Universidade Federal de Campina Grande - UFCG
Centro de Engenharia Elétrica e Informática - CEEI
Unidade Academica de Engenharia Elétrica - UAEE

Maria Priscilla Lima Medeiros

Trabalho de Conclusão de Curso
Protótipo de sistema de monitoramento de temperatura e umidade para *data centers*

Campina Grande, Brasil
8 de abril de 2022

Maria Priscilla Lima Medeiros

Trabalho de Conclusão de Curso
Protótipo de sistema de monitoramento de temperatura e umidade para *data centers*

Trabalho de conclusão de curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Áreas de Concentração: Instrumentação Eletrônica

Orientador: Jaidilson Jó da Silva

Campina Grande, Brasil
8 de abril de 2022

Maria Priscilla Lima Medeiros

Trabalho de Conclusão de Curso
Protótipo de sistema de monitoramento de temperatura e umidade para *data centers*

Trabalho de conclusão de curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Aprovado em: 07/04/2022

Jaidilson Jó da Silva
Orientador

Gutemberg Gonçalves dos Santos Júnior
Avaliador

Campina Grande, Brasil
8 de abril de 2022

Dedico este trabalho à meus pais, Dorgival e Lourdes, por nunca terem medido esforços para que este grande dia pudesse chegar.

Agradecimentos

Agradeço primeiramente a Deus, pela minha vida, por me dar forças para ultrapassar todos os obstáculos e por ter me dado saúde e coragem para que eu pudesse concluir este trabalho.

Agradeço a meus pais, Lourdes e Dorgival, por sempre acreditarem em mim e nunca medirem esforços para realizar meus sonhos. Obrigada por nunca terem me deixado desistir. Saibam que foi o exemplo de garra e coragem de vocês que me deu forças para continuar no meio de tantas adversidades. Agradeço a meu irmão Pablo, por ser a minha representação de família aqui em Campina Grande.

Agradeço ao meu amor, parceiro do curso e da vida, Aldemaro, por todo seu cuidado, carinho, paciência, apoio e compreensão em dias difíceis. Obrigada por me manter centrada e por nunca medir esforços para me ajudar a conquistar meus objetivos.

Agradeço à meus amigos e colegas de curso. Em especial, agradeço àqueles que estiveram mais próximos e compartilharam de todas as dores e dificuldades encontradas, à quem dedico também as minhas conquistas: Alison, Guerra, Bruno e Kaio. Agradeço também a meus amigos amados e vizinhos Ícaro e Anna Luísa, por sempre vibrarem com minhas conquistas e por estarem tão presentes, me apoiando e me ajudando no que fosse necessário. Amo todos vocês.

Agradeço à empresa ISPTec Telecomunicações, na pessoa de Rodrigo Abrantes, por ter aberto as portas de suas instalações físicas para que eu pudesse realizar os testes e obter os resultados descritos neste trabalho.

Agradeço também ao professor Jaidilson Jó da Silva, por ter aceito o convite de me orientar neste trabalho de conclusão de curso. Obrigada por toda paciência, compreensão. Seus ensinamentos, foram muitos importantes para que eu pudesse obter êxito neste trabalho.

Por fim, agradeço a todos que, direta ou indiretamente me ajudaram e fizeram parte dessa minha jornada acadêmica até aqui.

"Existem coisas melhores
adiante do que qualquer outra
que deixamos para trás."
C. S. Lewis

Resumo

A quantidade de dados trafegados na internet aumenta cada dia mais e, com isso, aumenta também a necessidade de se assegurar que os chamados *Data Centers*, ou CPD (Centro de Processamento de dados), mantenham uma alta disponibilidade de dados e segurança nos equipamentos que processam todas essas informações. Para isso, é fundamental que o ambiente o qual os servidores estão instalados seja monitorado e controlado a fim que variáveis como temperatura e umidade não venham a danificá-los. Neste trabalho foi proposto um protótipo de um sistema de monitoramento de temperatura e umidade que pode ser anexado a sistemas de monitoramento já existentes no mercado com o uso do protocolo SNMP e também consta com sua própria interface web de monitoramento. O projeto foi desenvolvido utilizando o sensor de temperatura e umidade DHT22 e foi testado em um *server room* de um *data center* em Campina Grande-PB.

Palavras-chaves: *Data center*, Temperatura, Umidade, Monitoramento.

Abstract

The amount of data transmitted on the internet increases every day and, with this, the need to ensure that the so-called Data Centers, or DPC (Data Processing Center), maintain high availability and security within the equipments that process all this information. For this, it is essential that the environment in which the servers are installed is monitored and controlled so that variables such as temperature and humidity do not damage them. In this work, a prototype of a temperature and humidity monitoring system was proposed that can be attached to existing monitoring systems on the market using the SNMP protocol and also has its own monitoring web interface. The project was developed using the DHT22 temperature and humidity sensor and was tested in a server room of a data center in Campina Grande, PB, Brazil.

Key-words: Data center, Temperature, Humidity, Monitoring.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

SNMP	Simple Network Management Protocol
SGMP	Simple Gateway Monitoring Protocol
MIB	Management Information Base
OID	Object Identifier
HTML	Hypertext Markup Language
API	Application Programming Interface
CPD	Centro de Processamento de Dados
HTTP	Hypertext Transfer Protocol
WWW	World Wide Web
CSS	Cascading Style Sheets
MAC	Media Access Control
BTU	British Thermal Unit
UPS	Uninterruptible Power Supply
TIA	Telecommunications Industry Association
TI	Tecnologia da Informação
ASHRAE	American Society of Heating, Refrigerating and Air-Conditioning Engineers

Sumário

1 Introdução

A alta disponibilidade das informações e a segurança dos equipamentos nas empresas, com certeza tem sido um grande desafio (OLIVEIRA e LOPES, 2017). Quando pensamos no conceito de segurança das informações, inicialmente pode nos vir à cabeça o conceito de antivírus e *firewalls*, entretanto, além disso, existem outros fatores que podem afetar diretamente a segurança desses dados, como o mal controle da temperatura e umidade no ambiente o qual os equipamentos estão instalados.

Os chamados *data centers* são os ambientes físicos em que se encontram instalados os mais variados equipamentos de rede, desde roteadores e *switches*, até as próprias máquinas servidoras que armazenam e processam os dados dos mais variados sistemas existentes. Para que estes equipamentos estejam em pleno funcionamento, é necessário que, além da disponibilidade de rede e energia, o ambiente físico ao qual eles encontram-se instalados seja altamente controlados (OLIVEIRA e LOPES, 2017).

Além do bom funcionamento dos equipamentos ali instalados, realizar o controle correto e eficiente de temperatura em *data centers* afeta outra variável muito importante no projeto desse ambiente que é o alto consumo de energia elétrica utilizado nos sistemas de refrigeração. Para Lange (2017), "como a climatização corresponde a um percentual em torno de 40% do consumo de energia, muitos gerentes de infraestrutura têm percebido que investir nesta área desde o projeto inicial pode gerar benefícios econômicos significativos durante a operação". Com isso, podemos notar que o correto controle e monitoramento da refrigeração dos *data centers* é bastante importante tanto pelo aspecto técnico como pelo financeiro, se revertendo em menores custos operacionais e de manutenção no longo prazo.

Diante desta problemática, é proposto neste trabalho o projeto de um protótipo de um sistema de monitoramento de temperatura e umidade. O protótipo conta com um microcontrolador, um sensor de temperatura e umidade DHT22 e um módulo Wi-Fi para realizar a conexão do sistema na rede.

1.1 Objetivo

Nesta seção, serão apresentados os objetivos gerais e específicos deste trabalho.

1.1.1 Objetivo Geral

Desenvolver um protótipo de um sistema de monitoramento de temperatura e umidade para *server rooms* de *data centers*, a fim de permitir o condicionamento adequado

e o perfeito funcionamento das máquinas nelas instaladas.

1.1.2 Objetivos Específicos

- Projetar os circuitos com o sensor responsável pela medição de temperatura e umidade;
- Fazer a integração do sensor com o microcontrolador Arduino para poder realizar a captação e tratamento dos dados;
- Desenvolver e implementar uma interface de monitoramento web;
- Desenvolver e implementar um servidor SNMP para realizar a integração com sistemas de monitoramento já existentes no mercado;
- Realizar testes de campo e analisar os resultados obtidos.

1.2 Estrutura do Trabalho

Esse Trabalho de Conclusão de Curso é composto por 5 capítulos, os quais são apresentados a seguir:

- Capítulo 1 - é apresentada a contextualização do tema e a motivação que levou ao desenvolvimento do protótipo, bem como a definição dos objetivos e a estruturação deste trabalho.
- Capítulo 2 - são apresentados os conceitos teóricos que embasaram o desenvolvimento do protótipo do sistema.
- Capítulo 3 - são apresentados os materiais e a metodologia utilizada no desenvolvimento do sistema de monitoramento de temperatura e umidade.
- Capítulo 4 - são descritos e analizados os resultados dos testes de campo realizados após a finalização do protótipo.
- Capítulo 5 - são apresentadas as conclusões acerca do trabalho.

2 Fundamentação Teórica

Neste capítulo serão apresentados os principais conceitos que embasaram teoricamente o estudo e desenvolvimento do protótipo de sistema de monitoramento de ambiente para *data centers*.

2.1 *Data Center*

Os *data centers* têm sua origem nos famosos CPDs (Centro de Processamento de Dados) dos anos 1980, que consistiam em salas enormes para abrigar uma única máquina de grande porte que necessitava de uma grande quantidade de energia e resfriamento (GARCIA, 2013). Segundo Faccioni Filho (2016), a principal diferença entre os CPDs e os *data centers* modernos é a densidade de equipamentos, visto que, os *server rooms* de hoje tem muito mais capacidade de armazenamento de máquinas que antigamente.

Um *data center* é um espaço físico, que comporta equipamentos diversos, com características específicas que proporciona condições de segurança ambiental e disponibilidade de conexão de alimentação e rede 24 horas por dia, 365 dias por ano (FARIA, 2017). Este tipo de segurança é alcançado através de redundância de interfaces de rede, de alimentação interna, de sistemas de monitoramento e controle de falhas nos equipamentos e aspectos inerentes ao ambiente, geração própria de energia elétrica, entre outros.

Com o avanço avassalador da tecnologia, cada vez mais as organizações utilizam *data centers* para hospedar seus sistemas. Uma das principais preocupações das organizações é manter a alta disponibilidade de seus negócios. Para isto é necessário que os *data centers* em que estes serviços estão hospedados possam oferecer padrões elevados de segurança para assegurar a integridade e funcionalidade de seu ambiente (GARCIA, 2013).

2.2 Controle do ambiente

Uma das variáveis que podem ocasionar uma indisponibilidade em serviços de *data center* é a má refrigeração e controle de umidade do ambiente. Para Soares (2016), a climatização é um sistema crítico pois, além de ser o segundo sistema que mais consome energia do conjunto, é, ao mesmo tempo, o responsável por manter o ambiente interno favorável à operação dos equipamentos que compõe o *Data center*, como servidores, *storages*, *switches*, entre outros.

A importância do controle de temperatura nestes ambientes se dá principalmente

pelo fato dos equipamentos instalados nos *server rooms* dos *data centers* aquecerem muito durante seu funcionamento, e esse tipo de aquecimento pode gerar desligamentos inesperados e até mesmo a danificação e queima permanente de seus componentes. Já quando falamos em umidade relativa do ar, a preocupação é com a condensação de água dentro dos servidores (quando a umidade está alta) e com descargas eletrostáticas causadas devido à baixa umidade relativa do ar (SOARES, 2016).

2.2.1 Ponto de Orvalho

Segundo Camargo (2016), o ponto de orvalho é a temperatura na qual o vapor d'água contido no ar se condensa quando resfriado a pressão e umidade absoluta constantes. Em consonância com a importância do monitoramento da umidade relativa do ar, realizar o monitoramento do ponto de orvalho do ambiente pode evitar danos causados devido à condensação do ar, que pode vir a infiltrar nos equipamentos causando oxidação dos materiais ou até mesmo queima dos componentes.

Uma aproximação para calcular o ponto de orvalho de posse somente a temperatura (T) e umidade relativa (UR) do ar é a "fórmula de Magnus":

$$\gamma(T, UR) = ln(\tfrac{UR}{100} exp((d - \tfrac{T}{d})(\frac{T}{c+T})) \tag{2.1}$$

$$T_{po} = \tfrac{c\gamma(T,UR)}{b-\gamma(T,UR)}; \tag{2.2}$$

onde b, c e d são constantes.

2.3 Normas e Regulamentações

No âmbito de certificações, normas e regulamentações para projeto, implementação e operação de *data centers*, se destacam algumas normas técnicas: ASHRAE-TC 9.9, ISO/IEC 24764, CENELEC-50600, ANSI/BICSI-002 e a ANSI/TIA-942-A.

A norma TI-942-A estipula como um intervalo de valores das variáveis do ambiente para um bom funcionamento dos equipamentos. São eles:

a) Temperatura de bulbo seco: 20°C (68°F) a 25°C (77°F);

b) Umidade relativa: 40 à 55%;

c) Máximo ponto de orvalho: 21°C (69,8°F);

d) Máxima variação de temperatura por hora: 5°C (9°F);

Por sua vez, a ASHRAE-TC 9.9, estipula um intervalo de 18° à 27°C, para a temperatura de bulbo seco.

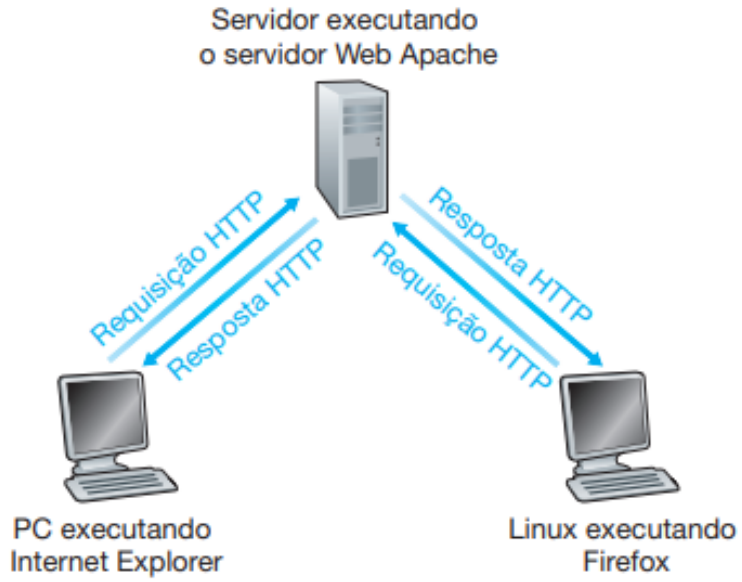
2.4 HTTP

O protocolo HTTP (*HyperText Transfer Protocol*) é um protocolo da camada de aplicação do modelo TCP/IP que é amplamente utilizado, pois constitui a base da World Wide Web (www) (TANEMBAUM, 2011). O protocolo é executado tanto no programa cliente como no programa servidor, estes por sua vez trocam mensagens padronizadas do tipo requisição-resposta (KUROSE e ROSS, 2013).

O servidor web (programa do lado do servidor), hospeda objetos que podem ser requisitados pelo cliente, tais como arquivos de texto (HTML - (*HyperText Markup Language*), arquivos de imagem ou vídeo, *scripts*, arquivos CSS (*Cascading Style Sheets*), entre outros (KUROSE e ROSS, 2013). Já o programa do lado, é um programa capaz de requisitar uma página web. Este programa é implementado pelo navegador (*brownser*).

Na Figura 1 encontra-se representada o comportamento da troca de mensagens entre clientes e um servidor web.

Figura 1 – Comportamento da troca de mensagens no protocolo HTTP



Fonte: Kurose e Ross, 2013.

2.5 SNMP

No início da década de 80 o protocolo *Simple Network Management Protocol* – SNMP começou a ser desenvolvido pelo *Internet Engineering Task Force* - IETF, com o objetivo ser o substituto do SGMP (*Single Gateway Management Procotocol*) - desenvolvido apenas para realizar o gerenciamento de roteadores na internet -, o SNMP nasce

com a premissa de disponibilizar uma forma simples e prática de realizar o controle de diversos equipamentos em uma rede de computadores (DIAS e ALVES JUNIOR, 2001).

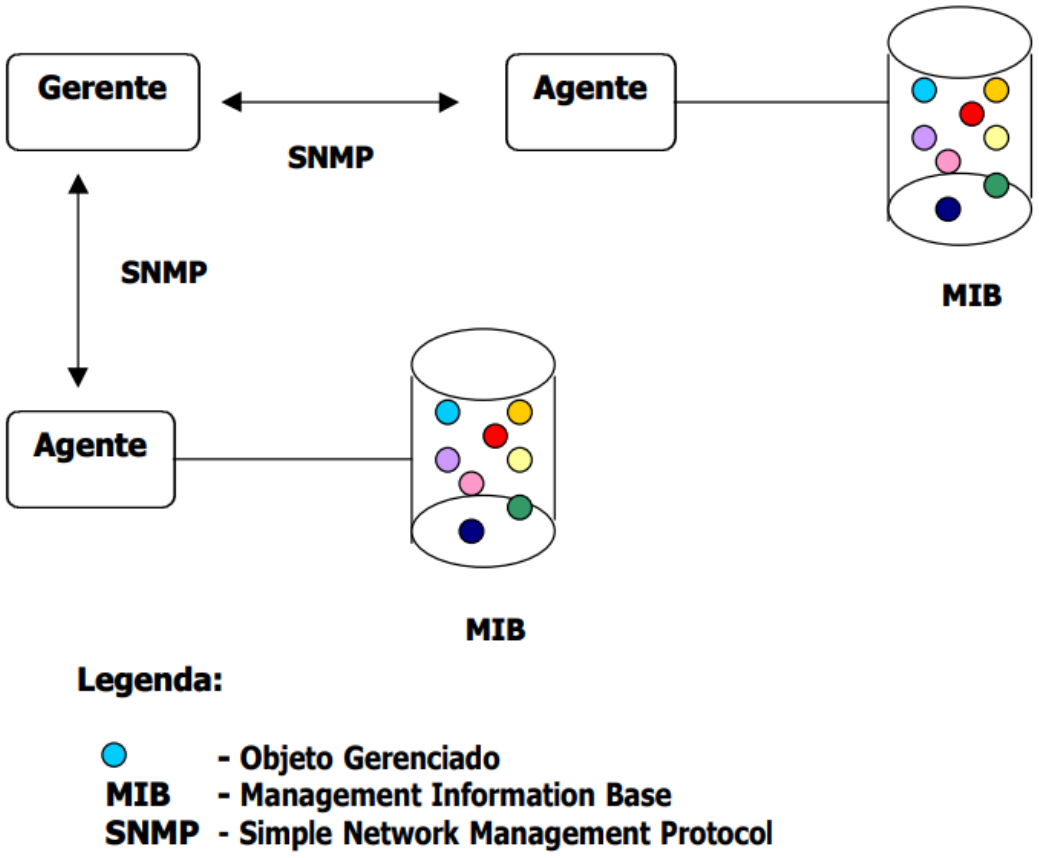
O SNMP é um protocolo da camada de aplicação do modelo OSI que utiliza na camada de transporte os serviços do protocolo UDP para enviar mensagens através da rede IP e que funciona baseado em dois tipos de entidades: os gerentes e os agentes.

Um gerente é um servidor operando de modo que executa algum tipo de sistema de *software* que pode lidar com tarefas de gerenciamento de uma rede (MAURO e SCHIMIDT, 2005). Basicamente, o gerente realiza duas operações: a operação de leitura dos valores monitorados e a operação de escrita, quando irá realizar alguma modificação nos valores do dispositivo.

Para Mauro e Schimidt (2005), "um agente é uma peça de *software* executada nos dispositivos a serem monitorados. O agente fornece ao gerente informações de gerenciamento rastreando os diversos aspectos operacionais dos dispositivos". As principais funções de um agente são atender requisições do gerente e enviar as informações solicitadas pelo mesmo. Uma outra função bastante importante do agente é o de notificar o gerente quando há a ocorrência de alguma evento não-previsto. Este tipo de aviso é enviado na forma de *traps*.

No modelo de gerenciamento do protocolo, as entidades estão dispostas da seguinte forma: uma estação é configurada como gerente e os demais elementos desempenham papel de agentes, onde cada agente possui uma MIB (*Management Information Base*) contendo as variáveis relativas aos objetos gerenciados (OLIVEIRA, 2001). Na Figura 2 está ilustrado o modelo de gerenciamento do protocolo.

Figura 2 – Estrutura de gerenciamento do SNMP



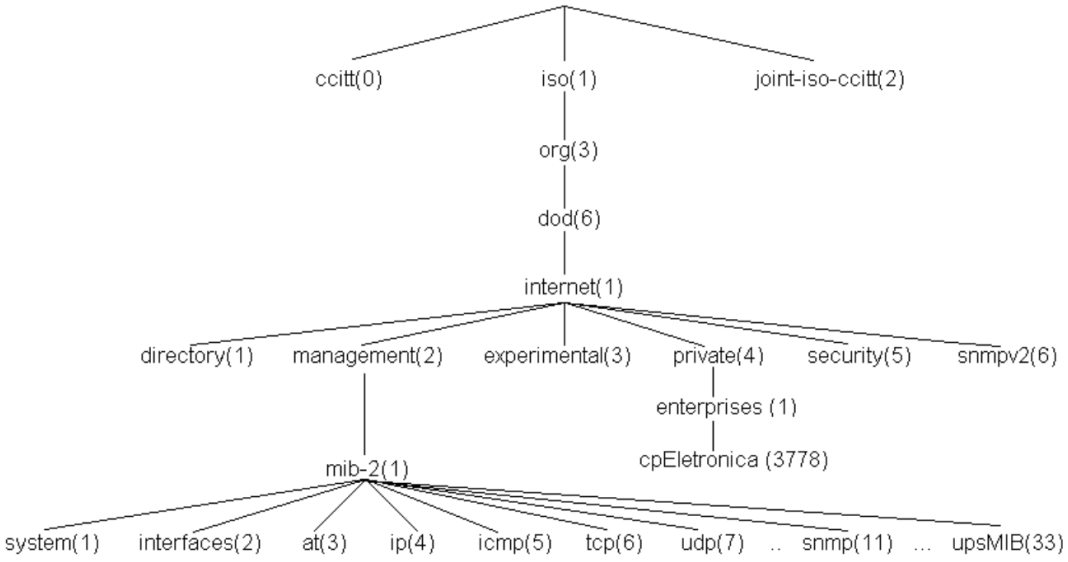
Fonte: Oliveira, 2001.

2.5.1 MIB

A MIB, como o próprio nome já sugere, é uma base de informações de gerenciamento. É através desta base de informações que o agente é capaz de responder ao gerente as consultas por SNMP. Para realizar esta comunicação, um arquivo MIB é criado e nele são adicionadas as informações necessárias para que o gerente saiba quais informações podem ser solicitadas e recebidas (informações de alerta - *traps*) de um agente (COTESSA e POLINA, 2013).

Uma MIB é constituída por uma estrutura em árvore (Figura 3) que contém as variáveis de gerenciamento de um determinado objeto. A MIB determina um OID (*Object Identifier*), que consiste em um identificador único formado por inteiros não negativos separados por ponto, para cada variável.

Figura 3 – Estrutura em árvore da MIB



Fonte: Coressa e Polina, 2013.

Segundo Coressa e Polina (2013), além do identificador, cada objeto possui também um nome, uma sintaxe, uma descrição e um controle de acesso. O nome do objeto (*Object Name*) é composto por uma *string* de texto curto. A sintaxe (*syntax*) descreve o formato ou valor e define o tipo do objeto. A descrição é uma string que informa o que a variável representa e o controle de acesso diz respeito ao tipo de controle que se pode ter sobre o objeto (somente leitura, leitura e escrita ou não acessível).

2.6 API

API, ou Interface de Programação de Aplicação em tradução livre, é o conjunto de padrões definidos por software que tem como objetivo permitir a interação com a aplicação utilizando rotinas conhecidas. De modo geral, APIs são comumente utilizadas para interação entre diferentes *software*, seja para controle, automação ou apenas coleta de dados.

Orenstein (2000) cita Josh Walker, analista da *Forrester Research Inc. em Cambridge, Massachusetts*, que diz que construir um *software* sem API "é basicamente como construir uma casa sem portas. A API para todos os propósitos de computação é como você abre as persianas e as portas e troca informações", validando a importância das APIs em *software* modernos.

3 Materiais e Métodos

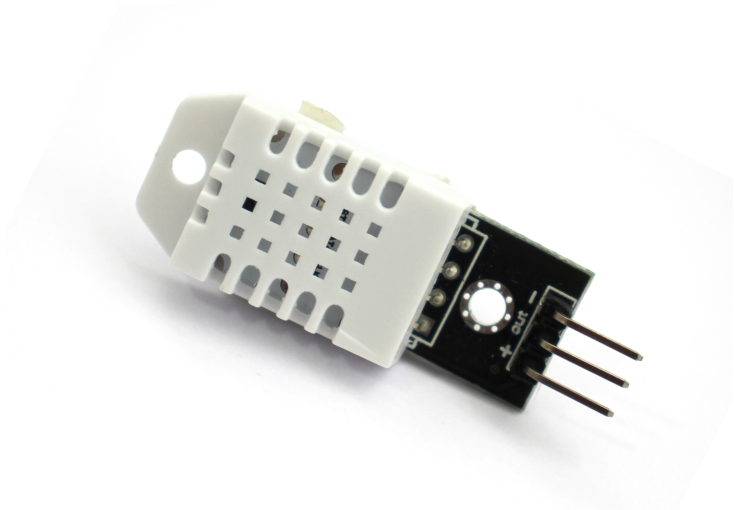
Neste capítulo será descrito como o protótipo foi projetado, quais sensores e microcontroladores foram utilizados, bem como foram desenvolvidas as interfaces de monitoramento e a implementação do protocolo de comunicação e monitoramento SNMP.

Ao decorrer do processo, devido a algumas limitações dos componentes envolvidos, foram desenvolvidos 2 protótipos. No primeiro protótipo, foram utilizados como microcontrolador o Arduino UNO R3 e como módulo Wi-Fi o ESP-01. Já no segundo protótipo, foi utilizado um único componente para realizar estas duas funções, o módulo Wi-Fi NodeMCU v3 (ESP-12). Em ambos os protótipos, foi utilizado o mesmo sensor de temperatura e umidade, o DHT22.

3.1 Módulo Sensor de Temperatura e Umidade - DHT22

O módulo sensor de temperatura e umidade DHT22 (Figura 4) é um sensor digital capaz de realizar medições com alta precisão. Da mesma família do já conhecido DHT11, o DHT22 se destaca pois realiza leituras a cada 2 segundos e é capaz de medir temperaturas entre -40º a 80º Celsius e umidade de 0% a 100%. De funcionamento simples, com apenas 3 pinos (+Vcc, saída e GND), o sensor conta com precisão de medição de umidade de aproximadamente 2% e de temperatura de 0,5º C.

Figura 4 – Módulo DHT22



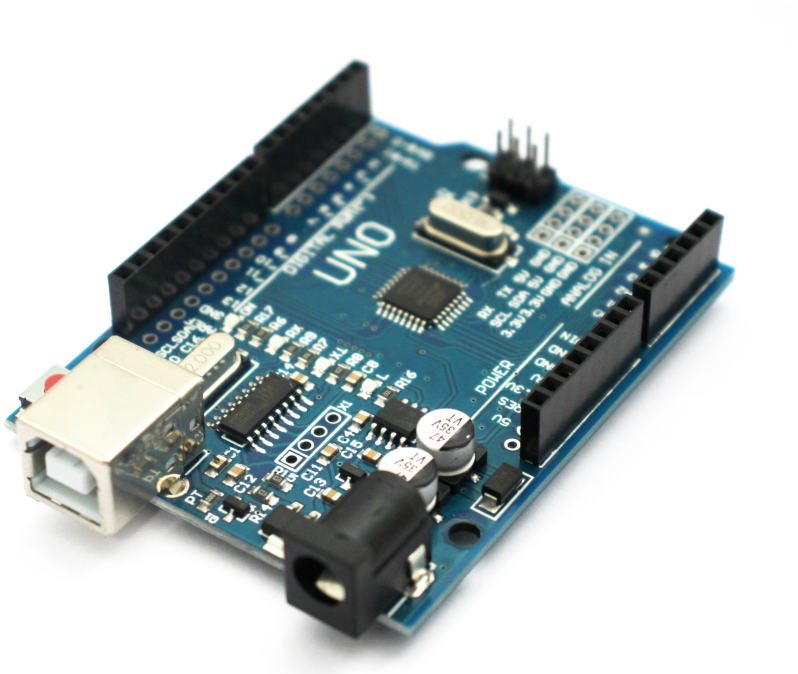
Fonte: Próprio autor, 2022.

3.2 Arduino UNO R3

O Arduino UNO (Figura 5) é uma placa microcontroladora que possui 14 pinos de entrada/saída digital (dos quais 6 podem ser usados como saídas PWM), 6 entradas analógicas, um cristal oscilador de 16 MHz, uma conexão USB, uma entrada de alimentação, um conector ICSP e um botão de *reset* e é baseada no chip ATmega328P (ARDUINO, 2022). O ATmega328 possui também 2KB de memória SRAM e 1KB de EEPROM (que pode ser lido ou gravado com a biblioteca EEPROM).

A terceira revisão da placa (R3) não utiliza o chip FTDI para conversão do sinal serial, utiliza no seu lugar um Atmega8U2 programado como conversor de USB para serial.

Figura 5 – Arduino Uno R3

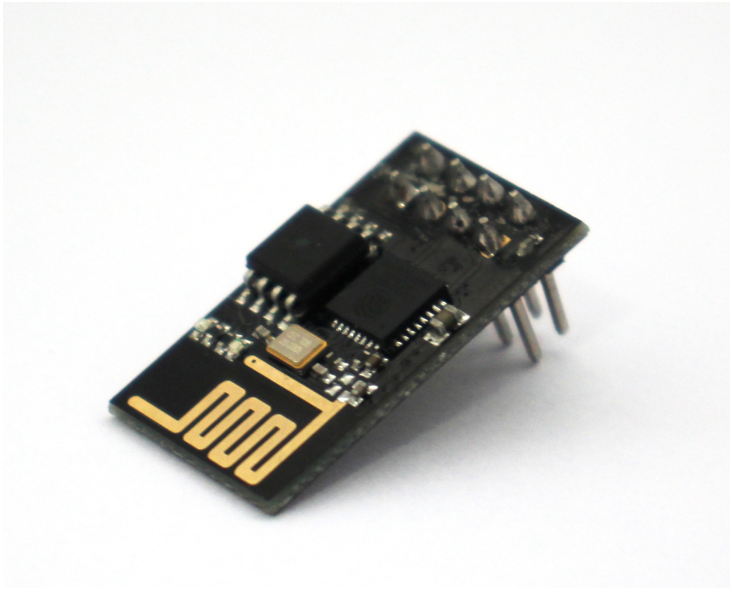


Fonte: Próprio autor, 2022.

3.3 ESP-01

O módulo Wi-Fi ESP8266 em sua primeira versão, mais conhecido como ESP-01 (Figura 6), possui suporte às redes 802.11 b/g/n, podendo trabalhar como um Ponto de Acesso (*Acess Point*) ou como uma Estação (*Station*), enviando e recebendo dados. Tem tensão de operação de 3.3 V, suporta comunicação TCP e UDP e, para realizar a comunicação do módulo com o Arduino, utiliza-se os pinos seriais RX e TX e a configuração é feita através de comandos AT.

Figura 6 – ESP-01



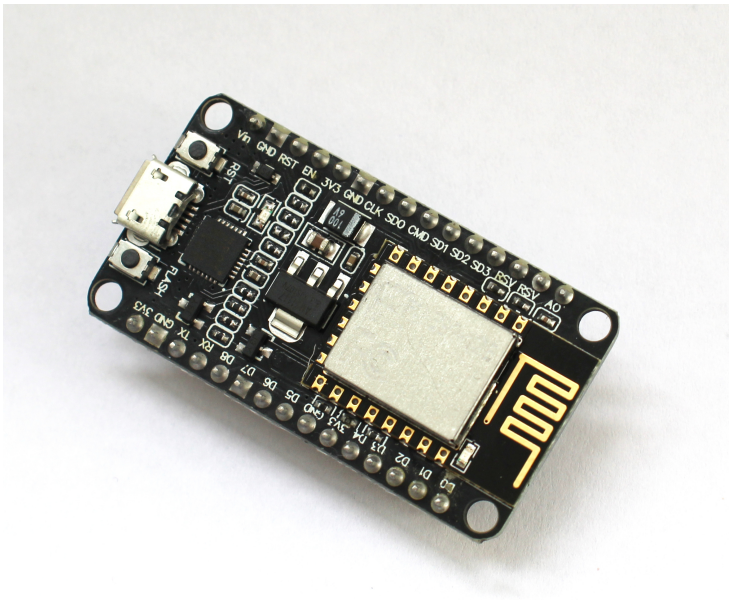
Fonte: Próprio autor, 2022.

3.4 NodeMCU v3 (ESP-12E)

O módulo Wi-Fi NodeMCU (Figura 7) é uma placa de desenvolvimento que possui o chip ESP8266, é alimentado através de porta micro USB e possui integrado um conversor USB-Serial e um regulador de tensão 3.3 V. Possui 9 pinos GPIOs (*General Purpose Input/Output*), ou seja, os pinos que fazem a comunicação de entrada e saída de sinais digitais.

Baseado no NodeMCU CH340 e com adicional extra de 32MB de memória flash, o NodeMCU v3 é ideal para projetos que utilizam quantidade maior de bibliotecas e códigos.

Figura 7 – NodeMCU

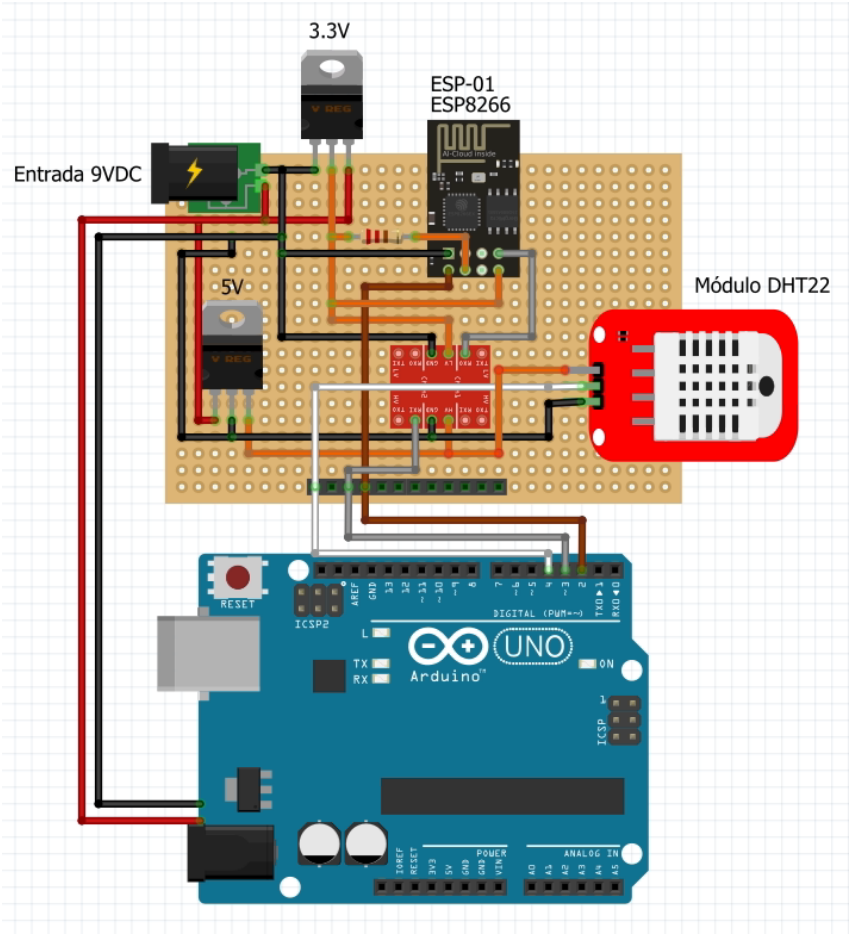


Fonte: Próprio autor, 2022.

3.5 Desenvolvimento do Protótipo I

Nesta seção será apresentada a primeira versão do sistema desenvolvido utilizando o microcontrolador Arduino UNO R3, o sensor de temperatura DHT22 e o módulo Wi-Fi ESP-01. Na Figura 8 está representado um esquemático de ligação da estrutura física do protótipo que conta com, além dos componentes já mencionados, um conversor de nível lógico, um resistor de 1kΩ e dois reguladores de tensão (para 5V e 3.3V), já que o protótipo é alimentado por uma fonte de 9V.

Figura 8 – Estrutura física do protótipo I



Fonte: Próprio autor, 2022.

A ligação do ESP-01 com o Arduino UNO foi realizada da seguinte forma: os pinos seriais do módulo (RX e TX) foram ligados respectivamente aos pinos digitais 2 e 3 do Arduino que, por sua vez, foram definidos no código como entradas seriais através da biblioteca *SoftwareSerial*. Como a saída dos pinos do Arduino encontram-se em 5V e o pino RX do ESP-01 deve ser alimentado com sinal de 3.3V, foi utilizado um conversor de nível lógico nessa ligação.

```
1 #include <SoftwareSerial.h>
2 SoftwareSerial esp8266(2, 3);
```

As demais ligações se deram da seguinte forma: os pinos Vcc e GND do ESP-01 foram ligados à tensão de 3.3V vinda do regulador de tensão instalado no protótipo e ao GND do circuito, respectivamente. O pino de *chip-enable* do ESP, foi ligado também à saída do regulador de tensão de 3.3V, através de um resistor de 1kΩ, para que a corrente de entrada fosse limitada. O módulo DHT22 foi alimentado com a tensão de 5V e seu pino de dados ligado ao pino 4 do Arduino.

3.5.1 Programação do ESP-01

O módulo Wi-Fi ESP-01 utiliza de comandos AT para poder realizar sua configuração e acionamento, quando este está sobre controle de algum outro dispositivo, como um Arduino, por exemplo.

Os comandos AT são comandos que são enviados com o prefixo AT (que significa "atenção") e são baseados nos comandos do chamado "Padrão Hayes", de 1981. Estes comandos são um conjunto de strings curtas que, combinadas geram diferentes comandos para realizar diferentes ações, como desligar, conectar ou alterar alguns parâmetros de conexão dos dispositivos (CAMPOS, 2015). Os comandos utilizados para configuração do ESP-01 neste projeto estão descritos na Tabela 1.

Tabela 1 – Comandos AT utilizados

Comando	**Descrição**
AT+RST	Reinicia o módulo
AT+CWJAP = 'SSID', 'senha'	Conecta o módulo à rede com SSID e senha fornecidos
AT+CWMODE = 1	Define o modo de operação do Wi-Fi como cliente (station)
AT+CIFSR	Obtém o endereço IP recebido pelo módulo
AT+CIPMUX = 1	Configura o módulo para múltiplas conexões
AT+CIPSERVER = 1,80	Inicia o servidor Web na porta 80
AT+CIPSEND = id, tamanho	Defina o comprimento dos dados que serão enviados paraa conexão cujo id foi enviado.
AT+CIPLOSE = id	Fecha a conexão TCP ou UDP da conexão com id enviado.

Fonte: Próprio autor, 2022.

3.5.2 Implementação do sensor DHT22 e Interface Web

O sensor DHT22 foi escolhido por ser mais preciso e abranger uma intervalo maior de possíveis leituras das variáveis a serem monitoradas. Para fazer a integração do mesmo, foi necessária a utilização de uma biblioteca específica, a DHT.h. A inclusão da biblioteca e as definições padrões para o funcionamento do sensor são mostrados a seguir:

```
1 #include <DHT.h>
2 #define DHTTYPE DHT22
3 #define DHT22_PIN 4
```

Para realizar o cálculo do ponto de orvalho foi escrita uma função que implementa a equação de Magnus (equações 2.1 e 2.2), e utiliza dos valores de temperatura e umidade recebidos do sensor. A função implementada pode ser visualizada na Figura 9.

Figura 9 – Função para cálculo do ponto de orvalho

```
//Função para calcular ponto de orvalho

float ponto_orvalho(float temp, float umi) {
  float a = 17.271;
  float b = 237.7;
  float lambida = (a * temp) / (b + temp) + log(umi/100);
  float Po = (b * lambida) / (a - lambida);
  return Po;
}
```

Fonte: Próprio autor, 2022.

No que diz respeito à página Web, com o servidor já iniciado na porta 80, devido ao comando AT+CIPSERVER = 1,80 enviado ao módulo, foi desenvolvida uma página web simples em HTML, a qual é enviada em forma de *string* para o ESP. O código da página pode ser visualizado na Figura 10.

Figura 10 – Montagem da *string* "webpage"

```
//Monta a página web para monitoramento das variáveis desejadas

webpage = "<head><meta http-equiv=\"refresh\" content=\"150\">";
webpage += "<meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">";
webpage += "</head><h2><center>Monitor de Temperatura e Umidade</center></h2>";
webpage += "<center> Temperatura atual: ";
webpage += t;
webpage += " &deg;C";
webpage += "</b> </center>";
webpage += "<center>Umidade atual: ";
webpage += u;
webpage += " %";
webpage += "</b> </center>";
webpage += "<center>Ponto de Orvalho: ";
webpage += Po;
webpage += " &deg;C </center>";
```

Fonte: Próprio autor, 2022.

3.5.3 API *Json*

A API Json foi configurada para ser acessada através do endereço */api.json*. Isso foi configurado utilizando a função *esp8266.readStringUntil(32) == "api.json"*, onde esp8266 é um objeto do tipo *SoftwareSerial*. Para a API, foram repassados os valores das 3 (três) variáveis monitoradas, seus valores e suas respectivas unidades, em forma de *string* que é enviada para o ESP quando requisitada.

Na Figura 11 podemos verificar o código da montagem da *string*.

Figura 11 – Montagem da *string* "webpage- API Json

```
//Monta a string webpage baseada no modelo de arquivo Json

webpage = "{\"temperatura\":{\"valor\": ";
webpage += t;
webpage += ", \"unidade\": \"C\"}, \"umidade\":{\"valor\": ";
webpage += u;
webpage += ", \"unidade\": \"%\"}, ptoOrvalho:{\"valor\": ";
webpage += Po;
webpage += ", \"unidade\": \"C\"}}";
```

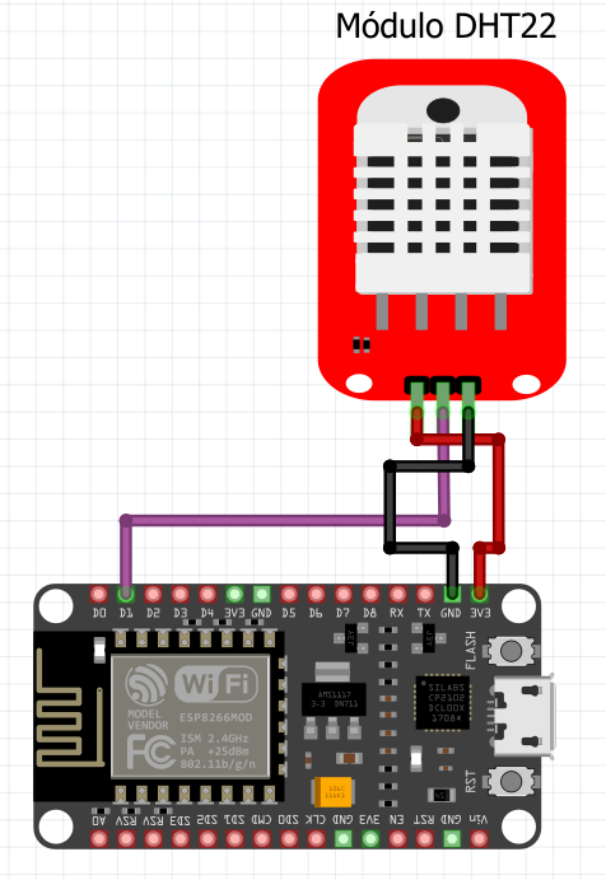
Fonte: Próprio autor, 2022.

O código completo com toda a implementação do sistema no Arduino se encontra no ANEXO I.

3.6 Desenvolvimento do Protótipo II

Nesta seção será apresentada a segunda versão do sistema desenvolvido utilizando o NodeMCU como microcontrolador e módulo Wi-Fi. Na Figura 12 está representado um esquemático de ligação da estrutura física do protótipo que conta apenas com um NodeMCU e o sensor DHT22. O pino de dados do módulo DHT22 foi ligado ao pino digital D1 do NodeMCU.

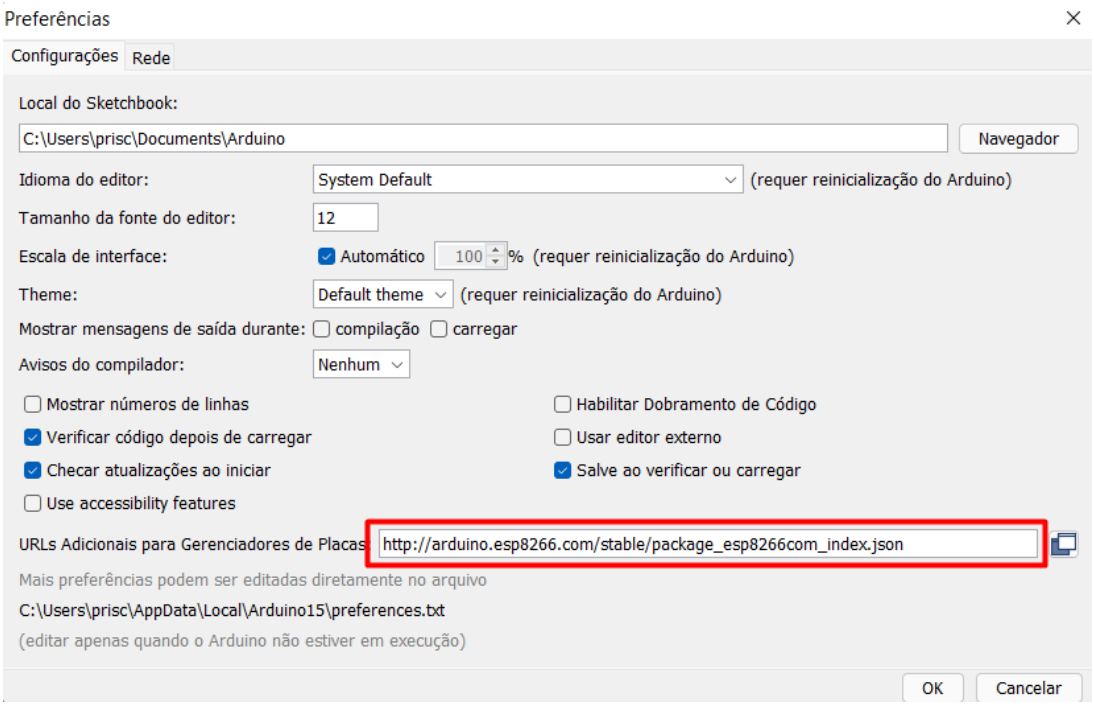
Figura 12 – Estrutura física do protótipo II



Fonte: Próprio autor, 2022.

Para utilizar o NodeMCU sem o auxílio do Arduino para programá-lo, foi necessário realizar a adição de um arquivo *json* com dados de gerenciamento das placas da família ESP8266. Esta configuração foi feita acessando a opção de preferências da IDE do Arduino, conforme ilustrado na Figura 13.

Figura 13 – Inserção do arquivo .json para gerenciamento das placas do ESP8266



Fonte: Próprio autor, 2022.

Neste protótipo, decidiu-se por utilizar bibliotecas prontas para realizar a comunicação e configuração do módulo Wi-Fi presente no NodeMCU. Isto foi feito pois, além de já ter sido mostrado a implementação no protótipo anterior, o código ficou mais limpo e menor. Sendo assim, além de incluir a biblioteca do sensor DHT, foram incluídas as seguintes bibliotecas:

```
1 #include <ESP8266WiFi.h>
2 #include <ESP8266WebServer.h>
3 #include <WiFiUdp.h>
4 #include <SNMP_Agent.h>
```

3.6.1 Programação do NodeMCU

Para a implementação do agente SNMP no sistema, foi necessário antes definir as comunidades de ordem pública e privada, que foram definidas conforme segue abaixo:

```
1 SNMPAgent snmp("prinpu", "prinpi");
```

Para definição das entidades MIB referentes a sensores, foram utilizados os valores referência de acordo com o *site* Cicitor (2021). Na Figura 14, pode-se visualizar todas as entidades declaradas para ambas as variáveis.

Figura 14 – Entidades MIB das variáveis monitoradas

```
// Itens do sensor de temperatura
int entPhySensorValue_1 = 999;                                  // Valor a ser atualizado
int entPhySensorType_1 = 8;                                     // Celsius
int entPhySensorScale_1 = 9;                                    // Escala de unidades (9 para unidades)
int entPhySensorPrecision_1 = 0;                                // Indicação de precisão do sensor
int entPhySensorOperStatus_1 = 1;                               // OK
std::string entPhySensorUnitsDisplay_1 = "Celsius";             // Unidade utilizada
uint32_t entPhySensorValueTimeStamp_1 = 0;                      // Valor da última atualização do sensor
int entPhySensorValueUpdateRate_1 = 0;                          // 0 na declaração, e será atualizado posteriormente

// Itens do sensor de umidade
int entPhySensorValue_2 = 999;                                  // Valor a ser atualizado
int entPhySensorType_2 = 9;                                     // Porcento
int entPhySensorScale_2 = 9;                                    // Escala de unidades (9 para unidades)
int entPhySensorPrecision_2 = 0;                                // Indicação de precisão do sensor
int entPhySensorOperStatus_2 = 1;                               // OK
std::string entPhySensorUnitsDisplay_2 = "Porcento";            // Unidade utilizada
int entPhySensorValueUpdateRate_2 = 0;                          // 0 na declaração, e será atualizado posteriormente
```

Fonte: Próprio autor, 2022.

De posse dessas entidades definidas, podem-se definir os OIDs dos objetos padrão e dos sensores de temperatura e umidade. Estas declarações podem ser verificados na Figura 15.

Figura 15 – OIDs utilizadas

```
// Definição as OIDs padrões do SNMPD
snmp.addReadOnlyStaticStringHandler(".1.3.6.1.2.1.1.1.0", "Sensor de Temperatura e Umidade");
snmp.addReadOnlyStaticStringHandler(".1.3.6.1.2.1.1.2.0", "");
snmp.addTimestampHandler(".1.3.6.1.2.1.1.3.0", &sysUptime);
snmp.addReadOnlyStaticStringHandler(".1.3.6.1.2.1.1.4.0", "Maria Priscilla (maria.lima@ee.ufcg.edu.br)");
snmp.addReadOnlyStaticStringHandler(".1.3.6.1.2.1.1.5.0", "Sensor SNMP");
snmp.addReadOnlyStaticStringHandler(".1.3.6.1.2.1.1.6.0", "Campina Grande, Brazil");
```

(a) OIDS Padrão

```
// OIDs do sensor de temperatura
snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.1.1.1", &entPhySensorType_1);
snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.1.2.1", &entPhySensorScale_1);
snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.1.3.1", &entPhySensorPrecision_1);
snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.1.4.1", &entPhySensorValue_1);
snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.1.5.1", &entPhySensorOperStatus_1);
snmp.addReadOnlyStaticStringHandler(".1.3.6.1.2.1.99.1.1.1.6.1", entPhySensorUnitsDisplay_1);
snmp.addTimestampHandler(".1.3.6.1.2.1.99.1.1.1.7.1", &entPhySensorValueTimeStamp_1);
snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.1.8.1", &entPhySensorValueUpdateRate_1);
```

(b) OIDs Sensor Temperatura

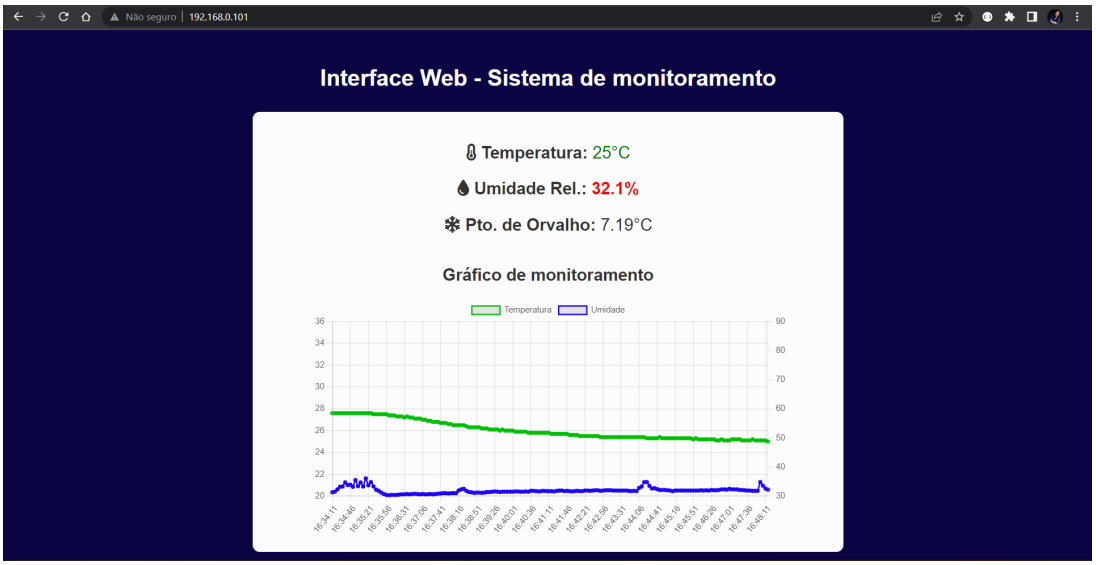
```
// OIDs do sensor de umidade
snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.2.1.1", &entPhySensorType_2);
snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.2.2.1", &entPhySensorScale_2);
snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.2.3.1", &entPhySensorPrecision_2);
snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.2.4.1", &entPhySensorValue_2);
snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.2.5.1", &entPhySensorOperStatus_2);
snmp.addReadOnlyStaticStringHandler(".1.3.6.1.2.1.99.1.1.2.6.1", entPhySensorUnitsDisplay_2);
snmp.addTimestampHandler(".1.3.6.1.2.1.99.1.1.2.7.1", &entPhySensorValueTimeStamp_1);
snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.2.8.1", &entPhySensorValueUpdateRate_2);
```

(c) OIDS Sensor Umidade

Fonte: Próprio autor, 2022.

No que se refere à aplicação web, não havendo mais a limitação de memória RAM encontrada no Arduino UNO, foi possível realizar a implementação de uma interface mais amigável, contendo um gráfico em tempo real do monitoramento das variáveis e objetos visuais de alerta quando os valores medidos encontram-se fora dos padrões desejáveis. Na Figura 16, podemos observar por exemplo que, no momento a temperatura encontrava-se dentro dos padrões desejáveis (valores em verde) e a umidade encontrava-se fora dos padrões desejáveis (valores em vermelho).

Figura 16 – Interface Web - Protótipo II



Fonte: Próprio autor, 2022.

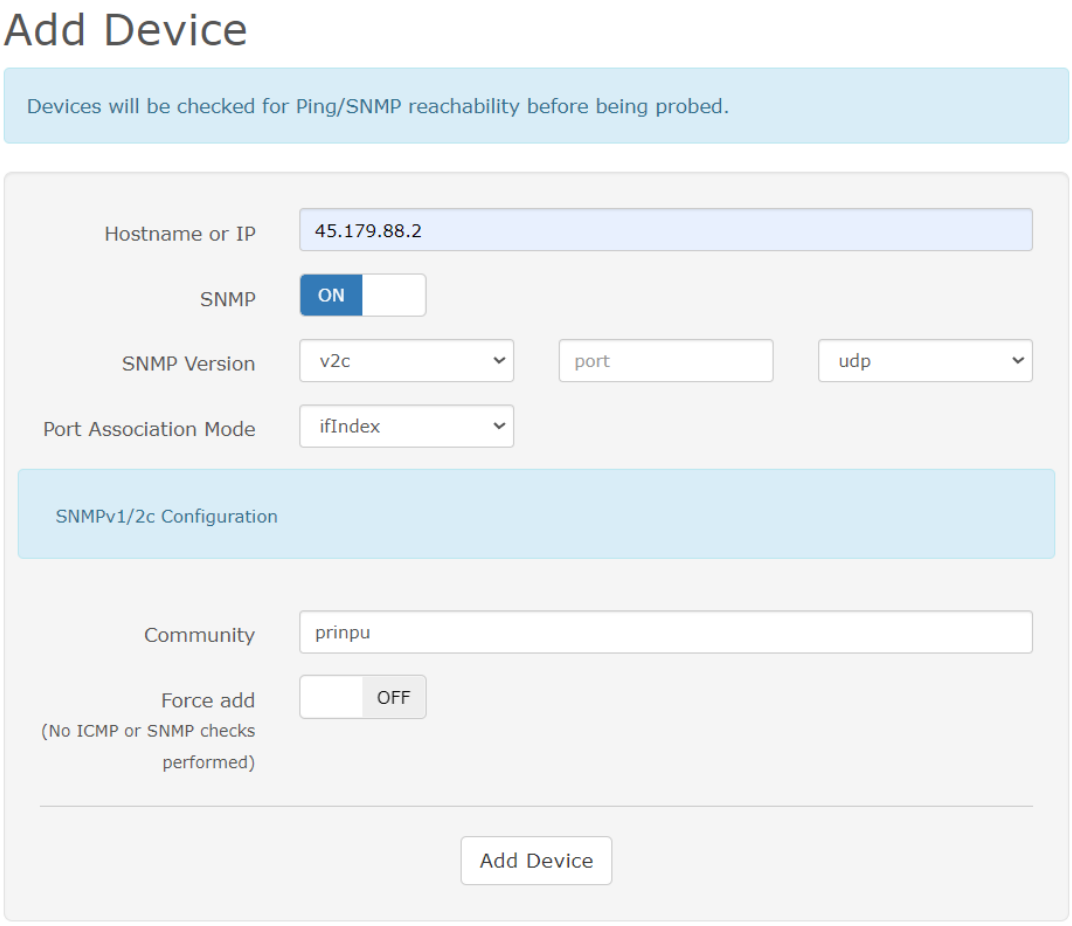
A programação para o funcionamento do sensor DHT22 com o NodeMCU e da API *Json* foi feita de forma semelhante ao primeiro protótipo, não sendo necessário realizar a descrição novamente nesta seção.

O código completo com toda a implementação do sistema no NodeMCU se encontra no ANEXO II.

3.6.2 Integração à aplicação de monitoramento SNMP

Afim de conferir os resultados da implementação do protocolo SNMP, o dispositivo foi adicionado à um sistema de monitoramento de código aberto, o LibreNMS, sistema este já utilizado pela empresa para realizar o monitoramento de diversos tipos de dispositivos em sua rede. Para adicionar um novo dispositivo ao sistema, em sua interface web, foi acessada a aba *Devices>Add Devices*, conforme podemos verificar na Figura 17.

Figura 17 – Página de configuração do dispositivo no LibreNMS



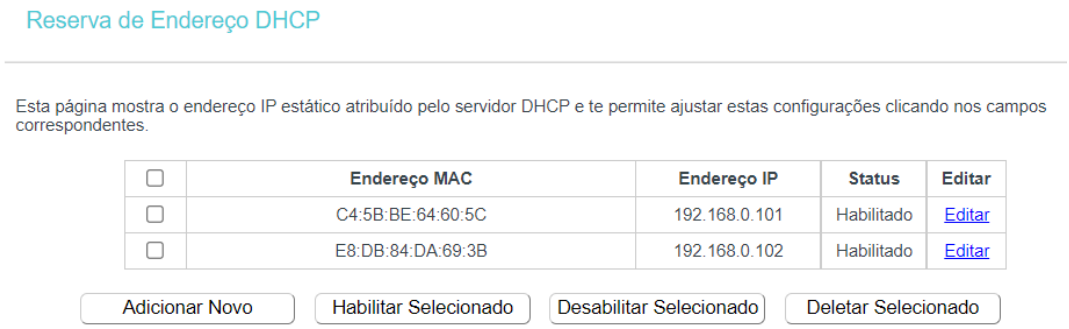
Fonte: Próprio autor, 2022.

3.7 Configuração de rede

Para que pudessem ser realizados testes, foi necessário realizar a configuração de uma rede dedicada para tal. O roteador utilizado foi configurado com o IP público 45.179.88.2, cedido pela empresa e a configuração do Wi-Fi foi feita da seguinte forma: SSID - "Datacenter" e senha - "datacenter123".

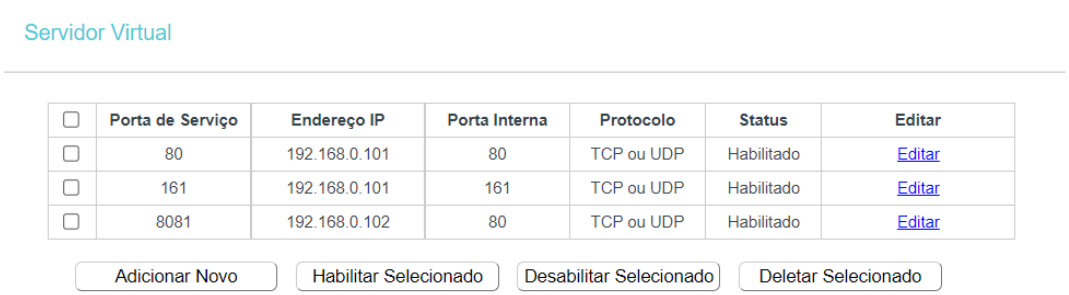
Em seguida, foram reservados os IPs 192.168.0.101 e 192.168.0.102 para os respectivos endereços MAC (*Media Access Control*) do módulo NodeMCU e o ESP-01, bem como também foi realizada a configuração do redirecionamento dos serviços utilizados no sistema, o serviço de SNMP e HTTP. Estas configurações podem ser observadas nas Figuras 18 e 19, respectivamente.

Figura 18 – Página de configuração das reservas de IP



Fonte: Próprio autor, 2022.

Figura 19 – Página de configuração do redirecionamento de serviços



Fonte: Próprio autor, 2022.

4 Resultados Obtidos

Neste capítulo apresentam-se os testes realizados com ambos os protótipos, seus respectivos resultados e análises, além de considerações a respeito da evolução do protótipo I para o protótipo II.

Os testes foram realizados em um ambiente real de data center. O local utilizado foi a sala de servidores do *data center* das empresas Hostzone Tecnologia LTDA e ISPTec Sistema de Comunicação Eireli, que conta com $36m^2$ de área, contendo *racks* com equipamentos de rede e servidores, *racks* com equipamentos do sistema de potência em Corrente Contínua, quadros de distribuição, sistemas de fornecimento elétrico ininterrupto (*UPS*), bancos de baterias estacionárias e sistema de refrigeração com capacidade de 132.000 BTUs.

4.1 Análise dos resultados do Protótipo I

Durante o desenvolvimento do primeiro protótipo foram encontrados alguns problemas. O primeiro deles se refere à um superaquecimento nos reguladores de tensão utilizados, que faziam com que eles atingissem suas temperaturas máximas e reduzissem a tensão, fazendo com que os demais componentes viessem a não ligar ou não funcionarem corretamente. Este problema foi resolvido adicionando uma massa térmica ao regulador de tensão afetado. Esta solução pode ser vista na Figura 20.

Figura 20 – Regulador de tensão acrescido de massa térmica

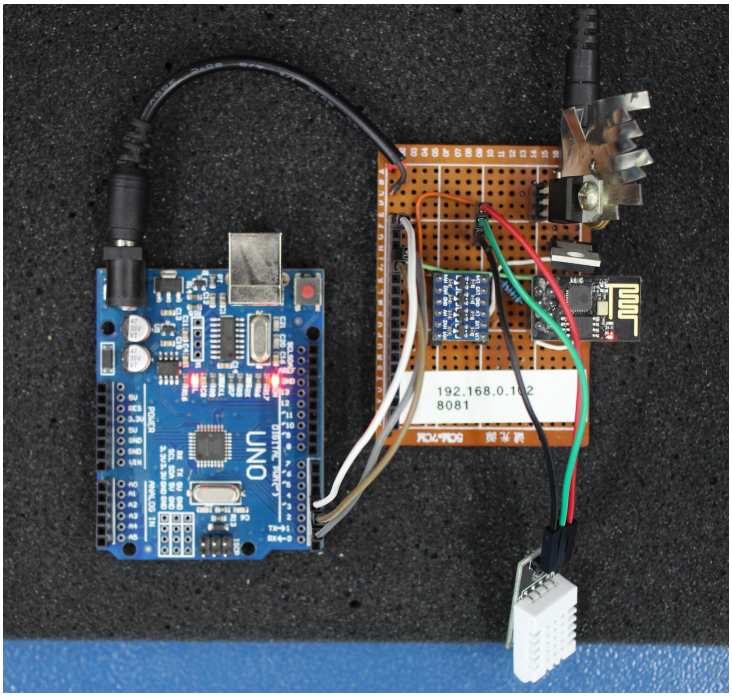
Fonte: Próprio autor, 2022.

Um outro problema encontrado foi em relação à limitação de memória RAM do Arduino UNO. Este problema foi identificado ainda nas etapas iniciais de desenvolvimento, haja visto que, ao adicionar novos códigos simples para a aplicação web, como alteração de cores nas fontes dos textos de acordo com os valores de temperatura e umidade por exemplo, ou a adição de imagens ou ícones, faziam com que o Arduino não conseguisse realizar todos os passos de conexão com o ESP ao ser realizadas requisições do cliente.

De posse desse problema, optou-se por realizar o desenvolvimento de um novo protótipo com o NodeMCU que conta com memória RAM de 20KB (10 vezes maior que a memória do Arduino UNO R3).

Sendo assim, o protótipo inicial foi desenvolvido com uma página web simples, contendo apenas texto e as variáveis e a API *json*. O circuito completo montado e instalado no local de testes encontra-se na Figura 21.

Figura 21 – Protótipo I instalado no local de teste



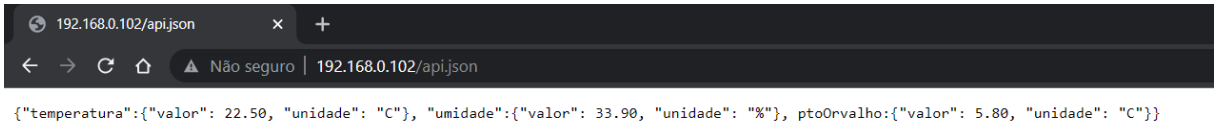
Fonte: Próprio autor, 2022.

Na Figura 22 e na Figura 23 são apresentadas a aplicação web e a API json, quando solicitadas ao servidor, respectivamente.

Figura 22 – Interface Web - Protótipo I

Fonte: Próprio autor, 2022.

Figura 23 – API Json - Protótipo I



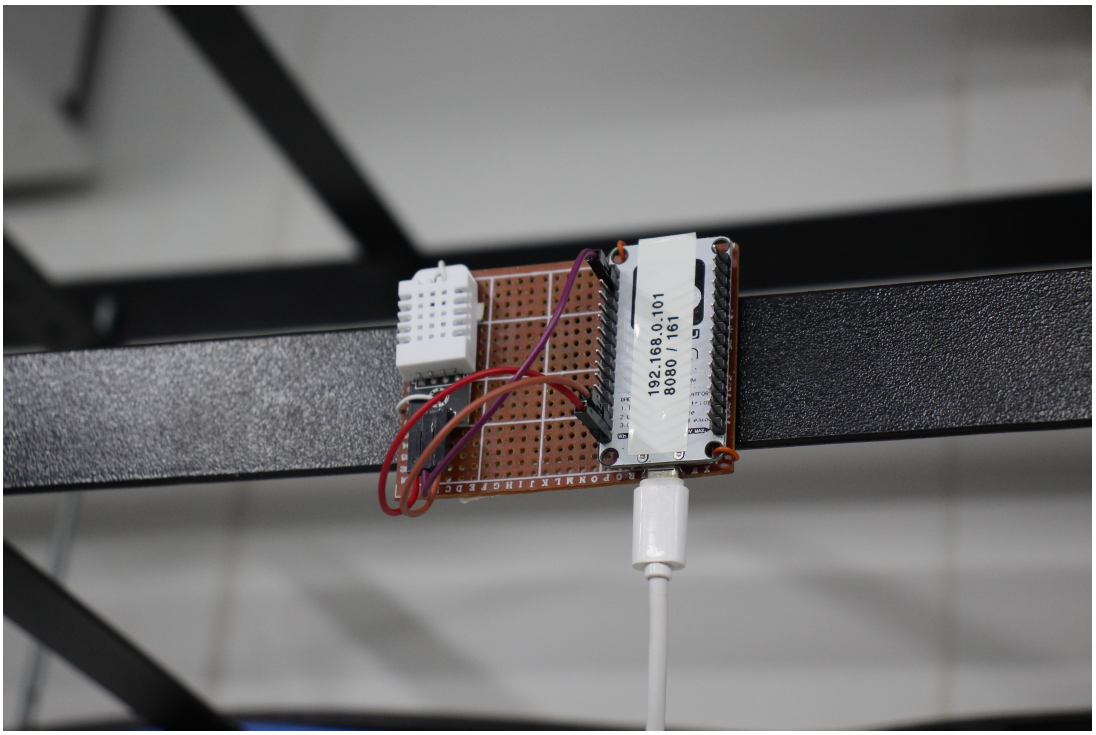
Fonte: Próprio autor, 2022.

Os valores obtidos no teste se mostraram muito próximos do ideal, visto que, no momento em que os testes foram realizados os ar-condicionados estavam configurados na temperatura de 22°C. Nas Figuras 22 e 23 temos resultados medidos pelo sensor de temperatura e umidade em momentos diferentes e, mesmo assim, os valores continuam muito próximos (23,4°C e 22,5°).

4.2 Análise dos resultados do Protótipo II

Nas mesmas condições em que o primeiro protótipo foi testado, temos o teste do protótipo II. O circuito do protótipo montado e instalado no local de testes encontra-se na Figura 24.

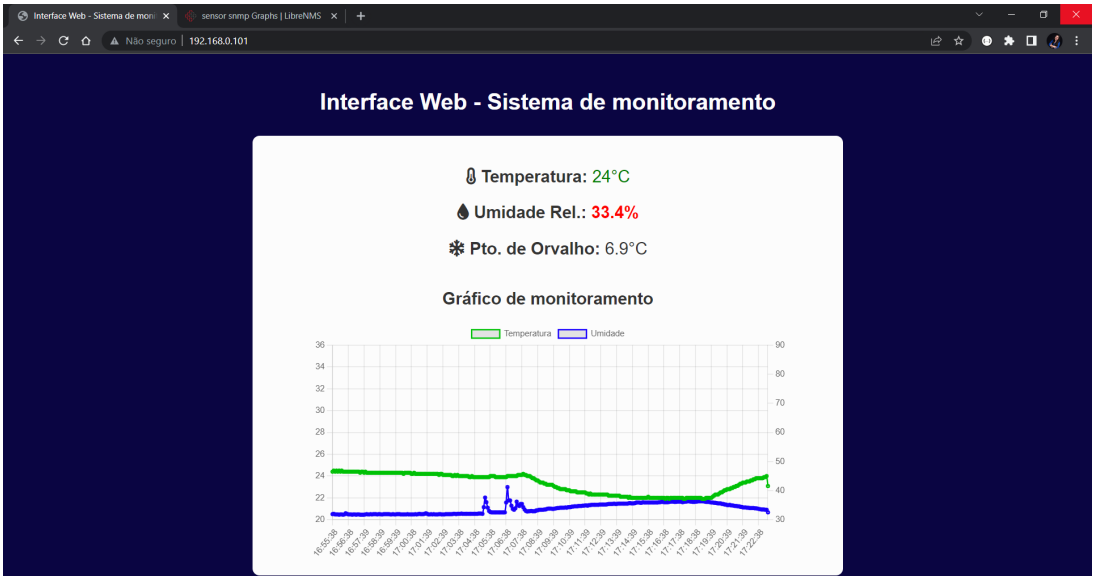
Figura 24 – Protótipo II instalado no local de teste



Fonte: Próprio autor, 2022.

Para a página web desta versão, foi adicionado além das informações de texto das variáveis, um gráfico contendo os dados de temperatura e umidade em função do tempo, onde os valores de temperatura encontram-se em verde e no eixo esquerdo do gráfico e os valores de umidade encontra-se em azul e no eixo direito do gráfico. Na Figura 25 podemos observar a aplicação web funcionando por um certo período de tempo no endereço definido anteriormente. A aplicação foi programada para exibir em verde quando as variáveis estiverem dentro dos padrões esperados e em vermelho quando ultrapassar esses limites.

Figura 25 – Aplicação Web - Protótipo II



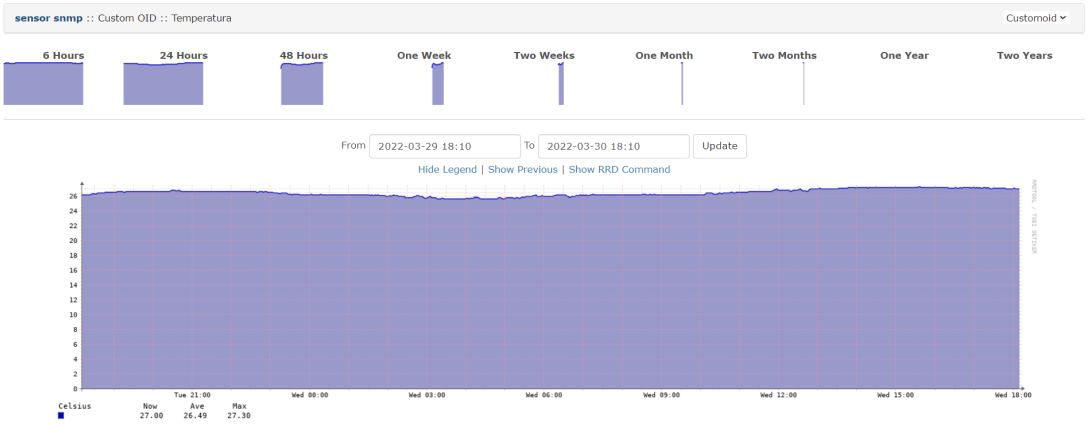
Fonte: Próprio autor, 2022.

Pode-se observar que a sala de servidores na qual o teste foi realizado conta com um sistema de refrigeração adequado, porém o controle de umidade do ambiente não é realizado.

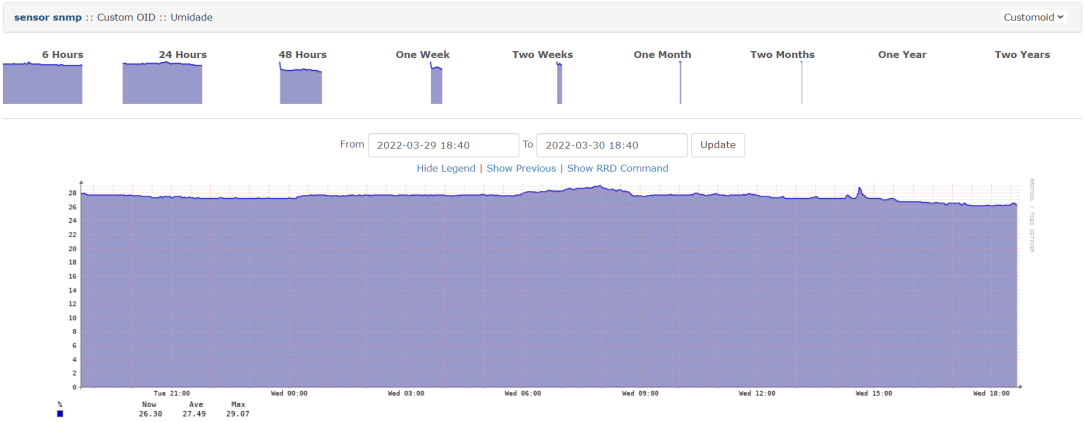
4.2.1 Análise de funcionamento da aplicação SNMP

No teste realizado foram coletados 24 horas de dados das variáveis e os gráficos gerados para temperatura e umidade no sistema de monitoramento LibreNMS, podem ser visualizados na Figura 26, onde foram encontrados os valores de 27,3°C de máximo e 26,49°C de média na temperatura da sala e os valores de 29,07% de máximo e 27,49% de média na umidade.

Figura 26 – Temperatura e Umidade - LibreNMS



(a) Temperatura

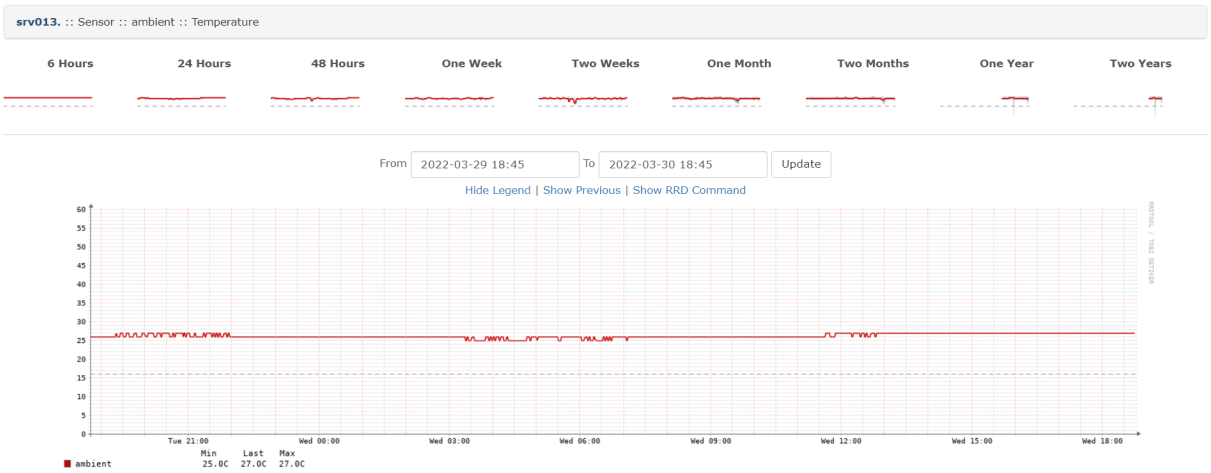


(b) Umidade

Fonte: Próprio autor, 2022.

Para fins de análises, podemos comparar os resultados obtidos durante o mesmo período do tempo com o sistema de monitoramento já existente no *data center*, também monitorado através do LibreNMS. Uma captura de tela foi retirada e o resultado encontra-se na Figura 27.

Figura 27 – Resultado do monitoramento utilizado pela empresa



Fonte: Próprio autor, 2022.

De posse destas informações, foi calculado o erro relativo da média da variável temperatura a fim de realizar uma análise matemática dos resultados. Este cálculo está descrito nas equações abaixo.

$$e_{\%} = \frac{|T_{comercial} - T_{prototipo}|}{T_{prototipo}} * 100 \tag{4.1}$$

$$e_{\%} = \frac{|26,8 - 26,49|}{26,49} * 100 \tag{4.2}$$

$$e_{\%} = 1,17\% \tag{4.3}$$

De posse destes resultados, podemos concluir que ambos os dispositivos monitorados, no que se refere à variável temperatura do ambiente, realizam o monitoramento de forma correta e eficiente. Sendo assim, podemos estender esta mesma consideração para a variável umidade relativa, não necessitando realizar nenhum tipo de calibração no sensor DHT22.

5 Considerações Finais

No presente trabalho foram projetados dois sistemas de monitoramento de temperatura e umidade para ambientes, especificamente para *server rooms* de *data centers*. Para isso foi utilizado como sensor um DHT22, a fim de realizar a aferição das variáveis, juntamente com um microcontrolador conectado à rede, para iniciar o servidor web e servir de agente SNMP. No primeiro sistema projetado foram utilizados o Arduino UNO R3 juntamente com o módulo Wi-Fi ESP8266 (ESP01) e no segundo sistema o módulo Wi-fi microcontrolado NodeMCU v3.

Apesar de simples e com algumas limitações, o protótipo I apresentou resultados satisfatórios, visto que realiza as medições de forma correta, medições essas que são apresentadas em interface web e podem ser facilmente utilizadas para realizar um monitoramento simples do ambiente.

O segundo protótipo desenvolvido se mostrou mais completo e com resultados similares aos utilizados comercialmente por empresas do ramo. Além de contar com uma interface web para monitoramento rápido que conta com gráficos e informações textuais, se mostrou muito eficiente quando monitorado através de um sistema de monitoramento por SNMP.

Durante o desenvolvimento dos sistemas, a aluna pode desenvolver habilidades no âmbito de implementação do protocolo de monitoramento SNMP, aprimorar suas habilidades no âmbito da instrumentação eletrônica, realizando a programação de dois tipos diferentes de microcontroladores, assim como também no âmbito da pesquisa, realizando uma vasta consulta bibliográfica a fim de aprender sobre as normas e recomendações de controle de ambiente dos *data centers*.

Como trabalhos futuros podem ser apontados, a adição de um atuador que realize o controle dos ar-condicionados através de sensores infra-vermelho, de acordo com a variação da temperatura e uma melhoria física no circuito, como a adição de uma fonte de alimentação própria para o circuito, como uma bateria, por exemplo.

Referências

1 ARDUINO. **UNO R3 | Arduino Documentation | Arduino Documentation**. 2021. Disponível em: <https://docs.arduino.cc/hardware/uno-rev3>. Acesso em 25 dez. 2021.

2 BERNAL FILHO. **SNMP: O que é?** Teleco.com.br. 2021. Disponível em: <https://www.teleco.com.br/tutoriais/tutorialsnmp/pagina_1.asp>. Acesso em 18 mar. 2022.

3 CAMARGO, D. S. **Sensoriamento e automação climática em sala de servidores utilizando software e hardware livre.** Joinville, 2016. Disponível em <https://sistemabu.udesc.br/pergamumweb/vinculos/000018/0000180e.pdf>. Acesso em 26 jan. 2022.

4 CAMPOS, A. **ESP8266: Comandos AT.** BR-Arduino.org. 2015. Disponível em: <https://br-arduino.org/2015/11/esp8266-comandos-at.html>. Acesso em 10 fev. 2022.

5 CIRCITOR. **Entity Sensor MIB.** 2021. Disponível em: <https://www.circitor.fr/Mibs/Html/E/ENTITY-SENSOR-MIB.php>. Acesso em 15 mar. 2022.

6 COTESSA, D. F., POLINA, E. R. **Gerenciamento de Equipamentos Usando o Protocolo SNMP.** Porto Alegre, 2013. Disponível em <http://paginas.unisul.br/carlos.luz/admredes/unidade1/ArtigoSNMP.pdf>. Acesso em 18 mar. 2022.

7 DEMETRIOU, D. A**SHRAE Technical Committee 9.9: Mission Critical Facilities, Data Centers, Technology Spaces, and Electronic Equipment.** electronics-cooling.com. 19 set. 2019. Disponível em: <https://www.electronics-cooling.com/2019/09/ashrae-technical-committee-9-9-\mission-critical-facilities-data-centers-technology-spaces-and-electronic-equipment/>. Acesso em 09 mar. 2022.

8 DIAS, B. Z., ALVES JUNIOR, N. **Protocolo de Gerenciamento SNMP**. 2001. Disponível em: <http://cbpfindex.cbpf.br/publication_pdfs/nt00601.2010_10_15_11_40_12.pdf>. Acesso em 18 mar. 2022.

9 FACCIONI FILHO, M. **Conceitos e infraestrutura de datacenters**. Palhoça, 2015.

10 FARIA, P. A. dos S. **Dimensionamento, Planeamento, Configuração e Colocação em Produção de um Data Center para uma Instituição de Ensino**

Superior. Coimbra, 2017. Disponível em: <https://comum.rcaap.pt/bitstream/10400.26/18274/1/Paulo-Santos-Faria.pdf>. Acesso em 22 jan. 2022.

11 GARCIA, D. **CENTRO DE DADOS (DATACENTERS).** 2013. Disponível em: <http://uniesp.edu.br/sites/_biblioteca/revistas/20170531143315.pdf>. Acesso em 05 jan. 2022.

12 KURUSE, J. F.; ROSS, K. W. **Redes de Computadores e a Internet**. 6ª ed. São Paulo: Pearson Education do Brasil, 2013.

13 LANGE, M. **Entrevista: Milena Lange, especialista em climatização em data centers.** WebArCondicionado. Disponível em: <https://www.webarcondicionado.com.br/entrevista-milena-lange-especialista-em-climatizacao-de-data-centers>. Acesso em: 30 mar. 2022.

14 MAURO, D. R.; SCHIMIDT, K. J. **Essential SNMP: Help for System and Network Administrators**. 2nd Edition. Sebastopol: O'Reilly Media, 2005.

15 OLIVEIRA, G. R., LOPES, R. A. **Controle de temperatura e umidade do CPD.** 2017. Disponível em: <https://www.conic-semesp.org.br/anais/files/2017/trabalho-1000026411.pdf>. Acesso em: 02 ago. 2021.

16 OLIVEIRA, L. S. **O Protocolo SNMP.** 2001. Disponível em <http://www.logicengenharia.com.br/mcamara/alunos/snmp_lecia.pdf>. Acesso em 19 mar. 2022.

17 ORENSTEIN, D. **Application Programming Interface**. 2000. Disponível em: <https://www.computerworld.com/article/2593623/application-programming-interface.html>. Acesso em 25 mar. 2022.

18 SOARES, L. C. **O Data Center e a importância da climatização**. Tiinside. 2015. Disponível em: <https://tiinside.com.br/02/06/2016/o-data-center-e-importancia-da-climatizacao/>. Acesso em 10 fev. 2022

Anexos

ANEXO A – Código do Protótipo I

```
1 // Prototipo I - TCC
2 // Autora: Maria Priscilla Lima Medeiros
3 // UFCG - Universidade Federal de Campina Grande
4
5 //
    ================================================================================
6
7
8
9 #include <SoftwareSerial.h>           //Inclusao da biblioteca para
    utilizacao do Serial
10 #include <DHT.h>                     //Inclusao da biblioteca do DHT22
    para utilizacao do sensor de temperatura e umidade
11
12
13 #define DHTTYPE DHT22                //Definicao do modelo de DHT
    utilizado
14 #define DHT22_PIN 4                  //Definicao do pino de dados
    conectado do arduino ao qual o DHT foi conectado
15
16
17
18 String resposta = "";                //Declaracao de variaveis globais
19
20
21
22 DHT dht(DHT22_PIN, DHTTYPE);         //Declaracao do objeto dht do tipo
     DHT
23
24 SoftwareSerial esp8266(2, 3);        //Definicao dos pinos 2 e 3 como
    seriais (RX pino 2, TX pino 3)
25
26
27 #define DEBUG true                   //Definicao de objeto de debug
28
29 //Funcao Setup
30
31 void setup()
32 {
33
34     Serial.begin(38400);             //Inicia Serial
35     esp8266.begin(38400);            //Inicia comunicacao com o modulo
```

```
    Wi-Fi
36
37     //Inicia envio de comandos AT (configuracao de modulo Wi-Fi na rede)
38
39     sendData("AT+RST\r\n", 1000, DEBUG);  //Comando de Reset
40     //Comando para conectar a rede wireless (envia SSID e Senha)
41     sendData("AT+CWJAP=\"Datacenter\",\"datacenter123\"\r\n", 1000,
    DEBUG);
42     delay(5000);
43     //Configura o modulo no modo 1 =>  estatico
44     sendData("AT+CWMODE=1\r\n", 1000, DEBUG);
45     //Comando para mostrar o endereco IP recebido pelo modulo
46     sendData("AT+CIFSR\r\n", 1000, DEBUG);
47     delay(1000);
48     //Configura o modulo para multiplas conexoes
49     sendData("AT+CIPMUX=1\r\n", 1000, DEBUG);
50     delay(1000);
51     //Comando para iniciar o servidor Web na porta 80
52     sendData("AT+CIPSERVER=1,80\r\n", 1000, DEBUG);
53     delay(1000);
54
55 }
56
57 //Funcao loop
58
59 void loop()
60 {
61
62    //Realiza a verificacao do ESP8266, se esta enviando dados
63   if (esp8266.available())
64   {
65       if (esp8266.find("+IPD,"))
66       {
67         //Realiza a leitura do sensor DHT22
68         int chk = dht.read(DHT22_PIN);
69
70         //Atribui as leituras de temperatura e umidade do sensor as
    respectivas variaveis
71         float t = dht.readTemperature();
72         float u = dht.readHumidity();
73
74         //Chama a funcao que calcula o ponto de orvalho e armazena o
    retorno na variavel Po
75         float Po = ponto_orvalho(t, u);
76
77         //Calcula o connection id
78         int connectionId = esp8266.read() - 48;
```

```
79
80        //procura na string "GET /"
81        esp8266.find("GET /");
82
83        //Declara variavel do tipo string "webpage"
84        String webpage = "";
85
86        //if para condicoes de requisicao
87        if(esp8266.readStringUntil(32) == "api.json"){
88
89          //Monta a string webpage baseada no modelo de arquivo Json
90
91          webpage = "{\"temperatura\":{\"valor\": ";
92          webpage += t;
93          webpage += ", \"unidade\": \"C\"}, \"umidade\":{\"valor\": ";
94          webpage += u;
95          webpage += ", \"unidade\": \"%\"}, ptoOrvalho:{\"valor\": ";
96          webpage += Po;
97          webpage += ", \"unidade\": \"C\"}}";
98        }else{
99
100         //Monta a pagina web para monitoramento das variaveis
    desejadas
101
102         webpage = "<head><meta http-equiv=\"refresh\" content=\"150\">
    ";
103         webpage += "<meta name=\"viewport\" content=\"width=device-
    width, initial-scale=1.0\">";
104         webpage += "</head><h2><center>Monitor de Temperatura e
    Umidade</center></h2>";
105         webpage += "<center> Temperatura atual: ";
106         webpage += t;
107         webpage += " &deg;C";
108         webpage += "</b> </center>";
109         webpage += "<center>Umidade atual: ";
110         webpage += u;
111         webpage += " %";
112         webpage += "</b> </center>";
113         webpage += "<center>Ponto de Orvalho: ";
114         webpage += Po;
115         webpage += " &deg;C </center>";
116       }
117
118       //Monta a string cipSend que sera enviada para o modulo atraves
    da funcao SendData
119
120       String cipSend = "AT+CIPSEND=";
```

```
121         cipSend += connectionId;
122         cipSend += ",";
123         cipSend += webpage.length();
124         cipSend += "\r\n";
125 
126         //Envia o comando cipSend atraves da funcao SendData
127 
128         sendData(cipSend, 1000, DEBUG);
129 
130         //Envia a string webpage atraves da funcao SendData
131 
132         sendData(webpage, 1000, DEBUG);
133 
134         //Monta a string closeCommand que sera enviada para o modulo
    atraves da funcao SendData
135 
136         String closeCommand = "AT+CIPCLOSE=";
137         closeCommand += connectionId; // append connection id
138         closeCommand += "\r\n";
139 
140         //Envia a string closeCommand atraves da funcao SendData
141 
142         sendData(closeCommand, 3000, DEBUG);
143      }
144   }
145 }
146 
147 //
    ================================================================================
148 
149 //Funcao para calcular ponto de orvalho
150 
151 float ponto_orvalho(float temp, float umi) {
152   float a = 17.271;
153   float b = 237.7;
154   float lambida = (a * temp) / (b + temp) + log(umi/100);
155   float Po = (b * lambida) / (a - lambida);
156   return Po;
157 }
158 
159 //
    ================================================================================
160 
161 //Funcao para enviar comandos para o modulo
162 
```

```
163
164 String sendData(String command, const int timeout, boolean debug)
165 {
166   // Envio dos comandos AT para o modulo
167   String response = "";
168   esp8266.print(command);
169   long int time = millis();
170   while ( (time + timeout) > millis())
171   {
172     while (esp8266.available())
173     {
174       //Enquanto o ESP estiver respondendo, ler o proximo caractere da
      saida serial
175       char c = esp8266.read();
176       response += c;
177     }
178   }
179   if (debug)
180   {
181     Serial.print(response);
182   }
183   return response;
184 }
```

ANEXO B – Código do Protótipo II

```
// Prot tipo II - TCC
// Autora: Maria Priscilla Lima Medeiros
// UFCG - Universidade Federal de Campina Grande

//
    ================================================================================


#include <ESP8266WiFi.h>                                                    //
    Inclus o da biblioteca para o Wifi do ESP
#include <ESP8266WebServer.h>                                               //
    Inclus o da biblioteca para o Webserver que ir  rodar na porta 80
#include "DHT.h"                                                            //
    Inclus o da biblioteca DHT22 para utiliza  o do sensor de
    temperatura

#include <WiFiUdp.h>                                                        //
    Inclus o da biblioteca WiFiUDP - protocolo SNMP roda em UDP
#include <SNMP_Agent.h>                                                     //
    inclus o da biblioteca para utiliza  o do SNMP

WiFiUDP udp;                                                                //
    Iniciamos o protocolo UDP
SNMPAgent snmp("prinpu", "prinpi");                                         //
    Defini  o das comunidades SNMP na ordem p blica ,privado

#define DHTTYPE DHT22                                                       //
    Define o DHT como sendo a vers o DHT22 para utiliza  o da
    biblioteca


/* Configura  o do Wifi */
const char* ssid = "Datacenter";
    // Nome da rede
const char* password = "datacenter123";
        // Senha

ESP8266WebServer server(80);                                                  //
    Inicializa  o do webserver na porta 80

// DHT Sensor
uint8_t DHTPin = D1;                                                       //
```

```
    Define o pino ao qual o DHT est  conectado
29 int offsetDHT = 0;                                                    //
    Offset para calibragem do sensor de temperatura
30
31 // Inicializa o sensor de temperatura e umidade.
32 DHT dht(DHTPin, DHTTYPE);
33
34 // Define vari veis com valores absurdos
35 float Temperatura = 999;
36 float Umidade = 999;
37
38 // Referencia para os valores abaixo: https://www.circitor.fr/Mibs/Html/
    E/ENTITY-SENSOR-MIB.php
39
40 // Itens do sensor de temperatura
41 int entPhySensorValue_1 = 999;
    // Valor a ser atualizado
42 int entPhySensorType_1 = 8;                                              //
     Celsius
43 int entPhySensorScale_1 = 9;                                             //
     Escala de unidades (9 para unidades)
44 int entPhySensorPrecision_1 = 0;
    // Indica  o de precis o do sensor
45 int entPhySensorOperStatus_1 = 1;
    // OK
46 std::string entPhySensorUnitsDisplay_1 = "Celsius";
              // Unidade utilizada
47 uint32_t entPhySensorValueTimeStamp_1 = 0;
          // Valor da  ltima  atualiza  o do sensor
48 int entPhySensorValueUpdateRate_1 = 0;
        // 0 na declara  o , e ser  atualizado posteriormente
49
50 // Itens do sensor de umidade
51 int entPhySensorValue_2 = 999;
    // Valor a ser atualizado
52 int entPhySensorType_2 = 9;                                              //
     Porcento
53 int entPhySensorScale_2 = 9;                                             //
     Escala de unidades (9 para unidades)
54 int entPhySensorPrecision_2 = 0;
    // Indica  o de precis o do sensor
55 int entPhySensorOperStatus_2 = 1;
    // OK
56 std::string entPhySensorUnitsDisplay_2 = "Porcento";
              // Unidade utilizada
57 int entPhySensorValueUpdateRate_2 = 0;
        // 0 na declara  o , e ser  atualizado posteriormente
```

```
58
59 uint32_t sysUptime = 0; // Uptime do sistema
60
61 // Vari veis para defini  o dos tempos de atualiza  o
62 static const unsigned long UPTIME_UPDATE_INTERVAL = 1000;
                  // Tempo entre atualiza  es do uptime (1000ms = 1s)
63 static unsigned long lastUptimeUpdateTime = 0;
64 static const unsigned long SENSOR_UPDATE_INTERVAL = 5000;
                // Tempo entre atualiza  es do uptime (5000ms = 5s)
65 static unsigned long lastSensorUpdateTime = 0;
66
67 void setup() {
68
69   Serial.begin(57600);                                                    //
      Inicializa nosso console serial para monitoramento e debugging
70   delay(100);
71
72   pinMode(DHTPin, INPUT);                                                  //
      Configura o sensor DHT com as vari veis selecionadas anteriormente
73
74   dht.begin();                                                        //
      Inicializa o sensor DHT
75
76   Serial.println("Conectando na rede ");
77   Serial.println(ssid);
78
79   WiFi.begin(ssid, password);                                              //
       Realiza a conex o com a rede Wifi configurada
80
81   // Verifica se o wifi est  conectado antes de continuar
82   while (WiFi.status() != WL_CONNECTED) {
83     delay(1000);
84     Serial.print(".");
85   }
86
87   Serial.println("");
88   Serial.println("Wifi conectado!");
89   Serial.print("IP recebido do DHCP: ");  Serial.println(WiFi.localIP())
      ;                   // Mostra o IP recebido no console para facilitar
      o acesso
90
91   server.on("/", handle_OnConnect);
      // Configura o webserver para receber requisi  es na raiz e as
      direciona para a fun  o handle_OnConnect()
92   server.on("/api.json", handle_api);
       // Configura o webserver para receber requisi  es em /api.jon e
      as direciona para a fun  o handle_api()
```

```
93  server.onNotFound(handle_NotFound);
       // Configura o webserver para direcionar todas as outras
     requisi  es para a fun  o handle_NotFound()
94
95  server.begin();                                                  //
     Inicia o servidor HTTP
96  Serial.println("Servidor HTTP iniciado");
97
98  snmp.setUDP(&udp);                                                //
     Instrui o SNMP a utilizar o UDP que configuramos
99  snmp.begin();                                                  // Inicia
     o servidor SNMP
100
101 Serial.println("SNMP Iniciado");
102 Serial.println("Servidor SNMP iniciado");
103
104 // Defini  o as OIDs padr es do SNMPD
105 snmp.addReadOnlyStaticStringHandler(".1.3.6.1.2.1.1.1.0", "Sensor de
     Temperatura e Umidade");       // Descri  o do dispositivo SNMP
106 snmp.addReadOnlyStaticStringHandler(".1.3.6.1.2.1.1.2.0", "");
                    // ID do dispositivo
107 snmp.addTimestampHandler(".1.3.6.1.2.1.1.3.0", &sysUptime);
                  // Uptime do dispositivo (utilizado para identificar se
      foi reiniciado)
108 snmp.addReadOnlyStaticStringHandler(".1.3.6.1.2.1.1.4.0", "Maria
     Priscilla (maria.lima@ee.ufcg.edu.br)"); // Pessoa respons vel pelo
     dispositivo
109 snmp.addReadOnlyStaticStringHandler(".1.3.6.1.2.1.1.5.0", "Sensor SNMP
     ");                  // Nome do dispositivo
110 snmp.addReadOnlyStaticStringHandler(".1.3.6.1.2.1.1.6.0", "Campina
     Grande, Brazil");            // Localiza  o do dispositivo
111
112 /* Defini  es dos sensores dispon veis por SNMP:
113 Como esses valores podem ser din micos, utilizamos apontadores.
114 Dessa forma o servidor SNMP ir  retornar sempre os valores mais
     recentes*/
115
116 entPhySensorValueUpdateRate_1 = SENSOR_UPDATE_INTERVAL;
               // Intervalo de atualiza  o do sensor
117
118 // OIDs do sensor de temperatura
119 snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.1.1.1", &
     entPhySensorType_1);                  // Tipo do sensor
120 snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.1.2.1", &
     entPhySensorScale_1);                    // Escala do sensor
121 snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.1.3.1", &
     entPhySensorPrecision_1);                  // Precis o do sensor
```

```
122   snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.1.4.1", &
       entPhySensorValue_1);                    // Valor do sensor
123   snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.1.5.1", &
       entPhySensorOperStatus_1);                // Status de opera  o do
       sensor
124   snmp.addReadOnlyStaticStringHandler(".1.3.6.1.2.1.99.1.1.1.6.1",
       entPhySensorUnitsDisplay_1);          // Unidade
125   snmp.addTimestampHandler(".1.3.6.1.2.1.99.1.1.1.7.1", &
       entPhySensorValueTimeStamp_1);            // Timestamp value
126   snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.1.8.1", &
       entPhySensorValueUpdateRate_1);             // Taxa de atualiza  o
       do sensor
127
128   // OIDs do sensor de umidade
129   snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.2.1.1", &
       entPhySensorType_2);                   // Tipo do sensor
130   snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.2.2.1", &
       entPhySensorScale_2);                    // Escala do sensor
131   snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.2.3.1", &
       entPhySensorPrecision_2);                 // Precis o do sensor
132   snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.2.4.1", &
       entPhySensorValue_2);                    // Valor do sensor
133   snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.2.5.1", &
       entPhySensorOperStatus_2);                // Status de opera  o do
       sensor
134   snmp.addReadOnlyStaticStringHandler(".1.3.6.1.2.1.99.1.1.2.6.1",
       entPhySensorUnitsDisplay_2);        // Unidade
135   snmp.addTimestampHandler(".1.3.6.1.2.1.99.1.1.2.7.1", &
       entPhySensorValueTimeStamp_1);            // Timestamp value
136   snmp.addIntegerHandler(".1.3.6.1.2.1.99.1.1.2.8.1", &
       entPhySensorValueUpdateRate_2);             // Taxa de atualiza  o
       do sensor
137
138   Serial.println("Setup conclu do");
139 }
140 void loop() {
141
142   // Esse    nosso loop principal. Nele iremos atualizar os valores dos
       sensores e receber requisi  es HTTP e SNMP
143
144   // Atualiza  o do uptime do dispositivo
145   if (millis() - lastUptimeUpdateTime >= UPTIME_UPDATE_INTERVAL)
                    // Verifica se j  deve-se atualizar o uptime, de
       acordo com o intervalo configurado
146   {
147     lastUptimeUpdateTime += UPTIME_UPDATE_INTERVAL;
              // Se sim, incrementamos o valor anterior com o intervalo
```

```
    de atualiza  o
148     sysUptime = pegaUptime();                                   //
    E atualizamos a vari vel de uptime com a fun  o pegaUptime()
149   }
150
151   // Leitura e atualaliza  o dos valores dos sensores
152   if (millis() - lastSensorUpdateTime >= SENSOR_UPDATE_INTERVAL)
                    // Verifica se j  deve-se atualizar os sensores, de
    acordo com o intervalo configurado
153   {
154     lastSensorUpdateTime += SENSOR_UPDATE_INTERVAL;
              // Se sim, incrementamos o valor anterior com o intervalo
    de atualiza  o
155
156     Temperatura = dht.readTemperature() + offsetDHT;
              // L  o valor da temperatura e atualiza a vari vel. O
    offset tem como objetivo a calibragem do sensor se necess rio
157     Umidade = dht.readHumidity();
    // L  o valor da umidade e atualiza a vari vel. O offset tem como
    objetivo a calibragem do sensor se necess rio
158     entPhySensorValue_1 = Temperatura*10.0;
          // Para o SNMP o valor   multiplicado por 10, j  que em SNMP
    n o temos valores flot, apenas int. Dessa forma temos uma casa
    decimal de precis o tamb m no SNMP
159     entPhySensorValue_2 = Umidade*10.0;
         // Para o SNMP o valor   multiplicado por 10, j  que em SNMP
    n o temos valores flot, apenas int. Dessa forma temos uma casa
    decimal de precis o tamb m no SNMP
160     entPhySensorValueTimeStamp_1 = sysUptime;
          // Indica  o de quando os sensores foram atualizados pela
     ltima  vez
161
162     Serial.println("Sensores atualizados");
163   }
164
165   server.handleClient();                                          //
      Recebe e trata as requisi  es HTTP
166   snmp.loop();                                                    //
    Recebe e trata as requisi  es SNMP
167 }
168
169 // Fun  o para receber milisegundos utilizados para uptime e taxa de
    atualiza  o
170 #if defined(ESP32)                                               //
    Se   um ESP32, utilizamos a fun  o nativa esp_timer_get_time()
171 uint64_t uptimeMillis()
172 {
```

```
173     return (esp_timer_get_time() / 1000);
174 }
175 #else
176 uint64_t uptimeMillis()                                                    //
        Caso contr rio utilizamos millis()
177 {
178     // https://arduino.stackexchange.com/questions/12587/how-can-i-
      handle-the-millis-rollover
179     static uint32_t low32, high32;
180     uint32_t new_low32 = millis();
181     if (new_low32 < low32)
182         high32++;
183     low32 = new_low32;
184     return (uint64_t)high32 << 32 | low32;
185 }
186 #endif
187
188 // Fun   o para indica  o do uptime atual do dispositivo
189 int pegaUptime()
190 {
191     return (int)(uptimeMillis() / 10);
            // Converte milisegundos para timeticks (cent simos de
      segundos)
192 }
193
194 // Fun   o para requsi  es na raiz do webserver
195 void handle_OnConnect() {
196   server.send(200, "text/html", montaHTML(Temperatura,Umidade));
                      // Define a resposta como 200 (OK), o tipo da
      p gna como text/html e o conte do de retorno de acordo com a
      fun   o montaHTML(Temperatura,Umidade)
197 }
198
199 // Fun   o para requsi  es no caminho /api.json do webserver
200 void handle_api() {
201   server.send(200, "text/html", montaHTMLJson(Temperatura,Umidade));
                           // Define a resposta como 200 (OK), o tipo da
      p gna como text/html e o conte do de retorno de acordo com a
      fun   o montaHTMLJson(Temperatura,Umidade)
202 }
203
204 // Fun   o para requsi  es em qualquer outra p gina no webserver
205 void handle_NotFound(){
206   server.send(404, "text/plain", "Not found");
207 }
208
209 // Fun   o para aproxima  o do c lculo do ponto de orvalho
```

```
210 float ponto_orvalho(float temp, float umi) {
211   float a = 17.271;
212   float b = 237.7;
213   float lambida = (a * temp) / (b + temp) + log(umi/100);
214   float Po = (b * lambida) / (a - lambida);
215   return Po;
216 }
217
218 /* Monta a p gina HTML que ser  servida na raiz do webserver
219
220 Para evitar v rias requisi  es desnecess rias ao servidor, essa
      p gina apenas    carregada uma vez no navegador do usu rio
221 e os valores s o atualizados a cada 5 segundos utilizando JavaScript
      consultado o endpoint /api.json
222
223 */
224 String montaHTML(float Temperaturastat,float Umidadestat){
225   String ptr = "<!DOCTYPE html> \n";
226   ptr +="<html>\n";
227   ptr +="<head>\n";
228   ptr +="<link rel=\"stylesheet\" href=\"//use.fontawesome.com/releases/
      v6.1.1/css/all.css\">\n";
229   ptr +="<meta name=\"viewport\" content=\"width=device-width, initial-
      scale=1.0, user-scalable=no\">\n";
230   ptr +="<title>Interface Web - Sistema de monitoramento</title>\n";
231
232   // Definimos o CSS diretamente na p gina para economizar uma
       requisi  o
233   ptr +="<style>\n";
234   ptr +="html {\n";
235   ptr +="font-family: Helvetica;\n";
236   ptr +="display: inline-block;\n";
237   ptr +="margin: 0px auto;\n";
238   ptr +="text-align: center;\n";
239   ptr +="}\n";
240   ptr +="body {\n";
241   ptr +="background-color: #0a0542;\n";
242   ptr +="margin-top: 25px;\n";
243   ptr +="}\n";
244   ptr +="h1 {\n";
245   ptr +="color: #fcfcfc;\n";
246   ptr +="margin: 50px auto 30px;\n";
247   ptr +="}\n";
248   ptr +="#interface {\n";
249   ptr +="width: 800px;\n";
250   ptr +="margin: auto;\n";
251   ptr +="background-color: #fcfcfc;\n";
```

```
252   ptr +="padding: 20px;\n";
253   ptr +="border-radius: 10px;\n";
254   ptr +="}\n";
255   ptr +="@media only screen and (max-width: 1024px) {\n";
256   ptr +="#interface {\n";
257   ptr +="width: calc(100% - 40px);\n";
258   ptr +="}\n";
259   ptr +="}\n";
260   ptr +="h2 {\n";
261   ptr +="color: #333333;\n";
262   ptr +="}\n";
263   ptr +="p {\n";
264   ptr +="font-size: 24px;\n";
265   ptr +="color: #333333;\n";
266   ptr +="margin-bottom: 5px;\n";
267   ptr +="}\n";
268   ptr +="</style>\n";
269   ptr +="<meta charset=\"UTF-8\">\n";
270   ptr +="</head>\n";
271
272   // In cio da p gina
273   ptr +="<body\">\n";
274   ptr +="\n";
275   ptr +="<h1>Interface Web - Sistema de monitoramento</h1>\n";
276   ptr +="<div id=\"interface\">\n";
277   ptr +="<div id=\"dados\">\n";
278   ptr +="<p><i class=\"fa-solid fa-temperature-half\"></i> <b>
       Temperatura:</b> <span id=\"stemp\">\n";
279   ptr += Temperaturastat;
280   ptr +="&deg;C</span></p>\n";
281   ptr +="<p><i class=\"fa-solid fa-droplet\"></i> <b>Umidade Rel.:</b> <
       span id=\"sumi\">\n";
282   ptr += Umidadestat;
283   ptr +="%</span></p>\n";
284   ptr +="<p><i class=\"fa-solid fa-snowflake\"></i> <b>Pto. de Orvalho:
       </b> <span id=\"sorv\">\n";
285   ptr += ponto_orvalho(Temperaturastat, Umidadestat);
286   ptr += "&deg;C</span></p><br>\n";
287   ptr +="<h2>Gr fico de monitoramento</h2>\n";
288   ptr +="</div>\n";
289   ptr +="<center><div id=\"grafico\" style=\"width:85%;\">\n";
290   ptr +="<canvas id=\"graficojs\"></canvas>\n";
291   ptr +="</div></center>\n";
292   ptr +="</div>\n";
293
294   //Inclus o do jquery e do chart.js para o gr fico
295   ptr +="<script src=\"https://cdnjs.cloudflare.com/ajax/libs/jquery
```

```
    /3.1.1/jquery.min.js\"></script>\n";
296 ptr +="<script src=\"https://cdn.jsdelivr.net/npm/chart.js\"></script
    >\n";
297 ptr +="<script id=\"rendered-js\" >\n";
298
299 // Configura  o do gr fico
300 ptr +="// Cria o grafico vazio inicial\n";
301 ptr +="var ctx_live = document.getElementById(\"graficojs\");\n";
302 ptr +="var myChart = new Chart(ctx_live, {\n";
303 ptr +="type: \"line\",\n";
304 ptr +="data: {\n";
305 ptr +="labels: [],\n";
306 ptr +="datasets: [{\n";
307 ptr +="label: \"Temperatura\",\n";
308 ptr +="data: [],\n";
309 ptr +="fill: false,\n";
310 ptr +="borderWidth: 2,\n";
311 ptr +="borderColor: \"#00c0ef\",\n";
312 ptr +="yAxisID: \"y\",\n";
313 ptr +="borderColor: \"rgb(0, 194, 6)\",\n";
314 ptr +="tension: 0.3,\n";
315 ptr +="},\n";
316 ptr +="{\n";
317 ptr +="label: \"Umidade\",\n";
318 ptr +="data: [],\n";
319 ptr +="fill: false,\n";
320 ptr +="borderWidth: 2,\n";
321 ptr +="borderColor: \"#00c0ef\",\n";
322 ptr +="yAxisID: \"y1\",\n";
323 ptr +="borderColor: \"rgb(25, 0, 255)\",\n";
324 ptr +="tension: 0.3,\n";
325 ptr +="}\n";
326 ptr +="]\n";
327 ptr +="},\n";
328
329 // Defini  o dos itens
330 ptr +="options: {\n";
331 ptr +="responsive: true,\n";
332 ptr +="elements: {\n";
333 ptr +="point:{\n";
334 ptr +="radius: 2\n";
335 ptr +="}\n";
336 ptr +="},\n";
337 ptr +="title: {\n";
338 ptr +="display: true,\n";
339 ptr +="text: \"Gr fico de temperatura\"\n";
340 ptr +="},\n";
```

```
341
342   ptr +="legend: {\n";
343   ptr +="display: true\n";
344   ptr +="},\n";
345
346   ptr +="scales: {\n";
347   ptr +="y: {\n";
348   ptr +="type: \"linear\",\n";
349   ptr +="display: true,\n";
350   ptr +="position: \"left\",\n";
351   ptr +="suggestedMin: 20,\n";
352   ptr +="suggestedMax: 35,\n";
353   ptr +="  },\n";
354   ptr +="y1: {\n";
355   ptr +="type: \"linear\",\n";
356   ptr +="display: true,\n";
357   ptr +="position: \"right\",\n";
358   ptr +="suggestedMin: 40,\n";
359   ptr +="suggestedMax: 90,\n";
360
361   ptr +="// Configuracao das linhas da grade\n";
362   ptr +="grid: {\n";
363   ptr +="  drawOnChartArea: false, // Apenas desenhamos a grade em um
       dos eixos\n";
364   ptr +="},\n";
365   ptr +="},\n";
366   ptr +="}\n";
367   ptr +="}\n";
368   ptr +="});\n";
369
370   // Fun  o para atualiza  o dos dados da p gina
371   ptr +="// Logica para receber os novos dados\n";
372   ptr +="var getData = function() {\n";
373   ptr +="var d = new Date();\n";
374   ptr +="var hora = d.toLocaleTimeString();\n";
375   ptr +="fetch(\"api.json\").then(result => result.json()).then((output)
       => {\n";
376   ptr +="myChart.data.labels.push(hora);\n";
377   ptr +="myChart.data.datasets[0].data.push(output.temperatura.valor);\n
       ";
378   ptr +="myChart.data.datasets[1].data.push(output.umidade.valor);\n";
379
380   ptr +="document.getElementById(\"stemp\").textContent=output.
       temperatura.valor + \" C \";\n";
381   ptr +="document.getElementById(\"sumi\").textContent=output.umidade.
       valor + \"%\";\n";
382   ptr +="document.getElementById(\"sorv\").textContent=output.orvalho.
```

```
      valor + \" C \";\n";
383   ptr +="if (output.temperatura.valor > 18 && output.temperatura.valor <
        27) {\n";
384   ptr +="document.getElementById(\"stemp\").setAttribute(\"style\", \"
      color:green;\");\n";
385   ptr +="}else{\n";
386   ptr +="document.getElementById(\"stemp\").setAttribute(\"style\", \"
      color:red;font-weight: bold;\");\n";
387   ptr +="}\n";
388   ptr +="if (output.umidade.valor > 40 && output.umidade.valor < 55) {\n
      ";
389   ptr +="document.getElementById(\"sumi\").setAttribute(\"style\", \"
      color:green;\");\n";
390   ptr +="}else{\n";
391   ptr +="document.getElementById(\"sumi\").setAttribute(\"style\", \"
      color:red;font-weight: bold;\");\n";
392   ptr +="}\n";
393
394   ptr +="}).catch(err => console.error(err));\n";
395   ptr +="myChart.update();\n";
396   ptr +="};\n";
397
398   ptr +="// Define intervalo de atualiza  o\n";
399   ptr +="setInterval(getData, 5000);\n";
400   ptr +="\n";
401   ptr +="</script>\n";
402   ptr +="</body>\n";
403   ptr +="</html>\n";
404
405   return ptr;
406 }
407
408 /* Monta a p gina HTML que ser  servida em /api;json
409
410 A p gina /api.json retorna em formato json todos os valores dos
      sensores. Al m desses dados serem utilizados para atualizar a
      p gina principal
411 eles tamb m podem ser utilizados pelo usu rio para realizar outras
      integra  es , j  que o formato json  facilamente tratado e
      amplamente documentado.
412
413 */
414
415 String montaHTMLJson(float Temperaturastat,float Umidadestat){
416   String ptr = "{\"temperatura\":{\"valor\": ";
417     ptr += Temperaturastat;
418     ptr += ", \"unidade\": \"C\"}, ";
```

```
419     ptr += "\"umidade\":{\"valor\": ";
420     ptr += Umidadestat;
421     ptr += ", \"unidade\": \"%\"},";
422     ptr += "\"orvalho\":{\"valor\": ";
423     ptr += ponto_orvalho(Temperaturastat, Umidadestat);
424     ptr += ", \"unidade\": \"C\"}";
425     ptr += "}";
426   return ptr;
427 }
```