PROCESSAMENTO DE DADOS DE TEMPERATURA E UMIDADE COLETADOS E ARMAZENADOS UTILIZANDO ESP8266 NodeMCU

Trabalho de Conclusão de Curso

Engenharia da Computação

Camilla Silvestre Lino Silva
Orientador: Prof. José Paulo Gonçalves de Oliveira

Universidade de Pernambuco
Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação

CAMILLA SILVESTRE LINO SILVA

PROCESSAMENTO DE DADOS DE TEMPERATURA E UMIDADE COLETADOS E ARMAZENADOS UTILIZANDO ESP8266 NodeMCU

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia de Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, Junho 2022.

Silva, Camilla Silvestre Lino
Processamento de dados de temperatura e umidade coletados e armazenados utilizando ESP8266 NodeMCU / Camilla Silvestre Lino Silva. – Recife - PE, 2022.
xi, 39 f.

Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação) Universidade de Pernambuco, Escola Politécnica de Pernambuco, Recife, 2022.

Orientador: Prof MSc. José Paulo Gonçalves de Oliveira.

Inclui referências.

1. Processamento de dados. 2. MicroPython. 3. ESP8266 NodeMCU. I. Processamento de dados de temperatura e umidade coletados e armazenados utilizando ESP8266 NodeMCU. II. Oliveira, José Paulo Gonçalves de. III. Universidade de Pernambuco.

ESCOLA POLITÉCNICA
DE PERNAMBUCO

MONOGRAFIA DE FINAL DE CURSO

Avaliação Final (para o presidente da banca)*

No dia 24/5/2022, às 15h00min, reuniu-se para deliberar sobre a defesa da monografia de conclusão de curso do(a) discente **CAMILLA SILVESTRE LINO SILVA**, orientado(a) pelo(a) professor(a) **JOSÉ PAULO G. DE OLIVEIRA**, sob título PROCESSAMENTO DE DADOS DE TEMPERATURA E UMIDADE COLETADOS E ARMAZENADOS UTILIZANDO ESP8266 NodeMCU, a banca composta pelos professores:

EDISON DE QUEIROZ ALBUQUERQUE (PRESIDENTE)
JOSÉ PAULO G. DE OLIVEIRA (ORIENTADOR)

Após a apresentação da monografia e discussão entre os membros da Banca, a mesma foi considerada:

☒ Aprovada ☐ Aprovada com Restrições* ☐ Reprovada

e foi-lhe atribuída nota: 9,0 (NOVE)

*(Obrigatório o preenchimento do campo abaixo com comentários para o autor)

O(A) discente terá 15 dias para entrega da versão final da monografia a contar da data deste documento.

Edison de Queiroz Albuquerque

AVALIADOR 1: Prof (a) **EDISON DE QUEIROZ ALBUQUERQUE**

José Paulo G. de Oliveira.

AVALIADOR 2: Prof (a) **JOSÉ PAULO G. DE OLIVEIRA**

AVALIADOR 3: Prof (a)

* Este documento deverá ser encadernado juntamente com a monografia em versão final.

Dedico este trabalho aos meus pais, que acreditaram no meu potencial quando nem eu mesma acreditei.

Agradecimentos

Em primeiro lugar agradeço aos meus pais, que durante toda a minha vida se esforçaram para que eu chegasse onde estou hoje.

Às amizades construídas durante o curso e as pessoas por elas trazidas, que sempre iluminaram meu caminho durante toda minha jornada.

Ao professor José Paulo, que me ajudou durante todo o semestre com o desenvolvimento do trabalho de conclusão de curso, sendo sempre paciente e solícito.

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Autorização de publicação de PFC

Eu, **Camilla Silvestre Lino Silva** autor(a) do projeto de final de curso intitulado: **PROCESSAMENTO DE DADOS DE TEMPERATURA E UMIDADE COLETADOS E ARMAZENADOS UTILIZANDO ESP8266 NodeMCU;** autorizo a publicação de seu conteúdo na internet nos portais da Escola Politécnica de Pernambuco e Universidade de Pernambuco.

O conteúdo do projeto de final de curso é de responsabilidade do autor.

Camilla Silvestre L. Silva
Camilla Silvestre Lino Silva

José Paulo G. de Oliveira
Orientador(a): **José Paulo G. de Oliveira**

Coorientador(a):

Prof, de TCC: **Daniel Augusto Ribeiro Chaves** Data: 24/5/2022

Resumo

No âmbito da tecnologia existe um rápido crescimento no uso de data centers, tanto em questão de quantidade quanto em tamanho, consumindo assim mais energia e dissipando mais calor [5] de forma que o acompanhamento desses ambientes também vem sendo necessário para seu melhor funcionamento. Com o crescente desenvolvimento de novas tecnologias, o monitoramento destes ambientes vem se mostrando cada vez mais acessível já que, com a popularização do acesso aos serviços de banda larga, o IoT (*Internet of Things*) se tornou vastamente conhecido pelo fato de facilitar tarefas do dia a dia com a utilização da internet para otimizar a manipulação de uma gama de objetos físicos. Este trabalho tem como objetivo o desenvolvimento de uma solução IoT que seja capaz de monitorar e armazenar dados de temperatura e umidade de um ambiente, juntamente com um serviço que possa processar estes dados, otimizando-os para que possa ser feita uma melhor análise da área monitorada pela solução.

Abstract

In the technology field there is a rapid growth of data center use has increased both their size and number, thereby dissipating more heat and consuming a larger amount of power [5] so that the monitoring of these environments has also been necessary for its better functioning. With the growing development of new technologies, the monitoring of these environments is proving to be increasingly accessible since, with the popularization of the access to broadband services, IoT (*Internet of Things*) has become widely known for the fact that it facilitates day-to-day tasks with the use of the internet to optimize the way to manipulate a range of physical objects. This work aims to develop an IoT solution capable of monitoring and storing temperature and humidity data from an environment, along with a service that can process these data, optimizing them so that a better analysis of the monitored area can be made by the solution.

Lista de Figuras

Lista de Símbolos e/ou Siglas

API - Application Programming Interface

FIFO – First In First Out

GPIO – General Purpose Input/Output

IoT – Internet of Things

JSON – JavaScript Object Notation

REST – Representational State Transfer

SQL - Structured Query Language

SSID – Service Set Identifier

Sumário

Capítulo 1

Introdução

Com a popularização do acesso aos serviços de banda larga, a IoT(*Internet of Things*) se tornou bastante conhecida pelo fato de facilitar tarefas do dia a dia com a utilização da Internet para otimizar a manipulação de uma gama de objetos físicos. A partir do momento que esses dispositivos estão conectados com o objetivo específico de coletar ou exibir dados, obtém-se o que é chamado de solução IoT, sendo essa “um conjunto de dispositivos projetados para produzir, consumir ou apresentar informações sobre algum evento ou séries de eventos e observações.” [1]. Dentre os diversos dados que podem ser produzidos e apresentados estão os referentes a temperatura e umidade de um ambiente, comumente obtidos através da utilização de sensores próprios para tal coleta como os sensores DHT11, DHT22 e até o BMP180 que também é capaz de coletar dados de pressão atmosférica.

Com o intuito de aumentar a quantidade de soluções IoT alguns microcontroladores, algumas vezes chamados de módulos, foram desenvolvidos. Um deles é o ESP8266, um microcontrolador que possui diversos pinos para que seja possível criar uma conexão com outros microcontroladores ou sensores utilizando jumpers, por exemplo. Ele também possui um sistema interno que permite a coleta de dados atráves destas conexões, existindo ainda a possibilidade de enviar informações através do protocolo TCP/IP, já que também é possível o conectar a uma rede Wi-Fi. Outro módulo é o ESP32, que é a versão sucessora do ESP8266, possuindo tudo que seu predecessor possui juntamente com um chip a mais que permite a conexão *bluetooth*.

Comumente estes microcontroladores são utilizados em pequenos projetos IoT com Arduino, mas com o decorrer dos anos outras tecnologias foram desenvolvidas para que sua utilização fosse mais fácil. Permitindo que desenvolvedores habituados com a linguagem Python pudessem criar seus próprios projetos IoT. Sabendo dessas informações é possível notar que, com algum

conhecimento de componentes eletrônicos e linguagens de programação, pode-se criar uma solução IoT capaz de monitorar um ambiente.

1.1 Objetivos do projeto

Este trabalho tem o objetivo de desenvolver um *data logger* para aplicações IoT. Inicialmente, a solução deve ser capaz de monitorar e armazenar dados de temperatura e umidade de um ambiente. Novas grandezas podem ser adicionadas ao sistema de forma simples, de acordo com a necessidade da aplicação final. Para desenvolver a solução proposta, se faz necessário o desenvolvimento de um sistema conectado a Internet, formado por um módulo sensor e uma aplicação local. Para o módulo sensor, utilizamos a placa ESP8266 NodeMCU para coletar e transferir as informações obtidas para um banco de dados, no módulo local. A linguagem utilizada para desenvolver o *firmware* da placa foi o MicroPython. A aplicação local foi feita em Python. Sua finalidade é processar os dados enviados de forma que as informações possam ser melhor aproveitadas após processadas. O processamento depende da finalidade específica e pode ser, por exemplo, monitorar e controlar as condições ambientais na indústria agropecuária.

1.2 Metodologia

1.2.1 Materiais e Métodos

Para o desenvolvimento deste projeto foram utilizados: um ESP8266 NodeMCU, uma *protoboard*, um sensor DHT11 e *jumpers* macho-macho e fêmea-macho. Além disso também foi necessária a criação de um banco de dados MySQL para o armazenamento das informações e uma API (*Application Programming Interface*) para que fosse possível fazer um POST para o banco de dados utilizando o ESP8266.

Após a montagem do circuito, a coleta dos dados de umidade e temperatura foi realizada utilizando as funções presentes na biblioteca *machine* da linguagem MicroPython. Para que o envio desses dados para a API – utilizando as bibliotecas

network e *urequests* – fosse possível, foi necessária a conversão dessas informações para o formato JSON (*JavaScript Object Notation*), que foi desenvolvida utilizando a função *dumps* disponível na biblioteca *urequests*. A API criada possui apenas a função *create* que é responsável pelo envio dos dados por ela recebidos para o banco de dados, que possui apenas 3 campos: *humidity, temperature* e *date_time.* O último campo é preenchido automaticamente com o horário em que as informações foram adicionadas. Por fim, para que fosse possível uma melhor análise dos dados, um código em Python foi desenvolvido utilizando a biblioteca *mysql* para coletar os dados do banco e exibi-los em gráficos que foram criados utilizando as funções da biblioteca *matplotlib*.

1.2.2 Resultados e Impactos Esperados

A principal contribuição da coleta de dados em MicroPython é apresentar uma forma menos conhecida de utilizar a placa ESP8266, que usualmente é utilizada em Arduino, apresentando uma forma mais didática para desenvolvedores que já possuem experiência com a linguagem Python.

Ao final do projeto, atingimos os objetivos exemplificados na Figura 1, que são: a criação do circuito; o desenvolvimento do código em MicroPython que controla o ESP8266 (este código é responsável por utilizar de suas bibliotecas internas para obter os dados de umidade e temperatura do ambiente); conectar a placa a uma rede de internet e enviar os dados coletados para uma API (esta faz a inserção dessas informações no banco de dados); por fim, também foi desenvolvido um código na linguagem Python para o processamento das informações contidas no banco (utilizando apenas as informações coletadas durante um certo período de tempo).

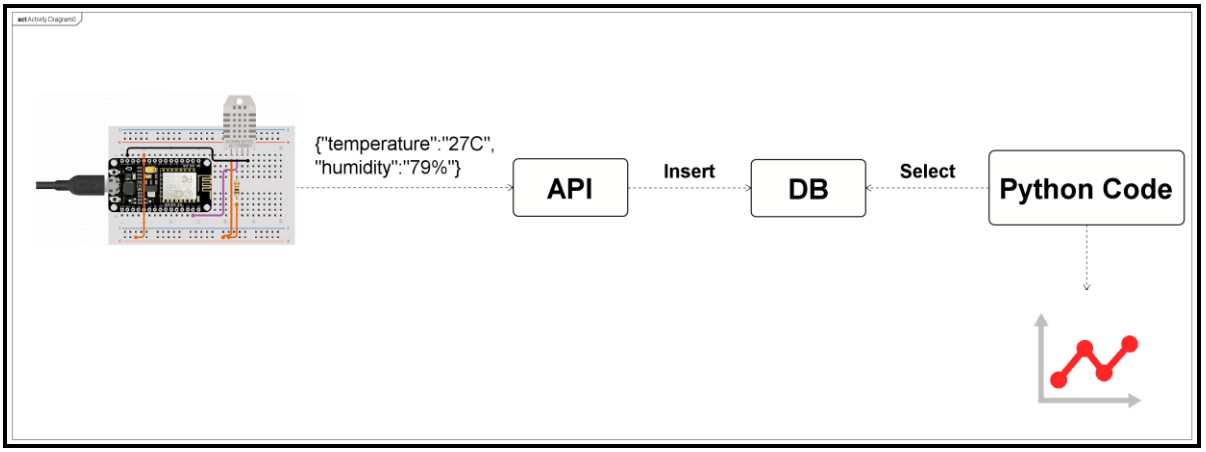

Figura 1. Exemplo do funcionamento geral do projeto
Fonte: Própria autora

O processamento é feito em tempo real, de forma que o código Python gera gráficos de “Temperatura x Tempo” e “Umidade x Tempo” exibindo as variações que podem ter ocorrido no decorrer de um certo espaço de tempo.

1.2.3 Organização do Trabalho

Este trabalho está organizado da forma descrita a seguir. No capítulo 2, encontram-se as informações da fundamentação teórica, que tem o objetivo de explicar as ferramentas e dispositivos do trabalho. O capítulo 3 apresenta tanto a construção do circuito, quanto o desenvolvimento dos códigos em MicroPython e Python. E, por último, o capítulo 4 contém as conclusões do trabalho.

Capítulo 2

Fundamentação Teórica

2.1 ESP8266

O ESP8266 é um microcontrolador projetado pela Espressif Systems que possui Wi-Fi e uma pilha TCP/IP completa, compactada em um pequeno circuito integrado. A versão utilizada neste projeto é a ESP8266 NodeMCU, que traz consigo um *firmware* baseado na linguagem Lua e possui uma entrada micro-USB que, ao se conectar no computador, é reconhecida automaticamente como uma conexão serial. O fato deste microcontrolador possuir uma entrada USB facilita o entendimento da comunicação entre código e placa. E simplifica o desenvolvimento do código para desenvolvedores que possuem menos experiência com Arduino ou MicroPython.

2.2 MicroPython

Apesar do ESP8266 ser comumente utilizado com Arduino, desde 2014 muito esforço tem sido colocado para fazer com que este dispositivo seja compatível com MicroPython. Essa linguagem nada mais é do que uma reimplementação da linguagem Python que visa a microcontroladores e sistemas embarcados [4], sendo então uma opção mais didática para programadores que já possuem experiência com a linguagem Python. O MicroPython possui bibliotecas internas que possibilitam uma fácil conexão da placa à internet e a coleta de dados através de outros sensores ligados a seus pinos, fazendo com que seja possível o envio de tais dados pela internet.

2.2.1 Conexão á internet

Essa linguagem possui diversas bibliotecas, também chamadas de módulos, que servem como suporte para o projeto. A biblioteca *network* permite que o ESP8266 se conecte à uma rede Wi-Fi de forma simples, necessitando apenas do

SSID (*Service Set Identifier*) e da senha da rede na qual vai ser conectada. Para melhor uso das funcionalidades desse módulo, dentro da função *do_connect* (Figura 2) é feita uma verificação de que o microcontrolador realmente está conectado e, após isso, já é possível fazer o envio de dados.

```
19  #Function to connect to networks
20  def do_connect():
21      sta_if = network.WLAN(network.STA_IF)
22      if not sta_if.isconnected():
23          print('connecting to network...')
24          sta_if.active(True)
25          sta_if.connect('NETSETUP-Camilla[illegible]', '0612[illegible]9')
26          while not sta_if.isconnected():
27              print("Not Connected")
28      print('network connected. config:', sta_if.ifconfig())
29
30  do_connect()
```

Figura 2. Função de conexão à internet

Fonte: Própria autora

2.2.2 Coleta de dados

A biblioteca *machine* faz com que seja possível coletar dados recebidos nos pinos do microcontrolador. No caso deste projeto, é utilizado o GPIO14 (*General Purpose Input/Output*) que é um dos pinos destinados a entrada e saída genérica digital. Ligado a esse pino está conectado o DHT11, um sensor de temperatura e umidade que, dentro do MicroPython, é manipulado pela biblioteca *dht.* Essa biblioteca possui a função *measure* responsável por medir os dados e armazená-los nas funções internas *temperature* e *humidity,* que neste projeto são vinculados a variáveis internas (Figura 3).

```
32    sensor = dht.DHT11(machine.Pin(14))
33    #Temperature and upload using HTTP Post
34    while True:
35        try:
36          sensor.measure()
37          temp = sensor.temperature()
38          hum = sensor.humidity()
39          print('Temperatura: ' + str(temp) + 'C')
40          print('Umidade: ' + str(hum) + '%')
41          temp = str(temp) + 'C'
42          hum = str(hum) + '%'
```

Figura 3. Coleta de dados de temperatura e umidade

Fonte: Própria autora

2.2.3 Envio de dados

O módulo *urequests* possui funções que fazem com que seja possível a execução de requisições HTTP, como POST e GET, por exemplo. Necessitando apenas do *link* para onde a requisição será enviada e da mensagem que será enviada (Figura 4). Já a biblioteca *ujson* permite que os parâmetros enviados dentro da função *dumps* sejam convertidos para o formato JSON, que é o formato aceito pela API desenvolvida.

```
43          msg = ujson.dumps({'temperature': temp, 'humidity': hum})
44          print(msg)
45          try:
46            response = requests.post(api_url, data= msg, headers=headers)
47            print(response.json())
48            response.close()
49          except OSError as e:
50            print(e)
```

Figura 4. Envio de dados via HTTP POST

Fonte: Própria autora

2.3 API RESTful

Uma transferência de dados representativa, REST (*Representational State Transfer*), define um conjunto de padrões para serviços da *web*. Uma API é uma interface que os programas de *software* usam para se comunicar entre si [3]. Então uma API RESTful nada mais é do que uma API em conformidade com a arquitetura REST. Esse tipo de API comumente utiliza as requisições HTTP mais comuns *GET*, *POST*, *PUT* e *DELETE* que, quando dentro da API desenvolvida utilizando Node.js, se tornam *Read, Create, Update* e *Delete*, respectivamente. Neste projeto, apenas o desenvolvimento da função *create* (Figura 5) se mostrou necessário, já que a API serve apenas como um intermediário entre o microcontrolador e o banco de dados. Essa função utiliza a conexão da API com o banco de dados para inserir as informações recebidas na tabela criada, através de uma query *INSERT*.

```
4   async function create(logInformation){
5     const result = await db.query(
6       `INSERT INTO data_logs
7       (temperature, humidity)
8       VALUES ("${logInformation.temperature}", "${logInformation.humidity}");`
9     );
10
11    let message = 'Error adding logs';
12
13    if (result.affectedRows) {
14      message = 'Logs uploaded';
15    }
16
17    return {message};
18  }
```

Figura 5. Função create da API

Fonte: Própria autora

2.4 Banco de Dados

Um banco de dados relacional é um *software* que permite a implementação de uma base dados com tabelas, colunas e índices. Ele não garante apenas uma integridade referencial entre as linhas de diversas tabelas mas também tem a capacidade de atualizar os índices automaticamente. Também consegue interpretar

qualquer linguagem do tipo SQL (*Structured Query Language*) e combinar informações de várias fontes [6]. Um dos bancos de dados relacionais *open-source* mais conhecidos é o MySQL, que é utilizado neste projeto para arquivar as informações coletadas pelo circuito. A criação da tabela *data_logs* (Figura 6) foi desenvolvida em sua forma mais básica criando apenas as variáveis necessárias para que seja possível arquivar temperatura, umidade e o horário que os dados foram coletados. Além do armazenamento dos dados, a criação de alguns eventos se mostrou necessária para o melhor aproveitamento da base de dados.

```
1  create table `data_logs`(
2      temperature varchar(10) NOT NULL,
3      humidity varchar(10) NOT NULL,
4      date_time datetime NOT NULL DEFAULT CURRENT_TIMESTAMP
5  );
```

Figura 6. Criação da tabela *data_logs*
Fonte: Própria autora

2.4.1 *Procedure*

Uma rotina armazenada é o nome dado a uma declaração em SQL que se localiza no servidor, para que não haja a necessidade do usuário executar declarações individuais quando podem acessar a rotina. Um dos tipos mais comuns de rotina são as *procedures,* que podem alterar valores utilizando saídas ou resultados definidos em suas declarações [6]. Para uma *procedure* ser executada ela tem que ser chamada pela função *call*. Por este projeto ser desenvolvido em um computador pessoal e, consequentemente, possuir uma limitação de memória para armazenamento, a criação de uma *procedure* se mostrou necessária. A rotina *delete_first* (Figura 7) faz com que a tabela apague o primeiro registro sempre que a tabela atinge um certo volume de dados, aplicando o algoritmo FIFO (*First In First Out*).

```
11      DELIMITER //
12  •   CREATE PROCEDURE `delete_first` ()
13      BEGIN
14          DECLARE total_rows int;
15          DECLARE first_row_date DATE;
16          SET SQL_SAFE_UPDATES = 0;
17          SET total_rows = (SELECT Count(*) FROM datalogger.data_logs);
18          SET first_row_date = (SELECT date_time from datalogger.data_logs LIMIT 1);
19          IF total_rows >= 1000
20              THEN DELETE FROM datalogger.data_logs WHERE date_time = first_row_date;
21          else
22              SET total_rows = total_rows;
23      END IF;
24      END//
25      DELIMITER ;
```

Figura 7. Procedure *delete_first*

Fonte: Própria autora

2.4.2 Evento

Em MySQL, eventos são tarefas executadas de acordo com um "agendamento", ou *schedule*. Para este projeto, foi feita a criação de um evento chamado *call_delete* (Figura 8) que a cada 50 segundos executa a *procedure delete_first*.

```
29  •   CREATE EVENT call_delete
30          ON SCHEDULE EVERY 50 SECOND
31          DO CALL delete_first();
```

Figura 8. Evento *call_delete*

Fonte: Própria autora

Capítulo 3

Funcionamento

Neste capitulo, estão detalhadas todas as etapas do desenvolvimento do projeto, desde a montagem do circuito até a análise dos gráficos gerados utilizando a linguagem Python.

3.1 Circuito

Todos os dados apresentados anteriormente, e também aqueles exibidos nos próximos tópicos, apenas são úteis quando o circuito exibido na Figura 9 está corretamente montado. Para o desenvolvimento deste circuito, foi necessário entender o ESP8266 NodeMCU e como suas entradas e saídas se conectam ao DHT11. Também foi preciso entender como funcionam as conexões do sensor para que nenhuma conexão fosse feita de maneira errônea e este fosse perdido. Após a montagem do circuito, é possível conectar o microcontrolador diretamente ao computador, que o reconhece como uma conexão serial (COM).

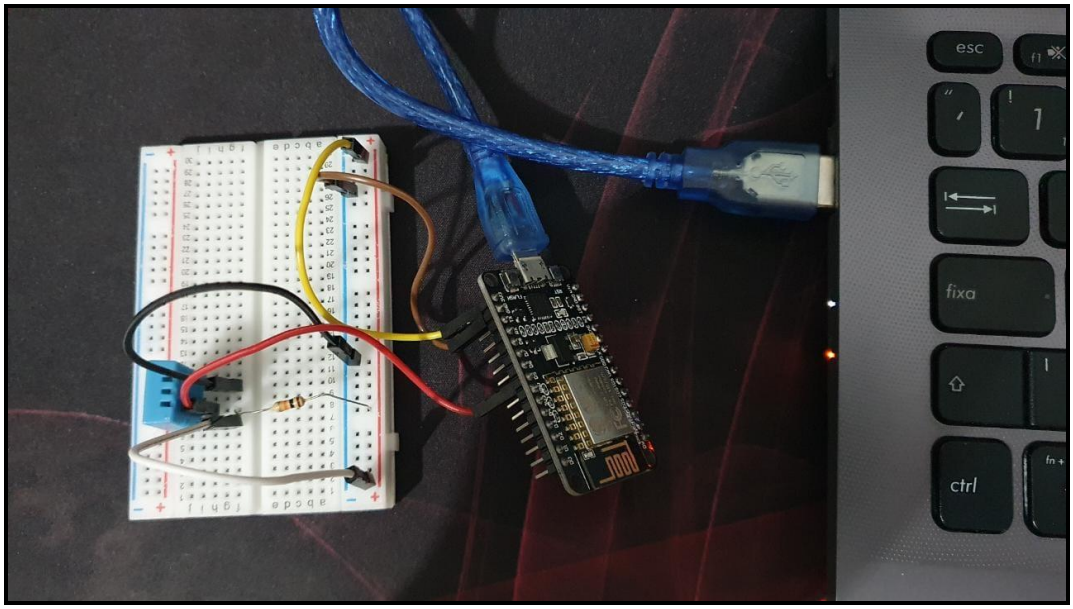

Figura 9. Circuito conectado ao computador

Fonte: Própria autora

3.2 API

Antes que o código em MicroPython seja executado, é necessário iniciar o serviço da API já que esse é o “conector” do circuito com o banco de dados. Caso o serviço tenha sido corretamente desenvolvido e iniciado, uma mensagem de sucesso contendo o link da API é exibida no terminal onde esta foi executada (Figura 10)

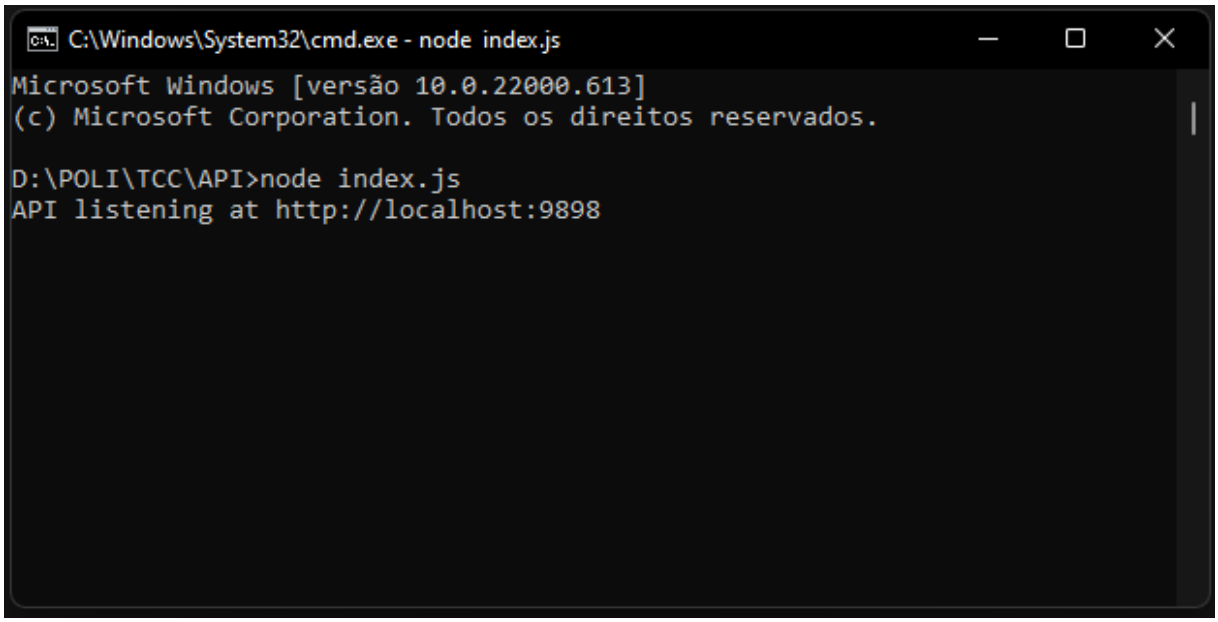

Figura 10. Mensagem exibida quando o serviço da API é iniciado

Fonte: Própria autora

3.3 Dados

Após a montagem do circuito e a execução da API, é possível executar o código em MicroPython. Após o início da execução, as informações de conexão com a internet, os dados de temperatura e umidade, e a mensagem de sucesso no envio para o banco de dados são exibidos no terminal com periodicidade de 1 minuto (Figura 11).

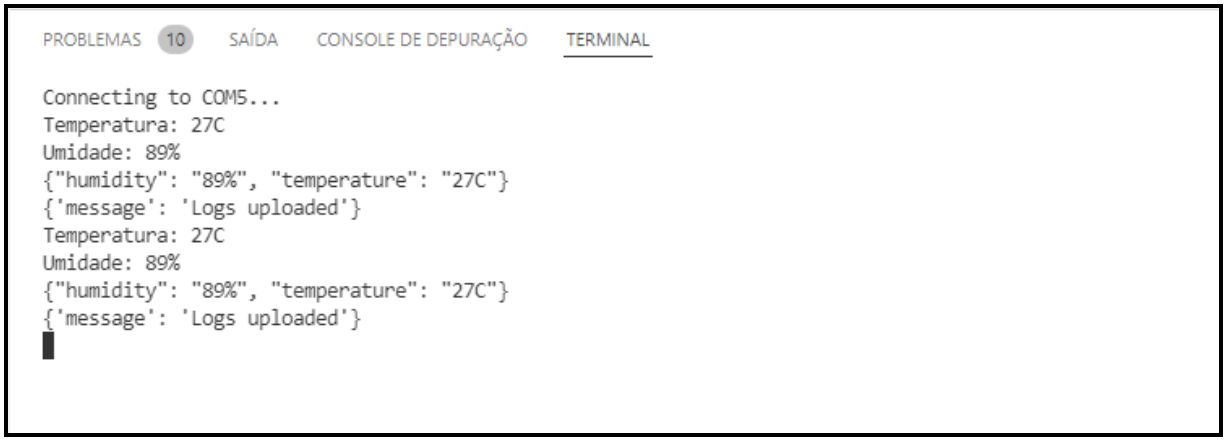

Figura 11. Mensagens do terminal durante a execução do código em MicroPython

Fonte: Própria autora

3.4 Banco de dados

No tópico anterior, é possível verificar que apenas dois dados são enviados para o banco de dados, mas como mencionado no tópico 2.4 esta tabela possui três campos: *temperature, humidity* e *date_time*. Devido ao fato de a linguagem MicroPython possuir recursos reduzidos, não foi possível fazer com que o horário fosse inserido como um dado, já que esta linguagem não possui nenhuma biblioteca que acesse estas informações. Para que este problema fosse contornado, a tabela foi criada de uma forma que o MySQL adicionasse o campo *date_time* com o horário em que os outros dois dados foram recebidos, como é possível ver na Figura 12.

temperature	humidity	date_time
27C	89%	2022-05-03 21:48:38
27C	89%	2022-05-03 21:49:39
26C	89%	2022-05-03 21:50:39
26C	89%	2022-05-03 21:51:40
26C	90%	2022-05-03 21:52:41
26C	91%	2022-05-03 21:53:41
26C	90%	2022-05-03 21:54:42
26C	90%	2022-05-03 21:55:43
27C	89%	2022-05-03 21:56:43

Figura 12. Dados inseridos no banco de dados

Fonte: Própria autora

3.5 Gráfico

Quando todas as funcionalidades nos tópicos anteriores estão funcionando em sincronia, é possível iniciar a execução do código em Python que, de forma similar ao MicroPython, atualiza a cada um minuto com uma nova informação dos dados coletados. As informações são apresentadas em dois gráficos separados que são continuamente atualizados e exibidos até que a execução seja interrompida. Para melhor visualização, os gráficos mostram apenas os dados dos últimos 10 minutos, como é possível observar nas Figuras 13 e 14.

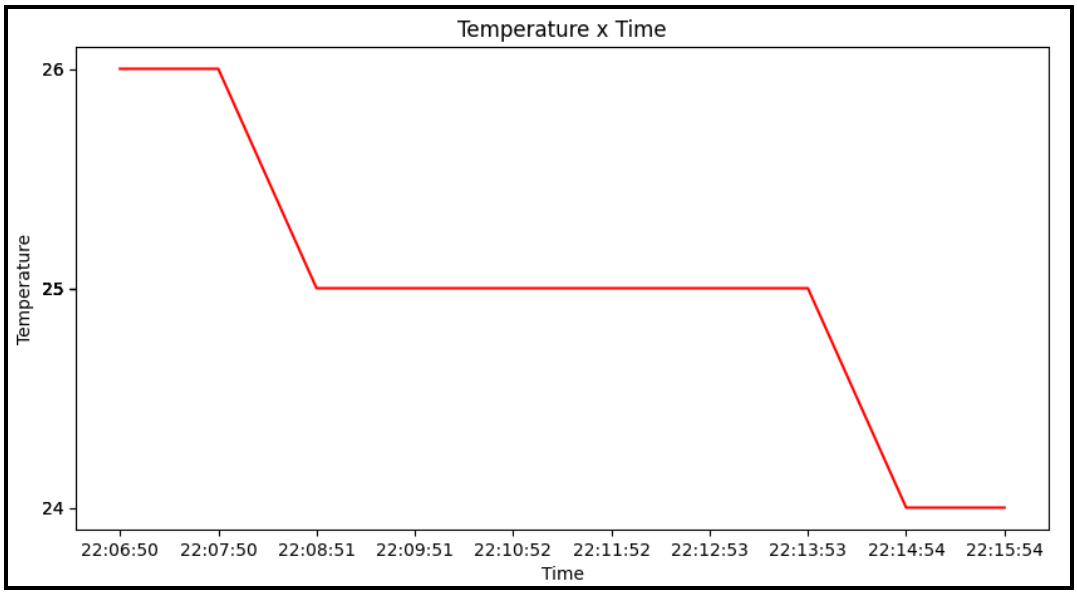

Figura 13. Gráfico demonstrando variação de temperatura ao decorrer de 10 minutos

Fonte: Própria autora

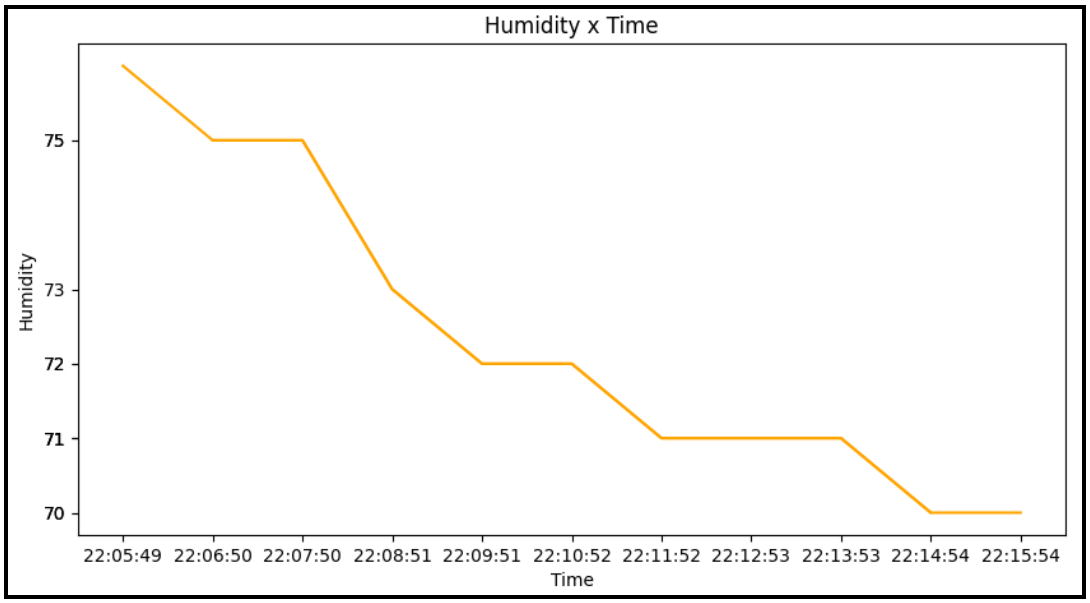

Figura 14. Gráfico demonstrando variação de umidade ao decorrer de 10 minutos

Fonte: Própria autora

Capítulo 4

Conclusão

O desenvolvimento deste projeto permitiu mostrar que, apesar da existência de algumas limitações na linguagem MicroPython (que não apresenta diversas bibliotecas presentes na linguagem Python, como as específicas para inserção de dados diretamente no banco), é possível utilizar o ESP8266 sem a necessidade do Arduino.

Também foi possível perceber que, com algum conhecimento sobre outras ferramentas, é possível fazer todo o caminho demonstrado na Figura 1 aproveitando todas as informações coletadas para uma análise. Apesar de apenas um gráfico ter sido criado para este projeto, fica claro que todas as ferramentas utilizadas são escaláveis. Ou seja, com a utilização de outros sensores e um circuito mais robusto uma maior análise de diversos dados também poderia ser executada.

Por fim, destacamos que a implementação de um *data logger* experimental e de baixo custo pode ser demonstrada. Para aplicações do IoT, esses dispositivos são imprescindíveis e muitas vezes possuem custos elevados. Além disso, a aquisição remota de dados viabiliza o processamento de informações com finalidades diversas, o que torna a solução proposta bastante atrativa para aplicações reais.

Referências

[1] BELL, Charles. **MicroPython for the Internet of Things: A Begginer's Guide to Programming with Python on Microcontrollers.** Apress, 2017, p 33.

[2] GEORGE, Damien P. *et al*. **MicroPython v1.18 Documentation.** MicroPython, 2022. Disponível em: https://docs.micropython.org. Acesso em: 20 de abr. de 2022.

[3] RASCIA, Tania. **Node.js, Express.js, and PostgreSQL: CRUD REST API example.** LogRocket, 2020. Disponível em: https://blog.logrocket.com/nodejs-expressjs-postgresql-crud-rest-api-example. Acesso em: 23 de abr.de 2022.

[4] TOLLERVEY, Nicholas H. **Programming with MicroPython: Embedded Programming with microcrontrolers and Python.** O'Reilly Media Inc, 2018, p 40.

[5] TRADATA, Mohammad I. *et al*. **Impact of elevated temperature on data center operation based on internal and external IT instrumentation***. 33rd Thermal Measurement, Modeling & Management Symposium (SEMI-THERM),* 2017, p. 108.

[6] UZAYR, Sufyan bin. **Mastering MySQL for Web: A Begginer's Guide.** CRC Press, 2022, p. 1 e 118-119.

Anexos

Anexo A – Código em MicroPython

Abaixo encontra-se o código em MicroPython para que outros desenvolvedores possam utilizá-lo.

```
import machine
from time import sleep
import network
import dht
import ujson
import urequests as requests

gc.collect()
gc.threshold(gc.mem_free() // 4 + gc.mem_alloc())
gc.collect()

pin = machine.Pin(5, machine.Pin.OUT)
sensor = dht.DHT11(machine.Pin(14))

api_url = 'http://192.168.0.17:9898/add'
headers = {'Content-Type': 'application/json'}

#Function to connect to networks
def do_connect():
     sta_if = network.WLAN(network.STA_IF)
     if not sta_if.isconnected():
         print('connecting to network...')
         sta_if.active(True)
         sta_if.connect('NETSETUP-CamillaeFatima-2.4G', '061297100369')
         while not sta_if.isconnected():
             print("Not Connected")
```

```
        print('network connected. config:', sta_if.ifconfig())

do_connect()

sensor = dht.DHT11(machine.Pin(14))
#Temperature and upload using HTTP Post
while True:
    try:
      sensor.measure()
      temp = sensor.temperature()
      hum = sensor.humidity()
      print('Temperatura: ' + str(temp) + 'C')
      print('Umidade: ' + str(hum) + '%')
      temp = str(temp) + 'C'
      hum = str(hum) + '%'
      msg = ujson.dumps({'temperature': temp, 'humidity': hum})
      print(msg)
      try:
        response = requests.post(api_url, data= msg, headers=headers)
        print(response.json())
        response.close()
      except OSError as e:
        print(e)
      sleep(60)
    except OSError as e:
      print('Failed to read sensor.')
```

Anexo B – Código da API em Node.js

Abaixo encontra-se o código da API em Node.js para que outros desenvolvedores possam utilizá-lo.

```
const mysql = require('mysql2/promise');
const config = require('../config');

async function query(sql, params) {
  const connection = await mysql.createConnection(config.database);
  const [results, ] = await connection.execute(sql, params);

  return results;
}

module.exports = {
  query
}
```

Figura 15. Código database.js

Fonte: Própria autora

```
const data_credentials = require("../credentials.json")

const config = {
    database: data_credentials,
    listPerPage: 10,
  };
  module.exports = config;
```

Figura 16. Código config.js

Fonte: Própria autora

```
const express = require('express');
const router = express.Router();
const addLogs = require('../services/addLogs');

router.post('/', async function(req, res, next) {
    try {
      res.json(await addLogs.create(req.body));
    } catch (err) {
      console.error(`Error adding logs`, err.message);
      next(err);
    }
  });

module.exports = router;
```

Figura 17. Código addLogs.js

Fonte: Própria autora

```
const express = require("express");
const app = express();
const port = 9898;
const addLogsRouter = require('./routes/addLogs');


app.use(express.json());

app.use(
  express.urlencoded({
    extended: true,
  })
);

app.get("/", (req, res) => {
  res.json({ message: "ok" });
});

app.use("/add", addLogsRouter);

/* Error handler middleware */

app.use((err, req, res, next) => {
  const statusCode = err.statusCode || 500;
  console.error(err.message, err.stack);
  res.status(statusCode).json({ message: err.message });
  return;
});
app.listen(port, () => {
  console.log(`API listening at http://localhost:${port}`);
});
```

Figura 18. Código index.js

Fonte: Própria autora

Anexo C – Código em Python

Abaixo encontra-se o código em Python para que outros desenvolvedores possam utilizá-lo.

```
from pickle import FALSE
from ssl import OPENSSL_VERSION
from time import sleep, time
```

```
import mysql.connector
from mysql.connector import Error
from datetime import datetime, timedelta
import numpy as np
import matplotlib.pyplot as plt
import json

with open("../credentials.json") as credentialsFile:
    credentialsData = json.load(credentialsFile)
    credentialsFile.close()

def getDatabaseInfo():
    try:
        conn = mysql.connector.connect(host= credentialsData['host'],
        port= credentialsData['port'],
        user= credentialsData['user'],
        password= credentialsData['password'],
        database= credentialsData['database'])

        if conn.is_connected():
            query = "SELECT * from (SELECT * from data_logs  WHERE
DATE(date_time) = CURDATE() ORDER BY date_time DESC LIMIT 10)var2 ORDER BY
date_time ASC;"
            cursor = conn.cursor()
            cursor.execute(query)
            data = cursor.fetchall()
            total_data = cursor.rowcount
            return data
        cursor.close()
        conn.close()
    except Error as msg:
        print(msg)

def update_plot():
    data = getDatabaseInfo()
    temperature = []
    humidity = []
    time_date = []
    for row in data:
        temperature.append(int(row[0].replace("C", "")))
        humidity.append(int(row[1].replace("%","")))
        time_date.append(row[2].strftime("%H:%M:%S"))
    return temperature, humidity, time_date

plt.ion()
plt.show()

while True:
```

```
    temperature, humidity, time_date = update_plot()

    plt.figure(1, figsize=(10,5))
    plt.xlabel("Time")
    plt.ylabel("Temperature")
    plt.yticks(temperature)
    plt.plot(time_date, temperature,color="red")
    plt.title("Temperature x Time")

    plt.figure(2, figsize=(10,5))
    plt.plot(time_date, humidity, color="orange")
    plt.xlabel("Time")
    plt.ylabel("Humidity")
    plt.yticks(humidity)
    plt.title("Humidity x Time")
    plt.draw()
    plt.pause(10)
    plt.clf()
    plt.figure(1).clear()
```

Anexo D – Circuito

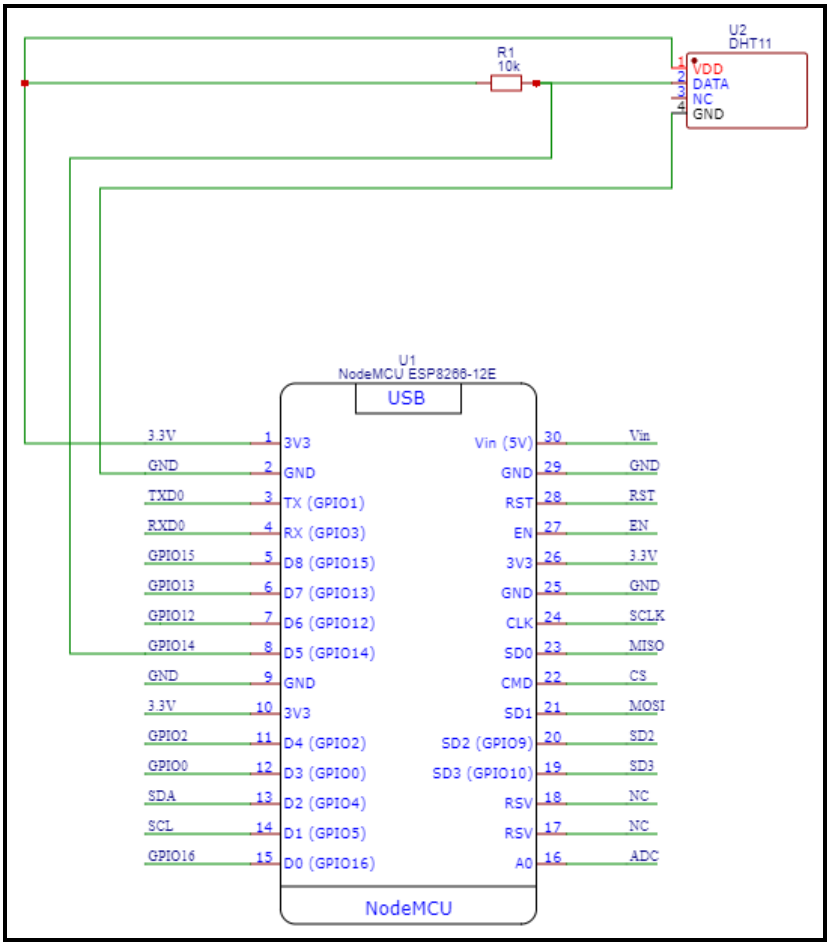

Figura 19. Simulação do circuito

Fonte: Própria autora