UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA

KEVIN VINICIUS FRANCO

MEDIDOR DE CONSUMO DE ENERGIA ELÉTRICA UTILIZANDO O MICROCONTROLADOR ESP

Uberlândia
2025

KEVIN VINICIUS FRANCO

MEDIDOR DE CONSUMO DE ENERGIA ELÉTRICA UTILIZANDO O MICROCONTROLADOR ESP

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia, como requisito parcial para a obtenção do título de bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Augusto Wohlgemuth Fleury Veloso da Silveira

Uberlândia
2025

KEVIN VINICIUS FRANCO

MEDIDOR DE CONSUMO DE ENERGIA ELÉTRICA UTILIZANDO O MICROCONTROLADOR ESP

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia, como requisito parcial para a obtenção do título de bacharel em Engenharia Elétrica.

Uberlândia, 23 de abril de 2025

BANCA EXAMINADORA

_____

Prof. Dr. Augusto Wohlgemuth Fleury Veloso da Silveira

Universidade Federal de Uberlândia

_____

Prof. Dr. Carlos Eduardo Tavares

Universidade Federal de Uberlândia

_____

Prof. Dr. Luciano Coutinho Gomes

Universidade Federal de Uberlândia

AGRADECIMENTOS

Agradeço, em primeiro lugar, aos meus pais, pelo apoio incondicional e incentivo constante, que me fortaleceram em cada etapa desta jornada. Expresso também minha gratidão aos professores, cujas orientações e ensinamentos foram fundamentais para o meu desenvolvimento acadêmico e pessoal. Aos laboratórios e ao Programa de Educação Tutorial, agradeço pelas experiências enriquecedoras e pelas oportunidades de crescimento profissional. A todos que, direta ou indiretamente, contribuíram para esta conquista, deixo meu mais sincero e profundo agradecimento.

RESUMO

A eficiência energética é um aspecto crucial no contexto atual de escassez de recursos e preocupações ambientais, pois busca otimizar o consumo de energia, reduzindo desperdícios e promovendo o uso sustentável. Neste cenário, o sistema de monitoramento de consumo de energia elétrica, desenvolvido com o microcontrolador ESP32 e o medidor PZEM-004T da Peacefair, surge como uma solução viável para realizar as medições necessárias. O principal objetivo é fornecer uma ferramenta eficiente para medir e analisar dados energéticos em tempo real, utilizando a comunicação Bluetooth Low Energy. Isso permite a conexão entre o ESP32 e o aplicativo móvel desenvolvido na plataforma Thunkable, proporcionando a visualização dos dados em dispositivos Android e iOS. A metodologia envolveu a integração do PZEM-004T com o ESP32 por meio da comunicação UART. Os testes realizados com uma lâmpada de LED de 8W foram validados por meio da comparação com o medidor Atorch S1. Os resultados demonstraram que o sistema é funcional e oferece uma alternativa de baixo custo e fácil implementação para o monitoramento de consumo energético em residências e empresas. Essa abordagem favorece a automação e o controle eficiente de energia, destacando-se como uma solução prática para a gestão do consumo energético. Além disso, promove a conscientização sobre a importância da eficiência energética.

Palavras-chave: Eficiência energética; Monitoramento; ESP32; PZEM-004T; Bluetooth Low Energy.

ABSTRACT

Energy efficiency is a crucial aspect in the current context of resource scarcity and environmental concerns, aiming to optimize energy consumption by reducing waste and promoting sustainable use. In this scenario, the electrical energy consumption monitoring system developed with the ESP32 microcontroller and the Peacefair PZEM-004T meter emerges as a viable solution for necessary measurements. The main goal is to provide an efficient tool for real-time energy data measurement and analysis using Bluetooth Low Energy communication. This enables the connection between the ESP32 and the mobile application developed on the Thunkable platform, allowing data visualization on Android and iOS devices. The methodology involved integrating the PZEM-004T with the ESP32 via UART communication. Tests conducted with an 8W LED bulb were validated by comparison with the Atorch S1 meter. Results demonstrated that the system is functional and offers a low-cost, easy-to-implement alternative for monitoring energy consumption in homes and businesses. This approach supports the automation and efficient control of energy, standing out as a practical solution for energy consumption management. Additionally, it promotes awareness of the importance of energy efficiency.

Keywords: Energy efficiency; Monitoring; ESP32; PZEM-004T; Bluetooth Low Energy.

LISTA DE ILUSTRAÇÕES

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ADC	Analog-to-Digital Converter
ANEEL	Agência Nacional de Energia Elétrica
BLE	Bluetooth Low Energy
DAC	Digital-to-Analog Converter
DC	Direct Current
GND	Ground
GPIO	General Purpose Input/Output
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
INMET	Instituto Nacional de Meteorologia
IoT	Internet of Things
IRENA	International Renewable Energy Agency
LED	Light Emitting Diode
MCU	Microcontroller Unit
ONS	Operador Nacional do Sistema Elétrico
PEE	Programa de Eficiência Energética
PROCEL	Programa Nacional de Conservação de Energia Elétrica
PWM	Pulse Width Modulation
RX	Receive
SPI	Serial Peripheral Interface
TC	Transformador de Corrente
TTL	Transistor-Transistor Logic
TX	Transmit
UART	Universal Asynchronous Receiver/Transmitter
URL	Uniform Resource Locator
USB	Universal Serial Bus
UUID	Universally Unique Identifier
VCC	Voltage Common Collector
VIN	Voltage In

Wi-Fi	Wireless Fidelity

SUMÁRIO

1 **INTRODUÇÃO**

A eficiência energética é um tema crucial, especialmente diante da escassez de recursos e das crescentes preocupações ambientais. A busca por alternativas que otimizem o consumo de energia tornou-se uma necessidade urgente em diversas esferas, incluindo residências, empresas e indústrias. Segundo Souza et al. (2019), essa eficiência envolve a adoção de tecnologias e práticas que reduzam o desperdício, promovendo um uso mais racional dos recursos disponíveis.

Nesse contexto, a automação e o controle eficiente de energia ganham relevância em um mundo onde a sustentabilidade é prioridade. A crescente demanda por energia, aliada à necessidade de mitigar impactos ambientais, torna indispensável o uso de ferramentas que permitam a gestão eficaz do consumo energético. No Brasil, o Programa de Eficiência Energética (PEE), coordenado pela Agência Nacional de Energia Elétrica (ANEEL), é um exemplo significativo de como políticas públicas incentivam a adoção de tecnologias voltadas para o uso racional de energia. A conscientização promovida por programas como o PEE é essencial para estimular mudanças de comportamento entre os consumidores e fomentar práticas sustentáveis (Brasil, 2022).

Diante dessa necessidade, o desenvolvimento de sistemas para monitorar o consumo de energia elétrica de forma eficiente torna-se fundamental. A integração de microcontroladores e dispositivos de medição representa uma abordagem promissora para facilitar o acompanhamento do uso energético. O monitoramento em tempo real permite que os usuários tenham uma visão clara de seus hábitos de consumo, possibilitando a adoção de medidas corretivas quando necessário (Ashokkumar et al., 2023).

Este trabalho tem como objetivo desenvolver um sistema de monitoramento de consumo de energia elétrica utilizando o microcontrolador ESP32 e o medidor PZEM-004T da Peacefair. A proposta é criar uma solução viável para medições precisas, oferecendo uma ferramenta eficaz para medir e analisar dados energéticos em tempo real. Além disso, será desenvolvido um aplicativo móvel na plataforma Thunkable, que permitirá a comunicação com o ESP32 por meio do protocolo Bluetooth Low Energy (BLE), oferecendo uma interface intuitiva compatível com dispositivos Android e iOS.

A integração entre o medidor PZEM-004T e o ESP32 será realizada por meio da interface UART, um protocolo de comunicação serial amplamente utilizado para transmitir dados de forma eficiente e simplificada entre dispositivos eletrônicos. Essa abordagem garante uma conexão confiável e facilita a implementação do sistema, atendendo à demanda por soluções acessíveis e de fácil instalação, aplicáveis em diversos contextos, como residências, empresas e indústrias.

Durante os testes do projeto, será utilizada uma lâmpada LED de 8W como carga. As medições obtidas serão comparadas com as do medidor comercial Atorch S1, com o objetivo de validar a precisão dos dados coletados. Essa etapa é essencial para garantir a confiabilidade do sistema e reforçar sua aplicabilidade no monitoramento de consumo energético em diferentes cenários. A proposta busca oferecer uma solução prática, de baixo custo e fácil implementação, alinhada às necessidades atuais de eficiência energética e controle sustentável do consumo.

Os resultados deste trabalho não apenas visam validar a funcionalidade do medidor, mas também contribuir para a disseminação de práticas sustentáveis e a redução do desperdício de energia. A análise dos dados coletados e a comparação com medições de dispositivos comerciais permitirão uma avaliação rigorosa da eficácia do sistema. Assim, busca-se demonstrar que a implementação de soluções acessíveis e eficientes pode transformar hábitos de consumo, proporcionando benefícios econômicos e um impacto positivo no meio ambiente. Este projeto insere-se em um esforço mais amplo de inovação e sustentabilidade, contribuindo para um futuro em que a eficiência energética seja parte fundamental do cotidiano.

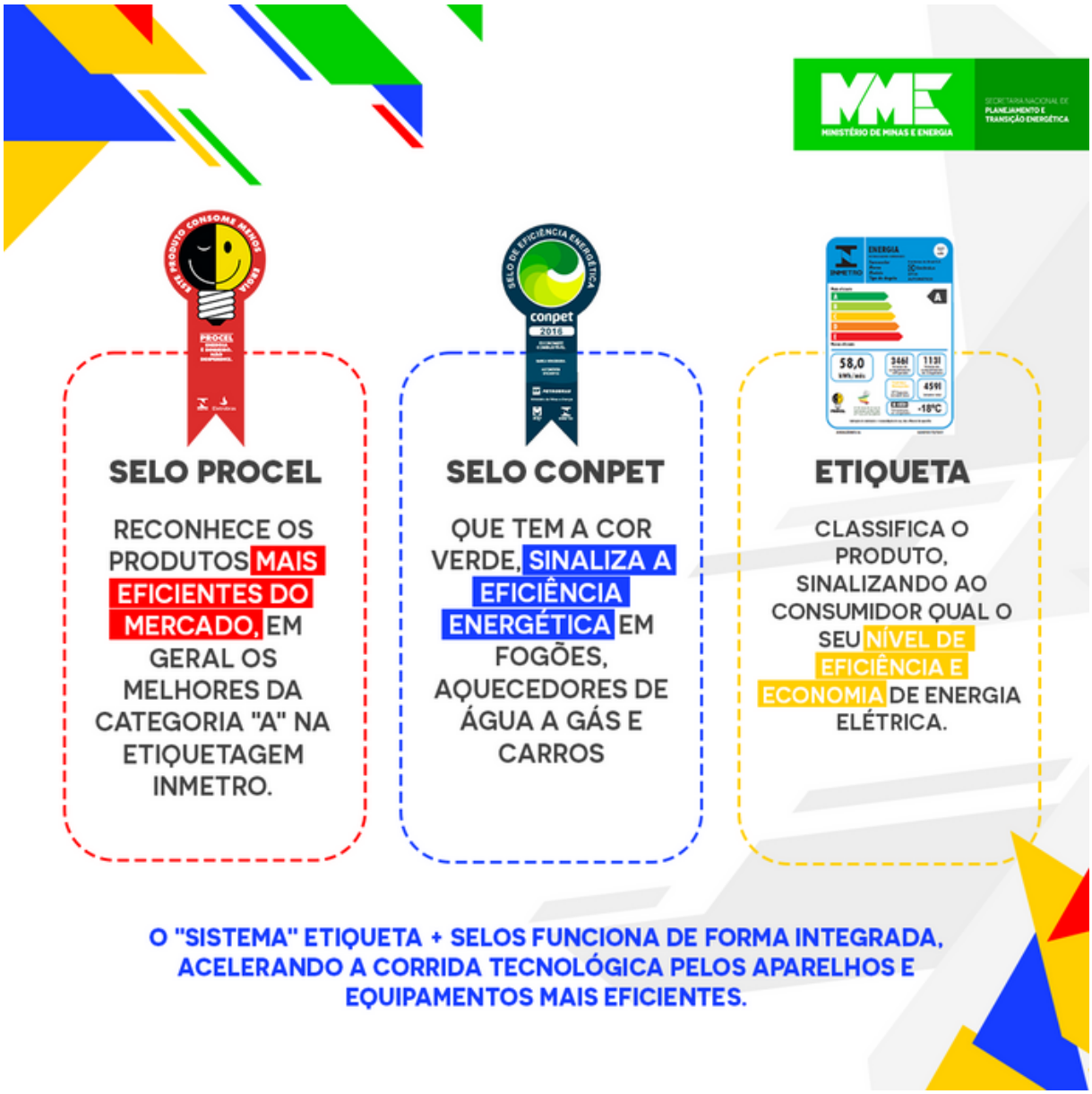
2 **EFICIÊNCIA ENERGÉTICA**

A eficiência energética tem se tornado um pilar fundamental na busca por um consumo de energia mais consciente e sustentável no Brasil. Conforme definido pela ANEEL, a eficiência energética consiste em utilizar menos energia para executar a mesma função, evitando desperdícios e otimizando os recursos disponíveis. Com a crescente demanda por eletricidade, aliada a recursos limitados, torna-se imperativo adotar políticas e tecnologias que reduzam o consumo energético e promovam o desenvolvimento sustentável do país (Brasil, 2022).

Um dos principais marcos regulatórios em vigor é o Programa de Eficiência Energética (PEE), criado pela ANEEL em 2000. O programa determina que as concessionárias de distribuição de energia invistam parte de sua receita operacional líquida em projetos que incentivem a redução do consumo. Tais projetos promovem a adoção de tecnologias mais eficientes e processos otimizados, além de fomentar a educação dos consumidores sobre a importância de práticas conscientes no uso da eletricidade. Dentre as principais ações do PEE, destacam-se o incentivo à adoção de eletrodomésticos mais eficientes e a implementação de soluções de automação e controle energético, especialmente nos setores industrial e comercial (Brasil, [2022?]).

Um exemplo bem-sucedido desse incentivo é o "Selo PROCEL de Economia de Energia" (Figura 1), que identifica produtos com alto desempenho energético, permitindo aos consumidores optarem por equipamentos que ajudam a reduzir o consumo em residências e empresas. As medidas de eficiência energética tornam-se ainda mais relevantes em cenários de crise hídrica, em que a geração de energia a partir de hidrelétricas pode ser comprometida (Brasil, 2023).

Figura 1 — Infográfico dos selos.



Fonte: Brasil (2023).

Além das iniciativas regulatórias, as inovações tecnológicas desempenham um papel crucial na promoção da eficiência energética. O desenvolvimento de dispositivos inteligentes, como medidores de energia conectados e sistemas de automação residencial, facilita o monitoramento e o controle do consumo. Esses dispositivos permitem que os usuários identifiquem padrões de uso e ajustem suas rotinas, resultando em economias significativas. A Internet das Coisas (IoT) se destaca como uma tendência importante, proporcionando soluções que otimizam o

uso de energia em tempo real, como a programação automática de horários para ligar e desligar equipamentos.

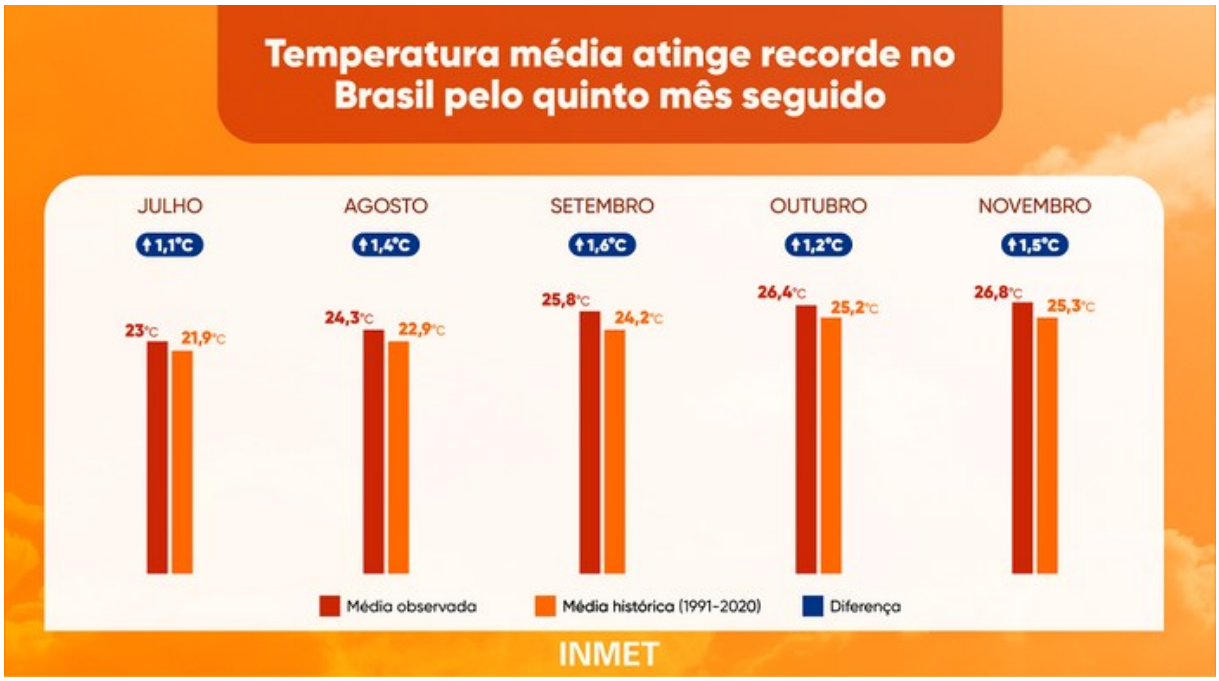
Os benefícios da eficiência energética vão além da redução de custos com eletricidade: a implementação de práticas e tecnologias eficientes pode contribuir para a criação de empregos verdes, impulsionando o crescimento econômico sustentável. Segundo a IRENA, o investimento em eficiência energética pode gerar milhões de empregos até 2050, especialmente nos setores da construção civil, energias renováveis e serviços de manutenção. Portanto, a eficiência energética não apenas contribui para a preservação dos recursos naturais, mas também representa uma oportunidade para o desenvolvimento econômico e a melhoria da qualidade de vida da população (IRENA, 2023).

2.1 CRISE HÍDRICA, ONDAS DE CALOR E O IMPACTO NO CONSUMO ENERGÉTICO

A escassez de chuvas em importantes regiões do Brasil vem agravando a crise hídrica, afetando diretamente a geração de energia hidrelétrica, que compõe cerca de 48% da matriz elétrica nacional (ONS, 2024). Essa situação tem levado o país a aumentar o uso de termelétricas, que são mais caras e poluentes. A crise hídrica, portanto, torna ainda mais urgente a adoção de medidas de eficiência energética e o desenvolvimento de tecnologias que auxiliem no monitoramento e na gestão do consumo de eletricidade.

Paralelamente, o Brasil tem enfrentado ondas de calor cada vez mais intensas, registrando temperaturas recordes em 2023, conforme mostra a Figura 2, com base em dados do INMET (2024). Esses aumentos substanciais de temperatura impulsionam o uso de aparelhos de resfriamento, como ventiladores, ar-condicionado e climatizadores.

Figura 2 — Temperatura média no Brasil de julho a novembro em 2023.



Fonte: INMET (2023).

Esses equipamentos são essenciais para o conforto térmico da população, mas contribuem significativamente para a sobrecarga do sistema elétrico, elevando o consumo de energia. Nesse contexto, tecnologias de monitoramento do consumo e ações de conscientização sobre o uso racional da eletricidade tornam-se ferramentas indispensáveis para mitigar esses impactos.

2.2 TECNOLOGIAS PARA MONITORAMENTO E CONTROLE DE ENERGIA

Diante do cenário de crise hídrica e do aumento do consumo energético causado por ondas de calor, a necessidade de soluções tecnológicas que possibilitem o uso mais inteligente da eletricidade torna-se ainda mais evidente. O desenvolvimento de ferramentas de monitoramento do consumo, que permitem aos consumidores acompanhar e ajustar seus hábitos em tempo real, tem sido fundamental para a promoção da eficiência energética.

Nesse contexto, o uso de microcontroladores se destaca como uma solução eficiente e acessível para a implementação de sistemas de automação residencial e

industrial, possibilitando o controle preciso do consumo de energia (Saleem, 2021). Entre os principais dispositivos utilizados para esse fim, estão os microcontroladores da Espressif, como o ESP8266 e o ESP32, amplamente reconhecidos por sua versatilidade, baixo custo e conectividade.

3 **EQUIPAMENTOS E TECNOLOGIAS**

3.1 MICROCONTROLADORES ESPRESSIF

A Espressif Systems é uma empresa chinesa fundada em 2008, amplamente reconhecida por desenvolver soluções inovadoras de conectividade para dispositivos IoT. Entre seus principais produtos, estão os microcontroladores ESP8266 e ESP32, que revolucionaram o mercado devido ao seu baixo custo, consumo eficiente de energia e suporte a múltiplos protocolos de comunicação. Esses microcontroladores se tornaram especialmente populares em projetos de automação residencial, industrial e de monitoramento, por sua capacidade de integrar sensores e atuadores com redes sem fio (Espressif, 2024).

3.1.1 **ESP8266**

O ESP8266 é um microcontrolador que rapidamente se tornou um marco no desenvolvimento de dispositivos conectados, por sua capacidade de prover conectividade Wi-Fi a um custo muito acessível. Lançado pela Espressif em 2014, possui um processador Tensilica L106 de 32 bits, que opera a uma frequência de até 160 MHz, com suporte à comunicação por Wi-Fi IEEE 802.11 b/g/n, permitindo a conexão direta a redes sem fio. Sua popularidade deve-se à combinação de baixo consumo de energia, baixo custo e facilidade de integração em projetos de hardware e software (Espressif, 2024).

Além do processador integrado, o ESP8266 oferece uma variedade de pinos GPIO, o que facilita a comunicação com sensores e outros componentes eletrônicos. A placa inclui ainda interfaces como SPI , I2C e UART, o que amplia as possibilidades de integração com diferentes dispositivos (Espressif, 2024).

Uma das principais características que tornou o ESP8266 amplamente utilizado foi a existência de bibliotecas e frameworks de fácil uso, como o *NodeMCU*, que oferece uma interface de programação em Lua e facilita o desenvolvimento de projetos IoT. Outro ponto relevante é o suporte da comunidade de desenvolvedores, que mantém uma vasta documentação, tutoriais e bibliotecas, o que simplifica sua

adoção em projetos de automação residencial, monitoramento e controle de dispositivos conectados.

O microcontrolador ESP8266 também se destaca por seu consumo eficiente de energia, especialmente nos modos de economia que permitem que o dispositivo entre em modo de suspensão profunda (*deep sleep*), utilizando apenas alguns microamperes de corrente, o que é ideal para dispositivos que precisam operar com baterias por longos períodos (Espressif, 2024).

3.1.2 **ESP32**

O ESP32 é uma evolução significativa do popular microcontrolador ESP8266. O ESP32 foi projetado para atender às necessidades de projetos que exigem conectividade robusta e múltiplos recursos integrados. Equipado com um processador *dual-core* Xtensa LX6 de 32 bits, que opera a uma frequência de até 240 MHz, o ESP32 proporciona maior poder de processamento e eficiência, sendo ideal para aplicações que exigem multitarefa ou alta capacidade de resposta (Espressif, 2024).

Um dos grandes destaques do ESP32 é sua capacidade de conectividade dupla, suportando tanto Wi-Fi quanto *Bluetooth*, incluindo *Bluetooth Low Energy* (BLE). Essa característica faz com que o ESP32 seja altamente versátil em projetos que requerem conectividade sem fio para a comunicação com dispositivos móveis, hubs de automação ou redes locais (Espressif, 2024).

Além da conectividade, o ESP32 conta com uma série de pinos GPIO e suporte para interfaces comuns como I2C, SPI, UART, PWM, e ADC/DAC, o que o torna adequado para conectar sensores, atuadores e outros dispositivos eletrônicos (Espressif, 2024).

Outro aspecto importante do ESP32 é seu foco na eficiência energética. Ele é projetado com modos de economia de energia, como *light sleep* e *deep sleep*, que permitem ao microcontrolador reduzir significativamente o consumo quando está em estado de inatividade. Esse recurso é particularmente útil em dispositivos que dependem de baterias ou precisam operar por longos períodos com consumo

mínimo de energia, como em sistemas de monitoramento de consumo energético (Espressif, 2024).

A Espressif também disponibiliza uma ampla documentação e bibliotecas de código que facilitam o desenvolvimento com o ESP32, tornando-o uma das escolhas preferidas para projetos IoT, automação residencial e sistemas de monitoramento. Sua capacidade de processamento aliada à conectividade sem fio e à eficiência energética o tornam uma solução completa para diversos tipos de aplicações (Espressif, 2024).

3.1.3 *NodeMCU* ESP32

A placa *NodeMCU* ESP32 (Figura 3) integra o microcontrolador ESP32, oferecendo uma solução prática para prototipagem e desenvolvimento de sistemas embarcados. Com suporte para Wi-Fi e *Bluetooth*, a placa é frequentemente utilizada em projetos de IoT e automação residencial. Seu design compacto inclui uma interface USB para programação e um regulador de tensão, que permite a alimentação por USB ou pelo pino VIN (5V) (Oliveira, 2017).

Figura 3 — NodeMCU ESP32.

Fonte: Oliveira (2017).

Além disso, oferece 32 pinos GPIO e compatibilidade com diversas interfaces de comunicação, o que facilita a integração com sensores e outros dispositivos eletrônicos. Neste trabalho, o módulo será responsável por receber dados do PZEM-004T por meio da interface UART, processá-los e transmiti-los via BLE para o aplicativo móvel. A escolha da *NodeMCU* ESP32 se justifica pela facilidade de uso e suporte da comunidade de desenvolvedores, além de sua capacidade de operar em modos de economia de energia — uma característica essencial para dispositivos focados em eficiência energética (Oliveira, 2017).

3.2 PEACEFAIR PZEM-004T

A Peacefair é uma empresa especializada na produção de instrumentos de medição elétrica, voltados principalmente para aplicações residenciais e comerciais. Seus produtos são amplamente utilizados em sistemas de monitoramento de consumo de energia, destacando-se pela precisão, durabilidade e fácil integração com microcontroladores. A empresa oferece uma linha de medidores de energia que atendem a diversas necessidades, variando de pequenos sistemas domésticos a soluções mais robustas para ambientes empresariais (Peacefair, 2024).

Entre os produtos mais usados está o PZEM-004T (Figura 4), um módulo compacto que mede informações como tensão, corrente, potência, e energia consumida ao longo do tempo. Ele também monitora frequência e fator de potência, que indicam a estabilidade e eficiência do consumo. Sua simplicidade na instalação e integração com microcontroladores torna esse módulo uma escolha popular para projetos de automação residencial e monitoramento energético.

Figura 4 — Módulo PZEM-004T.

Fonte: Peacefair (2024).

Projetado para fornecer medições precisas e estáveis, o PZEM-004T é capaz de medir correntes de até 100A com o uso de um transformador de corrente dedicado. Essa capacidade o torna ideal para monitorar o consumo de aparelhos com alta demanda energética, como equipamentos de refrigeração, motores elétricos e sistemas de aquecimento (Peacefair, 2024). No projeto deste trabalho, o transformador de corrente de 100A foi incorporado para garantir a leitura de correntes mais elevadas.

Por padrão, o PZEM-004T comunica-se com outros dispositivos através de uma interface UART. Esse tipo de comunicação é amplamente utilizado para conectar módulos eletrônicos a computadores e microcontroladores. No entanto, para que seja possível ler as medições diretamente em um computador, é necessário utilizar um conversor TTL para USB, que permite a conversão dos sinais de comunicação UART para um formato compatível com portas USB (Peacefair, 2024). Esse processo, no entanto, pode ser inconveniente e inviável em muitas aplicações práticas, já que exigiria o uso constante de um computador para monitorar os dados. Para projetos que visam maior praticidade, como o

monitoramento em tempo real por longos períodos ou em locais de difícil acesso, depender de um computador torna-se desvantajoso.

Para superar essa limitação, microcontroladores como o ESP32 ou ESP8266 podem ser integrados ao PZEM-004T. Eles leem os dados diretamente do medidor e, em vez de depender de um computador, transmitem essas informações por Wi-Fi ou *Bluetooth*. Isso permite que o usuário acesse as medições em tempo real por meio de *smartphones* ou *tablets*, tornando o sistema mais prático e eficiente. Com essa abordagem, é possível desenvolver soluções de monitoramento de energia de baixo custo e fácil instalação, eliminando a necessidade de infraestrutura complexa.

3.3 PROTOCOLO UART

A comunicação entre dispositivos eletrônicos pode ser realizada de várias maneiras, sendo o UART (*Universal Asynchronous Receiver-Transmitter*) uma das mais comuns em sistemas embarcados. O UART é um protocolo de comunicação serial assíncrono que permite a troca de dados entre dois dispositivos usando apenas dois fios: um para transmissão (TX) e outro para recepção (RX) (Fang; Chen, 2011).

Ao contrário de protocolos síncronos como SPI ou I2C, o UART não exige um sinal de relógio compartilhado entre os dispositivos para sincronizar a comunicação. Em vez disso, tanto o transmissor quanto o receptor precisam ser configurados com a mesma taxa de transmissão (*baud rate*), que define a velocidade com que os bits são enviados (Chen; Huang, 2023).

O UART funciona enviando os dados em série, bit por bit. A transmissão começa com um bit de início, que sinaliza o começo do envio. Em seguida, são transmitidos os bits de dados, geralmente de 5 a 8 bits por vez, dependendo da configuração. Por fim, um bit de parada é adicionado para indicar o fim da transmissão. Opcionalmente, pode ser incluído um bit de paridade, utilizado para verificar a integridade dos dados transmitidos, reduzindo erros (Chen; Huang, 2023).

Por ser um protocolo assíncrono, a sincronização entre transmissor e receptor é garantida apenas pela definição prévia do *baud rate*. Isso significa que os dispositivos não precisam coordenar constantemente suas operações via sinal de

relógio, como ocorre em protocolos síncronos, simplificando a implementação. Essa simplicidade e flexibilidade tornam o UART uma escolha comum para comunicação de curta distância entre microcontroladores e periféricos, como sensores e módulos de medição (Chen; Huang, 2023).

3.4 BLUETOOTH LOW ENERGY

O *Bluetooth Low Energy* é uma tecnologia de comunicação sem fio projetada para permitir a troca de dados entre dispositivos de forma eficiente em termos de consumo de energia. Ao contrário do *Bluetooth* clássico, que é utilizado para grandes transferências contínuas de dados, o BLE é otimizado para a troca de pequenas quantidades de informações, sendo ideal para dispositivos que exigem longos períodos de operação com baixo consumo de energia, como sensores, dispositivos de monitoramento e controles remotos (Tosi et al., 2017).

O BLE trabalha na frequência de 2,4 GHz, organizando a comunicação por pacotes. Esses pacotes são enviados por meio de 40 canais, sendo 37 para o envio de dados e 3 dedicados à fase inicial de descoberta, conhecida como publicidade. Quando um dispositivo quer se conectar a outro, ele envia informações iniciais nesses 3 canais específicos. Outros dispositivos próximos, como um *smartphone*, escaneiam esses canais para encontrar dispositivos disponíveis e iniciar a conexão (Tosi et al., 2017).

No BLE, cada dispositivo assume um dos dois papéis principais: servidor ou cliente. O servidor é o que oferece dados ou serviços, como um sensor que mede a temperatura do ambiente. Já o cliente é o dispositivo que acessa e utiliza esses dados, como um celular que exibe a temperatura medida. A comunicação entre os dispositivos é organizada em estruturas chamadas serviços, que agrupam várias informações específicas conhecidas como características. Por exemplo, um serviço de saúde pode conter uma característica que mede a frequência cardíaca e outra que indica o nível de oxigênio no sangue. Cada serviço e característica possui um UUID (Identificador Universal Único), garantindo que sejam identificados de maneira única no sistema BLE (Tosi et al., 2017).

Cada uma dessas características pode ser lida ou modificada pelo cliente, e o servidor pode também enviar atualizações sempre que houver uma nova medição, usando um sistema chamado notificação. Isso evita que o cliente tenha que solicitar informações a todo momento, o que economiza energia e otimiza a comunicação (Tosi et al., 2017).

Uma das razões pelas quais o BLE é tão eficiente é que ele foi desenvolvido para operar de forma intermitente. Os dispositivos permanecem a maior parte do tempo em modo *standby* e ativam sua comunicação apenas quando há necessidade de transmitir ou receber dados. Como resultado, o consumo de energia é muito menor do que o *Bluetooth* clássico, permitindo que dispositivos pequenos, como relógios e sensores, funcionem por longos períodos com baterias de pequena capacidade (Tosi et al., 2017).

Graças a essas características, o BLE é utilizado em diversas áreas, incluindo monitoramento de saúde (como medidores de pressão e glicemia), dispositivos portáteis (como *smartwatches* e fones de ouvido), automação residencial (sensores de temperatura e iluminação), e sistemas de localização, como *beacons* que detectam a presença de pessoas ou objetos próximos.

4 **PROGRAMAÇÃO DO MICROCONTROLADOR**

A programação do *NodeMCU* ESP32 neste projeto é realizada por meio da Arduino IDE, uma plataforma de desenvolvimento de *software* voltada para microcontroladores, amplamente utilizada devido à sua simplicidade e eficiência. Originalmente criada para as placas Arduino, sua compatibilidade foi expandida para incluir outros dispositivos, como o ESP32, fabricado pela Espressif Systems. Um dos grandes diferenciais da Arduino IDE é ser um programa *open source*, o que significa que seu código-fonte está disponível para que qualquer pessoa possa acessá-lo, modificá-lo e promover melhorias. Essa abordagem colaborativa permite que desenvolvedores ao redor do mundo criem novas bibliotecas e aprimorem funcionalidades, facilitando a inovação contínua da ferramenta. Muitas das bibliotecas utilizadas com o ESP32, por exemplo, foram desenvolvidas pela comunidade (Arduino..., 2024).

Ao programar o ESP32 na Arduino IDE, é utilizada uma linguagem de programação baseada em C/C++, o que possibilita a criação de projetos eficientes e versáteis. Mesmo sendo um microcontrolador complexo, o ESP32 pode ser configurado facilmente graças às bibliotecas prontas disponibilizadas pela plataforma. Estas bibliotecas abrangem uma variedade de funções, como controle de GPIOs (pinos digitais e analógicos), comunicação via Wi-Fi e *Bluetooth*, além da integração com sensores. A facilidade de uso e a documentação extensa tornam a Arduino IDE uma excelente opção para projetos que envolvem automação e conectividade.

4.1 PROGRAMANDO O ESP32 COM A ARDUINO IDE

Para programar o ESP32 utilizando a Arduino IDE, é necessário instalar e configurar o ambiente de desenvolvimento corretamente antes de enviar o código ao microcontrolador. A seguir, são apresentados os passos essenciais para a instalação e preparação da ferramenta.

1. Instalando a Arduino IDE

O primeiro passo é baixar e instalar a Arduino IDE no computador. A versão mais recente da IDE pode ser encontrada no site oficial do Arduino, sendo compatível com os sistemas operacionais Windows, Linux e macOS (Arduino..., 2024). Após o download, a instalação é simples e segue o padrão de instalação de softwares comuns.

2. Configurando o suporte ao ESP32

Por padrão, a Arduino IDE não inclui suporte ao ESP32, sendo necessário configurar o gerenciador de placas para adicioná-lo. Para isso, siga os passos abaixo:

- Abra a Arduino IDE e vá até o menu *File* (Arquivo) > *Preferences* (Preferências).
- Na aba *Settings* (Configurações), no campo *Additional Boards Manager URLs* (URLs Adicionais do Gerenciador de Placas), adicione a seguinte URL: *https://dl.espressif.com/dl/package_esp32_index.json.* Clique em OK para salvar as preferências.
- Em seguida, vá ao menu *Tools* (Ferramentas) > *Board* (Placa) > *Boards Manager* (Gerenciador de Placas).
- No *Boards Manager*, pesquise por ESP32 e instale o pacote chamado esp32.

Com isso, o suporte para o ESP32 estará habilitado na Arduino IDE.

3. Selecionando a placa ESP32

Após a instalação do pacote do ESP32, é necessário selecionar a placa correta na Arduino IDE. Para isso, vá ao menu *Tools* (Ferramentas) > *Board* (Placa) e selecione a opção "*ESP32 Dev Module"*, que corresponde à maioria das versões da placa *NodeMCU* ESP32.

4. Configurando a porta serial

Conecte a placa ESP32 ao computador usando um cabo USB. A Arduino IDE deve detectar automaticamente a porta serial associada ao dispositivo. Para

confirmar, vá ao menu *Tools* (Ferramentas) > *Port* (Porta) e selecione a porta correta (geralmente é exibida como COM no Windows).

5. Escrevendo e enviando o código ao microcontrolador

Com a placa ESP32 configurada, você pode começar a escrever o código na Arduino IDE. A estrutura básica de um programa para a Arduino IDE é composta pelas funções *setup()* e *loop()*, conforme a Figura 5.

Figura 5 — Estrutura básica de um algoritmo para Arduino.

```
1  void setup() {
2    // Configurações iniciais, como definir pinos de entrada/saída
3  }
4
5  void loop() {
6    // Código que será executado repetidamente
7  }
8
```

Fonte: O autor (2024).

Por exemplo, para piscar um LED conectado ao pino GPIO2 do ESP32, o código necessário está ilustrado na Figura 6.

Figura 6 — Algoritmo para piscar um LED conectado ao GPIO 2 do ESP32.

```
1  void setup() {
2    pinMode(2, OUTPUT); // Configura o pino 2 como saída
3  }
4
5  void loop() {
6    digitalWrite(2, HIGH); // Liga o LED
7    delay(1000);           // Aguarda 1 segundo
8    digitalWrite(2, LOW);  // Desliga o LED
9    delay(1000);           // Aguarda 1 segundo
10 }
```

Fonte: O autor (2024).

Depois de escrever o código, clique no botão de *Upload* (ícone de uma seta apontando para a direita) para compilar o programa e carregá-lo no ESP32. A Arduino IDE enviará o código para o microcontrolador e exibirá uma mensagem de sucesso após o carregamento.

6. Instalando a biblioteca do PZEM-004T

Para integrar o ESP32 ao módulo PZEM-004T, é necessário instalar uma biblioteca adicional. Com a Arduino IDE aberta, siga as etapas abaixo:

- No menu superior, selecione *Sketch* > *Include Library* > *Manage Libraries...*
- Na barra de pesquisa, digite "PZEM004T".
- Encontre a biblioteca "PZEM004Tv30" de "Jakub Mandula" e clique em *Install*.

Após a instalação, a biblioteca estará pronta para uso. As demais bibliotecas necessárias para o projeto já estarão inclusas na Arduino IDE.

4.2 MONTAGEM E PROGRAMAÇÃO DO ESP32

Neste projeto de medidor de consumo de energia elétrica, o ESP32 e o sensor PZEM-004T se comunicam por meio do protocolo UART, utilizando os pinos TX e RX de ambos os dispositivos. O pino TX do PZEM-004T é conectado ao RX do ESP32 (GPIO 16). O pino RX do PZEM-004T é conectado ao TX do ESP32 (GPIO 17), garantindo a troca de dados. A alimentação do PZEM-004T é feita conectando seus pinos VCC e GND à fonte DC de 5V e ao aterramento do ESP32, respectivamente.

A comunicação entre os dispositivos ocorre sem a necessidade de sincronização de relógio, desde que ambos operem com a mesma *baud rate* (neste caso, 9600 bps), o que garante a transmissão eficiente das leituras elétricas. A taxa de *baud rate* é configurada automaticamente pela biblioteca PZEM004Tv30, facilitando a integração e reduzindo a necessidade de configurações adicionais para estabelecer uma comunicação estável entre o ESP32 e o módulo PZEM-004T.

O ESP32 atua como um dispositivo central que coleta informações de tensão, corrente, potência e energia do PZEM-004T e transmite esses dados via *Bluetooth*

Low Energy para o aplicativo móvel. A integração entre UART e BLE permite o monitoramento em tempo real, aproveitando as capacidades de conectividade do ESP32. Além disso, a simplicidade da comunicação UART, com apenas dois fios de dados, facilita a configuração e possibilita que o PZEM-004T seja instalado próximo ao ponto de medição, enquanto o ESP32 pode ficar em uma posição mais conveniente para transmissão dos dados.

O código que será apresentado a seguir mostra como essas funcionalidades foram implementadas. Ele faz uso de bibliotecas específicas para controlar a comunicação serial UART, estabelecer conexões BLE e ler os parâmetros elétricos fornecidos pelo PZEM-004T, permitindo o monitoramento e a transmissão eficiente dos dados.

Figura 7 — Representação das bibliotecas utilizadas.

```
1   #include <PZEM004Tv30.h>
2   #include <BLEDevice.h>
3   #include <BLEServer.h>
4   #include <BLEUtils.h>
5   #include <BLE2902.h>
```

Fonte: O autor (2024).

Na Figura 7, o código é inicializado com a importação de algumas bibliotecas. Bibliotecas são como caixas de ferramentas que contêm várias funções prontas que podemos usar no nosso código sem precisar "reinventar a roda". As bibliotecas importadas são responsáveis por:

- *PZEM004Tv30*: Responsável por comunicar com o medidor de energia PZEM-004T, que fará a leitura da tensão, corrente, potência, energia consumida, frequência e fator de potência.
- *BLEDevice*, *BLEServer*, *BLEUtils*, *BLE2902*: Essas bibliotecas tratam da comunicação *Bluetooth Low Energy*. Com elas, o *NodeMCU* ESP32 pode se conectar a outro dispositivo, como um celular, e enviar ou receber dados.

Figura 8 — Inicialização do sensor PZEM-004T, configurando a comunicação via UART com o ESP32.

```
/*
  Inicializa a interface Serial2 (a segunda porta serial do ESP32),
  utilizando os pinos 16 (TX) e 17 (RX) para a comunicação UART com o PZEM004T
*/
#if defined(ESP32)
PZEM004Tv30 pzem(Serial2, 16, 17);
#else
PZEM004Tv30 pzem(Serial2);
#endif
```

Fonte: O autor (2024).

Na Figura 8, configura-se o ESP32 para se comunicar com o medidor de energia PZEM-004T por meio de uma conexão UART. Os pinos 16 e 17 são utilizados para conectar os fios de comunicação ao ESP32. O código também realiza uma verificação para determinar se o dispositivo em uso é realmente o ESP32. Caso verdadeiro, ele utiliza esses pinos específicos para a comunicação.

Figura 9 — Definição dos UUIDs para o servidor BLE, além dos buffers de leitura para manipular os dados.

```
// UUID do Serviço e das Características
// (identificadores únicos universais para o serviço e características BLE)
#define SERVICE_UUID "C09E244B-4E39-481B-87C8-1BBF28072FF0"
#define READ_UUID "C09E244B-4E39-481B-87C8-1BBF28072FF1"
#define WRITE_UUID "C09E244B-4E39-481B-87C8-1BBF28072FF2"

// Tamanho do Buffer para leitura e envio dos dados
#define LEITURA_SIZE 64
#define BUFFER_SIZE 16
```

Fonte: O autor (2024).

Na Figura 9, são definidos os UUIDs, endereços exclusivos usados pelo *Bluetooth* para identificar serviços e características no dispositivo. Os UUIDs funcionam como um "endereço de e-mail" único, identificando cada tipo de dado que será enviado ou recebido via *Bluetooth*.

- *SERVICE_UUID*: Define o serviço *Bluetooth* criado (neste caso, relacionado à medição de energia).

- *READ_UUID*: É utilizado pelo cliente para realizar a leitura dos dados de energia.
- *WRITE_UUID*: É usado para que o cliente consiga enviar um comando de *reset* no valor acumulado de consumo de energia.

Além disso, os valores *LEITURA_SIZE* e *BUFFER_SIZE* determinam o tamanho dos vetores usados para armazenar os dados temporariamente antes de serem processados e enviados.

Figura 10 — Variáveis que são utilizadas em todo o algoritmo.

```
27  BLEServer *pServer = NULL;  // Ponteiro para o servidor BLE
28  BLECharacteristic *pCharacteristic = NULL;  // Característica BLE que permite leitura/escrita de dados
29  float tensao, corrente, potencia, energia, frequencia, fp;  // Variáveis para armazenar leituras do PZEM
30  bool deviceConnected = false, oldDeviceConnected = false, flag = true;  // Flags de conexão BLE
31  char leitura[LEITURA_SIZE], buffer[BUFFER_SIZE];  // Buffer para envio via Bluetooth
```

Fonte: O autor (2024).

Na Figura 10, as variáveis declaradas são utilizadas ao longo do código para armazenar informações essenciais. As variáveis *tensao*, *corrente*, *potencia*, *energia*, *frequencia* e *fp* são empregadas para guardar temporariamente as medições realizadas pelo PZEM-004T. As variáveis *deviceConnected* e *oldDeviceConnected* garantem que um dispositivo, como um celular, esteja conectado ao ESP32 via *Bluetooth*. Já os vetores *leitura* e *buffer* armazenam temporariamente os dados de leitura do sensor PZEM antes que esses sejam enviados para o aplicativo móvel via *Bluetooth*. Essas variáveis desempenham um papel crucial no funcionamento do sistema, assegurando a captura, armazenamento e transmissão adequadas dos dados de energia.

Figura 11 — Callback de conexão BLE.

```
33  // Classe para callbacks do servidor BLE (lidar com conexão e desconexão)
34  class MyServerCallbacks : public BLEServerCallbacks {
35    void onConnect(BLEServer *pServer) {
36      deviceConnected = true;  // Dispositivo BLE conectado
37    };
38
39    void onDisconnect(BLEServer *pServer) {
40      deviceConnected = false;  // Dispositivo BLE desconectado
41    }
42  };
```

Fonte: O autor (2024).

Na Figura 11, a classe *MyServerCallbacks* é responsável por gerenciar as conexões *Bluetooth* no ESP32. Quando um dispositivo, como um celular, se conecta ao ESP32 via *Bluetooth*, o método *onConnect* é chamado, definindo a variável *deviceConnected* como *true*. Isso indica que um dispositivo está conectado. Por outro lado, quando o dispositivo se desconecta, o método *onDisconnect* é chamado, definindo *deviceConnected* como *false*. Este gerenciamento de conexões é crucial para que o código saiba quando deve iniciar ou interromper o envio dos dados de energia, garantindo que as leituras do PZEM-004T sejam transmitidas apenas quando houver um dispositivo conectado para recebê-las.

Figura 12 — Recebimento de comandos via BLE.

```
44  // Callback que lida com escrita de dados via BLE
45  class MyCallbacks : public BLECharacteristicCallbacks {
46    void onWrite(BLECharacteristic *pCharacteristic) {
47      String valor = pCharacteristic->getValue();
48      if (valor.length() > 0) {
49        Serial.println("*********");
50        // Comando recebido pelo aplicativo BLE conectado para resetar a energia
51        if (valor.indexOf("R") != -1) {
52          Serial.println("Resetando a contagem de energia");
53          pzem.resetEnergy(); // Reseta o contador de energia do PZEM004T
54        }
55        Serial.println("*********");
56      }
57    }
58  };
```

Fonte: O autor (2024).

Na Figura 12, o código está preparado para receber comandos via *Bluetooth*. Quando um dispositivo conectado (como o celular) envia uma mensagem, o ESP32 vai verificar se a mensagem contém a letra "R". Se sim, ele reseta a contagem de energia, ou seja, apaga o valor armazenado e começa a medição do zero.

Figura 13 — Função responsável pela leitura do sensor PZEM.

```
60  // Função para realizar a leitura dos dados do sensor PZEM004T
61  void leituraPZEM() {
62    // Captura os valores das leituras
63    tensao = pzem.voltage();
64    corrente = pzem.current();
65    potencia = pzem.power();
66    energia = pzem.energy();
67    frequencia = pzem.frequency();
68    fp = pzem.pf();
69
70    // Verifica se as leituras são válidas; se não forem, envia um erro por BLE
71    if (isnan(tensao) || isnan(corrente) || isnan(potencia)
72        || isnan(energia) || isnan(frequencia) || isnan(fp)) {
73      Serial.println("Erro na leitura dos dados");
74      pCharacteristic->setValue("Erro");
75      pCharacteristic->notify();  // Notifica o dispositivo BLE conectado (envia "Erro")
76    } else {
77      // Caso as leituras sejam válidas, mostra os valores e envia via BLE
78      Serial.printf("Tensao: %.1fV | Corrente: %.2fA | Potencia: %.2fW | Energia: %.3fkWh
79                    | Frequencia: %.1fHz | FP: %.2f\n", tensao, corrente, potencia,
80                    energia, frequencia, fp);
81      enviaDadosBluetooth();  // Chama a função para enviar os dados via BLE
82    }
83  }
```

Fonte: O autor (2024).

Na Figura 13, a função *leituraPZEM* é responsável por ler os dados do medidor de energia PZEM-004T. Ela coleta diversas informações, incluindo tensão, corrente, potência, energia consumida, frequência e fator de potência. Após coletar essas leituras, o algoritmo verifica se algum desses valores é inválido, ou seja, se aparece como "*NaN*" (que significa "não é um número"). Se qualquer uma das leituras for inválida, o ESP32 envia uma mensagem de erro para o dispositivo conectado, alertando sobre o problema. Caso todas as leituras sejam válidas, a função *enviaDadosBluetooth* é chamada.

Figura 14 — Função de envio dos dados via BLE.

```
85  // Função para enviar os dados lidos via Bluetooth
86  void enviaDadosBluetooth() {
87    // Prepara os dados no formato necessário e coloca no buffer
88    memset(leitura, 0, BUFFER_SIZE);  // Limpa o buffer antes de usar
89    dtostrf(tensao, 1, 1, leitura); strcat(leitura, "-");
90    dtostrf(corrente, 1, 2, buffer); strcat(leitura, buffer); strcat(leitura, "-");
91    dtostrf(potencia, 1, 2, buffer); strcat(leitura, buffer); strcat(leitura, "-");
92    dtostrf(energia, 1, 3, buffer); strcat(leitura, buffer); strcat(leitura, "-");
93    dtostrf(frequencia, 1, 1, buffer); strcat(leitura, buffer); strcat(leitura, "-");
94    dtostrf(fp, 1, 2, buffer); strcat(leitura, buffer);
95    leitura[strlen(leitura)] = '\0';     // Finaliza a string de leitura
96    pCharacteristic->setValue("");       // Limpa o valor anterior da característica BLE
97    pCharacteristic->setValue(leitura);  // Define o novo valor
98    pCharacteristic->notify();           // Notifica o dispositivo conectado via BLE
99  }
```

Fonte: O autor (2024).

Na Figura 14, a função *enviaDadosBluetooth* é responsável por enviar os dados coletados pelo sensor PZEM-004T para o dispositivo conectado via *Bluetooth*. Inicialmente, a função junta todas as informações lidas, como tensão, corrente, potência, energia, frequência e fator de potência, e organiza esses dados em um formato que o aplicativo irá processar. Todos os valores são convertidos para *strings* e concatenados com traços, resultando em um formato como "127.2-3.2-407.04...".

Após preparar os dados no formato correto, a função então os envia para o celular utilizando a característica BLE configurada para notificação (*READ_UUID*). A função limpa o *buffer* anterior, define o novo valor dos dados concatenados e notifica o dispositivo conectado via *Bluetooth*, garantindo que todas as informações sejam transmitidas corretamente e possam ser processadas pelo aplicativo móvel.

Figura 15 — Configuração inicial do microcontrolador ESP.

```
101  // Função de setup do ESP32
102  void setup() {
103    Serial.begin(115200);  // Inicializa a comunicação serial
104
105    // Inicializa o dispositivo BLE (ESP32) com o nome "Medidor de Energia"
106    BLEDevice::init("Medidor de Energia");
107
108    // Cria o servidor BLE e define os callbacks para eventos de conexão/desconexão
109    BLEServer *pServer = BLEDevice::createServer();
110    pServer->setCallbacks(new MyServerCallbacks());
111
112    // Cria o serviço BLE
113    BLEService *pService = pServer->createService(SERVICE_UUID);
114
115    // Cria uma característica de leitura/notificação para enviar dados via BLE
116    pCharacteristic = pService->createCharacteristic(READ_UUID,
117                      BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_NOTIFY);
118
119    // Adiciona um descritor BLE para a característica
120    pCharacteristic->addDescriptor(new BLE2902());
121
122    // Cria uma característica de escrita para receber dados via BLE
123    BLECharacteristic *pCharacteristic = pService->createCharacteristic(WRITE_UUID,
124                                         BLECharacteristic::PROPERTY_WRITE);
125
126    // Define o callback para tratar dados recebidos via BLE
127    pCharacteristic->setCallbacks(new MyCallbacks());
128
129    pService->start();                  // Inicia o serviço BLE
130    pServer->getAdvertising()->start();  // Começa a anunciar o dispositivo BLE
131    Serial.println("Aguardando a conexao de um dispositivo...");
132  }
```

Fonte: O autor (2024).

A função *setup* é a primeira parte do código a ser executada quando o ESP32 é ligado, realizando várias inicializações essenciais, conforme ilustrado na Figura 15. Primeiro, a comunicação serial é estabelecida para a utilização do *console*, ajudando no processo de depuração e monitoramento. Em seguida, o *Bluetooth* é ativado e o ESP32 é anunciado como "Medidor de Energia", permitindo que outros dispositivos possam detectá-lo e se conectar a ele. Um serviço *Bluetooth* é então criado, configurando características que permitem tanto a leitura dos dados de energia quanto a recepção de comandos, como o *reset* do medidor.

Após configurar essas características, o serviço é iniciado e o dispositivo começa a se anunciar para permitir conexões. Por fim, uma mensagem de *status* é impressa no *console*, indicando que o ESP32 está pronto para aceitar conexões via

Bluetooth. Essas etapas garantem que o dispositivo esteja preparado para interagir com outros dispositivos e transmitir os dados de energia coletados.

Figura 16 — Função loop, que repete continuamente enquanto o ESP32 estiver ligado.

```
// Função de loop, executada continuamente
void loop() {
  // Se o dispositivo BLE estiver conectado
  if (deviceConnected) {
    if (flag) {
      // Primeira vez conectado, imprime o endereço do PZEM
      Serial.println("Conectado com sucesso");
      Serial.println("Endereco PZEM: ");
      Serial.println(pzem.readAddress(), HEX);
      flag = false;  // Flag para evitar a repetição do endereço PZEM
    }
    leituraPZEM();  // Faz a leitura dos dados e envia via BLE
    delay(2000);    // Aguarda 2 segundos entre leituras
  }

  // Quando o dispositivo BLE se desconectar
  if (!deviceConnected && oldDeviceConnected) {
    delay(500);                  // Aguarda um tempo para o servidor BLE estabilizar
    flag = true;                 // Reseta a flag de conexão
    pServer->startAdvertising();  // Começa a anunciar o dispositivo BLE novamente
    Serial.println("Aguardando a conexao de um dispositivo...");
    oldDeviceConnected = deviceConnected;
  }

  // Quando um novo dispositivo se conectar
  if (deviceConnected && !oldDeviceConnected) {
    oldDeviceConnected = deviceConnected;  // Atualiza o status de conexão
  }
}
```

Fonte: O autor (2024).

Finalmente, na função *loop*, que é repetida continuamente enquanto o ESP32 estiver ligado, o código verifica constantemente se há algum dispositivo conectado via *Bluetooth*. Se houver um dispositivo conectado, o código realiza uma nova leitura dos dados de energia a cada 2 segundos, capturando todas as informações do PZEM-004T. Essas informações são então enviadas para o dispositivo conectado via *Bluetooth*, conforme ilustrado na Figura 16. Caso o dispositivo se desconecte, o código detecta a desconexão e reinicia o processo de anúncio do servidor BLE, permitindo que novos dispositivos possam se conectar. Esse ciclo contínuo garante

que os dados de energia sejam sempre atualizados e enviados para o dispositivo conectado, mantendo a comunicação eficiente e confiável.

5 **APLICATIVO MÓVEL**

Thunkable é uma plataforma de desenvolvimento de aplicativos móveis baseada em blocos de construção visual, permitindo que usuários criem aplicativos funcionais sem a necessidade de conhecimento avançado em programação. A plataforma oferece uma interface intuitiva onde os desenvolvedores podem arrastar e soltar componentes, configurando comportamentos e ações através de blocos de código visual. Isso facilita o desenvolvimento rápido de aplicativos para Android e iOS, proporcionando uma experiência acessível tanto para iniciantes quanto para desenvolvedores experientes (Thunkable, 2024).

A principal vantagem do Thunkable é sua capacidade de abstrair a complexidade da codificação, permitindo que os usuários foquem na lógica e no design do aplicativo. Além disso, a plataforma suporta a integração com vários serviços, como o Firebase, expandindo as funcionalidades dos aplicativos criados. Com suporte para componentes de interface do usuário, conectividade com banco de dados, funcionalidades de sensores e muito mais, se destaca como uma ferramenta poderosa para prototipagem e desenvolvimento de aplicativos completos (Thunkable, 2024).

5.1 CÓDIGO DO APLICATIVO

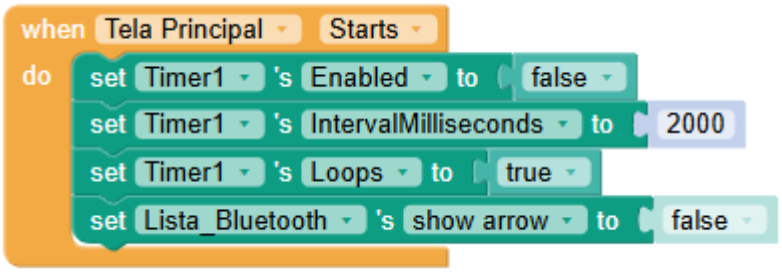
Neste tópico, será detalhado o funcionamento do código do aplicativo móvel desenvolvido na plataforma Thunkable. O aplicativo foi projetado para se comunicar com o ESP32 utilizando o *Bluetooth Low Energy* (BLE). Serão exploradas as funcionalidades implementadas, incluindo a inicialização da interface, a conexão e desconexão de dispositivos *Bluetooth*, o procedimento de escaneamento de dispositivos, e a recepção e exibição de dados do PZEM no aplicativo. Este tópico visa fornecer uma compreensão completa do fluxo de trabalho e das funcionalidades do aplicativo, explicando de forma clara e detalhada cada segmento do código. A tela principal do aplicativo é mostrada na Figura 17.

Figura 17 — Tela principal do aplicativo.

Fonte: O autor (2024).

Na Figura 18, está representado o bloco de código responsável pela inicialização da tela principal do aplicativo. Este bloco é ativado assim que a tela é iniciada. Primeiramente, o *Timer1* é configurado para iniciar desativado, com um intervalo de 2000 milissegundos (2 segundos), e programado para se repetir continuamente. A propriedade de exibição de seta da *Lista_Bluetooth* é definida como falsa, ocultando a seta de exibição ao lado dos itens da lista. Isso garante que a interface do usuário permaneça limpa e organizada, facilitando a navegação e o uso do aplicativo.

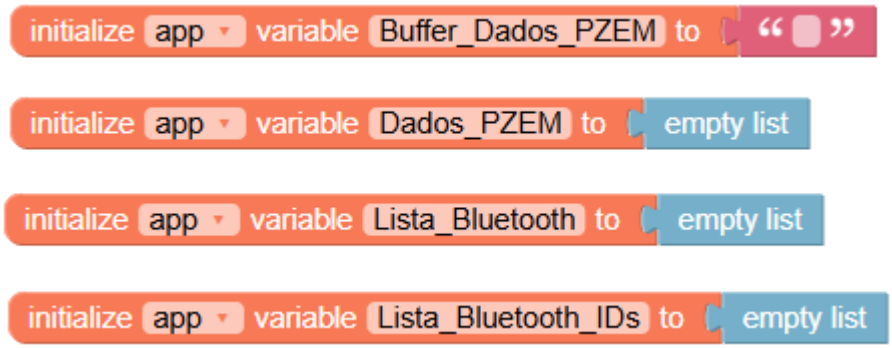
Figura 18 — Código que é inicializado ao abrir o aplicativo.



Fonte: O autor (2024).

Na Figura 19, são inicializadas quatro variáveis globais (*Buffer_Dados_PZEM*, *Dados_PZEM*, *Lista_Bluetooth* e *Lista_Bluetooth_IDs*). Três dessas variáveis são listas vazias, que serão utilizadas posteriormente no aplicativo para armazenar dados do módulo de medição de energia PZEM e dispositivos *Bluetooth*. Essas variáveis são essenciais para a organização e o gerenciamento eficiente dos dados coletados, assegurando o correto funcionamento do aplicativo.

Figura 19 — Declaração das variáveis globais do aplicativo.

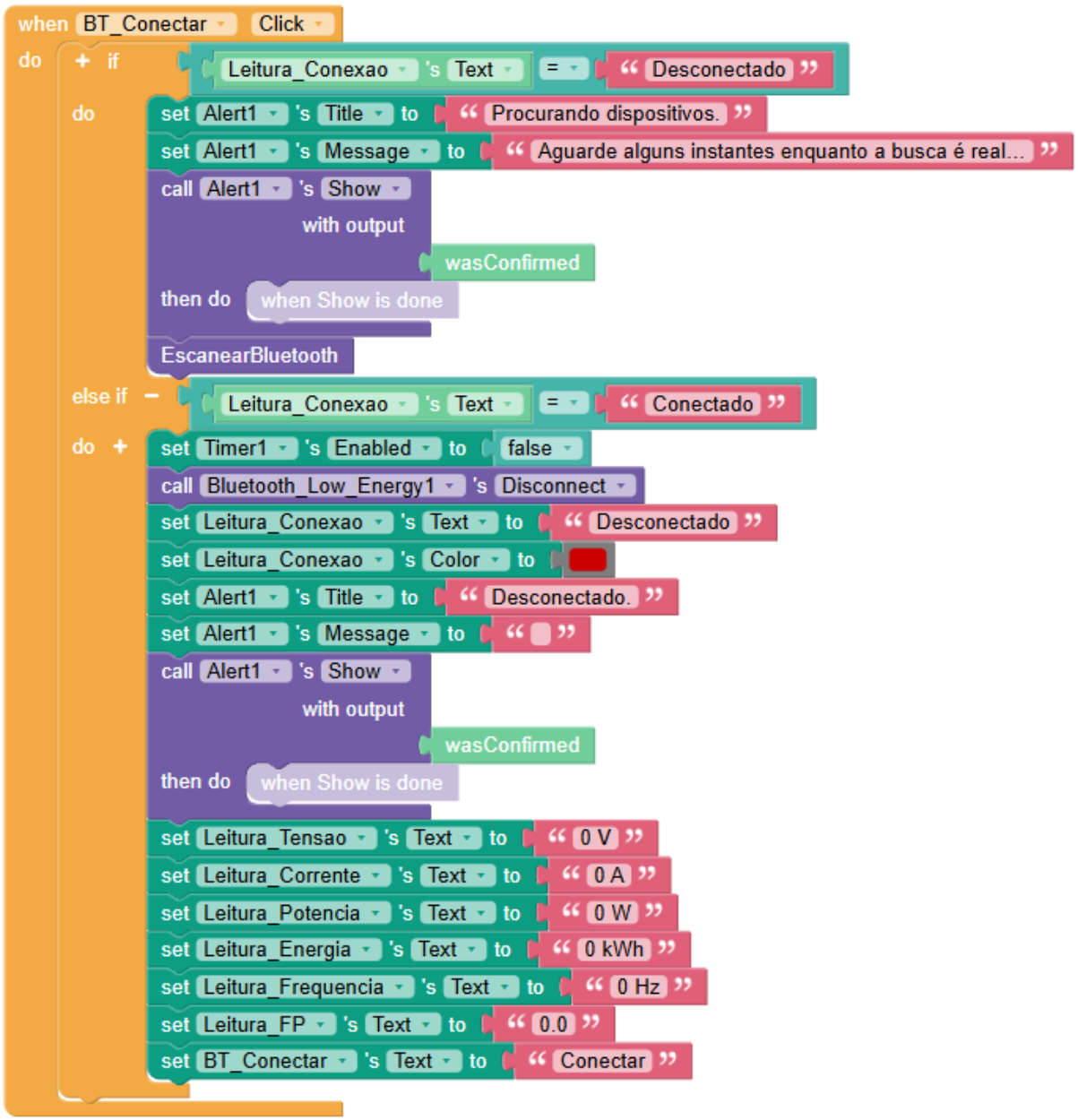


Fonte: O autor (2024).

Na Figura 20, é apresentado o bloco de código que define o comportamento do aplicativo ao clicar no botão "*BT_Conectar*". Quando o botão é pressionado, o aplicativo verifica o texto da variável "*Leitura_Conexao*". Se o texto for "Desconectado", o aplicativo exibe uma mensagem informando que a busca por dispositivos está em andamento e executa a função para escanear dispositivos *Bluetooth*. Caso o texto seja "Conectado", o *Timer1* é desativado, o dispositivo

Bluetooth é desconectado, e as leituras de tensão, corrente, potência, energia, frequência e fator de potência são redefinidas para seus valores iniciais. Após esse processo, uma mensagem de alerta é exibida para informar que a conexão foi encerrada. Essas ações garantem a correta gestão das conexões *Bluetooth*, permitindo ao usuário conectar e desconectar dispositivos de maneira eficiente e intuitiva.

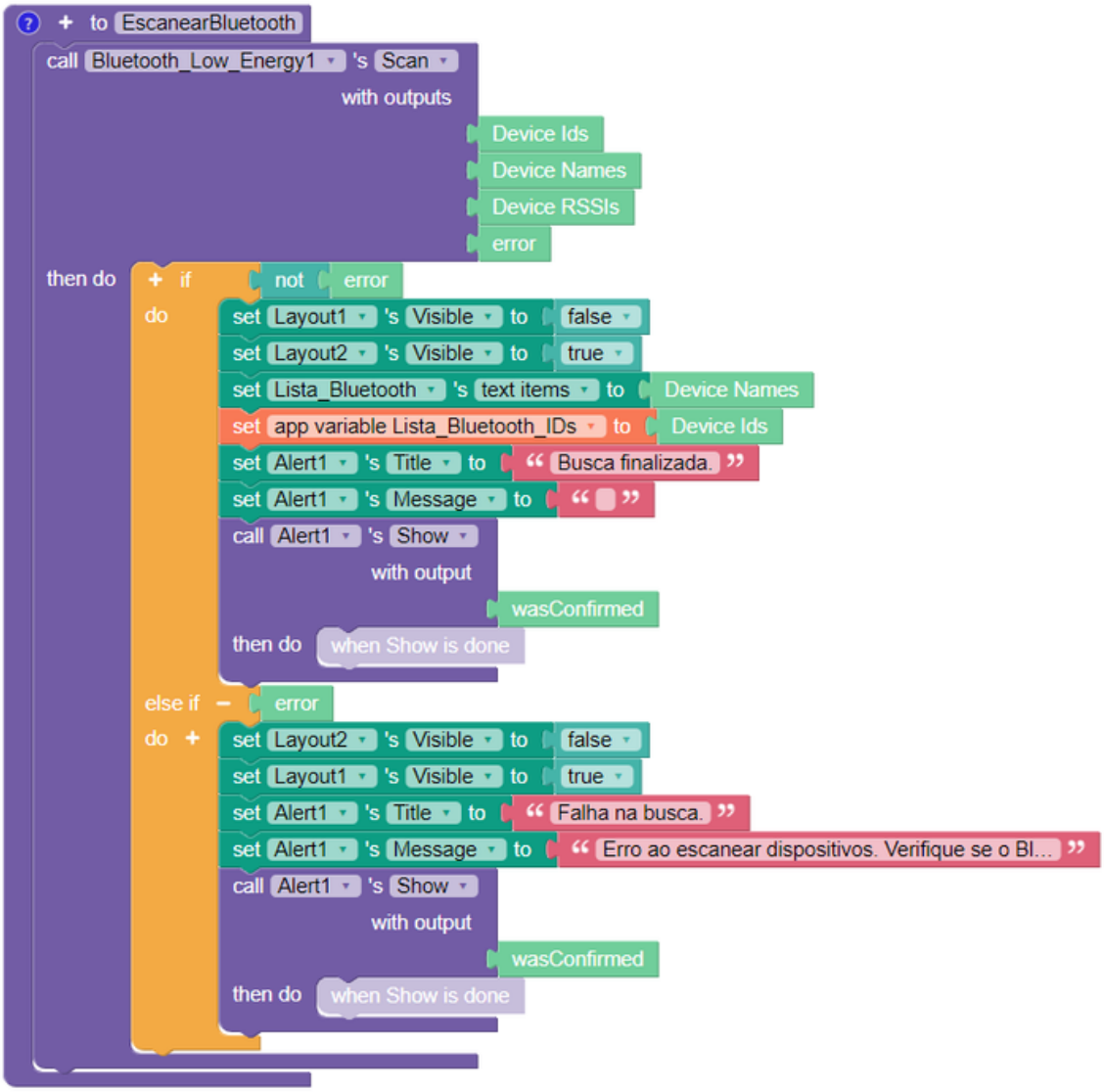
Figura 20 — Trecho do código que será executado ao clicar no botão conectar/desconectar.



Fonte: O autor (2024).

Na Figura 21, é demonstrado o procedimento para escanear dispositivos *Bluetooth*. Ao chamar o método "*Bluetooth_Low_Energy1 Scan"*, o aplicativo inicia a busca por dispositivos BLE nas proximidades. Se a busca for bem-sucedida, a visibilidade dos layouts da interface é alternada para exibir os dispositivos encontrados, e a lista com os nomes desses dispositivos é atualizada. Caso ocorra algum erro durante a busca, uma mensagem de alerta é exibida para informar o usuário sobre o problema, garantindo que ele esteja ciente de qualquer falha na detecção dos dispositivos *Bluetooth*. Esse procedimento é essencial para assegurar que o usuário possa localizar dispositivos BLE.

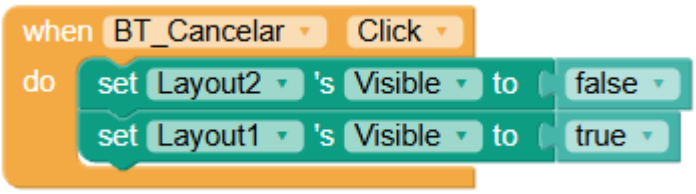
Figura 21 — Função para escanear dispositivos via Bluetooth.



Fonte: O autor (2024).

Na Figura 22, o código mostrado define o comportamento ao clicar no botão "*BT_Cancelar*". Quando o botão é pressionado, o aplicativo altera a visibilidade dos layouts, ocultando "*Layout2*", que é utilizado para exibir os resultados da busca de dispositivos BLE, e exibindo "*Layout1*", que é a interface principal do aplicativo. Isso permite que o usuário, caso decida não se conectar a nenhum dispositivo, cancele a tentativa de conexão via BLE e retorne à tela inicial logo após a busca pelos dispositivos disponíveis.

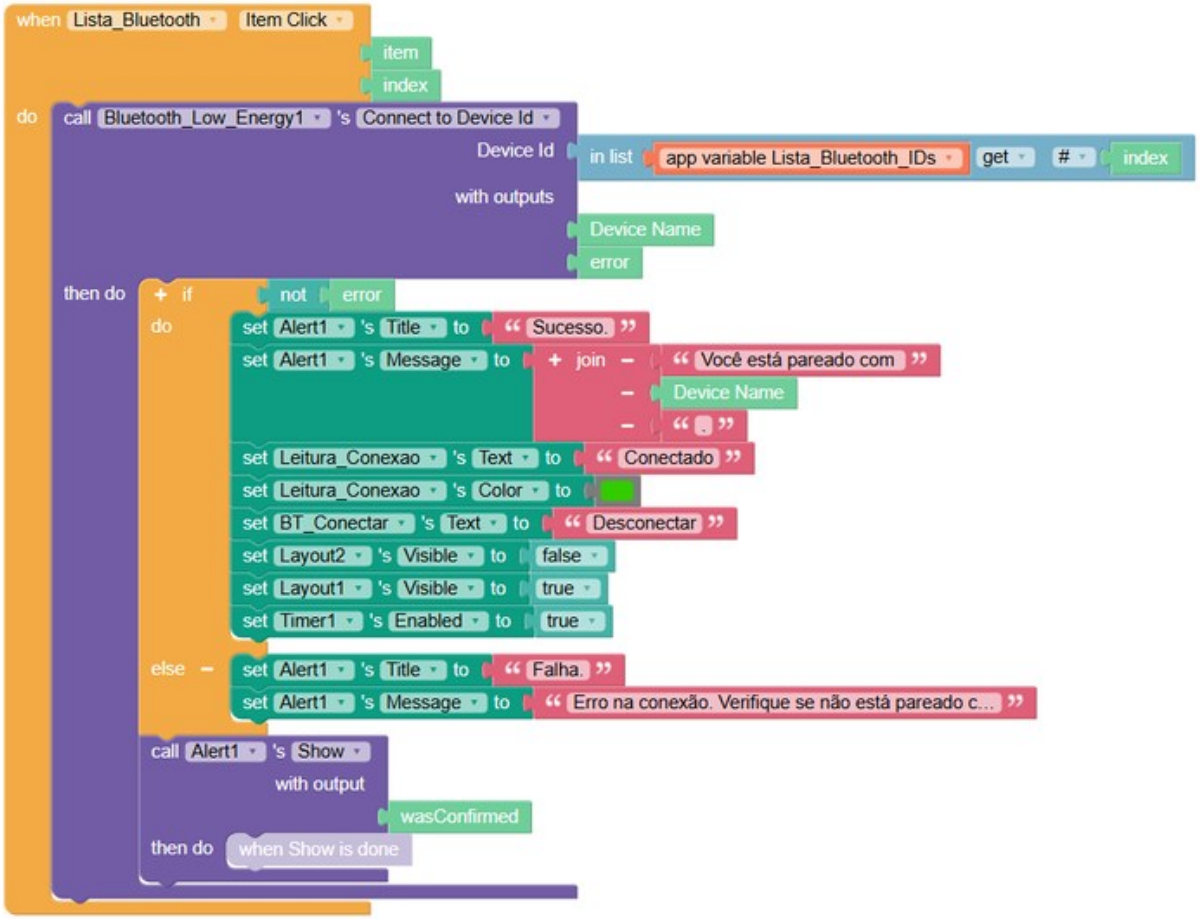
Figura 22 — Botão para voltar à tela inicial após a busca de dispositivos BLE.



Fonte: O autor (2024).

Na Figura 23, é exibido o bloco de código que lida com o evento de clique em um item da lista de dispositivos *Bluetooth*. Quando um item na *Lista_Bluetooth* é clicado, o aplicativo tenta se conectar ao dispositivo selecionado usando o identificador do dispositivo correspondente. Se a conexão for bem-sucedida, várias ações são realizadas: a mensagem de alerta é atualizada para indicar "Sucesso", o texto e a cor da variável *Leitura_Conexao* são ajustados para refletir que o dispositivo está "Conectado", o texto do botão *BT_Conectar* é alterado para "Desconectar", e os layouts são ajustados para exibir a interface apropriada. Além disso, o *Timer1* é ativado para permitir leituras periódicas. Caso ocorra um erro na conexão, a mensagem de alerta é atualizada para informar sobre a falha. Esse código garante que o aplicativo lide de forma adequada com as conexões *Bluetooth*, fornecendo feedback ao usuário sobre o status da conexão.

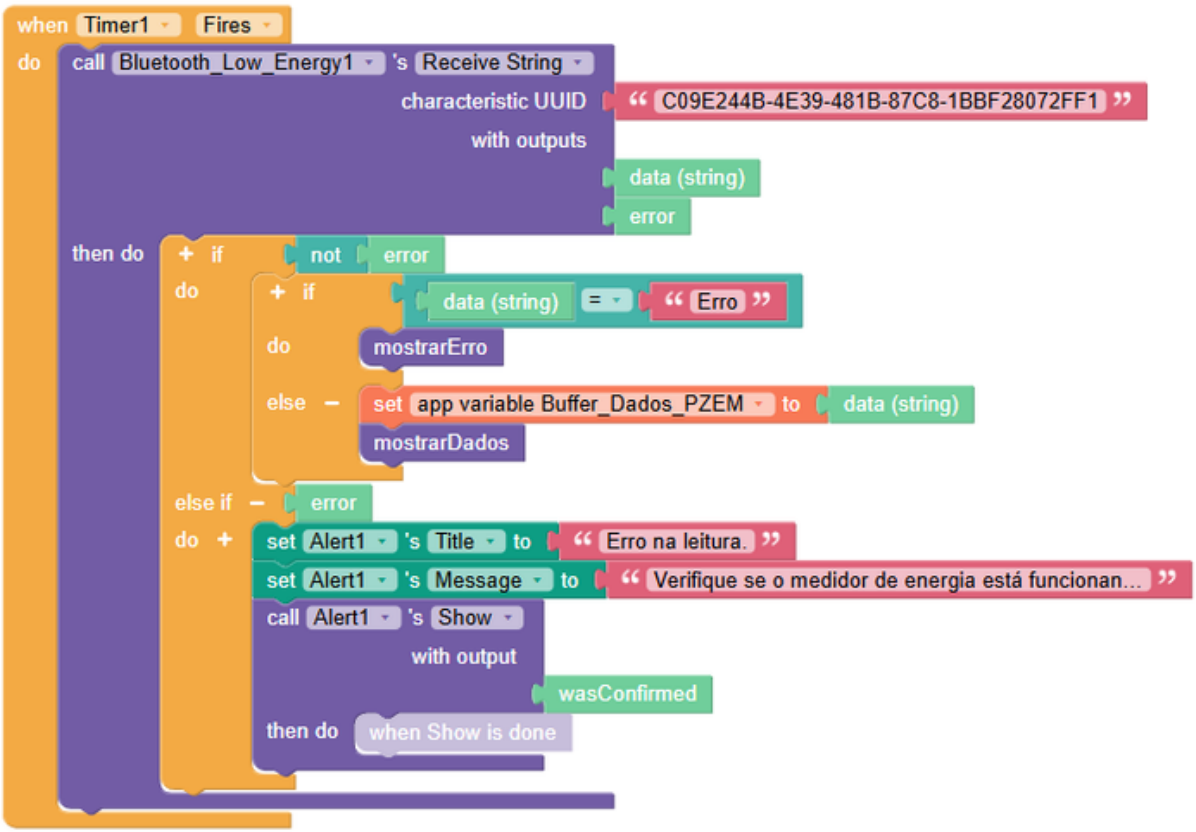
Figura 23 — Algoritmo responsável por gerenciar a conexão BLE ao selecionar um dispositivo na lista Bluetooth.



Fonte: O autor (2024).

Na Figura 24, o *Timer1* é um componente essencial para o funcionamento do aplicativo, pois é responsável pela execução periódica de determinadas ações. Na inicialização da tela principal, o *Timer1* é configurado para estar desativado, com um intervalo de 2 segundos e configurado para repetir continuamente. Quando o botão de conexão *Bluetooth* é acionado e a conexão é estabelecida, o *Timer1* é ativado, permitindo a leitura periódica dos dados do sensor PZEM. Dentro do evento *Timer1*, o aplicativo chama o método "*Bluetooth_Low_Energy1 Receive String*" para obter os dados do sensor. Se não houver erro na leitura, a função *mostrarDados* é chamada. Caso contrário, a função *mostrarErro* é executada.

Figura 24 — Algoritmo responsável pela leitura periódica dos dados do sensor PZEM.



Fonte: O autor (2024).

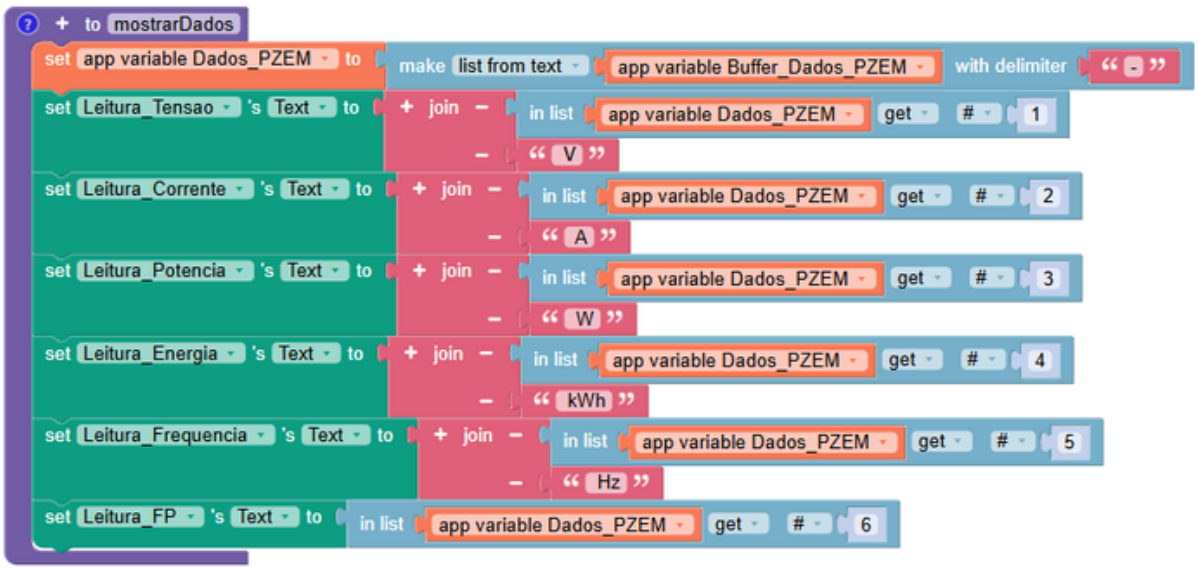
Na Figura 25, observa-se o método *mostrarErro*, responsável por definir os valores das leituras de tensão, corrente, potência, energia, frequência e fator de potência como "Erro" em caso de falha na leitura dos dados do módulo PZEM. Esse procedimento assegura que o usuário seja claramente informado sobre qualquer problema na obtenção dos dados, garantindo que ele perceba imediatamente qualquer falha na leitura das informações fornecidas pelo sensor.

Figura 25 — Função que define os valores das leituras como "Erro" em caso de falha na leitura dos dados do módulo PZEM.

Fonte: O autor (2024).

A Figura 26 apresenta o procedimento *mostrarDados*, que processa a *string* recebida do módulo PZEM e atualiza os valores das leituras na interface do usuário. Os valores de tensão, corrente, potência, energia, frequência e fator de potência são extraídos da *string* e exibidos de forma clara e compreensível na tela principal para o usuário.

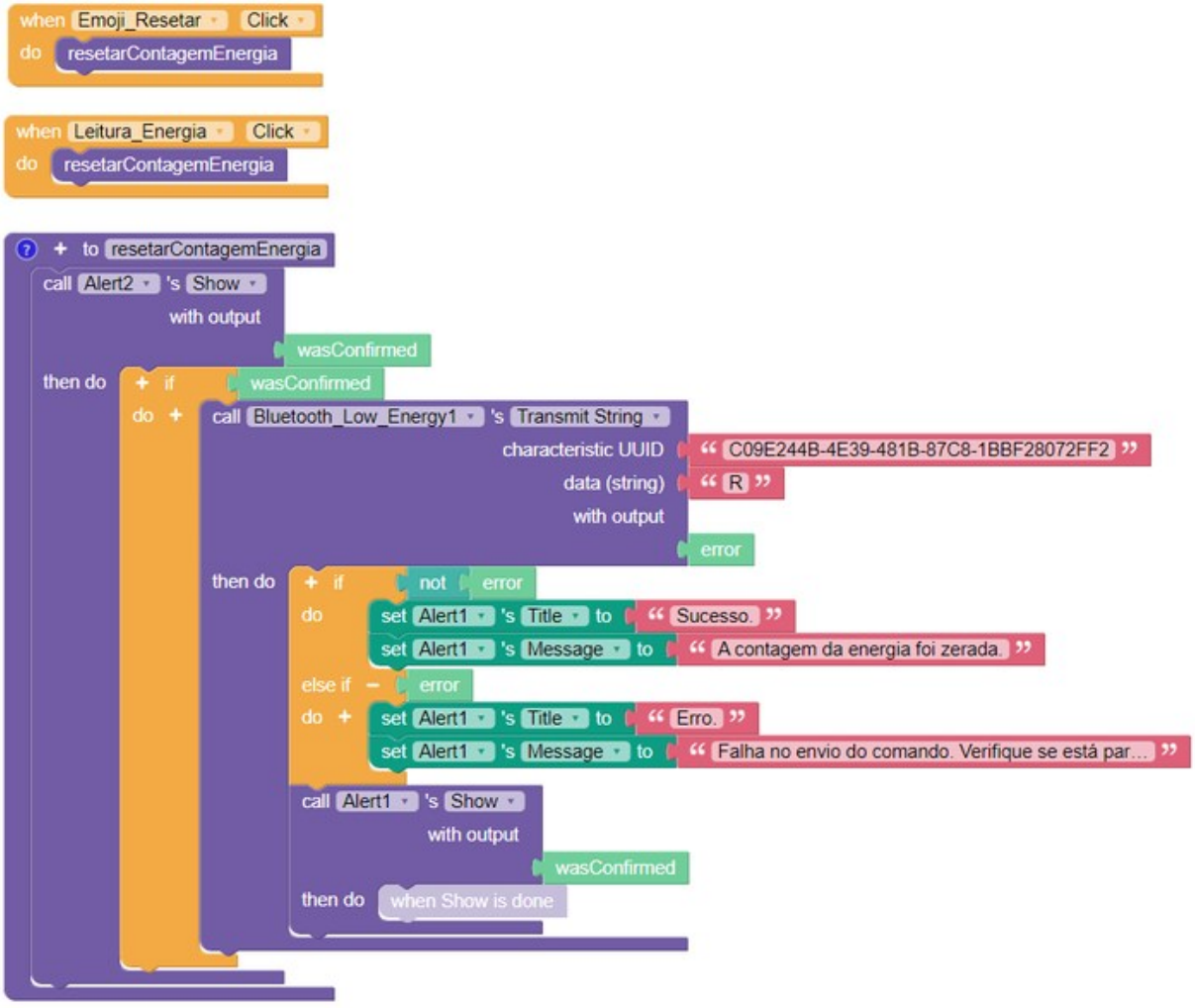
Figura 26 — Função que processa a string recebida do módulo PZEM e atualiza os valores das leituras na interface do usuário.



Fonte: O autor (2024).

Na Figura 27, o procedimento de resetar a energia é ativado quando o usuário clica no *emoji "Emoji_Resetar"* ou no texto *"Leitura_Energia"*. Quando acionado, o aplicativo exibe um alerta (*Alert2*) solicitando a confirmação do usuário para resetar a contagem de energia. Se o usuário confirmar, o aplicativo envia a *string* "R" para o módulo *Bluetooth*, utilizando o UUID específico para escrita. Este comando é interpretado pelo ESP32, que então executa o comando de resetar a energia no sensor PZEM004T. Se a transmissão for bem-sucedida, uma mensagem de sucesso é exibida ao usuário. Caso contrário, uma mensagem de erro é apresentada.

Figura 27 — Procedimento utilizado para resetar a contagem de energia no sensor PZEM-004T.



Fonte: O autor (2024).

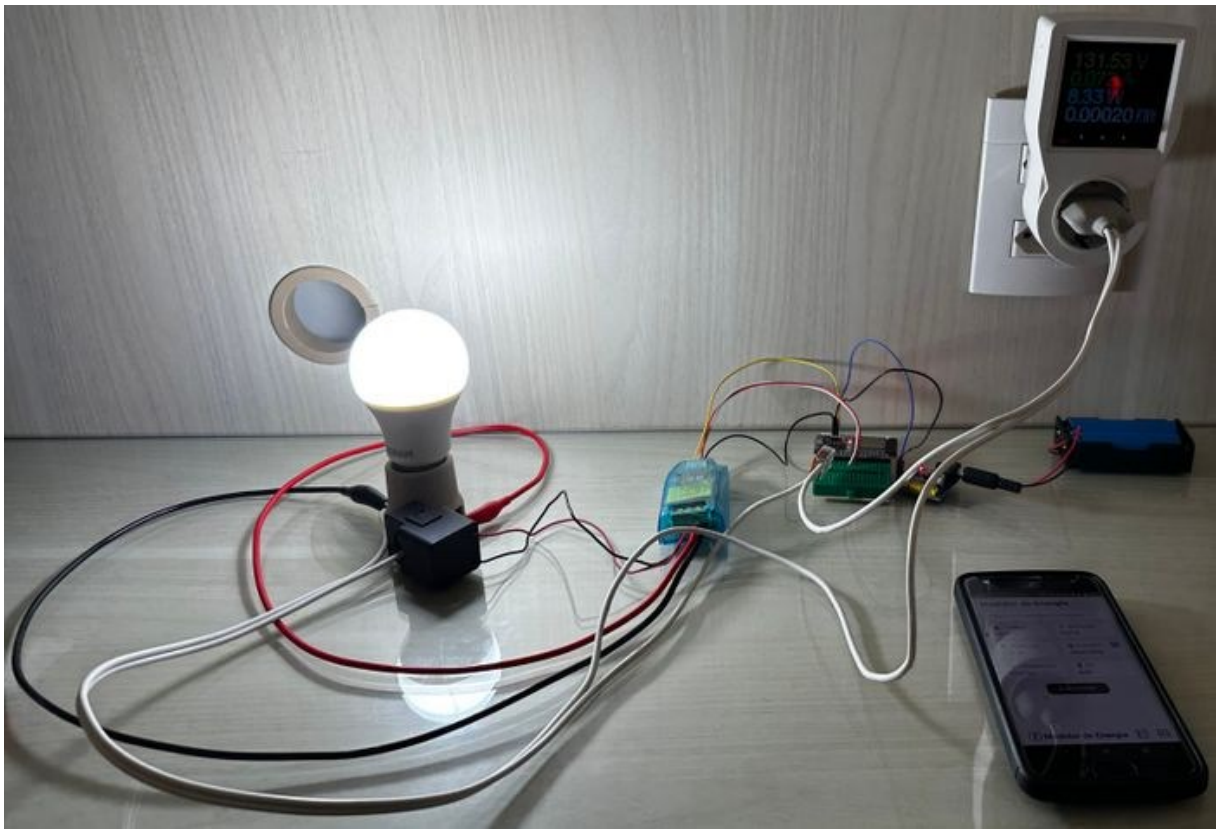
6 **RESULTADOS**

Para obter os resultados, foram realizadas medições em uma lâmpada LED da marca OSRAM com potência nominal de 8W. Durante o teste, utilizou-se dois medidores simultâneos: o PZEM-004T, desenvolvido neste projeto, e o medidor Atorch S1 (Figura 28), para garantir que ambos captassem as mesmas variações da rede elétrica. A Figura 29 mostra o circuito com a lâmpada sendo aferido pelos dois medidores, eliminando possíveis discrepâncias causadas por oscilações de tensão ou interferências. A medição teve duração total de 45 minutos.

Figura 28 — Medidor Atorch S1.

Fonte: O autor (2024).

Figura 29 — Análise simultânea do medidor S1 e do protótipo deste projeto, utilizando o PZEM-004T.



Fonte: O autor (2024).

Cada um dos dispositivos foi configurado para realizar as medições em tempo real. A integração do PZEM-004T com o microcontrolador ESP32 permitiu a transmissão dos dados para o aplicativo desenvolvido especificamente para este projeto, enquanto o Atorch S1 transmitiu suas leituras por meio do aplicativo *Smart Life*. A tela de ambos os aplicativos foi capturada para registro e análise dos resultados obtidos, permitindo uma visualização clara dos dados medidos por cada equipamento.

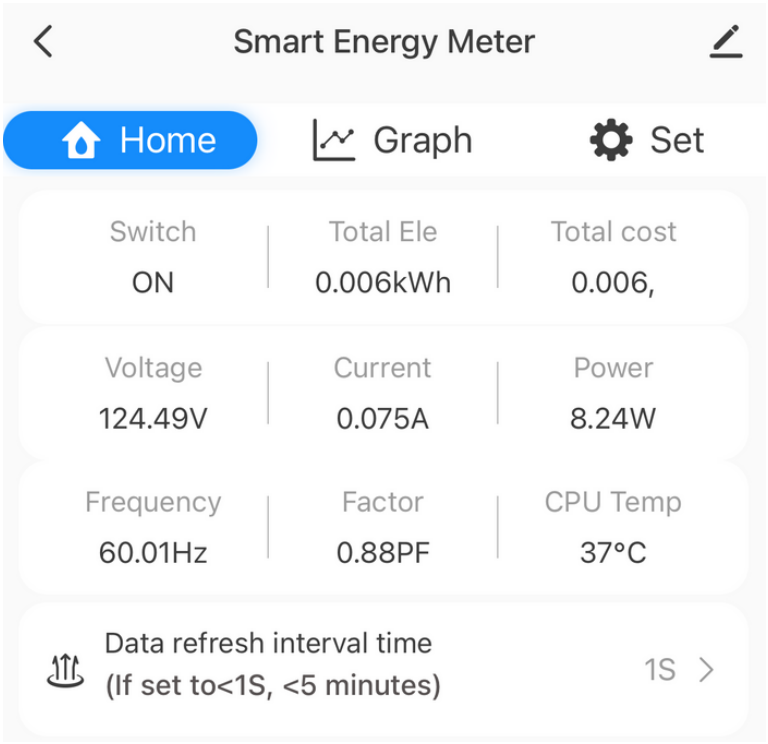
As imagens inseridas abaixo ilustram os resultados obtidos com os dois sistemas de medição utilizados. A Figura 30 reflete os dados do PZEM-004T sendo lidos e enviados para o aplicativo desenvolvido na plataforma Thunkable.

Figura 30 — Resultados obtidos com o medidor PZEM-004T.

Fonte: O autor (2024).

Na Figura 31, são exibidos dados semelhantes, fornecidos pelo aplicativo *Smart Life*, que se conecta ao medidor S1 da Atorch. Esses dados serão utilizados para comparar as leituras obtidas pelo PZEM, permitindo verificar a precisão e a consistência entre os dois dispositivos de medição. Essa comparação é fundamental para validar a eficácia do PZEM nas medições de energia.

Figura 31 — Resultados obtidos com o medidor da Atorch.



Fonte: O autor (2024).

A Tabela 1 apresenta os dados obtidos pelos medidores PZEM-004T e Atorch S1. São mostrados os valores de tensão, corrente, potência, energia, frequência e fator de potência registrados por cada medidor. Esses resultados permitem avaliar a precisão de cada dispositivo e serão utilizados para o cálculo do erro percentual entre as medições.

Tabela 1 — Resultados obtidos com os medidores PZEM-004T e Atorch S1.

-	**PZEM-004T**	**ATORCH S1**
Tensão [V]	124,3	124,49
Corrente [A]	0,07	0,075
Potência [W]	8,00	8,24
Energia [kWh]	0,006	0,006
Frequência [Hz]	60,0	60,01
Fator de Potência	0,87	0,88

Fonte: O autor (2024).

A Tabela 2 apresenta as especificações técnicas do medidor PZEM-004T, fornecidas pela Peacefair, em conjunto com o TC de 100A. São listadas as escalas de medida, resoluções e precisões para cada grandeza monitorada pelo dispositivo. Estas informações são fundamentais para a análise dos resultados de medição, permitindo a verificação da precisão e a comparação com os dados obtidos pelo medidor Atorch S1. A tabela servirá como referência para análise do erro percentual e para avaliar a confiabilidade das medições realizadas pelo PZEM-004T.

Tabela 2 — Especificações técnicas do medidor PZEM-004T com TC de 100A.

Função	**Escala de Medida**	**Resolução**	**Precisão**
Tensão	80~260V	0,1V	0,5%
Corrente	0~100A	0,01A	0,5%
Potência Ativa	0~23kW	0,1W	0,5%
Energia	9999,99kWh	1Wh	0,5%
Frequência	45~65Hz	0,1Hz	0,5%
Fator de Potência	0.00~1.00	0,01	1%

Fonte: Peacefair (2024).

A Tabela 3 apresenta as especificações técnicas do medidor S1, fornecidas pela fabricante Atorch. As informações incluem a escala de medida e a precisão para cada grandeza monitorada pelo dispositivo. No entanto, a fabricante Atorch não fornece a resolução das medidas, o que pode interferir no cálculo do erro percentual, uma vez que diferentes resoluções podem afetar a exatidão da medição do dispositivo. Essas especificações são fundamentais para a análise comparativa entre os medidores e para a avaliação da precisão das leituras do PZEM-004T em relação ao S1.

Tabela 3 — Especificações técnicas do medidor Atorch S1.

Função	Escala de Medida	Precisão
Tensão	85~265V	1%
Corrente	0~16A	1%
Potência Ativa	0~3,68kW	1%
Energia	0~99999kWh	1%
Frequência	50~60Hz	1%
Fator de Potência	0.00~1.00	1%

Fonte: Atorch (2024).

Para comparar a diferença entre os dois medidores, será calculado o erro percentual. Essa é uma abordagem comum e eficaz para comparar a precisão de diferentes medidores, pois fornece uma métrica quantitativa clara sobre a diferença entre os valores medidos e os valores reais. O erro percentual é calculado pela Equação 1 abaixo (Cálculo..., 2020).

$$Erro\ Percentual = \left(\frac{|Valor\ Medido\ S1 - Valor\ Medido\ PZEM|}{Valor\ Medido\ S1}\right) \times 100 \qquad (1)$$

Primeiramente, será calculado o erro percentual da tensão para verificar a diferença entre as medições dos dois medidores. A seguir, o erro percentual será calculado para as outras grandezas, incluindo corrente, potência, energia, frequência e fator de potência. Após o cálculo de cada grandeza, os resultados serão analisados e discutidos, fornecendo uma visão detalhada da precisão de cada medidor e das possíveis discrepâncias entre eles.

$$Erro\ Percentual - Tensão = \left(\frac{|124{,}49 - 124{,}3|}{124{,}49}\right) \times 100 = 0{,}1526\% \qquad (2)$$

A leitura da tensão realizada pelos medidores foi extremamente próxima. O erro percentual de 0,1526% é inferior à precisão de ambos os dispositivos (PZEM: 0,5%; S1: 1%). Esse resultado demonstra que ambos os medidores são confiáveis para leituras de tensão.

$$Erro\ Percentual - Corrente = \left(\frac{|0{,}075 - 0{,}07|}{0{,}075}\right) \times 100 = 6{,}67\% \qquad (3)$$

Neste caso, a leitura de corrente do PZEM foi prejudicada devido à sua resolução limitada de 0,01A, resultando em uma variação de 0,07A para 0,08A. O Atorch S1, por sua vez, fornece maior precisão decimal, com três casas, o que sugere uma leitura mais estável. Essa diferença pode ser significativa para aplicações mais sensíveis que envolvem baixo consumo.

É importante mencionar que, para medições em escalas maiores, a resolução do PZEM não afetará tanto o cálculo do erro percentual em comparação ao Atorch S1. Isso ocorre porque, em valores de corrente e potência mais elevados, a diferença absoluta entre as leituras dos dois medidores se torna relativamente menor. Dessa forma, a mesma diferença de resolução representa uma fração menor do valor total, reduzindo o impacto da limitação de resolução na precisão geral das medições. Assim, o PZEM continua sendo um medidor confiável e adequado para aplicações que envolvem correntes e potências mais elevadas.

$$Erro\ Percentual - Potência = \left(\frac{|8{,}24 - 8{,}00|}{8{,}24}\right) \times 100 = 2{,}913\% \quad (4)$$

A diferença de 2,913% nas leituras de potência não era o resultado esperado, mas é compreensível devido à resolução de medida do PZEM, que varia em incrementos de 0,1W. Embora essa variação seja aceitável devido à resolução limitada do PZEM, é importante ressaltar que esse detalhe influencia no cálculo do erro percentual, especialmente em medições de baixo consumo. Novamente, para medições em escalas maiores, essa limitação tem um impacto reduzido, tornando o PZEM um medidor confiável para aplicações que envolvem correntes e potências mais altas.

$$Erro\ Percentual - Energia = \left(\frac{|0{,}006 - 0{,}006|}{0{,}006}\right) \times 100 = 0\% \quad (5)$$

As leituras de energia são idênticas, demonstrando que ambos os medidores possuem alta precisão no acumulado de energia. Esse é um ponto positivo, especialmente para sistemas de monitoramento de consumo prolongado, onde a precisão acumulativa é crucial para a confiabilidade dos dados.

$$Erro\ Percentual - Frequência = \left(\frac{|60{,}01 - 60{,}0|}{60{,}01}\right) \times 100 = 0{,}017\% \quad (6)$$

A diferença nas leituras de frequência é muito pequena e está dentro da margem de erro de cada dispositivo, indicando que ambos os medidores conseguem acompanhar com precisão as variações mínimas da rede elétrica.

$$Erro\ Percentual - Fator\ de\ Potência = \left(\frac{|0{,}88 - 0{,}87|}{0{,}88}\right) \times 100 = 1{,}14\% \qquad (7)$$

Considerando que a precisão de ambos os medidores é de 1%, o valor medido poderia variar até 1% para cima ou para baixo em cada dispositivo. Portanto, a diferença de 1,14% no fator de potência está dentro da faixa aceitável, uma vez que cada medidor tem uma margem de erro que pode contribuir para essa variação.

A análise revela que, de modo geral, os dois dispositivos apresentaram resultados consistentes. Pequenas diferenças nas leituras de corrente e potência podem ser atribuídas às diferentes resoluções e precisões dos equipamentos. Por exemplo, o PZEM possui uma resolução de 0,01A para corrente e 0,1W para potência, o que pode causar pequenos arredondamentos em equipamentos de baixo consumo. No geral, a energia medida foi idêntica, indicando que ambos os dispositivos acompanharam corretamente o consumo da lâmpada ao longo do tempo. A tensão e a frequência também ficaram muito próximas, com variações leves dentro da margem esperada para cada dispositivo.

Ambos os medidores se mostraram confiáveis para o propósito da análise, apresentando diferenças mínimas nas leituras, em conformidade com a precisão e a resolução informadas pelos fabricantes. A metodologia aplicada, que utilizou medições simultâneas, garantiu que os dados fossem comparáveis e livres de fatores externos.

7 **CONCLUSÃO**

Este trabalho apresentou os principais resultados e aprendizados obtidos ao longo do desenvolvimento do projeto de medição de consumo energético com ESP32 e o medidor PZEM-004T. A solução proposta foi fundamentada na relevância da eficiência energética, especialmente em um cenário de crise hídrica e ondas de calor, que aumentam a demanda por monitoramento preciso e eficiente de energia elétrica.

A escolha do microcontrolador ESP32, destacando suas vantagens sobre o ESP8266, foi essencial para este projeto. O ESP32 permitiu maior versatilidade e eficiência na comunicação com o medidor PZEM-004T e foi programado com a plataforma Arduino IDE, o que simplificou seu desenvolvimento. A integração via UART foi um ponto-chave, garantindo que a comunicação entre os dispositivos fosse estável e precisa.

O uso do protocolo *Bluetooth Low Energy* possibilitou a comunicação do sistema com um aplicativo móvel desenvolvido na plataforma Thunkable. Esse aplicativo permitiu que os dados de consumo fossem acessados de forma intuitiva em tempo real, demonstrando a importância de interfaces amigáveis para sistemas de monitoramento.

Durante os testes, a medição de uma lâmpada de LED de 8W revelou que as leituras dos medidores PZEM004T e Atorch S1 foram consistentes, dentro das margens de precisão esperadas de cada aparelho. As pequenas discrepâncias nas leituras, como na corrente e na potência, foram analisadas e explicadas, demonstrando que o sistema desenvolvido é uma solução confiável para monitoramento. A metodologia adotada, utilizando ambos os medidores simultaneamente, garantiu que variações da rede elétrica afetassem igualmente os dois sistemas, eliminando interferências externas.

O projeto demonstrou não apenas a viabilidade técnica da solução, mas também destacou o potencial de uso do ESP32 e do PZEM-004T para criar sistemas personalizados e econômicos de monitoramento de energia. A solução atende a uma necessidade crescente de controle de consumo, considerando as mudanças no cenário energético global.

Com este projeto, foi possível explorar tanto aspectos técnicos quanto os desafios de desenvolvimento e implementação de uma solução prática. Em resposta ao problema de pesquisa, conclui-se que é viável desenvolver uma solução eficaz e precisa para monitoramento de energia utilizando microcontroladores e sensores de baixo custo. Para trabalhos futuros, recomenda-se a integração com plataformas em nuvem e novos protocolos de comunicação, como o Wi-Fi, ampliando a aplicabilidade do sistema. Aplicações em ambientes industriais e a implementação de automações baseadas no consumo, como ajustes automáticos de equipamentos para otimizar o uso de energia e reduzir custos, poderiam melhorar a eficiência energética, tornando o monitoramento ainda mais funcional e sustentável.

REFERÊNCIAS

ARDUINO - Home. Arduino. 2024. Disponível em: https://www.arduino.cc/. Acesso em: 17 out. 2024.

ARDUINO IDE | Arduino Documentation. Arduino Docs. 2024. Disponível em: https://docs.arduino.cc/software/ide/#ide-v2. Acesso em: 17 out. 2024.

ASHOKKUMAR, N. *et al*. Design and Implementation of IoT based Energy Efficient Smart Metering System for Domestic Applications. *In:* INTERNATIONAL CONFERENCE ON ADVANCED COMPUTING AND COMMUNICATION SYSTEMS (ICACCS), n. 9. 2023. Anais eletrônicos [...] Coimbatore, India: IEEE, 2023, p. 1208-1212. Disponível em: https://doi.org/10.1109/ICACCS57279.2023.10113012. Acesso em: 7 out. 2024.

ATORCH. **S1-WIFI version Power meter socket**: User Manual. 2024. Disponível em: http://en.atorch.cn/upload/20220629120019.pdf. Acesso em: 16 out. 2024.

BRASIL, Agência Nacional de Energia Elétrica. **Programa de Eficiência Energética (PEE)**. [Brasília]: Agência Nacional de Energia Elétrica, 2022. Disponível em: https://www.gov.br/aneel/pt-br/assuntos/programa-de-eficiencia-energetica. Acesso em: 7 out. 2024.

BRASIL, Ministério de Minas e Energia. **Agência Nacional de Energia Elétrica (Aneel) / PEE**. [Brasília]: Ministério de Minas e Energia, [2022?]. Disponível em: https://www.gov.br/mme/pt-br/assuntos/secretarias/sntep/quem-e-quem/setor-publico-1/aneel. Acesso em: 8 out. 2024.

BRASIL, Ministério de Minas e Energia. **Selo Procel leva mais economia e sustentabilidade aos brasileiros**. gov.br. [Brasília]: Ministério de Minas e Energia, 2023. Disponível em: https://www.gov.br/mme/pt-br/assuntos/noticias/selo-procel-leva-mais-economia-e-sustentabilidade-aos-brasileiros. Acesso em: 8 out. 2024.

CHEN, J.; HUANG, S.. Analysis and Comparison of UART, SPI and I2C. *In:* INTERNATIONAL CONFERENCE ON ELECTRICAL ENGINEERING, BIG DATA AND ALGORITHMS (EEBDA), n. 2. 2023. Anais eletrônicos [...] Changchun, China: IEEE, 2023, p. 272-276. Disponível em: https://doi.org/10.1109/EEBDA56825.2023.10090677. Acesso em: 16 out. 2024.

CÁLCULO Numérico: Um Livro Colaborativo. UFRGS, 2020, p. 24-26. Disponível em: https://www.ufrgs.br/reamat/CalculoNumerico/livro-py/livro-py.pdf. Acesso em: 22 out. 2024.

ESPRESSIF. **About Espressif**: Who We Are. Espressif. 2024. Disponível em: https://www.espressif.com/en/company/about-espressif. Acesso em: 11 out. 2024.

ESPRESSIF. **ESP32**. Espressif. 2024. Disponível em: https://www.espressif.com/en/products/socs/esp32. Acesso em: 11 out. 2024.

ESPRESSIF. **ESP8266**. Espressif. 2024. Disponível em: https://www.espressif.com/en/products/socs/esp8266. Acesso em: 11 out. 2024.

FANG, Y.; CHEN, X.. Design and Simulation of UART Serial Communication Module Based on VHDL. *In:* INTERNATIONAL WORKSHOP ON INTELLIGENT SYSTEMS AND APPLICATIONS, n. 3. 2011. Anais eletrônicos [...] Wuhan, China, 2011, p. 1-4. Disponível em: https://doi.org/10.1109/ISA.2011.5873448. Acesso em: 16 out. 2024.

INMET. **Ano de 2023 é o mais quente da série histórica no Brasil**. Instituto Nacional de Meteorologia. 2024. Disponível em: https://portal.inmet.gov.br/noticias/ano-de-2023-%C3%A9-o-mais-quente-da-hist%C3%B3ria-do-brasil. Acesso em: 10 out. 2024.

INMET. **Temperatura média atinge recorde no Brasil pelo quinto mês seguido**. Instituto Nacional de Meteorologia. 2023. Disponível em: https://portal.inmet.gov.br/noticias/temperatura-m%C3%A9dia-atinge-recorde-no-brasil-pelo-quinto-m%C3%AAs-seguido. Acesso em: 10 out. 2024.

IRENA. **Uma Transição Energética Acelerada Pode Acrescentar 40 milhões de Empregos no Setor Energético até 2050**. International Renewable Energy Agency. 2023. Disponível em: https://www.irena.org/News/pressreleases/2023/Nov/Accelerated-Energy-Transition-Can-Add-40-million-Energy-Sector-Jobs-by-2050-PT. Acesso em: 9 out. 2024.

OLIVEIRA, E.. **Conhecendo o NodeMCU-32S ESP32**. Blog MasterWalker Electronic Shop. 2017. Disponível em: https://blogmasterwalkershop.com.br/embarcados/esp32/conhecendo-o-nodemcu-32s-esp32. Acesso em: 12 out. 2024.

ONS. **Sobre o SIN**: O Sistema em Números. Operador Nacional do Sistema Elétrico. 2024. Disponível em: https://www.ons.org.br/paginas/sobre-o-sin/o-sistema-em-numeros. Acesso em: 10 out. 2024.

PEACEFAIR. **Company profile**. Peacefair. 2024. Disponível em: https://en.peacefair.cn/about_about/gsjj9f6.html. Acesso em: 15 out. 2024.

PEACEFAIR. **PZEM-004T Voltage Current Watt Meter Single Phase Small AC Voltmeter TTL Modbus Electric Energy Meters No outcase**. Peacefair. 2024. Disponível em: https://en.peacefair.cn/product/772.html. Acesso em: 15 out. 2024.

SALEEM, M. U.; USMAN, M. R.; SHAKIR, M.. Design, Implementation, and Deployment of an IoT Based Smart Energy Management System. *In:* IEEE ACCESS, n. 9. 2021. Anais eletrônicos [...] IEEE, 2021, p. 59649-59664. Disponível em: https://doi.org/10.1109/ACCESS.2021.3070960. Acesso em: 11 out. 2024.

SOUZA, D. F. *et al*. Efficiency, quality, and environmental impacts: A comparative study of residential artificial lighting. **Energy Reports**, v. 5, p. 409-424, 2019. Disponível em: https://doi.org/10.1016/j.egyr.2019.03.009. Acesso em: 7 out. 2024.

THUNKABLE. **Thunkable: Best no code app builder | No code app creation**. 2024. Disponível em: https://thunkable.com/. Acesso em: 20 out. 2024.

TOSI, J. *et al*. Performance Evaluation of Bluetooth Low Energy: A Systematic Review. **MDPI**, v. 17, n. 12, 13 dez. 2017. Disponível em: https://doi.org/10.3390/s17122898. Acesso em: 17 out. 2024.

APÊNDICE A — Código do Medidor de Energia

```
#include <PZEM004Tv30.h>
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
/*
  Inicializa a interface Serial2 (a segunda porta serial do ESP32),
   utilizando os pinos 16 (TX) e 17 (RX) para a comunicação UART com o PZEM004T
*/
#if defined(ESP32)
PZEM004Tv30 pzem(Serial2, 16, 17);
#else
PZEM004Tv30 pzem(Serial2);
#endif
// UUID do Serviço e das Características
// (identificadores únicos universais para o serviço e características BLE)
#define SERVICE_UUID "C09E244B-4E39-481B-87C8-1BBF28072FF0"
#define READ_UUID "C09E244B-4E39-481B-87C8-1BBF28072FF1"
#define WRITE_UUID "C09E244B-4E39-481B-87C8-1BBF28072FF2"
// Tamanho do Buffer para leitura e envio dos dados
#define LEITURA_SIZE 64
#define BUFFER_SIZE 16
BLEServer *pServer = NULL;  // Ponteiro para o servidor BLE
BLECharacteristic *pCharacteristic = NULL;  // Característica BLE que permite leitura/escrita de dados
float tensao, corrente, potencia, energia, frequencia, fp;  // Variáveis para armazenar leituras do PZEM
bool deviceConnected = false, oldDeviceConnected = false, flag = true;  // Flags de conexão BLE
```

```
char leitura[LEITURA_SIZE], buffer[BUFFER_SIZE];  // Buffer para envio via Bluetooth
// Classe para callbacks do servidor BLE (lidar com conexão e desconexão)
class MyServerCallbacks : public BLEServerCallbacks {
  void onConnect(BLEServer *pServer) {
    deviceConnected = true;  // Dispositivo BLE conectado
  };
  void onDisconnect(BLEServer *pServer) {
    deviceConnected = false;  // Dispositivo BLE desconectado
  }
};
// Callback que lida com escrita de dados via BLE
class MyCallbacks : public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic *pCharacteristic) {
    String valor = pCharacteristic->getValue();
    if (valor.length() > 0) {
      Serial.println("**********");
      // Comando recebido pelo aplicativo BLE conectado para resetar a energia
      if (valor.indexOf("R") != -1) {
        Serial.println("Resetando a contagem de energia");
        pzem.resetEnergy(); // Reseta o contador de energia do PZEM004T
      }
      Serial.println("**********");
    }
  }
};
// Função para realizar a leitura dos dados do sensor PZEM004T
void leituraPZEM() {
  // Captura os valores das leituras
  tensao = pzem.voltage();
  corrente = pzem.current();
  potencia = pzem.power();
  energia = pzem.energy();
```

```
        frequencia = pzem.frequency();
        fp = pzem.pf();
        // Verifica se as leituras são válidas; se não forem, envia um erro por BLE
         if (isnan(tensao) || isnan(corrente) || isnan(potencia) || isnan(energia) || isnan(frequencia) || isnan(fp)) {
          Serial.println("Erro na leitura dos dados");
          pCharacteristic->setValue("Erro");
           pCharacteristic->notify();   // Notifica o dispositivo BLE conectado (envia "Erro")
        }
        else {
         // Caso as leituras sejam válidas, mostra os valores e envia via BLE
          Serial.printf("Tensao: %.1fV | Corrente: %.2fA | Potencia: %.2fW | Energia: %.3fkWh | Frequencia: %.1fHz | FP: %.2f\n",
                        tensao, corrente, potencia, energia, frequencia, fp);
         enviaDadosBluetooth();  // Chama a função para enviar os dados via BLE
        }
       }
       // Função para enviar os dados lidos via Bluetooth
       void enviaDadosBluetooth() {
        // Prepara os dados no formato necessário e coloca no buffer
        memset(leitura, 0, BUFFER_SIZE);  // Limpa o buffer antes de usar
        dtostrf(tensao, 1, 1, leitura); strcat(leitura, "-");
        dtostrf(corrente, 1, 2, buffer); strcat(leitura, buffer); strcat(leitura, "-");
        dtostrf(potencia, 1, 2, buffer); strcat(leitura, buffer); strcat(leitura, "-");
        dtostrf(energia, 1, 3, buffer); strcat(leitura, buffer); strcat(leitura, "-");
        dtostrf(frequencia, 1, 1, buffer); strcat(leitura, buffer); strcat(leitura, "-");
        dtostrf(fp, 1, 2, buffer); strcat(leitura, buffer);
        leitura[strlen(leitura)] = '\0';  // Finaliza a string de leitura
        pCharacteristic->setValue("");  // Limpa o valor anterior da característica BLE
        pCharacteristic->setValue(leitura);  // Define o novo valor
        pCharacteristic->notify();  // Notifica o dispositivo conectado via BLE
       }
```

```
// Função de setup do ESP32
void setup() {
  Serial.begin(115200);  // Inicializa a comunicação serial
  // Inicializa o dispositivo BLE (ESP32) com o nome "Medidor de Energia"
  BLEDevice::init("Medidor de Energia");
   // Cria o servidor BLE e define os callbacks para eventos de conexão/desconexão
  BLEServer *pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());
  // Cria o serviço BLE
  BLEService *pService = pServer->createService(SERVICE_UUID);
  // Cria uma característica de leitura/notificação para enviar dados via BLE
      pCharacteristic = pService->createCharacteristic(READ_UUID, BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_NOTIFY);
   pCharacteristic->addDescriptor(new BLE2902());  // Adiciona um descritor BLE para a característica
  // Cria uma característica de escrita para receber dados via BLE
        BLECharacteristic *pCharacteristic = pService->createCharacteristic(WRITE_UUID, BLECharacteristic::PROPERTY_WRITE);
  pCharacteristic->setCallbacks(new MyCallbacks());  // Define o callback para tratar dados recebidos via BLE
  pService->start();  // Inicia o serviço BLE
  pServer->getAdvertising()->start();  // Começa a anunciar o dispositivo BLE
  Serial.println("Aguardando a conexao de um dispositivo...");
}
// Função de loop, executada continuamente
void loop() {
  // Se o dispositivo BLE estiver conectado
  if (deviceConnected) {
    if (flag) {
      // Primeira vez conectado, imprime o endereço do PZEM
      Serial.println("Conectado com sucesso");
      Serial.println("Endereco PZEM: ");
```

```
            Serial.println(pzem.readAddress(), HEX);
            flag = false;  // Flag para evitar a repetição do endereço PZEM
           }
           leituraPZEM();  // Faz a leitura dos dados e envia via BLE
           delay(2000);  // Aguarda 2 segundos entre leituras
          }
          // Quando o dispositivo BLE se desconectar
          if (!deviceConnected && oldDeviceConnected) {
           delay(500);  // Aguarda um tempo para o servidor BLE estabilizar
           flag = true;  // Reseta a flag de conexão
            pServer->startAdvertising();   // Começa a anunciar o dispositivo BLE
novamente
           Serial.println("Aguardando a conexao de um dispositivo...");
           oldDeviceConnected = deviceConnected;
          }
          // Quando um novo dispositivo se conectar
          if (deviceConnected && !oldDeviceConnected) {
           oldDeviceConnected = deviceConnected;  // Atualiza o status de conexão
          }
         }
```

APÊNDICE B — Link para baixar o aplicativo móvel

O aplicativo desenvolvido pode ser acessado e copiado através do link: https://x.thunkable.com/projectPage/670e8326f590329d40d91060.

A versão para instalação em dispositivos Android está disponível para download em: https://github.com/kvfranco/medidor-de-energia.