

AVALIAÇÃO 4 - RELATÓRIO DO PROJETO DE MONITORAMENTO COM ESP32, MQTT E SUPERVISÓRIO FUXA

Victor de Melo Lima Evangelista
Escola Politécnica e de Artes

20232003800032@pucgo.edu.br

Introdução

No âmbito das disciplinas do módulo profissionalizante, especialmente a disciplina **ELT1119**, a supervisão de sensores e o controle de máquinas se fazem presentes por meio do conceito de IoT — **Internet of Things** (*Internet das Coisas*). Dentro desse contexto, foram utilizados ao longo do semestre dois protocolos principais para comunicação com a ESP32: o protocolo HTTP (**HyperText Transfer Protocol**) e, posteriormente, o protocolo MQTT (**Message Queuing Telemetry Transport**).

A implementação via HTTP foi realizada por meio da criação de páginas web utilizando **HTML (HyperText Markup Language)** e **CSS**, possibilitando o controle básico dos dispositivos. Com a introdução do MQTT, adotou-se a plataforma **Node-RED**, que trouxe ganhos significativos em termos de desempenho, modularidade e qualidade visual da interface de supervisão.

Entretanto, embora o Node-RED ofereça uma interface gráfica eficiente, sua instabilidade em ambientes de execução virtualizados — especialmente em contextos educacionais com infraestrutura limitada — motivou a busca por alternativas mais leves e flexíveis. Foi nesse cenário que se explorou a plataforma **FUXA**, uma ferramenta open-source baseada em Node.js, capaz de integrar dados MQTT com boa estabilidade e visualização intuitiva.

Materiais e métodos

Hardwares e Softwares Utilizados

- **ESP32** — microcontrolador com Wi-Fi.
- **Sensor DHT11** — mede temperatura e umidade.
- **Sensor HC-SR04** — mede distância.
- **Plataforma FUXA** — supervisório open-source para dados via MQTT.
- **Broker MQTT: mqtt.cool**, broker em nuvem gratuito.
- **IDE e ferramentas:** Thonny, MicroPython, Node.js
- **Bancada da Sala 003/G**: Na ocasião, Inversor de Frequência modelo Danfoss VLT2800 e Motor Trifásico WEG.
- **Filtro Conversor de Tensão**: Montado em circuito impresso.

Arquitetura do Sistema

O ESP32 realiza leituras dos sensores conectados aos pinos 32 (**DHT11**), 26 e 27 (**TRIGGER/ECHO**) do sensor de distância e utiliza como saída o pino 25 (**DAC**). Os dados de temperatura, umidade e distância são interpretados e processados por uma lógica condicional em MicroPython. A lógica do sistema classifica os valores em faixas (baixa (**20Hz**), média(**40Hz**), alta(**60Hz**)) e libera no DAC uma faixa proporcional, que é então aplicada a um filtro passa-baixa com ganho 2, e conectada ao inversor de frequência Danfoss VLT2800.

Supervisório FUXA

O software FUXA é descrito, de acordo com o desenvolvedor: "FUXA é um software poderoso baseado na web para criar e implementar rapidamente sistemas escaláveis de SCADA, HMI, Dashboard ou IIoT. Com o FUXA, você pode criar visualizações modernas de processos com designs personalizados para suas máquinas, mostrar dados em tempo real e controlar instrumentos de plantas industriais automatizadas." Diferente do Node-RED, o FUXA é mais enxuto, rodando diretamente via Node.js. O Node-RED utiliza um sistema baseado em blocos, onde o usuário conecta módulos "pré-fabricados" para criar a lógica e supervisão do sistema. Essa metodologia facilita o desenvolvimento rápido, mas impacta a liberdade criativa ao deixar a criação de interfaces engessada. Em contrapartida, o FUXA oferece uma abordagem mais construtiva e flexível, permitindo que o usuário construa o dashboard com maior personalização e liberdade, sem depender de blocos pré-montados.

Resultados

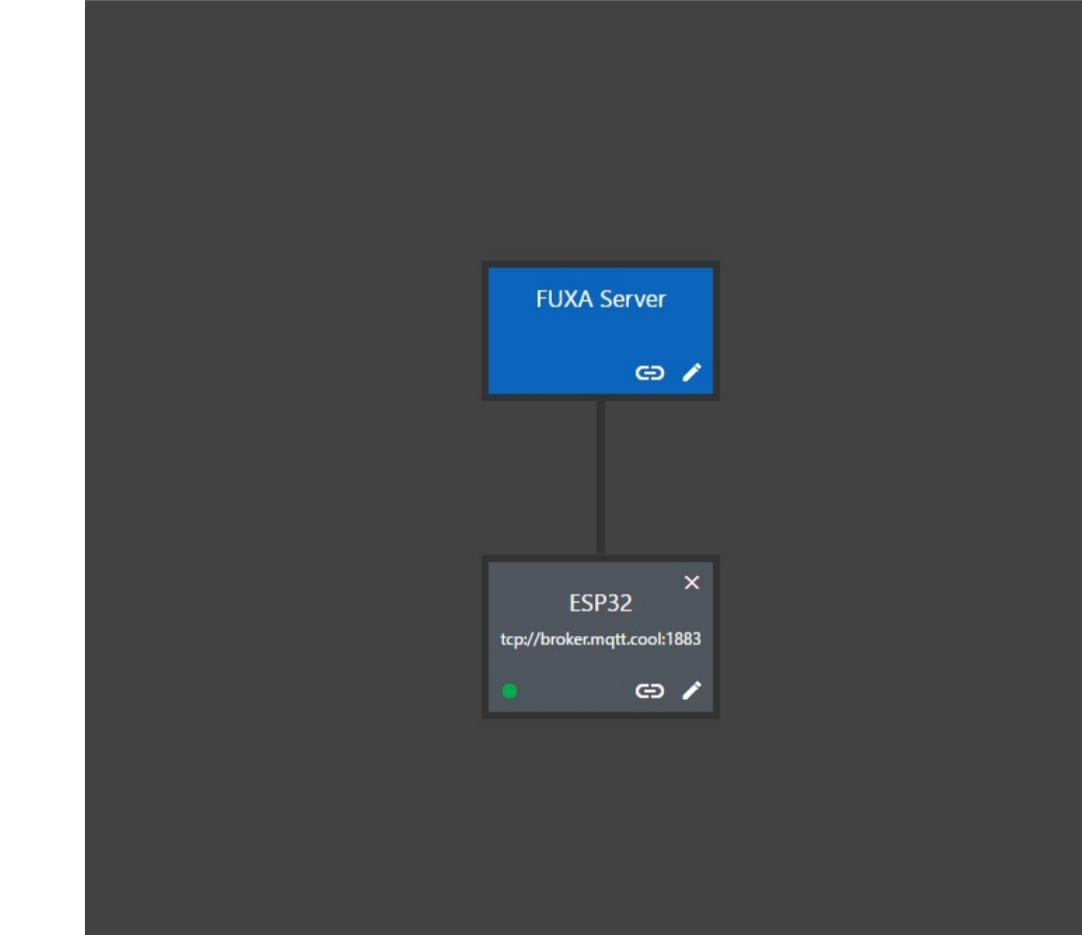


Figura 1: Página de Configuração do Server

Configurou-se o dispositivo ESP32 para enviar dados através do protocolo MQTT pelo broker **mqtt.cool**. Optou-se por utilizar um broker em nuvem no lugar de um local. Embora o broker local tenha menor latência e controle maior sobre o tráfego, o broker em nuvem garante uma acessibilidade remota maior, alta disponibilidade e a falta de necessidade em instalar quaisquer

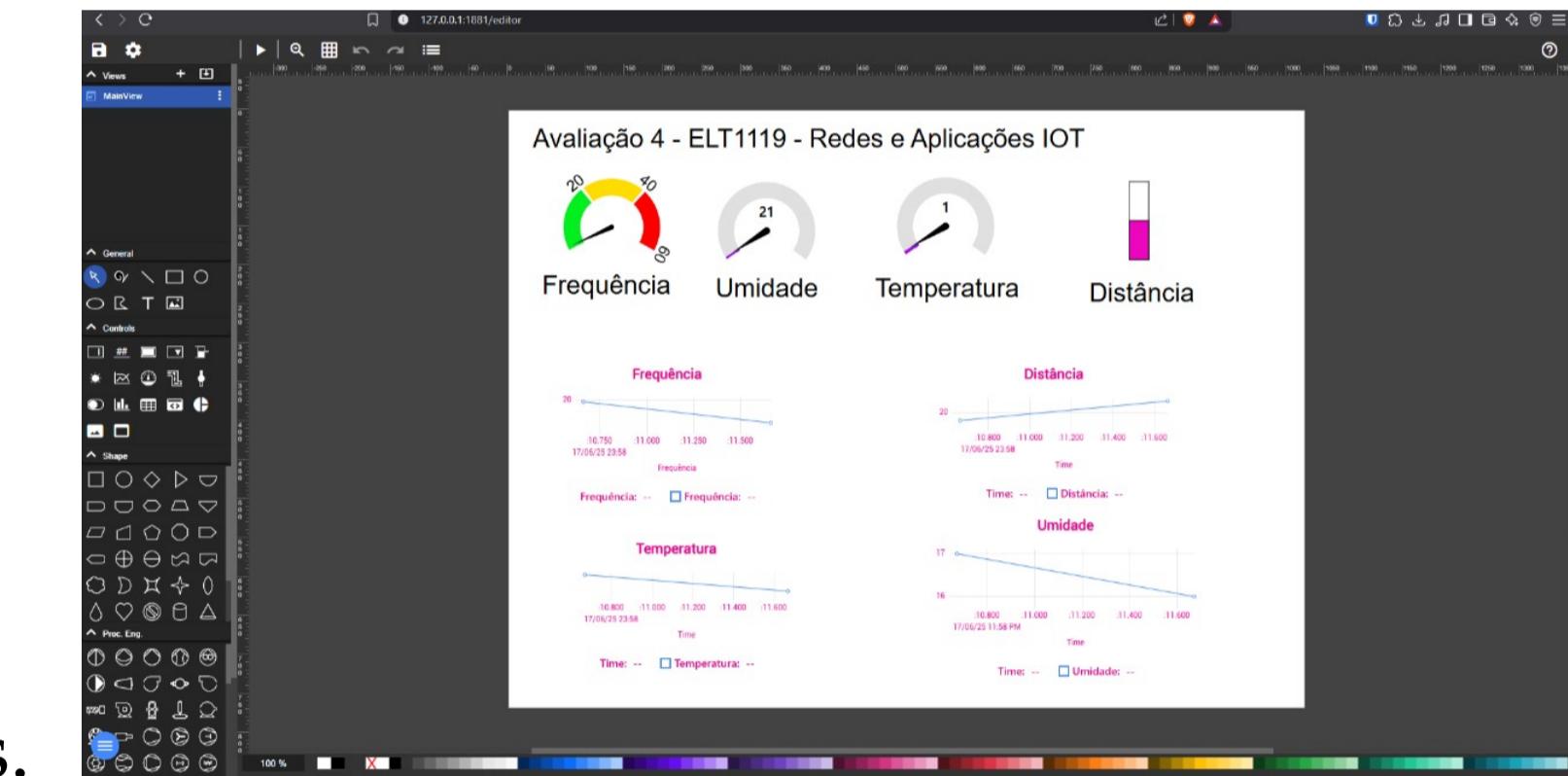
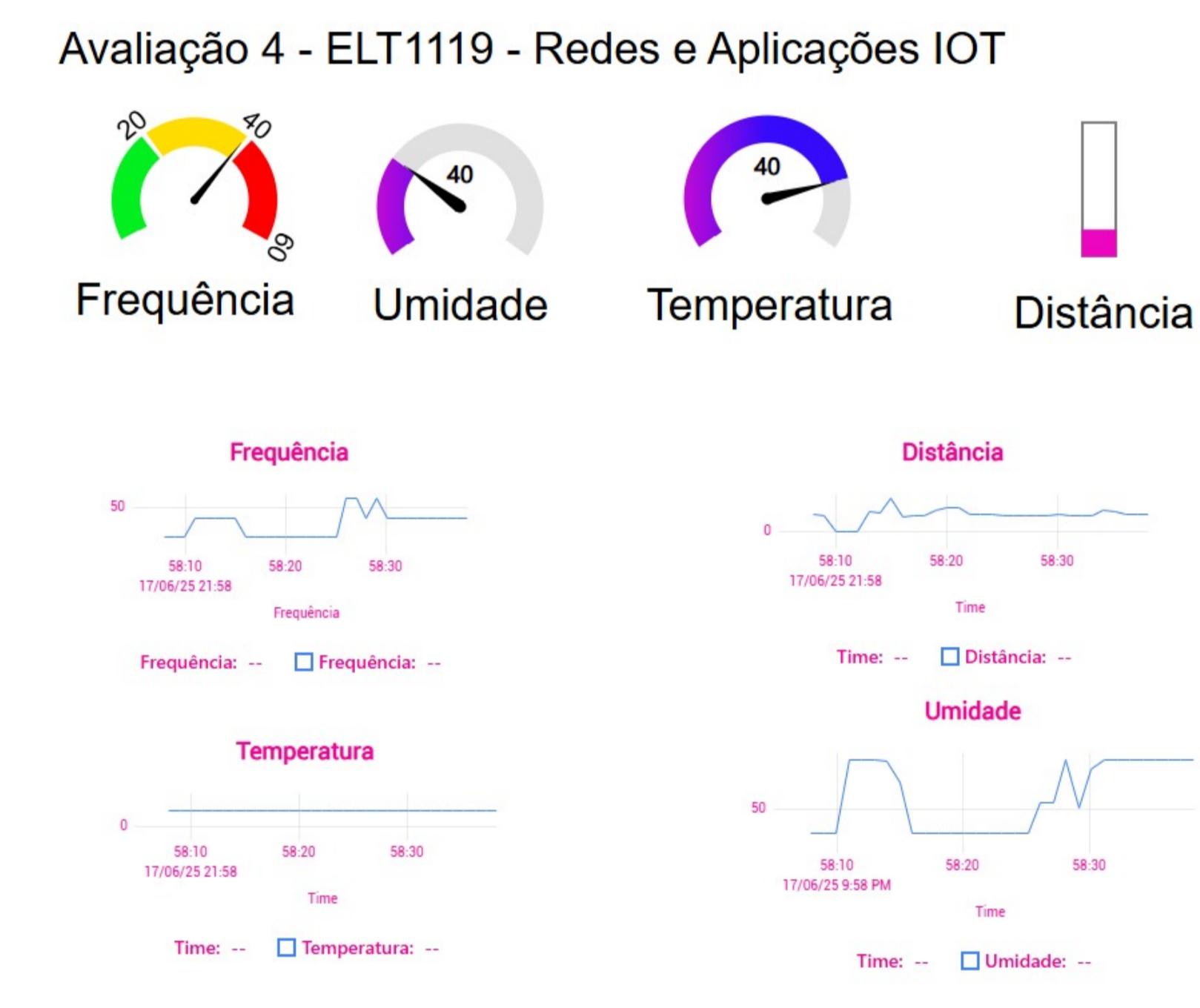


Figura 2: Página de Configuração do Dashboard

A página de configuração do Dashboard traz uma visão intuitiva, permitindo ao usuário atribuir gráficos, barras, gauges para cada tópico, sem a necessidade de formatar dados por linha de código. Não foi explorada a capacidade total do software, como por exemplo, a possibilidade de se colocar



imagens.

Figura 3: Dashboard Final

Considerações finais

Este estudo possibilitou a implementação prática e aprimoramento do estudo teórico realizado sobre sistemas de supervisão e IOT. Ao utilizar um microcontrolador de preço acessível (**ESP32**) e de plataformas Open-Source, torna-se viável a implementação de sistemas de supervisão para startups e pequenos empreendimentos, alinhado a metodologia PBL e Indústria 4.0. No que tange as plataformas de supervisão, a comparação entre Node-RED e FUXA revelou diferenças significativas em relação a metodologia de desenvolvimento e ao desempenho das interfaces. O Node-RED emprega uma abordagem visual baseada em blocos e uma lógica modular permite a criação de sistemas complexos, porém limitados visualmente e interativamente. O FUXA se destacou como opção mais leve e flexível, permitindo maior possibilidade de criação, embora seja de difícil instalação e manuseio a curto prazo.

Victor de Melo Lima Evangelista

20232003800032@pucgo.edu.br

Anexos

```

import machine
import time
import network
import ubinascii
from umqtt.simple import MQTTClient
from hcsr04 import HCSR04
import dht

# Configura os pins
TRIGGER_PIN = 26
ECHO_PIN = 27
DAC_PIN = 25
DHT_PIN = 32

# Valores de classificação
TEMP_LOW = 23
TEMP_HIGH = 28
HUMID_LOW = 30
HUMID_HIGH = 70

# Configuração WiFi e MQTT
WIFI_SSID = "REMOVIDAPORQUESTODESEGURANÁ"
WIFI_PASS = "REMOVIDAPORQUESTODESEGURANÁ"
MQTT_BROKER = "broker.hivemq.com"
TOPIC = b"esp32/freq"

class SensorSystem:
    def __init__(self):
        self.ultrasonic = HCSR04(trigger_pin=TRIGGER_PIN, echo_pin=ECHO_PIN)
        self.dac = machine.DAC(machine.Pin(DAC_PIN))

        self.dht_sensor = dht.DHT11(machine.Pin(DHT_PIN))

        self.wifi = network.WLAN(network.STA_IF)
        self.client = MQTTClient("esp32_" + ubinascii.hexlify(machine.unique_id()).decode(),
                               MQTT_BROKER)

    def setup_connections(self):
        if not self.wifi.isconnected():
            self.wifi.active(True)
            self.wifi.connect(WIFI_SSID, WIFI_PASS)
            for _ in range(10):
                if self.wifi.isconnected():
                    break
                time.sleep(1)
            try:
                self.client.connect()
            except:
                print("Failed to connect MQTT")
        def read_sensors(self):
            try:
                distance = self.ultrasonic.distance_cm()
            except:
                distance = (30 + 70) / 2
            try:
                self.dht_sensor.measure()
                temp = self.dht_sensor.temperature()
                humid = self.dht_sensor.humidity()
            except:
                temp = 25
                humid = 50
            return distance, temp, humid
        def classify(self, value, low, high):
            if value <= low:
                return "low"
            elif value >= high:
                return "high"
            return "mid"
        def calculate_freq(self, distance, temp, humid):
            d = self.classify(distance, 30, 70)
            t = self.classify(temp, TEMP_LOW, TEMP_HIGH)
            h = self.classify(humid, HUMID_LOW, HUMID_HIGH)
            lows = [d, t, h].count("low")
            highs = [d, t, h].count("high")
            if lows >= 2:
                return 20
            elif highs >= 2:
                return 40
            return 60
        def run(self):
            while True:
                try:
                    distance, temp, humid = self.read_sensors()
                    freq = self.calculate_freq(distance, temp, humid)

                    self.dac.write(int((freq / 60) * 255))

                    try:
                        self.client.publish(TOPIC, str(freq))
                    except:
                        print("MQTT error, reconnecting...")
                        self.setup_connections()

                    print(f"D: {distance:.1f}cm, T: {temp} C , H: {humid}% -> {freq}Hz")
                    time.sleep(1)

                except Exception as e:
                    print("Main loop error:", e)
                    time.sleep(5)
    system = SensorSystem()
    system.run()

```

Listing 1: CódigoESP32