main.cpp

```cpp
#include <WiFi.h>
#include <HTTPClient.h>
#include "config.h"
#include <ArduinoJson.h>

const char *ssid = WIFI_SSID; // Get SSID from config.h
const char *pass = WIFI_PASS; // Get PASSWORD from config.h
int counter = 0;

// Definición de pines
const int humiditySensorPin = 34; // Pin para el sensor de humedad HL-69
const int ldrPin = 32;            // Pin para el sensor LDR
const int lm35Pin = 33;           // Pin para el sensor LM35
const int numSamples = 10;        // Número de muestras para promediar

// Variables para el promedio móvil (Humedad)
const int numReadings = 20;          // Número de lecturas para el promedio
float humidityReadings[numReadings]; // Array para las lecturas de humedad
int humidityIndex = 0;               // Índice actual para la humedad
float totalHumidity = 0;             // Suma total para la humedad
float averageHumidity = 0;           // Humedad promedio

// Variables para el promedio móvil (20 segundos) para lm35
const int numSegundos = 20;      // Número de lecturas de 20 segundos
float lm35Readings[numSegundos]; // Array para las lecturas de temperatura
int lm35Index = 0;               // Índice actual para la temperatura
float totalTemp = 0;             // Suma total de temperatura
float averageTemp = 0;           // Temperatura promedio

// Variables para el promedio móvil (20 segundos) para luz
const int numLuxReadings = 20;     // Número de lecturas de 20 segundos para luz
float luxReadings[numLuxReadings]; // Array para las lecturas de luz
int luxIndex = 0;                  // Índice actual para la luz
float totalLux = 0;                // Suma total de luz
float averageLux = 0;              // Luz promedio

// Variables para calcular 5 minutos
unsigned long previousMillis = 0; // Almacena el último tiempo en el que se ejecutó la
función
const unsigned long interval = 0; // 2 minutos en milisegundos

const int plant_id = 1;

// Constantes para el rango de valores
float MAX_TEMP;
float MIN_TEMP;
float MAX_LUX;
float MIN_LUX;
float MAX_HUM;
float MIN_HUM;

unsigned long lastLuxAlertTime;
```

```cpp
unsigned long lastTempAlertTime;
unsigned long lastHumAlertTime;

void alerta(String message);

void connect_wifi()
{
    while (WiFi.status() != WL_CONNECTED)
    {
        WiFi.disconnect(true);
        WiFi.begin(ssid, pass);

        while (WiFi.status() != WL_CONNECTED)
        {
            delay(500);
            Serial.print(".");
            counter++;
            if (counter >= 60)
            { // 30 seconds timeout
                Serial.println("");
                Serial.print("Retrying connecting to ");
                Serial.println(ssid);
                counter = 0;
                break;
            }
        }
    }

    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
}

void get_plant_range()
{
    Serial.println("Getting plant ranges");

    if (WiFi.status() == WL_CONNECTED)
    { // Check WiFi connection status
        HTTPClient http;
        String url = "http://grillos.synology.me:8080/getrange/" + String(plant_id); // Update to POST URL
        http.begin(url);

        // Send the POST request with JSON data
        int httpResponseCode = http.GET();

        if (httpResponseCode > 0)
        {
            // Success, print the response
            String response = http.getString();
            Serial.println("HTTP Response code: " + String(httpResponseCode));
            Serial.println("Received JSON: ");
```

```
            Serial.println(response);

            // Parse JSON response
            DynamicJsonDocument doc(1024); // Adjust size as necessary
            DeserializationError error = deserializeJson(doc, response);

            if (!error)
            {
                // Extract maxTemp and minTemp from the JSON object
                MAX_TEMP = doc["max_temp"];
                MIN_TEMP = doc["min_temp"];
                MAX_LUX = doc["max_luz"];
                MIN_LUX = doc["min_luz"];
                MAX_HUM = doc["max_hum"];
                MIN_HUM = doc["min_hum"];
                const char *nombre = doc["nombre"];

                // Print the extracted values
                Serial.println("Plant Type: " + String(nombre));
                Serial.println("Max Humidity: " + String(MAX_HUM));
                Serial.println("Min Humidity: " + String(MIN_HUM));
                Serial.println("Max Light Level: " + String(MAX_LUX));
                Serial.println("Min Light Level: " + String(MIN_LUX));
                Serial.println("Max Temperature: " + String(MAX_TEMP));
                Serial.println("Min Temperature: " + String(MIN_TEMP));
            }
            else
            {
                Serial.println("Failed to parse JSON: " + String(error.c_str()));
            }
        }
        else
        {
            // Failure, print the error
            Serial.println("Error on HTTP request: " + String(httpResponseCode));
        }

        // End the HTTP connection
        http.end();
    }
    else
    {
        connect_wifi(); // Ensure Wi-Fi is connected
    }
}

void setup()
{
    Serial.begin(115200);
    delay(1000);

    Serial.println();
    Serial.print("Connecting to network: ");
    Serial.println(ssid);
```

```
159
160     WiFi.disconnect(true);
161     WiFi.begin(ssid, pass);
162
163     while (WiFi.status() != WL_CONNECTED)
164     {
165         delay(500);
166         Serial.print(".");
167         counter++;
168         if (counter >= 60)
169         { // 30 seconds timeout
170             ESP.restart();
171         }
172     }
173
174     Serial.println("");
175     Serial.println("WiFi connected");
176     Serial.println("IP address: ");
177     Serial.println(WiFi.localIP());
178
179     get_plant_range();
180
181     unsigned long currentMil = millis();
182     unsigned long lastLuxAlertTime = 0;
183     unsigned long lastTempAlertTime = 0;
184     unsigned long lastHumAlertTime = 0;
185     Serial.println(lastLuxAlertTime);
186     // Inicializar los arrays de lecturas a 0
187     for (int i = 0; i < numReadings; i++)
188     {
189         humidityReadings[i] = 0;
190     }
191     for (int i = 0; i < numSegundos; i++)
192     {
193         lm35Readings[i] = 0;
194     }
195 }
196
197 void medirLuz()
198 {
199     // ---- LUZ ----
200     // Leer el valor analógico del LDR
201     int lightSensorValue = analogRead(ldrPin);
202
203     // Convertir el valor analógico a voltaje
204     float lightVoltage = lightSensorValue * (3.3 / 4095.0); // Usar 3.3V
205
206     // Calcular lux a partir del valor de luz leído
207     float lux = (lightSensorValue / 4095.0) * 10000; // Ajustar el factor según el rango
    esperado
208
209     // ---- PROMEDIO MÓVIL DE 20 SEGUNDOS ----
210     totalLux -= luxReadings[luxIndex];          // Restar la lectura más antigua
211     luxReadings[luxIndex] = lux;                // Guardar la nueva lectura
```

```
212     totalLux += luxReadings[luxIndex];          // Añadir la nueva lectura
213     luxIndex = (luxIndex + 1) % numLuxReadings; // Mover al siguiente índice circular
214 
215     // Calcular el promedio de luz de los últimos 20 segundos
216     averageLux = totalLux / numLuxReadings;
217 
218     // Imprimir la luz en lux y el promedio
219     Serial.print("Luz: ");
220     Serial.print(lux, 2); // Imprimir la luz actual en lux
221     Serial.println(" lux");
222 
223     Serial.print("Promedio luz 20 segundos: ");
224     Serial.print(averageLux, 2); // Imprimir el promedio de luz
225     Serial.println(" lux");
226 }
227 
228 void medirTemperatura()
229 {
230     // ---- TEMPERATURA ----
231     float lm35Voltage = 0;
232 
233     // Tomar varias muestras para reducir fluctuaciones
234     for (int i = 0; i < numSamples; i++)
235     {
236         int sensorValue = analogRead(lm35Pin);     // Leer valor analógico del LM35DZ
237         lm35Voltage += sensorValue * (5 / 4095.0); // Convertir el valor ADC a voltaje
    (ajustado a 3.3V)
238         delay(10);                                 // Pequeño retardo entre lecturas
239     }
240     lm35Voltage /= numSamples; // Promediar las muestras
241 
242     // Convertir el voltaje a temperatura en grados Celsius
243     float temperatureC = lm35Voltage * 100 + 5; // 10mV por grado Celsius, multiplicado por
    100 para convertir
244 
245     // ---- PROMEDIO MÓVIL DE 20 SEGUNDOS ----
246     totalTemp -= lm35Readings[lm35Index];      // Restar la lectura más antigua
247     lm35Readings[lm35Index] = temperatureC;    // Guardar la nueva lectura
248     totalTemp += lm35Readings[lm35Index];      // Añadir la nueva lectura
249     lm35Index = (lm35Index + 1) % numSegundos; // Mover al siguiente índice circular
250 
251     // Calcular el promedio de temperatura de los últimos 20 segundos
252     averageTemp = totalTemp / numSegundos;
253 
254     // Imprimir la temperatura actual y la temperatura promedio
255     Serial.print("Temperatura: ");
256     Serial.print(temperatureC, 2); // Imprimir la temperatura actual con 2 decimales
257     Serial.println(" °C");
258 
259     Serial.print("Temperatura promedio 20 segundos: ");
260     Serial.print(averageTemp, 2); // Imprimir el promedio de 20 segundos
261     Serial.println(" °C");
262 }
263 
```

```cpp
void medirHumedad()
{
    // ---- HUMEDAD ----
    int humiditySensorValue = analogRead(humiditySensorPin);

    // Convertir el valor analógico a voltaje (3.3V es el voltaje de referencia)
    float humidityVoltage = humiditySensorValue * (3.3 / 4095.0); // Usar 3.3V

    // Convertir el voltaje a porcentaje de humedad (invertido)
    float humidityPercentage = 100 - (humidityVoltage / 3.3) * 100;
    humidityPercentage = map(humidityPercentage, 0, 70, 0, 100);

    // Aplicar el promedio móvil para la humedad
    totalHumidity -= humidityReadings[humidityIndex];     // Restar la lectura más antigua
    humidityReadings[humidityIndex] = humidityPercentage; // Guardar la nueva lectura
    totalHumidity += humidityReadings[humidityIndex];     // Añadir la nueva lectura
    humidityIndex = (humidityIndex + 1) % numReadings;    // Mover al siguiente índice

    // Calcular la humedad promedio
    averageHumidity = totalHumidity / numReadings;

    // Asegurarse de que el porcentaje esté dentro del rango 0-100
    if (averageHumidity < 0)
    {
        averageHumidity = 0;
    }
    else if (averageHumidity > 100)
    {
        averageHumidity = 100;
    }

    // Imprimir el valor instantáneo de humedad y el promedio
    Serial.print("Humedad instantánea: ");
    Serial.print(humidityPercentage, 2); // Imprimir la humedad instantánea en %
    Serial.println(" %");

    Serial.print("Humedad promedio: ");
    Serial.print(averageHumidity, 2); // Imprimir la humedad promedio en %
    Serial.println(" %");
}

// Función para ejecutar algo cada 5 minutos
void ejecutarFuncionCada5Minutos()
{
    Serial.println("Han pasado 2 minutos. Ejecutando función...");

    if (WiFi.status() == WL_CONNECTED)
    { // Check WiFi connection status
        HTTPClient http;
        String url = "http://grillos.synology.me:8080/logdata"; // Update to POST URL
        http.begin(url);

        // Set content type to JSON
        http.addHeader("Content-Type", "application/json");
```

```
318
319         Serial.println(averageHumidity);
320         Serial.println(String(averageHumidity));
321         // Create JSON data
322         String jsonData = "{\"plant_id\": \"" + String(plant_id) + "\", \"soil_humidity\": "
    + String(averageHumidity) + ", \"light_level\": " + String(averageLux) + ", \"temperature\":
    " + String(averageTemp) + "}";
323
324         // Send the POST request with JSON data
325         int httpResponseCode = http.POST(jsonData);
326
327         if (httpResponseCode > 0)
328         {
329             // Success, print the response
330             String response = http.getString();
331             Serial.println("HTTP Response code: " + String(httpResponseCode));
332             Serial.println("Received JSON: ");
333             Serial.println(response);
334         }
335         else
336         {
337             // Failure, print the error
338             Serial.println("Error on HTTP request: " + String(httpResponseCode));
339         }
340
341         // End the HTTP connection
342         http.end();
343     }
344     else
345     {
346         connect_wifi(); // Ensure Wi-Fi is connected
347     }
348 }
349
350 String serverUrl = "http://grillos.synology.me:8080/alert";
351
352 void alerta(String message)
353 {
354     if (WiFi.status() == WL_CONNECTED)
355     {
356         HTTPClient http;
357         http.begin(serverUrl);
358         http.addHeader("Content-Type", "application/json"); // Set header for JSON data
359
360         // Create JSON payload
361         StaticJsonDocument<200> doc;
362         doc["message"] = message;
363         doc["plant_id"] = plant_id;
364
365         String requestBody;
366         serializeJson(doc, requestBody);
367
368         // Send POST request
369         int httpResponseCode = http.POST(requestBody);
```

```
370
371         Serial.println("");
372         Serial.println("");
373         Serial.println("");
374         Serial.println("");
375         Serial.println("");
376         Serial.println("");
377         Serial.println("");
378         Serial.println(message);
379         Serial.println("Alert sent successfully!");
380         Serial.println("");
381         Serial.println("");
382         Serial.println("");
383
384         // Check response
385         if (httpResponseCode > 0)
386         { // 201 Created
387             String response = http.getString();
388             Serial.println("");
389             Serial.println("");
390             Serial.println("");
391             Serial.println("");
392             Serial.println("");
393             Serial.println("");
394             Serial.println("");
395             Serial.println("");
396             Serial.println("Alert sent successfully!");
397             Serial.println("Response: " + response);
398             Serial.println("");
399             Serial.println("");
400             Serial.println("");
401         }
402         else
403         {
404             Serial.print("Error sending alert, HTTP response code: ");
405             Serial.println(httpResponseCode);
406         }
407
408         http.end(); // Free resources
409     }
410     else
411     {
412         Serial.println("WiFi not connected");
413     }
414 }
415
416 const unsigned long alertInterval = 30000; // 30 secs in milliseconds
417
418 void checkRanges()
419 {
420     bool error = false;
421     String alert = "Alerta! Problemas con planta con id: " + String(plant_id) + "\n";
422     unsigned long currentMillis = millis(); // Get the current time
423
```

```
424     // Check Lux
425     if (averageLux < MIN_LUX && currentMillis - lastLuxAlertTime >= alertInterval)
426     {
427         alert += "Luz por debajo del rango minimo. Exponer a la planta a mas luz.\n";
428         error = true;
429         lastLuxAlertTime = currentMillis; // Update last alert time for Lux
430     }
431     else if (averageLux > MAX_LUX && currentMillis - lastLuxAlertTime >= alertInterval)
432     {
433         alert += "Luz por arriba del rango maximo. Alejar la planta de la luz.\n";
434         error = true;
435         lastLuxAlertTime = currentMillis; // Update last alert time for Lux
436     }
437 
438     // Check Temp
439     if (averageTemp < MIN_TEMP && currentMillis - lastTempAlertTime >= alertInterval)
440     {
441         alert += "Temperatura mas baja de lo recomendado. Subir la temperatura del
    ambiente.\n";
442         error = true;
443         lastTempAlertTime = currentMillis; // Update last alert time for Temp
444     }
445     else if (averageTemp > MAX_TEMP && currentMillis - lastTempAlertTime >= alertInterval)
446     {
447         alert += "Temperatura mas alta de lo recomendado. Bajar la temperatura del
    ambiente.\n";
448         error = true;
449         lastTempAlertTime = currentMillis; // Update last alert time for Temp
450     }
451 
452     // Check Hum
453     if (averageHumidity < MIN_HUM && currentMillis - lastHumAlertTime >= alertInterval)
454     {
455         alert += "Humedad de la tierra muy baja. Se recomienda regar la planta.\n";
456         error = true;
457         lastHumAlertTime = currentMillis; // Update last alert time for Humidity
458     }
459     else if (averageHumidity > MAX_HUM && currentMillis - lastHumAlertTime >= alertInterval)
460     {
461         alert += "Humedad de la tierra muy alta.\n";
462         error = true;
463         lastHumAlertTime = currentMillis; // Update last alert time for Humidity
464     }
465 
466     // Send alert if there is an error
467     if (error)
468     {
469         Serial.print(alert);
470         alerta(alert);
471     }
472 }
473 
474 void loop()
475 {
```

```
476     unsigned long currentMillis = millis(); // Tiempo actual
477 
478     // Comprueba si han pasado 5 minutos (300,000 ms)
479     if (currentMillis - previousMillis >= interval)
480     {
481         previousMillis = currentMillis; // Actualiza el tiempo para la próxima comparación
482 
483         ejecutarFuncionCada5Minutos(); // Llama a la función que quieres ejecutar cada 5
    minutos
484     }
485 
486     // Llamar a las funciones que miden luz, temperatura y humedad
487     medirLuz();
488     medirTemperatura();
489     medirHumedad(); // La función de humedad ya estaba implementada
490 
491     checkRanges();
492 
493     Serial.print("");
494     Serial.println("==============================");
495 
496     // Esperar un segundo antes de la siguiente lectura
497     delay(1000);
498 }
```