```
//Imports the secrets file that contains unknown values .
//Code that needs to be run on esp32 to convert it into a gateway that can enable two way
//communication between modbus slave device and AWS iot core (coverts modbus protocol to
//mqtt and vice versa)
#include "secrets.h"
#include <WiFi.h>
#include <ModbusIP_ESP8266.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <string>

using namespace std;
// Modbus Settings
const int REG = 0;
IPAddress IP_ModbusSERVER(10, 50, 59, 226);
const int number_REG = 8; // number of registers we are using on the server
ModbusIP mbClient;
uint16_t res[8]; // used to store the values that will be sent to the server

WiFiClientSecure net;
PubSubClient client(net); // create a pubSub client on top of tcp (wifi)

void connectWiFi()
{
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  Serial.println("Connecting to Wi-Fi");

  // if we are not connected ... will be printed
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  // if connected
  Serial.println("");
  Serial.println("WiFi Connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void connectModbus()
```

```
{
  mbClient.client(); // Initialize the Modbus client
}

void connectAWS()
{
  // Configure WiFiClientSecure to use the AWS IoT device credentials
  net.setCACert(AWS_CERT_CA);
  net.setCertificate(AWS_CERT_CRT);
  net.setPrivateKey(AWS_CERT_PRIVATE);

  // Connect to the MQTT broker on the AWS endpoint we defined earlier
  client.setServer(AWS_IOT_ENDPOINT, 8883);

  // Create a message handler for when the message is received
  client.setCallback(messageHandler);

  Serial.print("Connecting to AWS IoT");

  while (!client.connect(THINGNAME))
  {
    Serial.print(".");
    delay(1000);
  }

  if (!client.connected())
  {
    Serial.println("AWS IoT Timeout!");
    return;
  }

  // Subscribe to a topic
  client.subscribe("esp32/sub");
  Serial.println("AWS IoT Connected!");
}
// Used to read data that is sent from the device
void readModbusData()
{
  if (mbClient.isConnected(IP_ModbusSERVER))
  {
    mbClient.readHreg(IP_ModbusSERVER, REG, res, number_REG); // Read number_REG
number of holding registers from Modbus slave server and put it into the reg array
  }
  else
```

```
  {
    mbClient.connect(IP_ModbusSERVER); // Connect to the Modbus slave server
  }

  mbClient.task(); // Perform Modbus communication tasks
}

// To conver the data in the reg arry into a json document and pubish that data to AWS iot core
void convertAndPublishData()
{
  StaticJsonDocument<256> doc;
  // adding a time feild to the json document
  doc["time"] = millis();
  // adding the reg values to the json document
  for (int i = 0; i < number_REG; i++)
  {
    doc[String(i)] = res[i]; // Add Modbus data to the JSON document
  }

  char jsonBuffer[512];
  serializeJson(doc, jsonBuffer); // Serialize the JSON document to a string

  client.publish("esp32/pub", jsonBuffer); // Publish the JSON message to the "esp32/pub" topic
}
// To receive json from the AWS iot core
//  currently it requires the json to have age (int), name (string), isActive(bool) feilds
// the values in each of these feilds are sent to various regester on the modbus slave server
(details inside the function)
void messageHandler(char *topic, byte *payload, unsigned int length)
{
  // receive the message
  Serial.print("Incoming: ");
  Serial.println(topic);

  StaticJsonDocument<200> doc;
  DeserializationError error = deserializeJson(doc, payload); // Deserialize the JSON payload

  if (error)
  {
    Serial.print("Error parsing JSON: ");
    Serial.println(error.c_str());
    return;
  }
```

```
  // Assuming doc is the JSON document
  // uint16_t is only format that we can sent to modbus slave server using modbus protocol
(atleast using the writeHreg function)
  uint16_t numberValue = 0; // to store age feild
  char stringValue[5];      // to store name
  uint16_t boolValue = 0;   // to store isActive

  if (doc.containsKey("age"))
  {
    numberValue = doc["age"].as<uint16_t>();
  }

  if (doc.containsKey("name"))
  {
    // to convert the name string to char array and enter into stringValue array
    const char *nameValue = doc["name"].as<const char *>();
    strncpy(stringValue, nameValue, sizeof(stringValue) - 1);
    stringValue[sizeof(stringValue) - 1] = '\0'; // Null-terminate the string
  }
  // arr will store the values of the stringValue by coverting char to uint16_t
  uint16_t arr[5] = {0}; // Initialize the array elements to 0
  for (int i = 0; i < 5 && stringValue[i] != '\0'; i++)
  {
    // chat->uint16_t
    arr[i] = stringValue[i];
  }

  if (doc.containsKey("isActive"))
  {
    // bool to uint16_t may work better if the bool already comes as a 0 or 1 integer value
    boolValue = doc["isActive"].as<uint16_t>();
  }
  // sending all the values to regesters on the modbus slave
  if (mbClient.isConnected(IP_ModbusSERVER))
  {
    // sending age to the 8th(number_REG) register of the slave device using only 1 register
    mbClient.writeHreg(IP_ModbusSERVER, number_REG, &numberValue /*pointer to the
value*/, 1);
    // seding name to  9th to 14th registers of the slave device uisng 5 registers in total
    mbClient.writeHreg(IP_ModbusSERVER, number_REG + 1, arr /*pointer to register holding
the ascii values*/, 5);
    // sending the isActive to the 15th register of the slave device uisng 1 register
    mbClient.writeHreg(IP_ModbusSERVER, number_REG + 6, &boolValue, 1);
  }
```

```
  else
  {
    mbClient.connect(IP_ModbusSERVER); // Connect to the Modbus slave server
  }

  mbClient.task(); // Perform Modbus communication tasks
  Serial.println("Updated Modbus data with received JSON");
}

//----------------------------------------------------------------------
void setup()
{
  Serial.begin(9600);
  connectWiFi();   // Connect to Wi-Fi
  connectModbus(); // Connect to the Modbus slave server
  connectAWS();    // Connect to AWS IoT Core
}

void loop()
{
  client.loop(); // Handle incoming and outgoing MQTT messages

  readModbusData();        // Read data from Modbus slave server
  convertAndPublishData(); // Convert and publish data to AWS IoT Core

  delay(1000);
}
```