#include <SPI.h> // Include SPI library for serial peripheral interface communication

#include <WiFiNINA.h> // Include library for WiFi support on Arduino boards

#include <PubSubClient.h> // MQTT client library for publishing/subscribing messages over MQTT

#include <OneWire.h> // Include OneWire library for communication with OneWire devices

#include <DallasTemperature.h> // Library for Dallas temperature sensors

// WiFi credentials and MQTT server details

const char\* ssid = "psu-iot";

const char\* password = "y2nfu9jih82q";

const char\* mqttServer = "41.193.5.154";

const int mqttPort = 24500;

const char\* mqttClientName = "MKRWiFi1010Client";

// MQTT topics for publishing sensor data

const char\* tdsTopic = "tdsSensorData";

const char\* tempTopic = "temperatureSensorData";

const char\* waterLevelTopic = "waterLevelSensorData";

WiFiClient wifiClient;

PubSubClient client(wifiClient);

// Sensor pins configuration

#define TdsSensorPin A0

#define TempSensorPin A1

#define WaterLevelSensorPin A2

#define VREF 5.0 // Reference voltage for analog readings

#define SCOUNT 30 // Sample count for averaging

#define ONE\_WIRE\_BUS 4 // Pin for OneWire bus

OneWire oneWire(ONE\_WIRE\_BUS); // Setup OneWire bus

DallasTemperature sensors(&oneWire); // Create a DallasTemperature object

// Variables for sensor data processing

int analogBuffer[SCOUNT];

int analogBufferIndex = 0;

float averageVoltage = 0;

float tdsValue = 0;

void setup() {

Serial.begin(9600); // Start serial communication

setupWiFi(); // Connect to WiFi

setupMQTT(); // Setup MQTT connection

sensors.begin(); // Initialize temperature sensor

}

void loop() {

handleWiFi(); // Check and maintain WiFi connection

handleMQTT(); // Check and maintain MQTT connection

readSensors(); // Read sensor data

processSensorData(); // Process and publish sensor data

delay(2000); // Wait for 2 seconds before next loop iteration

}

void setupWiFi() {

WiFi.begin(ssid, password); // Connect to WiFi network

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

delay(500);

Serial.print(".");

}

Serial.println("Connected to WiFi");

}

void setupMQTT() {

client.setServer(mqttServer, mqttPort); // Set the MQTT server and port

client.setCallback(callback); // Set the callback function for MQTT messages

while (!client.connected()) {

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

if (client.connect(mqttClientName)) {

Serial.println("Connected to MQTT");

// Subscribe to topics if necessary

client.subscribe(tdsTopic);

client.subscribe(tempTopic);

client.subscribe(waterLevelTopic);

} else {

Serial.print("Failed to connect to MQTT with state ");

Serial.println(client.state());

delay(2000);

}

}

}

void handleWiFi() {

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

Serial.println("WiFi disconnected. Reconnecting...");

setupWiFi(); // Reconnect to WiFi

}

}

void handleMQTT() {

if (!client.connected()) {

Serial.println("MQTT disconnected. Reconnecting...");

setupMQTT(); // Reconnect to MQTT

}

client.loop(); // Process MQTT loop

}

void readSensors() {

// Reading TDS sensor

static unsigned long analogSampleTimepoint = millis();

if (millis() - analogSampleTimepoint > 40U) {

analogSampleTimepoint = millis();

analogBuffer[analogBufferIndex] = analogRead(TdsSensorPin); // Read TDS sensor value

analogBufferIndex = (analogBufferIndex + 1) % SCOUNT;

// Note: Reading of water level sensor is done in processSensorData()

}

}

void processSensorData() {

static unsigned long printTimepoint = millis();

if (millis() - printTimepoint > 800U) {

printTimepoint = millis();

// Process and publish TDS sensor data

averageVoltage = calculateAverageVoltage(); // Calculate average voltage from samples

float compensationCoefficient = 1.0 + 0.02 \* (getTemperature() - 25.0); // Temperature compensation

float compensationVoltage = averageVoltage / compensationCoefficient;

tdsValue = calculateTDS(compensationVoltage); // Calculate TDS value

Serial.print("TDS Value:");

Serial.print(tdsValue, 0);

Serial.println("ppm");

publishData(tdsValue, tdsTopic); // Publish TDS data to MQTT

// Process and publish temperature sensor data

float temperature = sensors.getTempCByIndex(0); // Get temperature from sensor

Serial.print("Temperature:");

Serial.print(temperature);

Serial.println("°C");

publishData(temperature, tempTopic); // Publish temperature data to MQTT

// Read and publish water level sensor data

int waterLevelValue = analogRead(WaterLevelSensorPin); // Read water level sensor

publishData(waterLevelValue, waterLevelTopic); // Publish water level data to MQTT

Serial.print("Water Level:");

Serial.print(waterLevelValue);

}

}

float calculateAverageVoltage() {

// Calculate average voltage from analog buffer

float sum = 0;

for (int i = 0; i < SCOUNT; i++) {

sum += analogBuffer[i];

}

return sum / SCOUNT \* VREF / 1024.0; // Convert sum to voltage

}

float calculateTDS(float voltage) {

// Calculate TDS value based on voltage reading

return (133.42 \* voltage \* voltage \* voltage - 255.86 \* voltage \* voltage + 857.39 \* voltage) \* 0.5;

}

void publishData(float data, const char\* topic) {

// Publish sensor data to the specified MQTT topic

char messageBuffer[50];

snprintf(messageBuffer, 50, "%.2f", data);

if (client.publish(topic, messageBuffer)) {

Serial.println("Publish succeeded");

} else {

Serial.println("Publish failed");

}

}

void callback(char\* topic, byte\* payload, unsigned int length) {

// Callback function to handle incoming MQTT messages (if needed)

}

float getTemperature() {

// Request and return temperature reading from temperature sensor

sensors.requestTemperatures();

return sensors.getTempCByIndex(0);

}