

Documentação: Datalogger de Temperatura e Umidade com DHT22

Autor: Victor Sarrís: 05 de Outubro de 2025

1. Introdução e Objetivo

Este projeto consiste em um datalogger multifuncional implementado na plataforma Single-Board Computer (SBC) Labrador. O objetivo principal é monitorar continuamente as condições ambientais de um local, coletando dados de temperatura e umidade através de um sensor DHT22.

O sistema foi projetado para registrar as leituras em intervalos de tempo periódicos, armazenando os dados com um timestamp em um arquivo de texto (`.txt`) localizado em um cartão microSD, permitindo a análise posterior dos dados coletados.

2. Escolha do Sensor (DHT22)

Para o monitoramento das variáveis de ambiente, o sensor escolhido foi o **DHT22** (também conhecido como AM2302).

- **Funcionalidade:** O DHT22 é um sensor digital capaz de medir tanto a temperatura quanto a umidade relativa do ar.
- **Justificativa da Escolha:** A escolha se deu por sua popularidade em projetos de prototipagem, baixo custo, boa precisão para aplicações gerais e, principalmente, pela simplicidade de conexão, utilizando apenas um pino de dados para a comunicação.

A conexão física foi realizada utilizando os pinos de alimentação (no meu caso a 5V) e aterramento (GND) da Labrador, e o pino de dados do sensor foi conectado a um pino de entrada/saída de propósito geral (GPIO) da placa.

3.1. Lógica de Coleta

A coleta de dados é realizada por um script em Python que interage diretamente com os pinos GPIO da placa Labrador.

1. **Mapeamento de Pinos:** A pinagem da Labrador organiza os GPIOs em **Grupos** (A, B, C, etc.) e **Linhas** (números). Para este projeto, foi utilizado o pino **GPIOA25** (Grupo A, Linha 25), localizado no pino 15 do conector de 40 pinos.
2. **Comunicação com o Sensor:** Em vez de usar bibliotecas de abstração que se mostraram incompatíveis, a solução final utiliza a biblioteca `gpiod-dht`. Esta biblioteca se comunica diretamente com o subsistema GPIO padrão do Linux, o `libgpiod`, que é como a Labrador gerencia seus pinos.

3. **Leitura dos Dados:** O script especifica o "chip" GPIO correspondente ao Grupo A (`/dev/gpiochip0`) e a linha (25) para acessar o pino `GPIOA25`. Uma função da biblioteca é chamada periodicamente para requisitar os dados de temperatura e umidade do sensor DHT22.

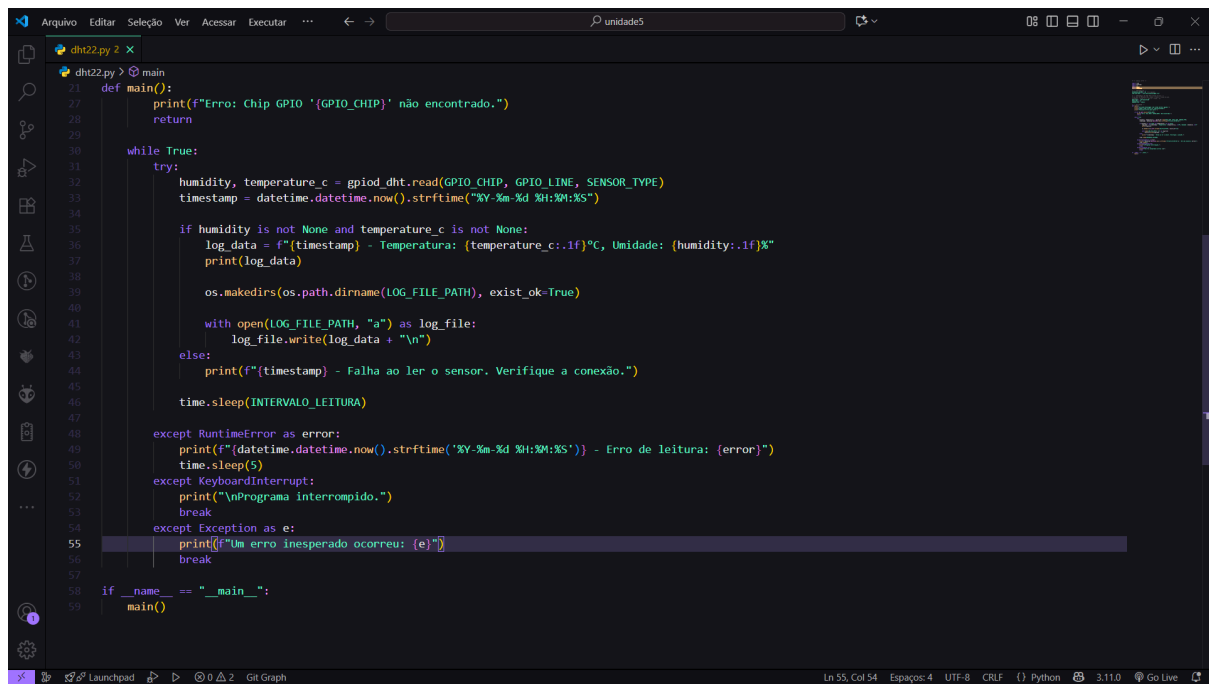
3.2. Lógica de Armazenamento

1. **Formato do Arquivo:** As leituras são armazenadas em um arquivo de texto simples (`datalogger.txt`) no cartão microSD para facilitar o acesso e a manipulação dos dados.
2. **Modo de Escrita:** O script abre o arquivo em modo "append" (`'a'`), o que adiciona cada nova leitura ao final do arquivo, preservando o histórico completo.
3. **Estrutura do Registro:** Cada linha no arquivo contém um timestamp, a leitura da temperatura e a leitura da umidade, seguindo um formato claro e consistente:
`AAAA-MM-DD HH:MM:SS - Temperatura: XX.X°C, Umidade: XX.X%`

4. Código-Fonte da Solução Final

```
Arquivo Editar Seleção Ver Acessar Executar ... unidade5
```

```
dht22.py 2 X  
dht22.py > ...  
1 # -*- coding: utf-8 -*-  
2  
3 import time  
4 import datetime  
5 import os  
6 import gpiod  
7 import gpiod_dht  
8  
9 --- CONFIGURAÇÕES ---  
10 INTERVALO_LEITURA = 60 # Em segundos  
11 LOG_FILE_PATH = "/mnt/sdcard/datalogger.txt"  
12  
13 --- CONFIGURAÇÃO DO PINO GPIO (Método Direto) ---  
14 # Com base na documentação, usamos o chip e a linha do pino.  
15 # GPIOA25 -> Grupo A e o chip 0. A linha é 25.  
16 GPIO_CHIP = '/dev/gpiochip0'  
17 GPIO_LINE = 25  
18 SENSOR_TYPE = "DHT22"  
19  
20 # --- DATALOGGER ---  
21 def main():  
22     print("Iniciando Datalogger com método direto (gpiod).")  
23     print(f"Dados serão salvos em: {LOG_FILE_PATH}")  
24     print("Pressione Ctrl+C para parar.")  
25  
26     if not os.path.exists(GPIO_CHIP):  
27         print(f"Erro: Chip GPIO '{GPIO_CHIP}' não encontrado.")  
28         return  
29  
30     while True:  
31         try:  
32             humidity, temperature_c = gpiod_dht.read(GPIO_CHIP, GPIO_LINE, SENSOR_TYPE)  
33             timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")  
34  
35             if humidity is not None and temperature_c is not None:  
36                 log_data = f"{timestamp} - Temperatura: {temperature_c:.1f}°C, Umidade: {humidity:.1f}%"  
37                 print(log_data)
```



```
Arquivo  Editar  Seleção  Ver  Acessar  Executar  ...  unidade5  dht22.py 2 X
dht22.py > main
21 def main():
22     print(f"Erro: Chip GPIO '{GPIO_CHIP}' não encontrado.")
23     return
24
25 while True:
26     try:
27         humidity, temperature_c = gpio_dht.read(GPIO_CHIP, GPIO_LINE, SENSOR_TYPE)
28         timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
29
30         if humidity is not None and temperature_c is not None:
31             log_data = f"{timestamp} - Temperatura: {temperature_c:.1f}°C, Umidade: {humidity:.1f}%"
32             print(log_data)
33
34             os.makedirs(os.path.dirname(LOG_FILE_PATH), exist_ok=True)
35
36             with open(LOG_FILE_PATH, "a") as log_file:
37                 log_file.write(log_data + "\n")
38         else:
39             print(f"{timestamp} - Falha ao ler o sensor. Verifique a conexão.")
40
41         time.sleep(INTERVALO_LEITURA)
42
43     except RuntimeError as error:
44         print(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')} - Erro de leitura: {error}")
45         time.sleep(5)
46     except KeyboardInterrupt:
47         print("\nPrograma interrompido.")
48         break
49     except Exception as e:
50         print(f"Um erro inesperado ocorreu: {e}")
51         break
52
53 if __name__ == "__main__":
54     main()
55
56
57
58
59
Ln 55, Col 54  Espaços: 4  UTF-8  CRLF  {} Python  3.11.0  Go Live
```

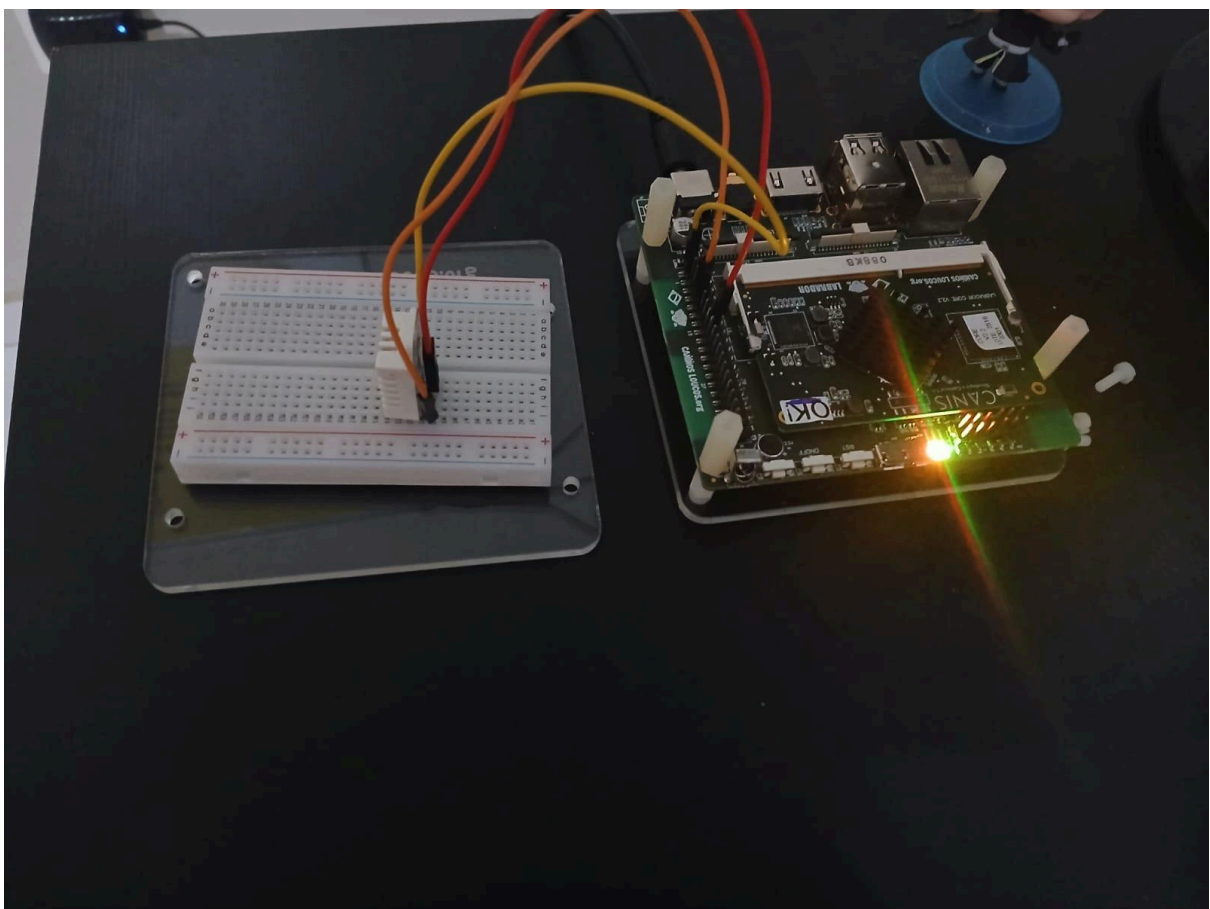
5. Desafios Enfrentados durante o Desenvolvimento

1. Incompatibilidade de Bibliotecas: O maior desafio foi a incompatibilidade da biblioteca de abstração **Adafruit-Blinka** com a placa Labrador. A biblioteca não conseguiu identificar o hardware, gerando um erro **NotImplementedError** que impediu o acesso inicial aos pinos GPIO.
2. Dependências de Compilação: A transição para uma biblioteca de mais baixo nível (**gpio-dht**) resolveu o problema de abstração, mas introduziu um novo desafio: a instalação da biblioteca falhou devido à ausência de ferramentas de compilação no sistema operacional. Foi necessário instalar pacotes como **build-essential** e **libgpio-dev** para que o **pip** pudesse compilar a biblioteca a partir do código-fonte.
3. Depuração de Ambiente: A necessidade de criar um ambiente virtual (**venv**) e instalar as dependências corretas dentro dele foi um passo crucial. Erros como **ModuleNotFoundError** ocorreram quando as bibliotecas foram instaladas no escopo global do sistema em vez de no ambiente isolado do projeto.

6. Anexo: Tabela Verdade da Porta AND (E)

Entrada A	Entrada B	Saída (A E B)
0	0	0
0	1	0
1	0	0
1	1	1

7. Imagens do projeto



[Link do repositório](#)