Revisão	Demanda	Descrição da revisão	Data	Responsável
00	01	Emissão Inicial	28/08/2024	

Monitoramento de Temperatura e Umidade em Estoques de Alimentos

Resumo

Monitoramento contínuo de temperatura e umidade em estoques de alimentos. O projeto consiste em implementar um sistema de medição automatizada, utilizando sensores ambientais (DHT22/BME280) integrados a um microcontrolador ESP32. Os dados serão enviados via protocolo MQTT para um servidor em nuvem ou local, armazenados em banco de dados e apresentados em um dashboard interativo (Node-RED ou Grafana). O sistema emitirá alertas automáticos em caso de condições fora dos padrões estabelecidos, contribuindo para a preservação da qualidade dos alimentos e redução de perdas.

Sumário

1. Introdução

Armazéns de alimentos e produtos perecíveis exigem controle rigoroso de temperatura e umidade para evitar perdas e garantir a qualidade dos produtos. Muitas vezes, esse monitoramento é feito de forma manual e esporádica, o que pode resultar em atrasos na identificação de problemas.

O projeto propõe um sistema IoT simples para monitorar temperatura e umidade em tempo real, armazenar os dados em nuvem e emitir alertas quando os limites forem ultrapassados.

2. Justificativa

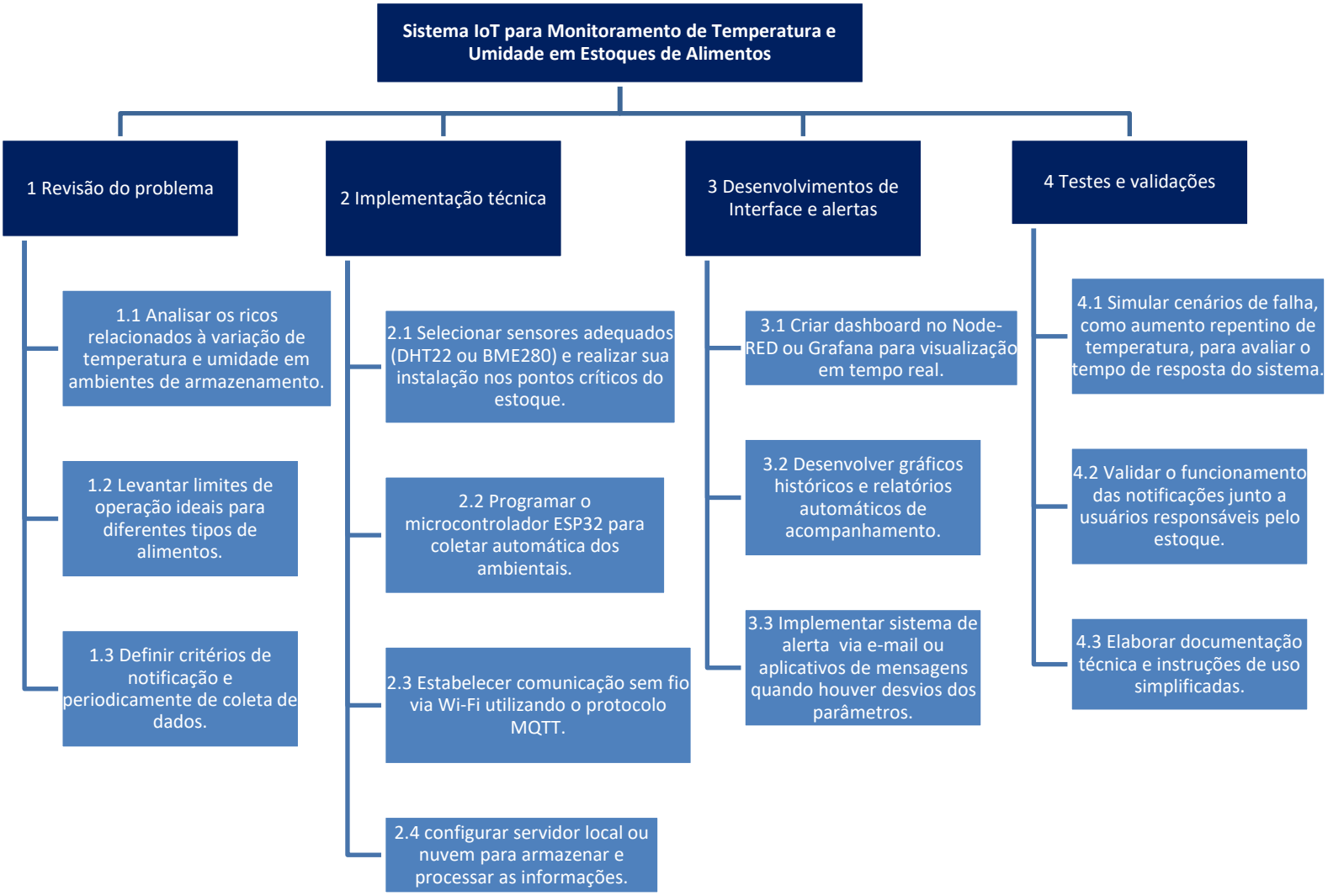
- Evitar perdas financeiras por deterioração de alimentos.
- Atender normas de conservação e segurança alimentar.
- Automatizar o processo de monitoramento, reduzindo erros humanos.
- Possibilidade de prever falhas de refrigeração antes que causem danos maiores.

3. Ganhos potenciais

- Redução de perdas em estoques.
- Melhor planejamento logístico com base em relatórios históricos.
- Suporte à manutenção preventiva (ex.: identificar falhas no sistema de refrigeração).
- Atender requisitos de fiscalização e auditoria.

4. Escopo do trabalho

O diagrama a seguir apresenta o escopo de trabalho para a execução do projeto, desde a identificação do problema, detalhamento dos requisitos



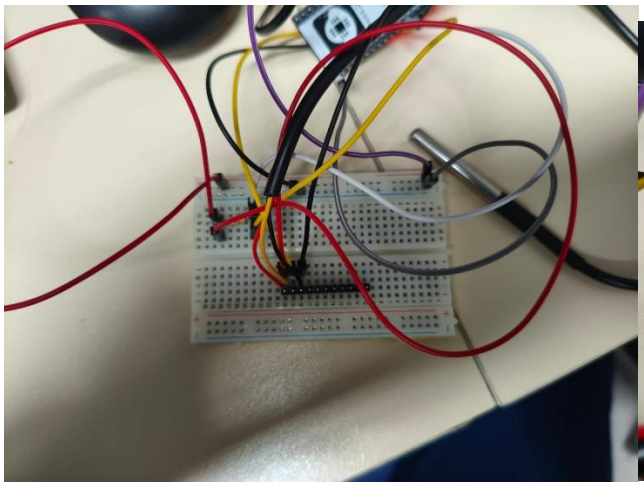
5. Tecnologias e equipamentos

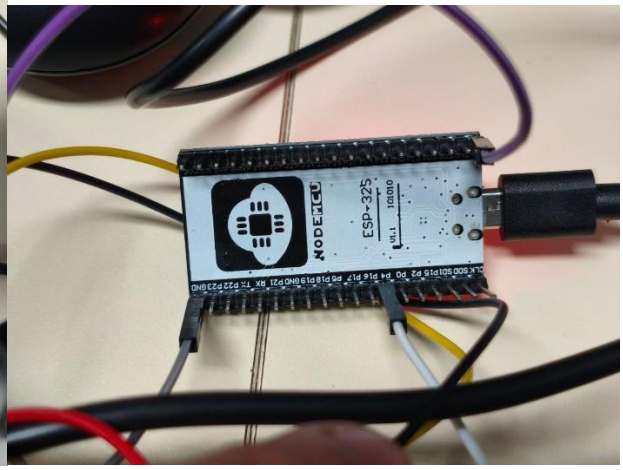
- Sensores: DHT22 ou BME280.
- Microcontrolador: ESP32.
- Protocolo de comunicação: MQTT.
- Plataforma de visualização: Node-RED / Grafana.
- Banco de dados: InfluxDB ou MySQL.
- Rede: Wi-Fi.

6. Partes interessadas

Parte interessada (área)	Representante(s)
Empresa exemplo	Coordenador do projeto
Aluno	

7. Diagrama do Circuito





8. Códigos

Arduino

```
#include <OneWire.h>
```

```
#include <DallasTemperature.h>

#define PINO_DADOS 4  // GPIO4 do DS18B20

// Inicialização do sensor
OneWire oneWire(PINO_DADOS);
DallasTemperature sensors(&oneWire);
DeviceAddress sensor;
bool sensorConectado = false;

void setup() {
  Serial.begin(115200);
  Serial.println("Iniciando...");

  // Inicializa o sensor DS18B20
  sensors.begin();

  if (sensors.getAddress(sensor, 0)) {
    sensorConectado = true;
    sensors.setResolution(sensor, 12);
    Serial.println("Sensor encontrado!");
  } else {
    Serial.println("Sensor NÃO encontrado. Verifique a conexão.");
  }
}

void loop() {
  if (!sensorConectado) {
    Serial.println("Sensor ainda não conectado.");
    delay(2000);
    return;
  }

  sensors.requestTemperatures();
  float soma = 0;
  int leituras = 5;

  for (int i = 0; i < leituras; i++) {
    float leitura = sensors.getTempC(sensor);

    if (leitura == DEVICE_DISCONNECTED_C) {
      Serial.println("Erro: Sensor desconectado!");
      return;
    }

    soma += leitura;
    delay(50);
  }

  float temperaturaMedia = soma / leituras;
  Serial.print("Temperatura média: ");
  Serial.print(temperaturaMedia);
```

```
 Serial.println(" °C");

 delay(1000);
}
```

serial_to_mqtt.py

```
import serial
import paho.mqtt.client as mqtt
import time

# Configurações da porta serial
serial_port = 'COM7'
baud_rate = 115200 # Deve ser IGUAL ao Serial.begin() no seu ESP32

# Configurações do broker MQTT
broker_address = "test.mosquitto.org"
broker_port = 1883
topic = "esp32/dados_sensores"
client_id = "Python-Serial-Publisher"

# --- Callbacks MQTT ---

# Assinatura correta para a V2 (5 argumentos)
def on_connect(client, userdata, flags, reason_code, properties):
    if reason_code == 0:
        print("Conectado ao broker MQTT!")
    else:
        print(f"Falha na conexão, código de retorno: {reason_code}")

# CORRIGIDO: Assinatura correta para a V2 (5 argumentos)
# A biblioteca envia rc, reason_code e properties
def on_disconnect(client, userdata, rc, reason_code, properties):
    print("Desconectado do broker MQTT!")

# Assinatura correta para a V2 (5 argumentos)
def on_publish(client, userdata, mid, reason_code, properties):
    # Este callback é opcional, mas bom para debug
    print(f"Mensagem (mid: {mid}) publicada.")

# --- Inicialização ---

# Usa a API Versão 2
client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2, client_id)

# Define os callbacks
client.on_connect = on_connect
client.on_disconnect = on_disconnect
client.on_publish = on_publish

# Conecta ao broker MQTT
try:
```

```
    client.connect(broker_address, broker_port, 60)
except Exception as e:
    print(f"Não foi possível conectar ao MQTT: {e}")
    exit(1) # Sai do script se não puder conectar

# Inicia o loop do MQTT em uma thread separada
client.loop_start()

try:
    # Abre a porta serial
    ser = serial.Serial(serial_port, baud_rate, timeout=1)
    print(f"Porta serial {serial_port} aberta. Lendo dados...")

    while True:
        # Lê uma linha da porta serial (ignora bytes malformados)
        line = ser.readline().decode('utf-8', errors='ignore').strip()

        # Se a linha não estiver vazia
        if line:

            # Filtro para publicar APENAS os dados de temperatura
            if line.startswith("Temperatura média:"):
                print(f"Dados recebidos da serial: {line}")

                # Publica a mensagem no tópico
                result = client.publish(topic, line)

                if result.rc != mqtt.MQTT_ERR_SUCCESS:
                    print(f"Falha ao enfileirar a mensagem: {result.rc}")
            else:
                # Ignora o "lixo" do boot, mas mostra no console
                print(f"Serial (ignorando): {line}")

except serial.SerialException as e:
    print(f"Erro ao abrir a porta serial: {e}")
except KeyboardInterrupt:
    print("Programa encerrado pelo usuário.")
finally:
    # Limpa as conexões
    print("Fechando conexões...")
    client.loop_stop()
    client.disconnect()
    if 'ser' in locals() and ser.is_open:
        ser.close()
    print("Conexão MQTT e porta serial fechadas.")
```

Node Red

```
[
    {
        "id": "305644662ef582d5",
        "type": "ui_gauge",
        "z": "f5117f73.8d076",
```

```
    "name": "",
    "group": "7825c470b9afb094",
    "order": 0,
    "width": 0,
    "height": 0,
    "gtype": "gage",
    "title": "gauge",
    "label": "units",
    "format": "{{value}}",
    "min": 0,
    "max": 10,
    "colors": [
      "#00b500",
      "#e6e600",
      "#ca3838"
    ],
    "seg1": "",
    "seg2": "",
    "diff": false,
    "className": "",
    "x": 1150,
    "y": 360,
    "wires": []
  },
  {
    "id": "7825c470b9afb094",
    "type": "ui_group",
    "name": "Monitoramento de Temperatura",
    "tab": "b252960b10a74f59",
    "order": 1,
    "disp": true,
    "width": "8",
    "collapse": false,
    "className": ""
  },
  {
    "id": "b252960b10a74f59",
    "type": "ui_tab",
    "name": "Monitoramento de Temperatura",
    "icon": "dashboard",
    "order": 3,
    "disabled": false,
    "hidden": false
  }
]
```