() {$.get(\"/off\");}</script>";

  //s += "<div class=\"container\">";
  s += "<h1>DHT11 WiFi Demo</h1>";
  s += "<div class=\"row\">";
  s += "<div class=\"col-md-2\">Temperature: </div><div class=\"col-md-2\">" + String(t) +
"</div>";
  s += "<div class=\"col-md-2\">Humidity: </div><div class=\"col-md-2\">" + String(h) +
"</div>";
  s += "</div>";

  s += "</div>";
```

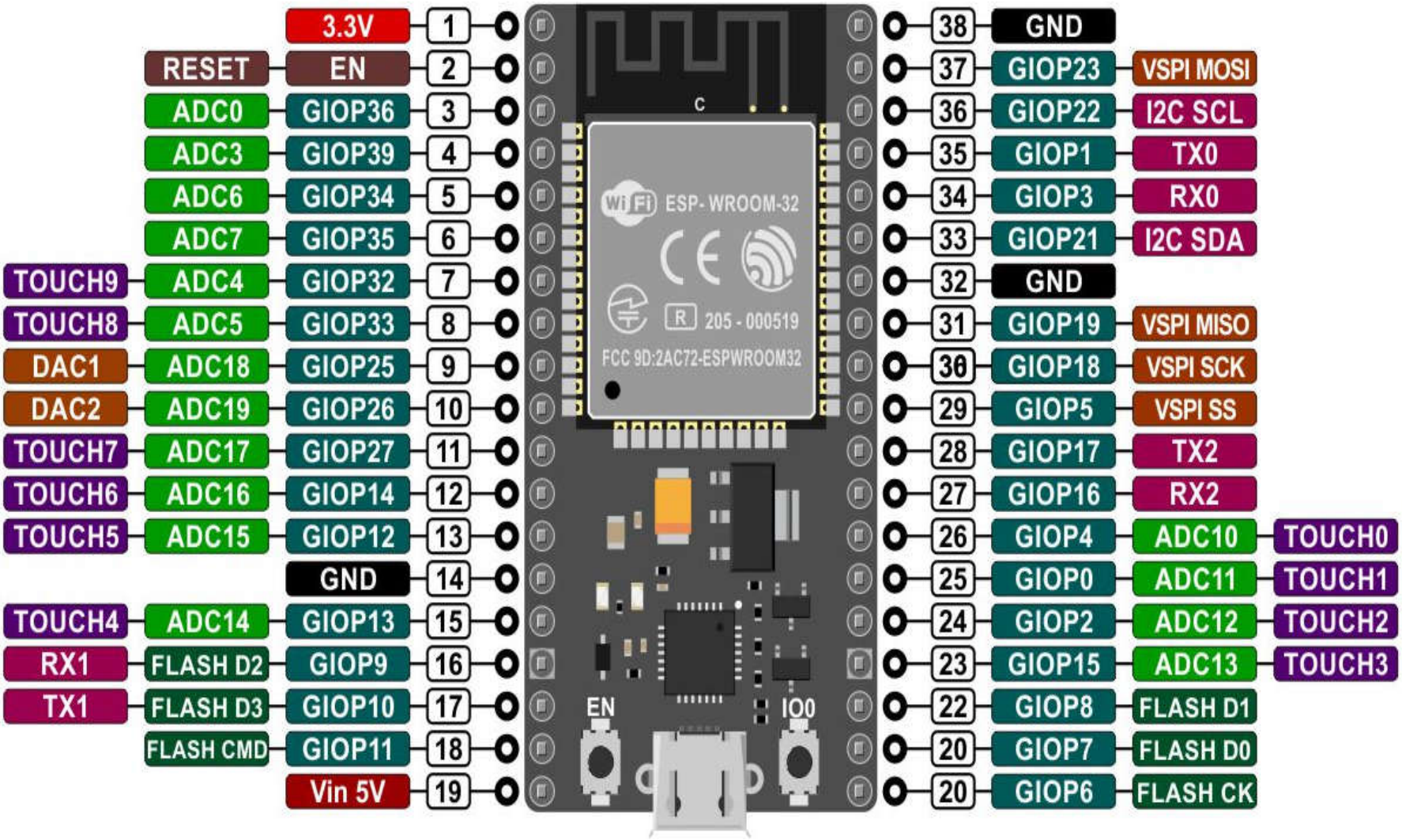
IoT using ESP8266 WiFi Module

```
// Send the response to the client
  client.print(s);
  delay(1);
  Serial.println("Client disconnected");

  // The client will actually be disconnected
  // when the function returns and 'client' object is detroyed
}
```

IoT using ESP32 Module



IoT using ESP32 Module



IoT using ESP32 Module

```
//ESP32_GPIO.ino
#include <WiFi.h>

// Replace with your network credentials
const char* ssid = “SSID";
const char* password = “PASSWORD";

// 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 output25 = 25;
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(output25, OUTPUT);
  pinMode(output27, OUTPUT);
  // Set outputs to LOW
  digitalWrite(output25, LOW);
  digitalWrite(output27, LOW);
```


IoT using ESP32 Module

```
// 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:
```

IoT using ESP32 Module

```
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 /25/on") >= 0) {
              Serial.println("GPIO 25 on");
              output26State = "on";
              digitalWrite(output25, HIGH);
            } else if (header.indexOf("GET /25/off") >= 0) {
              Serial.println("GPIO 25 off");
              output26State = "off";
              digitalWrite(output25, 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
```


IoT using ESP32 Module

```
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 Demo</h1>");

            // Display current state, and ON/OFF buttons for GPIO 26
            client.println("<p>GPIO 25 - State " + output26State + "</p>");
            // If the output26State is off, it displays the ON button
            if (output26State=="off") {
              client.println("<p><a href=\"/25/on\"><button
class=\"button\">ON</button></a></p>");
            } else {
              client.println("<p><a href=\"/25/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>");
```

IoT using ESP32 Module

```
// 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("");
  }
}
```


THANK YOU !

Q&A