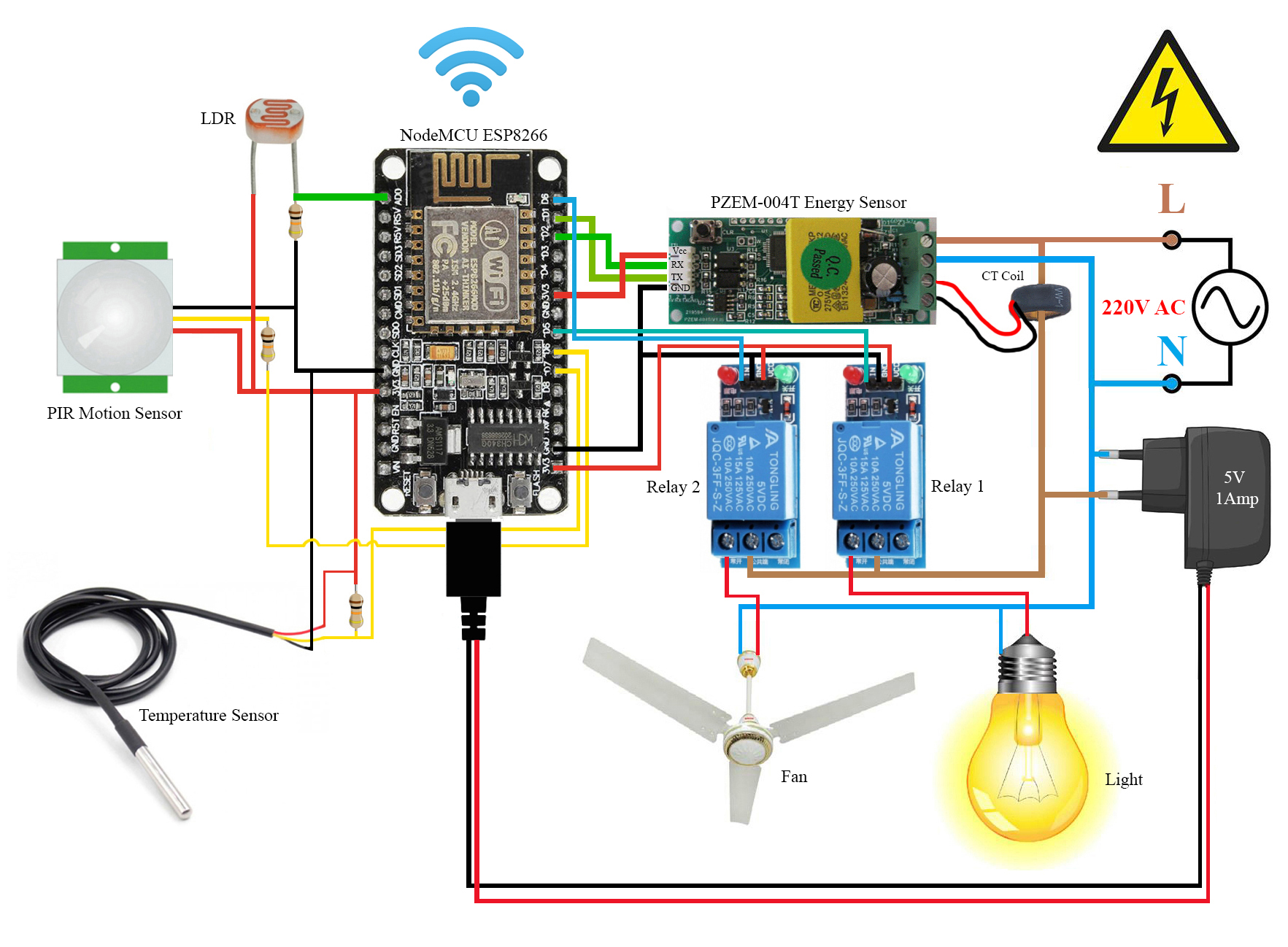
# **SERVERLESS IOT PROCESSING**

PHASE 4 : **DEVELOPMENT PART-2**

**PROBLEM DEFINITION:**

The project aims to transform a home into a smart living space using IBM Cloud Functions for IoT data processing. The goal is to collect data from various smart devices, process it in real-time, and automate routines for energy efficiency and home security. This involves designing the smart home setup, implementing data collection and processing, and leveraging IBM Cloud for storage and analysis.

**CIRCUIT DIAGRAM AND COMPONENTS:**



**CONNECT THE IBM CLOUD:**

To connect a NodeMCU to IBM Cloud using MQTT, you can follow these steps. I'll also provide a simplified example using the Arduino IDE for publishing data from NodeMCU to IBM Cloud.

Step 1: Set Up Your IBM Cloud Account

- If you don't already have an IBM Cloud account, sign up for one at [IBM Cloud](https://cloud.ibm.com/).

Step 2: Create an MQTT Service in IBM Cloud

- Log in to your IBM Cloud account and create an instance of the "Internet of Things Platform" or "MQTT" service. This service will provide you with the necessary MQTT broker information.

Step 3: Get MQTT Broker Details

- Within your IBM Cloud IoT service, you will find the MQTT broker details, including the server address, port, and authentication credentials. Make a note of these details as you will need them in the NodeMCU code.

Step 4: Set Up Arduino IDE for NodeMCU

- Install the Arduino IDE and configure it to work with the NodeMCU. You'll need the "ESP8266" board package installed.

Step 5: Write the Arduino Code

Here's a basic example of NodeMCU code to publish data to IBM Cloud:

#include <ESP8266WiFi.h>

#include <PubSubClient.h>

const char\* ssid = "YourWiFiSSID";

const char\* password = "YourWiFiPassword";

const char\* mqttServer = "your\_ibm\_mqtt\_server";

const int mqttPort = 1883;

const char\* mqttUser = "your\_ibm\_mqtt\_user";

const char\* mqttPassword = "your\_ibm\_mqtt\_password";

const char\* mqttTopic = "your/mqtt/topic";

WiFiClient espClient;

PubSubClient client(espClient);

void setup() {

Serial.begin(115200);

setupWiFi();

client.setServer(mqttServer, mqttPort);

}

void setupWiFi() {

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.println("Connecting to WiFi...");

}

Serial.println("Connected to WiFi");

}

void reconnect() {

while (!client.connected()) {

Serial.println("Connecting to MQTT...");

if (client.connect("NodeMCUClient", mqttUser, mqttPassword)) {

Serial.println("Connected to MQTT");

} else {

Serial.print("Failed, rc=");

Serial.print(client.state());

Serial.println(" Retrying in 5 seconds...");

delay(5000);

}

}

}

void loop() {

if (!client.connected()) {

reconnect();

}

// Read sensor data or other data to be published

float sensorData = 25.5; // Replace with your data

// Convert the data to a string

String payload = String(sensorData, 2);

// Publish the data to the MQTT topic

client.publish(mqttTopic, payload.c\_str());

delay(5000); // Adjust the publish rate as needed

}

Step 6: Upload the Code to NodeMCU

- Connect your NodeMCU to your computer, select the correct board and port in the Arduino IDE, and upload the code to your NodeMCU.

Step 7: Monitor in IBM Cloud

- In your IBM Cloud account, you can monitor the data being published by the NodeMCU to the specified MQTT topic.

This code example provides a basic framework for connecting a NodeMCU to IBM Cloud via MQTT. For a complete implementation, you would need to integrate your specific sensors and adapt the code accordingly.

**INTEGRATING VARIOUS SENSORS:**

Integrating various sensors, including temperature sensors, a camera, motion sensors, voltage sensors, and smart home devices, into a comprehensive smart home system is a complex project. Here are the general steps to guide you through this process:

Step 1: Define Use Cases and Requirements

- Identify the specific use cases for your smart home, such as security, energy management, or convenience. Determine the types of sensors and devices you need for each use case.

Step 2: Select Hardware

- Choose the sensors and devices that match your requirements. Ensure they are compatible with your chosen platform for smart home automation.

Step 3: Plan Your Network

- Design your home network to accommodate the various sensors and devices. Ensure they have access to a stable Wi-Fi connection or other suitable communication networks.

Step 4: Set Up Your Central Hub or Platform

- Choose a central hub or platform for your smart home system. This could be a smart home automation hub or a custom solution like a Raspberry Pi or NodeMCU. Configure the hub to manage the sensors and devices.

Step 5: Connect Temperature Sensors

- Install and configure temperature sensors at relevant locations in your home. Connect them to the central hub. Implement data collection and monitoring for temperature control.

Step 6: Integrate Cameras

- Choose IP cameras that support streaming protocols like RTSP or MJPEG. Set up the cameras and configure them to provide video feeds. Connect them to your central hub or cloud service for remote viewing and storage.

Step 7: Install Motion Sensors

- Position motion sensors at strategic locations for security and automation. Configure them to detect motion and trigger actions or alerts. Connect them to your central hub.

Step 8: Voltage Monitoring

- If you want to monitor voltage levels in your home, consider using IoT-based voltage sensors. These can be connected to your central hub to monitor power usage or voltage drops.

Step 9: Smart Home Devices

- Integrate smart home devices such as smart lights, smart locks, and smart thermostats. Ensure they are compatible with your central hub or chosen platform.

Step 10: Implement Automation and Control Logic

- Write automation scripts or rules to control your smart home devices based on sensor data. For example, you can set up rules to turn on lights when motion is detected or adjust the thermostat based on temperature sensor data.

Step 11: Configure Remote Access

- Set up remote access to your smart home system. This may include creating a mobile app or using an existing one to monitor and control your devices and view camera feeds from anywhere.

Step 12: Security and Privacy

- Implement robust security measures to protect your smart home system. Use strong passwords, encryption, and regular software updates. Pay attention to data privacy and compliance.

Step 13: Testing and Optimization

- Test your entire smart home system to ensure that all sensors, devices, and automation rules work as expected. Optimize your system for better performance and efficiency.

Step 14: Documentation

- Document your smart home setup, including device configurations, network details, and automation rules. This documentation will be valuable for troubleshooting and future expansion.

Step 15: Ongoing Maintenance

- Regularly maintain your smart home system, ensuring that sensors are calibrated, devices are updated, and security measures are up to date. Be prepared to troubleshoot and address issues as they arise.

Please note that this is a high-level overview, and each step can be quite detailed and may require specific knowledge and skills, depending on the hardware and software you choose for your smart home system.

**CODE IMPLEMENTATION:**

Connecting a temperature sensor (DS18B20), a motion sensor (PIR), and a voltage sensor to a NodeMCU and sending the data to IBM Cloud using MQTT. Please keep in mind that this is a basic example, and you would need to adapt and expand the code to suit your specific sensors, hardware connections, and IBM Cloud configuration.

Required Components:

- NodeMCU (ESP8266)

- DS18B20 temperature sensor

- PIR motion sensor

- Voltage sensor

- IBM Cloud account for MQTT

Arduino Code for NodeMCU:

#include <ESP8266WiFi.h>

#include <PubSubClient.h>

#include <OneWire.h>

#include <DallasTemperature.h>

// Wi-Fi and MQTT configuration

const char\* ssid = "YourWiFiSSID";

const char\* password = "YourWiFiPassword";

const char\* mqttServer = "broker.ibmcloud.com";

const int mqttPort = 1883;

const char\* mqttUser = "YourMQTTUser";

const char\* mqttPassword = "YourMQTTPassword";

const char\* temperatureTopic = "home/temperature";

const char\* motionTopic = "home/motion";

const char\* voltageTopic = "home/voltage";

// DS18B20 temperature sensor

OneWire oneWire(D2); // Data pin

DallasTemperature sensors(&oneWire);

// PIR motion sensor

const int pirSensorPin = D1;

// Voltage sensor (adjust values according to your sensor)

const float voltageDivider = 2.0; // Voltage divider ratio (R1 / (R1 + R2))

const float voltageReference = 5.0; // Voltage reference (usually 5V for NodeMCU)

const int voltageSensorPin = A0;

WiFiClient wifiClient;

PubSubClient client(wifiClient);

void setup() {

Serial.begin(115200);

setupWiFi();

setupSensors();

client.setServer(mqttServer, mqttPort);

}

void setupWiFi() {

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.println("Connecting to WiFi...");

}

Serial.println("Connected to WiFi");

}

void setupSensors() {

sensors.begin(); // Initialize temperature sensor

pinMode(pirSensorPin, INPUT); // Initialize PIR motion sensor

}

void loop() {

if (!client.connected()) {

reconnect();

}

client.loop();

readAndSendTemperature();

readAndSendMotion();

readAndSendVoltage();

}

void reconnect() {

while (!client.connected()) {

if (client.connect("NodeMCUClient", mqttUser, mqttPassword)) {

Serial.println("Connected to MQTT");

} else {

Serial.print("Failed to connect, rc=");

Serial.println(client.state());

delay(5000);

}

}

}

void readAndSendTemperature() {

sensors.requestTemperatures();

float temperatureC = sensors.getTempCByIndex(0);

if (temperatureC != -127.00) {

String payload = String(temperatureC);

client.publish(temperatureTopic, payload.c\_str());

}

}

void readAndSendMotion() {

int motionState = digitalRead(pirSensorPin);

String payload = (motionState == HIGH) ? "Motion Detected" : "No Motion";

client.publish(motionTopic, payload.c\_str());

}

void readAndSendVoltage() {

int rawValue = analogRead(voltageSensorPin);

float voltage = (rawValue / 1023.0) \* voltageReference / voltageDivider;

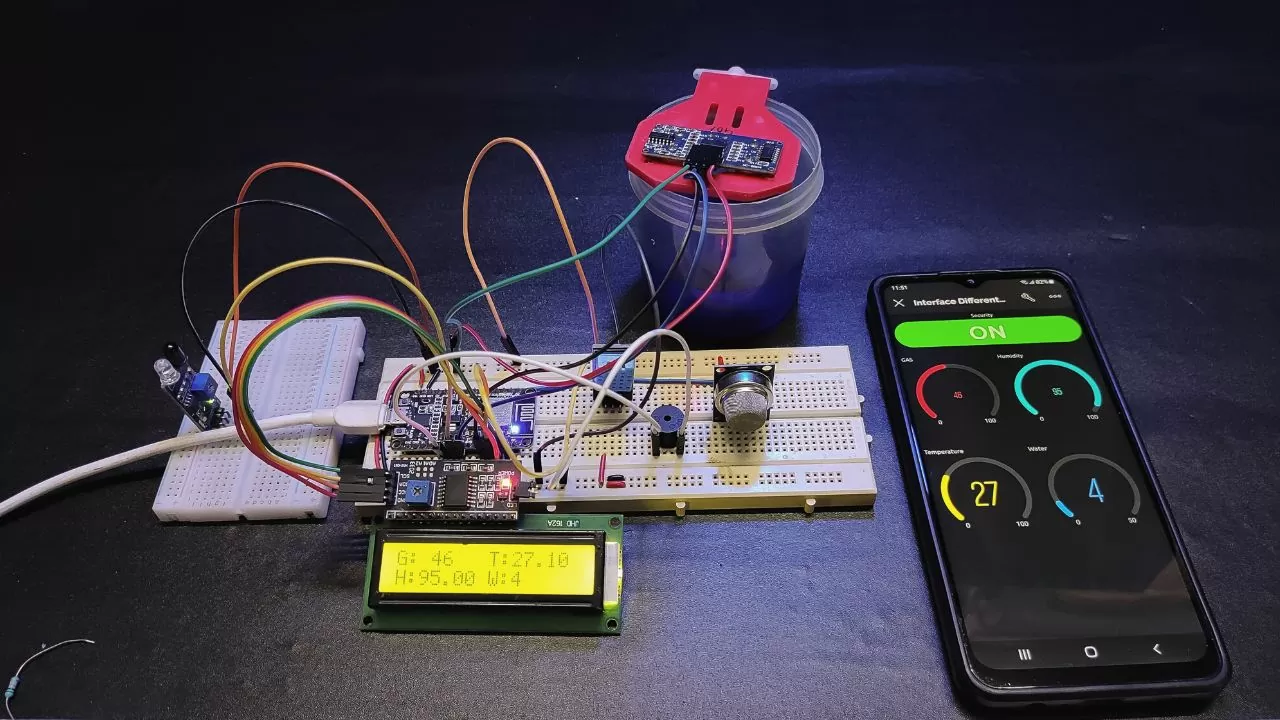
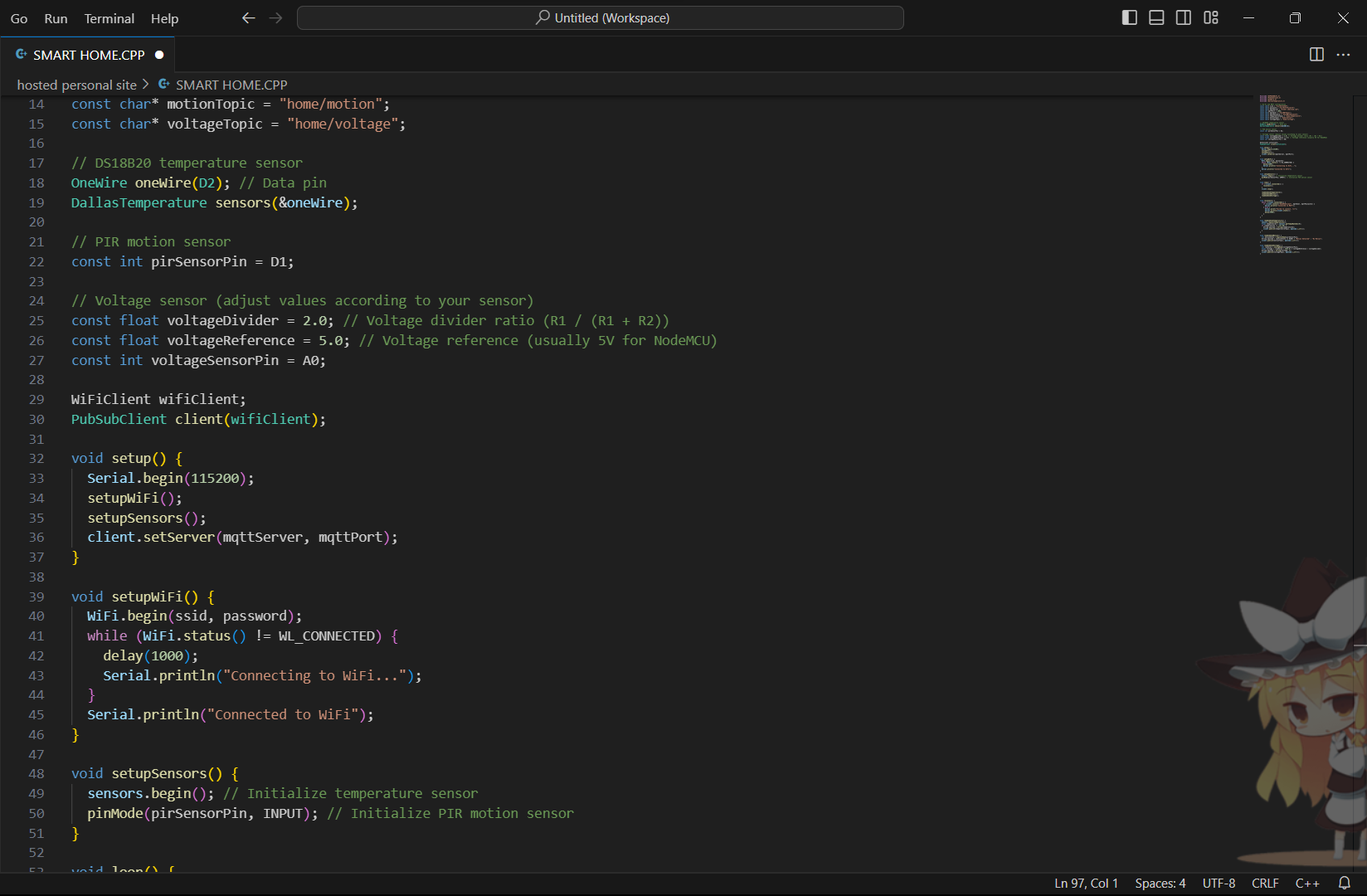
String payload = String(voltage, 2);

client.publish(voltageTopic, payload.c\_str());

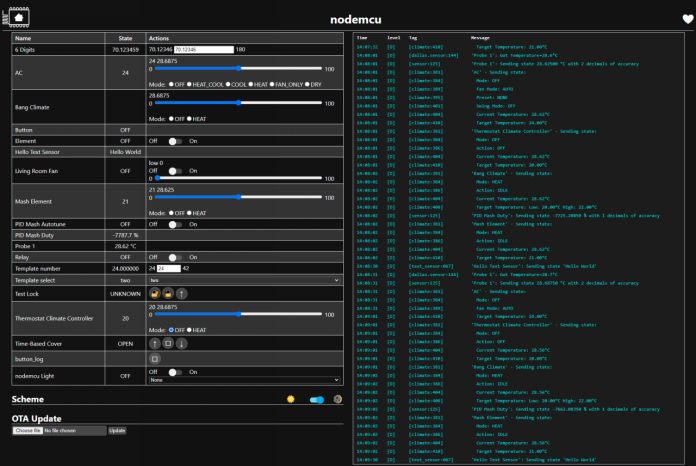
}

This code connects the NodeMCU to your Wi-Fi, reads data from the temperature sensor (DS18B20), motion sensor (PIR), and voltage sensor, and sends this data to IBM Cloud via MQTT. You will need to set up the corresponding topics and configuration in your IBM Cloud account. Also, make sure your sensors are correctly connected to the NodeMCU's pins.

**SAMPLE IMPLEMENTATION :**

****

**SAMPLE OUTPUT:**



**CONCLUSION:**

In this document, we have explored the process of integrating NodeMCU, a versatile microcontroller, with IBM Cloud to create a connected smart system with various sensors. This project enables you to monitor and control a range of sensors, including temperature sensors, motion sensors, and voltage sensors, all within a cloud-based environment. The key takeaways and concluding points are as follows:

1. Use Case and Requirements: It's essential to start by defining your use case and specific requirements for your smart system. This will guide your sensor and device selection, as well as the overall project scope.

2. Hardware Selection: Choose sensors and devices that match your requirements and are compatible with your chosen platform for smart home automation.

3. Network Planning: Design your home network to provide reliable and secure connectivity for your sensors and devices. Stable Wi-Fi access or alternative communication networks are crucial.

4. Central Hub or Platform: Select a central hub or platform for your smart system. This could be a commercial smart home hub or a custom solution, like a Raspberry Pi or NodeMCU. Configure the hub to manage your sensors and devices.

5. Sensor Integration: Install and configure sensors at relevant locations in your home, ensuring they are correctly connected to your central hub.

6. Automation and Control Logic: Write automation scripts or rules to control your smart devices based on sensor data. This allows you to create responsive and intelligent automation for your home.

7. Remote Access: Set up remote access to your smart system, allowing you to monitor and control devices and view sensor data from anywhere. This often involves creating a mobile app or using existing ones.

8. Security and Privacy: Implement robust security measures to protect your smart system and the data it generates. Utilise strong passwords, encryption, and regular software updates. Prioritise data privacy and compliance.

9. Testing and Optimization: Thoroughly test your system to ensure that sensors, devices, and automation rules work as expected. Optimise your system for better performance and efficiency.

10. Documentation: Maintain detailed documentation of your smart home setup. This documentation is valuable for troubleshooting and future expansion.

11. Ongoing Maintenance: Regularly maintain your system, ensuring sensors are calibrated, devices are updated, and security measures are kept up to date. Be prepared to address issues as they arise.

In summary, the integration of NodeMCU with IBM Cloud and various sensors opens up a world of possibilities for creating a smarter, more responsive home. By following the steps outlined in this document and adapting them to your specific needs, you can build a robust and efficient smart system that enhances your quality of life, provides security, and conserves energy. With the right sensors, devices, and thoughtful design, you can enjoy the benefits of a truly connected and intelligent home.