INSTITUTO FEDERAL DE SANTA CATARINA

FABIANO KRAEMER

Dispositivo IoT para monitoramento e diagnóstico de sistemas de refrigeração

São José - SC

Julho/2022

DISPOSITIVO IOT PARA MONITORAMENTO E DIAGNÓSTICO DE SISTEMAS DE REFRIGERAÇÃO

Monografia apresentada ao Curso de Engenharia de Telecomunicações do campus São José do Instituto Federal de Santa Catarina para a obtenção do diploma de Engenheiro de Telecomunicações.

Orientador: Professor Arliones Stevert Hoeller Junior, Dr.

Coorientador: Professor Joaquim Manoel Gonçalves, Dr.

São José - SC

Julho/2022

Fabiano Kraemer
Dispositivo IoT para monitoramento e diagnóstico de sistemas de refrigeração/ Fabiano Kraemer. – São José - SC, Julho/2022-
79 p. : il. (algumas color.) ; 30 cm.

Orientador: Professor Arliones Stevert Hoeller Junior, Dr.
Coorientador: Professor Joaquim Manoel Gonçalves, Dr.
Monografia (Graduação) – Instituto Federal de Santa Catarina – IFSC
Campus São José
Engenharia de Telecomunicações, Julho/2022.

1. Sistema embarcado. 2. Sistema operacional em tempo real. 2. Refrigeração. I. Arliones Stevert Hoeller Junior. II. Instituto Federal de Santa Catarina. III. Campus São José. IV. Dispositivo IoT para monitoramento e diagnóstico de sistemas de refrigeração

FABIANO KRAEMER

DISPOSITIVO IOT PARA MONITORAMENTO E DIAGNÓSTICO DE SISTEMAS DE REFRIGERAÇÃO

Este trabalho foi julgado adequado para obtenção do título de Engenheiro de Telecomunicações pelo Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, e aprovado na sua forma final pela comissão avaliadora abaixo indicada.

São José - SC, 29 de julho de 2022:

Professor Arliones Stevert Hoeller Junior, Dr.
Orientador
Instituto Federal de Santa Catarina

Professor Joaquim Manoel Gonçalves, Dr.
Coorientador
Instituto Federal de Santa Catarina

Professor Roberto de Matos, Dr.
Instituto Federal de Santa Catarina

Professor Tiago Semprebom, Dr.
Instituto Federal de Santa Catarina

RESUMO

Sistemas de refrigeração e climatização são muito utilizados no Brasil e no mundo, com crescimento expressivo na última década. Técnicos em refrigeração e climatização utilizam diversas ferramentas nos processos de instalação e manutenção desses sistemas. Essas ferramentas não são integradas, obrigando o técnico a utilizar quatro ou mais ferramentas para realizar os procedimentos de instalação ou manutenção corretamente. Essas ferramentas também não costumam possuir tecnologia embarcada, sendo em sua maioria analógicas ou com visores digitais simples. O objetivo deste trabalho é desenvolver um sistema embarcado integrando em um único dispositivo três sensores diferentes (temperatura, pressão e potência elétrica), processando os dados colhidos por esses sensores e disponibilizando-os em uma plataforma web e em um aplicativo de *smartphone*, comunicando-se via WiFi. A base do projeto é o sistema embarcado, que utilizará o sistema operacional de tempo real FreeRTOS e o microcontrolador ESP32.

Palavras-chave: Sistema embarcado. Sistema operacional de tempo real. FreeRTOS. Microcontrolador. ESP32. Node-RED. Grafana. MQTT. Bluetooth.

ABSTRACT

Title: IoT Device for Monitoring and Diagnosing of Refrigeration Systems

Refrigeration and air conditioning systems are widely used in Brazil and worldwide, with significant growth in the last decade. Refrigeration and air conditioning technicians use tools in the installation and maintenance processes of these refrigeration systems. These tools are not integrated, forcing the technician to use four or more tools to carry out the installation or maintenance procedures correctly. Also, these tools do not usually have embedded technology, being mostly analog or featuring simple digital displays. The goal of this work is to develop an embedded system to integrate, in a single device, three different sensors (temperature, pressure and electrical power), process the collected data and to make them available on a web platform and an smartphone application, communicating via WiFi. The basis of the project will be the embedded system, which will use the FreeRTOS real-time operating system and the ESP32 microcontroller.

Keywords: Embedded Systems, Real Time Operational Systems, FreeRTOS, Microcontroller, ESP32, Node-RED, Grafana, MQTT, Bluetooth.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

O mercado nos últimos anos vem adotando mais equipamentos considerados "inteligentes", dispositivos eletrônicos não apenas mais poderosos em poder de processamento, com hardware mais poderoso, mas também equipamentos conectados, seja à Internet ou outros meios de comunicação, preferencialmente sem fio. Estamos entrando na era da Internet of Things (IoT), onde esses equipamentos com sua eletrônica embarcada e conectados à Internet permitem funções e facilidades antes só vistas em filmes de ficção científica. Desde geladeiras que monitoram os alimentos e avisam o usuário de comidas estragadas ou quando precisam repor, máquinas de lavar que avisam o dono se ele usou muito sabão, cafeteiras que aquecem sozinhas e avisam o término do preparo enviando uma mensagem, ar condicionado que liga ao detectar a pessoa indo para casa, carros inteligentes e interligados, os usos e possibilidades para esses dispositivos são intermináveis. A "casa inteligente" enfim é uma realidade, com equipamentos como a Alexa[1] se popularizando nas residências (MOREIRA, 2020).

Na indústria a situação não é diferente, onde linhas de produção inteiras com seus maquinários são interligadas e conectadas, permitindo operação e monitoração remotas. Com sistemas embarcados cada vez menores, mais poderosos, mais conectados e mais baratos, a IoT avança em todos os segmentos, permitindo que pessoas e máquinas saibam e monitorem os mais diversos dados de sensores, desde de monitorar a umidade e a radiação solar em uma plantação, a velocidade e comportamento de veículos no trânsito, a qualidade do ar numa via movimentada, não há limites para onde possamos levar a medição de dados no nosso meio.

Um dos segmentos em maior expansão no Brasil e também no mundo é o de refrigeração e climatização. Segundo estimativas da Associação Brasileira de Refrigeração, Ar Condicionado, Ventilação e Aquecimento (ABRAVA), o mercado de Aquecimento, ventilação, ar condicionado e refrigeração (HVAC-R) foi próximo de R$32,9 bilhões (ABRAVA, 2020) em 2020. Um dos primeiros conceitos que vem a nossa mente ao pensar em IoT aplicado à refrigeração é de soluções de monitoramento de ambiente e consumo. Utilizar sensores para saber qual temperatura de um cômodo, um freezer, ou uma cadeia de suprimentos refrigerados em um mercado ou transportados em contêineres, podendo monitorá-los e controlá-los remotamente, são seus usos mais comuns e disseminados. Porém, todos esses equipamentos e estruturas de refrigeração e climatização têm algo em comum que vem sendo negligenciado na onda de digitalização: o processo de instalação.

Com um crescimento médio anual no número de novos aparelhos de refrigeração

[1] https://developer.amazon.com/pt-BR/alexa

Figura 1 – Equipamentos para instalação de um sistema HVAC. 1) Alicate amperímetro e voltímetro. 2) Vacuômetro Digital Testo552. 3) Termômetro penta Vulkan. 4) Ferramentas diversas. 5) Maçarico e gás MAPP. 6) Conjunto manifold. 7) Bomba de vácuo e óleo lubrificante. 8) Alicates.

Fonte: (VITAL, 2021).

residenciais de 9% entre 2005 e 2017 (ENERGéTICA, 2018), e estabilizado na área comercial/industrial entre 2014 e 2020 (ABRAVA, 2021), a tecnologia em IoT vem tendo lenta adoção onde cada vez mais se mostra necessário a utilização de equipamentos modernos, fornecendo melhores instalações e projetos de refrigeração, reduzindo o desperdício de material (cobre, fluído refrigerante, etc) e permitindo o melhor funcionamento do equipamento, reduzindo seu consumo elétrico e aumentando sua vida útil.

O processo de instalação ou manutenção de um sistema de refrigeração exige, além de um técnico devidamente habilitado, uma série de equipamentos específicos (ELETROFRIGOR, 2019), exemplificados na Figura 1.

Uma das etapas mais importantes no processo de instalação/manutenção de sistemas de refrigeração envolve a monitoração de grandezas de temperatura, pressão e consumo elétrico. Essa monitoração é realizada com o uso de três tipos de equipamentos: sensores de temperatura, um manômetro para medir pressão e um alicate multímetro para grandezas elétricas. Tipicamente esses três equipamentos são analógicos ou com quantidade mínima de eletrônica embarcada. Isso faz com que o operador técnico necessite constante observação sobre as grandezas medidas, tomando tempo e atenção. Também aumenta o peso e quantidade de equipamentos carregados.

Na Figura 2 temos um exemplo de um técnico realizando o procedimento, com sua atenção focada na leitura dos sensores. Esses dispositivos, com esse procedimento, fazem

Figura 2 – Refrigerista monitorando equipamentos.

Fonte: (SHUTTERSTOCK, 2020).

parte da maioria das instalações e rotina dos técnicos na área de refrigeração. Embora a pandemia de Covid-19 tenha feito a economia do país desabar em 2020, refletindo-se negativamente no faturamento do HVAC-R, com uma queda de 4% nas vendas, ao menos o mercado de ferramentas para sistemas de refrigeração e ar condicionado seguiu o caminho contrário. Razões não faltam para esse comportamento, como o fato de o trabalho dos técnicos ter sido considerado essencial, especialmente para serviços de instalações e de manutenção e limpeza de equipamentos. Em outra ponta, os *players* dessa indústria apostaram na diversificação de seus produtos, dando tração nas vendas com o apelo gerado pela tecnologia embarcada em novas ferramentas. Os equipamentos integram algumas funções, como medidor de temperatura e pressão em um mesmo dispositivo, exibindo a informação numa tela LCD, integrando diversas informações, e possibilitando alguma configuração simples, como emissores de alerta sonoros. Os mais avançados, permitem até a transferência via bluetooth da informação lida dos sensores para aplicativos rodando em dispositivos móveis, como celulares e tablets. Em comum entre esses dispositivos mais modernos, estão o custo elevado, não integração total dos sensores, serem importados, não terem idioma localizado (PT-BR) e assistência técnica especializada e cara.

1.1 Motivação e histórico do projeto

No 2° semestre letivo 2020 do IFSC-SJE, o autor desta monografia, sob orientação dos professores Carlos Boabaid, Joaquim Manoel Gonçalves e Samuel Luna de Abreu, realizou um projeto de pesquisa intitulado “Análise experimental dos parâmetros de controle e operacionais de uma máquina de fabricar gelo movida a energia solar” (SJ, 2021). Devido à restrição de acesso ao campus imposta pela pandemia de Covid-19,

Figura 3 – Protótipo do projeto de pesquisa IFSC-SJE.

Fonte: Autoria própria.

não houve possibilidade de usar os equipamentos e estruturas do *campus*. Com isso, foi prototipado um pequeno sistema de refrigeração residencial para permitir ao aluno desenvolver os experimentos do projeto. Com o foco em monitorar o comportamento do sistema, foi desenvolvido um protótipo de sistema embarcado com diversos sensores responsáveis por coletar dados do sistema. Sensores, estes, semelhantes aos usados pelos técnicos em refrigeração. Com este projeto de pesquisa, foi pesquisado se existiam produtos semelhantes no mercado, e identificado a possibilidade de desenvolver uma solução que agregue funcionalidades inovadoras e mostre um diferencial frente ao que existe de estado da arte do mercado.

1.2 Objetivos

O objetivo geral deste trabalho é desenvolver uma solução para monitoramento de um sistema de refrigeração, englobando um dispositivo eletrônico de aquisição de dados de sensores com conexão sem fio para armazenar e visualizar os dados na nuvem através de um navegador de Internet e em um aplicativo em smartphone ou tablet.

Para realização do objetivo geral, alguns objetivos específicos foram definidos:

- Projetar e construir um dispositivo de coleta dos dados de sensores de temperatura, pressão e energia elétrica com capacidade de comunicação sem fio;
- Projetar e desenvolver o software do dispositivo de coleta de dados com base em um sistema operacional em tempo real;
- Integrar dispositivo a uma plataforma online e um aplicativo Android para visualização dos dados.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão abordados as bases teóricas e ferramentas necessárias para o desenvolvimento da solução proposta.

2.1 Internet das Coisas

IoT - Internet das Coisas - é um conceito de objetos do cotidiano conectados entre si e à Internet. Embora a ideia de ter um dispositivo além de computadores e notebooks conectado à Internet não seja nova, tendo seu primeiro caso registrado em 1980 (IBM, 2018), foram só nos últimos anos que as soluções baseadas em IoT estão sendo exploradas e aplicadas em larga escala. Conforme descrevem Carrion e Quaresma (2019), "a origem semântica da expressão Internet das Coisas é composta por duas palavras e conceitos: em *Internet*, tem-se o protocolo de comunicação, e em *Coisas*, objetos não identificáveis com precisão. À vista disso, semanticamente, *Internet das Coisas* significa uma rede mundial de objetos interligados, com base em protocolos de comunicação". Tem-se, portanto, um conceito de que qualquer objeto conectado à Internet possa ser considerado como elemento da IoT.

Como representado na Figura 4, sensores estão espalhados nos mais diversos

Figura 4 – Internet das Coisas.

Fonte: (CóRDOBA, 2018).

Tabela 1 – Usos e aplicações típicos IoT.

Cenário	Edifício inteligente		Casa/escritório inteligente		Transporte inteligente		Medicina inteligente	
Caso típico de uso	Medição água	Monitoração predial	Monitoração residencial	Reunião inteligente	Monitoração de tráfego	Assistente de condução	Monitoração de pacientes	Alerta de sinal vital
Tipo dispositivo	Sensores, medidores	Sensores	Aplicações inteligentes, sensores	Notebooks, vídeos	Sensores, câmeras	Sensores, veículos	Sensores, equipamentos médicos	Sensores, equipamentos médicos
População de usuários	Grande	Baixa	Baixa	Média	Grande	Baixa	Baixa	Baixa
Consumo de energia	Baixo	Baixo	Alto	Alto	Alto	Médio	Baixo	Baixo
Custo manutenção	Baixo	Baixo	Baixo	Baixo	Baixo	Médio	Alto	Alto
Taxa de transferência	Baixa	Baixa	Baixa	Alta	Alta	Baixa	Baixo	Baixa
Tolerância a latência	Alta	Baixa	Alta	Média	Alta	Baixa	Média	Baixa
Mobilidade	Fixo	Fixo	Baixa	Baixa	Fixo	Alta	Fixo	Média
Confiabilidade	Média	Alta	Média	Baixa	Baixa	Alta	Alta	Alta
Segurança e Privacidade	Baixa	Alta	Alta	Alta	Baixa	Alta	Alta	Alta

Fonte: (HOU et al., 2016).

lugares, medindo temperatura, qualidade do ar, vibração, batimentos cardíacos, pressão, etc. Relógios, semáforos, carros, eletrodomésticos, qualquer objeto imaginável conectado à Internet pode ser considerado um dispositivo IoT. Essas conexões, preferencialmente sem fio, permitem uma cobertura de conectividade abrangente, além de alta mobilidade. Junto com essa alta conectividade, predomina sobre os dispositivos IoT o conceito de Machine to Machine (M2M), onde a maior parte das comunicações são feitas entre as máquinas, sem interferência ou observação humana. Estima-se que em 2020 o número de dispositivos IoT chegariam a 50 bilhões ao redor do mundo (HOU et al., 2016). As tecnologias de conectividade e hardware envolvidas variam de acordo com a aplicação/objetivo do sistema IoT. Na Tabela 1 temos exemplos de aplicações típicas que usam IoT.

2.2 Sensores

Sensores são dispositivos que respondem de maneira específica perante um estímulo físico ou químico produzindo um sinal, geralmente elétrico, permitindo sua análise/monitoramento. Um sensor associado a um módulo de transformação do sinal para outra grandeza pode ser definido como um transdutor, que converte um tipo de energia em outro, para fins de medição (BALBINOT; BRUSAMARELLO, 2011). Existe uma infinidade de sensores e transdutores para analisar as mais diversas grandezas encontradas no nosso meio. Para atender os requisitos definidos do projeto, serão analisados três tipos de sensores: de temperatura, pressão de fluído/gás e consumo de energia elétrica.

2.2.1 Sensor de temperatura

Sensores de temperatura são dispositivos de medição que verificam a temperatura a partir de uma característica físico-química do dispositivo, como radiação térmica, resistência elétrica e campo eletromagnético. Há cinco tipos comuns de sensores de temperatura que verificam-na através de diversas maneiras, cada um com suas vantagens e desvantagens: termistores, interruptores bimetálicos, Detectores de temperatura de resistência (RTD), termopares e sensor de temperatura infravermelho sem contato (SPEC, 2015).

Termistores são dispositivos semicondutores que verificam a temperatura através de uma resistência elétrica proporcional. São divididos em dois tipos: Negative Temperature Coeficient (NTC), com coeficiente de temperatura negativo, onde a resistência cai de forma não linear com o aumento da temperatura, e Positive Temperature Coeficient (PTC), de coeficiente de temperatura positivo, com a resistência aumentando conforme a temperatura aumenta. Entre as vantagens dos termistores estão seu pequeno tamanho e alto grau de estabilidade. Os de tipo NTC também possuem ótima durabilidade e são muito precisos. As desvantagens incluem uma não linearidade conforme a variação de temperatura e não recomendação para uso em temperaturas extremas.

Interruptores Bimetálicos utilizam uma mola com material bimetálico como elemento principal do sensor de temperatura. Esse material bimetálico é composto por dois tipos de metais, sendo um deles com suscetibilidade diferente a temperatura, alterando suas dimensões devido a dilatação térmica de maneira diferente do outro metal, causando uma curvatura na mola. Os tipos de metais podem incluir latão, alumínio, aço ou cobre, porém é preciso que haja diferença na sensibilidade ao calor dos dois metais. O movimento de curvatura geralmente é usado para modificar um ponteiro sobre uma escala calibrada, indicando a temperatura para o usuário. As vantagens dos interruptores bimetálicos estão no seu baixo custo e tenacidade. Eles também são fáceis de usar e instalar e precisos em uma ampla faixa de temperaturas. As desvantagens são a não possibilidade de mudar a calibração devido ao uso ou ao ambiente, e que são menos precisos que os termômetros de haste de vidro, por exemplo.

RTDs são normalmente constituídos de um enrolamento de fio, que muda sua resistência conforme as variações de temperatura. São feitos de materiais como Platina, Cobre ou Níquel. Ficam envoltos em um encapsulamento de vidro ou cerâmico para evitar oxidação. São altamente estáveis, e estão entre os sensores com a maior linearidade de sinal, tendo o valor da temperatura calculado com polinômios para melhorar a aproximação. Entre as vantagens dos RTDs estão sua saída estável por longos períodos de tempo, facilidade de calibrar e fornecer leituras muito precisas. As desvantagens são sua faixa de operação de temperatura menor, projeto menos robusto e custo elevado.

Termopares são sensores precisos, mostrando alta sensibilidade a mudanças peque-

nas de temperatura, respondendo rapidamente a mudanças no ambiente. São constituídos por um par de fios de metal com propriedades diferentes, unidos em uma extremidade. Esse par metálico cria uma diferença de tensão termoelétrica entre suas extremidades, indicando assim a diferença de temperatura entre essas extremidades. A calibração de um sensor termopar envolve usar temperaturas conhecidas, colocando uma das junções de metal em um local com temperatura conhecida e a outra no objeto que precisa ter a temperatura verificada. Com a tensão obtida, é usado uma fórmula de calibração para obtenção da temperatura. As vantagens dos termopares estão na sua alta precisão em uma ampla faixa de temperatura, operação confiável e são adequados para fazer medições automatizadas, apresentando baixo custo e alta durabilidade. As desvantagens são pelos materiais serem sujeitos à corrosão, causando deterioração da medição devido a um longo período de tempo, e a necessidade de precisar medir duas temperaturas.

Sensores de temperatura por infravermelho sem contato são sensores que trabalham sem contato físico, absorvendo radiação infravermelha emitida por uma superfície. São utilizados principalmente onde a medição direta de temperatura não é possível. O fotodetector transforma as ondas infravermelhas em um sinal elétrico, correspondente a uma determinada temperatura. As vantagens de sensores IR são a isolação da fonte a ser medida, uma ponteira a laser e a facilidade de uso. Entre as desvantagens estão custo elevado e a necessidade de alimentação elétrica para medição.

Para a escolha correta do tipo de sensor de temperatura devem ser consideradas suas especificações. Entre elas estão sua precisão, estabilidade, condições operacionais, durabilidade, expectativa de vida e custo. Para este projeto, o modelo de sensor escolhido foi o **DS18B20**.

2.2.1.1 DS18B20

O DS18B20 é um sensor de temperatura digital, do tipo termistor, sendo comum encontrá-lo com um encapsulamento de metal, coberto parcialmente por um invólucro de borracha, junto com o fio, tornando-o à prova de água. É encontrado no mercado brasileiro na faixa de R$10,00 a R$25,00. A Tabela 2 contém suas principais características, e na Figura 5 temos o chip em seu encapsulamento e no invólucro de metal, já com a fiação.

O DS18B20 é produzido pela Dallas Semiconductor (SEMICONDUCTOR, 2018). Sendo digital, utiliza um protocolo de comunicação chamado 1-Wire, que possui um conceito similar ao protocolo I^2C[1], mas com taxas mais baixas de dados (16.3 kbit/s)[2] e maior alcance. Uma grande vantagem ao se utilizar esse protocolo é a possibilidade de usar múltiplos dispositivos em uma mesma porta/pino do microcontrolador. Para identificação

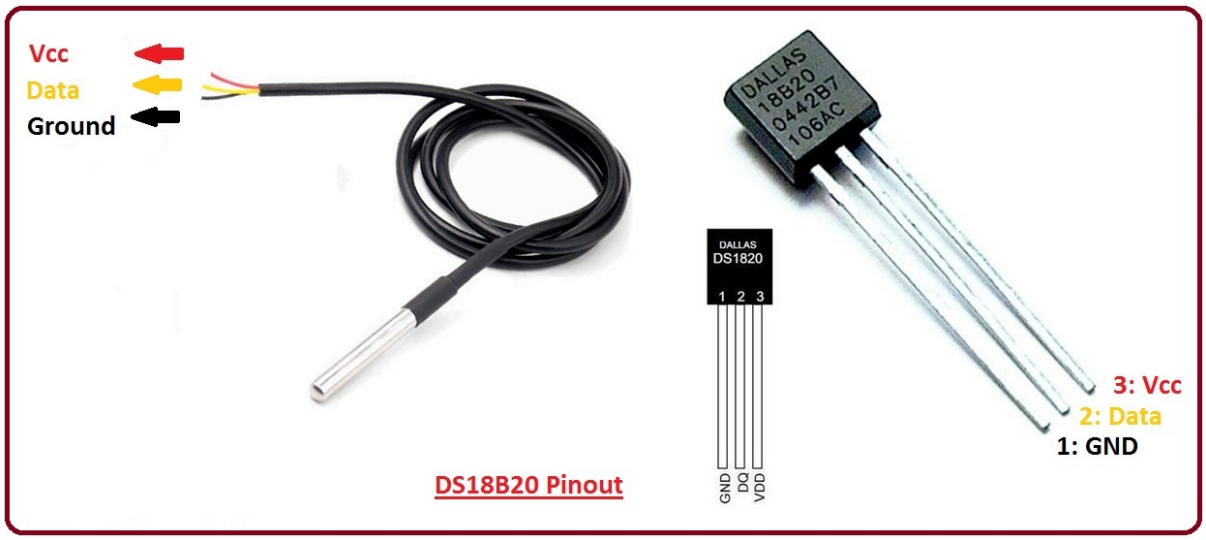
[1] Barramento serial multimestre usado para conectar periféricos de baixa velocidade a uma placa mãe, a um sistema embarcado ou a um telefone celular.
[2] https://www.maximintegrated.com/en/products/ibutton-one-wire/one-wire.html

Tabela 2 – Principais características DS18B20.

Característica	Valor
Alimentação	3,0 V a 5,5 V
Temperatura de operação	-55°C a 125°C
Precisão entre -10°C a 85°C	± 0,5°C
Pinagem	Vermelho: VCC Amarelo: Dados Preto: GND
Comprimento do cabo	1 m
Protocolo	1-Wire

Fonte: (SEMICONDUCTOR, 2018)

Figura 5 – Sensor de temperatura DS18B20.



Fonte: (ALI, 2019).

do sensor em uma malha de sensores usando um único fio, cada sensor é equipado com um código serial de 64 bits, atribuindo endereços diferentes a todos os sensores conectados. Os dados de temperatura recebidos do fio estão na faixa de resolução de 9 bits a 12 bits, podendo ser alterado na inicialização do dispositivo.

2.2.2 Sensor de pressão

Um sensor de pressão é um dispositivo que converte uma pressão aplicada nele em um sinal elétrico. São compostos de duas partes: (I) um material elástico que se deforma sob a aplicação de pressão e (II) uma parte elétrica que detecta esta deformação (SILVEIRA, 2018). Dependendo da aplicação e da pressão envolvida, os materiais de deformação recebem diferentes formas e tamanhos, como diafragmas, foles, pistão, tubo de furo. A conexão do dispositivo elétrico no material elástico pode ser de 3 tipos diferentes: resistivos, capacitivos e indutivos.

Um sensor de pressão **resistivo** possui um calibrador de tensão que é conectado ao material elástico deformável. Com qualquer alteração na deformação, é alterada a resistência elétrica do calibrador, podendo ser medida através de uma ponte de *Wheatstone*, que consiste de um esquema de montagem de elementos elétricos permitindo a medição do valor de uma resistência elétrica desconhecida.

Em um sensor de pressão **capacitivo**, a alteração de pressão é medida na alteração de capacitância entre duas placas. Uma placa do material elástico é colocada na superfície não pressurizada, enquanto a outra é fixada sobre o lado deformável. A deformação causa uma mudança na distância entre as duas placas causando a variação da capacitância. Isso é usado para variar a frequência de um oscilador ou usado como elemento em uma ponte de capacitores. Essa frequência pode ser lida por um dispositivo eletrônico, indicando o valor da pressão.

No sensor de pressão **indutivo**, a deformação do material elástico é usada para fornecer movimento linear de um núcleo ferromagnético, fazendo variar a corrente induzida. Essa mudança da corrente é transformada em variação de tensão, sendo lida posteriormente pelo sistema embarcado.

2.2.2.1 Diferenças entre sensor de pressão, transdutor e transmissor

Um sensor de pressão pode ser qualquer dispositivo que meça a pressão e a converta em um sinal elétrico. É um termo abrangente que inclui transdutores de pressão, transmissores e comutadores. Todos os transdutores, transmissores e comutadores são sensores, porém nem todos os sensores são transdutores, transmissores ou comutadores (SILVEIRA, 2018).

Transdutor de pressão e **transmissor de pressão** são dois dispositivos muito parecidos. O transmissor de pressão transforma os valores de pressão em corrente (na faixa de 4-20mA) através de uma carga de baixa impedância. São os mais utilizados, pois podem enviar o sinal em distâncias relativamente longas (até 2km). Já um transdutor de pressão é um dispositivo eletromecânico, convertendo os valores de pressão em variação de tensão através de uma carga de alta impedância (5kΩ ou mais).

Comutador ou pressostato é semelhante a um interruptor, fechando ou abrindo um contato elétrico quando uma pressão pré definida de fluído é atingida em sua entrada. O pressostato pode ser projetado para ser acionado tanto no aumento quanto na queda de pressão. São amplamente utilizados na indústria e nas residências (como nas máquinas de lavar), supervisionando e controlando automaticamente sistemas que usam fluídos pressurizados ou que precisem controlar o nível de fluído.

Nenhum dos tipos de sensores de pressão podem alterar ou controlar a pressão. São dispositivos exclusivos para medir a pressão e informá-la eletronicamente ou visualmente.

Figura 6 – Sensor de pressão adquirido.

Fonte: banggood.com (2021).

2.2.2.2 Sensor de pressão escolhido

Na maioria das aplicações que necessitam medição de pressão, são utilizados sensores trabalhando em 24V e transmitindo a informação via variação de corrente (4mA a 20mA). Isso dificulta sua utilização em sistemas embarcados, que costumam operar em tensões menores, aumentando o circuito necessário para manipulação do sinal e encarecendo o dispositivo. O custo desses sensores também é considerável, oscilando na faixa de R$350,00 a R$700,00 entre os modelos mais utilizados. Para este projeto foi escolhido um sensor encontrado numa loja virtual chinesa. A escolha se deu devido ao seu custo inferior (R$75,00), tamanho reduzido, alimentação em 5V e sinal da informação também em 5V. O modelo foi adquirido no site banggood.com (2021).

Tabela 3 – Principais características Sensor de Pressão adquirido.

Característica	Valor
Pressão	0 a 500 psi
Saída	0,5V a 4,5V, linear
Precisão	±0,5%
Temperatura de Operação	-40°C a +120°C

Fonte: banggood.com (2021).

Quanto ao transdutor adquirido, as informações sobre o mesmo são parcas, limitando-se a uma tabela no próprio site de venda, sem informar fabricante, modelo, ou um possível *datasheet*. Para garantir que o dispositivo informe leituras de pressão corretas, o produto

será testado e comparado com sensores certificados e calibrados.

2.2.3 Wattímetro

Wattímetros são dispositivos para medição de potência elétrica, fornecida ou dissipada, medida em Watts. Possuem circuitos internos para medição de corrente e tensão. Compostos por uma bobina móvel (bobina de potencial ou bobina de tensão) responsável por medir a tensão, ligada em paralelo com o ponto de medição. Uma segunda bobina, chamada como bobina de corrente, bobina de campo ou bobina fixa, é responsável por medir a corrente elétrica do circuito. Essa bobina é ligada em série com o elemento que terá a medição realizada (INSTRUSUL, 2018).

2.2.3.1 Tipos de wattímetro

Wattímetros eletrodinâmicos funcionam por meio de três bobinas: duas fixadas em série com a carga elétrica, e uma bobina móvel, em paralelo com ela. Por outro lado, há um resistor em série que limita a corrente através da bobina móvel que trabalha através do campo magnético para movimentar uma agulha indicadora da potência medida.

Wattímetros eletrônico é usado em equipamentos de rádio e de micro-ondas, onde mede a potência de sinais com frequências superiores aos 60 hertz da rede elétrica. Eles são ótimos para medições de Corrente Alternada.

Wattímetros digitais conseguem medir a corrente e tensão eletronicamente, com taxas de amostragem elevadas. Da mesma maneira podem calcular/medir as estatísticas como potências de pico, média, baixa e de quilowatts-hora consumidos. Eles também monitoram a linha de energia para achar picos de tensão e interrupções. São bastante utilizados para medir consumo em aparelhos domésticos, tendo seu uso se popularizando junto com o conceito de casas e edifícios inteligentes.

2.2.3.2 Wattímetro Pzem 004t

Para este projeto, o wattímetro escolhido foi o Pzem 004t. Desenvolvido pelo Innovators Guru[3] e fabricado pela empresa Peacefair[4]. É um dispositivo pequeno, com custo relativamente baixo (R$60,00 a R$150,00 no mercado nacional), porém robusto e preciso. Focado em medir sistemas de corrente alternada, possui recursos como um circuito medidor isolado do transmissor via opto-acoplamento, alta capacidade de medição (até 100A, 22kW), interface de comunicação UART[5], isolamento galvânico alto, exibição de parâmetro, comunicação direta com o computador, aquisição e armazenamento de dados

[3] <https://innovatorsguru.com>.
[4] <https://peacefair.en.made-in-china.com>.
[5] Interface de comunicação serial assíncrona

Figura 7 – Wattímetro Pzem004t.

Fonte: innovatorsguru.com.

Tabela 4 – Principais especificações Wattímetro Pzem004t.

Tensão	Fator de Potência	Frequência
Faixa de Medição: 80 a 260V Resolução: 0.1V Acuracidade de medição: 0.5%	Faixa de Medição: 0.00 a 1.00 Resolução: 0.01 Acuracidade de medição: 1%	Faixa de Medição: 45Hz a 65Hz Resolução: 0.1Hz Acuracidade de medição: 0.5%
Corrente	**Potência Ativa**	**Energia Ativa**
Faixa de Medição: 0 a 100A Corrente inicial de medição: 0.02A Resolução: 0.001A Acuracidade de medição: 0.5%	Faixa de Medição: 0 a 23kW Medição inicial de energia: 0.4W Resolução: 0.1W Acuracidade de medição: 0.5%	Faixa de Medição: 0 a 9999.99kWh Resolução: 1Wh Measurement accuracy: 0.5%

Baseado em (GURU, 2020).

com posterior visualização ou cópia para o computador. O desenvolvedor também fornece as bibliotecas de programação para operação do wattímetro.

Seu *datasheet* é encontrado no site do Innovators Guru (GURU, 2020). Suas dimensões físicas são 3,01x7,3cm, e o transformador de corrente externo possui diâmetro interno de 33mm. A Tabela 4 contém suas principais especificações.

2.3 Sistemas embarcados e microcontroladores

Enquanto computadores comuns são sistemas de uso geral, capazes de realizar tarefas diversas e novas conforme a adição de novos softwares, sistemas embarcados são sistemas computacionais de uso específico, tendo seus recursos computacionais como processamento, memória e comunicação projetados para propósitos específicos determinados no seu desenvolvimento. Possuem uma combinação de hardware e software, com interfaces de entrada e saída específicas e dedicadas. Realizam uma função especifica para a qual foi programado, interagindo com o ambiente por meio de sensores e atuadores, gerenciado por um software. Seus recursos computacionais costumam ser limitados, permitindo uma redu-

ção do tamanho, custo final, e aumentando a confiabilidade (BARROS; CAVALCANTE, 2010).

2.3.1 Escolha do microcontrolador do sistema embarcado

Existe uma infinidade de hardwares para sistemas embarcados, voltados para as mais diversas aplicações. Sistemas robustos e complexos em um carro, medidores de energia inteligentes capazes de hospedar internamente uma pequena página HTML, um *smartwatch*, ou até um pequeno sensor de medição de alguma grandeza, com consumo de energia ultra baixo. Para cada uma dessas aplicações existe ou é projetado um hardware específico. Porém, projetar um hardware do zero e fabricá-lo é extremamente custoso (YOUNG, 2002), tornando difícil o desenvolvimento por pequenas e médias empresas. Caso a decisão seja utilizar um hardware existente na indústria, tornando o projeto menos custoso, a escolha do sistema embarcado deverá atender aos requisitos do objetivo de sua utilização.

Em 2005 foi lançado o Arduino[6], um sistema embarcado cujos objetivos eram ser barato, funcional e fácil de programar, sendo acessível a estudantes e projetistas amadores. Utiliza um microcontrolador Atmel de 8 bits, tendo um conceito de hardware livre, permitindo personalizar, montar ou modificar partindo do hardware base, sendo dessa forma acessível a estudantes e projetistas amadores. Possuindo uma série de pinos de propósito geral e específico, permite conectar uma infinidade de dispositivos e sensores. Assim, foi criada uma placa com circuitos de entrada/saída, podendo ser conectada à um computador por interface USB e programada via Ambiente de desenvolvimento integrado (IDE) utilizando a linguagem C++. Seu sucesso foi tremendo, sendo utilizado até hoje por estudantes e amadores, principalmente no conceito de Do it yourself - faça você mesmo (DIY). Com o passar dos anos foram desenvolvidos novas versões, tendo mais de 30 derivações de placas e microcontroladores atualmente, mas sempre partindo da premissa de ser um sistema simples, confiável e de fácil manipulação. Com o passar dos anos, novos sistemas embarcados de propósito geral foram sendo desenvolvidos, agregando novas funcionalidades, hardwares mais poderosos e maior conectividade, principalmente sem fio. O que antes era uma área explorada por pessoas no conceito DIY, começou a despertar o interesse de empresas buscando desenvolver produtos mais rapidamente e com custos menores, junto com o advento da IoT.

Em 2021 estão disponíveis diversos microcontroladores e hardwares de sistemas embarcados no mercado, agregando funcionalidades capazes de atender aos requisitos de

[6] https://www.arduino.cc/

projetos de IoT. Chips baseados em arquitetura Atmel AVR[7], ARM[8], x86[9], RISC-V[10] ou arquiteturas proprietárias estão disponíveis em uma ampla faixa de aplicações e custos. Neste projeto serão comparados alguns desses hardwares, levando em conta requisitos como capacidade do hardware, conectividade, canais de entrada/saída e custo. O objetivo é fundamentar a escolha da plataforma para o desenvolvimento deste trabalho.

Foram comparados sistemas embarcados e microcontroladores de 5 fabricantes: Arduino, STMicroelectronics, ESPressif, Teensy e Microchip.

Arduino: foram escolhidos 2 produtos, sendo o popular Arduino Uno e o Arduino Nano 33 IoT, versão mais barata com comunicação sem fio. É possível adicionar conectividades sem fio na placa do arduino Uno, porém necessitam a adição de módulos específicos para isso, adicionando custo e tamanho. **STMicroelectronics**: fabricam 10 famílias diferentes de microcontroladores, divididos em mais de 1200 variações de chips, sendo a grande maioria com CPUs ARM. No comparativo foram colocados os modelos STM32F401CC e STM32WB5MMGH6TR com comunicação sem fio.

Teensy: Linha de sistemas embarcados desenvolvidos pela PJRC [11]. Possuem poucos modelos, todos baseados em ARM e sem conectividade sem fio. Porém são poderosos e versáteis, com grande quantidade de portas de entrada/saída. Está no comparativo seu último modelo lançado, Teensy 4.1. **Microchip**: Selecionado o microcontrolador WFI32E01PC, mas que não possuí módulos completos no mercado, sendo necessário projetar um para usá-lo. **ESPressif**: No comparativo estão seus dois produtos mais famosos, ESP8266 e ESP32. Assim como os microcontroladores da STMicroelectronics, possuem uma grande variedade de modelos de chips com pequenas mudanças nas especificações. Na Tabela 5 há a comparação de alguns microcontroladores mais comuns nas placas de desenvolvimento mais populares do mercado.

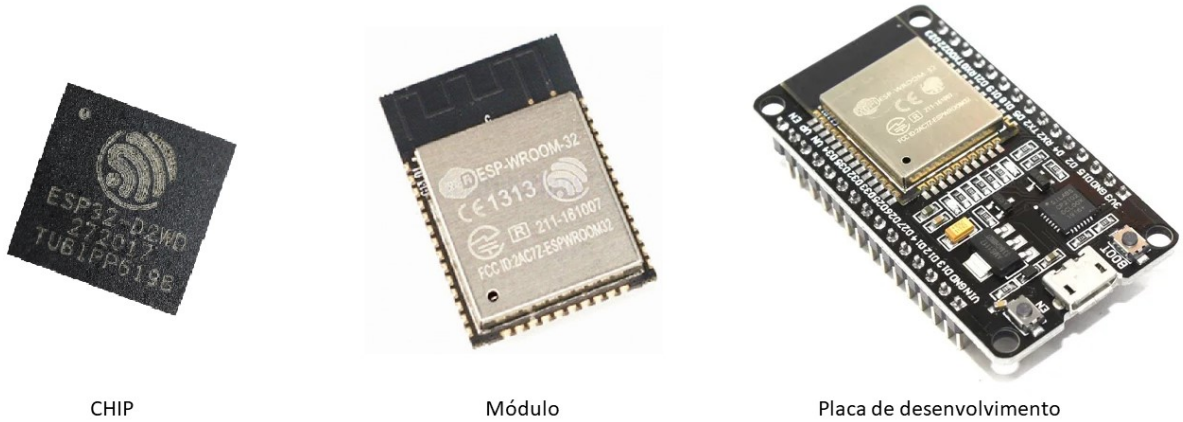
[7] https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/8-bit-mcus/avr-mcus
[8] https://www.arm.com/
[9] https://newsroom.intel.com.br/news-releases/o-intel-x86-completa-40-anos-e-segue-forte/
[10] https://riscv.org/
[11] https://www.pjrc.com/teensy/techspecs.html

Tabela 5 – Tabela comparativa de microcontroladores e módulos de sistemas embarcados.

Nome Comercial	**Arduino Uno**	**Arduino nano 33 IoT**	**Teensy 4.1**	**ESP8266**	**ESP32**	**STM32F401RE**	**STM32WB 5MMGH6TR**	**Microchip WFI32E01PC**
Microcontrolador	Atmega 328	SAMD21 Cortex®-M0+ 32bit low power ARM MCU	MXRT1062DVJ6 Cortex-M7	Tensilica Xtensa LX106	Tensilica Xtensa LX6 dual core	Cortex-M4	ARM Cortex M4 e ARM Cortex M0+ (para rádios e tarefas de segurança)	MIPS32® M-Class Microprocessor Core
Arquitetura	8 Bit	32 Bit	32 e 64 Bit	32 Bit	32 Bit	32 Bit	32 Bit	32 Bit
CPU Freq. (max)	16 MHz	48 MHz	600 MHz	160 MHz	240 Mhz	84 MHz	64 MHz	200 MHz
SRAM	2 KB	256 KB + 520 KB	1024 KB	50 KB	520 KB	64 KB	256 KB	
FLASH	32 KB	1 MB + 2 MB	8 MB	1,3 MB	4 MB	256 KB	1 MB	256 KB
Pinos analógicos	6	8	18	1	16	9	-	-
Pinos digitais	14	14	55	17	32	34	64	37
ADC resolução	10 bits	12 bits	12 bits	10 bits	12 bits	12 bits	-	12 bits
DAC resolução	-	10 bits	-	-	8 bits	-	-	-
Interfaces de comunicação	SPI, I2C, UART	SPI, I2C, UART	SPI, I2C, UART, I2S, CAN	SPI, I2C, UART, I2S	SPI, I2C, UART, I2S, CAN	SPI, I2C, USART, DMA	ADC, SPI, I2C, USART, QSPI	SPI, I2C, UART, I2S
WiFi	não	sim		sim	sim	não	não	sim
Bluetooth	não	sim		não	sim	não	sim	não
Vcc	5V	3.3V		3.3V	3.3V	3.3V	3.3V	3.3V
Outros recursos		ATECC608A crypto chip	Slot cartão SD, Ethernet		Sensor touch, Hall, temperatura		Zigbee	
Preço médio	U$ 3,20	U$ 35,00	U$ 49,00	U$ 1,90	U$ 4,70	U$ 3,00	U$ 12,72	U$ 11,11

Fonte: autoria própria.

Figura 8 – Chip, módulo e placa de desenvolvimento ESP-WROOM-32.



Fonte: aliexpress.com.

2.3.2 ESP32

O ESP32 é um chip com função de microcontrolador feito para trabalhar com sistemas embarcados e Internet das Coisas. O módulo contém WiFi e Bluetooth, podendo funcionar como um sistema *standalone* ou escravo de outro dispositivo.

Fabricado pela Espressif, empresa chinesa localizada em Shanghai, o ESP32 foi lançado em 2016, tendo sido desenvolvidas diversas variantes desde então. Utilizado em uma enorme variedade de módulos e kits de desenvolvimento, conseguiu grande sucesso devido a seu preço aliado a um hardware com mais recursos que soluções concorrentes na mesma faixa de preço, ótima documentação e suporte da fabricante, conectividade sem fio WiFi e Bluetooth e compatibilidade com a IDE Arduino.

Algumas empresas têm criado placas de desenvolvimento com o módulo ESP32 adicionando uma série de recursos interessantes, antes vistos apenas em hardwares muito maiores, mais complexos e mais caros. Em uma mesma placa, sem a adição de extensores, pode-se ter WiFi, Bluetooth, GPS, GSM/GPRS, LoRa, encaixe para bateria com conector para painel solar, tela OLED, slot para cartão micro SD e até saída VGA.

O sistema embarcado escolhido para este projeto foi a placa de desenvolvimento ESP-WROOM-32[12], que utiliza o microcontrolador ESP32. Conforme observado na Tabela 5, suas características, como capacidades do hardware, conectividade, bibliotecas desenvolvidas pela comunidade e preço se mostraram atrativas para o desenvolvimento do

[12] https://www.espressif.com/en/support/documents/technical-documents

projeto. O modelo adquirido possui como principais características:

- CPU: microprocessador Xtensa dual-core (ou single-core) de 32 bits LX6, operando em 160 ou 240 MHz e co-processador ULP de baixo consumo com memória (RTC) de 8KB;
- 520 KB SRAM (em tempo de execução, a DRAM de heap disponível pode ser menor do que a calculada em tempo de compilação, porque na inicialização alguma memória é alocada do heap antes que o agendador do FreeRTOS seja iniciado (incluindo memória para as pilhas de tarefas iniciais do FreeRTOS);
- 4 MB Flash (1,3MB disponíveis para o código do software);
- Wi-Fi: 802.11 b/g/n com recursos de segurança, incluindo WFA, WPA/WPA2 e WAPI;
- Bluetooth: v4.2 BR/EDR e BLE;
- 32 pinos digitais;
- 16 pinos analógicos, com resolução de 12 bit;
- Funciona em ambientes com faixa de temperatura de -40°C até 125°C.
- Regulador de tensão interno de baixa perda;
- Domínio individual de energia para o RTC;
- 5μA de consumo no sono profundo, com possibilidade de acordar através de interrupções no GPIO, timer, ADC ou pelo sensor de interrupção de toque capacitivo.

O gerenciamento de energia do ESP32 possui 5 modos de operação distintos, com detalhes sobre seu consumo na Tabela 6:

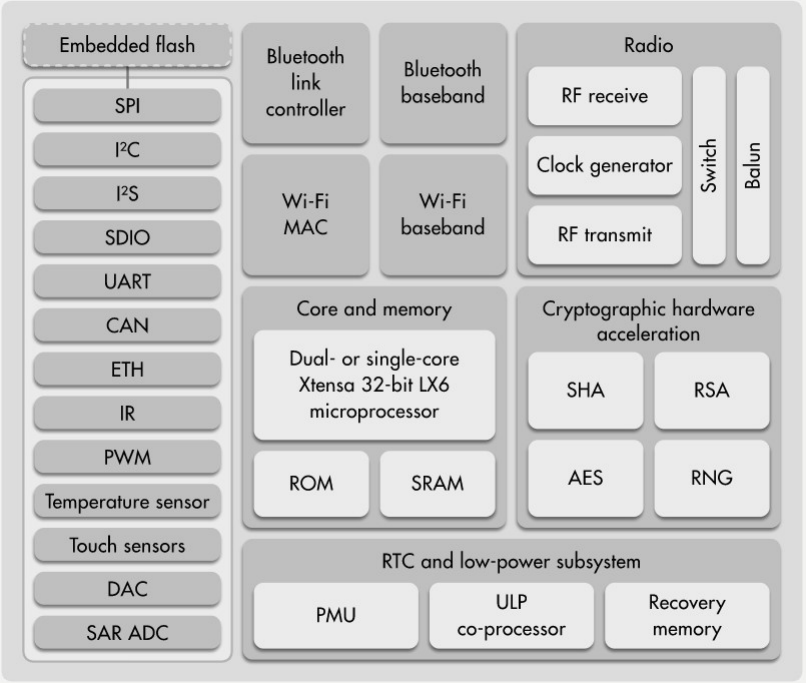
- Active mode: o chip de rádio do ESP está ligado. Ele pode receber, transmitir ou escutar;
- Modem sleep mode: a CPU está operacional e o clock é configurável. O WiFi e Bluetooth ficam desativados;
- Light sleep mode: a CPU está pausada. A memória, Real Time Clock (RTC) e seus periféricos, assim como o coprocessador de baixo consumo Ultra Low Power (ULP) continuam rodando. Qualquer evento nos periféricos (MAC, host, relógio RTC, ou interrupção externa) vão acordar o chip;

Tabela 6 – Modos de operação e consumo ESP-WROOM-32.

Modo de operação	Descrição		Consumo de energia
Ativo (RF trabalhando)	WiFi Tx packet		240 mA
	Bluetooth Tx packet		130 mA
	WiFi/BT Rx e escutando		100 mA
Modem-sleep	CPU permanece ativa	240 MHz	30 mA ∼68 mA
		160 MHz	27 mA ∼44 mA
		80 MHz	20 mA ∼31 mA
Light-sleep	-		0.8 mA
Deep-sleep	O co-processador ULP está ligado		150 µA
	Padrão monitorado por sensor ULP		100 µA
	Timer RTC + memória RTC		10 µA
Hibernation	Somente relógio RTC		5 µA
Power off	CHIP-PU é setado para level baixo, desligando		1 µA

Fonte: (ESPRESSIF, 2018)

Figura 9 – Diagrama de blocos de função ESP32.



Fonte: esp32.net.

- Deep sleep mode: somente a memória RTC e RTC são alimentados. Os dados das conexões WiFi e Bluetooth são armazenados na memória RTC. O co-processador continua ativo;

- Hibernation mode: o oscilador interno de 8 MHz e o co-processador ULP são desativados. A memória de recuperação RTC é desligada. Somente o relógio RTC no clock baixo e certos GPIOs do RTC continuam ativos. Os dois podem acordar o chip do modo de hibernação.

Para programá-lo usa-se linguagens C, C++ ou MicroPython. Para C/C++ a IDE Arduino é compatível com a ESP, bastando a instalação de um pacote através das configurações da IDE. A Espressif desenvolveu uma IDE própria, chamada ESP IDF[13], mas é pouco utilizada. O Visual Studio Code também é compatível, sendo necessário a instalação do PlatformIO[14]. Com a boa documentação fornecida pela Espressif, aliada ao custo baixo e grande quantidade de vendas, a ESP32 possui uma comunidade ativa desenvolvendo bibliotecas para as mais diversas funções e dispositivos/sensores compatíveis. Seu diagrama de blocos de funções pode ser visto na Figura 9.

2.4 Sistemas operacionais embarcados

Utilizar um sistema operacional (SO) pode facilitar e simplificar o desenvolvimento de softwares, livrando os desenvolvedores de se preocuparem com o gerenciamento do hardware de baixo nível em questões como escalonamento, controle de energia, gerenciamento de memória e processos. Os computadores pessoais são complexos e variados em tecnologia de hardware, tornando a utilização de um SO indispensável. Porém, sistemas embarcados possuem hardwares limitados, geralmente com muitas restrições de consumo de energia e custo, tornando o uso de um SO uma decisão de projeto.

Conforme (DENARDIN; BARRIQUELLO, 2019), para escolher utilizar um sistema operacional de tempo real em um sistema embarcado é necessário definir o que é um software embarcado de qualidade, adotando boas práticas no seu projeto e desenvolvimento. Assim, pode-se dizer que sistemas embarcados devem:

- Realizar as tarefas a que se destinam corretamente;
- Realizar essas tarefas no tempo esperado;
- Ter seu código desenvolvido de forma a facilitar a manutenção;
- Permitir a análise estática de sintaxe e fluxo de dados do código, bem como a análise de comportamento do código em tempo de execução;
- Ter comportamento previsível, tempo de execução e requisitos de memória previsíveis.

A lista de requisitos pode ser ampliada, dependendo das características da aplicação. Porém, à medida que um sistema embarcado cresce em complexidade, a implementação de um sistema tradicional torna-se inviável, ou pelo menos de difícil concepção e manutenção. A situação torna-se mais problemática ainda se o sistema necessitar de conexões com interfaces de rede, pilha TCP/IP, USB, RS-232, *displays*, memórias, entre outros. Unir essas

[13] https://docs.espressif.com/projects/esp-idf/en/latest/esp32/
[14] https://platformio.org/

diferentes operações em um sistema embarcado traz o problema do paralelismo de tarefas assíncronas, ou seja, a existência de concorrência entre tarefas que são independentes e que não possuem períodos de ativação múltiplos. Além disso, como muitos dos dispositivos desenvolvidos possuem projetos comuns de sistemas embarcados, é importante que o código desenvolvido possa ser reutilizável ao máximo possível para evitar, a cada novo projeto, a replicação do esforço já dispendido pelos desenvolvedores. A análise desse problema fica mais clara quando se percebe que (DENARDIN; BARRIQUELLO, 2019):

- Existem operações que devem ser executadas em intervalos regulares;
- Existem operações que são ativadas aleatoriamente (tarefas aperiódicas);
- Existem operações que devem ser executadas simultaneamente;
- Existem operações de longa duração na execução;
- Os requisitos de tempo de cada operação são muito diferentes;
- É necessário gerenciar os sistemas multitarefas, que normalmente implica ativar e desativar tarefas, configurar prioridade de tarefas, agendar tarefas, *etc*;
- É necessário realizar comunicação e sincronização entre tarefas;
- É preciso gerenciar memória, tempo, CPU, e outros recursos do sistema;
- Deve-se implementar mecanismos de comunicação assíncrona com interfaces de rede.

Em situações em que muitas destas características estão presentes, se torna mais adequado utilizar um sistema operacional de tempo real no projeto de sistema embarcado.

Sistemas operacionais de tempo real (RTOS) fornecem suporte básico para agendamento, gerenciamento de recursos, comunicação, sincronização, tempo preciso e entrada/-saída (IO). Entre as principais características que se espera de um RTOS em um sistema embarcado, estão (STANKOVIC; RAJKUMAR, 2004):

- Possuir uma troca rápida de contexto;
- Ter um tamanho pequeno (com sua funcionalidade mínima associada);
- Responder a interrupções externas rapidamente (às vezes com uma latência máxima garantida para postar um evento, mas, geralmente, nenhuma garantia é dada quanto a quando o processamento do evento será concluído. Esta garantia posterior às vezes pode ser calculada se as prioridades forem atribuídas corretamente);
- Minimizar os intervalos durante os quais as interrupções são desativadas;

- Fornecer partições de tamanho fixo ou variável para gerenciamento de memória (ou seja, sem memória virtual), bem como a capacidade de bloquear código e dados na memória;

- Fornecer arquivos sequenciais especiais (geralmente baseados em memória) que podem acumular dados em uma taxa rápida.

Para lidar com os requisitos de tempo, o *kernel*, componente central do sistema operacional, deve idealmente:

- Suportar multitarefa;

- Fornecer um mecanismo de agendamento preemptivo baseado em prioridades;

- Fornecer tempo de execução limitado para a maioria das funções do SO;

- Manter um relógio em tempo real de alta resolução;

- Fornecer alarmes e temporizadores especiais;

- Suportar políticas de escalonamento em tempo real baseados em prioridades (ex.: Rate Monotonic, Earliest Deadline First (EDF));

- Fornecer primitivas para atrasar o processamento por um período fixo de tempo e para suspender/retomar a execução.

Um exemplo bastante relevante de sistema operacional de tempo real, que será empregado neste projeto, é o FreeRTOS[15], que é um sistema operacional de tempo real para microcontroladores. De código aberto, ele facilita o desenvolvimento, programação, implantação, a segurança e o gerenciamento de sistemas embarcados. Inclui um *kernel* e um conjunto crescente de bibliotecas com diversas funcionalidades. Desenvolvido em 2003 por Richard Barry e adquirido pela Amazon em 2017, vem tendo aprimoramentos e adições de novas funcionalidades e bibliotecas, como uso com os serviços em nuvem da Amazon Web Service (AWS) e atualização Over-the-air (OTA), *i.e.*, atualização de software remota, via conexão sem fio. É suportado por 35 plataformas de microcontroladores, incluindo a ESP32. Também disponibiliza bibliotecas para uso de MQTT, JSON e WiFi sobre IPv4. Escrito na linguagem de programação C, ocupando cerca de 9KB de ROM e 4KB de RAM, o FreeRTOS está sob uma licença de distribuição MIT[16].

[15] www.freertos.org
[16] https://opensource.org/licenses/MIT

Figura 10 – Exemplo código JSON.

```
[
      {
        "description": "quarter",
        "mode": "REQUIRED",
        "name": "qtr",
        "type": "STRING"
      },
      {
        "description": "sales representative",
        "mode": "NULLABLE",
        "name": "rep",
        "type": "STRING"
      },
      {
        "description": "total sales",
        "mode": "NULLABLE",
        "name": "sales",
        "type": "INTEGER"
      }
    ]
```

Fonte: cloud.google.com

2.5 Protocolos e softwares usados

2.5.1 JSON

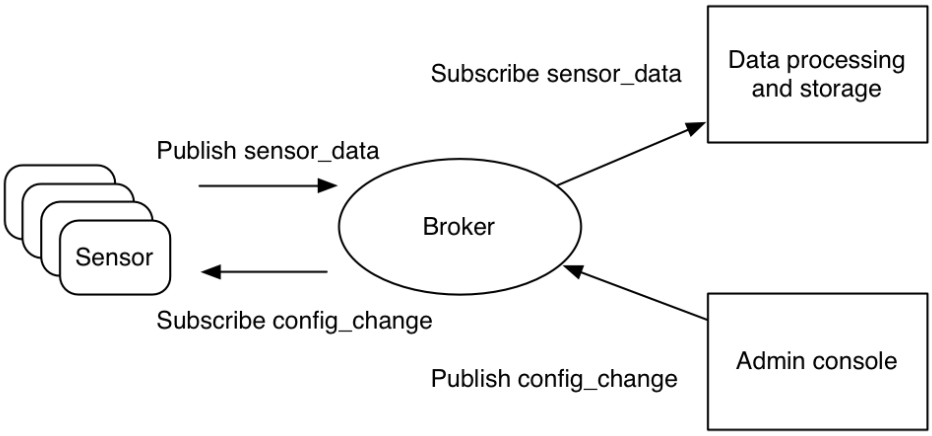
JavaScript Object Notation (JSON)[17] é um modelo de transmissão de informações no formato texto, permitindo troca de dados entre sistemas, independente de linguagem de programação. De padrão aberto, compacto, utilizando texto legível por humanos e fácil para máquinas analisar e gerar. Derivado do JavaScript mas usando um formato de texto completamente independente da linguagem, possui convenções que são familiares aos programadores das linguagens C, Java, Python e outras. É baseado em duas estruturas: uma coleção de pares de nome/valor e uma lista ordenada de valores. Um objeto JSON é um conjunto não ordenado de pares nome/valor. O objeto começa com uma chave esquerda **{** e termina com uma chave direita **}**. Cada nome é seguido por dois pontos **:** e os pares nome/valor são separados por vírgula. O FreeRTOS oferece uma biblioteca nativa para usar JSON. Na Figura 10 está um exemplo de um código com estrutura JSON.

2.5.2 MQTT

Message Queuing Telemetry Transport (MQTT) é um protocolo de comunicação focado em IoT. Desenvolvido em 1998 pela IBM[18], foi projetado com o esquema de troca de mensagens Publicador-Subscritor. Com suporte para comunicação assíncrona, é simples, leve, ideal para dispositivos restritos e redes com baixa largura de banda, alta latência ou não confiáveis, tentando manter algum grau de garantia de entrega. Isso o torna excelente para ser usados em dispositivos de IoT e M2M. Em 2014 tornou-se um formato padrão e

[17] www.json.org
[18] www.ibm.com

Figura 11 – Diagrama fluxo MQTT.



Fonte: (YUAN, 2017)

aberto, certificado pelo OASIS[19].

O MQTT trabalha com dois tipos de entidades na rede: clientes e um *message broker*. O *broker* é um servidor que concentra as informações dos clientes, como endereço IP. Ele recebe as mensagens dos clientes e roteia para os clientes de destino. Um cliente é qualquer coisa que possa interagir com o *broker*, recebendo e transmitindo mensagens. O cliente publica as mensagens em um tópico, enviando-os para o *broker*. Em seguida, o *broker* encaminha a mensagem para todos os clientes que assinaram o tópico (YUAN, 2017). Na Figura 11 está um exemplo de diagrama de fluxo do MQTT.

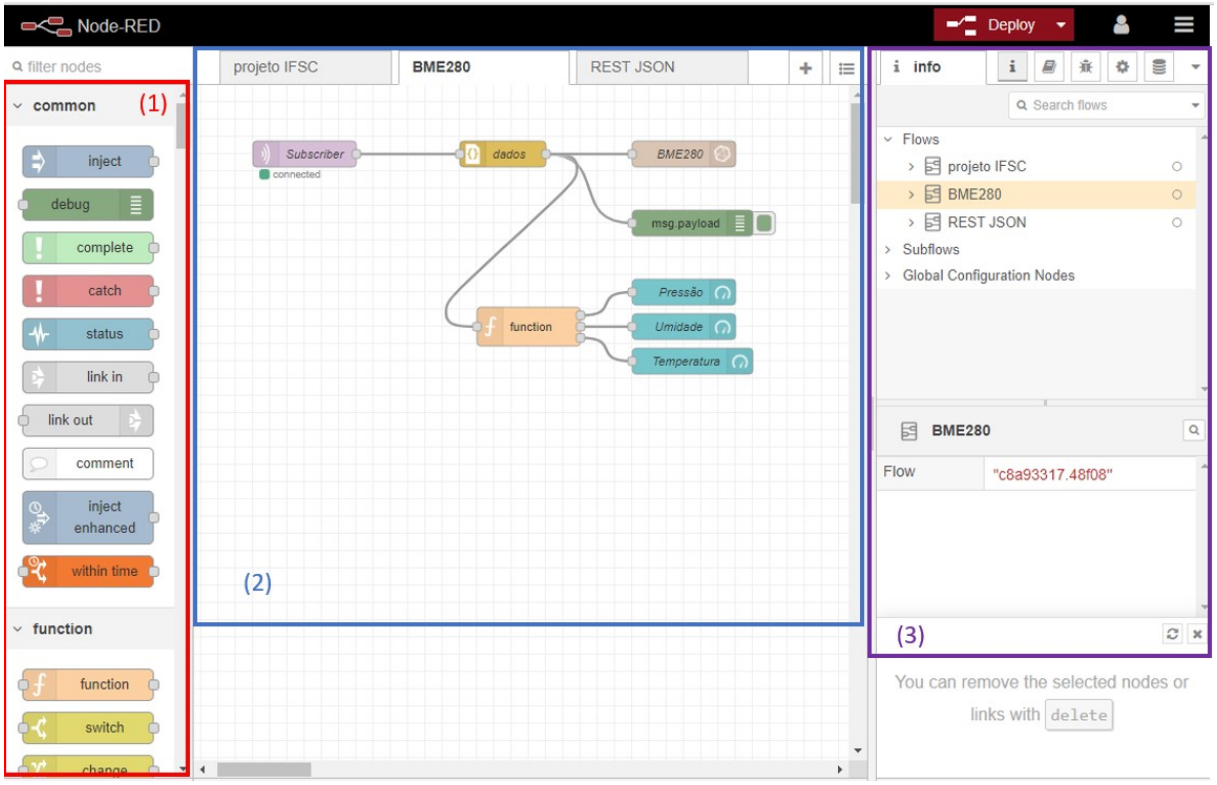
2.5.3 Node-RED

Node-RED é uma ferramenta de programação visual, baseada em fluxos (*flows*), com a programação sendo feita através de uma interface gráfica *web*, com os fluxos podendo ser implantados em tempo de execução com um único clique. Cada nó do fluxo realiza uma função específica, sendo que alguns nós podem ser programados com *JavaScript*. Novos nós com novas funções (ex: broker MQTT, banco de dados) podem ser adicionados na biblioteca do Node-RED via instalação de *plugins* dentro da própria interface do Node-RED. A ferramenta também fornece um painel de controle (*dashboard*), exibindo gráficos e botões para interatividade com o usuário, sendo também programado com fluxos e nós. É uma ferramenta gratuita, estando sob uma licença Apache[20].

A interface gráfica de programação é dividida em três partes principais: área de trabalho, paleta de nós e uma barra lateral. Conforme indicado na Figura 12, a barra lateral (1) mostra a paleta de nós que podem ser utilizados, sendo divididos em três tipos

[19] www.oasis-open.org - Consórcio global que conduz o desenvolvimento, convergência e adoção de padrões para e-business e web services.
[20] http://www.apache.org/licenses/

Figura 12 – Interface do Node-RED.



Fonte: autoria própria.

principais: nós de entrada de dados, nós de processo e nós de saída. Em (2) está a área de trabalho, para onde os nós são arrastados, a programação é feita e os fluxos entre os nós conectados. Já em (3) são mostrados 6 abas: uma aba de informações sobre a seleção atual do usuário, uma de ajuda, debug, configuração do nó, contexto dos dados e por último uma aba do painel de controle.

O Node-RED possui nodos específicos para Broker MQTT e InfluxDB, ferramentas que serão utilizadas neste projeto.

2.5.4 InfluxDB

InfluxDB é um banco de dados de séries temporais - Time Series Database (TSDB), de código aberto, gratuito para usar, sob licença MIT[21]. Desenvolvido pela empresa InfluxData[22] e escrito na linguagem de programação Go, é focado em armazenar e recuperar dados de séries temporais de dispositivos de IoT, dados de sensores de monitoração e análises em tempo real.

Séries temporais são sequências ordenadas de valores de variáveis (por exemplo, temperatura) em intervalos de tempo definidos pelo momento de coleta do valor (VINICIUS, 2018). Os valores que constituem uma série temporal são ordenados em uma linha do

[21] https://opensource.org/licenses/MIT
[22] https://www.influxdata.com

tempo, sendo que essa ordenação é fundamental para revelar informações sobre os padrões envolvidos, podendo alterar o significado dos dados caso não esteja ordenada.

Bancos de dados de série temporal (TSDB) são bancos de dados otimizados para séries temporais ou dados com registro de data e hora, sendo construídos e otimizados para lidar com estes dados. Um TSDB permite que seus usuários criem, enumerem, atualizem, destruam e organizem várias séries temporais de maneira mais eficiente. Entre as suas vantagens sobre banco de dados regulares estão no tamanho reduzido dos dados, escalabilidade, desempenho e custos mais baixos(NAQVI; YFANTIDOU; ZIMÁNYI, 2017).

Com uma sintaxe semelhante ao SQL, o InfluxDB não tem dependências externas. Cada linha de dado armazenado consiste em pares de valores-chave chamados *fieldset* e *timestamp*. Quando agrupados por um conjunto de pares de valores-chave chamados de *tags* fica definida uma série. Os valores podem ser inteiros de 64 bits, pontos flutuantes, strings e booleanos. As políticas de retenção dos dados são definidas pelo administrador do banco de dados[23].

2.5.5 Grafana

Grafana é uma plataforma web para visualizar e analisar dados através de gráficos, tabelas e alertas disponibilizados como um *dashboard*. Personalizável, com o usuário podendo adicionar, modificar e instalar novos tipos de tabelas e gráficos através de *plugins*, sendo compatível com uma grande quantidade de bancos de dados, como o InfluxDB. Bastante utilizado por sistemas de monitoramento, gerando gráficos em tempo real. Possui uma comunidade grande e participativa, desenvolvendo plugins para adição de novos tipos de gráficos, tabelas e funções. É gratuito, estando sob uma licença GNU Affero General Public License[24]. Na Figura 13 está uma imagem com um exemplo de *dashboard*. A ferramenta também permite fazer *download* dos dados de cada gráfico em formato CSV, com os dados desse arquivo conforme o período selecionado no gráfico. Embora possa parecer ter semelhança com o Node-RED na questão de poder exibir gráficos em um *dashboard*, o Grafana é especializado nesta tarefa, tendo mais funcionalidades e mais personalização neste quesito.

[23] https://docs.influxdata.com/influxdb/v1.8/
[24] https://www.gnu.org/licenses/agpl-3.0.pt-br.html

Figura 13 – Exemplo de painel do Grafana.

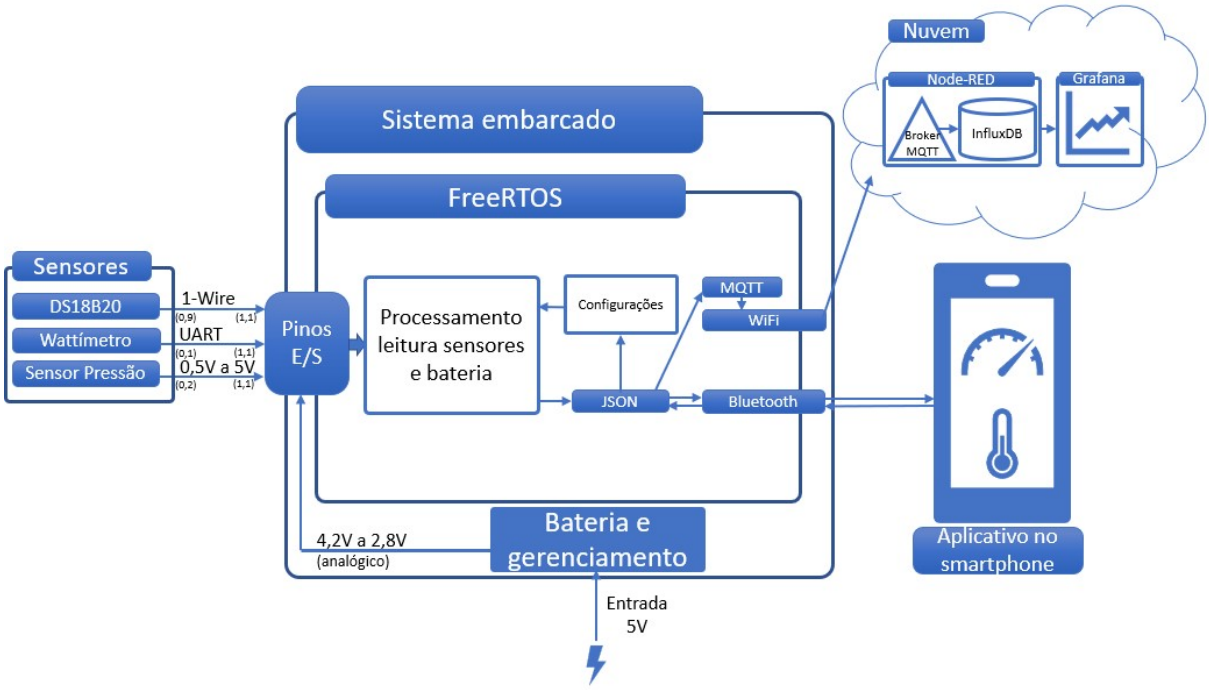
Fonte: autoria própria.

3 PROPOSTA DE PROJETO

3.1 Descrição geral do sistema

A proposta deste projeto é o desenvolvimento de um sistema embarcado capaz de coletar dados de sensores de temperatura, pressão e potência elétrica, com foco de uso em sistemas de refrigeração. Os dados coletados serão enviados para um aplicativo em um *smartphone* via conexão sem fio *Bluetooth* e via WiFi para um servidor na internet. Esse servidor rodará as ferramentas Node-Red, Broker MQTT, InfluxDB e Grafana, armazenando os dados coletados e exibindo-os em gráficos numa página *web* com acesso remoto. O aplicativo no *smartphone* rodará em dispositivos Android e mostrará os dados lidos, sem armazená-los, usando para isso o software *Blynk IoT*[1]. Esse aplicativo também permitirá a configuração do tempo de leitura e envio dos dados do sistema embarcado para ele e a escolha do tipo de gás refrigerante usado no sistema. Na Figura 14 é mostrado o diagrama de blocos do sistema, com uma visão geral do projeto.

Figura 14 – Diagrama proposta projeto.



Fonte: autoria própria.

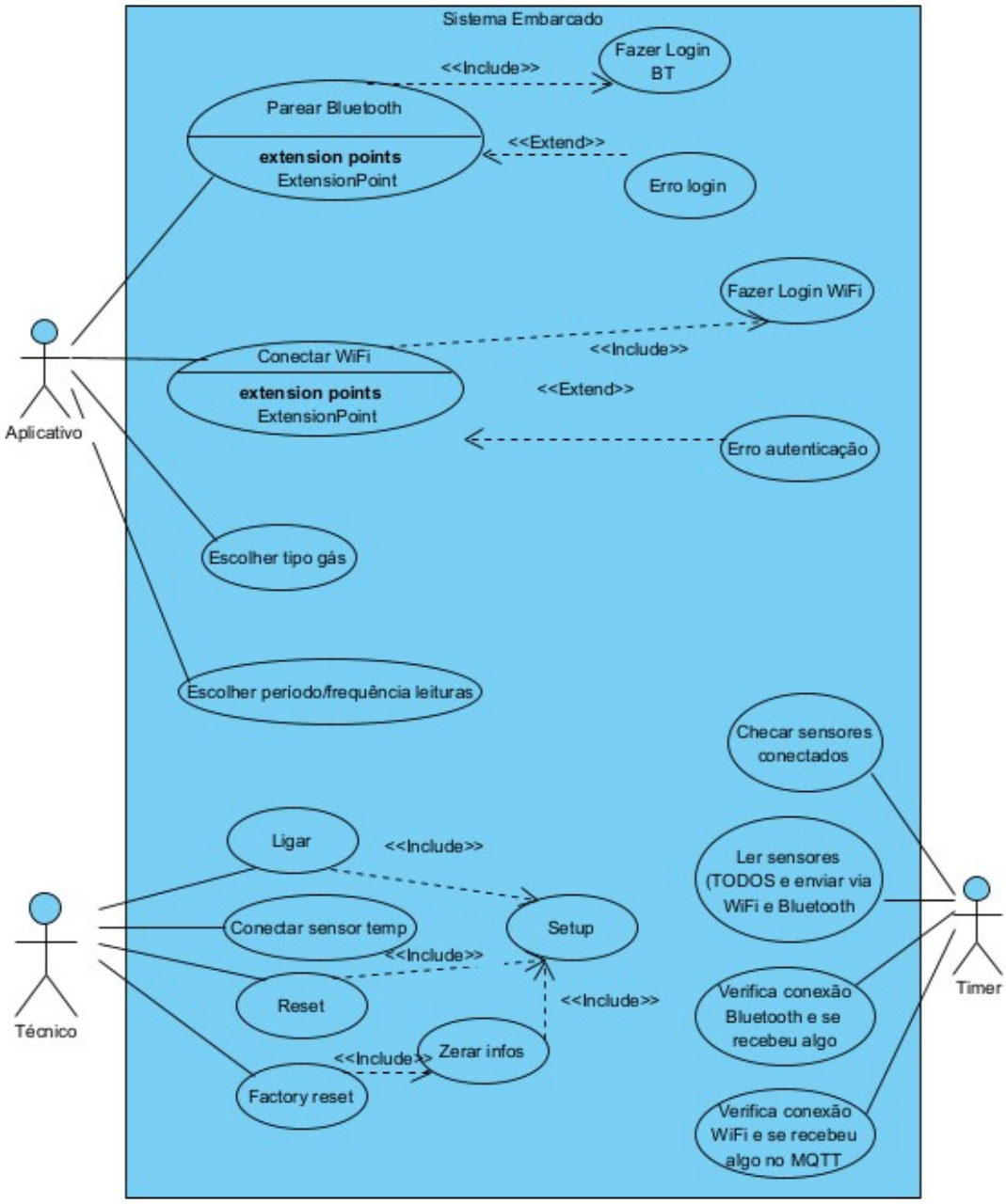
[1] https://play.google.com/store/apps/details?id=cloud.blynk

3.2 Casos de uso sistema embarcado

Para o desenvolvimento do projeto, foram considerados três atores envolvidos nos casos de uso, com seu diagrama mostrado na Figura 15:

- **Aplicativo:** Ator responsável por realizar 4 tarefas em conjunto com o software no celular, com os seguintes casos de uso:
  - Login das conexões *wireless*: Através da interface gráfica no dispositivo móvel, o usuário poderá selecionar a rede que deseja se conectar, escolhe-lá, e efetuar o login, tanto para o WiFi, como para o Bluetooth;
  - Escolha do tipo do fluído refrigerante que utilizará;
  - Tempo de leitura dos sensores, escolhendo o período/frequência com que se deseja fazer as leituras e envio dos dados dos sensores para o aplicativo.
- **Técnico:** Ator das ações do operador humano e suas interações com o dispositivo. Possui os seguintes casos de uso:
  - Ligar ou desligar o dispositivo, através de um botão;
  - Conectar os sensores que deseja no dispositivo;
  - Reiniciar o dispositivo e realizar um *factory reset*, fazendo o sistema embarcado retornar com as configurações de fábrica.
- **Timer:** Ator do relógio interno do sistema embarcado responsável pelas trocas de contexto/tasks do FreeRTOS. Seus casos de uso previstos são:
  - Checar se houveram sensores conectados ou retirados do sistema embarcado;
  - Ler com periodicidade todos os sensores conectados, e enviá-los via conexão *wireless*;
  - Verificar com periodicidade se o usuário enviou algum comando para o sistema embarcado via Bluetooth, realizando as ações determinadas pelo comando;
  - Verificar com periodicidade se o usuário enviou algum comando para o sistema embarcado via WiFi, realizando as ações determinadas pelo comando.

Figura 15 – Casos de uso sistema embarcado.



Fonte: autoria própria.

3.3 Requisitos funcionais

Os requisitos funcionais, apresentados na Tabela 7, descrevem as funcionalidades do sistema.

Tabela 7 – Requisitos funcionais.

Código	**Descrição**
RF01	Ajustar configuração do sistema de monitoramento através de comandos enviados pelo *smartphone* ou nuvem;
RF02	Exibir gráficos em série temporal dos dados coletados dos sensores;
RF03	Exibir os dados das medições dos sensores;
RF04	Informar ao usuário quais sensores estão conectados no sistema embarcado.
RF05	Seis conexões do tipo P2 para sensores de temperatura;
RF06	Permitir a modificação do *layout* dos gráficos;
RF07	Permitir atualizações do software por interface remotas OTA (sem fio);
RF08	Permitir o *download* dos dados visualizados nos gráficos;
RF09	Sistema de emissão de alerta caso algum valor dos sensores atinja determinado limiar.
RF10	Transmitir a informação de quais sensores estão conectados para o aplicativo no *smartphone* via *Bluetooth*;
RF11	Uma entrada e uma saída de tomada de energia padrão N para alimentação do sistema embarcado e alimentação/medição de consumo do sistema de refrigeração;
RF12	Duas conexões de rosca 1/4” para conectar tubulação de fluído para medição da pressão;

Fonte: autoria própria.

3.4 Requisitos não funcionais

Requisitos não funcionais, também chamado de atributos de qualidade, definem como o sistema fará, não estando relacionados diretamente às funcionalidades de um sistema. Tratados normalmente como premissas e restrições técnicas de um projeto, são praticamente todas as necessidades que não podem ser atendidas através de funcionalidades. Geralmente mensurável, os requisitos não funcionais definem características e impõe limites do sistema como método de desenvolvimento, tempo, espaço, Sistema Operacional, dentre outros e cuja medida pode ser determinada. Na Tabela 8 estão os requisitos não funcionais.

Tabela 8 – Requisitos não funcionais.

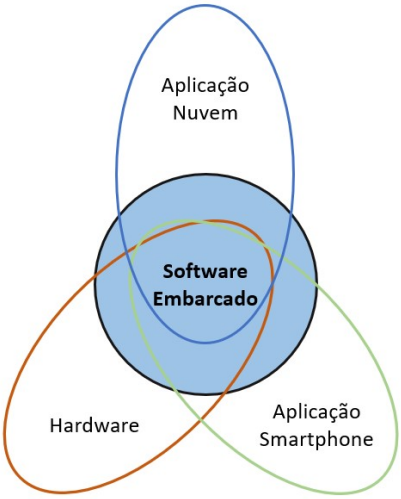
Código	**Descrição**
RNF01	Permitir execuções de tarefas em tempo real;
RNF02	Desativar funções não utilizadas para economia de bateria;
RNF03	Gerenciamento automático sobre conexões *WiFi*, conectando-se automaticamente em redes previamente salvas, ou abrindo ponto de acesso para usuário fazer pareamento com o dispositivo, via aplicativo no *smartphone*;
RNF04	Fornecer gerenciamento sobre conexões *Bluetooth*, conectando-se automaticamente em redes previamente salvas, ou pareando com um *smartphone*;
RNF05	Hospedar o servidor *broker* MQTT;
RNF06	Hospedar e gerenciar o InfluxDB, o Sistema de Gerenciamento de Banco de Dados (SGBD);
RNF07	Hospedar o *dashboard* Grafana;
RNF08	Estabelecer conexão e manter-se conectado no *broker* MQTT se o usuário quiser transmitir os dados para a nuvem, sendo a configuração desta opção recebida via *Bluetooth* pelo aplicativo;
RNF09	O software deverá ser capaz de filtrar possíveis oscilações e picos na leitura dos sensores, não enviando essas leituras inconsistentes;
RNF10	Peso inferior a 850g;
RNF11	Proteção contra surtos na rede elétrica e sensores;
RNF12	Salvar dados de configurações na memória *Flash*, não perdendo essa informação caso o dispositivo seja desligado ou acabe a bateria;
RNF13	Tamanho inferior a 200mm x 150mm;
RNF14	Tempo mínimo de leitura dos sensores e envio dos dados de 2 segundos.
RNF15	Verificar carga da bateria a cada intervalo de tempo específico, a ser definido durante o desenvolvimento;
RNF16	Verificar sensores conectados, tanto na inicialização quanto durante o funcionamento do dispositivo, para saber se os sensores foram conectados ou desconectados;
RNF17	Armazenar os dados lidos dos sensores e da carga da bateria em variáveis com estrutura *JSON* para envio pelas redes *Bluetooth* e *WiFi*;
RNF18	Bateria com duração mínima de 8 horas;

Fonte: autoria própria.

4 DESENVOLVIMENTO

Neste capítulo serão descritas as etapas realizadas no desenvolvimento do hardware e software, juntamente com premissas e boas práticas de desenvolvimento de softwares embarcados utilizados no projeto. O desenvolvimento foi separado em quatro grandes blocos, conforme Figura 16.

Figura 16 – Diagrama tópicos principais do projeto.



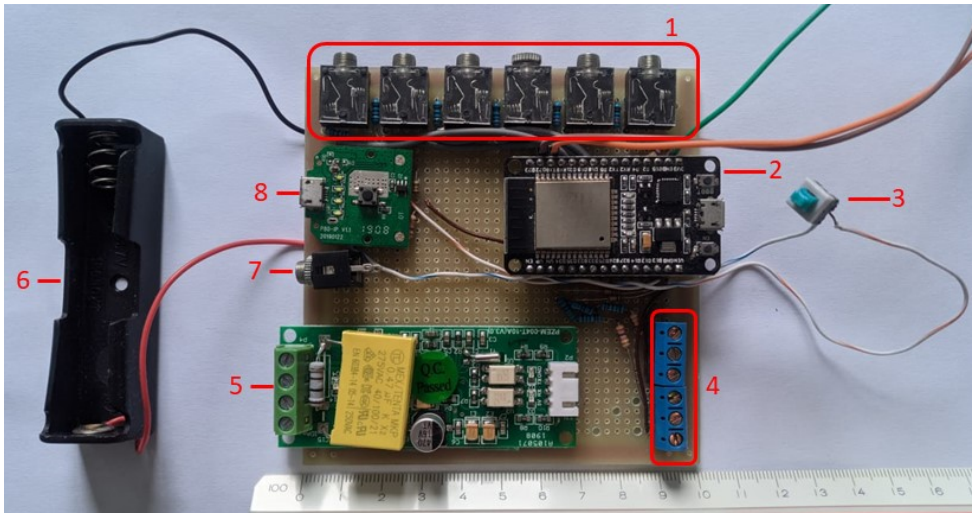
Fonte: autoria própria.

4.1 Hardware

O primeiro protótipo com os componentes de hardware foi desenvolvido numa placa de fenolite furada, com tamanho de 10cm x 10 cm. Na Figura 17 está a placa desenvolvida para testes dos sensores e desenvolvimento do código do software embarcado, com os principais componentes destacados numericamente na imagem, que são:

1. **Conectores P2 estéreo fêmea**. Com 6 unidades, usados para a conexão dos sensores de temperatura DS18B20. O modelo do conector é composto de 3 contatos de encaixe na recepção, sendo dois para alimentação (+ e -) e um para dados. Possui 5 contatos físicos para conexão na placa de circuito, sendo que 4 estão sendo usados: tensão positiva (+), terra (-), dados, e um para indicação de sensor conectado. Quando um conector P2 macho é inserido neste conector, internamente no circuito do conector um contato é aberto, alterando o estado de positivo (com tensão) para 0 (sem tensão) na saída do mesmo. Cada conector P2 tem uma dessas saídas conectadas em um GPIO da ESP32 para detecção dessa mudança de tensão, indicando se um

Figura 17 – Placa de desenvolvimento com seus elementos.



Fonte: autoria própria.

sensor foi conectado ou desconectado. As informações/dados de cada sensor DS18B20 são transmitidas para o mesmo GPIO da ESP32, sob o protocolo 1-Wire da Dallas Semiconductor Corp[1]. Esse protocolo, aliado a um identificador (ID) único de cada sensor DS18B20, permite que os dados dos 6 sensores sejam trafegados por uma única porta GPIO da ESP. Já os 6 GPIOs restantes, usados para detecção de mudança de tensão no encaixe de um sensor, permitem identificar em qual conector P2 o sensor foi inserido. O formato P2 estéreo macho possui 3 canais de contato, sem lado ou posições específicas de encaixe, facilitando o uso/conexão dos sensores.

2. **ESP32 modelo WROOM D**. Placa com microcontrolador descrito na sessão 2.3.2. Conforme o diagrama de blocos da Figura 18, 14 pinos estão sendo utilizados. Os pinos VIN para (alimentação 5V) e GND (terra); GPIO15 com sua porta configurada para comunicação digital 1-Wire com os 6 DS18B20; GPIOs 05, 16, 17, 18, 19 e 21 configurados como entrada digital para detecção de mudança de tensão sinalizando para interrupções na ESP que algo foi conectado; GPIOs 01 e 03 com protocolo UART[2] comunicando com o wattímetro; GPIOs 34 e 35 em modo analógico de leitura para detecção de nível de tensão emitidos pelos sensores de pressão, variando entre 0,5V e 3,3V; e GPIO32 também em modo de entrada analógico para detecção de nível de tensão da bateria para indicar percentual de carga;

3. **Botão liga/desliga**. Como o projeto visa a utilização em campo, utiliza-se alimentação por bateria e se faz necessário um botão para ligar/desligar a placa;

4. **Conexões dos sensores de pressão**. Os sensores utilizam 3 fios (+ ,- , e sinal analógico). Como serão fixos na placa, não há a necessidade de uma conexão de

[1] https://en.wikipedia.org/wiki/1-Wire
[2] https://docs.freebsd.org/pt-br/articles/serial-uart/

encaixe/desencaixe como os sensores de temperatura. Utilizam os GPIOs 34 e 35 da ESP32;

5. **Wattímetro PZEM 004t**. responsável pela medição das grandezas elétricas. Possui 4 conexões de entrada para medições, sendo duas para o sensor de corrente e duas para medição da tensão. Se comunica com a ESP32 através das portas 01 e 03, utilizando o protocolo UART;

6. **Compartimento para a bateria de lítio**. No formato 18650 (18mm x 65mm), para bateria cilíndrica;

7. **Conector P2 mono**. Usado para encaixe do sensor de corrente do Wattímetro, visando mais praticidade na hora de utilização do mesmo, sem necessidade de conectar e parafusar no borne/conector do wattímetro; Esse sensor vem do fabricante com fios expostos, sendo adicionado um conector P2 mono (2 canais) ao mesmo.

8. ***Step Up* e gerenciador de bateria**. Placa responsável por controlar o carregamento e descarregamento da bateria de lítio e elevar para tensão constante de 5V na saída, alimentando a ESP32.

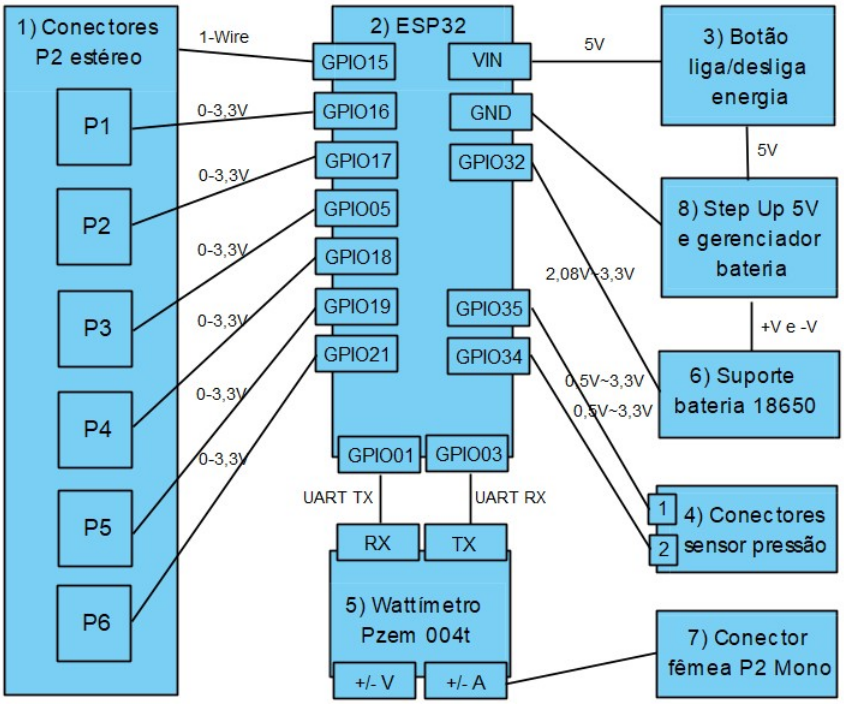
Tabela 9 – Custo do protótipo.

Componente	**Qtde**	**Custo Unit**	**Total:**
ESP32 WROOM	1	R$ 40,00	R$ 40,00
Wattímetro PZEM 004T	1	R$ 125,00	R$ 125,00
Sensor temperatura DS18B20	6	R$ 15,00	R$ 90,00
Placa Fenolite 10cmx10cm	1	R$ 20,00	R$ 20,00
Powerbank (bateria + circuito)	1	R$ 15,80	R$ 15,80
Sensor pressão	2	R$ 75,00	R$ 150,00
Conectores P2 estéreo fêmea	6	R$ 1,50	R$ 9,00
Conectores P2 estéreo macho	6	R$ 1,50	R$ 9,00
Conector P2 mono fêmea	1	R$ 1,50	R$ 1,50
Conector P2 mono macho	1	R$ 1,50	R$ 1,50
Conector borne KRE	2	R$ 3,50	R$ 7,00
Resistores	17	R$ 0,25	R$ 4,25
Botão liga/desliga	1	R$ 2,50	R$ 2,50
Suporte bateria lítio 18650	1	R$ 6,00	R$ 6,00
		TOTAL:	**R$ 481,55**

Fonte: autoria própria.

Na Tabela 9 estão os componentes usados e seus custos, não constando alguns itens como cabos metálicos, soldas e ferramental utilizado. Na Figura 18 está o diagrama de blocos do protótipo. Neste diagrama, são destacados os principais componentes e conexões, sendo omitidas conexões de alimentação. Essa placa provisória foi a principal ferramenta

Figura 18 – Diagrama de blocos do hardware.



Fonte: autoria própria.

utilizada para programação do software embarcado, enquanto ocorria paralelamente o desenvolvimento de uma placa de circuito impresso.

4.1.1 Protótipo com placa de circuito impresso

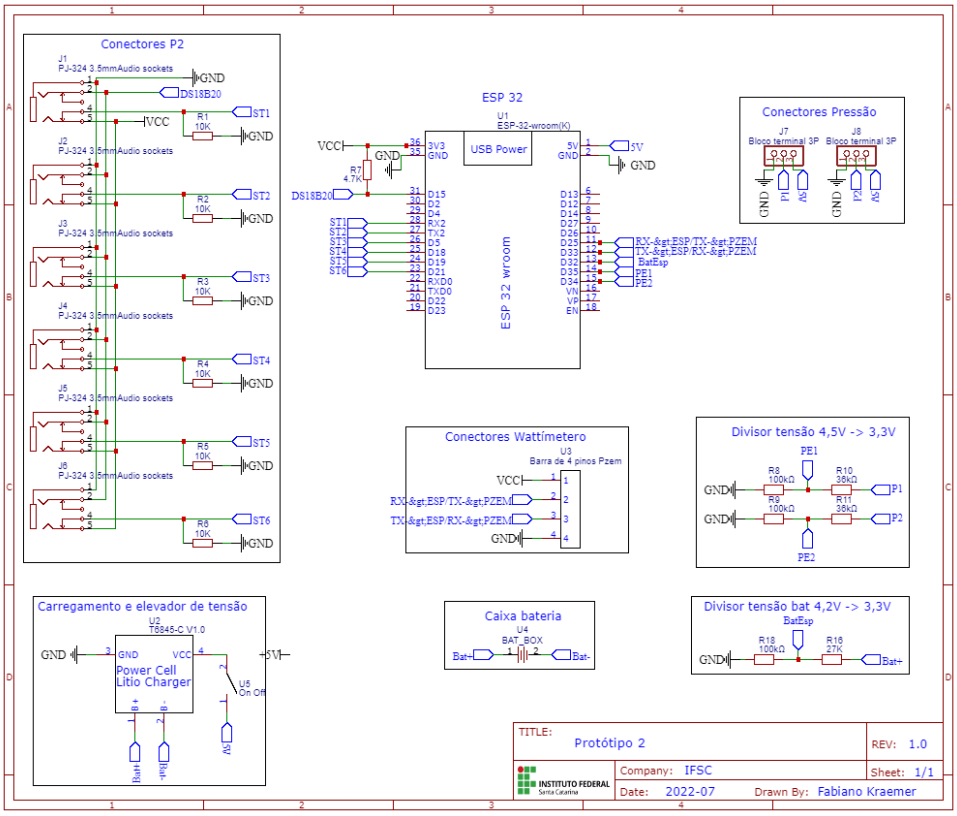
O projeto do segundo protótipo inclui uma placa de circuito impresso desenvolvido utilizando a ferramenta online EasyEDA. Seu esquemático está na Figura 19, o *layout* do PCB está na Figura 20 e a placa desenvolvida está na Figura 21.

4.2 Software embarcado

Analisando os itens expostos na seção 2.4, junto com os requisitos do projeto, verificou-se a necessidade da utilização de um sistema operacional de tempo real, tendo sido escolhido para este fim o sistema operacional de tempo real FreeRTOS. A linguagem de programação utilizada é C++. Além disso, no desenvolvimento do software embarcado, adotou-se as seguintes práticas (DENARDIN; BARRIQUELLO, 2019):

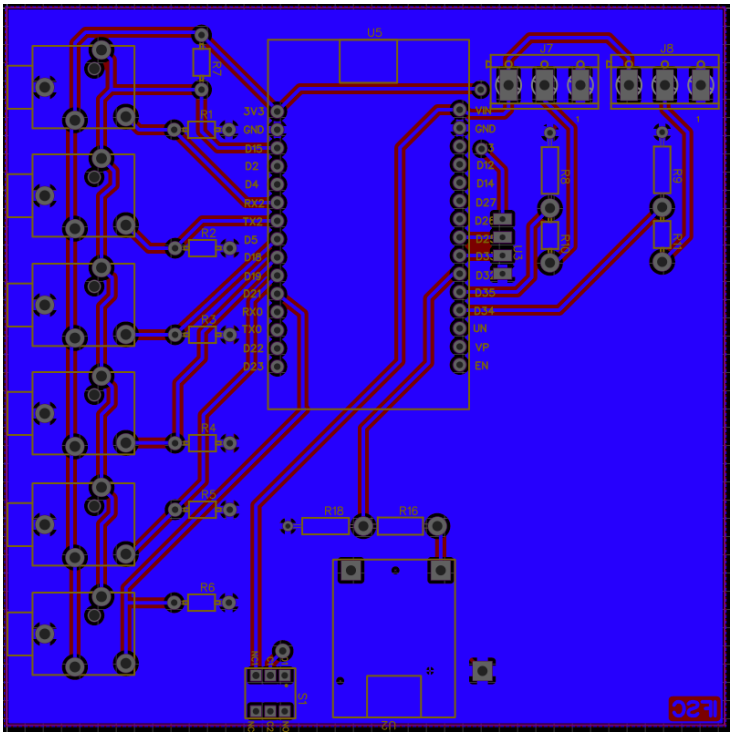
- Separação do código em vários arquivos para evitar o código “macarrônico”;
- Variáveis listadas em um arquivo *header* para não poluir visualmente o arquivo principal;

Figura 19 – Esquemático do protótipo com placa de circuito impresso.



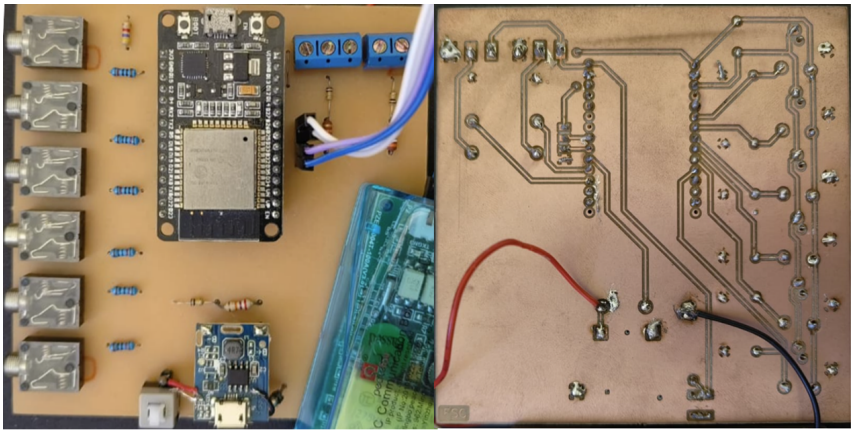
Fonte: autoria própria.

Figura 20 – Layout da placa de circuito impresso do protótipo.



Fonte: autoria própria.

Figura 21 – Protótipo 2.

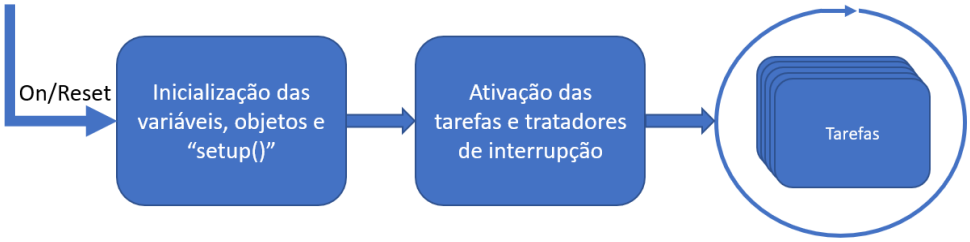


Fonte: autoria própria.

- Variáveis com nomes auto descritíveis, permitindo a fácil identificação sobre o que ou onde ela se refere;
- Código priorizando a facilidade de leitura, não a de escrita;
- Utilização de comentários para descrever as funções e variáveis;
- Arquivo *header* com *includes* para não poluir visualmente o arquivo principal;
- Variáveis globais e estáticas: embora o uso de variáveis globais não seja recomendada, em projetos embarcados a utilização com o devido cuidado delas pode ser interessante, junto com o modificador *static*. Isso permite que o compilador saiba de maneira mais efetiva, em tempo de compilação, o total de memória que será utilizada, auxiliando o programador em saber qual o tamanho da *stack* e *heap* da RAM, podendo alocar os recursos mais eficientemente. Uma variável estática é um local de armazenamento de dados no código. Seu conteúdo pode ser alterado e existirá durante toda a execução da aplicação, não importa onde ela tenha sido declarada. O local de declaração dela define o escopo, a visibilidade da variável, mas não o tempo de vida;
- Utilização de semáforos para acesso concorrente das variáveis, permitindo correto manuseio das mesmas, evitando conflitos de acesso e modificação por serem variáveis globais;

Na Figura 22 está um diagrama simplificado do fluxo do software desenvolvido, para visualização do comportamento geral do sistema. Com exceção das interrupções, todas as funções são controladas pelas *tasks*. As **tasks** (tarefas) são fluxos de execução independentes dentro do programa, similares a threads. O comportamento de cada *task* é definido em uma função, onde cada uma efetua operações específicas e isoladas, possuindo *loops* infinitos de execução e que nunca retornarão um valor. O FreeRTOS, através do escalonador, fará a troca de contexto entre as tarefas, dando a impressão de que todas

Figura 22 – Diagrama de fluxo do software embarcado.



Fonte: Autoria própria.

estão sendo executadas ao mesmo tempo, permitindo que códigos e funções rodem de forma "paralela" na percepção do usuário. Seu funcionamento será melhor detalhado na subseção 4.2.3. Nas seções seguintes serão explicados os elementos que compõem o sistema.

4.2.1 Inicialização e Setup

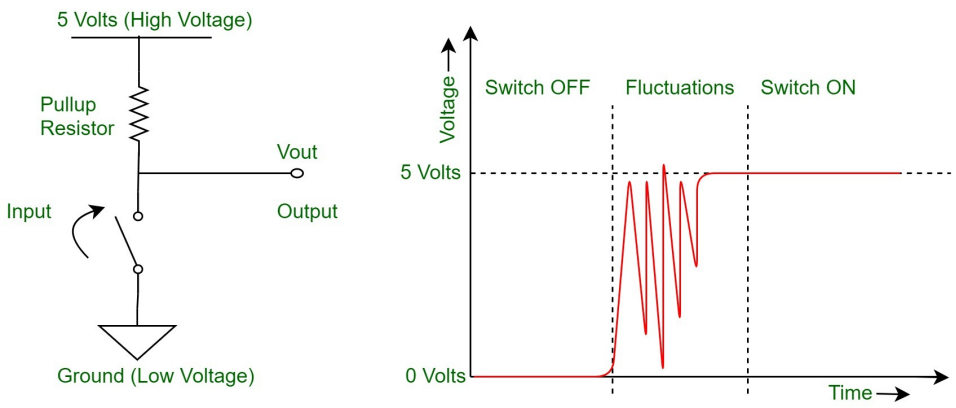
Toda vez que o dispositivo é ligado ou reiniciado, o programa criará alguns objetos para funções específicas do código, como a criação das interrupções e dos semáforos. Em sequência, a função *setup* é executada. Nela são iniciados a comunicação serial, o modo dos pinos da ESP32, variáveis com valores padrão na estrutura JSON, a inicialização das interrupções, da biblioteca dos sensores de temperatura, dos semáforos, do *WiFi* e configuração para conexão com o *Broker* MQTT. Finalmente, no final são criadas as *Tasks*. Após esse procedimento, essa função do código não será mais executada enquanto o dispositivo permanecer ligado. O fluxo de execução do programa é passado para o escalonador, que definirá qual tarefa será executada a cada momento. O escalonador tem como função implementar os critérios da política de escalonamento. Todo o compartilhamento do processador depende desta funcionalidade. O escalonamento é de responsabilidade do FreeRTOS, cabendo ao programador a definição de algumas regras, como o nível de prioridade de cada tarefa.

4.2.2 Interrupção por hardware

Para o software detectar quando um sensor de temperatura DS18B20 foi conectado ou desconectado do sistema embarcado, foi implementado interrupção por hardware. Conforme o diagrama da Figura 18, são 6 os pinos de entrada dos sensores de temperatura. Para cada um deles, foi criado uma rotina de interrupção específica. No hardware, ao se pressionar um botão ou inserir um conector, pode ocorrer um efeito indesejável chamado "bouncing", ou repique, que é quando o hardware, por instabilidade mecânica, interpreta múltiplos toques quando na verdade só ouve um pressionamento do botão ou encaixe de um conector. Este fenômeno está ilustrado na Figura 23.

Para contornar esse efeito, pode-se adotar medidas contentoras via hardware ou

Figura 23 – Efeito de *bouncing* (repique).



Fonte: (GEEKSFORGEEKS.ORG, 2016).

software. Pela simplicidade e eficiência, a opção escolhida foi via software, onde, ao ocorrer a interrupção pela primeira vez, é determinado um intervalo de tempo que ignorará qualquer novo ativamento da interrupção. Esse tempo está definido em 300 milissegundos. Após o procedimento, um vetor booleano de 6 posições é alterado com o valor correspondente (*true* para conectado e *false* para desconectado) em sua respectiva posição. Esse vetor é utilizado pelas *Tasks* para saber em qual posição o sensor está e gravar o dado da leitura na posição corretamente. O acesso a ele é protegido por um semáforo. Em essência, a rotina de interrupção implementada serve para atualizar um vetor booleano de 6 elementos, indicando qual pino foi encaixado ou retirado um sensor.

4.2.3 Tarefas/Tasks

A criação das Tasks é realizada no *Setup* do código. Envolve a passagem de parâmetros conforme código da listagem 1. Na Tabela 10 estão os parâmetros e dados medidos das cinco tarefas criadas, como tamanho ocupado e tempo médio de execução das funções internas.

```
1  xTaskCreate(
2    nome_funcao,     // Função a ser chamada
3    "nome da tarefa",  // Nome da tarefa
4    stack_size,      // Tamanho máximo disponível(bytes)
5    NULL,            // Parametro a ser passado
6    3,               // Prioridade da tarefa, maior o número, maior a prioridade
7    &xTask_Handle    // Task handle
8    0,               // Núcleo da ESP que deseja rodar a tarefa (0 ou 1)
9  );
```

Listagem 1 – Parâmetros para criação de uma Task. Fonte: autoria própria.

Entre esses parâmetros, destaca-se o tamanho máximo de memória alocada para a pilha da tarefa (*stack_ size*), a prioridade da mesma e o núcleo da *CPU* que ela rodará.

Tabela 10 – Dados e parâmetros das tarefas criadas.

Nome da Tarefa	Tamanho da pilha (bytes)	Prioridade	Núcleo da CPU	Período de execução (ms)	Tempo máximo de execução medido (ms)	Tamanho máximo de pilha (bytes)
Conexões Wireless	3000	3	0	1000	4,11	1890
Ler sensores	2000	1	1	2000	1480	1040
Enviar dados	2000	1	0	3000	4,2	1200
Receber comandos	2000	2	0	1000	2,11	1490
Debug	3000	1	0	1000	85,1	1210

Fonte: autoria própria.

A prioridade de cada tarefa foi definida na etapa de projeto do sistema. O padrão do FreeRTOS é de quanto maior o número, maior será a prioridade da tarefa, e pode assumir qualquer valor dentro do razoável - mas por motivos de eficiência de uso de RAM, deve-se manter o valor mínimo realmente necessário[3]. Já para a alocação de memória, na etapa de desenvolvimento foi atribuído um valor arbitrário elevado para que ela rodasse sem problemas. Durante a etapa de testes foi utilizado uma função do FreeRTOS para verificação de quanta memória estava disponível do total alocado para a tarefa. Com essa função e os testes realizados (Capítulo 5), foi possível dimensionar o tamanho adequado da pilha da tarefa, com um adicional de no mínimo 10% como margem de segurança. Os valores da quantidade de memória ocupada por cada Task foram armazenados em um objeto JSON e enviados para o Broker MQTT na nuvem, salvos no banco de dados e analisados usando o Grafana. Na listagem 2 estão exemplificados os códigos utilizados para essa verificação.

```
1 // Variável para armazenar a informação do consumo máximo de memória de uma task.
2 UBaseType_t uxHighWaterMark;
3
4 // Pega o valor em bytes de memória disponível da respectiva tarefa.
5 uxHighWaterMark = uxTaskGetStackHighWaterMark(xTask_Handle);
```

Listagem 2 – Verificação de memória ocupada pela tarefa. Fonte: autoria própria.

Dentro da função passada na criação da Task existe um *for*, de *loop* infinito que é ativado periodicamente. Isso garante que essa função será executada repetidamente enquanto não houverem outras condicionantes que alterem o estado e/ou o funcionamento da respectiva função e Task. A duração de tempo de cada ciclo (*loop*) do *for* é determinado pelo tempo de processamento do código escrito nele. Porém, em determinadas situações, o programador deseja que esse ciclo seja executado em intervalos de tempo regulares. Por exemplo, uma tarefa que verifique o estado da conexão WiFi pode ser feita a um intervalo de 5 segundos. Para esse controle, o FreeRTOS fornece uma função chamada **vTaskDelay**. Ao chamá-la, passa-se como parâmetro o intervalo de tempo que deseja que a Task fique ociosa, sendo o intervalo medido em *Ticks*. O kernel em tempo real do

[3] https://www.freertos.org/RTOS-task-priority.html

FreeRTOS mede o tempo usando essa variável, e no caso da ESP32, cada Tick[4] possui uma duração de 10 milissegundos, ou seja, é um sinal periódico de 100hz. Entretanto, caso uma operação realizada dentro do *for* leve um tempo elevado para ser executada, esse tempo de processamento deverá ser adicionado ao tempo de *delay* definido na função vTaskDelay. Se, por exemplo, o requisito do projeto exija que a tarefa seja executada em intervalos regulares de 5 segundos, e na função do vTaskDelay seja informado esse tempo, mas o tempo de processamento do ciclo leve mais de 500 milissegundos, o tempo real de intervalo de execução da Task será de no mínimo 5,5 segundos. Esse adicional de tempo, causado pelo tempo de processamento é chamado de *jitter*. Para reduzir esse efeito, em cada tarefa foi calculado o tempo (em Ticks) que o ciclo levou para ser executado, e no final do *for* é usado a função **vTaskDelayUntil**, que garante uma execução em intervalos constantes. Na listagem 3 estão exemplificados os códigos utilizados para esse objetivo.

```
1  // Transforma o tempo de delay de milissegundos para ticks
2  const TickType_t xDelay = pdMS_TO_TICKS(tempo_taskDelay);
3
4  // Variável para pegar os ticks do escalonador do FreeRTOS.
5  TickType_t xLastWakeTime;
6
7  for (;;){
8      xLastWakeTime = xTaskGetTickCount();
9      .......
10     vTaskDelayUntil(&xLastWakeTime, xDelay);
11 }
```

Listagem 3 – Reduzir jitter. Fonte: autoria própria.

4.2.3.1 Semáforos

Para proteger as variáveis e funções do acesso concorrente entre as Tasks, foram utilizados 4 semáforos diferentes, todos do tipo *mutex*[5]. Semáforos desse tipo indicam para o escalonador do FreeRTOS que o intervalo entre *take* e *give* possui prioridade máxima entre as tarefas envolvidas na disputa pelo semáforo, ou seja, utiliza o protocolo de herança de prioridades. Com exceção do semáforo das interrupções, os outros 3 são declarados no *main.cpp* e inicializados no *setup*. Na listagem 4 estão a declaração dos três semáforos utilizados.

4.2.3.2 Task conexões wireless

Tarefa responsável por verificar os estados das conexões *WiFi*, *Bluetooth*, gerenciar o serviço de seleção de rede WiFi e o *server handle* do serviço OTA. Conecta/reconecta em

[4] https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html
[5] No FreeRTOS, “mutexes” são implementados como semáforos binários.

```
SemaphoreHandle_t mutexJSON; // Mutex que protege o acesso ao objeto JSON.
SemaphoreHandle_t mutexWifi; // Mutex para ativar ou suspender tasks que dependem
    das conexão WiFi e Bluetooth.
SemaphoreHandle_t mutexMQTT; // Mutex para lidar com as chamadas do client do MQTT.
```

Listagem 4 – Semáforos utilizados. Fonte: autoria própria.

conexões salvas previamente e ativa o gerenciador de conexões caso não consiga estabelecê-las. Possui a maior prioridade entre as tarefas, pois é também responsável por habilitar ou suspender as outras. Esse controle é feito usando um semáforo. Caso o sistema esteja conectado em uma rede sem fio, a função libera o semáforo (*give*). Caso contrário, ela toma para si o semáforo (*take*), e permanece assim até que estabeleça uma conexão. As outras tarefas possuem o mesmo semáforo logo no inicio de seu ciclo de execução, solicitando-o e liberando-o logo em seguida. Caso o semáforo não esteja disponível, as tarefas são colocadas em suspensão automaticamente pelo FreeRTOS, sendo reativadas assim que o semáforo é liberado. Essa suspensão foi adotada dado que não é necessário a execução das mesmas se não existir uma conexão sem fio ativa, pois os dados não seriam enviados e não são armazenados em *buffers* na memória da ESP32. Na listagem 5 está exemplificado esse sistema utilizado para o controle de execução das tarefas.

```
void conexoes_wireless(void *pvParameters){
  for (;;){
    if (WiFi.status() == desconectado){
      xSemaphoreTake(mutexWifi, portMAX_DELAY );
    }
    if (WiFi.status() == conectado){
      xSemaphoreGive(mutexWifi);
    }
  }
}

outras_tasks(void *pvParameters){
  for (;;){
    xSemaphoreTake(mutexWifi, portMAX_DELAY );
    xSemaphoreGive(mutexWifi);
    ......
  }
}
```

Listagem 5 – Controle das tasks pelos semáforos. Fonte: autoria própria.

A tarefa possui *delay* de execução de 1000 milissegundos. Conforme os testes realizados mostrados na seção 5.1, o tempo médio de processamento da tarefa é inferior a 5 milissegundos, e seu máximo de pilha ocupada é de 1892 bytes, sendo alocado o total de 3000 bytes de memória de pilha. É executada no núcleo 0 da ESP32. Nesta tarefa destacam-se duas funções que foram desenvolvidas:

- **WiFi**: Função que tenta conectar em uma rede salva na memória *flash* da ESP32. Caso não consiga, ativa o portal do gerenciador de conexões, permitindo que um celular se conecte na ESP32 e, através do navegador, rastreie as conexões WiFi disponíveis e faça login, salvando a rede na memória *flash*.

- **Reconnect**: Conecta no broker MQTT.

4.2.3.3 Task ler sensores

Tarefa responsável por chamar as funções de leitura dos sensores e armazenar as informações em um objeto JSON. Possui *delay* de execução de 2000 milissegundos. O tamanho máximo ocupado da pilha é de 1036 bytes, sendo alocado o total de 2000 bytes de memória. Possui prioridade 1 e é executada no núcleo 1 da ESP32, sendo a única tarefa a rodar nesse núcleo, pois, conforme os testes realizados, seu tempo de processamento é superior ao tempo de todas as outras tarefas somadas, ficando na média de 1180 milissegundos, com picos de 1500 milissegundos. É responsável por chamar 4 funções, descritas abaixo:

- **Índice bateria**: Faz uma leitura analógica do GPIO32, convertendo a informação para tensão e percentual da bateria. É um método simples e rápido, mas não o mais preciso para saber a capacidade real e estado da bateria. Métodos mais precisos precisam da adição de circuitos específicos, encarecendo e aumentando o nível de complexidade do projeto[6].

- **Temperatura**: Função para leitura dos sensores DS18B20. Utiliza a biblioteca padrão do fabricante do sensor, sendo a função com o maior tempo de processamento de todo o projeto, causado justamente por essa biblioteca. Com apenas um sensor de temperatura, leva em média 550 milissegundos de tempo de processamento, aumentando cerca de 100 milissegundos para cada sensor adicionado.

- **Pressão**: Faz 50 leituras analógicas dos GPIOs 34 e 35, tira a média, converte para unidade de pressão Bar[7] e armazena no objeto JSON.

- **Wattímetro**: Faz a leitura dos dados do wattímetro Pzem 004t utilizando a biblioteca do fabricante do dispositivo. É a segunda função com maior tempo de processamento, levando em média 98 milissegundos.

4.2.3.4 Task enviar dados

Tarefa responsável por serializar os dados armazenados no objeto JSON e enviá-los para o *broker* MQTT via WiFi e para o aplicativo do celular via Bluetooth. Possui tempo

[6] https://embarcados.com.br/medicao-do-estado-da-carga/
[7] https://pt.wikipedia.org/wiki/Bar_(unidade)

de processamento inferior a 10 milissegundos, ocupando 1196 bytes de pilha, com alocação de 2000 bytes e tempo de ciclo de 3000 milissegundos. É executada no núcleo 0 e possui prioridade 1.

4.2.3.5 Task receber comandos

Verifica se recebeu comandos do usuário pelo WiFi ou Bluetooth, armazenando as informações em um objeto JSON, e executa os comandos, como por exemplo, apagar os dados das redes WiFi salvas ou reiniciar a ESP32. Possui tempo de execução inferior a 3 milissegundos, sendo seu ciclo definido para rodar a cada 1000 milissegundos. Ocupa em média 1350 bytes de pilha, com picos de 1492 bytes, sendo alocados 2000 bytes para a tarefa. É executada no núcleo 0 e possui prioridade 2.

4.2.3.6 Task debug para a nuvem

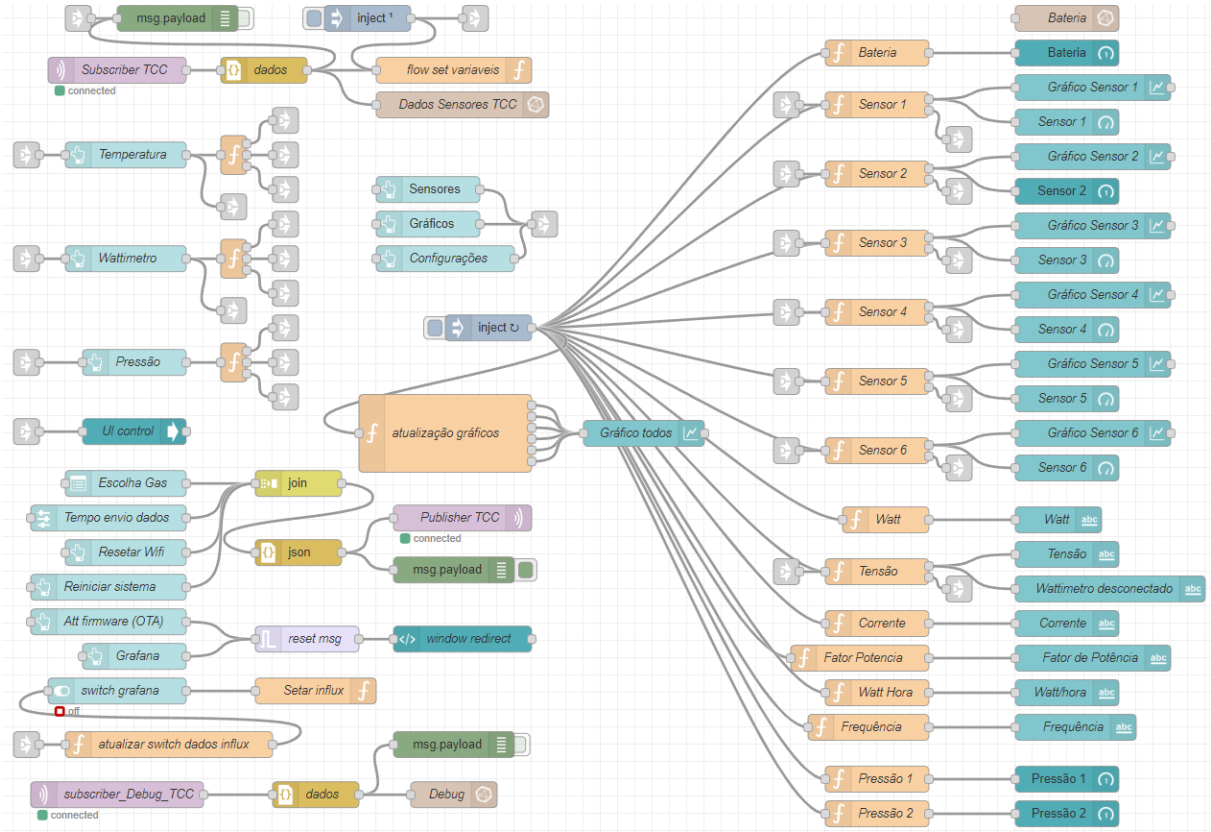
Esta tarefa foi criada para monitorar dados do comportamento do sistema embarcado. Ela verifica informações como memória/pilha ocupada pelas tarefas, memória total livre, potência do sinal WiFi e os tempos de processamento das tarefas. Esses dados são armazenados em um objeto JSON, serializados e enviados para a nuvem via MQTT e conexão WiFi. O tempo do ciclo da tarefa está definido para 1000 milissegundos, e possui tempo de execução médio de 9 milissegundos, ocupando 1208 bytes de pilha, sendo alocados 3000 bytes para a tarefa. Esse valor está bem acima da média do ocupado nos testes, porém, ao usar valores menores (2000 bytes por exemplo), a tarefa apresentava erros, pois em alguns testes na sua primeira execução existia um pico que passava de 2000 bytes, estabilizando posteriormente em valores menores. É executada no núcleo 0 e possui prioridade 1.

4.3 Nuvem

Os serviços de nuvem estão rodando em uma Raspberry Pi 3B[8] instalada em uma residência com conexão de internet com IP dinâmico. Para poder acessar o Node-RED e Grafana, sempre pelo mesmo endereço, foi utilizado um serviço de **DDNS**[9], o **Duck DNS**[10]. Esse serviço faz o roteamento da URL para o IP do modem local, e no modem, foi criado uma regra para o redirecionamento do IP local para o da *Raspberry* e a porta do Node-RED e Grafana. Para avisar o serviço de DDNS quando o IP local muda, foi criado um script na *Raspberry* que roda a cada 5 minutos, verificando se houve mudança de IP.

[8] https://www.raspberrypi.com/products/raspberry-pi-3-model-b/
[9] https://pt.wikipedia.org/wiki/DNS_din%C3%A2mico
[10] https://www.duckdns.org/

Figura 24 – *Flow* do Node-RED.



Fonte: autoria própria.

4.3.1 Node-RED

A criação do *broker* MQTT e banco de dados InfluxDB foi realizada na plataforma Node-RED, assim como o serviço de atualização remota do *firmware*, *subscribe* e *publisher* do MQTT. Na Figura 24 está o *flow* criado do Node-RED com seus elementos.

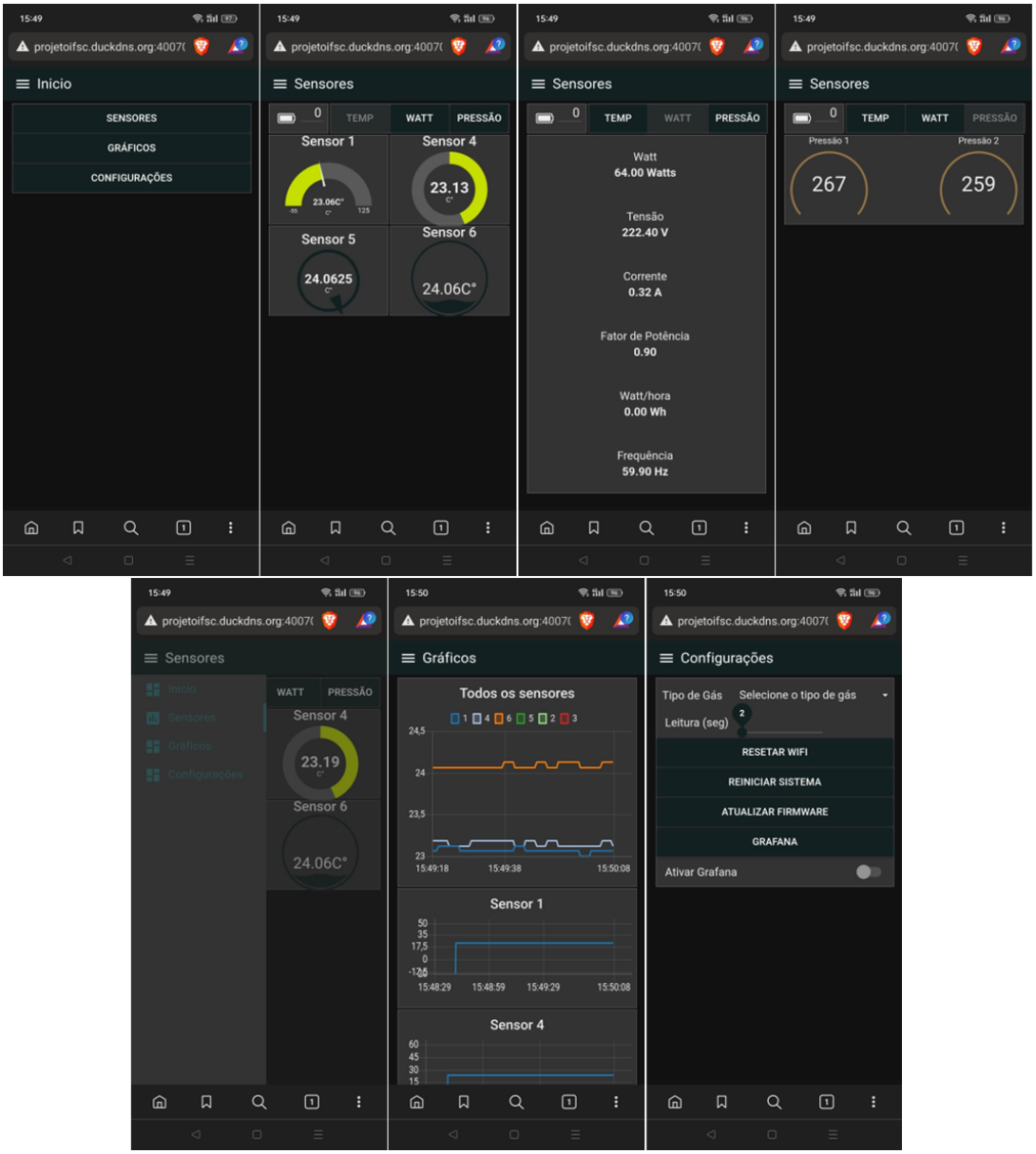
Para o usuário utilizar o dispositivo pelo navegador, foi desenvolvido uma interface de usuário usando as ferramentas do Node-RED. Essa interface pode ser acessada através do endereço *http://projetoifsc.duckdns.org:40070/ui* [11]. Na Figura 25 estão as janelas da interface de usuário criadas para operação do sistema embarcado. O layout dos gráficos é temporário, possuindo diversos estilos como sugestão para escolha. O acesso para o Grafana é realizado no link *http://projetoifsc.duckdns.org:40090/d/nBcnsWRgk/tcc?orgId=2* [12].

4.3.2 Atualização remota OTA

Aproveitando os recursos WiFi da ESP32, foram utilizados dois métodos para atualização do software embarcado de maneira remota, chamado popularmente de **OTA** (over-the-air)[13]. O primeiro método, mais simples, cria um servidor HTTP na ESP32 para

[11] Login: projetoIFSC. Senha: projetoIFSC
[12] Login: TCC IFSC. Senha: projetoIFSC
[13] https://www.techtarget.com/searchmobilecomputing/definition/OTA-update-over-the-air-update

Figura 25 – Interface de usuário do Node-RED.

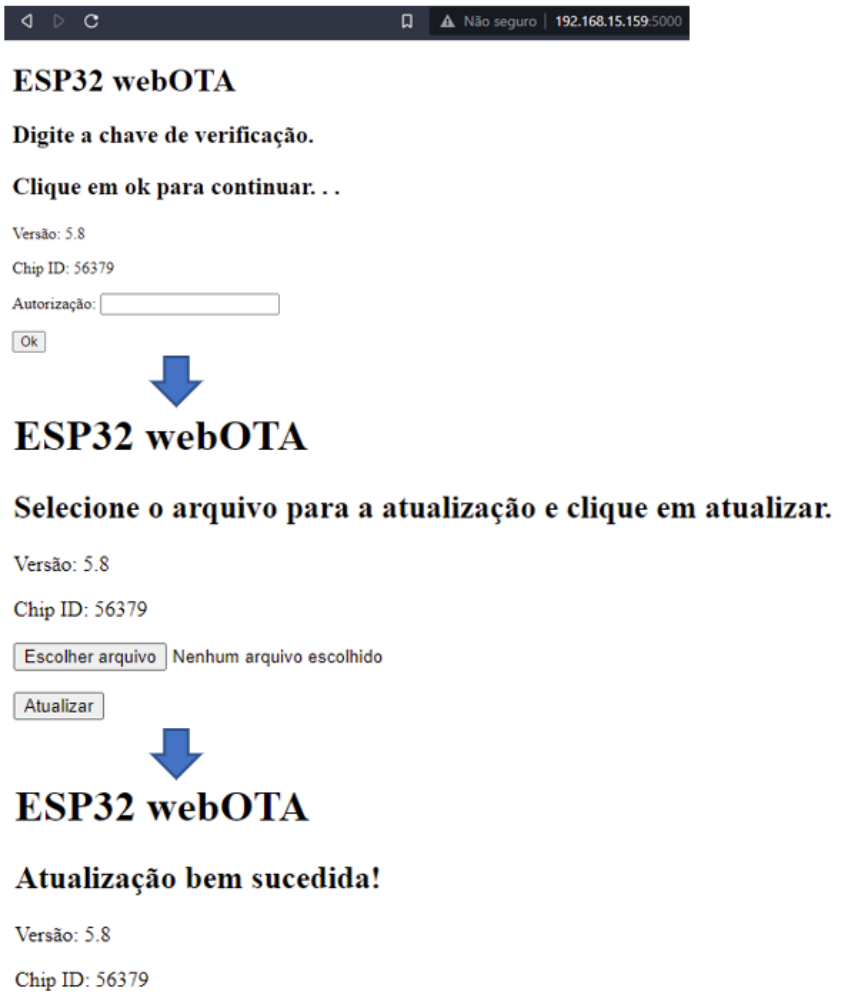


Fonte: autoria própria.

acesso via rede local. Acessando ele, introduz-se uma senha, seguida da escolha do arquivo binário e finalizando pela confirmação da atualização. Na Figura 26 estão as janelas do procedimento.

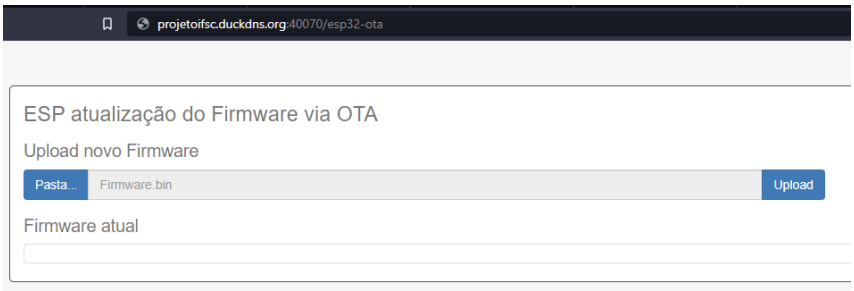
O procedimento de atualização via rede local é simples e eficaz, mas tem-se o problema de só poder ser realizado caso o desenvolvedor esteja conectado na mesma rede local. Um produto em um cenário mais real, em que diversas unidades do dispositivo estão nos mais variados locais e redes de internet diferentes, não seria viável a adoção deste procedimento, pois não seria possível acessar tais dispositivos devido a fatores como IPs dinâmicos, regras de NAT e outros problemas relacionados a roteamento em redes IP. Para contornar esses problemas, a opção adotada foi usar o caminho inverso: ao invés

Figura 26 – Atualização local.



Fonte: autoria própria.

Figura 27 – Atualização remota.



Fonte: autoria própria.

do desenvolvedor conectar na ESP32 e atualizá-la, é a própria ESP que conecta em um servidor remoto, com endereço fixo. Nesse servidor, a ESP verifica as versões de *firmware* existentes, e se for identificado uma nova versão, ela automaticamente realiza o download do arquivo e se atualiza. Na Figura 27 está a janela do procedimento.

5 TESTES E RESULTADOS

Neste capítulo serão descritos os testes realizados e seus resultados. Para as informações dos sensores de temperatura, wattímetro e dados do sistema embarcado, foram utilizadas as próprias ferramentas desenvolvidas para o projeto. Durante a rotina de testes, os dados são enviados via WiFi para o servidor MQTT, gravados no InfluxDB e visualizados no Grafana. Para o sensor de pressão, por ser um dispositivo adquirido na China, com poucas informações e sem dados do fabricante, foi realizado uma homologação pelo Laboratório de Pesquisa em Refrigeração e Termofísica - UFSC (POLO)[1].

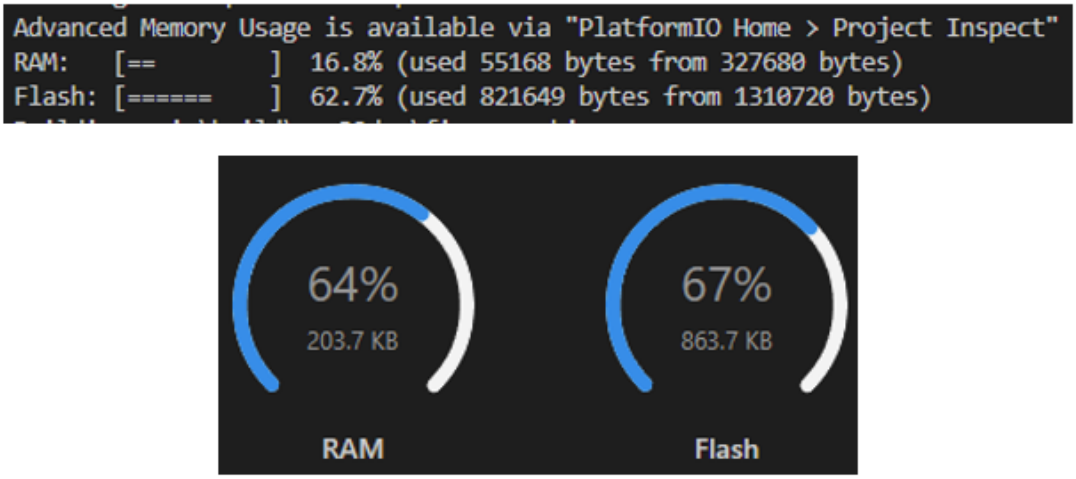
Os aplicativos desenvolvidos em ESP32 usam os padrões comuns de arquitetura de computador de pilha (memória dinâmica alocada pelo fluxo de controle do programa) e *heap* (memória dinâmica alocada por chamadas de função), bem como memória alocada estaticamente (alocada em tempo de compilação)[2]. Com a utilização do FreeRTOS cada tarefa tem sua própria pilha. Por padrão, cada uma dessas pilhas é alocada do *heap* quando a tarefa é criada. Como o ESP32 usa vários tipos de RAM, ele também contém vários *heaps* com recursos diferentes. Um alocador de memória permite que os aplicativos façam alocações de *heaps* para diferentes propósitos. Na inicialização, o *heap DRAM* contém toda a memória de dados que não é alocada estaticamente pelo aplicativo.

A verificação de quanta memória *flash* e RAM foi utilizada pelo sistema desenvolvido foi feita de duas formas, sendo as duas através da ferramenta de desenvolvimento **PlatformIO**, extensão utilizada no Visual Studio Code para desenvolver aplicações baseadas na IDE do Arduino. No término da compilação do software, a ferramenta apresenta na janela do terminal as informações de quanta RAM e Flash são utilizadas pelo código. O PlatformIO também fornece uma ferramenta de inspeção dessas informações, chamada de *Inspect*. Na Figura 28 estão os valores e conforme observado, são divergentes, principalmente na RAM utilizada. Devido a falta de documentação e informações da extensão, não foi identificado o motivo dessa diferença. Porém, em qualquer dos dados levantados, houveram valores livres de RAM suficientes para que não afetasse a execução do código do sistema embarcado.

Além dos testes teóricos, todos os sensores de temperatura e wattímetro foram utilizados em ensaios controlados utilizando o sistema de refrigeração criado para a bolsa de iniciação científica descrita na introdução deste trabalho. Foram realizados 11 ensaios, com duração variando entre 4 e 8 horas em cada ensaio e alterações no sistema de refrigeração, como, por exemplo, o uso de ventilação forçada no condensador. Esses testes de validação dos sensores de temperatura e pressão não utilizaram os transdutores de pressão chineses e

[1] https://www.polo.ufsc.br/
[2] https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/mem_alloc.html

Figura 28 – RAM e Flash ocupadas conforme terminal e ferramenta no PlatformIO.



Fonte: autoria própria.

foram feitos no hardware e software antigos, portanto não serão mostrados neste capítulo. Entretanto serviram para atestar a confiabilidade dos DS18B20 e Wattímetro.

5.1 Tempo de execução e tamanho das Tasks

O teste foi realizado durante um período de aproximadamente 27 horas, com amostras sendo realizadas a cada segundo, totalizando mais de 90.000 leituras. Na Figura 29 estão os gráficos com as informações do tempo e tamanho das tarefas, e a memória *heap* total disponível do sistema. Por usar a própria ESP32 para levantamentos dos dados, não é o teste mais acurado que pode-se fazer, porém, com as limitações de recursos e equipamentos disponíveis, essa metodologia adotada fornece dados interessantes sobre o comportamento do sistema, sendo demonstrado nos tópicos seguintes.

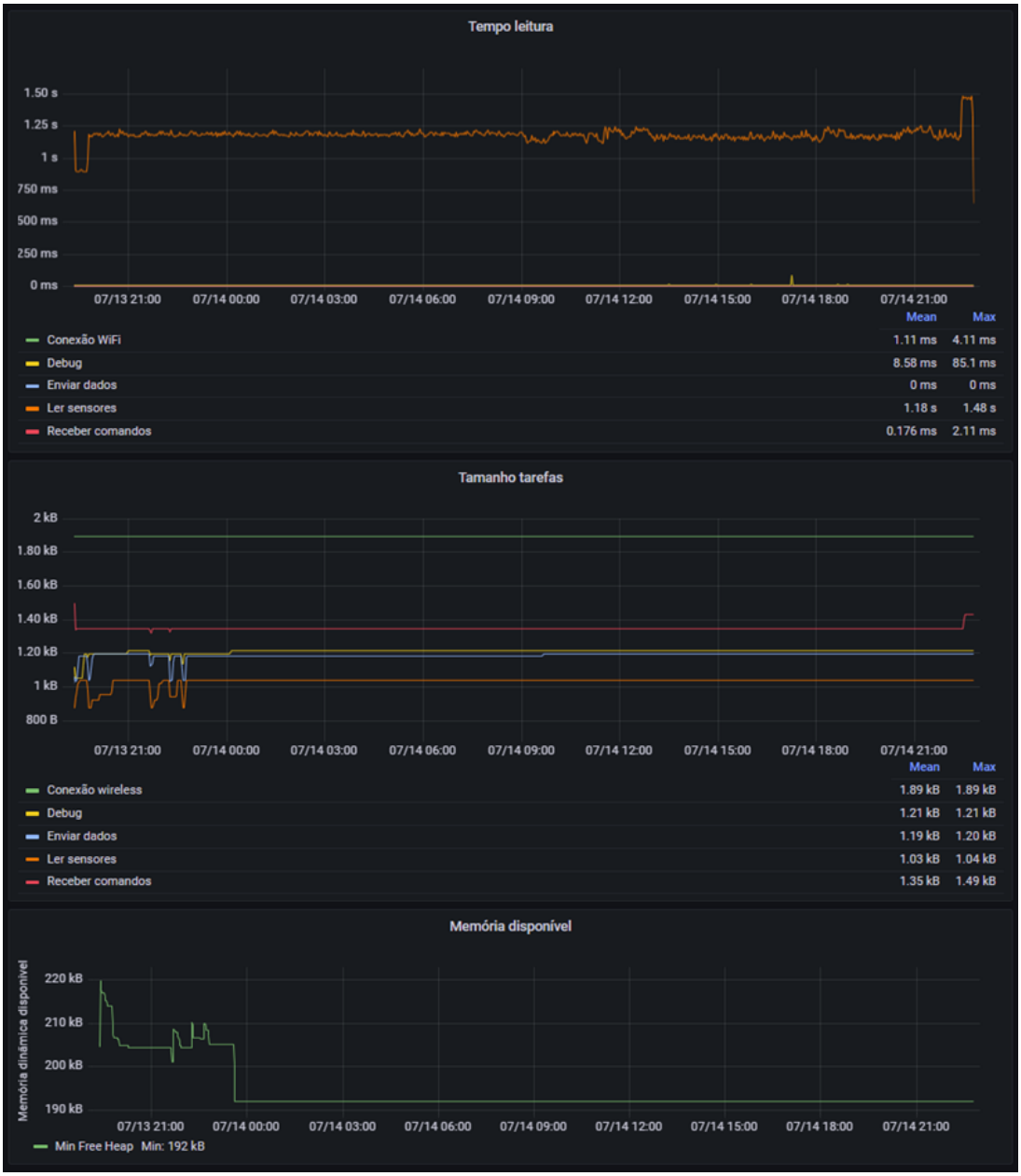
5.2 Sensor de temperatura DS18B20

Para os testes dos sensores de temperatura, foram utilizados sensores de temperatura NTC analógicos para comparativo. Como o sistema embarcado desenvolvido não contempla tais sensores, os testes considerados são do circuito usado na bolsa de iniciação científica descrita na introdução. O comparativo está na Figura 30. Na Figura 31 é mostrado o comportamento dos sensores no teste em um sistema de refrigeração.

5.3 Sensores de pressão

As informações do teste e validação estão nas Tabela 11 e Tabela 12. Conforme verificado na Tabela 13, o desvio do estimado (Pe) para o medido (Pm) foi inferior a 0,5%.

Figura 29 – Gráficos do teste de tempo de execução e tamanho das tarefas.



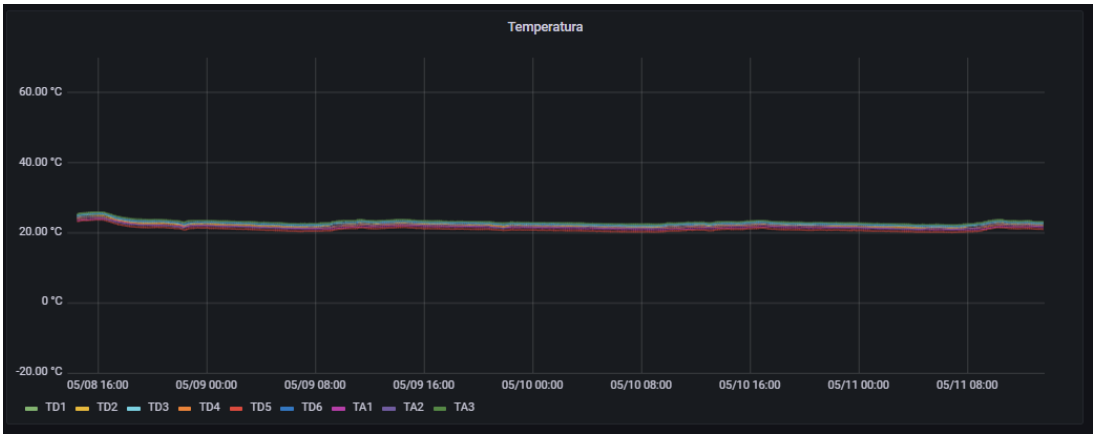
Fonte: autoria própria.

Tabela 11 – Dados do transdutor de pressão aferido.

Requerente:	Laboratório Aplicação - POLO
Data de calibração	16/09/2021
Modelo	Transd.Press.Arduino_34.47bar_5V
Número de série	163510227
Faixa de operação	34,47 bar
Incremento digital	0,001
Responsável	Jorge Lubas

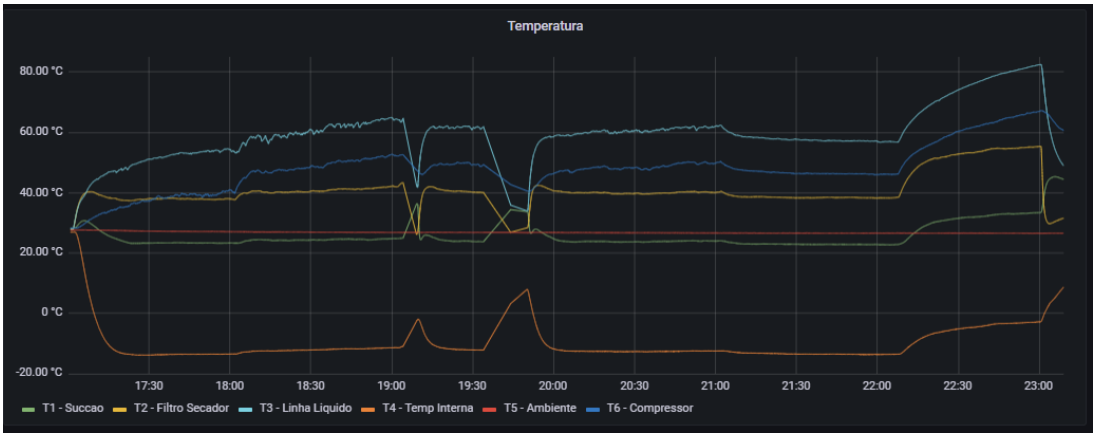
Elaboração: Autor. Fonte: POLO - UFSC.

Figura 30 – Gráficos do teste dos sensores DS18B20 e NTC analógicos.



Fonte: autoria própria.

Figura 31 – Gráficos do teste em um sistema de refrigeração.



Fonte: autoria própria.

Tabela 12 – Parâmetros de aferição do sensor de pressão.

Pressão atmosférica [bar]	1,025
Temperatura ambiente [°C]	22,6 +- 0,8
Fonte do transdutor [V]	5,01
Sistema de medição padrão	Máquina de peso morto DH-Budenberg 580 Series
Sistema de aquisição	Agilent Technologies / Data Aquisition - 34970 A

Elaboração: Autor. Fonte: POLO - UFSC.

5.4 Wattímetro PZEM-004T

O teste do wattímetro, devido a falta de equipamentos precisos para comparação, envolveu o uso de uma resistência com potência conhecida teórica de 1300 Watts e comparação com um medidor de consumo do tipo tomada inteligente. Os dados da medição estão na Figura 32, e conforme observado, estão próximos do valor teórico da resistência, lendo em média 1360 Watts. O dispositivo usado para comparação apresentou uma leitura de 1404 Watts, uma diferença de 3%.

Tabela 13 – Resultados da aferição do sensor de pressão.

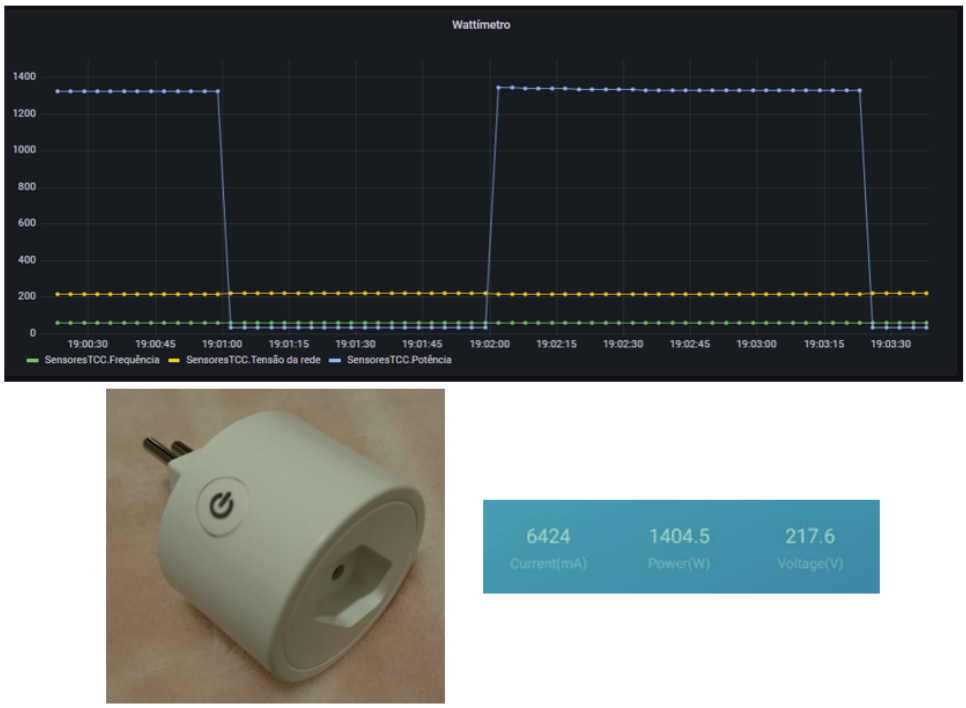
V [Tensão}	Pm [bar]	Pe [bar]	Desvio (%)
0,614781	2,024805	2,01483	0,49%
0,964942	5,024498	5,023708	0,02%
1,428756	9,023766	9,009182	0,16%
1,894353	13,022817	13,00998	0,10%
2,246148	16,0222	16,03289	0,07%
2,595606	19,021533	19,03572	0,07%
2,944273	22,020895	22,03176	0,05%
3,292456	25,020288	25,02363	0,01%
3,643448	28,019517	28,03965	0,07%
3,991094	31,019485	31,02691	0,02%
4,451846	35,019326	34,98607	0,09%
0,615935	2,024738	2,024751	0,00%
3,989887	31,019884	31,01654	0,01%
0,966885	5,024217	5,040398	0,03%
3,641312	28,020147	28,02129	0,00%
1,429353	9,023472	9,014306	0,10%

Elaboração: Autor. Fonte: POLO - UFSC.

5.5 Bateria

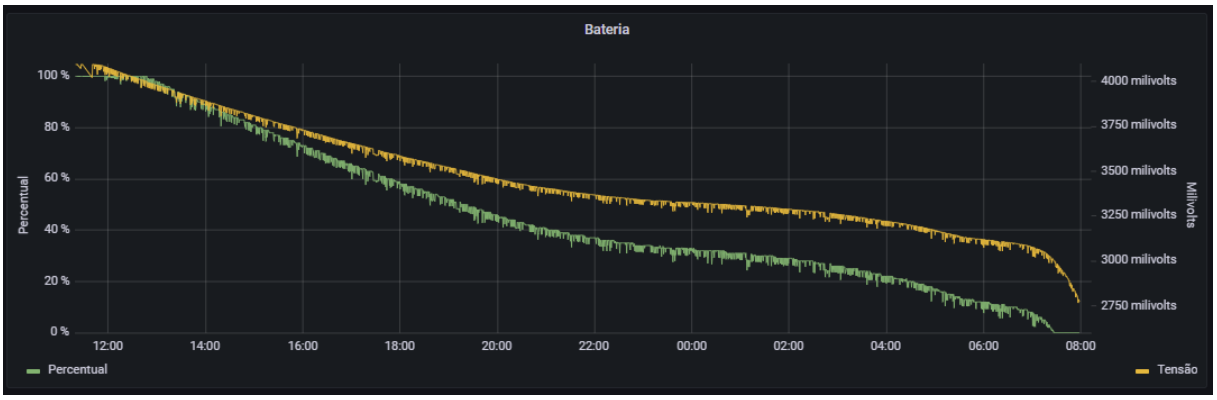
Os testes com a bateria foram realizados com o sistema embarcado conectado na rede WiFi e enviando informações a cada 3 segundos, com todos os sensores sendo utilizados. O cálculo do percentual é feito com a tensão medida, estabelecido como padrão a tensão acima de 4000 milivolts para 100% e abaixo de 3000 milivolts para 0%. Foi usado uma célula de lítio padrão 18650 e teóricos 2500mAh de capacidade, extraída de um *power bank* genérico comprado na internet. Não existe informação nenhuma na célula além de sua capacidade, não sendo possível consultar seu *datasheet* para maiores detalhes. Na Figura 33 está o gráfico dos testes, e conforme verificado, o sistema embarcado funcionou por aproximadamente 19 horas, atendendo ao requisito do projeto.

Figura 32 – Teste wattímetro e dispositivo usado para comparação.



Fonte: autoria própria.

Figura 33 – Tensão medida e percentual da bateria.



Fonte: autoria própria.

6 CONCLUSÕES

Foram dois os motivadores para a realização desse trabalho: criar um sistema embarcado para monitoração de sensores em procedimentos de instalação e manutenção de sistemas de refrigeração, conforme descrito na seção 1.1, aperfeiçoando o sistema desenvolvido na bolsa de iniciação científica para um protótipo viável, e a utilização de um sistema operacional em tempo real em um sistema embarcado - seção 2.4 -, com objetivo de aprendizado. Esses objetivos foram realizados com a utilização de um microcontrolador ESP32, o sistema operacional de tempo real FreeRTOS e recursos de comunicação sem fio.

Entre os objetivos não atingidos, estão o aplicativo para celular e sua interface de comunicação *Bluetooth* com a ESP32. Seu desenvolvimento é sugerido como um trabalho futuro para o projeto.

Com foco no software embarcado, o trabalho teve como base o FreeRTOS e seus recursos, como Tasks e Semáforos, amplamente utilizados. O desenvolvimento do software embarcado teve como base a placa desenvolvida na seção 4.1. Embora seja simples e visualmente não polida, ela permitiu que o desenvolvimento do software ocorresse sem maiores problemas e também possibilitou testes do código e dos sensores. E sobre o código desenvolvido, a utilização do FreeRTOS se mostrou poderosa e robusta, permitindo o uso de recursos como comunicação sem fio, interrupções, leitura de sensores e hospedagem de serviços *web* (*wifi manager*) fossem processados com rapidez e em ciclos de tempo regulares, causando a impressão de estarem de fato sendo executados paralelamente. Aliado ao ESP32, o FreeRTOS é uma ferramenta eficaz, permitindo que os principais requisitos fossem atingidos. Comparado com o uso tradicional do *loop* sequencial na plataforma IDE Arduino, as vantagens em usar um sistema operacional em tempo real destacadas anteriormente se mostraram evidentes. Gerenciar a execução de múltiplas funções em intervalos e tempos regulares, principalmente envolvendo recursos de comunicação via rede, somente em um *loop*, não seria viável, com atrasos em execução de determinadas funções. Por exemplo, o envio e recebimento de dados via conexão WiFi em intervalos de um segundo estaria comprometido se tivesse que esperar a execução das funções de leitura dos sensores, que, conforme medido, levam mais de 1,1 segundos para serem executadas.

As ferramentas utilizadas na nuvem, como o Node-RED, Grafana, InfluxDB, MQTT permitiram, junto com o JSON, o desenvolvimento de uma interface gráfica e de recursos para monitoração do comportamento do sistema.

Entre os requisitos funcionais (Tabela 7), somente o RF10 não foi atendido. Já para os requisitos não funcionais (Tabela 8), o requisito RNF04 não foi atendido, e os requisitos RNF08 e RNF17 foram atendidos parcialmente.

Todo o código desenvolvido para este projeto, junto com as imagens, tabelas e documentação estão disponíveis no *Github* do autor[1], sob licença MIT[2]. Todos os demais códigos e bibliotecas proprietárias utilizadas estão sob suas respectivas licenças.

6.1 Trabalhos futuros

Como sugestões de melhorias e trabalhos futuros para o sistema desenvolvido, as propostas são:

- Sincronizar os dados lidos e enviados por *Bluetooth* para o celular com a nuvem, utilizando as conexões WiFi ou 4G do próprio celular;
- Adicionar recursos de armazenamento das leituras em memória interna, como por exemplo, um cartão SD;
- Explorar mais a biblioteca do DS18B20 para otimizações nos tempos de processamento;
- Muitas operações estão sendo feitas dentro do *for* das Tasks, ao invés de chamar uma função específica e poluir menos o código dentro da Task;
- Utilizar mais a divisão do código em arquivos, evitando códigos longos em um único arquivo;
- Implementar recursos de debug na IDE de desenvolvimento via WiFi;
- Evitar o uso do *Serial.print* para verificação do comportamento do código;
- Caso haja perda de conexão, continuar lendo os sensores, armazenando as leituras em um *buffer*, utilizando os recursos oferecidos pelo FreeRTOS para evitar o uso de variáveis globais e problemas como do produtor e consumidor (*Bounded Buffer Problem*);
- Refazer a placa de circuitos, evitando trilhas com angulações elevadas;
- Implementar o serviço na nuvem com capacidade multiusuário. No sistema desenvolvido, com Node-RED e Grafana, para cada usuário é necessário ter uma instância específica no servidor para essas aplicações. É possível contornar isso usando *Docker*, porém o resultado seria um uso demasiado elevado de recursos de hardware;
- Desenvolver uma placa com leds indicadores de status de recursos como WiFi e Bluetooth;

[1] https://github.com/FabianoKraemer/TCC2
[2] pt.wikipedia.org/wiki/Licença_ MIT

- Desenvolver uma placa de PCB com componentes SMD[3], e não somente PTH[4];
- Implementar recursos de segurança e criptografia nas comunicações sem fio;
- Explorar a possibilidade de adicionar outras interfaces de comunicação sem fio, como LoRa[5] e Zigbee[6].

[3] https://pt.wikipedia.org/wiki/Tecnologia_de_montagem_superficial
[4] https://pt.wikipedia.org/wiki/Montagem_through-hole
[5] https://lora-alliance.org/
[6] https://pt.wikipedia.org/wiki/Zigbee

REFERÊNCIAS

ABRAVA. *Faturamento do setor de refrigeração e ar condicionado cai 4% em 2020*. 2020. Disponível em: <https://blogdofrio.com.br/faturamento-do-setor-de-refrigeracao-e-ar-condicionado-cai-4-em-2020>. Acesso em: 16 ago. 2021. Citado na página 17.

ABRAVA. Boletim econômico 1o trimestre de 2021. 2021. Disponível em: <https://abrava.com.br/wp-content/uploads/2021/05/boletim-Economico-maio-21.pdf>. Acesso em: 10 ago. 2021. Citado na página 18.

ALI, Z. *Introduction to DS18B20*. 2019. Disponível em: <https://www.theengineeringprojects.com/2019/01/introduction-to-ds18b20.html>. Acesso em: 21 ago. 2021. Citado na página 25.

BALBINOT, A.; BRUSAMARELLO, V. Instrumentação e fundamentos de medidas–volume 2, 2ª edição. *Rio de Janeiro: LTC/Grupo Gen*, 2011. Citado na página 22.

BANGGOOD.COM. *Transdutor de Pressão*. 2021. Disponível em: <https://banggood.onelink.me/zMT7/lqxugw65>. Acesso em: 21 ago. 2021. Citado na página 27.

BARROS, E.; CAVALCANTE, S. Introdução aos sistemas embarcados. *Artigo apresentado na Universidade Federal de Pernambuco-UFPE*, p. 36, 2010. Citado na página 30.

CARRION, P.; QUARESMA, M. Internet da coisas (iot): Definições e aplicabilidade aos usuários finais. p. 52–53, 2019. Disponível em: <https://periodicos.udesc.br/index.php/hfd/article/view/2316796308152019049/9858>. Acesso em: 20 ago. 2021. Citado na página 21.

CóRDOBA, U. *Internet de las Cosas*. 2018. Disponível em: <https://www.academia.frc.utn.edu.ar/?pIs=3369>. Acesso em: 20 ago. 2021. Citado na página 21.

DENARDIN, G.; BARRIQUELLO, C. *Sistemas operacionais de tempo real e sua aplicação em sistemas embarcados*. Blucher, 2019. ISBN 9788521213970. Disponível em: <https://books.google.com.br/books?id=FrqxDwAAQBAJ>. Citado 3 vezes nas páginas 36, 37 e 54.

ELETROFRIGOR. *Ferramentas para refrigeração: a lista definitiva*. 2019. Disponível em: <https://www.eletrofrigor.com.br/ferramentas-para-refrigeracao-a-lista-definitiva>. Acesso em: 14 ago. 2021. Citado na página 18.

ENERGéTICA, E. de P. Uso de ar condicionado no setor residencial brasileiro: Perspectivas e contribuições para o avanço em eficiência energética. 2018. Disponível em: <https://www.epe.gov.br/sites-pt/publicacoes-dados-abertos/publicacoes/PublicacoesArquivos/publicacao-341/NT%20EPE%20030_2018_18Dez2018.pdf>. Acesso em: 10 ago. 2021. Citado na página 18.

ESPRESSIF. *Datasheet ESP32*. 2018. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>. Acesso em: 02 set. 2021. Citado na página 35.

GEEKSFORGEEKS.ORG. Bouncing effect. geeksforgeeks.org, 2016. Disponível em: <https://media.geeksforgeeks.org/wp-content/uploads/20191113173218/Switch_Debounce_2.jpg>. Citado na página 58.

GURU, I. *Datasheet PZEM004t*. 2020. Disponível em: <https://innovatorsguru.com/wp-content/uploads/2019/06/PZEM-004T-V3.0-Datasheet-User-Manual.pdf>. Acesso em: 21 ago. 2021. Citado na página 29.

HOU, L. et al. Internet of things cloud: Architecture and implementation. *IEEE Communications Magazine*, IEEE, v. 54, n. 12, p. 32–39, 2016. Citado na página 22.

IBM. The little-known story of the first iot device. 2018. Disponível em: <https://www.ibm.com/blogs/industries/little-known-story-first-iot-device/>. Acesso em: 20 ago. 2021. Citado na página 21.

INSTRUSUL. Como funciona o wattímetro. 2018. Disponível em: <https://blog.instrusul.com.br/como-funciona-o-wattimetro/>. Acesso em: 20 ago. 2021. Citado na página 28.

MOREIRA, I. *Vendas de assistentes virtuais crescem 50% no ano: Alexa é modelo mais desejado*. 2020. Disponível em: <https://www.metroworldnews.com.br/home/2020/07/06/vendas-de-assistentes-virtuais-crescem-50-no-ano-alexa-e-modelo-mais-desejado.html>. Acesso em: 07 jul. 2022. Citado na página 17.

NAQVI, S. N. Z.; YFANTIDOU, S.; ZIMÁNYI, E. Time series databases and influxdb. *Studienarbeit, Université Libre de Bruxelles*, v. 12, 2017. Citado na página 42.

SEMICONDUCTOR, D. *Datasheet DS18B20*. 2018. Disponível em: <https://pdf1.alldatasheet.com/datasheet-pdf/view/58557/DALLAS/DS18B20.html>. Acesso em: 21 ago. 2021. Citado 2 vezes nas páginas 24 e 25.

SHUTTERSTOCK. *Blog do frio*. 2020. Disponível em: <https://blogdofrio.com.br/governo-inclui-refrigeracao-no-rol-de-atividades-essenciais/>. Acesso em: 17 ago. 2021. Citado na página 19.

SILVEIRA, C. B. Como funciona o transdutor de pressão. *IEEE Communications Magazine*, 2018. Disponível em: <https://www.citisystems.com.br/transdutor-de-pressao/#:~:text=Um%20transdutor%20de%20press%C3%A3o%20%C3%A9,el%C3%A9trica%20que%20detecta%20esta%20deforma%C3%A7%C3%A3o.> Acesso em: 21 ago. 2021. Citado 2 vezes nas páginas 25 e 26.

SJ, I. *Análise experimental dos parâmetros de controle e operacionais de uma máquina de fabricar gelo movida a energia solar*. 2021. Disponível em: <https://www.youtube.com/watch?v=ZpB8BcRS8iI&t=1952s>. Acesso em: 17 ago. 2021. Citado na página 19.

SPEC, G. *Temperature Sensors Information*. 2015. Disponível em: <https://www.globalspec.com/learnmore/sensors_transducers_detectors/temperature_sensing/temperature_sensors>. Acesso em: 20 ago. 2021. Citado na página 23.

STANKOVIC, J. A.; RAJKUMAR, R. Real-time operating systems. *Real-Time Systems*, Citeseer, v. 28, n. 2-3, p. 237–253, 2004. Citado na página 37.

VINICIUS, S. *Análise de séries temporais*. 2018. Disponível em: <https://www.monolitonimbus.com.br/analise-de-series-temporais/>. Acesso em: 02 set. 2021. Citado na página 41.

VITAL, A. *Kit Refrigeração Bomba De Vacuo*. 2021. Disponível em: <https://produto.mercadolivre.com.br/MLB-1411667194-kit-refrigeraco-bomba-de-vacuo-cliente-vitalalexandre2018-_JM?matt_tool=97626976&matt_word>. Acesso em: 16 ago. 2021. Citado na página 18.

YOUNG, S. The risk/reward realities of chip development. *Embedded.com*, 2002. Disponível em: <https://www.embedded.com/the-risk-reward-realities-of-chip-development/#:~:text=At%20launch%2C%20a%20major%20IC,K%2Fyear%20per%20staff%20member.> Acesso em: 02 set. 2021. Citado na página 30.

YUAN, M. Conhecendo o mqtt. IBM, 2017. Disponível em: <https://developer.ibm.com/br/articles/iot-mqtt-why-good-for-iot/>. Citado na página 40.