INFORMAÇÕES PARA CRIAÇÃO DO RELATÓRIO

1. CÓDIGO C++

. Implementação sistema **WIFI** para conexão com a rede e envio do de dados.

*A biblioteca **WiFi** para ESP32 é uma ferramenta oficial fornecida pela Espressif (fabricante do ESP32) que facilita a conexão de microcontroladores ESP32 a redes Wi-Fi. Essa biblioteca é amplamente utilizada em projetos de IoT devido à capacidade integrada do ESP32 de se conectar à internet.*

. Implementação **PubSubClient** permite a comunicação entre dispositivos IoT e servidores usando o protocolo MQTT.

O PubSubClient é uma biblioteca para Arduino que permite a comunicação entre dispositivos IoT e servidores usando o protocolo MQTT (Message Queuing Telemetry Transport). Ele é leve e eficiente, projetado para dispositivos com recursos limitados, como microcontroladores ESP8266 ou ESP32.

Principais características:

- *Publicação e assinatura: Permite que dispositivos enviem (publish) mensagens para tópicos e recebam (subscribe) mensagens de tópicos específicos.*
- *Conexão persistente: Mantém uma conexão TCP ativa para troca contínua de mensagens.*
- *Customizável: Oferece opções para gerenciar QoS (Qualidade de Serviço) e autenticação com nome de usuário e senha.*
- *Leve: Ideal para sistemas embarcados devido ao uso eficiente de memória.*

A biblioteca é amplamente utilizada em projetos de automação residencial e IoT para comunicação em tempo real.

. Implementação da biblioteca **ArduinoJson** para enviar os dados coletados dos sensores no formato JSON, padronizando para que qualquer sistema possa receber os dados para análise.

*A biblioteca **ArduinoJson** é uma ferramenta poderosa e eficiente para manipulação de dados no formato JSON em microcontroladores como Arduino e ESP32. Desenvolvida com foco em dispositivos com recursos limitados, ela é amplamente utilizada em projetos IoT para facilitar a serialização, a desserialização e o processamento de dados JSON.*

. Implementação das bibliotecas **WiFiUdp** e **NTPClient** para sincronização do tempo do servidor NTP, assim obter data e hora local.

*As bibliotecas **WiFiUdp** e **NTPClient** são amplamente utilizadas em projetos que requerem a sincronização de tempo via servidores NTP (Network Time Protocol), especialmente em dispositivos IoT como ESP8266 e ESP32.*

. Implementação do display LCD I2C para visualização dos dados capturados dos sensores.

*A biblioteca **LiquidCrystal_I2C** é usada para controlar displays LCD conectados por meio de um módulo I2C (Inter-Integrated Circuit), simplificando a comunicação entre microcontroladores, como Arduino e ESP32, e displays LCD de 16x2, 20x4 ou similares.*

. Melhorias para **otimização da memória ESP32**

1. **char date[11]; e char time[9]; em vez de String**

- **Comentário**: "Substituí String por char para economizar memória."
- **Motivo**: Usar char em vez de String economiza memória, especialmente em sistemas embarcados, pois o tipo String pode ser mais pesado e consumir mais recursos do heap. O uso de char mantém o controle manual da memória e é mais eficiente.

2. **const uint16_t mqtt_port = 1883;**

- **Comentário**: "Usando uint16_t para economizar memória."
- **Motivo**: O tipo uint16_t ocupa menos memória do que um int padrão (geralmente 4 bytes). Isso é relevante quando se trabalha com microcontroladores que têm recursos limitados.

3. **uint16_t moistureThreshold = 600;**

- **Comentário**: "Representando umidade como inteiro (600 = 60.0%)."
- **Motivo**: Representar a umidade como um valor inteiro em vez de um float reduz o uso de memória, pois um inteiro de 16 bits ocupa menos memória do que um float de 32 bits.

4. **uint16_t phMin = 60; e uint16_t phMax = 80;**

- **Comentário**: "pH representado como inteiro (6.0 -> 60)."
- **Motivo**: Como no caso da umidade, a representação do pH como inteiros evita o uso de float, economizando memória.

5. **uint8_t phosphorusDetected = 0; e uint8_t potassiumDetected = 0;**

- **Comentário**: "Usando uint8_t para flags booleanas."
- **Motivo**: O tipo uint8_t ocupa apenas 1 byte, o que é mais eficiente para representar variáveis booleanas, comparado ao tipo bool, que pode ter uma implementação mais pesada dependendo do compilador.

6. **Uso de snprintf para criar clientId em vez de concatenação de strings**

- **Comentário**: "Usando snprintf para segurança."
- **Motivo**: O uso de snprintf evita problemas com overflow de buffer e permite o controle mais preciso sobre o espaço de memória, em comparação com a concatenação de strings, que pode ser mais vulnerável ao consumo excessivo de memória.