

BRUNO SCALERCIO MARQUES BORGES DA FONSECA – 1230112185

FELIPE CELESTRINO MONTEIRO – 1230112049

GABRIEL DUARTE RIZZO DO NASCIMENTO – 1230112376

LUCAS DE SOUZA NASCIMENTO NABA – 1230102780

RAPHAEL BÍSSIMOS COSTA DOS SANTOS – 1230113376

**SISTEMA INTELIGENTE DE MONITORAMENTO AMBIENTAL PARA
CONSERVAÇÃO DE EQUIPAMENTOS MÉDICOS**

Rio de Janeiro

2025

SUMÁRIO

1. Apresentação do projeto.....	3
2. Objetivos.....	4
3. Tecnologias Utilizadas.....	5
3.1 ESP32.....	5
3.2 Sensor DHT22.....	5
3.3 Protocolo MQTT.....	5
3.4 CloudAMQP (Broker MQTT).....	6
3.5 Dashboard Python.....	6
3.6 Biblioteca Matplotlib.....	6
3.7 Biblioteca Paho-MQTT.....	6
3.8 Display LCD 16x2 com Interface I2C.....	7
3.9 Linguagem C/C++ (Arduino Core para ESP32).....	7
4. Arquitetura do Sistema.....	8
4.1 Camada de Captura de Dados.....	8
4.2 Camada de Comunicação.....	9
4.3 Camada de Processamento e Intermediação (Broker).....	9
4.4 Camada de Aplicação (Software Desktop).....	10
4.5 Fluxo de Funcionamento do Sistema.....	10
5. Funcionamento do ESP32.....	11
5.1 Inicialização dos Componentes.....	11
5.2 Conexão com a Rede Wi-Fi.....	12
5.3 Comunicação com o Broker MQTT (CloudAMQP).....	12
5.4 Leitura Periódica do Sensor DHT22.....	13
5.5 Exibição das Informações no Display LCD.....	13
5.6 Publicação dos Dados via MQTT.....	14
6. Comunicação MQTT.....	14
6.1 Funcionamento do Protocolo MQTT.....	14
6.2 CloudAMQP como Broker MQTT.....	15
7. Fluxo de Dados.....	16
8. Anexos – Código Fonte Completo.....	17
8.1 ESP 32.....	17
8.2 Dashboard / Python.....	21

1. Apresentação do projeto

O projeto tem como finalidade o desenvolvimento de um Sistema de Monitoramento Ambiental para Conservação de Materiais de Uso Médico, utilizando tecnologias de Internet das Coisas (IoT) para garantir a manutenção adequada das condições ambientais de temperatura e umidade em ambientes destinados à armazenagem de itens sensíveis.

A proposta consiste na implementação de um dispositivo de borda baseado no microcontrolador ESP32, integrado a um sensor digital DHT22 para coleta contínua de dados. As informações captadas são transmitidas por meio do protocolo MQTT (Message Queuing Telemetry Transport) a uma **Aplicação Desktop Profissional em Python**, que permite a visualização em tempo real das medições.

Diferente de sistemas genéricos, esta solução integra perfis de materiais específicos (vacinas, insulinas, reagentes, antibióticos), realizando cálculos automáticos de estabilidade e emitindo alertas visuais de conformidade de acordo com os padrões farmacológicos de cada insumo. O projeto busca demonstrar uma solução acessível, robusta e segura, destacando boas práticas de arquitetura distribuída e interfaces gráficas modernas.

2. Objetivos

O projeto tem como objetivo desenvolver um sistema completo de monitoramento ambiental, capaz de realizar a leitura, o registro e a transmissão em tempo real de dados de temperatura e umidade.

O sistema visa garantir que os materiais médico-hospitalares permaneçam dentro dos parâmetros adequados de conservação. Para isso, são empregados sensores ambientais, um microcontrolador ESP32 com criptografia TLS e o protocolo MQTT para comunicação segura com o software de gestão.

Este objetivo geral se desdobra nos seguintes objetivos específicos:

- Monitorar continuamente temperatura e umidade do ambiente.
- Transmitir os dados para um software Desktop com interface gráfica avançada.
- Implementar perfis de segurança customizáveis (ex: Vacinas 2°C a 8°C).
- Calcular e exibir índice de estabilidade do armazenamento em tempo real.
- Emitir alertas visuais (Seguro, Atenção, Crítico) na interface de monitoramento.
- Garantir comunicação assíncrona e não-bloqueante entre o hardware e o software.

3. Tecnologias Utilizadas

O desenvolvimento do sistema empregou um conjunto de tecnologias de hardware e software que possibilitam a coleta, transmissão e análise profissional dos dados.

3.1 ESP32

O microcontrolador ESP32 foi utilizado como núcleo do sistema embarcado. Ele gerencia a leitura dos sensores e a conexão segura via Wi-Fi ([WiFiClientSecure](#)), permitindo a transmissão de dados criptografados para a nuvem.

3.2 Sensor DHT22

O DHT22 é um sensor digital de temperatura e umidade selecionado por sua precisão superior ao modelo DHT11, essencial para a validação de ambientes hospitalares e laboratoriais.

3.3 Protocolo MQTT

O MQTT (Message Queuing Telemetry Transport) é um protocolo de comunicação leve voltado para aplicações IoT. Baseado no modelo publish/subscribe, ele permite que o ESP32 publique dados em tópicos específicos e que clientes autorizados os recebam de forma eficiente. Sua baixa demanda de recursos o torna ideal para sistemas embarcados e redes com largura de banda limitada.

3.4 CloudAMQP (Broker MQTT)

O sistema utiliza o CloudAMQP como serviço de broker MQTT baseado em nuvem. O CloudAMQP oferece alta disponibilidade, conexões seguras e infraestrutura escalável, eliminando a necessidade de configurar e hospedar um broker local. Sua compatibilidade com o protocolo MQTT permite o roteamento eficiente das mensagens enviadas pelo ESP32 e recebidas pela interface web.

3.5 Dashboard Python

A interface de monitoramento foi desenvolvida em Python 3, utilizando o framework PyQt6. Esta escolha permite a criação de uma aplicação desktop nativa, performática e com visual moderno (estilos *Dark Mode* e *Light Mode*), superior a soluções web simples em termos de integração com o sistema operacional.

3.6 Biblioteca Matplotlib

Para a visualização de dados, foi utilizada a biblioteca Matplotlib integrada ao PyQt. Ela gera gráficos vetoriais em tempo real, permitindo a análise histórica da temperatura e umidade com renderização de alta qualidade.

3.7 Biblioteca Paho-MQTT

A biblioteca Paho-MQTT para Python foi responsável pela camada de rede da aplicação desktop. Ela gerencia a conexão assíncrona com o broker, garantindo que a interface gráfica não trave enquanto aguarda novas mensagens do sensor.

3.8 Display LCD 16x2 com Interface I2C

Foi empregado um display LCD conectado ao ESP32 para *feedback* local imediato, permitindo que técnicos verifiquem a temperatura e o status da conexão Wi-Fi diretamente no equipamento, sem necessidade de computador.

3.9 Linguagem C/C++ (Arduino Core para ESP32)

A programação do ESP32 foi realizada utilizando C/C++ por meio do Arduino Core, devido à vasta documentação, suporte da comunidade, variedade de bibliotecas disponíveis e facilidade de integração com sensores e protocolos de comunicação.

4. Arquitetura do Sistema

A arquitetura do sistema foi projetada com base nos princípios de Internet das Coisas (IoT), contemplando camadas distintas de captura de dados, comunicação, processamento e visualização. O objetivo é garantir confiabilidade, escalabilidade e facilidade de manutenção, mantendo o fluxo de dados contínuo entre os dispositivos físicos e a interface de monitoramento.

4.1 Camada de Captura de Dados

A camada de captura é composta pelos dispositivos físicos responsáveis pela coleta das variáveis ambientais. Ela inclui:

- ESP32, atuando como unidade de processamento embarcado;
- Sensor DHT22, responsável pela medição de temperatura e umidade;
- Display LCD 16x2 I2C, utilizado para exibição local dos dados coletados.

Nessa camada, o ESP32 realiza leituras periódicas do sensor e atualiza o display, garantindo que as informações estejam acessíveis de forma imediata.

4.2 Camada de Comunicação

A comunicação entre o ESP32 e a plataforma de monitoramento ocorre através do protocolo MQTT, utilizando o serviço CloudAMQP como broker.

Essa camada é responsável por:

- Gerenciar a conexão Wi-Fi do microcontrolador;
- Publicar os dados coletados em tópicos específicos;
- Garantir entrega, roteamento e autenticação das mensagens;
- Permitir que múltiplos clientes acessem os dados em tempo real.

4.3 Camada de Processamento e Intermediação (Broker)

A camada intermediária corresponde ao broker CloudAMQP, que atua como núcleo de distribuição das mensagens enviadas pelo ESP32. Suas funções incluem:

- Receber os valores de temperatura e umidade publicados;
- Armazenar temporariamente e repassar as mensagens aos clientes conectados;
- Manter escalabilidade e disponibilidade por ser uma solução em nuvem;
- Prover segurança adicional por meio de autenticação e certificados, quando habilitados.

4.4 Camada de Aplicação (Software Desktop)

A aplicação Python atua como o centro de inteligência. Diferente de um painel passivo, ela processa os dados recebidos:

- **Decodificação:** Transforma as cargas úteis MQTT em valores numéricos (`float`).
- **Análise:** Compara os valores com o `Profile` selecionado (ex: Vacinas).
- **Visualização:** Renderiza os gráficos e atualiza as barras de estabilidade.

4.5 Fluxo de Funcionamento do Sistema

O funcionamento do sistema pode ser resumido da seguinte forma:

1. O ESP32 realiza leituras periódicas do sensor DHT22;
2. Os valores obtidos são exibidos no display LCD local;
3. O ESP32 publica temperatura e umidade em tópicos MQTT no CloudAMQP;
4. O broker recebe e distribui as mensagens aos clientes conectados;
5. O Software Python recebe as mensagens e atualiza a exibição dos dados;
6. Os gráficos e indicadores são atualizados instantaneamente, permitindo monitoramento contínuo.

5. Funcionamento do ESP32

O código do ESP32 foi estruturado em C++ para eficiência máxima.

- **Setup:** Inicializa o display I2C, configura o pino do sensor e estabelece conexão segura (`setInsecure` para certificados autoassinados ou simplificados) com o Wi-Fi e o Broker.
- **Loop:** Verifica a conexão constantemente. A cada ciclo de 2 segundos, lê o DHT22. Se a leitura for válida (`!isnan`), atualiza o LCD e publica as mensagens MQTT.
- **Callback:** O firmware também escuta o tópico `climatizador/comando`, permitindo receber instruções remotas (como "ligar/desligar" um atuador).

5.1 Inicialização dos Componentes

Ao ser energizado, o ESP32 executa uma sequência de inicialização que inclui:

- Configuração das portas digitais, como o LED indicador de conexão Wi-Fi;
- Inicialização do sensor DHT22;
- Ativação e configuração do display LCD 16x2 via I2C;
- Configuração do módulo Wi-Fi interno;
- Configuração da comunicação MQTT por meio do cliente PubSubClient;
- Definição do callback responsável por interpretar comandos recebidos via MQTT.

5.2 Conexão com a Rede Wi-Fi

O ESP32 tenta se conectar à rede Wi-Fi previamente configurada no código.

Durante esse processo:

- Um LED pisca indicando tentativa de conexão;
- O módulo aguarda até 15 segundos para autenticação;
- Em caso de falha, uma mensagem de erro é exibida no LCD;
- Em caso de sucesso, o endereço IP obtido é apresentado brevemente no display.

A conexão Wi-Fi é um pré-requisito para toda a comunicação MQTT subsequente.

5.3 Comunicação com o Broker MQTT (CloudAMQP)

Após conectar-se à internet, o ESP32 estabelece comunicação com o broker MQTT.

O processo inclui:

- Autenticação utilizando *username* e *password* fornecidos pelo serviço CloudAMQP;
- Assinatura do tópico “climatizador/comando”, por onde comandos externos podem ser enviados ao dispositivo;
- Manutenção da conexão, com tentativas automáticas de reconexão caso ela seja perdida.

5.4 Leitura Periódica do Sensor DHT22

O microcontrolador realiza leituras de temperatura e umidade a cada 2 segundos. O fluxo envolve:

- Solicitação dos dados ao sensor;
- Armazenamento dos valores em variáveis internas;
- Verificação de integridade da leitura (tratamento de valores inválidos como *NaN*);
- Impressão dos dados no monitor serial para depuração.

5.5 Exibição das Informações no Display LCD

Após cada leitura válida, o ESP32 atualiza o display LCD com:

- Temperatura (com casa decimal);
- Umidade relativa do ar (valor inteiro);
- Símbolos especiais, como o grau Celsius (°), para melhor legibilidade.

5.6 Publicação dos Dados via MQTT

Com cada atualização de leitura, o ESP32 publica:

- A temperatura no tópico “climatizador/temperatura”;
- A umidade no tópico “climatizador/umidade”.

6. Comunicação MQTT

A comunicação entre o ESP32 e a interface web é realizada por meio do protocolo MQTT (Message Queuing Telemetry Transport), escolhido por sua leveza, eficiência e confiabilidade em ambientes de Internet das Coisas (IoT). O uso desse protocolo permite transmitir informações em tempo real com baixo consumo de banda e processamento.

6.1 Funcionamento do Protocolo MQTT

O MQTT opera segundo o modelo publish/subscribe, no qual:

- Dispositivos *publicam* mensagens em tópicos específicos;
- Clientes interessados *assinam* esses tópicos e recebem automaticamente as mensagens;
- Um broker central — no caso deste projeto, o *CloudAMQP* — gerencia toda a distribuição das mensagens.

6.2 CloudAMQP como Broker MQTT

O projeto utiliza o serviço CloudAMQP, uma plataforma em nuvem baseada em RabbitMQ com suporte nativo a MQTT. Ele foi adotado por oferecer:

- Alta disponibilidade e operação 24/7;
- Painel de controle para monitoramento das mensagens;
- Conexões seguras utilizando TLS/SSL (quando habilitado);
- Suporte a credenciais exclusivas de acesso;
- Confiabilidade e baixa latência.

7. Fluxo de Dados

O software desenvolvido trata de maneira inteligente os dados:

1. **Entrada:** O dado chega via callback do Paho-MQTT.
2. **Thread Safety:** Um sistema de `pyqtSignal` transfere o dado da thread de rede para a thread da interface gráfica.
3. **Lógica de Negócio:** O software consulta a classe `Profile`. Exemplo: Se o perfil for "Insulina" (2-8°C) e a leitura for 10°C, o sistema altera o status visual para "Crítico" e exibe alerta.
4. **Cálculo de Estabilidade:** Uma fórmula ponderada (65% temperatura, 35% umidade) gera um índice de 0 a 100% que indica a qualidade do armazenamento ao longo do tempo.

8. Anexos – Código Fonte Completo

8.1 ESP 32

```
#include <WiFi.h>
#include <WiFiClientSecure.h> // <-- ESSENCIAL para usar CloudAMQP (TLS)
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include "DHT.h"
#include <PubSubClient.h>

// ===== CONFIGURAÇÕES DE PINOS =====
#define LED_WIFI 4
#define DHTPIN 5
#define DHTTYPE DHT22

// ===== OBJETOS =====
DHT dht(DHTPIN, DHTTYPE);
LiquidCrystal_I2C lcd(0x27, 16, 2);

// ===== VARIÁVEIS =====
float temperatura = 0.0;
float umidade = 0.0;
unsigned long ultimoUpdate = 0;

// ===== CONFIG WI-FI =====
const char* ssid = "12345678";
const char* password = "abcd2945";

// ===== CONFIG MQTT (CloudAMQP) =====
const char* mqtt_server = "jaragua-01.lmq.cloudamqp.com"; // HOST
const int mqtt_port = 8883;
const char* mqtt_user = "refqurtq:refqurtq"; // USUARIO
const char* mqtt_pass = "JTeV6YUekaRgi6G0lbp7XNA_i3c71jqO"; // SENHA MQTT

WiFiClientSecure espClient; // Cliente TLS
PubSubClient client(espClient); // Cliente MQTT

// ===== CALLBACK =====
void mqttCallback(char* topic, byte* payload, unsigned int length) {
    String msg = "";
    for (int i = 0; i < length; i++) msg += (char)payload[i];
```

```

Serial.print("Comando recebido: ");
Serial.println(msg);

lcd.clear();
lcd.setCursor(0, 0);
lcd.print("CMD:");
lcd.print(msg);

if (msg == "ligar") {
  Serial.println("Climatizador LIGADO");
}
else if (msg == "desligar") {
  Serial.println("Climatizador DESLIGADO");
}
else if (msg.startsWith("set_temp:")) {
  String valor = msg.substring(9);
  Serial.print("Temp alvo = ");
  Serial.println(valor);
}
}

// ===== CONECTAR AO MQTT =====
void conectarMQTT() {
  while (!client.connected()) {
    Serial.print("Conectando ao MQTT... ");

    if (client.connect("ESP32Lucas", mqtt_user, mqtt_pass)) {
      Serial.println("Conectado!");
      client.subscribe("climatizador/comando");
    } else {
      Serial.print("Erro = ");
      Serial.print(client.state());
      Serial.println(" | Tentando de novo em 5s...");
      delay(5000);
    }
  }
}

// ===== CONECTAR AO WI-FI =====
void conectarWiFi() {
  WiFi.begin(ssid, password);
  Serial.print("Conectando ao WiFi");
  lcd.clear();
  lcd.setCursor(0, 0);

```

```

lcd.print("Conectando WiFi");

unsigned long inicio = millis();

while (WiFi.status() != WL_CONNECTED) {
  digitalWrite(LED_WIFI, !digitalRead(LED_WIFI));
  delay(500);
  Serial.print(".");
  if (millis() - inicio > 15000) {
    Serial.println("\nFalha na conexao WiFi!");
    lcd.clear();
    lcd.print("WiFi Falhou!");
    digitalWrite(LED_WIFI, LOW);
    return;
  }
}

digitalWrite(LED_WIFI, HIGH);
Serial.println("\nWiFi Conectado!");
Serial.print("IP: ");
Serial.println(WiFi.localIP());

lcd.clear();
lcd.setCursor(0, 0);
lcd.print("WiFi Conectado!");
lcd.setCursor(0, 1);
lcd.print(WiFi.localIP().toString());
delay(2000);
lcd.clear();
}

// ===== SETUP =====
void setup() {
  Serial.begin(115200);
  pinMode(LED_WIFI, OUTPUT);
  digitalWrite(LED_WIFI, LOW);

  dht.begin();
  lcd.init();
  lcd.backlight();

  conectarWiFi();

  // MQTT
  espClient.setInsecure(); // Ignora certificado TLS (necessário CloudAMQP)

```

```

    client.setServer(mqtt_server, mqtt_port);
    client.setCallback(mqttCallback);
}

// ===== LOOP =====
void loop() {
    if (!client.connected()) conectarMQTT();
    client.loop();

    if (millis() - ultimoUpdate >= 2000) {
        ultimoUpdate = millis();

        temperatura = dht.readTemperature();
        umidade = dht.readHumidity();

        if (isnan(temperatura) || isnan(umidade)) {
            Serial.println("Falha ao ler DHT!");
            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print("Erro no Sensor!");
            return;
        }

        Serial.print("Temp: ");
        Serial.print(temperatura);
        Serial.print(" | Umidade: ");
        Serial.println(umidade);

        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("T: ");
        lcd.print(temperatura, 1);
        lcd.print((char)223);
        lcd.print("C ");
        lcd.print("U: ");
        lcd.print(umidade, 0);
        lcd.print("%");

        client.publish("climatizador/temperatura", String(temperatura).c_str());
        client.publish("climatizador/umidade", String(umidade).c_str());
    }
}

```

8.2 Dashboard / Python

```
import sys
import ssl
from datetime import datetime
from collections import deque
from dataclasses import dataclass

from PyQt6.QtWidgets import (
    QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout,
    QLabel, QComboBox, QPushButton, QFrame, QGridLayout, QSpacerItem,
    QSizePolicy, QGroupBox, QProgressBar, QScrollArea
)
from PyQt6.QtCore import Qt, QTimer, pyqtSignal, QObject
from PyQt6.QtGui import QFont, QPalette, QColor

import paho.mqtt.client as mqtt
from matplotlib.backends.backend_qtagg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.figure import Figure
import matplotlib.pyplot as plt

# ===== CONFIGURAÇÕES MQTT =====
MQTT_BROKER = "jaragua.lmq.cloudamqp.com"
MQTT_PORT = 8883 # Porta TLS para conexão direta
MQTT_USER = "refqurtq:refqurtq"
MQTT_PASS = "JTeV6YUekaRgi6G0lbp7XNA_i3c71jqO"
MQTT_TOPIC_TEMP = "climatizador/temperatura"
MQTT_TOPIC_UMID = "climatizador/umidade"
MQTT_TOPIC_CMD = "climatizador/comando"

# ===== PERFIS DE MATERIAIS =====
@dataclass
class Profile:
    name: str
    tmin: float
    tmax: float
    optimum: float
    umin: float
    umax: float

PROFILES = {
    "vacina": Profile("Vacinas", 2, 8, 5, 20, 60),
```

```

"insulina": Profile("Insulina", 2, 8, 5, 20, 60),
"reagente": Profile("Reagentes", 15, 25, 20, 20, 60),
"solucao": Profile("Soluções", 15, 25, 20, 20, 60),
"antibiotico": Profile("Antibióticos", 8, 15, 11.5, 20, 60),
}

```

```

# ===== ESTILOS =====

```

```

STYLE_LIGHT = ""

```

```

QMainWindow, QWidget {
    background-color: #f5f7fb;
    color: #475569;
    font-family: 'Segoe UI', Arial, sans-serif;
}

```

```

QLabel {
    color: #475569;
}

```

```

QLabel#title {
    color: #0a84ff;
    font-size: 20px;
    font-weight: bold;
}

```

```

QLabel#reading {
    color: #0a84ff;
    font-size: 42px;
    font-weight: bold;
}

```

```

QLabel#status-ok {
    background-color: rgba(24, 195, 123, 0.15);
    color: #18c37b;
    padding: 10px 20px;
    border-radius: 12px;
    font-weight: bold;
    font-size: 13px;
}

```

```

QLabel#status-warn {
    background-color: rgba(242, 201, 76, 0.15);
    color: #c9a227;
    padding: 10px 20px;
    border-radius: 12px;
    font-weight: bold;
    font-size: 13px;
}

```

```

QLabel#status-critical {
    background-color: rgba(255, 90, 90, 0.15);
}

```

```

        color: #ff6b6b;
        padding: 10px 20px;
        border-radius: 12px;
        font-weight: bold;
        font-size: 13px;
    }
    QGroupBox {
        background-color: rgba(255, 255, 255, 0.95);
        border: 1px solid rgba(0, 0, 0, 0.08);
        border-radius: 14px;
        margin-top: 16px;
        padding: 18px;
        font-weight: bold;
    }
    QGroupBox::title {
        color: #0a84ff;
        font-weight: bold;
        font-size: 13px;
        subcontrol-origin: margin;
        left: 14px;
        padding: 0 10px;
        background-color: #f5f7fb;
    }
    QComboBox {
        background-color: white;
        border: 1px solid rgba(0, 0, 0, 0.12);
        border-radius: 8px;
        padding: 10px 14px;
        color: #475569;
        font-size: 12px;
    }
    QComboBox:hover {
        border: 1px solid #0a84ff;
    }
    QComboBox::drop-down {
        border: none;
        padding-right: 10px;
    }
    QPushButton {
        background-color: white;
        border: 1px solid rgba(0, 0, 0, 0.1);
        border-radius: 8px;
        padding: 10px 16px;
        color: #475569;
    }

```

```

QPushButton:hover {
    background-color: #e8f4ff;
    border: 1px solid #0a84ff;
}
QPushButton#primary {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:0, stop:0 #0a84ff, stop:1 #00eaff);
    color: white;
    border: none;
    font-weight: bold;
}
QPushButton#primary:hover {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:0, stop:0 #0070dd, stop:1 #00c8dd);
}
QFrame#alert {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:0, stop:0 rgba(255,100,100,0.12), stop:1
rgba(255,80,80,0.06));
    border-left: 4px solid #ff6b6b;
    border-radius: 12px;
}
QProgressBar {
    border: none;
    border-radius: 7px;
    background-color: #e6eefb;
    height: 14px;
    text-align: center;
}
QProgressBar::chunk {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:0, stop:0 #0a84ff, stop:1 #00eaff);
    border-radius: 7px;
}
"""

STYLE_DARK = """
QMainWindow {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:1,
        stop:0 #0a0f1a, stop:0.3 #0d1526, stop:0.7 #111d2e, stop:1 #0a1628);
}
QWidget {
    background-color: transparent;
    color: #e1e8f0;
    font-family: 'Segoe UI', 'SF Pro Display', Arial, sans-serif;
}
QLabel {
    color: #b8c5d6;
}

```

```

QLabel#title {
    color: #00d4ff;
    font-size: 22px;
    font-weight: 700;
    letter-spacing: 0.5px;
}
QLabel#reading {
    color: #ffffff;
    font-size: 48px;
    font-weight: 300;
    letter-spacing: -1px;
}
QLabel#reading-temp {
    color: #00d4ff;
    font-size: 48px;
    font-weight: 300;
    letter-spacing: -1px;
}
QLabel#reading-hum {
    color: #00ff9d;
    font-size: 48px;
    font-weight: 300;
    letter-spacing: -1px;
}
QLabel#unit {
    color: #5a7a9a;
    font-size: 18px;
    font-weight: 400;
}
QLabel#status-ok {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
        stop:0 rgba(0, 255, 157, 0.15), stop:1 rgba(0, 212, 255, 0.08));
    color: #00ff9d;
    padding: 12px 24px;
    border-radius: 16px;
    font-weight: 600;
    font-size: 13px;
    border: 1px solid rgba(0, 255, 157, 0.25);
}
QLabel#status-warn {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
        stop:0 rgba(255, 193, 7, 0.18), stop:1 rgba(255, 152, 0, 0.10));
    color: #ffc107;
    padding: 12px 24px;
    border-radius: 16px;
}

```

```

    font-weight: 600;
    font-size: 13px;
    border: 1px solid rgba(255, 193, 7, 0.3);
}
QLabel#status-critical {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
        stop:0 rgba(255, 82, 82, 0.2), stop:1 rgba(255, 23, 68, 0.12));
    color: #ff5252;
    padding: 12px 24px;
    border-radius: 16px;
    font-weight: 600;
    font-size: 13px;
    border: 1px solid rgba(255, 82, 82, 0.35);
}
QGroupBox {
    background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
        stop:0 rgba(20, 35, 55, 0.95), stop:1 rgba(15, 28, 45, 0.85));
    border: 1px solid rgba(0, 212, 255, 0.12);
    border-radius: 20px;
    margin-top: 20px;
    padding: 20px;
    font-weight: 600;
}
QGroupBox::title {
    color: #00d4ff;
    font-weight: 700;
    font-size: 14px;
    subcontrol-origin: margin;
    left: 18px;
    padding: 4px 14px;
    background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
        stop:0 rgba(0, 212, 255, 0.2), stop:1 rgba(0, 255, 157, 0.1));
    border-radius: 10px;
}
QGroupBox#card-temp {
    border: 1px solid rgba(0, 212, 255, 0.2);
}
QGroupBox#card-temp::title {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
        stop:0 rgba(0, 212, 255, 0.25), stop:1 rgba(0, 150, 255, 0.15));
}
QGroupBox#card-hum {
    border: 1px solid rgba(0, 255, 157, 0.2);
}
QGroupBox#card-hum::title {

```

```

    color: #00ff9d;
    background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
        stop:0 rgba(0, 255, 157, 0.25), stop:1 rgba(0, 200, 120, 0.15));
}
QComboBox {
    background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
        stop:0 rgba(30, 50, 75, 0.95), stop:1 rgba(20, 38, 60, 0.9));
    border: 1px solid rgba(0, 212, 255, 0.2);
    border-radius: 12px;
    padding: 12px 18px;
    color: #e1e8f0;
    font-size: 13px;
    font-weight: 500;
}
QComboBox:hover {
    border: 1px solid rgba(0, 212, 255, 0.5);
    background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
        stop:0 rgba(35, 58, 85, 0.95), stop:1 rgba(25, 45, 70, 0.9));
}
QComboBox::drop-down {
    border: none;
    padding-right: 12px;
    width: 20px;
}
QComboBox QAbstractItemView {
    background-color: rgba(15, 30, 50, 0.98);
    border: 1px solid rgba(0, 212, 255, 0.25);
    border-radius: 12px;
    color: #e1e8f0;
    selection-background-color: rgba(0, 212, 255, 0.25);
    padding: 8px;
}
QComboBox QAbstractItemView::item {
    padding: 10px 14px;
    border-radius: 8px;
}
QComboBox QAbstractItemView::item:hover {
    background-color: rgba(0, 212, 255, 0.15);
}
QPushButton {
    background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
        stop:0 rgba(30, 50, 75, 0.9), stop:1 rgba(20, 38, 60, 0.85));
    border: 1px solid rgba(0, 212, 255, 0.15);
    border-radius: 12px;
    padding: 12px 20px;
}

```

```

        color: #b8c5d6;
        font-weight: 500;
    }
    QPushButton:hover {
        background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
            stop:0 rgba(0, 212, 255, 0.25), stop:1 rgba(0, 150, 200, 0.15));
        border: 1px solid rgba(0, 212, 255, 0.4);
        color: #00d4ff;
    }
    QPushButton#primary {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
            stop:0 #00d4ff, stop:0.5 #00a8cc, stop:1 #00ff9d);
        color: #0a1628;
        border: none;
        font-weight: 700;
        letter-spacing: 0.5px;
    }
    QPushButton#primary:hover {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
            stop:0 #33ddff, stop:0.5 #00c4e8, stop:1 #33ffb5);
    }
    QFrame#alert {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
            stop:0 rgba(255, 82, 82, 0.18), stop:1 rgba(255, 23, 68, 0.08));
        border-left: 4px solid #ff5252;
        border-radius: 16px;
        border: 1px solid rgba(255, 82, 82, 0.2);
        border-left: 4px solid #ff5252;
    }
    QProgressBar {
        border: none;
        border-radius: 8px;
        background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
            stop:0 rgba(0, 212, 255, 0.1), stop:1 rgba(0, 255, 157, 0.05));
        height: 16px;
        text-align: center;
    }
    QProgressBar::chunk {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
            stop:0 #00d4ff, stop:0.5 #00a8cc, stop:1 #00ff9d);
        border-radius: 8px;
    }
    QScrollBar:vertical {
        background: rgba(0, 212, 255, 0.05);
        width: 10px;
    }

```

```

border-radius: 5px;
margin: 2px;
}
QScrollBar::handle:vertical {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
        stop:0 rgba(0, 212, 255, 0.4), stop:1 rgba(0, 255, 157, 0.3));
    border-radius: 5px;
    min-height: 40px;
}
QScrollBar::handle:vertical:hover {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
        stop:0 rgba(0, 212, 255, 0.6), stop:1 rgba(0, 255, 157, 0.5));
}
QScrollBar::add-line:vertical, QScrollBar::sub-line:vertical {
    height: 0px;
}
"""

```

===== SINAIS MQTT =====

```

class MQTTSignals(QObject):
    """Sinais para comunicação thread-safe com a GUI"""
    connected = pyqtSignal()
    disconnected = pyqtSignal()
    error = pyqtSignal(str)
    temperature_received = pyqtSignal(float)
    humidity_received = pyqtSignal(float)
    connection_status = pyqtSignal(bool) # True = conectado, False = desconectado
    data_received = pyqtSignal() # Sinal quando qualquer dado é recebido

```

===== CLIENTE MQTT =====

```

class MQTTClient:
    def __init__(self, signals: MQTTSignals):
        self.signals = signals
        self.is_connected = False
        self.client = mqtt.Client(
            client_id=f"desktop_{datetime.now().strftime('%H%M%S')}",
            protocol=mqtt.MQTTv311
        )

    # Configurar callbacks
    self.client.on_connect = self._on_connect
    self.client.on_disconnect = self._on_disconnect
    self.client.on_message = self._on_message

```

```

# Configurar autenticação
self.client.username_pw_set(MQTT_USER, MQTT_PASS)

# Configurar TLS/SSL
self.client.tls_set(cert_reqs=ssl.CERT_NONE)
self.client.tls_insecure_set(True)

def connect(self):
    """Conecta ao broker MQTT"""
    try:
        self.client.connect(MQTT_BROKER, MQTT_PORT, keepalive=60)
        self.client.loop_start()
    except Exception as e:
        self.is_connected = False
        self.signals.error.emit(f"Erro de conexão: {str(e)}")
        self.signals.connection_status.emit(False)

def disconnect(self):
    """Desconecta do broker MQTT"""
    self.is_connected = False
    self.client.loop_stop()
    self.client.disconnect()

def check_connection(self) -> bool:
    """Verifica se está conectado ao broker"""
    return self.is_connected and self.client.is_connected()

def publish_command(self, command: str):
    """Publica comando no tópico de comandos"""
    if self.is_connected:
        self.client.publish(MQTT_TOPIC_CMD, command)

def _on_connect(self, client, userdata, flags, rc):
    if rc == 0:
        self.is_connected = True
        self.signals.connected.emit()
        self.signals.connection_status.emit(True)
        # Inscrever nos tópicos
        client.subscribe(MQTT_TOPIC_TEMP)
        client.subscribe(MQTT_TOPIC_UMID)
        print(f"[MQTT] Conectado ao broker {MQTT_BROKER}")
    else:
        self.is_connected = False
        error_messages = {

```

```

        1: "Protocolo incorreto",
        2: "ID de cliente inválido",
        3: "Servidor indisponível",
        4: "Usuário/senha incorretos",
        5: "Não autorizado"
    }
    error_msg = error_messages.get(rc, f"Código desconhecido: {rc}")
    self.signals.error.emit(f"Falha na conexão: {error_msg}")
    self.signals.connection_status.emit(False)
    print(f"[MQTT] Erro de conexão: {error_msg}")

def _on_disconnect(self, client, userdata, rc):
    self.is_connected = False
    self.signals.disconnected.emit()
    self.signals.connection_status.emit(False)
    if rc != 0:
        print(f"[MQTT] Desconectado inesperadamente (código: {rc})")
    else:
        print("[MQTT] Desconectado")

def _on_message(self, client, userdata, msg):
    try:
        value = float(msg.payload.decode())
        if msg.topic == MQTT_TOPIC_TEMP:
            self.signals.temperature_received.emit(value)
            self.signals.data_received.emit()
        elif msg.topic == MQTT_TOPIC_UMID:
            self.signals.humidity_received.emit(value)
            self.signals.data_received.emit()
    except ValueError:
        pass

```

===== GRÁFICO MATPLOTLIB =====

```

class ChartCanvas(FigureCanvas):
    # Modos de visualização
    MODE_BOTH_SPLIT = 0    # Dois gráficos separados
    MODE_TEMP_ONLY = 1     # Apenas temperatura
    MODE_HUM_ONLY = 2      # Apenas umidade
    MODE_BOTH_COMBINED = 3 # Ambos no mesmo gráfico

    def __init__(self, parent=None, dark_mode=False):
        self.dark_mode = dark_mode
        self.view_mode = self.MODE_BOTH_SPLIT # Modo padrão

```

```

# Criar figura
self.fig = Figure(figsize=(10, 5), dpi=100)
self.fig.patch.set_alpha(0)

super().__init__(self.fig)
self.setParent(parent)

# Configurar política de tamanho para expansão
self.setSizePolicy(QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Expanding)
self.setMinimumHeight(280)

# Inicializar eixos (serão criados dinamicamente)
self.ax_temp = None
self.ax_hum = None
self.ax_combined = None
self.ax_combined_hum = None # Eixo Y secundário para modo combinado

self._setup_axes()

# Dados
self.max_points = 60
self.times = deque(maxlen=self.max_points)
self.temp_data = deque(maxlen=self.max_points)
self.hum_data = deque(maxlen=self.max_points)

def set_view_mode(self, mode: int):
    """Altera o modo de visualização do gráfico"""
    if mode != self.view_mode:
        self.view_mode = mode
        self._setup_axes()
        self.update_chart()

def _setup_axes(self):
    """Configura os eixos baseado no modo de visualização"""
    self.fig.clear()

    if self.view_mode == self.MODE_BOTH_SPLIT:
        # Dois subplots separados
        self.ax_temp = self.fig.add_subplot(211)
        self.ax_hum = self.fig.add_subplot(212)
        self.ax_combined = None
        self.ax_combined_hum = None
        self.fig.subplots_adjust(hspace=0.35, left=0.1, right=0.95, top=0.95, bottom=0.12)

    elif self.view_mode == self.MODE_TEMP_ONLY:

```

```

# Apenas temperatura
self.ax_temp = self.fig.add_subplot(111)
self.ax_hum = None
self.ax_combined = None
self.ax_combined_hum = None
self.fig.subplots_adjust(left=0.1, right=0.95, top=0.95, bottom=0.12)

elif self.view_mode == self.MODE_HUM_ONLY:
    # Apenas umidade
    self.ax_temp = None
    self.ax_hum = self.fig.add_subplot(111)
    self.ax_combined = None
    self.ax_combined_hum = None
    self.fig.subplots_adjust(left=0.1, right=0.95, top=0.95, bottom=0.12)

elif self.view_mode == self.MODE_BOTH_COMBINED:
    # Ambos no mesmo gráfico com eixo Y único
    self.ax_temp = None
    self.ax_hum = None
    self.ax_combined = self.fig.add_subplot(111)
    self.ax_combined_hum = None # Não usar eixo secundário
    self.fig.subplots_adjust(left=0.1, right=0.95, top=0.95, bottom=0.12)

self._style_axes()

def _style_axes(self):
    """Aplica estilos aos eixos ativos"""
    text_color = '#b8c5d6' if self.dark_mode else '#475569'
    temp_color = '#00d4ff' if self.dark_mode else '#0a84ff'
    hum_color = '#00ff9d' if self.dark_mode else '#18c37b'

    self.fig.patch.set_facecolor('none')

    # Estilizar eixo de temperatura (se existir)
    if self.ax_temp is not None:
        self.ax_temp.set_facecolor('none')
        self.ax_temp.tick_params(axis='y', colors=temp_color, labels=10)
        self.ax_temp.tick_params(axis='x', colors=text_color, labels=8)
        self.ax_temp.spines['bottom'].set_color(text_color)
        self.ax_temp.spines['bottom'].set_alpha(0.3)
        self.ax_temp.spines['left'].set_color(temp_color)
        self.ax_temp.spines['left'].set_alpha(0.6)
        self.ax_temp.spines['top'].set_visible(False)
        self.ax_temp.spines['right'].set_visible(False)

```

```

        self.ax_temp.set_ylabel('Temperatura (°C)', color=temp_color, fontsize=11,
fontweight='600')
        self.ax_temp.grid(True, alpha=0.15, color=text_color, linestyle='--', linewidth=0.5)

# Estilizar eixo de umidade (se existir)
if self.ax_hum is not None:
    self.ax_hum.set_facecolor('none')
    self.ax_hum.tick_params(axis='y', colors=hum_color, labels=10)
    self.ax_hum.tick_params(axis='x', colors=text_color, labels=8, rotation=30)
    self.ax_hum.spines['bottom'].set_color(text_color)
    self.ax_hum.spines['bottom'].set_alpha(0.3)
    self.ax_hum.spines['left'].set_color(hum_color)
    self.ax_hum.spines['left'].set_alpha(0.6)
    self.ax_hum.spines['top'].set_visible(False)
    self.ax_hum.spines['right'].set_visible(False)
    self.ax_hum.set_ylabel('Umidade (%)', color=hum_color, fontsize=11,
fontweight='600')
    self.ax_hum.grid(True, alpha=0.15, color=text_color, linestyle='--', linewidth=0.5)

# Estilizar eixo combinado (se existir)
if self.ax_combined is not None:
    self.ax_combined.set_facecolor('none')
    self.ax_combined.tick_params(axis='y', colors=text_color, labels=10, pad=5)
    self.ax_combined.tick_params(axis='x', colors=text_color, labels=8, rotation=30)
    self.ax_combined.spines['bottom'].set_color(text_color)
    self.ax_combined.spines['bottom'].set_alpha(0.3)
    self.ax_combined.spines['left'].set_color(text_color)
    self.ax_combined.spines['left'].set_alpha(0.4)
    self.ax_combined.spines['top'].set_visible(False)
    self.ax_combined.spines['right'].set_visible(False)
    self.ax_combined.set_ylabel('°C | %', color=text_color, fontsize=12,
fontweight='600', labelpad=10)
    self.ax_combined.grid(True, alpha=0.15, color=text_color, linestyle='--', linewidth=0.5)

def update_theme(self, dark_mode: bool):
    self.dark_mode = dark_mode
    self._setup_axes()
    self.update_chart()

def add_point(self, temp: float, hum: float):
    now = datetime.now().strftime('%H:%M:%S')
    self.times.append(now)
    self.temp_data.append(temp)
    self.hum_data.append(hum)
    self.update_chart()

```

```

def resizeEvent(self, event):
    super().resizeEvent(event)
    if len(self.times) > 0:
        self.update_chart()

def update_chart(self):
    """Atualiza o gráfico baseado no modo atual"""
    # Limpar eixos
    if self.ax_temp: self.ax_temp.clear()
    if self.ax_hum: self.ax_hum.clear()
    if self.ax_combined: self.ax_combined.clear()
    if self.ax_combined_hum: self.ax_combined_hum.clear()

    self._style_axes()

    if len(self.times) == 0:
        self.draw()
        return

    times_list = list(self.times)
    temp_list = list(self.temp_data)
    hum_list = list(self.hum_data)
    x_indices = list(range(len(times_list)))

    temp_color = '#00d4ff' if self.dark_mode else '#0a84ff'
    hum_color = '#00ff9d' if self.dark_mode else '#18c37b'
    text_color = '#b8c5d6' if self.dark_mode else '#475569'

    # Configurar labels do eixo X
    num_labels = min(8, len(times_list))
    if len(times_list) > num_labels:
        step = max(1, len(times_list) // num_labels)
        tick_positions = x_indices[::step]
        tick_labels = [times_list[i] for i in tick_positions]
    else:
        tick_positions = x_indices
        tick_labels = times_list

    show_markers = len(temp_list) < 20

    if self.view_mode == self.MODE_BOTH_SPLIT:
        self._plot_split(x_indices, temp_list, hum_list, temp_color, hum_color,
                        tick_positions, tick_labels, show_markers)
    elif self.view_mode == self.MODE_TEMP_ONLY:

```

```

        self._plot_temp_only(x_indices, temp_list, temp_color,
                             tick_positions, tick_labels, show_markers)
    elif self.view_mode == self.MODE_HUM_ONLY:
        self._plot_hum_only(x_indices, hum_list, hum_color,
                             tick_positions, tick_labels, show_markers)
    elif self.view_mode == self.MODE_BOTH_COMBINED:
        self._plot_combined(x_indices, temp_list, hum_list, temp_color, hum_color,
                             text_color, tick_positions, tick_labels, show_markers)

    self.draw()

def _plot_split(self, x, temp, hum, tc, hc, ticks, labels, markers):
    """Plotar em dois gráficos separados"""
    # Temperatura
    self.ax_temp.plot(x, temp, color=tc, linewidth=2.5,
                      marker='o' if markers else None, markersize=5,
                      markerfacecolor='#0a1628', markeredgecolor=tc, markeredgewidth=2)
    self.ax_temp.fill_between(x, temp, alpha=0.15, color=tc)
    if temp:
        self.ax_temp.axhline(y=temp[-1], color=tc, linestyle=':', alpha=0.4, linewidth=1)
        self.ax_temp.annotate(f'{temp[-1]:.1f}°C', xy=(len(temp)-1, temp[-1]),
                              xytext=(5, 0), textcoords='offset points',
                              color=tc, fontsize=10, fontweight='bold', va='center')
        t_min, t_max = min(temp), max(temp)
        t_margin = max(2, (t_max - t_min) * 0.2)
        self.ax_temp.set_ylim(t_min - t_margin, t_max + t_margin)

    # Umidade
    self.ax_hum.plot(x, hum, color=hc, linewidth=2.5,
                     marker='s' if markers else None, markersize=4,
                     markerfacecolor='#0a1628', markeredgecolor=hc, markeredgewidth=2)
    self.ax_hum.fill_between(x, hum, alpha=0.12, color=hc)
    if hum:
        self.ax_hum.axhline(y=hum[-1], color=hc, linestyle=':', alpha=0.4, linewidth=1)
        self.ax_hum.annotate(f'{hum[-1]:.0f}%', xy=(len(hum)-1, hum[-1]),
                              xytext=(5, 0), textcoords='offset points',
                              color=hc, fontsize=10, fontweight='bold', va='center')
        h_min, h_max = min(hum), max(hum)
        h_margin = max(5, (h_max - h_min) * 0.2)
        self.ax_hum.set_ylim(max(0, h_min - h_margin), min(100, h_max + h_margin))

    self.ax_temp.set_xticks(ticks)
    self.ax_temp.set_xticklabels([" for _ in ticks])
    self.ax_hum.set_xticks(ticks)
    self.ax_hum.set_xticklabels(labels, rotation=30, ha='right', fontsize=8)

```

```

def _plot_temp_only(self, x, temp, tc, ticks, labels, markers):
    """Plotar apenas temperatura"""
    self.ax_temp.plot(x, temp, color=tc, linewidth=2.5,
                      marker='o' if markers else None, markersize=6,
                      markerfacecolor='#0a1628', markeredgecolor=tc, markeredgewidth=2)
    self.ax_temp.fill_between(x, temp, alpha=0.18, color=tc)
    if temp:
        self.ax_temp.axhline(y=temp[-1], color=tc, linestyle=':', alpha=0.4, linewidth=1)
        self.ax_temp.annotate(f'{temp[-1]:.1f}°C', xy=(len(temp)-1, temp[-1]),
                              xytext=(8, 0), textcoords='offset points',
                              color=tc, fontsize=12, fontweight='bold', va='center')
        t_min, t_max = min(temp), max(temp)
        t_margin = max(2, (t_max - t_min) * 0.2)
        self.ax_temp.set_ylim(t_min - t_margin, t_max + t_margin)

    self.ax_temp.set_xticks(ticks)
    self.ax_temp.set_xticklabels(labels, rotation=30, ha='right', fontsize=9)

def _plot_hum_only(self, x, hum, hc, ticks, labels, markers):
    """Plotar apenas umidade"""
    self.ax_hum.plot(x, hum, color=hc, linewidth=2.5,
                     marker='s' if markers else None, markersize=6,
                     markerfacecolor='#0a1628', markeredgecolor=hc, markeredgewidth=2)
    self.ax_hum.fill_between(x, hum, alpha=0.15, color=hc)
    if hum:
        self.ax_hum.axhline(y=hum[-1], color=hc, linestyle=':', alpha=0.4, linewidth=1)
        self.ax_hum.annotate(f'{hum[-1]:.0f}%', xy=(len(hum)-1, hum[-1]),
                              xytext=(8, 0), textcoords='offset points',
                              color=hc, fontsize=12, fontweight='bold', va='center')
        h_min, h_max = min(hum), max(hum)
        h_margin = max(5, (h_max - h_min) * 0.2)
        self.ax_hum.set_ylim(max(0, h_min - h_margin), min(100, h_max + h_margin))

    self.ax_hum.set_xticks(ticks)
    self.ax_hum.set_xticklabels(labels, rotation=30, ha='right', fontsize=9)

def _plot_combined(self, x, temp, hum, tc, hc, text_color, ticks, labels, markers):
    """Plotar ambos no mesmo gráfico com eixo Y único"""
    # Temperatura (linha sólida ciano)
    line1, = self.ax_combined.plot(x, temp, color=tc, linewidth=2.5,
                                   marker='o' if markers else None, markersize=5,
                                   markerfacecolor='#0a1628', markeredgecolor=tc,
                                   markeredgewidth=2, label='Temperatura (°C)')
    self.ax_combined.fill_between(x, temp, alpha=0.12, color=tc)

```

```

# Umidade (linha tracejada verde)
line2, = self.ax_combined.plot(x, hum, color=hc, linewidth=2.5,
                                marker='s' if markers else None, markersize=4,
                                markerfacecolor='#0a1628', markeredgecolor=hc,
                                markeredgewidth=2, linestyle='--', label='Umidade (%)')
self.ax_combined.fill_between(x, hum, alpha=0.08, color=hc)

# Calcular limites considerando ambos os valores
all_values = temp + hum
if all_values:
    v_min, v_max = min(all_values), max(all_values)
    v_margin = max(5, (v_max - v_min) * 0.15)
    self.ax_combined.set_ylim(max(0, v_min - v_margin), v_max + v_margin)

# Anotações dos valores atuais
if temp:
    self.ax_combined.annotate(f'{temp[-1]:.1f}°C', xy=(len(temp)-1, temp[-1]),
                              xytext=(8, 5), textcoords='offset points',
                              color=tc, fontsize=10, fontweight='bold', va='bottom')
if hum:
    self.ax_combined.annotate(f'{hum[-1]:.0f}%', xy=(len(hum)-1, hum[-1]),
                              xytext=(8, -5), textcoords='offset points',
                              color=hc, fontsize=10, fontweight='bold', va='top')

# Legenda
self.ax_combined.legend(loc='upper left', facecolor='none',
                        edgecolor='none', labelcolor=text_color, fontsize=9)

self.ax_combined.set_xticks(ticks)
self.ax_combined.set_xticklabels(labels, rotation=30, ha='right', fontsize=9)

# ===== JANELA PRINCIPAL =====
class DashboardWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.dark_mode = True
        self.current_temp = None
        self.current_hum = None

        self.setWindowTitle("Dashboard Profissional — Monitoramento Medicinal")
        self.setMinimumSize(1000, 700)
        self.resize(1150, 800)

```

```

# Configurar MQTT
self.mqtt_signals = MQTTSignals()
self.mqtt_client = MQTTClient(self.mqtt_signals)

# Conectar sinais
self.mqtt_signals.connected.connect(self._on_mqtt_connected)
self.mqtt_signals.disconnected.connect(self._on_mqtt_disconnected)
self.mqtt_signals.error.connect(self._on_mqtt_error)
self.mqtt_signals.temperature_received.connect(self._on_temperature)
self.mqtt_signals.humidity_received.connect(self._on_humidity)
self.mqtt_signals.data_received.connect(self._on_data_received)

# Controle de timeout de dados (ESP32 envia a cada 2s)
self.last_data_time = None
self.data_timeout = 10 # segundos sem dados = ESP32 offline

self._setup_ui()
self._apply_style()

# Conectar ao MQTT
self.mqtt_client.connect()

# Timer para reconexão
self.reconnect_timer = QTimer()
self.reconnect_timer.timeout.connect(self._try_reconnect)

# Timer para verificar conexão periodicamente (a cada 3 segundos)
self.connection_check_timer = QTimer()
self.connection_check_timer.timeout.connect(self._check_mqtt_connection)
self.connection_check_timer.start(3000)

# Contador de tentativas de reconexão
self.reconnect_attempts = 0
self.max_reconnect_attempts = 10

def _setup_ui(self):
    central = QWidget()
    self.setCentralWidget(central)

    main_layout = QVBoxLayout(central)
    main_layout.setContentsMargins(24, 20, 24, 20)
    main_layout.setSpacing(16)

    # === HEADER ===

```

```

header = QHBoxLayout()
header.setSpacing(12)

# Logo e título
brand = QHBoxLayout()
brand.setSpacing(16)
logo = QLabel("$")
logo.setFixedSize(56, 56)
logo.setAlignment(Qt.AlignmentFlag.AlignCenter)
logo.setStyleSheet("""
    background: qlineargradient(x1:0, y1:0, x2:1, y2:1,
        stop:0 #00d4ff, stop:0.5 #00a8cc, stop:1 #00ff9d);
    color: #0a1628;
    font-weight: bold;
    font-size: 28px;
    border-radius: 16px;
    border: 2px solid rgba(0, 255, 157, 0.3);
""")
brand.addWidget(logo)

title_layout = QVBoxLayout()
title_layout.setSpacing(4)
title = QLabel("Monitoramento Medicinal")
title.setObjectName("title")
subtitle = QLabel("⚡ ESP32 • 🌡️ DHT22 • 📡 MQTT • 💻 Desktop")
subtitle.setStyleSheet("font-size: 12px; color: #5a7a9a; letter-spacing: 0.5px;")
title_layout.addWidget(title)
title_layout.addWidget(subtitle)
brand.addLayout(title_layout)
brand.addStretch()

header.addLayout(brand, 1)

# Status e controles
self.status_label = QLabel("🔄 Conectando...")
self.status_label.setMinimumWidth(320)
self.status_label.setStyleSheet("""
    background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
        stop:0 rgba(0, 212, 255, 0.15), stop:1 rgba(0, 255, 157, 0.08));
    padding: 14px 20px;
    border-radius: 14px;
    color: #00d4ff;
    font-weight: 600;
    font-size: 13px;
    border: 1px solid rgba(0, 212, 255, 0.2);
""")

```

```

        letter-spacing: 0.3px;
        """)
header.addWidget(self.status_label)

main_layout.addLayout(header)

# === ALERTA ===
self.alert_frame = QFrame()
self.alert_frame.setObjectName("alert")
self.alert_frame.setVisible(False)
self.alert_frame.setMinimumHeight(50)
alert_layout = QHBoxLayout(self.alert_frame)
alert_layout.setContentsMargins(16, 12, 16, 12)
alert_icon = QLabel("⚠")
alert_icon.setStyleSheet("font-size: 22px;")
self.alert_text = QLabel("Nenhum alerta")
self.alert_text.setStyleSheet("color: #ff7b7b; font-weight: bold; font-size: 13px;")
self.alert_text.setWordWrap(True)
alert_layout.addWidget(alert_icon)
alert_layout.addWidget(self.alert_text, 1)
main_layout.addWidget(self.alert_frame)

# === GRID PRINCIPAL ===
grid = QGridLayout()
grid.setSpacing(16)
# Configurar stretch para colunas iguais
grid.setColumnStretch(0, 1)
grid.setColumnStretch(1, 1)
grid.setColumnStretch(2, 1)
# Configurar stretch para linhas (gráfico maior)
grid.setRowStretch(0, 0)
grid.setRowStretch(1, 1)

# Card: Perfil do Material
profile_card = QGroupBox("📋 Perfil do Material")
profile_card.setSizePolicy(QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Preferred)
profile_layout = QVBoxLayout(profile_card)
profile_layout.setSpacing(12)
self.profile_combo = QComboBox()
self.profile_combo.setMinimumHeight(36)
self.profile_combo.addItem("💉 Vacinas (2–8°C)", "vacina")
self.profile_combo.addItem("💊 Insulina (2–8°C)", "insulina")
self.profile_combo.addItem("🧪 Reagentes (15–25°C)", "reagente")
self.profile_combo.addItem("💧 Soluções (15–25°C)", "solucao")
self.profile_combo.addItem("💊 Antibióticos (8–15°C)", "antibiotico")

```

```

self.profile_combo.currentIndexChanged.connect(self._on_profile_changed)
profile_layout.addWidget(self.profile_combo)

profile_desc = QLabel("Escolha o perfil que melhor descreve o conteúdo armazenado.")
profile_desc.setWordWrap(True)
profile_desc.setStyleSheet("font-size: 11px; color: #8899a6;")
profile_layout.addWidget(profile_desc)
profile_layout.addStretch()
grid.addWidget(profile_card, 0, 0, 1, 1)

# Card: Temperatura
temp_card = QGroupBox("🌡 Temperatura")
temp_card.setObjectName("card-temp")
temp_card.setSizePolicy(QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Preferred)
temp_layout = QVBoxLayout(temp_card)
temp_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
temp_layout.setSpacing(8)

# Valor da temperatura com ícone animado
temp_value_container = QHBoxLayout()
temp_value_container.setAlignment(Qt.AlignmentFlag.AlignCenter)
temp_value_container.setSpacing(4)

self.temp_reading = QLabel("--")
self.temp_reading.setObjectName("reading-temp")
self.temp_reading.setAlignment(Qt.AlignmentFlag.AlignCenter)

temp_unit = QLabel("°C")
temp_unit.setObjectName("unit")
temp_unit.setStyleSheet("color: #00d4ff; font-size: 20px; font-weight: 300;")

temp_value_container.addWidget(self.temp_reading)
temp_value_container.addWidget(temp_unit)

self.temp_time = QLabel("🕒 Aguardando...")
self.temp_time.setAlignment(Qt.AlignmentFlag.AlignCenter)
self.temp_time.setStyleSheet("font-size: 11px; color: #5a7a9a;")

temp_layout.addStretch()
temp_layout.addLayout(temp_value_container)
temp_layout.addWidget(self.temp_time)
temp_layout.addStretch()
grid.addWidget(temp_card, 0, 1, 1, 1)

# Card: Umidade

```

```

hum_card = QGroupBox("💧 Umidade")
hum_card.setObjectName("card-hum")
hum_card.setSizePolicy(QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Preferred)
hum_layout = QVBoxLayout(hum_card)
hum_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
hum_layout.setSpacing(8)

# Valor da umidade com ícone
hum_value_container = QHBoxLayout()
hum_value_container.setAlignment(Qt.AlignmentFlag.AlignCenter)
hum_value_container.setSpacing(4)

self.hum_reading = QLabel("--")
self.hum_reading.setObjectName("reading-hum")
self.hum_reading.setAlignment(Qt.AlignmentFlag.AlignCenter)

hum_unit = QLabel("%")
hum_unit.setObjectName("unit")
hum_unit.setStyleSheet("color: #00ff9d; font-size: 20px; font-weight: 300;")

hum_value_container.addWidget(self.hum_reading)
hum_value_container.addWidget(hum_unit)

self.hum_time = QLabel("🕒 Aguardando...")
self.hum_time.setAlignment(Qt.AlignmentFlag.AlignCenter)
self.hum_time.setStyleSheet("font-size: 11px; color: #5a7a9a;")

hum_layout.addStretch()
hum_layout.addLayout(hum_value_container)
hum_layout.addWidget(self.hum_time)
hum_layout.addStretch()
grid.addWidget(hum_card, 0, 2, 1, 1)

# Card: Gráfico (grande) - ocupa toda a largura
chart_card = QGroupBox("📈 Histórico de Leituras")
chart_card.setSizePolicy(QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Expanding)
chart_layout = QVBoxLayout(chart_card)
chart_layout.setSpacing(10)

# === Seletor de modo de visualização ===
view_mode_container = QHBoxLayout()
view_mode_container.setSpacing(8)

view_label = QLabel("Visualização:")
view_label.setStyleSheet("font-weight: 600; font-size: 12px; color: #5a7a9a;")

```

```

view_mode_container.addWidget(view_label)

# Botões de modo
self.btn_both_split = QPushButton("📊 Separados")
self.btn_temp_only = QPushButton("🌡️ Temperatura")
self.btn_hum_only = QPushButton("💧 Umidade")
self.btn_both_combined = QPushButton("📈 Combinado")

# Estilo base para botões de modo
mode_btn_style = """
QPushButton {
    background: rgba(20, 35, 55, 0.6);
    border: 1px solid rgba(0, 212, 255, 0.15);
    border-radius: 8px;
    padding: 6px 12px;
    color: #8899a6;
    font-size: 11px;
    font-weight: 500;
}
QPushButton:hover {
    background: rgba(0, 212, 255, 0.15);
    border: 1px solid rgba(0, 212, 255, 0.3);
    color: #00d4ff;
}
"""

mode_btn_active_style = """
QPushButton {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
        stop:0 rgba(0, 212, 255, 0.3), stop:1 rgba(0, 255, 157, 0.2));
    border: 1px solid rgba(0, 212, 255, 0.5);
    border-radius: 8px;
    padding: 6px 12px;
    color: #00d4ff;
    font-size: 11px;
    font-weight: 600;
}
"""

self.btn_both_split.setStyleSheet(mode_btn_active_style) # Ativo por padrão
self.btn_temp_only.setStyleSheet(mode_btn_style)
self.btn_hum_only.setStyleSheet(mode_btn_style)
self.btn_both_combined.setStyleSheet(mode_btn_style)

# Armazenar estilos para uso posterior
self._mode_btn_style = mode_btn_style

```

```

self._mode_btn_active_style = mode_btn_active_style
self._mode_buttons = [self.btn_both_split, self.btn_temp_only,
                      self.btn_hum_only, self.btn_both_combined]

# Conectar botões
self.btn_both_split.clicked.connect(lambda: self._set_chart_mode(0))
self.btn_temp_only.clicked.connect(lambda: self._set_chart_mode(1))
self.btn_hum_only.clicked.connect(lambda: self._set_chart_mode(2))
self.btn_both_combined.clicked.connect(lambda: self._set_chart_mode(3))


view_mode_container.addWidget(self.btn_both_split)
view_mode_container.addWidget(self.btn_temp_only)
view_mode_container.addWidget(self.btn_hum_only)
view_mode_container.addWidget(self.btn_both_combined)
view_mode_container.addStretch()

chart_layout.addLayout(view_mode_container)

# Gráfico
self.chart = ChartCanvas(dark_mode=self.dark_mode)
chart_layout.addWidget(self.chart, 1) # stretch factor 1 para expandir

# Barra de status inferior do gráfico
stats_frame = QFrame()
stats_frame.setStyleSheet("""
    QFrame {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
            stop:0 rgba(0, 212, 255, 0.08), stop:0.5 rgba(0, 168, 204, 0.05), stop:1 rgba(0,
255, 157, 0.08));
        border-radius: 14px;
        border: 1px solid rgba(0, 212, 255, 0.1);
    }
""")
stats_layout = QHBoxLayout(stats_frame)
stats_layout.setContentsMargins(20, 14, 20, 14)
stats_layout.setSpacing(30)

# Estabilidade
stab_container = QWidget()
stab_layout = QHBoxLayout(stab_container)
stab_layout.setContentsMargins(0, 0, 0, 0)
stab_layout.setSpacing(12)

stab_label = QLabel(" Estabilidade")
stab_label.setStyleSheet("font-weight: 600; font-size: 13px; color: #b8c5d6;")

```

```

stab_layout.addWidget(stab_label)

self.stability_bar = QProgressBar()
self.stability_bar.setRange(0, 100)
self.stability_bar.setValue(0)
self.stability_bar.setTextVisible(False)
self.stability_bar.setFixedHeight(16)
self.stability_bar.setMinimumWidth(150)
self.stability_bar.setMaximumWidth(250)
stab_layout.addWidget(self.stability_bar)

self.stability_label = QLabel("-- %")
self.stability_label.setStyleSheet("font-weight: 700; min-width: 50px; font-size: 14px;
color: #00d4ff;")
stab_layout.addWidget(self.stability_label)

stats_layout.addWidget(stab_container)
stats_layout.addStretch()

# Status do perfil
status_container = QWidget()
status_layout_h = QHBoxLayout(status_container)
status_layout_h.setContentsMargins(0, 0, 0, 0)
status_layout_h.setSpacing(14)

status_title = QLabel("🎯 Status")
status_title.setStyleSheet("font-weight: 600; font-size: 13px; color: #b8c5d6;")
status_layout_h.addWidget(status_title)

self.profile_status = QLabel("Aguardando...")
self.profile_status.setObjectName("status-ok")
self.profile_status.setAlignment(Qt.AlignmentFlag.AlignCenter)
self.profile_status.setMinimumWidth(180)
status_layout_h.addWidget(self.profile_status)

stats_layout.addWidget(status_container)

chart_layout.addWidget(stats_frame)
grid.addWidget(chart_card, 1, 0, 1, 3)

main_layout.addLayout(grid, 1) # stretch factor 1 para expandir

def _set_chart_mode(self, mode: int):
    """Altera o modo de visualização do gráfico"""
    # Atualizar estilos dos botões

```

```

for i, btn in enumerate(self._mode_buttons):
    if i == mode:
        btn.setStyleSheet(self._mode_btn_active_style)
    else:
        btn.setStyleSheet(self._mode_btn_style)

# Alterar modo do gráfico
self.chart.set_view_mode(mode)

def _apply_style(self):
    self.setStyleSheet(STYLE_DARK)
    self.chart.update_theme(True)

def _on_mqtt_connected(self):
    self.reconnect_attempts = 0
    self.status_label.setText("🟡 MQTT: Conectado | Aguardando ESP32...")
    self.status_label.setStyleSheet("""
        background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
            stop:0 rgba(255, 193, 7, 0.2), stop:1 rgba(255, 152, 0, 0.1));
        padding: 14px 20px;
        border-radius: 14px;
        color: #ffc107;
        font-weight: 600;
        font-size: 13px;
        border: 1px solid rgba(255, 193, 7, 0.3);
        letter-spacing: 0.3px;
    """)
    self.reconnect_timer.stop()
    print("[STATUS] Conectado ao broker MQTT. Aguardando dados do ESP32...")

def _on_mqtt_disconnected(self):
    self.status_label.setText("🔴 MQTT: Desconectado")
    self.status_label.setStyleSheet("""
        background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
            stop:0 rgba(255, 82, 82, 0.2), stop:1 rgba(255, 23, 68, 0.12));
        padding: 14px 20px;
        border-radius: 14px;
        color: #ff5252;
        font-weight: 600;
        font-size: 13px;
        border: 1px solid rgba(255, 82, 82, 0.35);
        letter-spacing: 0.3px;
    """)
    # Tentar reconectar
    if not self.reconnect_timer.isActive():

```

```

        self.reconnect_timer.start(5000)
        print("[STATUS] Conexão MQTT perdida!")

def _on_mqtt_error(self, error: str):
    self.status_label.setText(f"🔴 MQTT: Erro")
    self.status_label.setStyleSheet("""
        background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
            stop:0 rgba(255, 82, 82, 0.2), stop:1 rgba(255, 23, 68, 0.12));
        padding: 14px 20px;
        border-radius: 14px;
        color: #ff5252;
        font-weight: 600;
        font-size: 13px;
        border: 1px solid rgba(255, 82, 82, 0.35);
        letter-spacing: 0.3px;
    """)
    self.status_label.setToolTip(error)
    if not self.reconnect_timer.isActive():
        self.reconnect_timer.start(5000)
    print(f"[STATUS] Erro MQTT: {error}")

def _check_mqtt_connection(self):
    """Verifica periodicamente o status da conexão MQTT e recebimento de dados"""
    is_connected = self.mqtt_client.check_connection()

    if not is_connected and not self.reconnect_timer.isActive():
        self._on_mqtt_disconnected()
        return

    # Verificar se está recebendo dados do ESP32
    if is_connected and self.last_data_time is not None:
        elapsed = (datetime.now() - self.last_data_time).total_seconds()
        if elapsed > self.data_timeout:
            self.status_label.setText(f"🟡 MQTT: Conectado | ESP32 Offline ({int(elapsed)}s sem dados)")
            self.status_label.setStyleSheet("""
                background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
                    stop:0 rgba(255, 193, 7, 0.2), stop:1 rgba(255, 152, 0, 0.1));
                padding: 14px 20px;
                border-radius: 14px;
                color: #ffc107;
                font-weight: 600;
                font-size: 13px;
                border: 1px solid rgba(255, 193, 7, 0.3);
                letter-spacing: 0.3px;
            """)

```

```

        """
    elif is_connected and self.last_data_time is None:
        # Conectado mas nunca recebeu dados
        self.status_label.setText("🟡 MQTT: Conectado | Aguardando ESP32...")
        self.status_label.setStyleSheet("""
            background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
                stop:0 rgba(255, 193, 7, 0.2), stop:1 rgba(255, 152, 0, 0.1));
            padding: 14px 20px;
            border-radius: 14px;
            color: #ffc107;
            font-weight: 600;
            font-size: 13px;
            border: 1px solid rgba(255, 193, 7, 0.3);
            letter-spacing: 0.3px;
        """)

    def _on_data_received(self):
        """Chamado quando dados são recebidos do ESP32"""
        self.last_data_time = datetime.now()
        self.status_label.setText("🟢 MQTT: Conectado | ESP32 Online")
        self.status_label.setStyleSheet("""
            background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
                stop:0 rgba(0, 255, 157, 0.2), stop:1 rgba(0, 212, 255, 0.1));
            padding: 14px 20px;
            border-radius: 14px;
            color: #00ff9d;
            font-weight: 600;
            font-size: 13px;
            border: 1px solid rgba(0, 255, 157, 0.3);
            letter-spacing: 0.3px;
        """)

    def _try_reconnect(self):
        self.reconnect_attempts += 1
        if self.reconnect_attempts > self.max_reconnect_attempts:
            self.status_label.setText("🔴 MQTT: Falha na reconexão")
            self.status_label.setToolTip(f"Máximo de {self.max_reconnect_attempts} tentativas atingido. Reinicie o aplicativo.")
            self.reconnect_timer.stop()
            print(f"[STATUS] Máximo de tentativas de reconexão atingido ({self.max_reconnect_attempts})")
            return

        self.status_label.setText(f"🟡 MQTT: Reconectando... ({self.reconnect_attempts}/{self.max_reconnect_attempts})")

```

```

self.status_label.setStyleSheet("""
    background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
        stop:0 rgba(255, 193, 7, 0.2), stop:1 rgba(255, 152, 0, 0.1));
    padding: 14px 20px;
    border-radius: 14px;
    color: #ffc107;
    font-weight: 600;
    font-size: 13px;
    border: 1px solid rgba(255, 193, 7, 0.3);
    letter-spacing: 0.3px;
    """)
print(f"[STATUS] Tentativa de reconexão
{self.reconnect_attempts}/{self.max_reconnect_attempts}")
self.mqtt_client.connect()

def _on_temperature(self, value: float):
    self.current_temp = value
    self.temp_reading.setText(f"{value:.1f}")
    self.temp_time.setText(f"⌚ {datetime.now().strftime('%H:%M:%S')}")
    self._update_chart_and_status()

def _on_humidity(self, value: float):
    self.current_hum = value
    self.hum_reading.setText(f"{value:.0f}")
    self.hum_time.setText(f"⌚ {datetime.now().strftime('%H:%M:%S')}")
    self._update_chart_and_status()

def _update_chart_and_status(self):
    if self.current_temp is not None and self.current_hum is not None:
        self.chart.add_point(self.current_temp, self.current_hum)
        self._update_status(self.current_temp, self.current_hum)

def _on_profile_changed(self):
    if self.current_temp is not None and self.current_hum is not None:
        self._update_status(self.current_temp, self.current_hum)

def _compute_stability(self, temp: float, hum: float) -> int:
    profile_key = self.profile_combo.currentData()
    profile = PROFILES.get(profile_key)
    if not profile:
        return 0

    t_range = profile.tmax - profile.tmin
    t_dist = abs(temp - profile.optimum)
    t_score = max(0, 100 - (t_dist / t_range) * 100)

```

```

u_score = 100
if hum < profile.umin:
    u_score -= (profile.umin - hum) * 2
if hum > profile.umax:
    u_score -= (hum - profile.umax) * 2
u_score = max(0, u_score)

return round(t_score * 0.65 + u_score * 0.35)

def _update_status(self, temp: float, hum: float):
    profile_key = self.profile_combo.currentData()
    profile = PROFILES.get(profile_key)
    if not profile:
        return

    # Determinar estado
    state = 'OK'
    if temp < profile.tmin or temp > profile.tmax or hum < profile.umin or hum > profile.umax:
        state = 'CRITICAL'
    else:
        t_edge = (profile.tmax - profile.tmin) * 0.10
        if (temp - profile.tmin) < t_edge or (profile.tmax - temp) < t_edge:
            state = 'WARN'
        u_edge = max(1, (profile.umax - profile.umin) * 0.10)
        if (hum - profile.umin) < u_edge or (profile.umax - hum) < u_edge:
            state = 'WARN'

    # Atualizar estabilidade
    stability = self._compute_stability(temp, hum)
    self.stability_bar.setValue(stability)
    self.stability_label.setText(f"{stability}%")

    # Atualizar status
    if state == 'CRITICAL':
        self.profile_status.setText(f"{profile.name} — Crítico")
        self.profile_status.setObjectName("status-critical")
        self.alert_text.setText(f"CRÍTICO: {profile.name} fora da faixa! T={temp:.1f}°C
U={hum:.0f}%")
        self.alert_frame.setVisible(True)
    elif state == 'WARN':
        self.profile_status.setText(f"{profile.name} — Atenção")
        self.profile_status.setObjectName("status-warn")
        self.alert_text.setText(f"ATENÇÃO: {profile.name} aproximando limites.
T={temp:.1f}°C U={hum:.0f}%")

```

```

        self.alert_frame.setVisible(True)
    else:
        self.profile_status.setText(f"{profile.name} — Seguro")
        self.profile_status.setObjectName("status-ok")
        self.alert_frame.setVisible(False)

    # Reaplicar estilo para atualizar objectName
    self._apply_style()

def closeEvent(self, event):
    self.mqtt_client.disconnect()
    super().closeEvent(event)

# ===== MAIN =====
def main():
    app = QApplication(sys.argv)
    app.setStyle('Fusion')

    window = DashboardWindow()
    window.show()

    sys.exit(app.exec())

if __name__ == "__main__":
    main()

```