Relatório de Integração

Cibersistemas

Monitoramento de Motor

Augusto P. Dantas

Jaraguá do Sul - 17/02/2026

Senai

Relatório de Integração

Cibersistemas

Monitoramento de Motor

Relatório Técnico apresentado à Unidade Curricular de Programação para Coleta de Dados em Automação como requisito para avaliação da prova 2

Autor: Augusto P. Dantas

Docente: Lucas

Jaraguá do Sul - 17/02/2026

Senai

Sumário

Introdução

Este artigo tem como objetivo apresentar a implementação teórica e prática de um sistema de monitoramento de um motor em um contexto industrial. Devido à natureza da aplicação, um monitoramento estável e preciso é requerido.

Arquitetura do Sistema

Foi escolhido o MQTT como protocolo de comunicação por ser extremamente leve, padrão em IoT, e que utiliza um Broker como intermediário para conectar os dispositivos.

No quesito redes, foi optado pelo uso da topologia de estrela, onde os componentes do sistema não conversam diretamente entre si, mas sim passando por um intermediário (servidor Broker MQTT).

O fluxo de dados começa com a coleta pelo sensor DHT22 e pelos sensores de simulados de corrente e vibração e pelo processamento inicial do Esp32, em seguida, é enviado à nuvem para ser redirecionado pelo servidor Broker MQTT para o App coletor.

Ambiente de Desenvolvimento

IDE: Codespaces;

Simulação: Wokwi.

Arquitetura de Dados e Protocolo

Relativo aos requisitos de dados, o sistema foi arquitetado para realizar a aquisição de variáveis ambientais através dos sensores temperatura (DHT22) , corrente (simulado) e vibração (simulado) , que efetuam a leitura dos parâmetros citados num ciclo contínuo de 1000 milissegundos. Contudo, o *firmware* realiza a converção do valor de ponto flutuante (*float*) para uma cadeia de caracteres (*string*) antes da transmissão, descartando os dados higrométricos na camada de aplicação. No que concerne à infraestrutura de comunicação, a solução implementa o protocolo MQTT sobre TCP/IP

numa topologia de estrela , utilizando o *Broker* público HiveMQ (broker.hivemq.com) acessível através da porta 1883. O intercâmbio de mensagens é centralizado no tópico senai/Augusto/motor/dados , operando com Qualidade de Serviço (QoS) nível 0 (*At most once*), dada a utilização dos métodos padrão da biblioteca PubSubClient que não implementam persistência ou confirmação de entrega. Para assegurar a integridade das sessões, o dispositivo de borda (ESP32) autentica-se com o identificador estático ESP32_Augusto_Dantas , enquanto a aplicação consumidora (Java) gera um *Client ID* dinâmico, concatenando a raiz JavaClient_Augusto_Dantas ao *timestamp* do sistema, mitigando assim colisões de conexão no servidor.

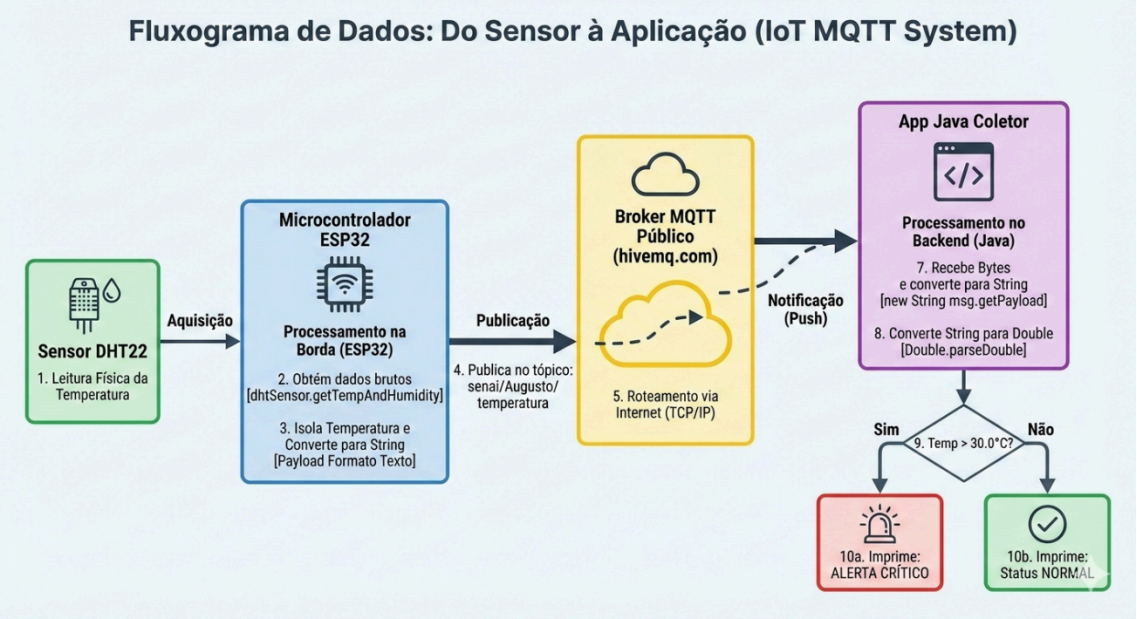

Figura 1 - Fluxograma

Esquema Elétrico e Firmware

Lista de Componentes:

1. Placa de desenvolvimento Esp32 wroom32;
2. Sensor DHT22;
3. Cabo USB-C para programação e alimentação;
4. Fios para a conexão do sensor com o Esp32.

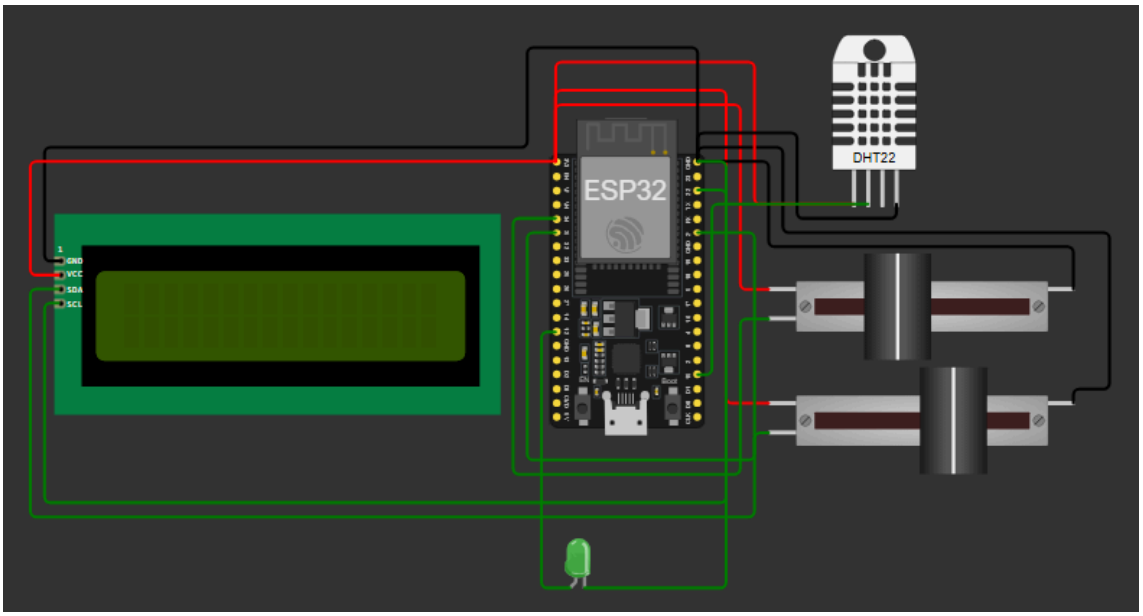

Figura 2 – Esquema Elétrico

```
#include <WiFi.h>              /* Biblioteca para conexão Wi-Fi do ESP32 */
#include <PubSubClient.h>      /* Biblioteca para comunicação via protocolo MQTT */
#include <DHTesp.h>            /* Biblioteca para controle do sensor DHT */
#include <LiquidCrystal_I2C.h>  /* Bibliteca para controle do lcd I2C */

/* Definições da rede Wi-Fi e do Broker MQTT */
const char* ssid = "Wokwi-GUEST";             /* Nome da rede Wi-Fi */
```

```
const char* password = "";                    /* Senha da rede Wi-Fi */
const char* mqtt_server = "broker.hivemq.com";   /* Endereço do servidor Broker */

/* Criação dos objetos de conexão e sensor */
WiFiClient espClient;          /* Cria o cliente TCP */
PubSubClient client(espClient); /* Cria o cliente MQTT usando a conexão TCP */
DHTesp dhtSensor;              /* Cria a instância do sensor DHT */

#define TEMPSENSOR 15
#define VIBRATIONSENSOR 34
#define CURRENTSENSOR 35
#define LED 12

#define MQTT_PORT 1883

LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup()
{
 Serial.begin(11520);                    /* Inicia a comunicação Serial para debug */
 dhtSensor.setup(TEMPSENSOR, DHTesp::DHT22);   /* Configura o sensor no pino 15 como
DHT22 */
 WiFi.begin(ssid, password);                          /* Inicia a conexão com a rede Wi-Fi */
 client.setServer(mqtt_server, MQTT_PORT);     /* Define o endereço e porta do Broker MQTT */
 lcd.init();
 lcd.backlight();
 lcd.setCursor(0, 0);
}

void loop()
{
 /* Verifica se a conexão com o Broker está ativa */
 if(!client.connected())
 {
  digitalWrite(LED, LOW);
  /* Tenta reconectar usando um ID de cliente único */
  client.connect("ESP32_Augusto_Dantas");
 }
 digitalWrite(LED, HIGH);
 /* Realiza a leitura dos dados dos sensores */
 int temperature = dhtSensor.getTempAndHumidity().temperature;
 int vibration = analogRead(VIBRATIONSENSOR);
 int current = analogRead(CURRENTSENSOR) / 100;

 /* Converte o valor numérico da temperatura para String*/
 String payload = String(temperature) + "," + String(vibration) + "," + String(current);

 /* Imprime a temperatura no LCD */
 lcd.clear();
 lcd.print(String(teperature) + "°C");

 /* Publica a mensagem no tópico específico */
 client.publish("senai/augusto/motor/dados", payload.c_str());
```

```
  /* Exibe no monitor serial para conferência */
  Serial.println("Enviado: " + payload);

  /* Aguarda 1 segundo antes de reiniciar o loop */
  delay(2000);
}
```

Figura 3 - Firmware

Backend

```
package senai.augusto;


import org.eclipse.paho.client.mqttv3.MqttClient;

import org.eclipse.paho.client.mqttv3.MqttConnectOptions;

import org.eclipse.paho.client.mqttv3.MqttException;


public class AppColetor {

   public static void main(String[] args) {

      String broker = "tcp://broker.hivemq.com:1883";

      String clientId = "JavaClient_Augusto_Dantas" + System.currentTimeMillis();

      String topic = "senai/Augusto/temperatura";


      try {

         MqttClient client = new MqttClient(broker, clientId);

         MqttConnectOptions options = new MqttConnectOptions();

         options.setCleanSession(true);


         System.out.println("Conectando ao Broker...");

         client.connect(options);

         System.out.println("Conectado com sucesso!");


         // Assinando o tópico

         client.subscribe(topic, (t, msg) -> {

            String payload = new String(msg.getPayload());
```

```
            String[] data = payload.split(",");
            double temperature = Double.parseDouble(data[0]);
            double vibration = Double.parseDouble(data[1]);
            double current = Double.parseDouble(data[2]);

            System.out.println("\n----------------------------");
            System.out.println("DADO RECEBIDO DA AUTOMAÇÃO:");
            System.out.println("Temperatura: " + temperature + "°C");
            System.out.println("Vibration: " + vibration);
            System.out.println("Corrente: " + current);

            if (temperature > 30.0) {
               System.err.println("STATUS: [ALERTA] Temperatura Crítica!");
            } else {
               System.out.println("STATUS: [NORMAL] Operação estável.");
            }
         });

      } catch (MqttException e) {
         System.out.println("Erro de conexão: " + e.getMessage());
      }
   }
}
```

Figura 4 – APP Coletor

Planejamento e Adaptação

Atividade	Responsável	Prazo	Status
Configuração Broker	Lucas dos Santo	16/02	Concluído
Firmware C++	Augusto	16/02	Concluído
Backend Java	Augusto	16/02	Concluído

Testes

ID	Descrição do Teste	Procedimento Executado	Resultado Esperado	Resultado Obtido	Status (P/F)
01	Conexão MQTT	Verificar o monitor serial do Esp32 após o boot.	Mensagem "Connected to MQTT Broker" exibida.	Mensagem "Connected to MQTT Broker" exibida.	P
02	Leitura Sensor	Modificar a leitura dos sensores	Valores de leitura variam no monitor serial.	Valores de leitura variam no monitor serial.	P
03	Handshake Java	Iniciar o Codespaces e observar o terminal Java.	Terminal indica recepção de nova mensagem do Broker.	Terminal indica recepção de nova mensagem do Broker.	P

Problemas no Desenvolvimento

Devido ao alto tráfego de dados no *build server* da plataforma Wokwi, a compilação tomou ritmos inesperados, demorando horas para ser realizada, o que prejudicou de maneira significativa o desenvolvimento do projeto final.

Conclusão

O desenvolvimento e a implementação do sistema de automação e monitorização ambiental atingiram com êxito os objetivos propostos, garantindo a recolha teórica e prática de dados críticos para a refrigeração do sistema de armazenamento. A solução arquitetada demonstrou robustez ao integrar o microcontrolador ESP32 com o sensor DHT22 e uma aplicação backend em Java, mediados pelo protocolo MQTT.

Fontes:

Arduino Client for MQTT. Disponível em:
<http://pubsubclient.knolleary.net/api>.

BEEGEE_TOKYO. **DHT sensor library for ESPx**. Disponível em:
<https://www.arduinolibraries.info/libraries/dht-sensor-library-for-es-px>.
Acesso em: 17 fev. 2026.

WIFI library for ESPx Disponível em: <https://docs.arduino.cc/libraries/wifi/>.