Arquitetura do Sistema

O sistema é composto por uma rede de sensores sem fio (WSN) operando na topologia estrela, onde um

nó sensor de baixo consumo comunica-se com um Gateway central.

- **Protocolo RF:** Proprietário via NRF24L01+ (2.4GHz) com ACK Payloads.
- **Protocolo de Transporte:** MQTT (TCP/IP) sobre Wi-Fi.
- **Aplicação:** Node-RED para visualização e controle.

Bibliotecas e Dependências

BIBLIOTECA	CABEÇALHO	UTILIZAÇÃO
RF24	`RF24.h`	Controle do transceptor SPI NRF24L01+.
PubSubClient	`PubSubClient.h`	Cliente MQTT para ESP32.
ArduinoJson	`ArduinoJson.h`	Serialização e Deserialização de JSON.
RTClib	`RTClib.h`	Comunicação I2C com RTC DS3231.
DHT Sensor Lib	`DHT.h`	Leitura de sensores DHT11/22.
ESP32 WS2812	`ESP32_WS2812_Lib.h`	Controle do LED RGB Built-in (RMT Driver).
ESP-IDF RTC IO	`driver/rtc_io.h`	Funções de baixo nível para isolamento de GPIO no Deep Sleep.

Hardware: Gateway (ESP32-S3)

O Gateway atua como ponte transparente. Não entra em modo de suspensão.

PINO / MACRO	GPIO	FUNÇÃO	DESCRIÇÃO
CE_PIN	14	**OUTPUT**	Chip Enable do Rádio. Controla TX/RX mode.
CSN_PIN	10	**OUTPUT**	Chip Select Not (SPI). Seleção do escravo.

PINO / MACRO	GPIO	FUNÇÃO	DESCRIÇÃO
LED_PIN	48	OUTPUT	LED RGB Endereçável (WS2812).
MOSI	11	SPI	Master Out Slave In.
MISO	13	SPI	Master In Slave Out.
SCK	12	SPI	Serial Clock.

Hardware: Nó Sensor (ESP32-S3)

Dispositivo de baixo consumo alimentado a bateria.

PINO / MACRO	GPIO	FUNÇÃO	DESCRIÇÃO
DHT_PIN	4	INPUT	Dados do sensor DHT11.
LDR_PIN	5	ANALOG	Leitura ADC do sensor de luz.
RTC_SQW	1	WAKEUP	Pino de interrupção do RTC DS3231 (Active LOW).
BTN_PIN	2	WAKEUP	Botão de acionamento manual (Active LOW).
SDA	8	I2C	Dados I2C (RTC).
SCL	9	I2C	Clock I2C (RTC).

Protocolo de Comunicação RF

A comunicação utiliza pacotes binários de tamanho fixo (structs C++) para eficiência.

DataPacket (Sensor -> Gateway)

```
struct DataPacket {
    uint8_t nodeId;        // Identificador do nó (Ex: 1)
    float temperatura;     // Valor em graus Celsius
    float umidade;         // Valor em porcentagem
    int luminosidade;      // Valor bruto ADC (0-4095)
    uint32_t timestamp;    // Unix Epoch Time (segundos desde 1970)
    bool isManualRead;     // true = Botão, false = Timer RTC
};
```

ConfigPacket (Gateway -> Sensor)

Enviado embutido no pacote de confirmação (ACK Payload).

```
struct ConfigPacket {
    uint32_t serverTime;      // Reservado para sincronização futura
    uint32_t newIntervalMin;  // Novo intervalo de sono (minutos). 0 = sem alteração.
};
```

Firmware Gateway: Variáveis Globais

TIPO	NOME	DESCRIÇÃO
RF24	`radio`	Instância de controle do hardware NRF24L01+.
PubSubClient	`client`	Cliente MQTT.
ESP32_WS2812	`strip`	Controlador do LED RGB.
const uint64_t	`pipeAddress`	Endereço do pipe de comunicação (0xE8E8F0F0E1LL).
ConfigPacket	`pendingConfig`	Buffer que armazena a configuração recebida via MQTT, aguardando conexão do sensor.

Firmware Gateway: Funções Principais

void setup()

Inicializa o sistema. Configurações críticas:

- **LED:** Inicializa biblioteca RMT e define cor Vermelha (Boot).
- **WiFi/MQTT:** Chama funções de conexão.
- **Radio:**
  - Canal 76 (2476 MHz).
  - Taxa de dados 1MBPS.
  - `enableAckPayload()`: Habilita envio de dados no ACK.
  - `writeAckPayload()`: Pré-carrega o buffer com dados vazios para a primeira transação.

void loop()

Loop infinito de processamento:

1. Verifica conexão MQTT. Se cair, chama `reconnect()`.
2. Executa `client.loop()` para manter Keep-Alive do MQTT.
3. Verifica `radio.available()`. Se houver dados:
   - Lê o `DataPacket`.
   - Altera LED para Azul (Recebendo).
   - Converte Timestamp Unix para formato legível (struct tm).
   - Imprime log na Serial.
   - Chama `publicarDadosMQTT()`.
   - Recarrega o buffer de ACK com `pendingConfig`.
   - Limpa `pendingConfig.newIntervalMin` para evitar reenvio duplicado.

void mqtt_callback(char* topic, byte* payload, uint length)

Callback disparado assincronamente quando o Broker envia uma mensagem para o tópico subscrito.

- Deserializa o JSON recebido.
- Verifica se existe a chave `"intervalo"`.
- Se existir, atualiza `pendingConfig.newIntervalMin`.
- Chama `radio.writeAckPayload` imediatamente para garantir que a próxima comunicação do sensor já pegue o novo valor.

Firmware Gateway: Funções Auxiliares

void publicarDadosMQTT(DataPacket d)

Converte a estrutura binária C++ em JSON string.

- Cria documento `StaticJsonDocument<256>`.
- Mapeia campos: node, temp, umid, lux, ts, tipo.
- Publica no tópico `fazenda/estufa/dados`.

void setup_wifi()

Tenta conectar ao Wi-Fi em loop bloqueante (while). Imprime pontos na serial até conectar.

void reconnect()

Gerencia reconexão MQTT.

- Entra em loop enquanto `!client.connected()`.
- Tenta conectar com ID "ESP32_Gateway_WSN".
- Se conectar, subscreve ao tópico `fazenda/estufa/config`.
- Se falhar, aguarda 5 segundos antes de tentar novamente.

void setStatusColor(uint8_t r, uint8_t g, uint8_t b)

Wrapper para a biblioteca WS2812. Define a cor do LED no índice 0 e chama `show()`.

Firmware Sensor: Variáveis e Memória

Memória RTC (Persistente)

Variáveis marcadas com `RTC_DATA_ATTR` são salvas na memória RTC Slow Memory, que permanece alimentada durante o Deep Sleep.

TIPO	NOME	USO

TIPO	NOME	USO
uint32_t	`sleepIntervalMinutes`	Armazena o intervalo de sono configurado. Padrão inicial: 1 minuto. Persiste entre resets de Deep Sleep.

Variáveis Voláteis (RAM)

Reiniciadas a cada ciclo de acordar.

TIPO	NOME	USO
bool	`acordouPorBotao`	Flag para identificar a causa do despertar.
DataPacket	`payload`	Estrutura de dados para envio.
ConfigPacket	`receivedConfig`	Estrutura para receber resposta do Gateway.

Firmware Sensor: Ciclo de Vida

Devido ao uso de Deep Sleep, a função `loop()` é vazia. Toda a lógica ocorre no `setup()`.

Sequência de Execução (Setup):

1. **Configuração de Energia:** `esp_sleep_pd_config` mantém o domínio RTC ligado.
2. **Inicialização de Pinos:** Configura Botão e SQW como INPUT_PULLUP.
3. **Inicialização de Periféricos:** I2C (Wire), DHT, RTC DS3231 e Rádio NRF24.
4. **Configuração do Rádio:** Define potência máxima, retries (15 tentativas) e abre pipe de escrita.
5. **Lógica Principal:**
   - `verificarWakeup()`: Descobre por que acordou.
   - `lerSensores()`: Coleta dados ambientais.
   - `enviarDados()`: Transmite e verifica resposta.
   - `agendarProximoAcordar()`: Reprograma o RTC.
6. **Debounce:** Se acordou por botão, aguarda o botão ser solto.
7. **Dormir:** Chama `entrarEmDeepSleep()`.

Firmware Sensor: Funções Detalhadas

void verificarWakeup()

Identifica a fonte do despertar.

- Usa `esp_sleep_get_wakeup_cause()`.
- Se a causa for `ESP_SLEEP_WAKEUP_EXT1` (Múltiplos pinos RTC):
  - Lê a máscara de bits com `esp_sleep_get_ext1_wakeup_status()`.
  - Verifica bitwise (`&`) se o bit do `BTN_PIN` está ativo.
  - Define `acordouPorBotao = true` se for botão, caso contrário assume Timer RTC.

void lerSensores()

Popula a struct `payload`.

- Lê temperatura e umidade do DHT11. Trata valores `NaN` como 0.0.
- Lê luminosidade via `analogRead(LDR_PIN)`.
- Obtém data/hora atual do RTC (`rtc.now()`) e converte para Unix Timestamp.
- Preenche campo `isManualRead` baseado na verificação de wakeup.

void enviarDados()

Gerencia a transação RF.

- Chama `radio.write(&payload, sizeof(payload))`. Esta função bloqueia até receber ACK ou timeout.
- Se retornar `true` (Sucesso):
  - Verifica `radio.isAckPayloadAvailable()`.
  - Se houver payload no ACK, lê para a struct `receivedConfig`.
  - Se `receivedConfig.newIntervalMin > 0`, atualiza a variável persistente `sleepIntervalMinutes`.
- Chama `radio.powerDown()` para economizar energia imediatamente.

void agendarProximoAcordar()

Interage com o DS3231.

- Limpa alarmes pendentes (`clearAlarm`).
- Calcula o tempo futuro: `rtc.now() + TimeSpan(minutos)`.
- Define o **Alarme 1** para disparar quando hora, minuto e segundo coincidirem (Match Date).

- O disparo do alarme coloca o pino SQW do DS3231 em nível LOW.

> **void entrarEmDeepSleep()**

Prepara o ESP32 para desligar.

- **Isolamento de Pinos:** Usa `rtc_gpio_pullup_en()` nos pinos BTN e SQW. Isso é crucial para que os pinos não flutuem e consumam energia ou gerem falsos disparos enquanto o núcleo digital está desligado.
- **Configuração de Wakeup:** `esp_sleep_enable_ext1_wakeup(mask, ESP_EXT1_WAKEUP_ANY_LOW)`. Configura o ESP32 para acordar se qualquer um dos pinos da máscara for para nível LOW.
- Chama `esp_deep_sleep_start()`. O processador para aqui e só reinicia no `setup()`.

Node-RED: Fluxo e Lógica

Recebimento de Dados (Uplink)

O fluxo inicia no nó `mqtt in` subscrito em `fazenda/estufa/dados`.

- **JSON Parser:** Converte string para objeto JS.
- **Function (Distribuidor):**
  - Extrai propriedades do payload.
  - Aplica correção de fuso horário: `ts + 10800` (Adiciona 3 horas ao timestamp UTC para compensar fuso ou erro de RTC).
  - Gera data formatada (Locale String).
  - Retorna array de 5 mensagens para saídas distintas (Temperatura, Umidade, Lux, Texto Data, Texto Tipo).
- **Dashboard Nodes:** Gauges e Charts (ui-chart) plotam os valores recebidos.

Envio de Configuração (Downlink)

Controlado pelo nó `ui-slider`.

- **Slider:** Gera um valor numérico (1 a 60).
- **Function (Cria JSON):** Cria objeto `{"intervalo": valor}`.
- **MQTT Out:** Publica no tópico `fazenda/estufa/config`.