UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

LEONARDO CÉSAR MENDES SANTOS PORTELA

DESENVOLVIMENTO DE PLATAFORMA PARA AUTOMAÇÃO RESIDENCIAL COM DISPOSITIVOS INTELIGENTES UTILIZANDO PROTOCOLO MQTT E NODE-RED.

FORTALEZA

2022

LEONARDO CÉSAR MENDES SANTOS PORTELA

DESENVOLVIMENTO DE PLATAFORMA PARA AUTOMAÇÃO RESIDENCIAL COM DISPOSITIVOS INTELIGENTES UTILIZANDO PROTOCOLO MQTT E NODE-RED.

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Luiz Henrique Silva Colado Barreto

FORTALEZA

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

P877d Portela, Leonardo Cesar Mendes Santos.
Desenvolvimento de plataforma para automação residencial com dispositivos inteligentes utilizando protocolo MQTT e node-RED / Leonardo Cesar Mendes Santos Portela. – 2022.
57 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2022.
Orientação: Prof. Dr. Luiz Henrique Silva Colado Barreto.

1. MQTT. 2. Internet das Coisas. 3. Automação Residencial. 4. Node-RED. 5. Dispositivos inteligentes. I. Título.

CDD 621.3

LEONARDO CÉSAR MENDES SANTOS PORTELA

DESENVOLVIMENTO DE PLATAFORMA PARA AUTOMAÇÃO RESIDENCIAL COM DISPOSITIVOS INTELIGENTES UTILIZANDO PROTOCOLO MQTT E NODE-RED.

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Aprovada em: 29 de Julho de 2022

BANCA EXAMINADORA

Prof. Dr. Luiz Henrique Silva Colado Barreto (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Dalton de Araújo Honório
Universidade Federal do Ceará (UFC)

Prof. Dr. Eng. Nícolas de Araújo Moreira
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

Ao Prof. Dr. Luiz Henrique Silva Colado Barreto por me orientar no desenvolvimento deste trabalho de conclusão de curso.

À minha mãe, Tania Santos, por sempre ter me apoiado em minhas decisões e me ajudado nos momentos de dificuldades, sempre com muito amor. À Maria, por sempre me aconselhar quando precisei. Aos meus avôs, que apesar de não estarem mais aqui, devem estar orgulhosos onde estiverem.

Aos meus amigos de colégio, Georgia Gondim, Thiago Rodrigues, Pedro Portela, Bruno Mourão, Daniel Costa, Beatriz Mourão e Vitor Premoli, obrigado por sempre estarem perto de mim, nos momentos de tristeza e alegrias.

Aos meus amigos da UFC, Caio Sampaio, Eduardo Sobral, Karl Biermann, Jadir Araujo, Nathanael Vasconcelos, Pedro Rocha e Samuel da Silveira, obrigado por terem feito meus dias mais felizes e me ajudado a levantar em momentos de tristeza. Pedi amigos, ganhei irmãos.

Ao Doutorando em Engenharia Elétrica, Ednardo Moreira Rodrigues, e seu assistente, Alan Batista de Oliveira, aluno de graduação em Engenharia Elétrica, pela adequação do *template* utilizado neste trabalho para que o mesmo ficasse de acordo com as normas da biblioteca da Universidade Federal do Ceará (UFC).

“Não sei, só sei que foi assim!”

(Ariano Suassuna)

RESUMO

Este trabalho apresenta o desenvolvimento de uma plataforma de automação residencial capaz de gerenciar e monitorar dispositivos via internet. Inicialmente é apresentado o protocolo de comunicação *Message Queuing Telemetry Transport* (MQTT) que foi utilizado neste projeto, após isso é feito a apresentação da plataforma de desenvolvimento *Node-RED* que apresenta boa integração com o protocolo escolhido, o *Node-RED* foi utilizado tanto para a criação da interface do usuário quanto para os processos de tratamento dos dados e armazenamento dos mesmo em banco de dados ou no próprio computador. Os dispositivos utilizados nessa plataforma foram um interruptor inteligente, um monitor de temperatura/umidade e uma tomada inteligente, os dois primeiros utilizam o microcontrolador ESP8266 e o ultimo utiliza o ESP32. A interface de usuário criada se mostrou simples e funcional, nela é possível controlar tanto o interruptor quanto a tomada inteligente e visualizar o histórico dos dados em relação ao tempo, além disso os dados também são salvos no computador em formato "csv". Além disso, também foi feito a integração da plataforma com o assistente de voz Alexa. Portanto, a partir dos resultados e do desenvolvimento, é possível evidenciar que o conjunto MQTT e *Node-RED* se apresenta como uma solução de fácil aplicação, eficiente e generalista, podendo ser aplicada para diversos tipos de projetos, desde automação residencial simples até monitoramento e estudos de aplicações fotovoltaicas.

Palavras-chave: MQTT. Internet das Coisas. Automação Residencial. *Node-RED*. Dispositivos inteligentes. IoT. ESP32. NODEMCU ESP8266. Alexa.

ABSTRACT

This work presents the development of a home automation platform capable of managing and monitoring devices via the internet. Initially, the MQTT communication protocol that was used in this project is presented, after that, the *Node-RED* development platform is presented, which presents good integration with the chosen protocol, *Node-RED* was used both for the creation of of the user interface and for the processes of data treatment and storage in a database or on the computer itself. The devices used in this platform were a smart switch, a temperature/humidity monitor and a smart socket, the first two use the ESP8266 microcontroller and the last one uses the ESP32. The user interface created proved to be simple and functional, it is possible to control both the switch and the smart socket and view the history of the data in relation to time, in addition the data is also saved on the computer in "csv"format. The platform was also integrated with the Alexa voice assistant. Therefore, from the results and the development, it is possible to evidence that the set MQTT and *Node-RED* presents itself as an easy-to-apply, efficient and generalist solution, which can be applied to different types of projects, from a simple home automation project to monitoring a photovoltaic plant.

Keywords: MQTT. Internet of things. Home automation. Smart devices. *Node-RED*. ESP32. NODEMCU ESP8266

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

MQTT	*Message Queuing Telemetry Transport*
HTTP	*Hypertext Transfer Protocol*
DIY	*Do-It-Yourself*
IoT	*Internet of Things* / Internet das Coisas
TPC	*Transmission Control Protocol*
QoS	*Quality of Service* / Qualidade de Serviço
IBM	*International Business Machines Corporation*
IU	Interface de Usuário
GPIO	*General Purpose Input/Output*

SUMÁRIO

1 INTRODUÇÃO

1.1 Objetivos

Este trabalho tem como objetivo apresentar um método de criação de um sistema de monitoramento e controle de dispositivos inteligentes no âmbito da Internet das Coisas, voltado para a cultura *Maker* e *Do-It-Yourself* (DIY).

1.2 Motivação

Com o avanço nas tecnologias de sistemas embarcados, microeletrônica, telecomunicações, sensoriamento e também da necessidade da geração/utilização de dados, surge o conceito da Internet das Coisas, que se refere ao fato dos dispositivos estarem cada vez mais conectados, gerando dados (informações) que podem ser compartilhadas entre dispositivos ou utilizadas para monitoramento/tomadas de decisões por humanos.

Além da geração de informações uteis, a Internet das Coisas também abrange o controle de dispositivos via internet, ou seja, proporciona conforto para o usuário poder controlar seus dispositivos sem a necessidade de estar perto, como por exemplo, ligar o aquecedor de sua casa enquanto está no escritório para ao chegar na sua residencia a temperatura já ser a desejada ou iniciar algum processo em uma fábrica sem necessariamente existir algum funcionário in loco.

Com a *Internet of Things* / Internet das Coisas (IoT), também começa as surgir as casas inteligentes, que se utiliza dos dispositivos IoT para criar um ecossistema inteligente que o usuario pode controlar pelo seu *Smartphone* ou por dispositivos como a *Alexa* ou *Google Home*. Diversos são os fabricantes de dispositivos IoT, alguns deles são *SONOFF*®, Positivo®, Intelbras®, entre outros e algumas vezes podem ser caros, pouco personalizáveis e muitas vezes sendo necessário instalar diversos aplicativos diferentes para cada dispositivo de fabricante diferente.

A também popularização da chamada cultura *Maker* e DIY, faz com que as pessoas criem seus próprios dispositivos IoT e interfaces para controlá-los de forma mais barata e muitas vezes mais integradas que os dispositivos comerciais, além de também servir para criação de uma vasta documentação e ideias que podem ser utilizadas/personalizas de forma gratuita.

1.3 Metodologia

Este trabalho abordará todo o processo para a criação de um ecossistema no âmbito da Internet das Coisas, iniciando na escolha do protocolo de comunicação, passando pela criação da interface do usuário, desenvolvimento de protótipos e por fim a apresentação do sistema completo, de forma que seja de fácil entendimento e que possa servir de ponto de partida para trabalhos futuros ou inspiração para quem quiser fazer seu próprio sistema IoT.

1.4 Estrutura do Trabalho

No capítulo 1, são abordados os objetivos, motivação e a metodologia utilizada no desenvolvimento deste trabalho.

No capítulo 2, é apresentado o protocolo de comunicação escolhido para esse projeto, explicando cada componente e padrão de comunicação deste protocolo.

No capítulo 3, é explicado e apresentado a interface do usuário, utilizando a ferramenta *Node-RED* e suas bibliotecas.

No capítulo 4, são apresentados os *hardwares* utilizados para desenvolvimento de cada protótipo, além de detalhar a ideia por trás de cada dispositivo.

No capítulo 5, é apresentado a versão final da interface do usuário, além de sua interação com os respectivos protótipos.

Por fim, serão apresentadas as conclusões do desenvolvimento deste trabalho, além de sugerir algumas ideias para futuros trabalhos.

2 COMUNICAÇÃO

2.1 Introdução

Apesar de existir diversos protocolos de comunicação, nesse trabalho em específico, foi feito a escolha entre o MQTT e o HTTP, por ambos serem baseados em *Transmission Control Protocol* (TPC) e por apresentarem uma vasta quantidade informações pela internet.

Apesar de ambos serem baseados em TPC, o HTTP funciona da forma solicitação/-resposta, ou seja, o dispositivo envia a solicitação para o servidor e espera o servidor enviar a resposta (por isso existem as diversas respostas do HTTP, como por exemplo, a *404 Not Found*), já o MQTT funciona por meio do **padrão *Publisher/Subscriber***, onde diferente do HTTP, o cliente (*Subscriber*) não está diretamente conectado ao servidor *Publisher*, pois existe um intermediário que repassa essas informações (*Broker*), logo enquanto no HTTP o dispositivo necessita saber o endereço dos outros dispositivos, no MQTT, o cliente só precisa focar na mensagem que está enviando ou recebendo de outro dispositivo (LAMPKIN *et al.*, 2012).

Além disso, quando se trata de dispositivos IoT, principalmente quando ligados a baterias, o MQTT apresenta a vantagem de ser um protocolo leve e de baixo consumo energético, enquanto o HTTP, apesar de ser utilizado em algumas soluções de dispositivos inteligentes, é um protocolo mais pesado, que pode enviar mensagens mais complexas e pesadas, visto que é utilizado para transferências de dados na internet e em *websites* (LAMPKIN *et al.*, 2012).

Outro ponto interessante, é que o MQTT apresenta nativamente três tipos de garantia da entrega de uma mensagem (*Quality of Service* / Qualidade de Serviço (QoS)), enquanto no HTTP, é necessário que o desenvolvedor implemente esse mecanismo. Os três tipos de QoS podem ser vistos na tabela 1.

Tabela 1 – Qualidade de Serviço - QoS

Nivel do QoS	**Descrição**
0	Mensagem é enviada no máximo uma vez
1	Mensagem é enviada no mínimo uma vez
2	Mensagem é enviada exatamente uma vez

Fonte: Próprio Autor.
Nota: Baseado em Lampkin *et al.* (2012)

A figura 1 exemplifica a estrutura do protocolo MQTT, nela é possível notar que o processo é composto por 4 partes, sendo elas o tópico da mensagem, o *Broker*, o *Publisher* e o *Subscriber*. Por essa figura também é possível notar o fato que nesse padrão de comunicação os dispositivos não estão diretamente conectados, tudo passa pelo *Broker*, que analogamente, pode ser comparado com um centro de distribuição. Além disso, é interessante notar que vários clientes podem estar conectados ao mesmo *Broker* e receber/enviar informações ao mesmo tempo, sem problemas.

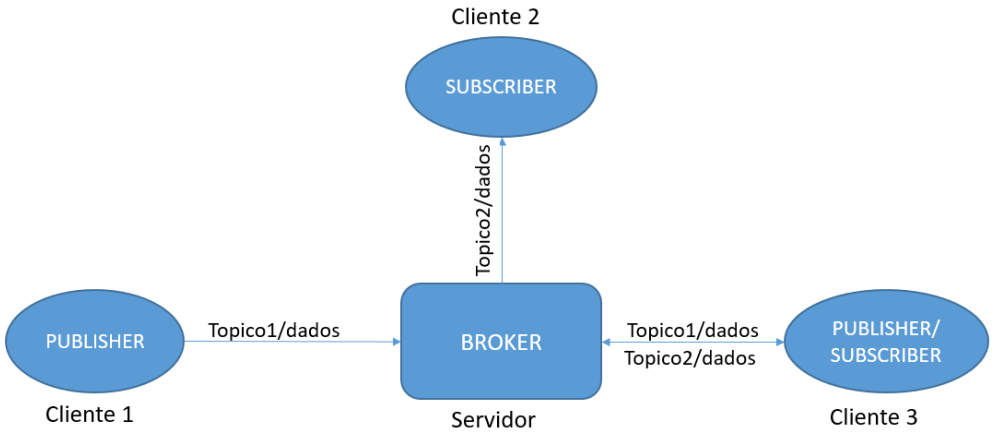
Por fim, conclui-se que para o projeto em questão, o uso do MQTT traz vantagens, como por exemplo o baixo custo de energia que esse protocolo demanda, pacote de dados menores, comunicação com maior qualidade de serviço nativo, entre outras. Na tabela 2 é possível ver uma comparação resumida entre MQTT e HTTP, com ela é possível ver qual protocolo se adapta melhor em relação a natureza do projeto.

Tabela 2 – Comparação MQTT x HTTP

	MQTT	**HTTP**
Orientação do projeto	Centrado em dados	Centrado no documento
Padrão	Publisher/Subscriber	Requisição/Resposta
Complexidade	Simples	Mais complexo
Tamanho da mensagem	Pequeno, 2 bytes é o menor tamanho	Grande, pois é baseado em texto
Níveis de serviço	Três configurações de níveis de serviço (QoS)	Todas as mensagens apresentam o mesmo nível
Bibliotecas extras	Para C (30 kB) e para Java (100 kB)	Depende do tipo da aplicação (JSON, XML)
Distribuição de dados	Suporta 1 para 0, 1 para 1 e 1 para n	Somente 1 para 1

Fonte: Adaptado de Lampkin *et al.* (2012)

Figura 1 – Exemplo de estrutura do MQTT



Fonte: Próprio Autor.

2.2 Tópico de Mensagem

O primeiro ponto para entender esse protocolo, é entender o canal por onde as mensagens são enviadas. No MQTT as mensagens são enviadas por meio de tópicos, por exemplo na figura 1, temos dois tópicos, o *Topico1/dados* e o *Topico2/dados*, ambos são totalmente separados, ou seja, ao mesmo tempo que o *Topico1/dados* está recebendo uma mensagem, o *Topico2/dados* também pode receber, sem interferência.

Outro fato importante é que os tópicos são canais de duas vias, ao mesmo tempo que um cliente pode enviar informações por um tópico, outro cliente pode receber informações por esse tópico, essa dupla funcionalidade é que vai fazer surgir as nomenclaturas *Subscriber* e *Publisher*.

2.3 Subscriber/Publisher

Os clientes no protocolo MQTT podem ser *Subscriber* ou *Publisher*, quando o cliente é um *Publisher*, ele publica informações em um dado tópico de mensagem, já o quando o cliente é *Subscriber*, ele assina esse tópico, ou seja, ele recebe as informações por meio desse tópico, é importante ressaltar que um mesmo cliente pode ser *Publisher* em um tópico e *Subscriber* em outro tópico.

Conforme a figura 1, é possível notar que o Cliente 1 é um *Publisher*, pois ele está enviado informações por meio do **Topico1/dados**, o Cliente 2 é *Publisher*, pois o mesmo recebe informações por meio do **Topico2/dados** e por fim o Cliente 3 faz ambos os papéis, tanto recebe informações do **Topico1/dados**, quanto envia informações pelo **Topico2/dados**.

2.4 Broker

Por fim, existe o *Broker*, que é o ponto principal do funcionamento deste protocolo, o *Broker* nada mais é que o servidor, por onde os dados vão chegar e vão ser enviados pelos protocolos. Assim como um servidor, ele tanto pode ser local quanto online e existem várias opções disponíveis de acordo com as expectativas do projeto, por exemplo, existe o *Mosquitto*, que é um *Broker Open Source* que pode ser instalado em um PC ou um *Raspberry Pi* e ser utilizado de modo local, outra opção é o *CloudMQTT* que é um *Broker* online, porém apresenta a desvantagem de ser pago. **Para esse projeto foi escolhido o *Mosquitto***, pois apresenta a vantagem de ser *Open Source*, ou seja, não é preciso pagar para utilizá-lo e também por que o

projeto foi feito de modo local.

Na figura 2 é possível ver de modo simplificado o funcionamento de um *Broker* por meio do *Prompt* de comando do Windows. A janela no lado esquerdo da figura 2 está funcionando como o *publisher*, publicando nos tópicos **topico1/dados** e **"topico2/dados"**, já no lado direito, as duas janelas estão funcionando como *subscriber*, sendo a de cima inscrita no tópico **"topico2/dados"** e a de baixo inscrita no **"topico1/dados"**.

Figura 2 – Exemplo do funcionamento do *Broker Mosquitto*

Fonte: Próprio Autor.

3 INTERFACE DO USUÁRIO

3.1 Introdução

O *Node-RED* é uma ferramenta de programação baseada em fluxo, criada na *International Business Machines Corporation* (IBM) (assim como o protocolo MQTT) em 2013 por Nick O'Leary e Dave Conway-Jones (OPENJS, 2020). Sua criação está diretamente ligada ao protocolo MQTT, pois sua função inicial era ser uma ferramenta de visualização e manipulação rápida de tópicos do protocolo MQTT, porém acabou se tornando bem mais do que isso, sendo atualmente uma ferramenta muito útil e utilizada em projetos de IoT (CLERISSI *et al.*, 2018), além disso, por ser uma plataforma *Open Source* e colaborativa, existem uma grande documentação e bibliotecas feitas por outros usuários, que são atualizadas frequentemente.

O *Node-RED*, por se tratar de uma ferramenta de programação, pode ser utilizado tanto no âmbito do *Backend* quanto no *Frontend* dos projetos, pois com ele é possível criar todo o fluxo de tratamento dos dados vindo dos diversos dispositivos ou equipamentos presentes no projeto, criar rotinas de controle manual ou mesmo automático para os dispositivos e também desenvolver toda a interface de usuário, onde serão apresentados dados importantes, controles, entre outras coisas (FERENCZ; DOMOKOS, 2019).

> "Podemos dizer que o *front-end* está ligado à parte visual de uma aplicação [...] "O *back-end* é o servidor, aquele que recebe requisições, realiza processamentos pesados e devolve informações para o cliente."
>
> (NOLETO, 2020).

Outro ponto que faz o *Node-RED* ser uma ferramenta bastante interessante é que por meio de bibliotecas é possível fazer conexão com servidores SQL, serviços baseados em nuvens, central de dados da própria empresa, entre outras, o que facilita a integração dos dispositivos com os setores de inteligência das empresas que utilizam os dados para tomada de decisões, além disso, também é possível fazer a integração dos sistemas desenvolvidos com assistentes virtuais, como por exemplo, a *Alexa*® e o *Google Home*®.

Por esses pontos citados acima, o *Node-RED* vem sendo apresentado em diversos trabalhos que relacionam seu uso com o ambiente industrial (FERENCZ; DOMOKOS, 2019), visto que facilita o tratamento e gerenciamento de dados de diversas fontes que vão ser utilizados nas industrias na tomada de decisão ou até mesmo para a comunicação entre dispositivos, e aliado ao MQTT, torna todo o processo simples e leve, não necessitando de dispositivos robustos,

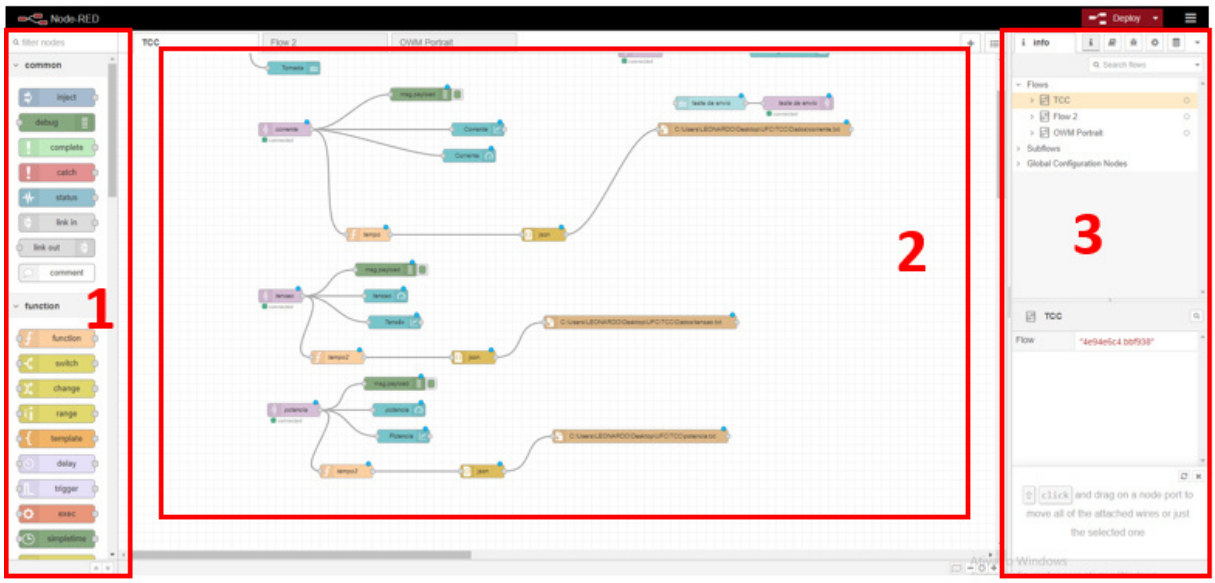
pois o MQTT não exige uma alta capacidade de processamento dos dispositivos, conforme mostrado no capítulo 2.

3.1.1 O Node-RED

Para iniciar a utilização do *Node-RED*, ele deve ser instalado no dispositivo que servirá como *broker* (o dispositivo pode ser um computador ou um raspberry, como mencionado no tópico 2.4). A programação no ambiente do *Node-RED* é feita por meio de blocos, os chamados **Nodes** (Nós), estes blocos nada mais são que conjuntos de códigos em *Javascript* que ao serem encaixados ajudam a criar a programação por meio de fluxos, o que torna o *Node-RED* uma plataforma de desenvolvimento tão intuitiva, além disso, também é possível criar blocos e programar blocos já existentes.

Na figura 3 é apresentada a interface de desenvolvimento do Node-red, na área lateral esquerda indicada por (1) é onde se encontram os Nodes (nós) que vêem pré-instalados ou que podem ser instalados depois, por meio de bibliotecas. Na parte central, indicada por (2) é a área onde é feito o desenvolvimento dos fluxos, utilizando os blocos da área (1). Por fim, na área lateral direita, indicada por (3), é onde as configurações do sistema são feitas, por exemplo, é onde pode ser baixado novos blocos e onde pode ser configurado a interface do usuário.

Figura 3 – Interface de desenvolvimento do *Node-RED*

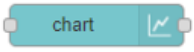
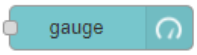
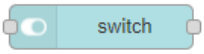
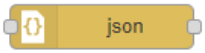
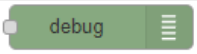


Fonte: Próprio Autor.

3.1.2 A interface

Para criação da Interface de Usuário (IU) foi necessário a instalação do pacote *node-red-dashboard* que adiciona uma grande quantidade de nodes voltados para a criação de uma interface para o usuário, alguns exemplos são os nodes que servem para criar gráficos, criar botões interativos, entre outras funções, os nodes utilizados para esse projeto estão listados na figura 4.

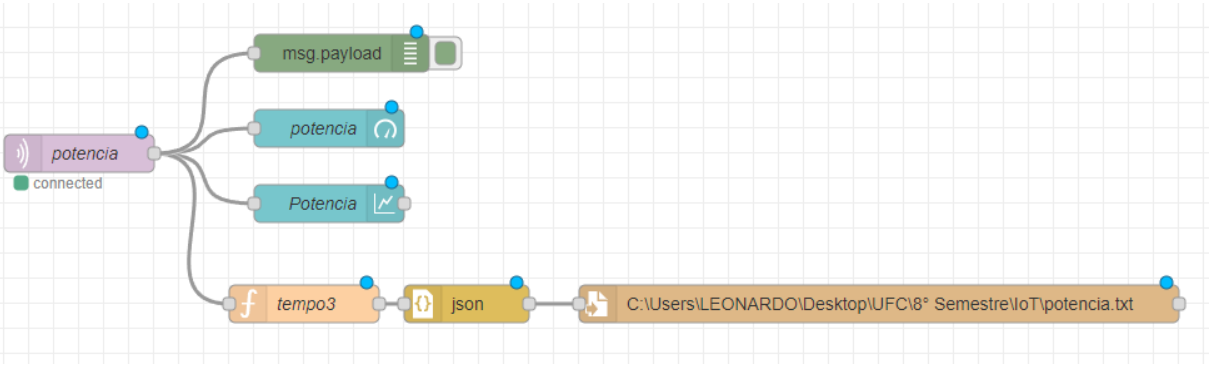
Figura 4 – Nós utilizados no projeto

Nodes (Nós)	Função
	Este nó se conecta no *MQTT Broker* como um *Subscriber* em um determinado tópico.
	Este nó se conecta no *MQTT Broker* como um Publisher em um determinado tópico.
	Cria um gráfico de linha com os valores que recebe.
	Cria um gráfico do tipo medidor com os valores que recebe.
	Adiciona um botão do tipo interruptor na UI
	Possibilita criação/utilização de funções com a linguagem Javascript
	Este nó transforma os dados para o modelo JSON.
	Transforma os dados/mensagem escolhida em um arquivo.
	Mostra as propriedades de uma mensagem.

Fonte: Próprio Autor.

Com a combinação desses nós foi possível criar os fluxos de funcionamento deste projeto, que além de criarem a interface do usuário também servem para receberem os dados e também enviar dados em tópicos específicos e assim fazer a comunicação do *dashboard* com os equipamentos, nesse projeto foram criados dois tipos de fluxos que serão descritos abaixo.

Figura 5 – Fluxo para tratamento e plotagem de dados



Fonte: Próprio Autor.

O fluxo da figura 5, por exemplo, recebe as informações do tópico **topico/potencia** que é referente aos dados de potência que o equipamento envia para o *broker* e plota um gráfico de linha por tempo desse dado, cria um medidor na interface de usuário que mostra o último dado recebido e além disso salva os dados na pasta escolhida em formato JSON e arquivo .txt.

Figura 6 – Fluxo para controle e envio de comandos para os equipamentos

Fonte: Próprio Autor.

O fluxo da figura 6, é um exemplo de um fluxo criado para controle e envio de informações para o equipamento inscrito no tópico que finaliza o fluxo. No caso desse fluxo em questão, foi utilizado o node que cria uma botão do tipo interruptor na interface do usuário, que envia a informação "Ligado"ou "Desligado"para o tópico de MQTT, no caso da figura, o tópico é o **topico/lampada**, na figura 7 pode ser visto o resultado do fluxo na interface do usuário.

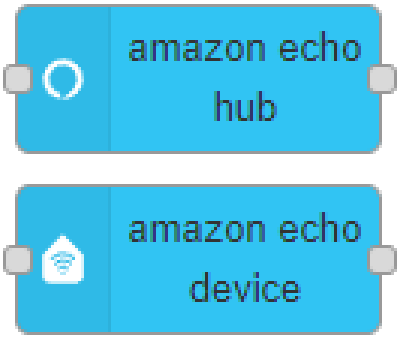
Figura 7 – Demonstração do fluxo de controle na interface de usuário.

Fonte: Próprio Autor.

3.2 Integração com assistente de voz

Como dito na introdução deste capítulo 3.1, também é possível fazer a integração da interface criado no *NODE-RED* com assistentes como a *Alexa*. Para fazer isso foi necessário instalar um pacote de nós chamado *node-red-contrib-amazon-echo*, este pacote apresenta dois novos nós que possibilitam a integração a assistente de voz. Os novos nós podem ser vistos na figura 8, o *Amazon Echo Hub* serve para simular um hub de dispositivos que suportam a integração com a *Alexa*, já o *Amazon Echo Device* simula os dispositivos.

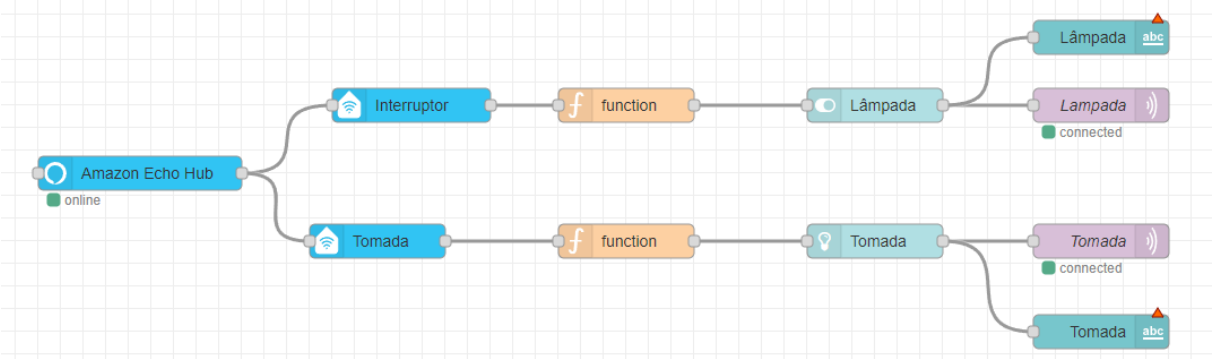
Figura 8 – Nós necessários para integração.



Fonte: Próprio Autor.

Para utilizar essa funcionalidade, primeiro é necessário configurar o *Amazon Echo Hub* para a mesma porta da rede que a utilizada pelos dispositivos mais atuais da *Amazon*, no caso essa porta é a 80, após isso, todos os dispositivos que serão integrados (*Amazon Echo Device*) devem ser conectados ao nó que cria o Hub, e sua configuração é bem simples, sendo necessário apenas colocar o nome que a *Alexa* deve reconhecer, após esse processo ao procurar dispositivos no app do assistente de voz, ele deverá reconhecer os dispositivos inseridos no *Node-RED*.

Figura 9 – Fluxo utilizado para integração.



Fonte: Próprio Autor.

A figura 9 apresenta como foi feito o fluxo para a integração, foram simulados dois dispositivos diferentes para a *Alexa*, um que representa o controle da lâmpada e outro que representa o controle da tomada, para realizar o controle é necessário apenas falar "Alexa, ligar (ou desligar) nome do dispositivo". Além disso, comparando com o fluxo de controle da figura 6, só foi necessário fazer a conexão dos novos nós que representam os dispositivos que já eram controlados manualmente, por conta disso, não foram necessárias modificações no código dos protótipos desenvolvidos, pois as informações dos tópicos do MQTT estão no fluxo da figura 6.

4 PROTOTIPAGEM

4.1 Introdução

Para demonstrar formas de como a interface criada por meio do *Node-RED* e como o MQTT é um protocolo bastante útil e versátil, foi feito a prototipagem de dois dispositivos para trabalhar em conjunto com o projeto deste trabalho de conclusão de curso. As ideias de protótipos pensados para este trabalho foram uma tomada IoT, um monitor de temperatura/umidade e também um interruptor IoT, importante ressaltar que a plataforma e o protocolo não se limitam aos microcontroladores utilizados nesse projeto e é possível aplica-los em diversas situações além da automação residencial, como por exemplo, monitoramento de geração de energia em fazendas de geração solar, monitoramento de geradores, entre outras coisas, dependendo apenas da escolha dos sensores que entregam as variáveis desejadas para monitoramento.

4.2 Tomada inteligente

Este dispositivo IoT tem como ideia principal disponibilizar ao usuário informações importantes de consumo de aparelhos eletrônicos, além de também garantir um controle desses aparelhos. O dispositivo foi desenvolvido para entregar dados de tensão (V), corrente (A) e potência (W) de aparelhos ligados a ele, também disponibilizando opção de ligá-lo e desligá-lo. Com essa tomada inteligente, o usuário pode entender o quanto certo eletrodoméstico afeta na sua conta de energia, além disso, utilizando a mesma ideia, porém com sensores diferentes, seria possível fazer um medidor digital para o quadro de distribuição de uma casa, por exemplo.

4.2.1 ESP32

Dentre as opções de placas de desenvolvimentos disponíveis no mercado, para esse dispositivo foi escolhido a placa *ESP32*, em relação aos Arduinos, apresenta a vantagem de ser menor e além disso já apresenta conectividade Wifi e bluetooth de fábrica, sem necessidade de utilização de modulos, além de apresentar um menor consumo. Além do arduino, também poderia ter sido escolhido a placa *NODEMCU ESP8266*, porém apesar de também possuir conectividade wifi de fábrica, apresenta menos entradas *GPIO* e possui apenas uma entrada analógica, o que impossibilita o projeto, pois tanto o sensor de tensão quanto o de corrente necessitam de entrada analógica, já o *ESP32* apresenta 18 entradas analógicas, por conta desses

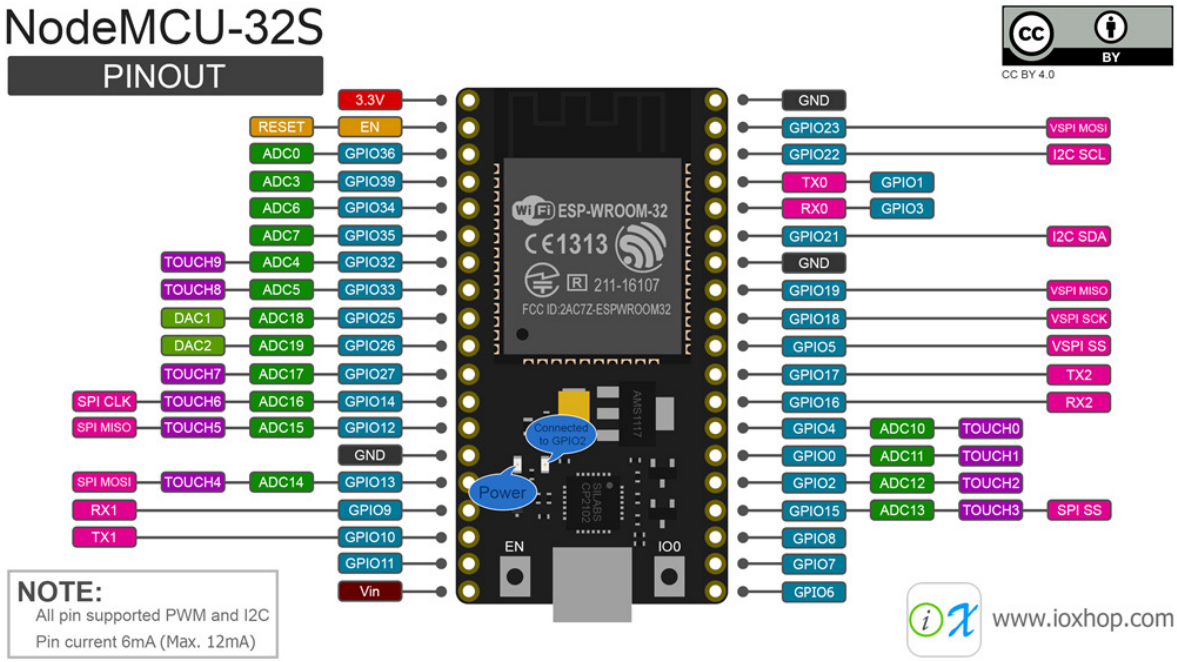
fatores, o *ESP32* foi a placa escolhida. A figura 10 mostra um comparativo entre as possíveis placas de desenvolvimento para esse projeto, é possível ver a superioridade da *ESP32*.

Figura 10 – Comparativo ESP32 x ESP83266 x Arduino UNO

	ESP32	ESP8266	ARDUINO UNO R3
Cores	2	1	1
Arquitetura	32 bits	32 bits	8 bits
Clock	160MHz	80MHz	16MHz
WiFi	Sim	Sim	Não
Bluetooth	Sim	Não	Não
RAM	512KB	160KB	2KB
FLASH	16Mb	16Mb	32KB
GPIO	36	17	14
Interfaces	SPI / I2C / UART / I2S / CAN	SPI / I2C / UART / I2S	SPI / I2C / UART
ADC	18	1	6
DAC	2	0	0

Fonte: <https://blogmasterwalkershop.com.br/embarcados/esp32/conhecendo-o-nodemcu-32s-esp32> - Acesso em: 05/01/2022

Figura 11 – Pinagem do ESP32



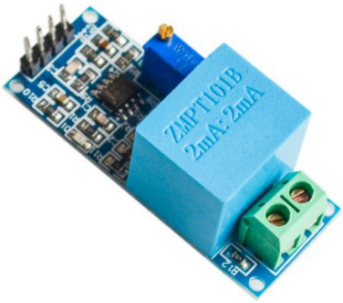
Fonte: <https://www.fernandok.com/2018/03/esp32-detalhes-internos-e-pinagem.html> - Acesso em: 05/01/2022

4.2.2 Sensores Utilizados e Outros Componentes

Para obter os dados de tensão foi utilizado o **módulo sensor de tensão AC ZMPT101B**, esse módulo consegue medir uma faixa de tensão alternada de 0 V - 250 V, com uma precisão

de ±1% e a tensão de entrada para seu funcionamento está na faixa DC de 5 V - 30 V. Para ser utilizado com o *ESP32*, foi necessário adaptar sua saída com a utilização um divisor de tensão (R2 = 10kΩ e R1 = 10kΩ) pois a entrada analógica do microcontrolador tem uma tensão máxima de 3,3 V.

Figura 12 – Módulo Sensor de Tensão AC ZMPT101B



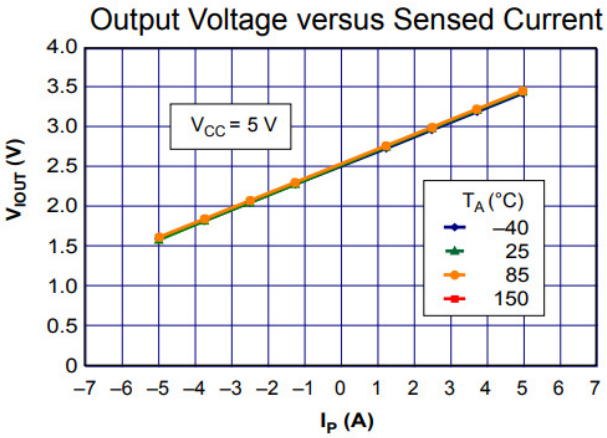
Fonte: <https://www.masterwalkershop.com.br/sensor-de-tensao-ac-0-a-250vac-zmpt101b>. Acesso em: 05/01/2022

Para os dados de corrente foi utilizado o **módulo sensor de corrente ACS712**, esse módulo está disponível para limites de corrente de **5A/20A/30A**, a princípio foi utilizado o de 20A, porém como o objetivo era medir a corrente de aparelhos domésticos, os valores medidos eram muito pequenos em relação ao fundo de escala, por conta disso foi substituído pelo de **5A**.

O módulo utiliza o chip **ACS712**, que permite detectar correntes AC ou DC por meio do efeito hall, conforme a folha de dados (ALLEGROMICROSYSTEMS, 2020), este sensor (na sua versão 5A) apresenta uma faixa de medição de **±5A** e uma margem de erro de **±1,5%**, além disso, sua alimentação deve ser de **5V**.

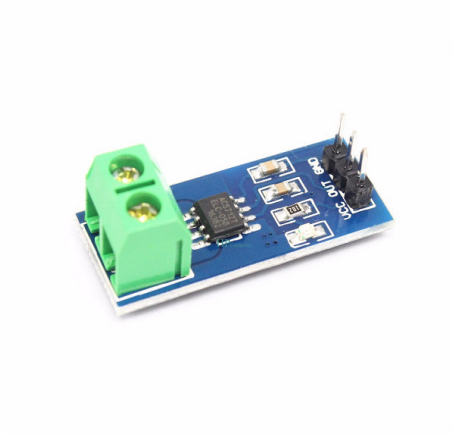
Conforme a figura 13, é possível ver que o *Offset* da corrente ocorre quando a tensão de saída é 2,5 V, além disso a faixa de tensão de saída vai de 1,5 V (-5 A) até 3,5 V (+5 A). Como a placa em questão apresenta resolução de 12 bits para sua entrada analógica (ESPRESSIF, 2020), ao realizarmos a leitura da porta analógica do ***ESP32***, na plataforma de programação, deverá ser mostrado um valor de 4095, que significa 2^{12} bits, no circuito em questão foi utilizado um divisor de corrente para que os 2,5 V do *offset* fossem 1,67 V (metade da tensão máxima que o ***ESP32*** pode ler), os valores dos resistores foram R2 = 2kΩ e R1 = 1kΩ.

Figura 13 – Tensão de saída x Corrente medida no ACS712 5A



Fonte: (ALLEGROMICROSYSTEMS, 2020)

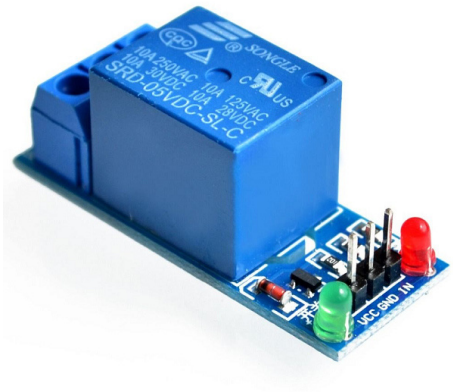
Figura 14 – Módulo Sensor de Corrente ACS712 5A



Fonte: <https://blogmasterwalkershop.com.br/arduino/como-usar-com-arduino-sensor-de-corrente-ac-e-dc-acs712-5a-20a-30a>. Acesso em: 05/01/2022

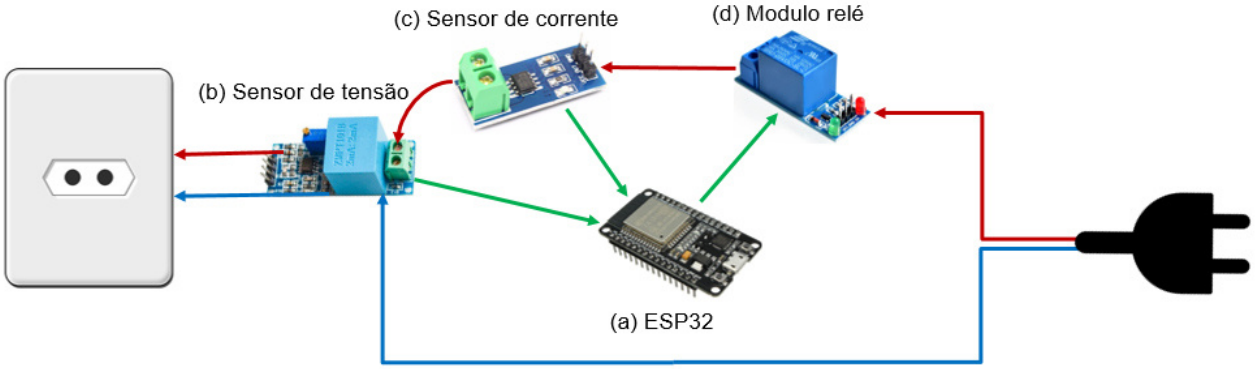
Além dos sensores de corrente e de tensão, foi utilizado um **módulo relé**, para que o usuário possa controlar o eletrodoméstico conectado na tomada por meio da plataforma escolhida (Computador ou Smartphone). O módulo em questão utiliza o relé **SRD-05VDC-SL-C**, que apresenta uma capacidade de **250 VAC/10 A** ou **30 VDC/10 A**, além disso necessita de uma tensão de no mínimo **3,3 V** para funcionar ((SONGLE, 2020)).

Figura 15 – Módulo Relé



Fonte: <https://blogmasterwalkershop.com.br/arduino/como-usar-com-arduino-modulo-rele-5v-1-canal>. Acesso em: 05/01/2022

Figura 16 – Esquemático Tomada Inteligente



Fonte: Próprio Autor

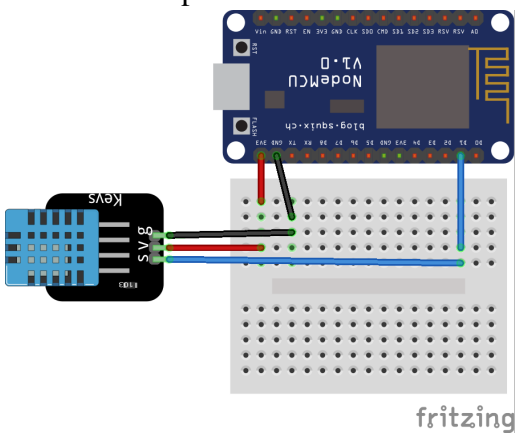
4.3 Monitor de Temperatura e Umidade

Outro dispositivo que foi pensado para ser utilizado nesse projeto foi um monitor de temperatura/umidade IoT, que pode ter diversas utilidades, por exemplo, caso seja instalado em uma sala que necessite de uma temperatura especifica, para conservar algum produto ou mesmo para o usuário ter noção da temperatura do ambiente ou se o ar está muito seco, além disso, em conjunto com um relé pode ser utilizado para automatizar um ar condicionado para ligar em uma determinada temperatura ambiente ou ligar um umidificador caso a umidade do ar esteja baixa.

A multinacional *Google* já apresenta soluções desse tipo como o *Nest Thermostat*, porém no Brasil seu valor é elevado e essa solução proposta nesse tópico apresenta um menor custo. O *Nest Thermostat* pode ser encontrado por um valor de $ 199,00 na *Amazon*, já para

produzir o monitor de Temperatura e Umidade o valor seria a soma de R$ 45,99 para o NodeMCU ESP8266 e R$ 26,99 para o sensor DHT11 (na loja FilipeFlop).

Figura 17 – Esquemático monitor de temperatura e umidade

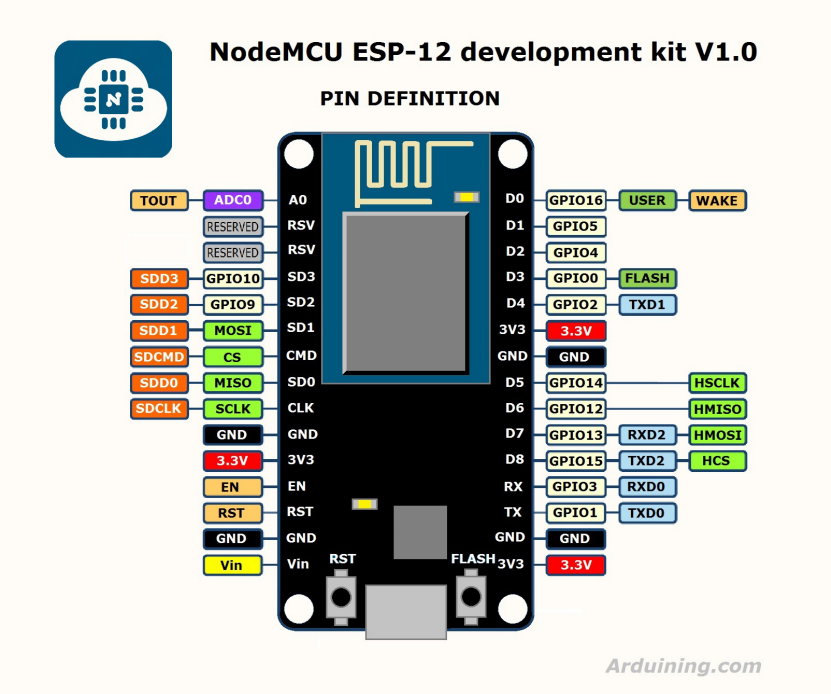


Fonte: Próprio Autor

4.3.1 NodeMCU ESP8266

Para esse protótipo foi escolhido a placa de desenvolvimento **NodeMCU ESP8266**, que assim como o *ESP32* também foi desenvolvido pela *Espressif*, como pode ser visto na figura 10, se trata de um microcontrolador menos robusto que o *ESP32*, porém apresenta quantidades de portas digitais (*General Purpose Input/Output* (GPIO)) suficientes para o projeto, além de também possuir conexão wifi nativa. A programação do **NodeMCU ESP8266** é praticamente igual ao do *ESP32*, podendo seguir basicamente a mesma lógica, além disso também é capaz de se comunicar por meio do protocolo MQTT, ou seja, por meio desse protocolo pode-se comunicar até mesmo as duas placas.

Figura 18 – Pinagem do NodeMCU ESP8266

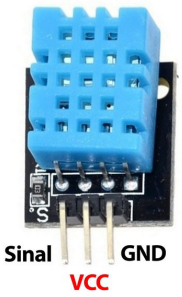


Fonte: <https://www.fernandok.com/2018/05/nodemcu-esp8266-detalhes-e-pinagem.html>. Acesso em: 05/01/2022

4.3.2 Sensor de Temperatura e Umidade DHT11

Para obter os dados de temperatura e umidade, foi utilizado o **sensor DHT11**, esse sensor de baixo custo possibilita a medição tanto da temperatura quanto da umidade, utilizando um único pino digital, o que torna sua utilização interessante em casos em que uma grande quantidade de pinos GPIO serão utilizados. Este sensor apresenta uma tensão de operação iniciando em 3,3 V, o que possibilita seu uso com o NodeMCU ESP8266, além disso pode medir temperaturas na faixa de 0 ºC até 50 ºC, com uma precisão de ±2º C, já na questão da umidade, suas medidas podem estar entre 20 % até 90 % de umidade do ar, com uma precisão de ±5% (MOUSER, 2020).

Figura 19 – Sensor de Temperatura e Umidade DHT11



Fonte: <https://www.eletrogate.com/tzk5zvewe-modulo-sensor-de-umidade-e-temperatura-dht11>. Acesso em: 05/01/2022

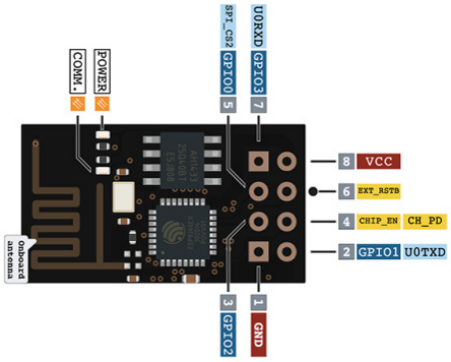
4.4 Interruptor Inteligente

Este dispositivo foi pensado com o objetivo de possibilitar o usuário ligar ou desligar luzes sem precisar ir até o interruptor, apenas utilizando o celular ou o computador onde tiver acesso à interface de usuária (tópico 3), porém sua função pode ser ampliada dependendo de como o usuário desejar, por exemplo, pode ser utilizado em parceria com os dados do monitor de temperatura/umidade (tópico 4.3) e ligar/desligar um ar condicionado ou ventilador de forma automática, toda comunicação feita por meio do protocolo MQTT.

4.4.1 ESP8266

O ESP8266 ou ESP01 é o microcontrolador utilizado para criar a placa de desenvolvimento NodeMCU ESP8266, por conta disso, é algo mais simples e menos robusto (apresenta apenas 2 pinos GPIO), porém com wifi nativo ainda presente e com a mesma lógica de programação do NodeMCU ESP8266 e *ESP32*.

Figura 20 – Microcontrolador ESP8266 (ESP01)

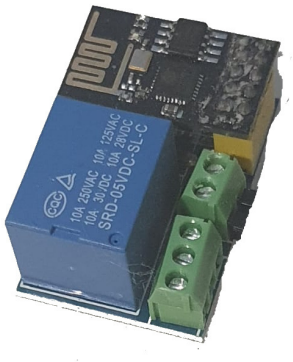


Fonte: <https://circuits4you.com/2016/12/14/esp8266-pin-diagram/>. Acesso em: 05/01/2022

4.4.2 Outros Componentes

Além do ESP8266, para desenvolvimento do interruptor inteligente foi utilizado um módulo relé próprio para o ESP8266, onde só é necessário encaixar o microcontrolador e energizar o módulo, o relé do modulo é o mesmo utilizado no tópico 4.2, o **SRD-05VDC-SL-C**, logo, suas características podem ser vistas no tópico 4.2.2. Para energizar o conjunto do módulo relé + ESP8266, foi utilizado uma mini fonte conversora aberta que converte de 220 V AC para 5 V DC, com uma potência máxima de 3,5 W.

Figura 21 – Módulo Relé com ESP8266



Fonte: Próprio Autor.

5 RESULTADOS

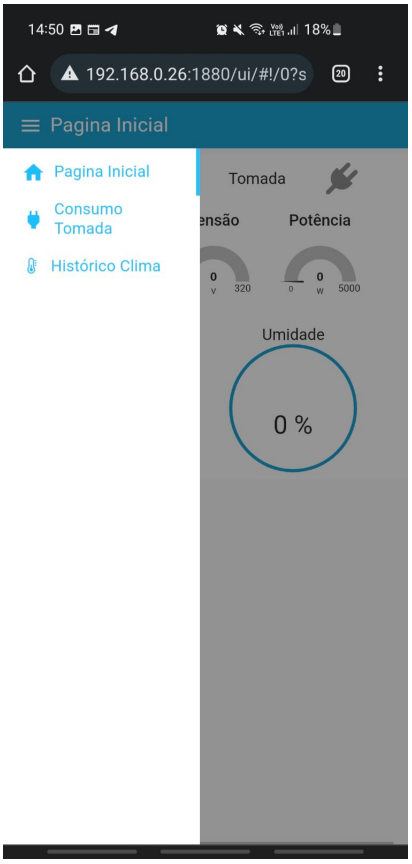
Nesse tópico serão apresentados os resultados referentes ao aplicativo desenvolvido utilizando a ferramenta do *Node-RED* e os dados obtidos pelos protótipos desenvolvidos. Está IU pode ser acessada tanto por um computador ou um celular conectado à mesma rede Wi-Fi que o computador no qual está instalado o *Node-RED* (tópico 3.1.1) e o *Broker* (tópico 2.4).

A interface foi dividida em três partes:

- Página inicial onde o usuário pode controlar os dispositivos (tomada IoT e Interruptor IoT) e também pode visualizar as leituras instantâneas da tomada IoT e do Monitor de temperatura e umidade.
- Página com as leituras da tomada IoT em relação ao tempo, sendo dividida em 3 quadros, Tensão x Tempo, Corrente x Tempo e Potência x Tempo.
- Página com as leituras do monitor de temperatura e umidade em relação ao tempo.

O usuário pode controlar as telas tanto acessando uma barra lateral que pode ser vista na figura 22, onde se encontram todas as páginas, quanto deslizando a tela para os lados.

Figura 22 – Barra Lateral na interface do usuário

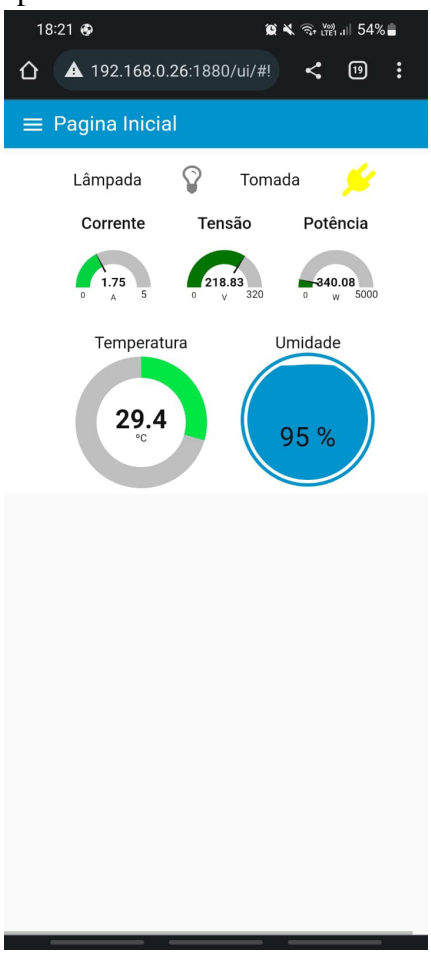


Fonte: Próprio Autor.

5.1 Página Inicial

A página inicial foi escolhida para apresentar opções rápidas ao usuário, podendo controlar com apenas um toque os dispositivos IoT e também poder fazer a leitura dos dados em tempo real. A figura 23 mostra como a página inicial se apresenta ao ser utilizado, no momento em questão o interruptor inteligente estava desligado e seu ícone se mostrava cinza, já a tomada estava ligada e seu ícone aparecia em amarelo, além disso pode ser visto a corrente, tensão, potência, temperatura e umidade registrados na hora da foto, nesse momento o carregador do celular estava conectado e foi registrado uma corrente próxima dos parâmetros do carregador. Além disso, a utilização dos comandos via *Alexa* (como visto no capítulo 3.2) permite o usuário controlar o interruptor inteligente e a tomada mesmo longe do app, as mudanças também são registradas no ícone dos dispositivos.

Figura 23 – Página Inicial do Aplicativo

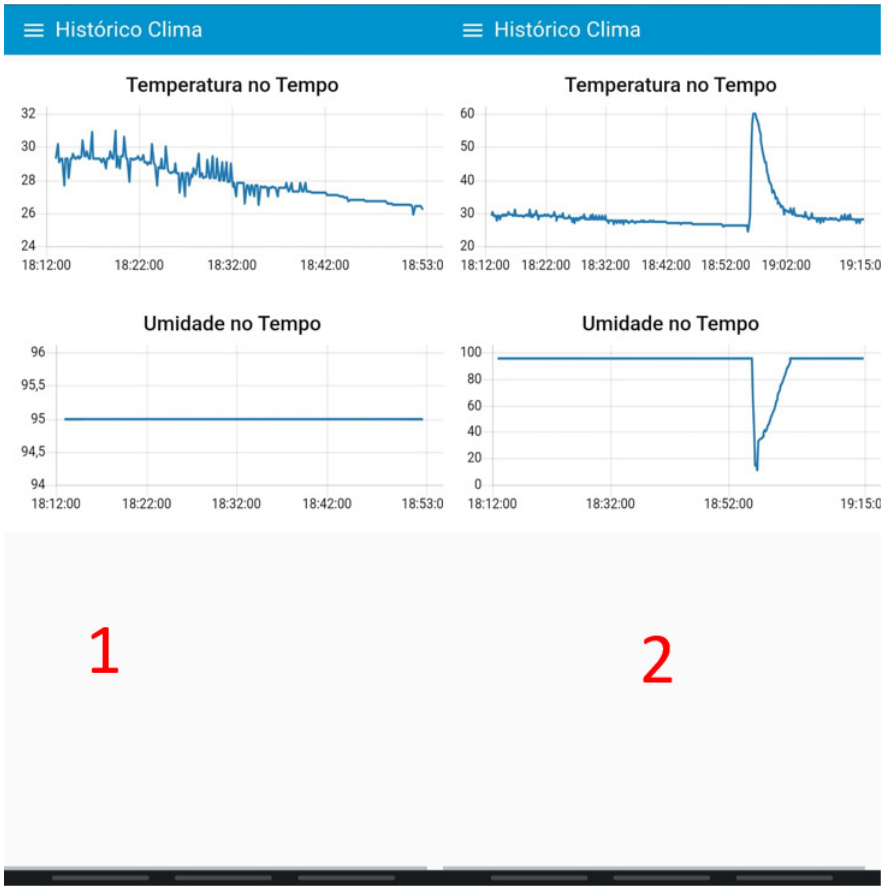


Fonte: Próprio Autor.

5.2 Histórico Monitor de Temperatura e Umidade

A página com o histórico do monitor de temperatura e umidade foi criada para mostrar as variações desses dois dados em um certo período de tempo, essa informação possibilita que a pessoa saiba como se comporta o clima no local desejado, por exemplo, em um frigorifico seria possível acompanhar as condições de temperatura e umidade de forma a realizar estudos que poderiam gerar economia para essa empresa, outra situação em que poderia ser utilizado é no estudo de instalação de placas fotovoltaicas, visto que a geração de energia é afetada pela temperatura.

Figura 24 – Histórico Monitor de Temperatura e Umidade



Fonte: Próprio Autor.

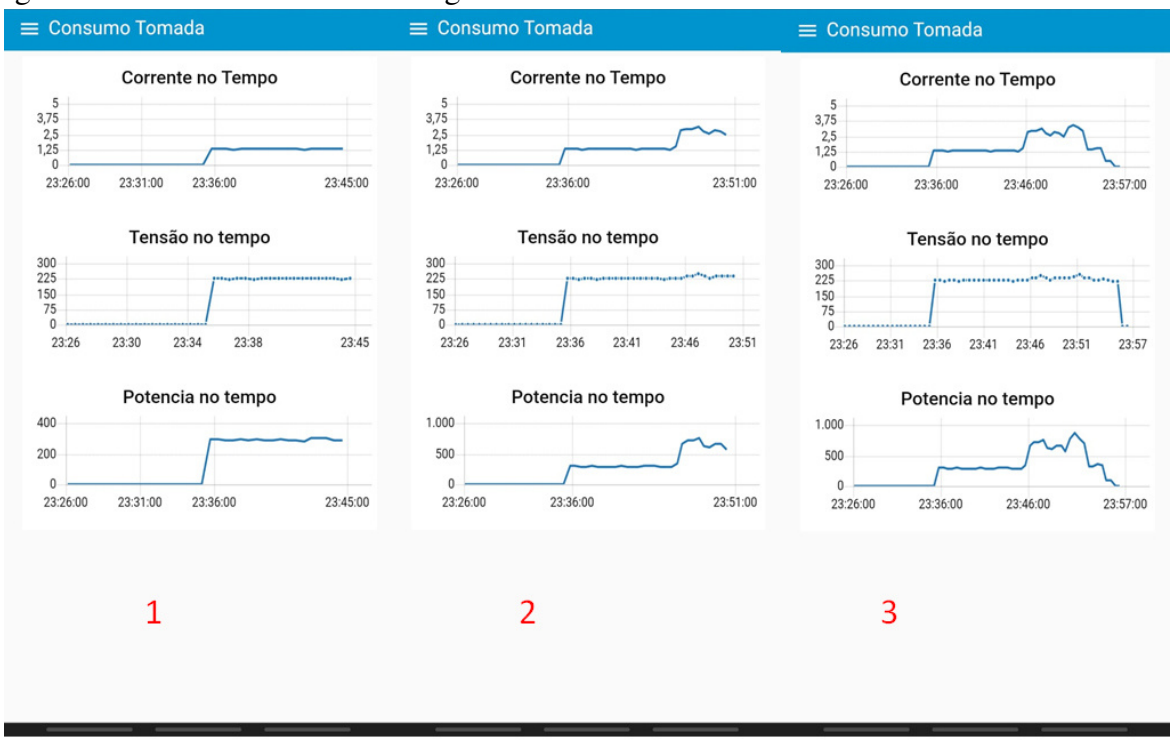
Na figura 24 é possível ver como os dados se comportaram em duas situações diferentes, no quadro 1 temos a variação de temperatura após um ar condicionado ser ligado, é possível notar que a temperatura ambiente cai depois de um tempo, já no quadro 2 é apresentado um pico de temperatura ao mesmo tempo que se nota um vale na umidade, esse comportamento ocorreu por ter sido utilizado um secador de cabelo para testar a variação da temperatura, além

disso é possível notar que após o secador ter sido desligado, tanto a umidade quanto a temperatura voltaram para as condições do início.

5.3 Histórico Tomada Inteligente

Está página tem como objetivo apresentar o histórico dos dados de tensão, corrente e potência em relação ao tempo, para isso o dispositivo faz a medição a cada 30 segundos e envia via protocolo MQTT para o computador que o *Node-RED* está instalado, onde será feito o tratamento via nós para os dados serem apresentados nos gráficos. Essa informação é útil em diversas ocasiões, por exemplo, ao fazer estudo da geração de energia renováveis em determinadas condições, observar o funcionamento de bombas em estações de tratamento de água, entre outras coisas.

Figura 25 – Histórico Tomada Inteligente



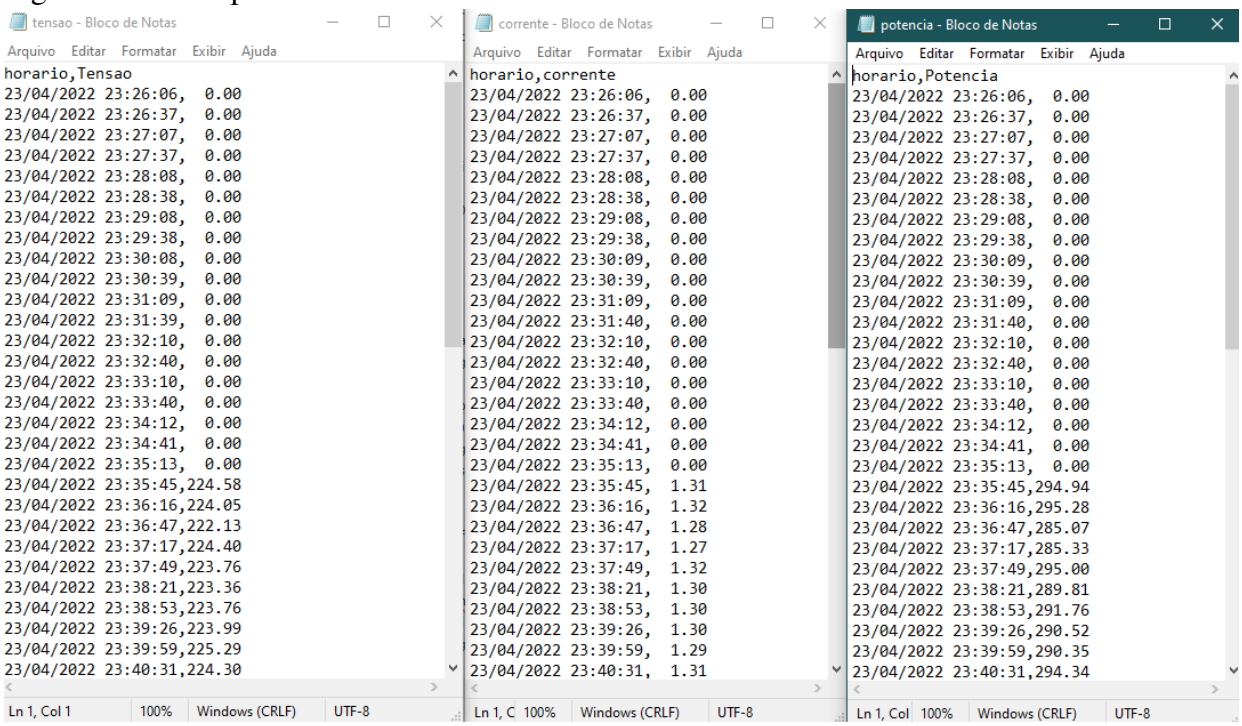
Fonte: Próprio Autor.

Na figura 25 é apresentado o funcionamento da tomada inteligente em 3 situações seguidas, no quadro 1 a tomada estava desligada (relé aberto) e logo em seguida foi ativada com uma televisão conectada na mesma, a fonte desta televisão apresentava a especificação de tensão 100-240 V e corrente de 1 A, pelo gráfico é possível ver que os dados estavam bem próximos, com pequenas variações, após isso (no quadro 2), foi feito a conexão com um ventilador, ocorreu

aumento na corrente e a tensão apresentou algumas variações, causadas provavelmente pelo tipo de carga que é o ventilador, por fim o quadro 3 mostra todo o período com a tomada sendo desligada no fim.

Os dados também são registrados e salvos no computador em arquivos de texto do tipo *csv*, possibilitando que o dispositivo funcione como um *Datalogger* e que esses dados possam ser utilizados futuramente em estudos com *softwares* como *PowerBI*, *Excel*, entre outros, sendo também possível utilizá-los em conjunto com *Python* ou *R* para estudos utilizando *Machine Learning*. Importante ressaltar que a utilização do *Node-RED* fez com que não fosse necessário utilizar um modulo para cartão SD (onde seriam registrados os dados) e o uso de um modulo de tempo para registrar as horas. Por fim, é possível visualizar o projeto funcionando na prática por meio do seguinte video: https://youtu.be/VT4Jp5bX_0A.

Figura 26 – Exemplo de dados salvos em CSV



tensao - Bloco de Notas

Arquivo Editar Formatar Exibir Ajuda

```
horario,Tensao
23/04/2022 23:26:06,  0.00
23/04/2022 23:26:37,  0.00
23/04/2022 23:27:07,  0.00
23/04/2022 23:27:37,  0.00
23/04/2022 23:28:08,  0.00
23/04/2022 23:28:38,  0.00
23/04/2022 23:29:08,  0.00
23/04/2022 23:29:38,  0.00
23/04/2022 23:30:08,  0.00
23/04/2022 23:30:39,  0.00
23/04/2022 23:31:09,  0.00
23/04/2022 23:31:39,  0.00
23/04/2022 23:32:10,  0.00
23/04/2022 23:32:40,  0.00
23/04/2022 23:33:10,  0.00
23/04/2022 23:33:40,  0.00
23/04/2022 23:34:12,  0.00
23/04/2022 23:34:41,  0.00
23/04/2022 23:35:13,  0.00
23/04/2022 23:35:45,224.58
23/04/2022 23:36:16,224.05
23/04/2022 23:36:47,222.13
23/04/2022 23:37:17,224.40
23/04/2022 23:37:49,223.76
23/04/2022 23:38:21,223.36
23/04/2022 23:38:53,223.76
23/04/2022 23:39:26,223.99
23/04/2022 23:39:59,225.29
23/04/2022 23:40:31,224.30
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

corrente - Bloco de Notas

Arquivo Editar Formatar Exibir Ajuda

```
horario,corrente
23/04/2022 23:26:06,  0.00
23/04/2022 23:26:37,  0.00
23/04/2022 23:27:07,  0.00
23/04/2022 23:27:37,  0.00
23/04/2022 23:28:08,  0.00
23/04/2022 23:28:38,  0.00
23/04/2022 23:29:08,  0.00
23/04/2022 23:29:38,  0.00
23/04/2022 23:30:09,  0.00
23/04/2022 23:30:39,  0.00
23/04/2022 23:31:09,  0.00
23/04/2022 23:31:40,  0.00
23/04/2022 23:32:10,  0.00
23/04/2022 23:32:40,  0.00
23/04/2022 23:33:10,  0.00
23/04/2022 23:33:40,  0.00
23/04/2022 23:34:12,  0.00
23/04/2022 23:34:41,  0.00
23/04/2022 23:35:13,  0.00
23/04/2022 23:35:45,  1.31
23/04/2022 23:36:16,  1.32
23/04/2022 23:36:47,  1.28
23/04/2022 23:37:17,  1.27
23/04/2022 23:37:49,  1.32
23/04/2022 23:38:21,  1.30
23/04/2022 23:38:53,  1.30
23/04/2022 23:39:26,  1.30
23/04/2022 23:39:59,  1.29
23/04/2022 23:40:31,  1.31
```

Ln 1, C 100% | Windows (CRLF) | UTF-8

potencia - Bloco de Notas

Arquivo Editar Formatar Exibir Ajuda

```
horario,Potencia
23/04/2022 23:26:06,  0.00
23/04/2022 23:26:37,  0.00
23/04/2022 23:27:07,  0.00
23/04/2022 23:27:37,  0.00
23/04/2022 23:28:08,  0.00
23/04/2022 23:28:38,  0.00
23/04/2022 23:29:08,  0.00
23/04/2022 23:29:38,  0.00
23/04/2022 23:30:09,  0.00
23/04/2022 23:30:39,  0.00
23/04/2022 23:31:09,  0.00
23/04/2022 23:31:40,  0.00
23/04/2022 23:32:10,  0.00
23/04/2022 23:32:40,  0.00
23/04/2022 23:33:10,  0.00
23/04/2022 23:33:40,  0.00
23/04/2022 23:34:12,  0.00
23/04/2022 23:34:41,  0.00
23/04/2022 23:35:13,  0.00
23/04/2022 23:35:45,294.94
23/04/2022 23:36:16,295.28
23/04/2022 23:36:47,285.07
23/04/2022 23:37:17,285.33
23/04/2022 23:37:49,295.00
23/04/2022 23:38:21,289.81
23/04/2022 23:38:53,291.76
23/04/2022 23:39:26,290.52
23/04/2022 23:39:59,290.35
23/04/2022 23:40:31,294.34
```

Ln 1, Col | 100% | Windows (CRLF) | UTF-8

Fonte: Próprio Autor.

6 CONCLUSÃO E TRABALHOS FUTUROS

6.1 Conclusão

Neste trabalho foi apresentado o conceito e a concepção de uma plataforma de automação residencial, capaz de tanto gerenciar quanto monitorar dispositivos via wi-fi, podendo também tratar os dados recebidos, tudo na rede. Além disso, também foi abordado o desenvolvimento de alguns dispositivos IoT, para serem utilizados em conjunto com está plataforma, foram desenvolvidos 3 dispositivos, sendo eles: monitor de temperatura e umidade (capítulo 4.3), tomada inteligente (capítulo 4.2) e o interruptor inteligente (capítulo 4.4).

O protocolo de comunicação MQTT encaixou muito bem na proposta do projeto, foi possível que os três dispositivos enviassem diversas informações (temperatura, umidade, tensão, corrente, potencia, além dos acionamentos) ao mesmo tempo sem problema, pois cada um desses dados apresenta um tópico especifico, sendo assim, não ocorre interferência, além disso é importante também ressaltar que os dispositivos apresentavam microcontroladores diferentes (*ESP8266* e *ESP32*) e isso não gerou problema de compatibilidade na comunicação. Outra vantagem importante observada é que a adição de novos dispositivos ou sensores não afetava os anteriores, sendo necessário apenas a criação de tópicos novos, além disso o protocolo MQTT se encaixa muito bem com o *Node-RED*.

Já em relação ao *Node-RED*, a plataforma de desenvolvimento se mostrou bastante intuitiva e com bastante recursos, não só permitindo a criação de uma interface de usuário como também o desenvolvimento do *Backend* da aplicação, ou seja, o tratamento dos dados, conexão com banco de dados ou armazenamento dos dados no computador, de forma simples, pois a programação nessa ferramenta pode ser feita por meio de blocos (nodes). A ferramenta se encaixa muito bem com o protocolo MQTT, os blocos (nodes) com integração ao protocolo já vem por padrão, além disso, é possível baixar outras bibliotecas e integrar o sistema com outras funcionalidades, como por exemplo, nesse trabalho o sistema foi integrado com a assistente de voz *Alexa*.

Na criação dos dispositivos, tanto o *ESP8266* quanto o *ESP32* se encaixaram muito bem com a proposta do projeto, pois ambos apresentam conectividade wifi de fábrica, no caso poderia ter sido utilizado *Arduino* também, porém ele não apresenta wifi, sendo necessário integrá-lo com outros dispositivos. O desenvolvimento do monitor de temperatura/umidade e do interruptor inteligente foi bem simples, porém a tomada inteligente apresentou alguns pontos de

dificuldades, pois ambos os sensores (tensão e corrente) foram feitos para trabalhar com uma tensão de 5 V, porém as portas analógicas de ambos os microcontroladores trabalham com no máximo 3,3 V, sendo necessário o tratamento da saída desses sensores para se adequar.

Por fim, a interação entre a plataforma e os dispositivos funcionou como o esperado, os dados foram apresentados tanto no histórico quanto em tempo real, o controle pela interface de usuário se mostrou eficaz e o controle via assistente de voz também funcionou como o esperado. Este projeto também mostra que é possível realizar automação residencial com dispositivos de baixo custo e com o usuário criando sua plataforma conforme seus desejos.

Este projeto teve o foco em relação à automação residencial, porém como o protocolo MQTT e o *Node-RED* não são restritos em relação aos microcontroladores ou dispositivos, essa solução pode ser aplicada em outros projetos de automação e internet das coisas, como por exemplo, o monitoramento de baixo custo de uma usina fotovoltaica ou monitoramento de estações de tratamento de água, sendo necessário apenas escolher os microcontroladores e sensores que melhor se encaixam no projeto.

6.2 Trabalhos Futuros

- *Utilizar um Raspberry PI como Broker*: tanto o *broker Mosquitto* quanto o Node-RED funcionam no *Raspberry PI* e com a utilização de uma tela sensível ao toque e alguns outros acessórios é possível criar uma central de controle/monitoramento.
- *Utilização da combinação MQTT e Node-RED no monitoramento de uma usina fotovoltaica*: como visto no decorrer do texto, a solução apresentada não é restrita a um hardware ou tipo de sensor, portanto, com sensores adequados (sensor de tensão, sensor de corrente, luminosidade, entre outros) é possível fazer o controle e monitoramento de uma usina fotovoltaica.
- *Utilizar a compatibilidade do Node-RED com bancos de dados*: o Node-RED permite a conexão com bancos de dados, sendo assim, é possível a criação de uma *Data Warehouse* com os dados obtidos no projeto, podendo utiliza-los até para previsão ou machine learning.
- *Estudos sobre segurança e criptografia no protocolo MQTT*: Buscar entender meios para que o sistema desenvolvido esteja protegido contra ataques hackers ou vazamento dos dados que estão sendo repassados.

REFERÊNCIAS

ALLEGROMICROSYSTEMS. **DATASHEET ACS712**. 2020. Disponível em: <https://www.allegromicro.com/-/media/files/datasheets/acs712-datasheet.ashx>. Acesso em: 05 jan. 2022.

ATMOKO, R. A.; RIANTINI, R.; HASIN, M. K. Iot real time data acquisition using mqtt protocol. **Journal of Physics: Conference Series**, IOP Publishing Ltd, 2017.

CLERISSI, D.; LEOTTA, M.; REGGIO, G.; RICCA, F. Towards an approach for developing and testing node-red iot systems. **ACM SIGSOFT International Workshop on EnsembleBased Software Engineering**, 2018.

COELHO, A. D.; DIAS, B. G.; ASSIS, W. de O.; MARTINS, F. de A.; PIRES, R. C.; KUKE, A. da S. Monitoring of soil and atmospheric sensors with internet of things (iot) applied in precision agriculture monitorização de sensores do solo e atmosféricos com internet das coisas (iot) aplicados em agricultura de precisão. **Brazilian Journal of Development**, v. 8, n. 3, p. 16453–16465, 2022.

ESPRESSIF. **DATASHEET ESP32**. 2020. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>. Acesso em: 05 jan. 2022.

FERENCZ, K.; DOMOKOS, J. Using node-red platform in an industrial environment. **XXXV. Jubileumi Kandó Konferencia**, 2019.

FERNANDES, A. F. 5g e internet das coisas. **BIUS-Boletim Informativo Unimotrisaúde em Sociogerontologia**, v. 31, n. 25, p. 1–3, 2022.

LAMPKIN, V.; LEONG, W. T.; OLIVERA, L.; RAWAT, S.; SUBRAHMANYAM, N.; XIANG, R. **Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry**. [*S. l.*]: International Technical Support Organization, 2012. v. 1.

MOUSER. **DATASHEET DHT11**. 2020. Disponível em: <https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf>. Acesso em: 05 jan. 2022.

MQTT Use Cases. 2020. Casos de uso do MQTT. Disponível em: https://mqtt.org/use-cases/. Acesso em: 07 nov. 2021.

NOLETO, C. **O que é Front-end e back-end e qual a melhor opção?** 2020. Disponível em: <https://www.alura.com.br/artigos/o-que-e-front-end-e-back-end>. Acesso em: 10 jun. 2022.

OPENJS, F. **Node-RED About**. 2020. Sobre Node-RED. Disponível em: <https://nodered.org/about/>. Acesso em: 02 nov. 2021.

ORACLE. **O que é Internet of Things (IoT)?** 2022. Disponível em: <https://www.oracle.com/br/internet-of-things/what-is-iot/>. Acesso em: 12 jun. 2022.

SONGLE. **DATASHEET SRD05VDCSLC**. 2020. Disponível em: <https://pdf1.alldatasheet.com/datasheet-pdf/view/1131944/SONGLERELAY/SRD05VDCSLC.html>. Acesso em: 05 jan. 2022.

TURETTA, C. d. M.; TOKIMATSU, T. G. Indústria 4.0: implicações da aplicação no período da pandemia de covid-19. Universidade Presbiteriana Mackenzie, 2022.

APÊNDICE A – CÓDIGOS UTILIZADOS NO PROJETO

Código-fonte 1 – Código interruptor IoT

```
1  //BIBLIOTECA PARA WIFI
2  #include <ESP8266WiFi.h>
3  //BIBLIOTECA PARA O MQTT
4  #include <PubSubClient.h>
5  //DECLARACAO DOS PINOS
6  const int lamp = 0;
7  //DECLARACAO DE VARIAVEIS UTILIZADAS NO DECORRER DO CODIGO
8  long now = millis();
9  long lastMeasure = 0;
10 int control_rele;
11 int control_anterior = 0;
12 typedef struct struct_message
13 {
14     float temperatura;
15     float umidade;
16 } struct_message;
17
18 struct_message incomingReadings;
19
20 //DECLARACAO WIFI E MQTT
21 WiFiClient espClient;
22 PubSubClient client(espClient);
23
24 //DADOS WIFI
25 const char *ssid = "nome do wifi";
26 const char *password = "senha do wifi";
27 const char *mqtt_server = "ip do broker";
28
```

```
29 //FUNCAO PARA CONFIGURACAO DO WIFI DE ACORDO COM OS DADOS
       ACIMA.
30 void setup_wifi()
31 {
32     delay(10);
33     WiFi.mode(WIFI_AP_STA);
34     Serial.println();
35     Serial.print("Conectando na rede: ");
36     Serial.println(ssid);
37     WiFi.begin(ssid, password);
38
39     while (WiFi.status() != WL_CONNECTED)
40     {
41         delay(500);
42         Serial.print(".");
43     }
44     Serial.println("");
45     Serial.print("Conectado na rede - ESP IP: ");
46     Serial.println(WiFi.localIP());
47 }
48
49 //FUNCAO PARA IDENTIFICAR O TOPICO E A MENSAGEM RECEBIDA.
50 void callback(String topic, byte *message, unsigned int
       length)
51 {
52     Serial.print("Mensagem recebida no topico: ");
53     Serial.print(topic);
54     Serial.print(". Mensagem: ");
55     String messageTemp;
56
57     for (int i = 0; i < length; i++)
58     {
```

```
59          Serial.print((char)message[i]);
60          messageTemp += (char)message[i];
61      }
62      Serial.println();
63
64      //ACIONAMENTO DE ACORDO COM O TOPICO ESCOLHIDO
65      if (topic == "topico/lampada")
66      {
67          Serial.print("Mudando o status do rele para: ");
68          if (messageTemp == "Ligado")
69          {
70              control_rele = 1;
71          }
72          else if (messageTemp == "Desligado")
73          {
74              control_rele = 0;
75          }
76      }
77      Serial.println();
78  }
79
80  //FUNCAO PARA SE CONECTAR AO BROKER MQTT (MOSQUITTO)
81  void reconnect()
82  {
83
84      while (!client.connected())
85      {
86          Serial.print("Conectando-se ao broker");
87
88          if (client.connect("ESP01Client")) /*O DISPOSITIVO
                FOI NOMEADO NO BROKER COMO ESP32Client, cada
                dispositivo deve ter um identificador*/
```

```
89          {
90              Serial.println("conectado");
91              //NESSE MOMENTO E INDICADO EM QUAL TOPICO O
                   DISPOSITIVO VAI SE INSCREVER.
92              client.subscribe("topico/lampada");
93              //CASO EXISTA FALHA NA CONEXAO, ELE TENTARA SE
                   RECONECTAR EM 5 SEGUNDOS
94          }
95          else
96          {
97              Serial.print("falha, rc=");
98              Serial.print(client.state());
99              Serial.println(" Tentando novamente em 5
                   segundos");
100
101             delay(5000);
102         }
103     }
104 }
105
106 void setup()
107 {
108     //PINO DO RELE DECLARADO COMO SAIDA
109     pinMode(lamp, OUTPUT);
110     Serial.begin(115200);
111     setup_wifi();
112     client.setServer(mqtt_server, 1883);
113     client.setCallback(callback);
114 }
115
116 void loop()
117 {
```

```
118     //FUNCAO PARA VERIFICAR SE O DISPOSITIVO AINDA ESTA
            CONECTADO DURANTE O FUNCIONAMENTO E RECONECTAR.
119     if (!client.connected())
120     {
121         reconnect();
122     }
123     if (!client.loop())
124         client.connect("ESP8266Client");
125
126     //CASES PARA O CONTROLE DA LAMPADA CONFORME MENSAGEM
            RECEBIDA DO BROKER
127     if (control_rele != control_anterior)
128     {
129         if (control_rele == 1)
130         {
131             digitalWrite(lamp, HIGH);
132             Serial.println("Acesso");
133         }
134         if (control_rele == 0)
135         {
136             digitalWrite(lamp, LOW);
137             Serial.println("Desligado");
138         }
139         control_anterior = control_rele;
140     }
141 }
```

Código-fonte 2 – Monitor de Temperatura e Umidade

```
1  //BIBLIOTECA PARA WIFI
2  #include <ESP8266WiFi.h>
3  //BIBLIOTECA PARA O MQTT
4  #include <PubSubClient.h>
5  //BIBLIOTECA PARA O SENSOR DE TEMPERATURA/UMIDADE
6  #include "DHT.h"
7
8
9  //DEFININDO O MODELO DO SENSOR
10 #define DHTTYPE DHT11   // DHT 11
11
12 //DEFININDO PINO DO SENSOR
13 const int DHTPin = 5;
14
15 //DECLARACAO DE VARIAVEIS UTILIZADAS NO DECORRER DO CODIGO
16 float h;
17 float t;
18 typedef struct struct_message {
19   float temperatura;
20   float umidade;
21 } struct_message;
22 struct_message incomingReadings;
23 long now = millis();
24 long lastMeasure = 0;
25
26 //DECLARACAO WIFI E MQTT
27 WiFiClient espClient;
28 PubSubClient client(espClient);
29
30 //CRIANDO A INSTANCIA DO SENSOR
31 DHT dht(DHTPin, DHTTYPE);
```

```
32
33 //DADOS WIFI
34 const char* ssid = "WiFiCasa2ghz";
35 const char* password = "dextherabc123";
36 const char* mqtt_server = "192.168.0.26";
37
38 //FUNCAO PARA CONFIGURACAO DO WIFI DE ACORDO COM OS DADOS
      ACIMA.
39 void setup_wifi() {
40   delay(10);
41
42   WiFi.mode(WIFI_AP_STA);
43   Serial.println();
44   Serial.print("Conectando na rede: ");
45   Serial.println(ssid);
46   WiFi.begin(ssid, password);
47   while (WiFi.status() != WL_CONNECTED) {
48     delay(500);
49     Serial.print(".");
50   }
51   Serial.println("");
52   Serial.print("Conectado na rede - ESP IP: ");
53   Serial.println(WiFi.localIP());
54 }
55
56 //FUNCAO PARA IDENTIFICAR O TOPICO E A MENSAGEM RECEBIDA.
57 void callback(String topic, byte* message, unsigned int
      length) {
58   Serial.print("Mensagem recebida no topico: ");
59   Serial.print(topic);
60   Serial.print(". Mensagem: ");
61   String messageTemp;
```

```
62
63
64    for (int i = 0; i < length; i++) {
65      Serial.print((char)message[i]);
66      messageTemp += (char)message[i];
67    }
68  }
69
70  //FUNCAO PARA SE CONECTAR AO BROKER MQTT (MOSQUITTO)
71  void reconnect() {
72    while (!client.connected()) {
73      Serial.print("Conectando-se ao broker");
74
75      if (client.connect("ESP8266Client"))/*O DISPOSITIVO FOI
            NOMEADO NO BROKER COMO ESP32Client, cada
           dispositivo deve ter um identificador*/ {
76        Serial.println("conectado");
77
78      } else {
79        Serial.print("falha, rc=");
80        Serial.print(client.state());
81        Serial.println(" Tentando novamente em 5 segundos");
82
83        delay(5000);
84      }
85    }
86  }
87
88
89  void setup() {
90    //FUNCAO PARA CONFIGURAR O SENSOR DE TEMPERATURA/UMIDADE
91    dht.begin();
```

```
92
93    Serial.begin(115200);
94    setup_wifi();
95    client.setServer(mqtt_server, 1883);
96    client.setCallback(callback);
97
98  }
99
100
101 void loop() {
102   //FUNCAO PARA VERIFICAR SE O DISPOSITIVO AINDA ESTA
        CONECTADO DURANTE O FUNCIONAMENTO E RECONECTAR.
103   if (!client.connected()) {
104     reconnect();
105   }
106   if(!client.loop())
107     client.connect("ESP8266Client");
108
109   now = millis();
110   //A MEDICAO E FEITA A CADA 10 SEGUNDOS
111   if (now - lastMeasure > 10000) {
112     lastMeasure = now;
113     //FUNCOES PARA LER A UMIDADE E TEMPERATURA
114     h = dht.readHumidity();
115     t = dht.readTemperature();
116
117     //CASO OCORRA ERRO NA LEITURA SERA APRESENTADO NO
          PAINEL
118     if (isnan(h) || isnan(t)) {
119       Serial.println("Error ao ler dados do DHT11");
120       return;
121     }
```

```
122
123
124     //TRATAMENTO PARA ENVIAR OS DADOS DE TEMPERATURA E
           UMIDADE PARA O BROKER
125     //TRANSFORMA OS DADOS FLOAT EM CHAR
126     static char temperatureTemp[7];
127     dtostrf(t, 6, 2, temperatureTemp);
128     static char humidityTemp[7];
129     dtostrf(h, 6, 2, humidityTemp);
130
131     //OS DADOS SAO PUBLICADOS PARA O BROKER NOS TOPICOS
           ABAIXO
132     client.publish("topico/temperatura", temperatureTemp);
133     client.publish("topico/umidade", humidityTemp);
134
135     delay(2000);
136   }
137 }
```

Código-fonte 3 – Código Tomada IoT

```
1    //BIBLIOTECAS PARA WIFI
2  #include <WiFi.h>
3  #include <WiFiClient.h>
4  //BIBLIOTECA PARA O MQTT
5  #include <PubSubClient.h>
6  //BIBLIOTECA PARA OS SENSORES DE TENSAO E CORRENTE
7  #include "EmonLib.h"
8
9  //DEFINIR VALOR PARA CALIBRAR A MEDIDA DA TENSAO
10 #define VOLT_CAL 1200
11 //DEFINIR VALOR PARA CALIBRAR A MEDIDA DA CORRENTE
```

```
12 #define CURRENT_CAL 35
13 EnergyMonitor emon1;
14
15 //DECLARACAO DOS PINOS
16 const int pinoSensor_tensao = 32;
17 const int pinoSensor_corrente = 34;
18 const int lamp = 25;
19
20 //DECLARACAO DE VARIAVEIS UTILIZADAS NO DECORRER DO CODIGO
21 float resolucao  = 3.3 / 4096;
22 static char tensaoTemp[7] = "0";
23 static char correnteTemp[7] = "0";
24 static char powerTemp[7] = "0";
25 float corrente;
26 float c;
27 int control_rele;
28 int control_anterior = 0;
29 float h;
30 float t;
31 long now = millis();
32 long lastMeasure = 0;
33 long lastMeasure2 = 0;
34 typedef struct struct_message {
35   float tensao;
36   float corrente;
37 } struct_message;
38
39 struct_message incomingReadings;
40
41 //DECLARACAO WIFI E MQTT
42 WiFiClient espClient;
43 PubSubClient client(espClient);
```

```
44
45 //DADOS WIFI
46 const char* ssid = "nome do wifi";
47 const char* password = "senha do wifi";
48 const char* mqtt_server = "ip do dispositivo";
49
50 //FUNCAO PARA CONFIGURACAO DO WIFI DE ACORDO COM OS DADOS
      ACIMA.
51 void setup_wifi() {
52   delay(10);
53
54   WiFi.mode(WIFI_AP_STA);
55   Serial.println();
56   Serial.print("Conectando na rede: ");
57   Serial.println(ssid);
58   WiFi.begin(ssid, password);
59   while (WiFi.status() != WL_CONNECTED) {
60     delay(500);
61     Serial.print(".");
62   }
63   Serial.println("");
64   Serial.print("Conectado na rede - ESP IP: ");
65   Serial.println(WiFi.localIP());
66 }
67
68 //FUNCAO PARA IDENTIFICAR O TOPICO E A MENSAGEM RECEBIDA.
69 void callback(String topic, byte* message, unsigned int
      length) {
70   Serial.print("Mensagem recebida no topico: ");
71   Serial.print(topic);
72   Serial.print(". Mensagem: ");
73   String messageTemp;
```

```
74
75    for (int i = 0; i < length; i++) {
76      Serial.print((char)message[i]);
77      messageTemp += (char)message[i];
78    }
79
80    Serial.println();
81
82    //ACIONAMENTO DE ACORDO COM O TOPICO ESCOLHIDO
83    if (topic == "topico/tomada") {
84      Serial.print("Mudando o status do rele para: ");
85      if (messageTemp == "Desligado") {
86        control_rele = 0;
87      }
88      else if (messageTemp == "Ligado") {
89        control_rele = 1;
90
91      }
92    }
93    Serial.println();
94  }
95
96
97  //FUNCAO PARA SE CONECTAR AO BROKER MQTT (MOSQUITTO)
98  void reconnect() {
99
100   while (!client.connected()) {
101     Serial.print("Conectando-se ao broker");
102
103     if (client.connect("ESP32Client")) { /*O DISPOSITIVO
          FOI NOMEADO NO BROKER COMO ESP32Client, cada
          dispositivo deve ter um identificador*/
```

```
104         Serial.println("conectado");
105         //NESSE MOMENTO E INDICADO EM QUAL TOPICO O
               DISPOSITIVO VAI SE INSCREVER.
106         client.subscribe("topico/tomada");
107
108         //CASO EXISTA FALHA NA CONEXAO, ELE TENTARA SE
               RECONECTAR EM 5 SEGUNDOS
109       } else {
110         Serial.print("falha, rc=");
111         Serial.print(client.state());
112         Serial.println(" Tentando novamente em 5 segundos");
113
114         delay(5000);
115       }
116     }
117 }
118
119
120 void setup() {
121   //PINO DO RELE DECLARADO COMO SAIDA
122   pinMode(lamp, OUTPUT);
123   //E FEITO A PASSAGEM DOS PARAMETROS PARA O CALCULO DA
        TENSAO E DA CORRENTE
124   emon1.voltage(pinoSensor_tensao, VOLT_CAL, 1.7);
125   emon1.current(pinoSensor_corrente, CURRENT_CAL);
126
127   Serial.begin(115200);
128   setup_wifi();
129   client.setServer(mqtt_server, 1883);
130   client.setCallback(callback);
131 }
132
```

```
133
134 void loop() {
135   //FUNCAO PARA VERIFICAR SE O DISPOSITIVO AINDA ESTA
        CONECTADO DURANTE O FUNCIONAMENTO E RECONECTAR.
136   if (!client.connected()) {
137     reconnect();
138   }
139   if (!client.loop())
140     client.connect("ESP32Client");
141
142   now = millis();
143   //A CADA 0,1 SEGUNDOS SERA FEITO A MEDICAO
144   if (now - lastMeasure > 100) {
145     lastMeasure = now;
146     //FUNCAO PARA CALCULO DA CORRENTE E TENSAO RMS,
          PRIMEIRO PARAMETRO E O NUMERO DE SEMICICLOS E O
          SEGUNDO O TEMPO DE MEDICAO
147     emon1.calcVI(20, 2000);
148     //VARIAVEL RECEBE O VALOR DE TENSAO RMS OBTIDO
149     float t    = emon1.Vrms;
150     //VARIAVEL RECEBE O VALOR DE CORRENTE RMS OBTIDO
151     c = emon1.Irms;
152
153     //FUNCAO PARA TRATAMENTO DE RUIDOS
154     if (control_rele == 0) {
155       c = 0;
156       t = 0;
157     } else {
158       c = emon1.Irms;
159       float t    = emon1.Vrms;
160     }
161     corrente = c - 0.6;
```

```
162     if (corrente < 0) {
163       c = 0;
164     }
165     else {
166       c = corrente;
167     }
168 
169     //CALCULO DA POTENCIA.
170     float p = c * t;
171 
172     //TRATAMENTO DAS VARIAVEIS DE TENSAO, CORRENTE E
          POTENCIA PARA SER PUBLICADO NOS TOPICOS RESPECTIVOS
173     //A FUNCAO TRANSFORMA AS VARIAVEIS QUE SAO DO TIPO
          FLOAT EM CHAR
174     dtostrf(t, 6, 2, tensaoTemp);
175     dtostrf(c, 6, 2, correnteTemp);
176     dtostrf(p, 6, 2, powerTemp);
177 
178     //PRINTA OS DADOS SERIAL DO ARDUINO PARA ACOMPANHAMENTO
          .
179     Serial.print("tensao AC: ");
180     Serial.print(t);
181     Serial.print(" V");
182     Serial.println();
183     Serial.print("Corrente AC: ");
184     Serial.print(c);
185     Serial.print(" A");
186     Serial.println();
187     Serial.print("Potencia: ");
188     Serial.print(p);
189     Serial.println(" W");
190 
```

```
191   }
192
193   //CONTROLE DO RELE CONFORME MENSAGEM RECEBIDA DO BROKER
194   if (control_rele != control_anterior) {
195     if (control_rele == 1) {
196       digitalWrite(lamp, HIGH);
197     }
198     if (control_rele == 0) {
199       client.publish("topico/corrente", 0);
200       digitalWrite(lamp, LOW);
201     }
202     control_anterior = control_rele;
203   }
204
205   //ENVIA A CADA 30 SEGUNDOS OS DADOS PARA COMPOR OS
        GRAFICOS
206   if (now - lastMeasure2 > 30000) {
207     lastMeasure2 = now;
208     Serial.println("ENVIADO");
209     //FUNCAO RESPONSAVEL POR PUBLICAR UM DADO EM UM TOPICO
210     client.publish("topico/tensaograph", tensaoTemp);
211     client.publish("topico/correntegraph", correnteTemp);
212     client.publish("topico/potencia", powerTemp);
213
214   }
215   //FUNCAO RESPONSAVEL POR PUBLICAR UM DADO EM UM TOPICO
216   client.publish("topico/tensao", tensaoTemp);
217   client.publish("topico/corrente", correnteTemp);
218
219 }
```