```
// NANO-021, mqtt data to Googlesheet
//last update: 03/16/25, 12:50PM

#include <WiFiNINA.h>         // WiFi support for Arduino Nano 33 IoT
#include <PubSubClient.h>     // MQTT client library

// WiFi and MQTT settings
const char* ssid = "ENTER_HERE";                          // WiFi SSID
const char* password = "ENTER_HERE";           // WiFi password
const char* mqtt_server = "192.168.1.241";           // Local MQTT broker IP
const int mqtt_port = 1883;                           // MQTT port

// MQTT Topics to subscribe to
const char* topicList[] = {
  "mqtt.date2", "mqtt.time2", "mqtt.ecowittOA", "mqtt.burnerState2",
  "mqtt.infraRedTemp_Boiler",
  "mqtt.hws_boiler", "mqtt.hwr_boiler",
  "mqtt.zone1_HWS", "mqtt.zone2_HWS", "mqtt.zone3_HWS",
  "mqtt.spaceTemp.MBR", "mqtt.spaceTemp.LR", "spaceTemp7",
  "mqtt.projTherms", "mqtt.projCost", "mqtt.solarRadiation", "mppt.pv2InWatts",
  "pd.invInWatts", "pd.pv2ChargeWatts", "pd.wattsInSum", "pd.soc", "pd.wattsOutSum"
};
const int numTopics = sizeof(topicList) / sizeof(topicList[0]); // Auto-calculate number of topics

// Client instances
WiFiClient espClient;               // Basic TCP client for MQTT
PubSubClient client(espClient);     // MQTT client using espClient
WiFiSSLClient httpsClient;          // Secure HTTPS client for Google Sheets

// Variables to hold incoming MQTT data
char dateMessage[12] = "00-00-0000";
char timeMessage[10] = "00:00:00";
float outdoorAirTemp = 0.0;
char burnerStatus[10] = "OFF";
float infraredTemp = 0.0;
float hws_boiler = 0.0;
float hwr_boiler = 0.0;
float zone1_HWS = 0.0;
float zone2_HWS = 0.0;
float zone3_HWS = 0.0;
float spaceTempMBR = 0.0;
float spaceTempLR = 0.0;
float spaceTemp7 = 0.0;
float projTherms = 0.0;
float projCost = 0.0;
float solarRadiation = 0.0;
float pv2InWatts = 0.0;
float invInWatts = 0.0;
float pv2ChargeWatts = 0.0;
float wattsInSum = 0.0;
float soc = 0.0;
float wattsOutSum = 0.0;

static unsigned long lastGoogleUpdate = 0;         // Last time Google Sheets was updated
unsigned long updateInterval = 120000;             // Update interval for Google Sheets (2 min)
unsigned long lastWiFiOK = 0;                      // Tracks last known good WiFi connection time

// Function: Ensure WiFi is connected or attempt to reconnect
void checkWiFiConnection() {
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("⚠ WiFi lost. Forcing full reset...");

        WiFi.end();               // Force WiFi chip reset
        delay(1000);
        WiFi.begin(ssid, password);

```

```
        unsigned long startTime = millis();
        while (WiFi.status() != WL_CONNECTED && millis() - startTime < 10000) {
            delay(500);
            Serial.print(".");
        }

        if (WiFi.status() == WL_CONNECTED) {
            Serial.println("\n☑ WiFi reconnected.");
        } else {
            Serial.println("\n✗ WiFi reconnection failed.");
        }
    }
}

// Function: Connect or reconnect to MQTT broker and resubscribe
void reconnectMQTT() {
    if (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        if (client.connect("Arduino_021")) {  // Unique client ID
            Serial.println("☑ Connected!");
            for (int i = 0; i < numTopics; i++) {
                client.subscribe(topicList[i]);
                Serial.print("☑ Subscribed: ");
                Serial.println(topicList[i]);
            }
        } else {
            Serial.print("✗ Failed, rc=");
            Serial.print(client.state());
            Serial.println(" - will retry later");
        }
    }
}

// Function: Handle incoming MQTT messages and update local variables
void callback(char* topic, byte* payload, unsigned int length) {
    char message[length + 1];
    memcpy(message, payload, length);
    message[length] = '\0';

    Serial.print("Received MQTT Message - Topic: ");
    Serial.print(topic);
    Serial.print(" | Message: ");
    Serial.println(message);

    // Match topic and store values
    if (strcmp(topic, "mqtt.date2") == 0) {
        strncpy(dateMessage, message, sizeof(dateMessage) - 1);
    } else if (strcmp(topic, "mqtt.time2") == 0) {
        strncpy(timeMessage, message, sizeof(timeMessage) - 1);
    } else if (strcmp(topic, "mqtt.ecowittOA") == 0) {
        outdoorAirTemp = atof(message);
    } else if (strcmp(topic, "mqtt.burnerState2") == 0) {
        strncpy(burnerStatus, message, sizeof(burnerStatus) - 1);
    } else if (strcmp(topic, "mqtt.infraRedTemp_Boiler") == 0) {
        infraredTemp = atof(message);
    } else if (strcmp(topic, "mqtt.hws_boiler") == 0) {
        hws_boiler = atof(message);
    } else if (strcmp(topic, "mqtt.hwr_boiler") == 0) {
        hwr_boiler = atof(message);
    } else if (strcmp(topic, "mqtt.zone1_HWS") == 0) {
        zone1_HWS = atof(message);
    } else if (strcmp(topic, "mqtt.zone2_HWS") == 0) {
        zone2_HWS = atof(message);
    } else if (strcmp(topic, "mqtt.zone3_HWS") == 0) {
        zone3_HWS = atof(message);
    } else if (strcmp(topic, "mqtt.spaceTemp.MBR") == 0) {
        spaceTempMBR = atof(message);
    } else if (strcmp(topic, "mqtt.spaceTemp.LR") == 0) {
        spaceTempLR = atof(message);
```

```
    } else if (strcmp(topic, "spaceTemp7") == 0) {
        spaceTemp7 = atof(message);
    } else if (strcmp(topic, "mqtt.projTherms") == 0) {
        projTherms = atof(message);
    } else if (strcmp(topic, "mqtt.projCost") == 0) {
        projCost = atof(message);
    } else if (strcmp(topic, "mqtt.solarRadiation") == 0) {
        solarRadiation = atof(message);
    } else if (strcmp(topic, "mppt.pv2InWatts") == 0) {
        pv2InWatts = atof(message);
    } else if (strcmp(topic, "pd.invInWatts") == 0) {
        invInWatts = atof(message);
    } else if (strcmp(topic, "pd.pv2ChargeWatts") == 0) {
        pv2ChargeWatts = atof(message);
    } else if (strcmp(topic, "pd.wattsInSum") == 0) {
        wattsInSum = atof(message);
    } else if (strcmp(topic, "pd.soc") == 0) {
        soc = atof(message);
    } else if (strcmp(topic, "pd.wattsOutSum") == 0) {
        wattsOutSum = atof(message);
    } else {
        Serial.println("⚠ Unrecognized MQTT Topic Received!");
    }
}

// Function: Send current sensor data to Google Sheets via HTTPS POST
void sendToGoogleSheet() {
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("❌ WiFi not connected. Trying to reconnect...");
        checkWiFiConnection();
        if (WiFi.status() != WL_CONNECTED) {
            Serial.println("❌ Still offline. Skipping send.");
            return;
        } else {
            Serial.println("☑ Reconnected. Proceeding with send...");
        }
    }

    Serial.print("🕒 sendToGoogleSheet() called at millis: ");
    Serial.println(millis());

    const char* host = "script.google.com";
    int httpsPort = 443;

    // Replace with your actual Google Apps Script endpoint URL
    String url = "https://script.google.com/ENTER_HERE";

    httpsClient.stop(); // Clear any existing session
    if (!httpsClient.connect(host, httpsPort)) {
        Serial.println("❌ HTTPS connection to Google Sheets failed.");
        return;
    }

    // Construct JSON payload
    String payload;
    payload.reserve(512);  // Reduce memory fragmentation
    payload = "{";
    payload += "\"date\":\"" + String(dateMessage) + "\",";
    payload += "\"time\":\"" + String(timeMessage) + "\",";
    payload += "\"OA\":" + String(outdoorAirTemp) + ",";
    payload += "\"burnerStatus\":\"" + String(burnerStatus) + "\",";
    payload += "\"infraredTemp\":" + String(infraredTemp) + ",";
    payload += "\"hws_boiler\":" + String(hws_boiler) + ",";
    payload += "\"hwr_boiler\":" + String(hwr_boiler) + ",";
    payload += "\"zone1_HWS\":" + String(zone1_HWS) + ",";
    payload += "\"zone2_HWS\":" + String(zone2_HWS) + ",";
    payload += "\"zone3_HWS\":" + String(zone3_HWS) + ",";
    payload += "\"spaceTempMBR\":" + String(spaceTempMBR) + ",";
    payload += "\"spaceTempLR\":" + String(spaceTempLR) + ",";
```

```
    payload += "\"spaceTemp7\":" + String(spaceTemp7) + ",";
    payload += "\"projTherms\":" + String(projTherms) + ",";
    payload += "\"projCost\":" + String(projCost) + ",";
    payload += "\"solarRadiation\":" + String(solarRadiation) + ",";
    payload += "\"pv2InWatts\":" + String(pv2InWatts) + ",";
    payload += "\"invInWatts\":" + String(invInWatts) + ",";
    payload += "\"pv2ChargeWatts\":" + String(pv2ChargeWatts) + ",";
    payload += "\"wattsInSum\":" + String(wattsInSum) + ",";
    payload += "\"soc\":" + String(soc) + ",";
    payload += "\"wattsOutSum\":" + String(wattsOutSum);
    payload += "}";

    // Send HTTPS POST request
    httpsClient.println("POST " + url + " HTTP/1.1");
    httpsClient.println("Host: script.google.com");
    httpsClient.println("Content-Type: application/json");
    httpsClient.print("Content-Length: ");
    httpsClient.println(payload.length());
    httpsClient.println();
    httpsClient.println(payload);

    Serial.println("📤 POST sent, awaiting response...");

    // Wait for response (timeout in 3 sec)
    unsigned long timeout = millis();
    while (httpsClient.connected() && !httpsClient.available()) {
        if (millis() - timeout > 3000) {
            Serial.println("❌ Timeout waiting for Google response.");
            httpsClient.stop();
            return;
        }
    }

    // Read response line-by-line
    while (httpsClient.available()) {
        String response = httpsClient.readStringUntil('\n');
        Serial.println("📄 Google Response: " + response);
    }

    Serial.println("🔚 Closing HTTPS session.");
    httpsClient.stop();
}

// Arduino Setup: Connect to WiFi and MQTT
void setup() {
    Serial.begin(115200);
    delay(1000);
    Serial.println("\n🔌 Starting Arduino Nano 33 IoT...");

    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\n☑ WiFi connected!");

    client.setServer(mqtt_server, mqtt_port);
    client.setCallback(callback);
    Serial.println("☑ MQTT client initialized");
}

// Arduino Loop: Maintain system, send data, and reconnect if needed
void loop() {
    unsigned long currentMillis = millis();

    // Check WiFi every 30 seconds
    static unsigned long lastWiFiCheck = 0;
    if (currentMillis - lastWiFiCheck >= 30000) {
        checkWiFiConnection();
```

```
            lastWiFiCheck = currentMillis;
        }

        // Reconnect MQTT every 10 seconds if needed
        static unsigned long lastMQTTCheck = 0;
        if (currentMillis - lastMQTTCheck >= 10000) {
            reconnectMQTT();
            lastMQTTCheck = currentMillis;
        }

        // Send to Google Sheets every updateInterval
        if (currentMillis - lastGoogleUpdate >= updateInterval) {
            Serial.println("📤 Time to send data to Google Sheets...");
            sendToGoogleSheet();
            lastGoogleUpdate = millis();
        }

        // Print heartbeat every 10 seconds
        static unsigned long lastBeat = 0;
        if (currentMillis - lastBeat > 10000) {
            Serial.println("❤️ Loop heartbeat alive...");
            lastBeat = currentMillis;
        }

        // Watchdog: Restart if WiFi offline too long
        static uint8_t previousWiFiStatus = WL_IDLE_STATUS;
        uint8_t currentWiFiStatus = WiFi.status();
        if (currentWiFiStatus == WL_CONNECTED) {
            if (previousWiFiStatus != WL_CONNECTED) {
                Serial.println("🔄 WiFi reconnected — resetting lastWiFiOK timer");
            }
            lastWiFiOK = millis();
        }
        previousWiFiStatus = currentWiFiStatus;

        if (currentWiFiStatus != WL_CONNECTED && millis() - lastWiFiOK > 600000) {
            Serial.println("⚠️ WiFi offline for 10+ minutes — rebooting Nano...");
            NVIC_SystemReset();  // Perform a soft reboot
        }

        // Keep MQTT connection alive
        client.loop();
    }
```