RELATÓRIO DE MEDIÇÃO DE CAMPO MAGNÉTICO COM SENSOR KY-024 E ARDUINO ESP8266

Mauricio Cepinho

Rodrigo de Andrade

4° Semestre – Desenvolvimento de Software Multiplataforma

Professor Henrique Louro

Jacareí - SP
MAIO/2025

Relatório de Medição de Campo Magnético com Sensor KY-024 e Arduino ESP8266

1. Objetivo

Documentar o sistema integrado de aquisição, transmissão Wi-Fi e armazenamento de medições de campo magnético usando:

- Sensor KY-024 acoplado ao ESP8266
- Backend em Node.js/Express para recebimento de dados
- Banco de dados MongoDB para persistência

2. Código Arduino (ESP8266 via Wi-Fi)

Arquivo: esp8266.ino

```
#include <ESP8266HTTPClient.h>

#include <WiFiClient.h>



// Configurações de WiFi

const char* ssid = "POCOIoT";

const char* password = "teste123";



// Configurações do servidor

const char* serverUrl = "http://192.168.22.25:3000/leituras";

const int sensorPin = A0;    // KY-024 no pino analógico

const int ledPin = 16;       // LED no GPIO16 (D0)

const int threshold = 874;   // Valor limite ajustado
```

```
void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);

  // Conecta ao WiFi
  WiFi.begin(ssid, password);
  Serial.println("Conectando ao WiFi...");

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi conectado");
  Serial.println("Endereço IP: ");
  Serial.println(WiFi.localIP());
}
```

```
void loop() {
  if (WiFi.status() == WL_CONNECTED) {
    int valorSensor = analogRead(sensorPin);
    bool imaDetectado = valorSensor > threshold;

    // Adicionado: Print dos valores no Serial Monitor
    Serial.print("Valor do sensor: ");
    Serial.print(valorSensor);
    Serial.print(" | Estado: ");
    Serial.println(imaDetectado ? "IMÃ DETECTADO" : "SEM IMÃ");

    // Controle do LED
    digitalWrite(ledPin, imaDetectado ? HIGH : LOW);

    // Envia dados para o servidor
    WiFiClient client;
    HTTPClient http;

    http.begin(client, serverUrl);
    http.addHeader("Content-Type", "application/json");
```

```
    // Cria o JSON com os dados
    String jsonPayload = "{\"valor\":" + String(valorSensor) +
                         ",\"estado\":\"" + (imaDetectado ? "IMÃ_DETECTADO" : "SEM_IMÃ")
+
                         "\"}";

    // Print do JSON que será enviado
    //Serial.print("Enviando JSON: ");
    Serial.println(jsonPayload);

    // Envia a requisição POST
    int httpResponseCode = http.POST(jsonPayload);

    if (httpResponseCode > 0) {
      Serial.print("Dados enviados. Resposta HTTP: ");
      Serial.println(httpResponseCode);
    } else {
      Serial.print("Erro no envio. Código: ");
      Serial.print(httpResponseCode);
      Serial.print(" | Descrição: ");
      Serial.println(http.errorToString(httpResponseCode));
    }
```

```
    http.end();
  } else {
    Serial.println("Conexão WiFi perdida");
  }

  delay(2000);  // Intervalo entre leituras
}
```

Funcionalidade:

Conecta-se ao Wi-Fi e envia dados via HTTP POST para o servidor

Ativa LED (GPIO16) quando o valor do sensor ultrapassa 874

Formata dados em JSON, exemplo:{"valor":900, "estado":"IMÃ_DETECTADO"}

Resultados Esperados:

```
Conectando ao WiFi...
.....
WiFi conectado
Endereço IP: 192.168.22.30

HTTP Code: 201
{"valor":900,"estado":"IMÃ_DETECTADO"}

HTTP Code: 201
{"valor":600,"estado":"SEM_IMÃ"}
```

LED: Acende quando `valor > 874` (ímã detectado).

Saída: VALOR:[valor], ESTADO:[IMÃ_DETECTADO/SEM_IMÃ]

3. Banco de Dados (MongoDB) e o Backend

Coleção: Leitura

Estrutura: Do arquivo leitura.ts

```
import { Schema, model, Document } from 'mongoose';

export interface ILeitura extends Document {
  valor: number;
  estado: string;
  timestamp: Date;
}

const leituraSchema = new Schema<ILeitura>({
  valor: { type: Number, required: true },
  estado: { type: String, enum: ['IMÃ_DETECTADO', 'SEM_IMÃ'], required: true },
  timestamp: { type: Date, default: Date.now }
});

export default model<ILeitura>('Leitura', leituraSchema);
```

Funcionalidade:

Validação de Dados: Garante integridade dos dados antes de salvar no banco.
Tipagem Estática (TypeScript): Autocompletar e verificação de tipos durante desenvolvimento.
Timestamp Automático: Registra o momento da criação do documento.
Enumeração Controlada: Restringe os valores possíveis para estado.

Exemplo de Documento:

```
{
  "valor": 900,
  "estado": "IMÃ_DETECTADO",
  "timestamp": "2024-05-20T14:30:00.000Z"
}
```

Coleção: Conexão com o MongoDB

Estrutura: Do arquivo index.ts

```
import express from 'express';
import mongoose from 'mongoose';
import bodyParser from 'body-parser';
import path from 'path'; // Adicione esta linha
import Leitura from './leitura';

const app = express();
app.use(bodyParser.json());

// Configuração para servir arquivos estáticos
app.use(express.static(path.join(__dirname, 'public')));

const PORT = 3000;
const MONGODB_URI = 'mongodb://127.0.0.1:27017/campo_magnetico';

// Conexão com MongoDB
async function connectToDatabase() {
  try {
    await mongoose.connect(MONGODB_URI, {
      serverSelectionTimeoutMS: 30000,
      socketTimeoutMS: 45000,
    });
    console.log('✅ Conectado ao MongoDB');
```

```
  } catch (err) {
    console.error('❌ Falha na conexão com MongoDB:', err);
    process.exit(1);
  }
}

connectToDatabase();

// Rota raiz - serve o arquivo index.html
app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname,'index.html'));
});

// Rota para obter leituras
app.get('/leituras', async (req, res) => {
  try {
    const leituras = await Leitura.find().sort({ timestamp: -1 });
    res.json(leituras);
  } catch (error) {
    res.status(500).json({ error: 'Erro ao buscar leituras' });
  }
});

// Rota para receber dados do ESP8266
app.post('/leituras', async (req, res) => {
  try {
    const { valor, estado } = req.body;

    const novaLeitura = await Leitura.create({
      valor,
      estado,
      timestamp: new Date(),
    });

    res.status(201).json(novaLeitura);
  } catch (error) {
    res.status(500).json({ error: 'Erro ao salvar leitura' });
```

```
  }
});

app.listen(PORT, () => {
  console.log(`🚀 Servidor rodando em http://localhost:${PORT}`);
});
```

Funcionalidade:

Conecta-se à porta serial COM4 (baud rate: 115200).

Processa dados no formato VALOR:[valor],ESTADO:[estado] .

Salva leituras no MongoDB após validação do limiar (THRESHOLD = 874).

4. Fluxo Completo

ESP8266 lê o sensor e envia dados via HTTP POST para http://192.168.22.25:3000/leituras

Recebe dados via Wi-Fi conectado, válida estrutura e salva no MongoDB.

5. Dependências

- Arduino: Biblioteca ESP8266WiFi (ESP8266HTTPClient e WiFiClient.)
- Backend: Express, Mongoose

```
"dependencies": {
  "express": "^5.1.0",
  "mongoose": "^8.14.1",
  "body-parser": "^2.2.0"
}
```

6. Testes e Calibração

Monitor Serial do Arduino:

WiFi conectado

HTTP Code: 201

Logs do Servidor:

✅ Conectado ao MongoDB

🚀 Servidor rodando em http://localhost:3000

7. Conclusão:

O sistema integra hardware e software para monitoramento de campo magnético, armazenamento em banco de dados e visualização em tempo real conectado via Wi-Fi.

A arquitetura proposta combina sensores de alta sensibilidade, microcontroladores para processamento local e uma interface intuitiva que permite visualização imediata dos dados coletados. Além disso, a implementação de um banco de dados estruturado garante o armazenamento seguro e eficiente das informações, facilitando análises históricas e a identificação de padrões ao longo do tempo.

Em síntese, este trabalho não apenas comprova a viabilidade técnica da integração entre hardware dedicado, gestão de dados e visualização dinâmica, mas também abre caminho para aplicações inovadoras em áreas como segurança industrial, pesquisa científica e monitoramento ambiental, reforçando a importância de sistemas inteligentes na era da transformação digital.