UNIVERSIDADE ESTADUAL PAULISTA “JÚLIO DE MESQUITA FILHO”
FACULDADE DE ENGENHARIA
CÂMPUS DE ILHA SOLTEIRA

MURILO ROQUE DA SILVA

SISTEMA DE CONTROLE E PROTEÇÃO DE TOMADAS ELÉTRICAS COM ESP32 E MONITORAMENTO REMOTO VIA IOT PARA GALPÕES INDUSTRIAIS

Ilha Solteira
2025

MURILO ROQUE DA SILVA

SISTEMA DE CONTROLE E PROTEÇÃO DE TOMADAS ELÉTRICAS COM ESP32 E MONITORAMENTO REMOTO VIA IOT PARA GALPÕES INDUSTRIAIS

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia de Ilha Solteira – UNESP como parte dos requisitos para obtenção do grau de engenheiro eletricista.

Orientador
Prof. Dr. Jean Marcos de Souza Ribeiro

Ilha Solteira
2025

FICHA CATALOGRÁFICA
Desenvolvida pela Diretoria Técnica de Biblioteca e Documentação

S586s

Silva, Murilo Roque da.
Sistema de controle e proteção de tomadas elétricas com ESP32 e monitoramento remoto via IOT para galpões industriais / Murilo Roque da Silva. -- Ilha Solteira: [s.n.], 2025
53 f. : il.

Trabalho de conclusão de curso (Graduação em Engenharia Elétrica) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira, 2025

Orientador: Jean Marcos de Souza Ribeiro
Inclui bibliografia

1. NBR5410. 2. Automação industrial. 3. ESP32. 4. Internet das coisas. 5. Blynk.

Elaborada por Raiane da Silva Santos - CRB-8/9999

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de Ilha Solteira

ATA DE DEFESA DE TRABALHO DE GRADUAÇÃO

Aos dezesseis dias do mês de dezembro do ano de dois mil e vinte e cinco, o discente ***Murilo Roque da Silva,*** matriculado sob o nº 181051435, tendo como banca examinadora o seu orientador, *o Prof. Dr. Jean Marcos de Souza Ribeiro, o Dr. Lucas do Carmo Yamaguti e o Eng. Bruno Hiromoto Oikawa*, apresentou o Trabalho de Graduação intitulado **"Sistema de controle e proteção de tomadas elétricas com ESP32 e monitoramento remoto via IOT para galpões industriais"**, obtendo a nota 9,0 (NOVE) e conceito aprovado.

Prof. Dr. Jean Marcos de Souza Ribeiro
- orientador-

Murilo Roque da Silva
- discente-

Dr. Lucas do Carmo Yamaguti
- Membro da Banca -

Eng. Bruno Hiromoto Oikawa
- Membro da Banca -

AGRADECIMENTOS

Agradeço primeiramente a Deus, pelo dom da vida e pela força necessária para superar os desafios ao longo desta jornada acadêmica.

Ao meu orientador, Prof. Dr. Jean Marcos de Souza Ribeiro, pela orientação, paciência e pelos conhecimentos compartilhados, que foram fundamentais para a estruturação e conclusão deste trabalho.

À minha família, pelo apoio incondicional, pelo incentivo constante e pela compreensão nos momentos de ausência dedicados aos estudos. Vocês são a minha base.

RESUMO

O presente trabalho apresenta o desenvolvimento de um sistema embarcado para o controle, seccionamento e monitoramento remoto de tomadas em galpões industriais, utilizando o ESP32 integrado à Internet das Coisas (IoT). O objetivo principal da pesquisa foi implementar um sistema de intertravamento lógico e físico que, ao detectar o funcionamento de uma carga em uma tomada, corte automaticamente a alimentação das demais tomadas, prevenindo a simultaneidade de uso em conformidade com a NBR 5410. A metodologia adotada foi experimental e aplicada, envolvendo um protótipo com sensores de efeito Hall (ACS712), conversores analógico-digitais de 16 bits (ADS1115) e módulos relés. O *firmware* foi desenvolvido com arquitetura não bloqueante, incorporando cálculos de *True RMS* para medição das cargas com o modelo de máquina de estados finitos e, para a supervisão remota, utilizou-se a plataforma Blynk para telemetria de dados. Os resultados obtidos com o protótipo validaram a lógica de proteção e medição de dados, mesmo em cenários de falha de conectividade na internet. Conclui-se que a solução proposta oferece uma alternativa de baixo custo frente aos controladores lógicos programáveis, unindo a segurança da instalação elétrica com a modernidade da indústria 4.0, mesmo que o protótipo não tenha dispositivos utilizados normalmente na indústria.

Palavras-chave: NBR 5410; automação industrial; ESP32; internet das coisas; Blynk.

ABSTRACT

This work presents the development of an embedded system for the control, disconnection, and remote monitoring of sockets in industrial warehouses, using the ESP32 integrated with the Internet of Things (IoT). The main objective of the research was to implement a logical and physical interlocking system that, upon detecting the operation of a load in one socket, automatically disconnects the power supply to the others, preventing simultaneous use in compliance with NBR 5410. The methodology adopted was experimental and applied, involving a prototype with Hall effect sensors (ACS712), 16-bit analog-to-digital converters (ADS1115), and relay modules. The firmware was developed with a non-blocking architecture, incorporating True RMS calculations for load measurement within a finite state machine model; for remote supervision, the Blynk platform was used for data telemetry. The results obtained with the prototype validated the protection logic and data measurement, even in scenarios of internet connectivity failure. It is concluded that the proposed solution offers a low-cost alternative to Programmable Logic Controllers, combining electrical installation safety with the modernity of Industry 4.0, even though the prototype does not employ standard industrial-grade devices.

Keywords: NBR 5410; industrial automation; ESP32; internet of things; Blynk.

LISTA DE FIGURAS

Figura 1 - Pinagem da placa ESP32..... 15
Figura 2 - Visualização da IDE Arduino..... 16
Figura 3 - Wokwi..... 18
Figura 4 - Área de desenvolvimento na Wokwi..... 19
Figura 5 - Página incial do site da Blynk..... 20
Figura 6 - Diagrama de conexões na Wokwi..... 21
Figura 7 - Dashboard parte 1..... 25
Figura 8 - Dashboard parte 2..... 26
Figura 9 - Datastream parte 1..... 26
Figura 10 - Datastream parte 2..... 27
Figura 11 - Protótipo desenvolvido..... 28
Figura 12 - Fluxograma do sistema de controle e proteção..... 29
Figura 13- Verificação do intertravamento pelos multímetros..... 30
Figura 14- Verificação do intertravamento pelos LEDs dos relés..... 31
Figura 15 - Velocidade 2 do liquidificador..... 32
FIgura 16 - Velocidade 4 do liquidificador..... 33
Figura 17 - Velocidade 6 do liquidificador..... 34
Figura 18 - Monitor serial com alterações feitas pela nuvem..... 36
Figura 19 - Dashboard do Blynk..... 37
Figura 20- Terminal do sistema Blynk..... 37

Sumário

1. INTRODUÇÃO..... 10
1.1 Objetivo geral..... 12
1.2 Objetivos específicos..... 12
2. DESENVOLVIMENTO..... 13
2.1 Fundamentação teórica..... 13
2.1.1 *A segurança em instalações elétricas (NBR 5410)*..... 13
2.1.2 *Monitoramento de energia em corrente alternada*..... 14
2.1.3 *Microcontroladores e IoT*..... 15
2.1.4 *Ferramentas de software*..... 16
2.3 Materiais..... 20
2.4 Diagrama e mapeamento das conexões..... 21
2.5 Algoritmo desenvolvido no ESP32..... 23
2.6 Interfaces..... 24
2.7 Método experimental..... 27
3. RESULTADOS..... 30
3.1 Teste de intertravamento..... 30
3.2 Análise de precisão das medições..... 31
3.3 Análise de dados no Blynk..... 36
4. CONCLUSÃO..... 38
5. REFERÊNCIAS BIBLIOGRÁFICAS..... 39
APÊNDICE I - CÓDIGO DO ESP32..... 40

1. INTRODUÇÃO

Em ambientes industriais, as instalações elétricas são projetadas para atender a demandas de corrente específicas de um tipo de maquinário. No entanto, com a necessidade de flexibilização e otimização de recursos, torna-se um risco aos proprietários de galpões industriais que desejam utilizar uma mesma instalação elétrica para alimentar múltiplos equipamentos, sem a necessidade de refazer todo o projeto elétrico. Essa prática, embora economicamente vantajosa, pode levar a sobrecargas na rede elétrica, resultando em riscos de incêndio, danos aos equipamentos, interrupções na produção e, em casos extremos, acidentes graves (De Lima Caneppele *et al.)*.

Por conseguinte, a Norma Brasileira NBR 5410, que estabelece as condições para instalações elétricas de baixa tensão, destaca a importância de garantir a segurança e a funcionalidade das instalações: "as instalações elétricas devem ser projetadas e construídas de forma a garantir a segurança das pessoas e dos bens, a funcionalidade e a conservação de energia" (ABNT, 2004). Além disso, a norma estabelece que os circuitos devem ser dimensionados para suportar a corrente máxima prevista, evitando sobrecargas que possam comprometer a integridade do sistema.

Nesse contexto, surge a necessidade de desenvolver um sistema de controle e proteção de tomadas trifásicas que permita o compartilhamento seguro de uma instalação elétrica existente (Creeder, 2016). O sistema proposto neste projeto utiliza sensores de corrente para monitorar o consumo em tempo real e controlar relés para garantir que os equipamentos conectados não excedam a capacidade máxima da instalação. Dessa forma, evita-se a sobrecarga da rede, mantendo a segurança e a integridade do sistema elétrico, em conformidade com as diretrizes da NBR 5410.

Ademais, a norma também aborda a importância da proteção contra sobrecargas, estabelecendo que "os condutores devem ser protegidos contra sobrecargas por dispositivos de proteção adequados, que interrompam a corrente antes que ela atinja valores perigosos" (ABNT, 2004). O sistema proposto no presente projeto atende a essa exigência ao monitorar continuamente a corrente e desligar automaticamente as tomadas quando a corrente se aproxima do limite máximo da instalação.

Além da segurança, este projeto visa promover a eficiência energética, outro aspecto destacado pela NBR 5410, a qual recomenda que "as instalações elétricas devem ser projetadas e operadas de forma a minimizar o consumo de energia, sem comprometer a segurança e a funcionalidade" (ABNT, 2004). Ao permitir o compartilhamento seguro de uma instalação elétrica, o sistema evita a necessidade de novas instalações, reduzindo custos e o consumo de materiais.

Com o avanço da tecnologia, a Internet das Coisas (do inglês, *Internet of Things* - IoT) tem se tornado uma ferramenta essencial para o monitoramento e controle remoto de sistemas industriais. A IoT permite a integração de dispositivos físicos à internet, possibilitando a coleta e o compartilhamento de dados em tempo real, além do controle remoto de equipamentos. No contexto deste projeto, a incorporação da IoT permitirá que o sistema seja monitorado e controlado à distância (Kurniawan, 2019), oferecendo maior flexibilidade e praticidade para o usuário. Por exemplo, o proprietário de um galpão poderá verificar o consumo de corrente de cada tomada e o estado dos relés por meio de um aplicativo ou interface web, além de receber alertas em caso de sobrecarga ou falha no sistema.

A utilização da IoT no projeto também está condizente com as tendências atuais da Indústria 4.0 (Arana-Landin *et al*, 2023), que busca a integração de tecnologias digitais para otimizar processos industriais. Segundo estudos recentes (Arun *et al*., 2024), a IoT pode reduzir custos operacionais, aumentar a eficiência energética e melhorar a segurança (Carlos Alberto *et al.*, 2020) em ambientes. Ao incorporar essa tecnologia, o sistema proposto não apenas atende às exigências da NBR 5410, mas também se posiciona como uma solução moderna e inovadora para o gerenciamento de instalações elétricas.

Em suma, o sistema proposto não apenas atende às exigências da NBR 5410, mas também oferece uma solução prática, econômica e moderna para a modernização de instalações industriais, garantindo segurança, eficiência e flexibilidade operacional, com o adicional de monitoramento e controle remoto por meio da IoT.

1.1 Objetivo geral

O presente projeto possui como objetivo geral desenvolver um sistema de controle e proteção de tomadas trifásicas com monitoramento remoto via IoT, que permita o compartilhamento seguro de uma instalação elétrica existente em galpões industriais, evitando sobrecargas na rede.

1.2 Objetivos específicos

- Implementar um sistema de medição de corrente trifásica utilizando sensores ACS712 e microcontrolador ESP32;
- Desenvolver um algoritmo para controle automático de relés;
- Implementar uma lógica de intertravamento onde o uso de uma tomada bloqueia fisicamente o uso das demais;
- Integrar um *display* OLED para visualização em tempo real do consumo de corrente e estado das tomadas;
- Implementar funcionalidades de IoT para monitoramento e controle remoto do sistema;
- Realizar testes práticos em um modelo experimental para observar o funcionamento do algoritmo;
- Documentar o projeto, incluindo detalhes técnicos, resultados obtidos e possíveis aplicações práticas;

2. DESENVOLVIMENTO

Este capítulo detalha as etapas de implementação lógica e funcional do sistema proposto, para garantir um sistema eficiente e capaz de operar de forma autônoma com segurança, independentemente da disponibilidade de conexão à internet.

2.1 Fundamentação teórica

Nessa etapa, apresenta-se os conceitos que embasaram a estruturação do *firmware* desenvolvido.

2.1.1 *A segurança em instalações elétricas (NBR 5410)*

A segurança em instalações elétricas de baixa tensão no Brasil é regida pela norma ABNT NBR 5410, que estabelece as condições mínimas para instalações de baixa tensão (até 1000V em corrente alternada e 1500V em corrente contínua) para garantir a segurança de pessoas e animais, o funcionamento adequado da instalação e a conservação dos bens. Assim, um dos pilares fundamentais da NBR 5410 é a proteção contra choques elétricos e a segurança durante a operação e manutenção em instalações elétricas (Kurniawan, 2019).

O princípio do seccionamento automático da fonte de energia, ou seja, cortar a alimentação de toda ou uma parte determinada do circuito elétrico sempre que houver uma corrente superior ao valor nominal previsto ou quando houver risco de choque (Fernando *et al*, 2013). Segundo a NBR 5410 no item 6.3.7.2.7, “O seccionamento deve ser garantido por dispositivo que seccione todos os polos da alimentação correspondente” (ABNT, 2004, p. 140).

Ainda, a utilização dos módulos de relés independentes para cada fase, sendo também uma alternativa para os contatores, para permitir o corte total da alimentação em caso de uso não autorizado de algum equipamento no circuito. Diferentemente de uma proteção residencial que atua em apenas uma fase, em galpões industriais, a abertura ou fechamento de todas as fases é necessária para evitar o funcionamento bifásico de motores ou a energização acidental durante as manutenções.

2.1.2 *Monitoramento de energia em corrente alternada*

Em ambientes industriais, a presença de motores, inversores de frequência e reatores introduz componentes indutivos e capacitivos, além de distorções harmônicas (Boylestad, 2012) e para a medição correta da corrente elétrica nestes cenários não se deve utilizar a média aritmética simples, visto que o valor médio de uma onda senoidal pura em um período é zero (Boylestad, 2012). Portanto, utiliza-se o valor eficaz ou RMS (*Root Mean Square*). O valor RMS de uma corrente variável é equivalente ao valor de uma corrente contínua que dissiparia a mesma potência em um resistor. Assim, para uma função contínua *i(t) em um período T*, o valor RMS é dado por:

$$I_{RMS} = \sqrt{\frac{1}{T}\int_0^T i(t)^2 dt} \quad (1)$$

Em sistemas digitais, onde o sinal é amostrado discretamente, utiliza-se a técnica de *True RMS* (RMS verdadeiro). Segundo Alexander e Sadiku (2013), esta técnica consiste em elevar cada amostra instantânea lida pelo conversor analógico-digital ao quadrado, calcular a média desses valores ao longo de um número inteiro de ciclos da rede e extrair a raiz quadrada. Dessa forma, este método é superior a retificação da média, pois garante precisão mesmo na medição de cargas não lineares ou sinais com ruídos harmônicos.

Como o protótipo desenvolvido ao longo deste trabalho terá uma tensão de referência fixa já em valor RMS, sem a leitura da defasagem angular entre a tensão e a corrente (fator de potência), a grandeza calculada será a Potência Aparente (S), medida em Volt-Ampere (VA). A potência aparente representa a potência total fornecida pela fonte ao circuito, sendo o conjunto da potência ativa, que realiza trabalho, e a potência reativa, que mantém os campos eletromagnéticos (Boylestad, 2012), e é calculada como:

$$S = V_{RMS} * I_{RMS} \quad (2)$$

O monitoramento baseado na potência aparente S e a acumulação de energia em kVAh (quilo-volt-ampere-hora) fornecerá uma estimativa do consumo das cargas ligadas ao protótipo a fins de controle e segurança deles.

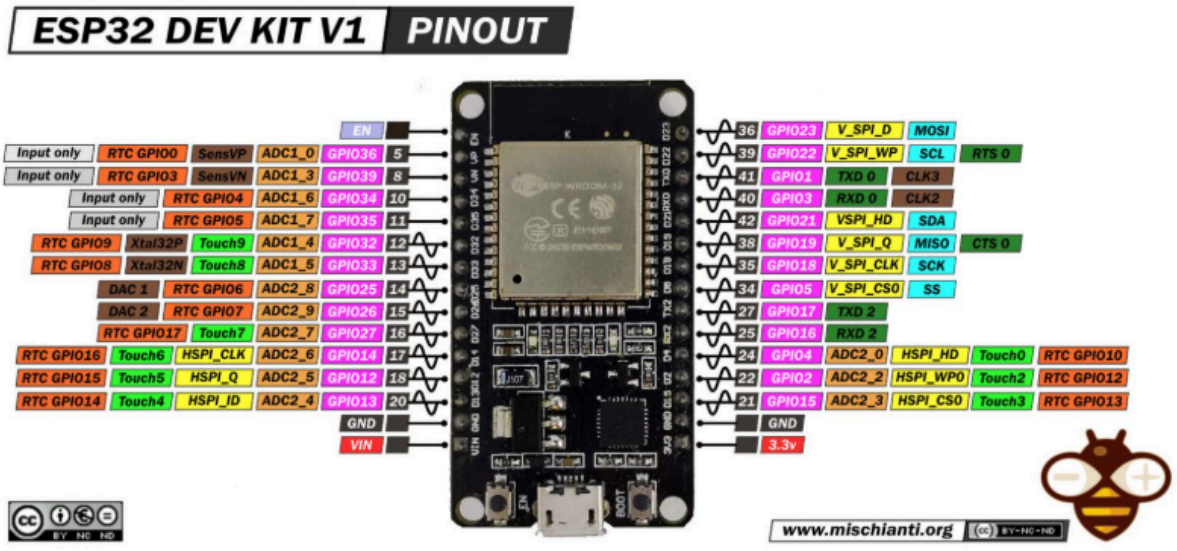
2.1.3 *Microcontroladores e IoT*

A evolução da eletrônica permitiu a transição de controle centralizado (Babiuch *et al*). e custosos para soluções distribuídas e acessíveis, impulsionada pelo IoT (De Freitas, 2020). Assim, a interconexão digital de dispositivos com a internet permite a coleta e troca de dados e possibilita o monitoramento remoto, que é um dos pilares da Indústria 4.0 (Arana-Landin *et al*, 2023).

O núcleo de processamento é o ESP32 (*Marek*, 2019), um *System on a Chip* (SoC) desenvolvido pela *Espressif Systems*, com uma arquitetura de 32 bits dual-core Xtensa LX6, operando a até 240 MHz, com conectividade ao Wi-Fi e Bluetooth integrada nativamente com pinos digitais GPIO de 16 bits (Espressif Systems, 2023). Essa capacidade de processamento é essencial para realizar os cálculos matemáticos de *True RMS* em tempo real sem bloquear as rotinas de comunicação de rede.

Na Figura 1 é ilustrada a pinagem completa de um microcontrolador ESP32 idêntico ao utilizado no projeto, que será utilizado como base para as conexões do protótipo desenvolvido neste trabalho (Upesy, 2025).

Figura 1 - Pinagem da placa ESP32

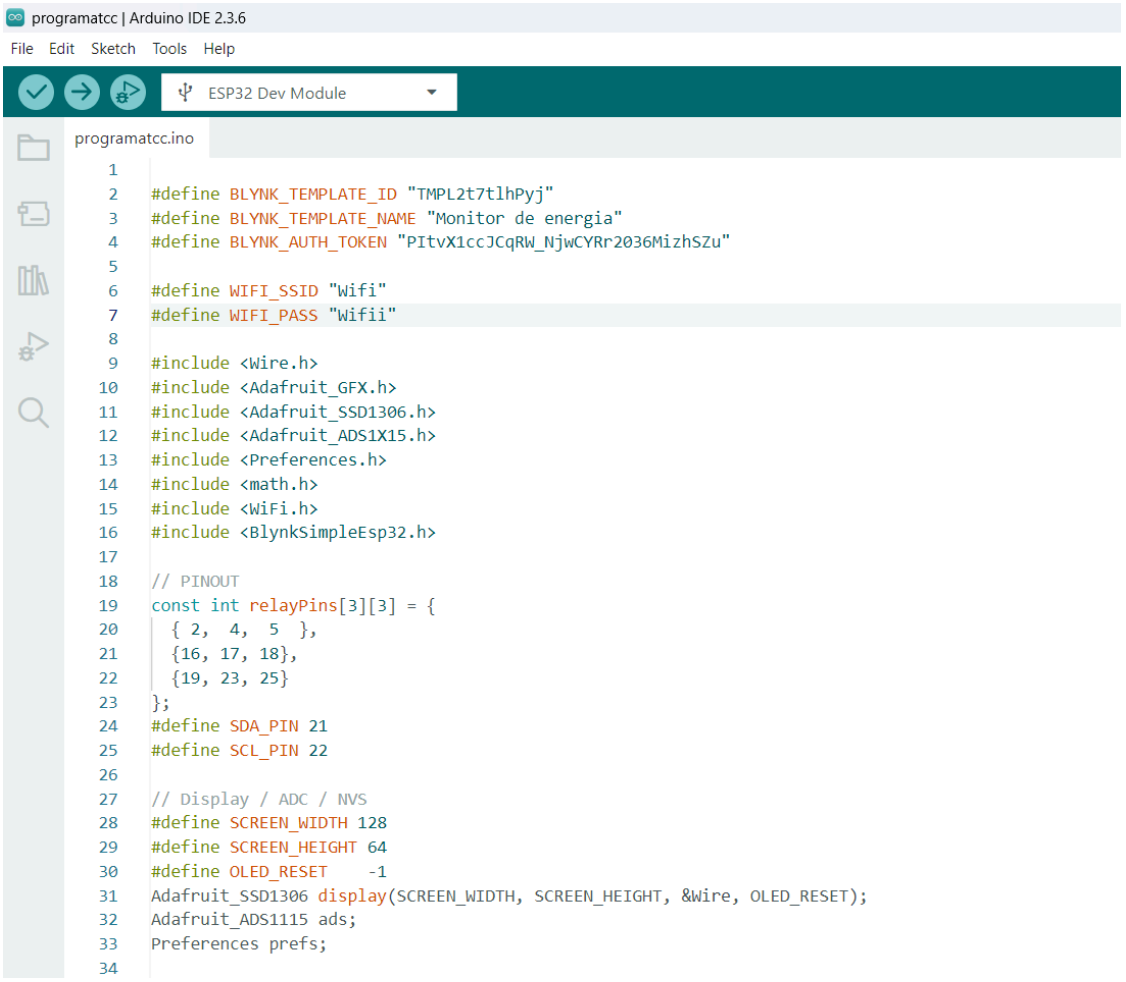


Fonte:Mischianti, 2025

2.1.4 *Ferramentas de software*

O Arduino *Integrated Development Environment* - Ambiente de Desenvolvimento Integrado (do inglês, IDE) é uma plataforma aberta de prototipação eletrônica composta pela placa controladora e o ambiente de desenvolvimento integrado. A programação para a ESP32 foi desenvolvida em C++ e é possível visualizar a interface na Figura 2.

Figura 2 - Visualização da IDE Arduino



Fonte: Elaborado pelo autor

A seguir, está uma descrição das bibliotecas utilizadas no código com sua devida finalidade:

a) Wire (Wire.h): Biblioteca I2C utilizada para comunicação com periféricos I2C;

b) Adafruit_GFX (Adafruit_GFX.h): Biblioteca de desenho 2D genérica que fornece primitivas gráficas (texto, linhas, retângulos) para displays. Usada como base pelo driver do SSD1306;

c) Adafruit_SSD1306 (Adafruit_SSD1306.h): Driver para displays OLED SSD1306 (I2C/SPI). Permite inicializar o display, limpar, desenhar texto e imagens e atualizar a tela;

d) Adafruit_ADS1X15 (Adafruit_ADS1X15.h): Driver para os conversores ADC ADS1015/ADS1115.;

e) Preferences (Preferences.h): API do ESP32 para armazenamento não volátil (NVS). Importante para armazenar energia acumulada e configurações do usuário;

f) math.h: Biblioteca padrão C para funções matemáticas (sqrt, fabs, pow, etc.). Utilizada para cálculos como sqrtf(3.0F) e sqrt() no cálculo RMS e potência aparente;

g) WiFi (WiFi.h): Biblioteca de controle Wi-Fi nativa do ESP32 . Usada para conectar o módulo à rede sem fio;

h) BlynkSimpleEsp32 (BlynkSimpleEsp32.h): Biblioteca cliente Blynk para ESP32. Fornece integração com a plataforma Blynk para telemetria, *widgets* virtuais e *callbacks*;

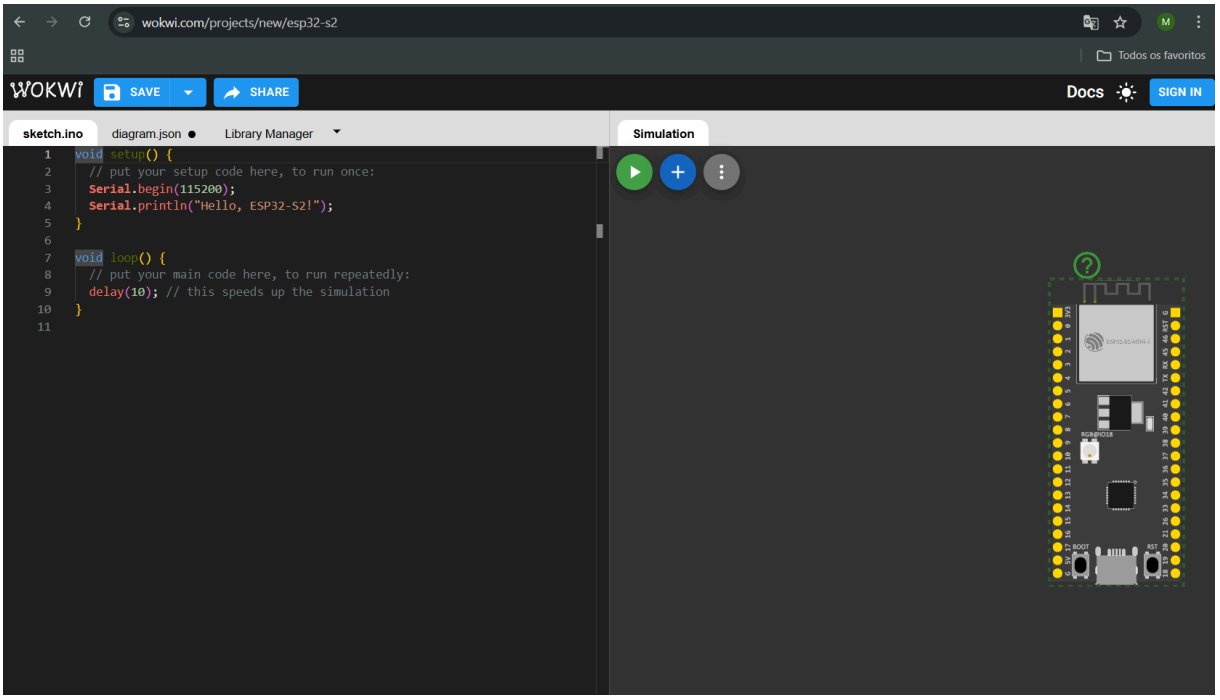
Para a simulação das conexões do sistema e validação da pinagem antes da montagem física, utilizou-se o simulador de eletrônica online Wokwi. O Wokwi é um ambiente online que possibilita o teste de diversas placas microcontroladores para testes e o acesso inicial a página do site é mostrado na Figura 3.

Figura 3 - Wokwi

Fonte: https://wokwi.com/

Ao selecionar a placa de desenvolvimento ESP32, é possível ver alguns projetos já desenvolvidos pela plataforma ou começar o projeto do zero. Na Figura 4, observa-se a área de desenvolvimento do usuário, sendo possível a programação em código, visível na parte esquerda da ilustração, ou a montagem do circuito com componentes disponibilizados pelo site.

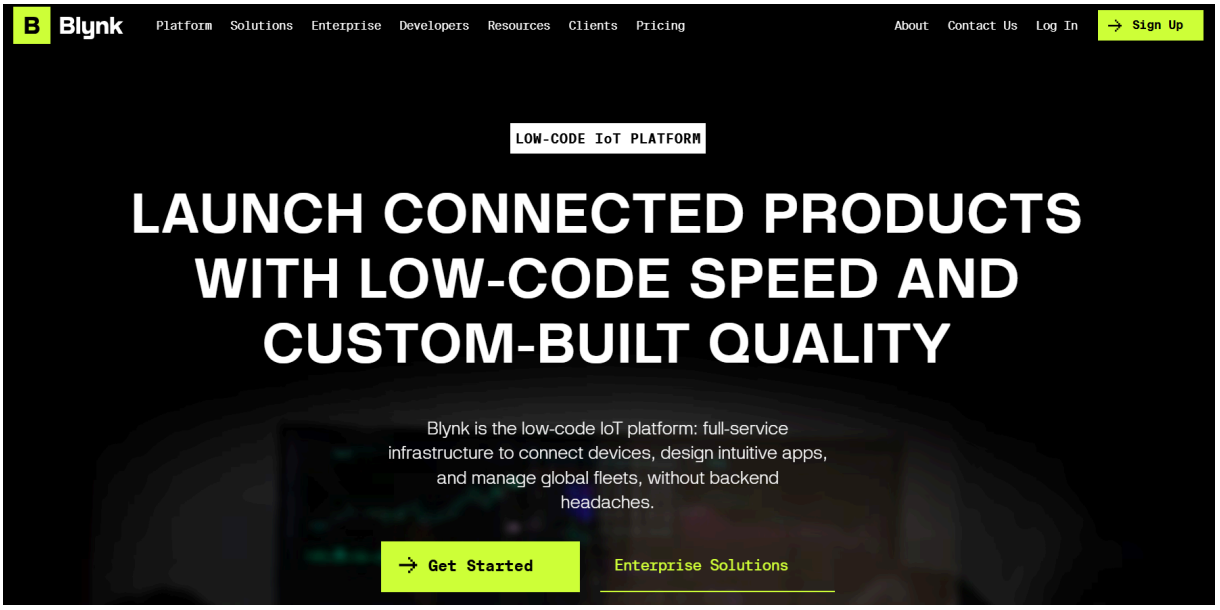
Figura 4 - Área de desenvolvimento na Wokwi



Fonte: Adaptado de Wokwi, 2025

Para a interface de telemetria e controle remoto, utilizou-se a plataforma Blynk de IoT para a comunicação entre a ESP32 e o usuário por meio da nuvem. A arquitetura da Blynk baseia-se em “Pinos Virtuais” para a troca de dados do tipo inteiros, flutuantes e strings entre o firmware e os widgets da aplicação e foi escolhida para afastar o firmware da complexidade dos protocolos de redes adjacentes. A Figura 5 mostra a página inicial do site da Blynk.

Figura 5 - Página incial do site da Blynk



Fonte: Adaptado de Blynk, 2025

Assim, a lógica crítica de intertravamento e corte de relés roda localmente na ESP32, garantindo a segurança do funcionamento mesmo em falha de conexão a internet, enquanto o monitoramento e configurações mais detalhadas ocorrem remotamente via nuvem

2.3 Materiais

Para a construção do protótipo funcional, foram utilizados componentes que são de baixo custo e disponíveis em coleções de aprendizagem de Arduino. Os principais componentes e *softwares* são:

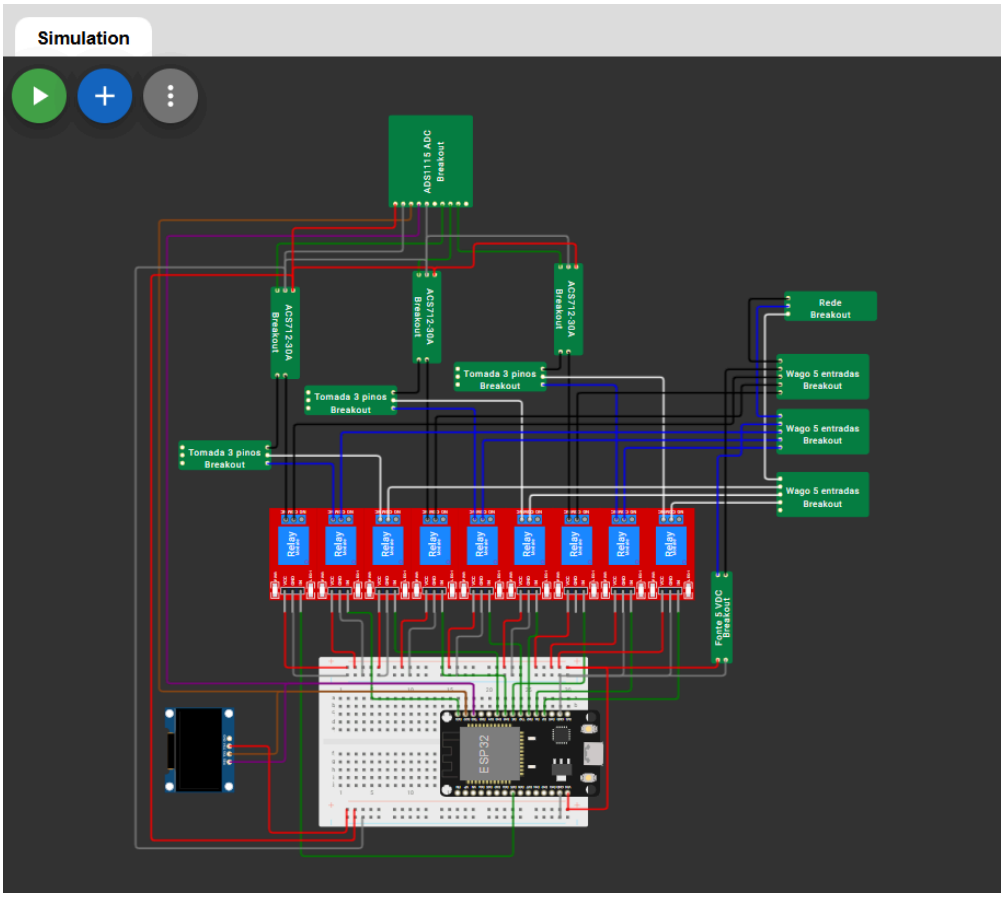
- Microcontrolador ESP32 (Modelo DevKit V1);
- Sensores de Corrente ACS712 de 30A;
- Módulo relé de 8 canais de 5V com optoacoplador;
- Módulo relé de 5V com optoacoplador;
- *Display* OLED 0,96” (SSD1306);
- Fonte de Chaveada 5V 3A;
- *Protoboard* de 400 pontos;
- *Jumpers* Macho-Macho e Macho-Fêmea;
- Bornes de emenda 5 vias (Wago);
- Tomada 20A quádrupla (Margirius);

- Conversor Analógico-Digital ADS1115;
- *Software* Arduino IDE;
- Multímetro digital hikari HM-1001;
- Alicate amperímetro CA/CC capacímetro 4000c/600V (Astroai);
- Cabo flexível de 2,5mm² de diversas cores;
- Chapa de madeira compensada;
- *Software* de simulação Wokwi;
- Arduino IDE;
- Liquidificador Philco 1200W;
- Plataforma IoT da Blynk.

2.4 Diagrama e mapeamento das conexões

Para garantir a integridade das conexões antes da montagem física, utilizou-se da plataforma Wokwi para montar o diagrama de conexões completo do sistema, conforme a Figura 6.

Figura 6 - Diagrama de conexões na Wokwi



Fonte: Adaptado de Wokwi, 2025

Observa-se na Figura 6 a seguinte distribuição de cores para os fios:

1) Fios vermelho: Conexão 5V;
2) Fio cinza: Conexão de aterramento (GND);
3) Fio verde: Conexão de GPIO;
4) Fios azul, preto e branco: Fases R, S e T;
5) Fios roxo e laranja: Conexão I2C.

Assim, conforme ilustrado na Figura 6, o ESP32 atua como mestre no barramento I2C para a comunicação com o display OLED e o módulo ADS1115. As saídas digitais do ESP32 são conectadas às entradas do módulo de relés, que simulam o controle individual das fases R, S e T para cada uma das três tomadas projetadas.

O Quadro 1 detalha a pinagem utilizada no ESP32, relacionando as portas de GPIOs com os periféricos do sistema que foram vistos na Figura 6.

Quadro 1 - Mapeamento de conexões dos GPIOs do ESP32

Periférico / Função	Pino do Módulo	GPIO ESP32	Descrição
Barramento I2C	SDA	GPIO 21	Dados Seriais (OLED e ADS1115)
Barramento I2C	SCL	GPIO 22	Clock Serial (OLED e ADS1115)
Relé - Tomada 1	IN1 (Fase R)	GPIO 2	Controle de Seccionamento
Relé - Tomada 1	IN2 (Fase S)	GPIO 4	Controle de Seccionamento
Relé - Tomada 1	IN3 (Fase T)	GPIO 5	Controle de Seccionamento
Relé - Tomada 2	IN4 (Fase R)	GPIO 16	Controle de Seccionamento
Relé - Tomada 2	IN5 (Fase S)	GPIO 17	Controle de

			Seccionamento
Relé - Tomada 2	IN6 (Fase T)	GPIO 18	Controle de Seccionamento
Relé - Tomada 3	IN7 (Fase R)	GPIO 19	Controle de Seccionamento
Relé - Tomada 3	IN8 (Fase S)	GPIO 23	Controle de Seccionamento
Relé - Tomada 3	IN9 (Fase T)	GPIO 25	Controle de Seccionamento
Sensor ACS712 (T1)	OUT	A0 (ADS1115)	Entrada Analógica via I2C
Sensor ACS712 (T2)	OUT	A1 (ADS1115)	Entrada Analógica via I2C
Sensor ACS712 (T3)	OUT	A2 (ADS1115)	Entrada Analógica via I2C

Fonte: Elaborado pelo autor

2.5 Algoritmo desenvolvido no ESP32

O firmware foi desenvolvido em linguagem C++ utilizando a IDE do Arduino com a utilização de algumas bibliotecas disponíveis para a ESP32. A estrutura do código baseia-se em um sistema não bloqueante, gerenciado por temporizadores de software que permite a leitura dos sensores, o acionamento dos relés e a comunicação Wi-Fi ocorram sem que uma tarefa impeça a execução da outra.

O algoritmo de leitura True RMS realiza a amostragem do sinal analógico proveniente do sensor ACS712 através do conversor ADS1115 configurado para uma taxa de 560 amostras por segundo. Assim, o *firmware* captura uma janela de 75 amostras em cerca de 87 milissegundos, abrangendo cerca de 5 ciclos completos da rede elétrica de 60 Hz.

O cálculo da corrente é discretizado como: Subtrai-se o offset (ponto zero) de cada leitura para obter a tensão instantânea; eleva-se cada valor de tensão ao quadrado e acumula-se ao somatório; calcula-se a média dos quadrados; extrai-se a

raiz quadrada da média para obter a tensão eficaz, que é convertida para Ampères por meio da sensibilidade do sensor ACS712 de 30A, que é de 66 mV/A.

Com a corrente em RMS obtida, o algoritmo calcula a potência aparente em VA multiplicando a corrente pela tensão nominal em RMS definida pelo usuário. A energia acumulada E, em kVAh, é obtida pela integração da potência no tempo e somando-se os incrementos a cada ciclo de medição

A lógica de intertravamento opera por meio de uma máquina de estados finitos, que garante o seccionamento físico das tomadas não utilizadas, alternando entre dois estados principais:

Estado ocioso (*IDLE*): É o estado inicial do sistema e de quando nenhuma carga está em uso. Neste estado, todos os relés permanecem fechados (energizados), permitindo que qualquer tomada seja utilizada.

Estado ativo (*ACTIVE*): Assim que uma corrente superior ao limiar de detecção (*threshold*) é detectada por uma tomada, o sistema altera-se para o estado ativo. Imediatamente, o algoritmo envia um sinal para abrir todos os relés das demais tomadas, que bloqueia a energização de cargas indesejadas.

Analogamente, o sistema permanece ativo monitorando a tomada em uso e informando a tela OLED, enquanto a corrente cai abaixo do limiar definido ou apresenta uma variação que indique o não uso da carga.

Para garantir a confiabilidade dos dados de consumo em um ambiente industrial sujeito a falhas de energia, utilizou-se a biblioteca *Preferences.h* para gravar dados na partição de memória não-volátil do ESP32. Assim, as variáveis de energia acumulada, tensão configurada e custo tarifário são salvas periodicamente para não perder dados de telemetria.

2.6 Interfaces

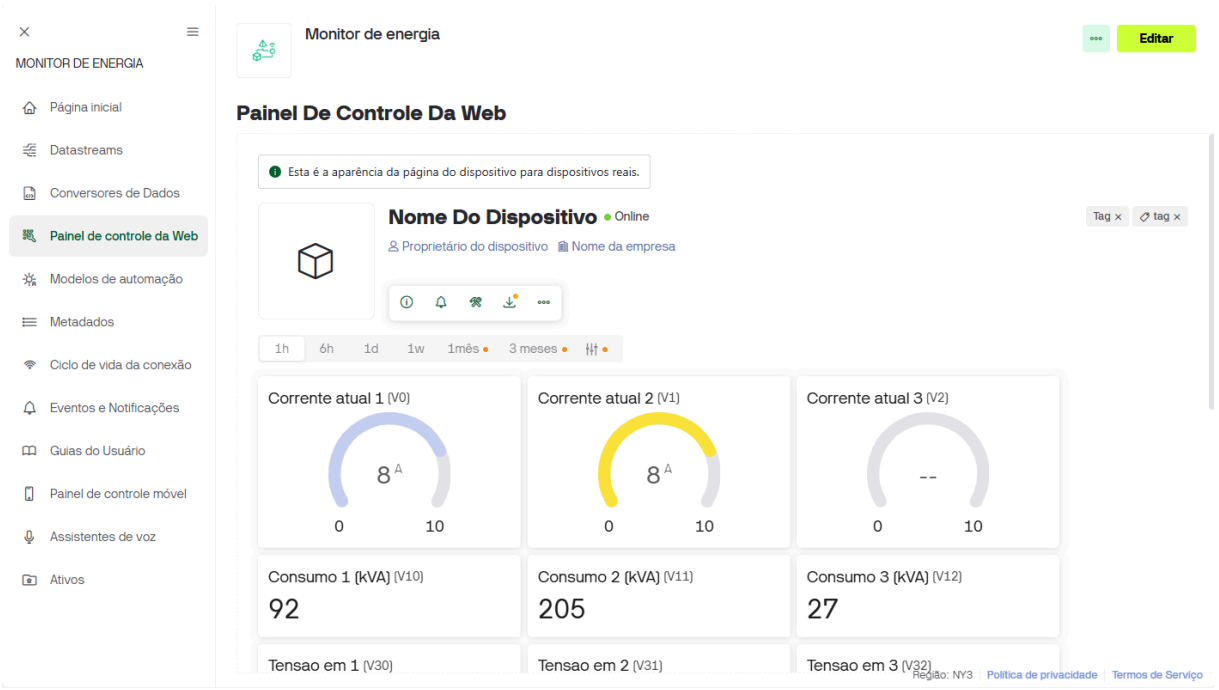
A interface remota pela plataforma Blynk é composta dos seguintes elementos:

1) Indicadores de correntes: Três *gauges* (medidores analógicos virtuais) que exibem a corrente instantânea de cada tomada;
2) Displays de valor: Caixas de texto que mostram a energia acumulada (kVAh);

3) Seletores de tensão: Menus que permitem ao usuário definir se a tomada está operando em 127V, 220V ou 380V, que permite ajustar o cálculo de potência no firmware;
4) Terminal de comandos: Uma interface de texto que recebe os logs do sistema e permite o envio de comandos de manutenção, como RESET ALL (para zerar os contadores) e SHOW (para ver status detalhado).
5) Gráfico de histórico de consumo: Gráfico com o registro de consumo que foi enviado para a plataforma Blynk;
6) Botões: Possui botões para zerar o consumo individualmente de cada tomada;
7) Seletor de entrada de número: É possível aumentar ou diminuir o valor do custo do kVAh;
8) Seletor de proteção: É possível desativar o sistema de segurança de intertravamento.

A seguir, nas Figuras 7 e 8, estão duas imagens com a visualização da *dashboard* pelo usuário.

Figura 7 - Dashboard parte 1



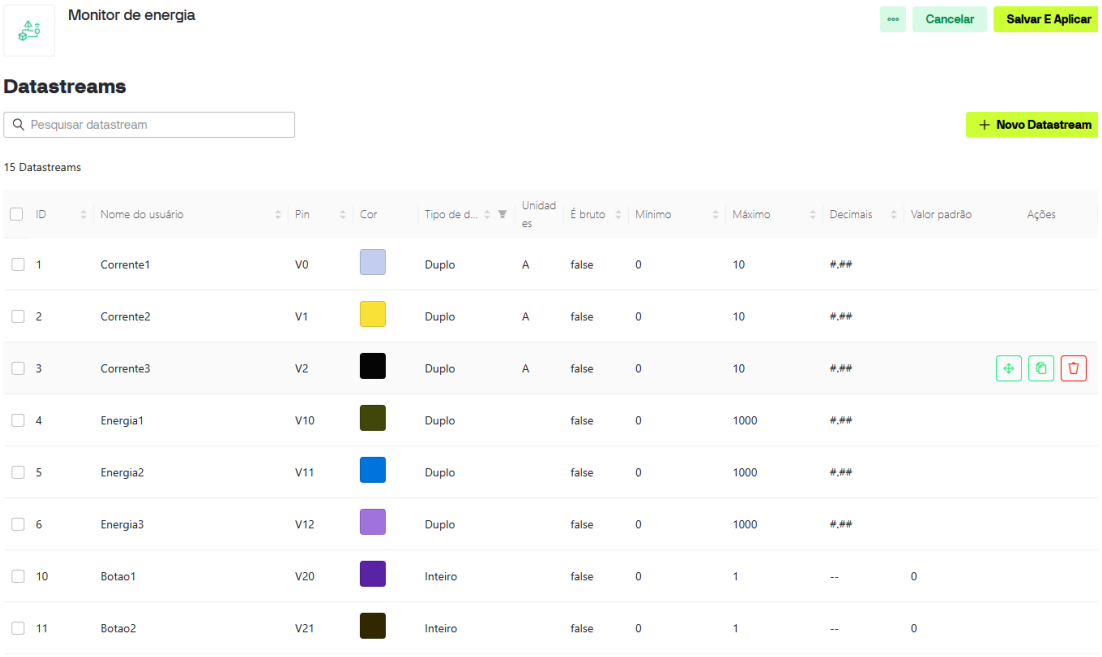
Fonte: Adaptado de Blynk, 2025

Figura 8 - Dashboard parte 2

Fonte: Adaptado de Blynk, 2025

As variáveis associadas aos pinos digitais fazem parte da *datastreams* configuradas conforme as Figuras 9 e 10.

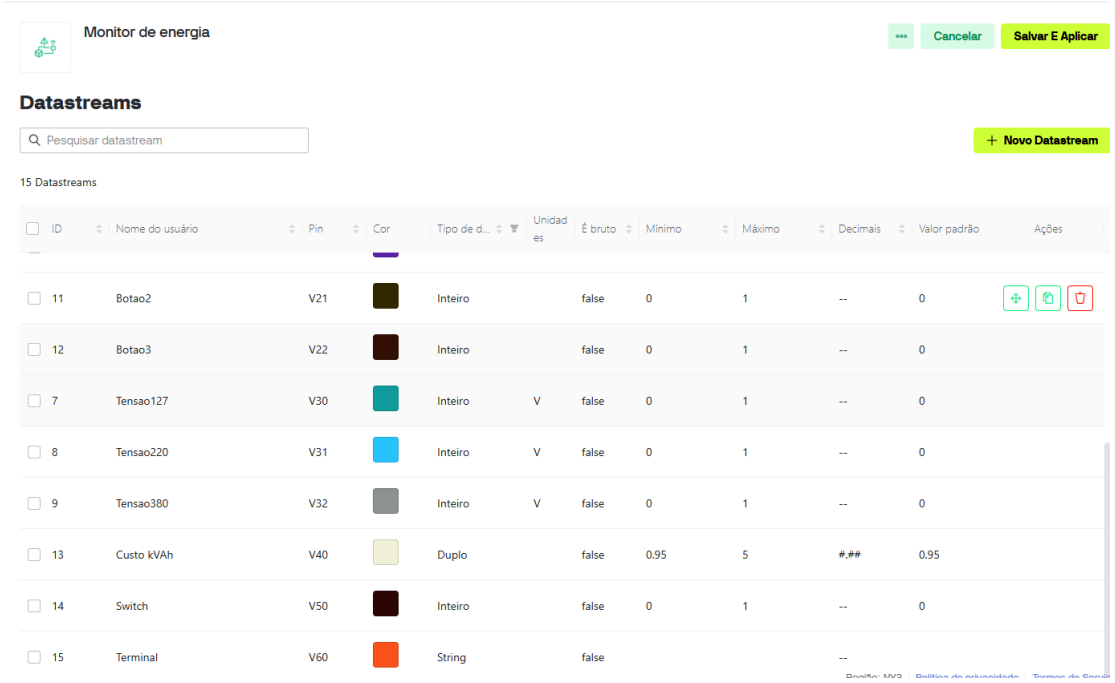
Figura 9 - Datastream parte 1



ID	Nome do usuário	Pin	Cor	Tipo de d...	Unidades	É bruto	Mínimo	Máximo	Decimais	Valor padrão	Ações
1	Corrente1	V0		Duplo	A	false	0	10	#.##		
2	Corrente2	V1		Duplo	A	false	0	10	#.##		
3	Corrente3	V2		Duplo	A	false	0	10	#.##		
4	Energia1	V10		Duplo		false	0	1000	#.##		
5	Energia2	V11		Duplo		false	0	1000	#.##		
6	Energia3	V12		Duplo		false	0	1000	#.##		
10	Botao1	V20		Inteiro		false	0	1	--	0	
11	Botao2	V21		Inteiro		false	0	1	--	0	

Fonte: Adaptado de Blynk, 2025

Figura 10 - Datastream parte 2



Fonte: Adaptado de Blynk, 2025

A interface local no display OLED fornece um *feedback* imediato ao usuário próximo ao equipamento, que apresenta as informações contextuais:

Modo ocioso: Exibe a mensagem “Aguardando uso…” e o resumo a energia acumulada nas tomadas

Modo ativo: Quando uma carga é detectada, o display altera-se para destacar a tomada em uso e exibe a corrente em tempo real.

A interface local é importante para garantir que o gestor do aplicativo e o operador tenham acesso às informações de segurança e consumo.

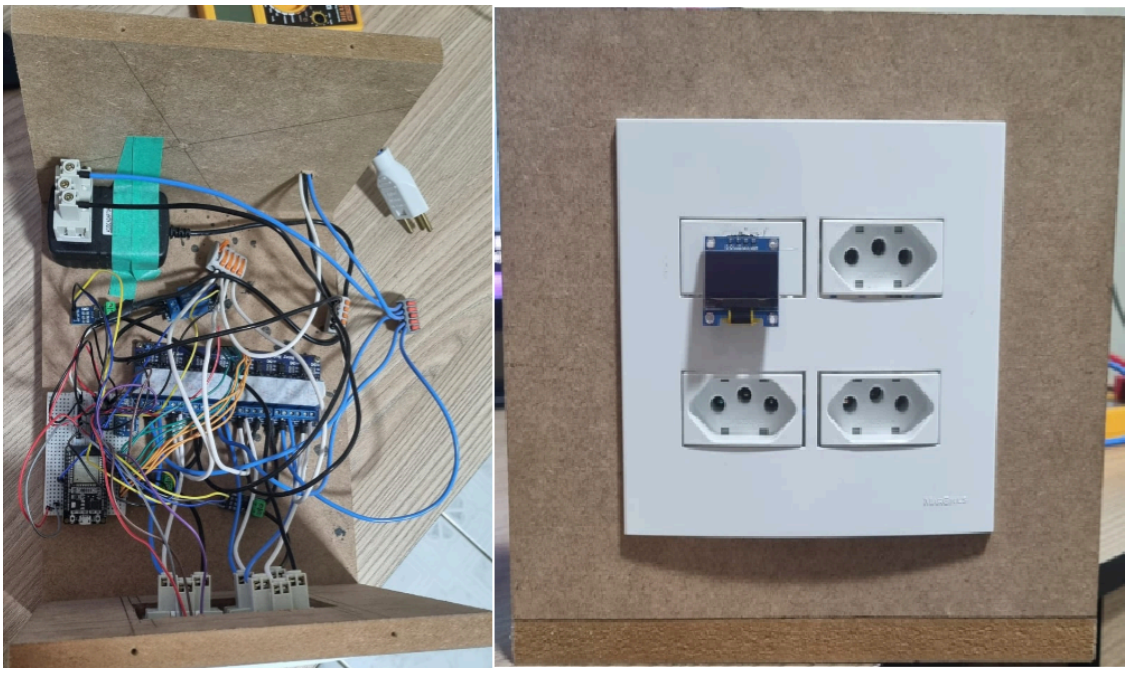
2.7 Método experimental

O procedimento experimental foi dividido em montagem, configuração e testes de validação com carga real.

Inicialmente, o circuito foi montado em *protoboard* seguindo o diagrama da Figura 6 e o mapeamento do Quadro 1,conforme mostrado na Figura 11. Embora o projeto vise o controle de cargas trifásicas industriais (utilizando três relés por tomada para corte das fases R, S e T), para fins de segurança e viabilidade em ambiente acadêmico, os testes foram realizados em uma rede monofásica

(127V/220V) residencial. A lógica de acionamento, entretanto, manteve-se inalterada, validando a lógica de controle industrial.

Figura 11 - Protótipo desenvolvido



Fonte: Elaborado pelo autor

Como carga de teste dinâmica, utilizou-se um equipamento residencial para desenvolver alguns testes. Este equipamento foi escolhido por representar uma carga indutiva (motor universal) com fator de potência variável e por permitir a modulação da corrente consumida através da seleção de velocidades, simulando diferentes estados de operação de uma máquina industrial.

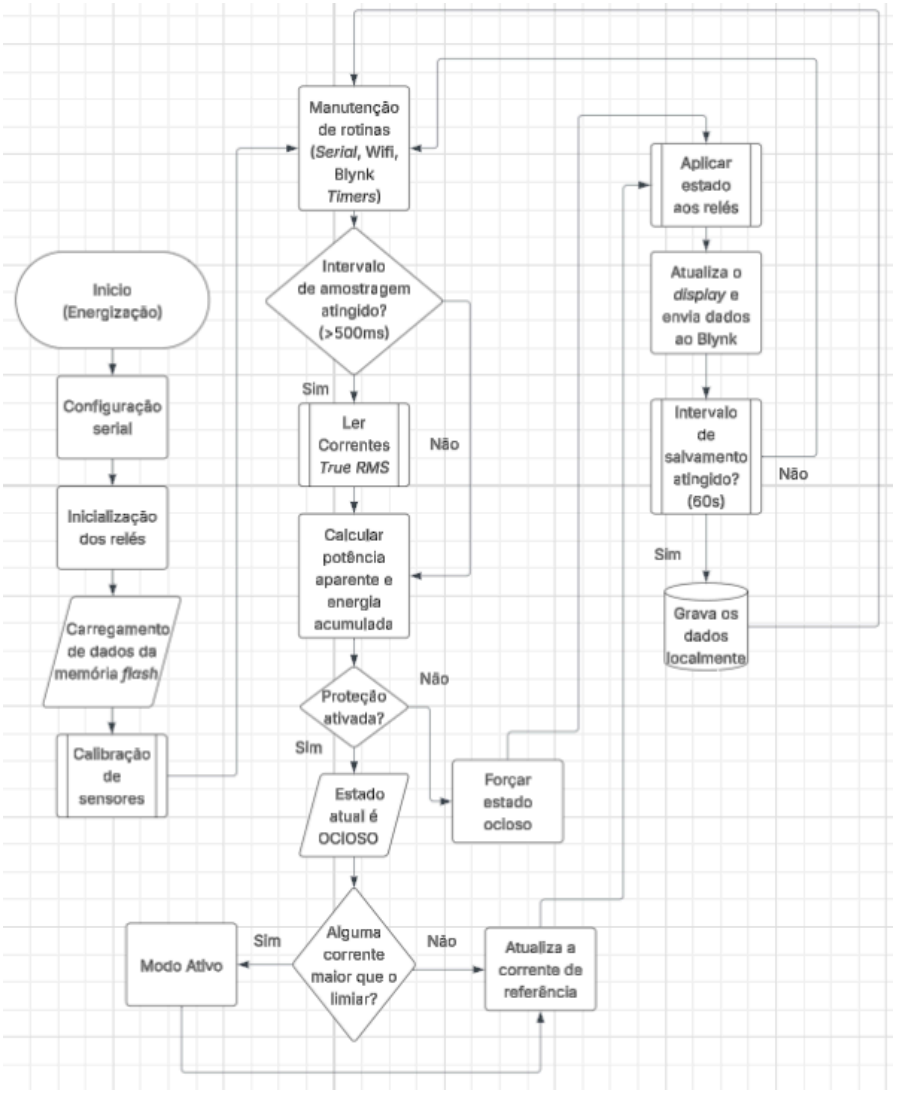
Os experimentos consistiram em:

1. Teste de Intertravamento: Com o liquidificador em funcionamento na Tomada 1, tentou-se acionar cargas nas Tomadas 2 e 3, verificando a atuação física dos relés para impedir a energização simultânea, conforme a prioridade estabelecida no algoritmo de proteção.
2. Análise de precisão das medições: Medição comparativa entre os valores apresentados pelo protótipo (OLED/Blynk) e os valores aferidos por um multímetro/alicate amperímetro de referência, em diferentes velocidades do liquidificador.

3. Análise de dados no Blynk: Funcionamento correto da dashboard e o devido envio das informações para a nuvem.

O *software* desenvolvido verifica periodicamente se o intervalo de amostragem (500ms) foi atingido e quando positivo, realiza-se a leitura True RMS e o cálculo de potência. Em seguida, uma máquina de estados finitos determina se o sistema deve permanecer em modo OCIOSO (monitorando todas as tomadas) ou transitar para o modo ATIVO (bloqueando tomadas adjacentes), baseando-se na detecção de corrente da carga. Por fim, o estado resultante é aplicado fisicamente aos relés e as interfaces (OLED e Blynk) são atualizadas antes que o ciclo se reinicie. O fluxograma está disponível na Figura 12.

Figura 12 - Fluxograma do sistema de controle e proteção



Fonte: Elaborado pelo autor

3. RESULTADOS

Para simular um ambiente de operação com carga variável e indutiva, utilizou o liquidificador doméstico da marca Philco com potência nominal de 1200W e 12 seleções de velocidade. O experimento foi montado utilizando a rede elétrica monofásica de 127V, o multímetro Hakari HM-1001 e do alicate amperímetro Astroai CM4KOR devidamente conectados ao protótipo como referência dos valores reais de corrente RMS. O código desenvolvido no ESP32 está disponível no Apéndice I - Código do ESP32.

3.1 Teste de intertravamento

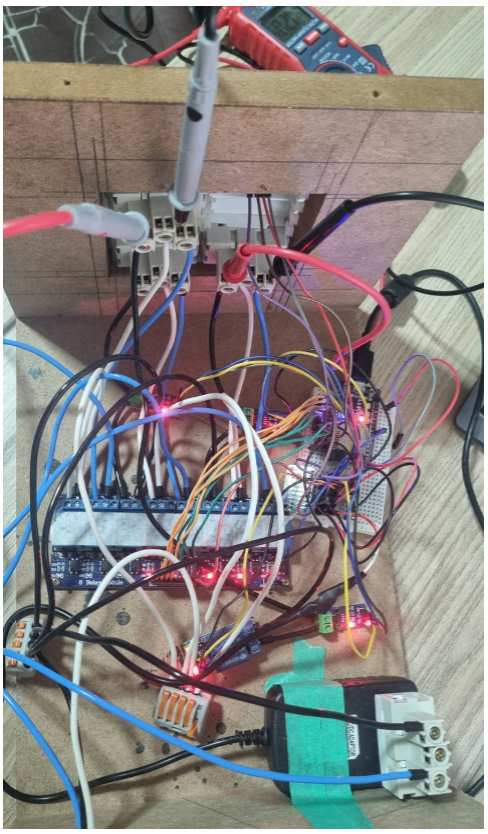
O objetivo primário deste trabalho foi garantir o seccionamento automático das tomadas não autorizadas e esse teste verificou a resposta da máquina ao detectar o uso de uma tomada. Ao acionar a carga conectada a Tomada 1, o sistema detectou a corrente superior ao *threshold* de 0,25 A e acionou os relés de bloqueio.

O intertravamento físico foi confirmado pelo barulho do acionamento dos relés e visualmente pelos LEDs indicadores dos módulos dos relés. Então, comprovou-se com o multímetro que não havia tensão nas demais tomadas, ou seja, os demais circuitos estavam fisicamente abertos, conforme as Figuras 13 e 14.

Figura 13- Verificação do intertravamento pelos multímetros

Fonte:Elaborado pelo autor

Figura 14- Verificação do intertravamento pelos LEDs dos relés



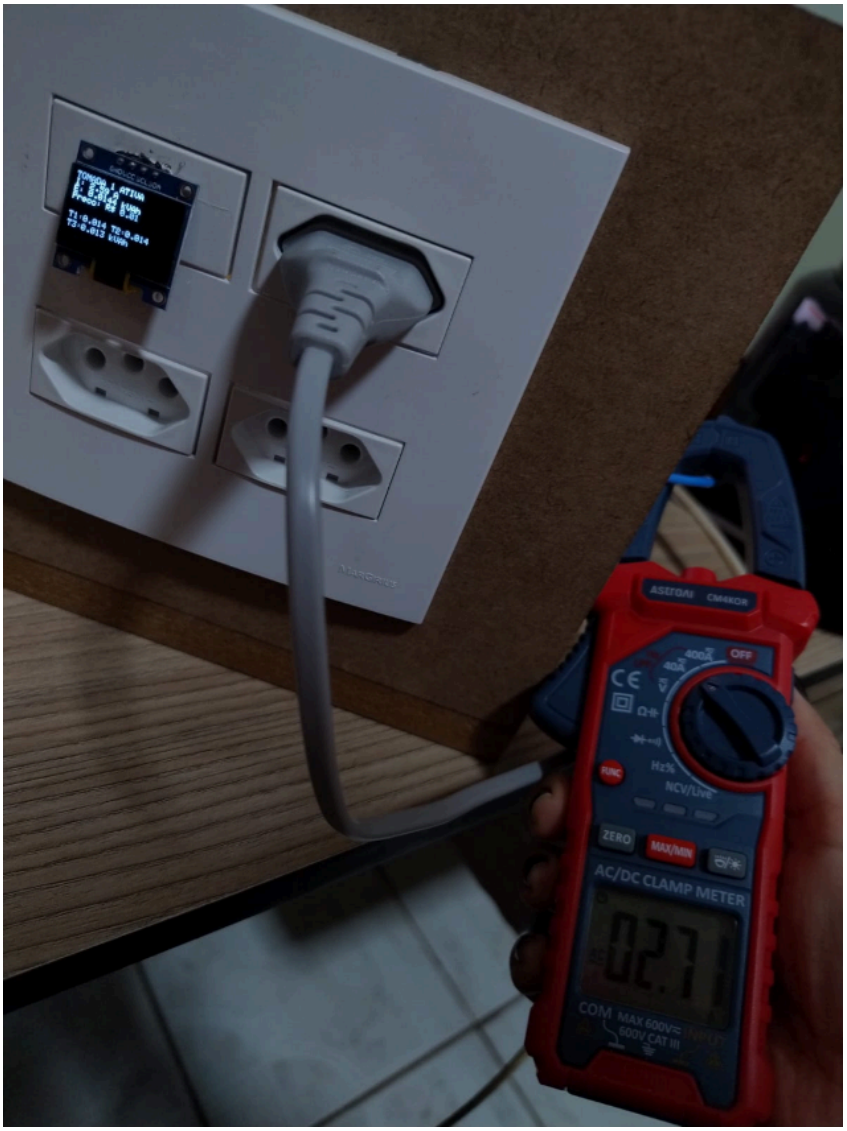
Fonte: Elaborado pelo autor

Após o desligamento do liquidificador, o sistema continuou ativo ainda por alguns segundos até retornar ao estado ocioso, religando todas as tomadas para um novo ciclo de uso. Este comportamento comprovou a eficácia lógica de proteção proposta para evitar a sobrecarga por simultaneidade

3.2 Análise de precisão das medições

A precisão do monitoramento de energia foi avaliada comparando-se os valores de corrente fornecidos pelo display OLED e os valores aferidos pelos instrumentos de referência após a estabilização da corrente de acordo com diversas referências de velocidade do liquidificador conforme a demonstração das medições disponíveis nas Figuras de 15 a 17.

Figura 15 - Velocidade 2 do liquidificador



Fonte: Elaborado pelo autor

Na Figura 15, lê-se os valores de 2,71 A no amperímetro e 2,59 A no *display* protótipo.

FIgura 16 - Velocidade 4 do liquidificador

Fonte: Elaborado pelo autor

Na Figura 16, lê-se os valores de 4,25 A no amperímetro e de 4,29 A no *display* protótipo.

Figura 17 - Velocidade 6 do liquidificador

Fonte: Elaborado pelo autor

Na Figura 17, lê-se os valores de 5,02 A no amperímetro e de 4,78 A no protótipo.

Além das amostragens que estão nas figuras anteriores, montou-se as Tabelas de 1 a 3, com 10 amostras em cada uma, para verificar o erro absoluto e o erro relativo do sistema em regime permanente.

Tabela 1 - Velocidade 2 do liquidificador

Baixa (V2)				
Carga	**Corrente de Referência (A)**	**Corrente Medida - Protótipo (A)**	**Erro Absoluto (A)**	**Erro Relativo (%)**
1	2,710	2,590	0,120	4,43%
2	2,690	2,570	0,120	4,46%
3	2,760	2,600	0,160	5,80%
4	2,740	2,600	0,140	5,11%
5	2,650	2,560	0,090	3,40%
6	2,700	2,600	0,100	3,70%
7	2,630	2,520	0,110	4,18%
8	2,700	2,570	0,130	4,81%
9	2,680	2,550	0,130	4,85%
10	2,710	2,580	0,130	4,80%
Média (10 amostras)	**2,697**	**2,574**	**0,123**	**4,55%**

Fonte:Elaborado pelo autor

Tabela 2 - Velocidade 4 do liquidificador

Média (V4)				
Carga	**Corrente de Referência (A)**	**Corrente Medida - Protótipo (A)**	**Erro Absoluto (A)**	**Erro Relativo (%)**
1	4,250	4,290	0,040	0,94%
2	4,200	4,260	0,060	1,43%
3	4,300	4,340	0,040	0,93%
4	4,220	4,280	0,060	1,42%
5	4,270	4,300	0,030	0,70%
6	4,190	4,220	0,030	0,72%
7	4,240	4,270	0,030	0,71%
8	4,260	4,270	0,010	0,23%
9	4,200	4,240	0,040	0,95%
10	4,280	4,300	0,020	0,47%
Média (10 amostras)	**4,241**	**4,277**	**0,036**	**0,85%**

Fonte: Elaborado pelo autor

Tabela 3 - Velocidade 6 do liquidificador

Alta (V6)				
Carga	**Corrente de Referência (A)**	**Corrente Medida - Protótipo (A)**	**Erro Absoluto (A)**	**Erro Relativo (%)**
1	5,020	4,780	0,240	4,78%
2	5,000	4,750	0,250	5,00%
3	5,100	4,800	0,300	5,88%
4	4,900	4,750	0,150	3,06%
5	5,030	4,780	0,250	4,97%
6	5,080	4,820	0,260	5,12%
7	5,000	4,750	0,250	5,00%
8	5,010	4,760	0,250	4,99%
9	5,000	4,720	0,280	5,60%
10	5,020	4,770	0,250	4,98%
Média (10 amostras)	**5,016**	**4,768**	**0,248**	**4,94%**

Fonte: Elaborado pelo autor

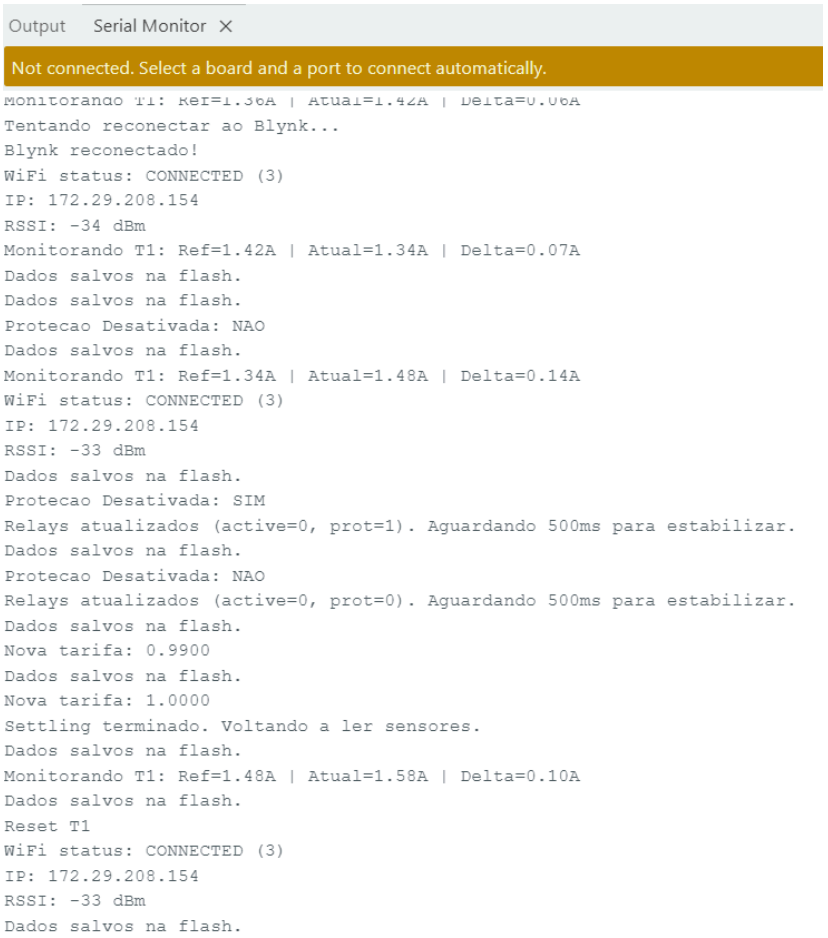
Analisando os dados, observa-se que o erro relativo ficou na faixa de 4,5% a 5% para as velocidades 2 e 4, enquanto que para a velocidade 4 ficou menor que 1%. Além das imprecisões que um sensor de corrente de baixo custo tem e a taxa de

atualização de dados entre o amperímetro e o protótipo serem diferentes, nota-se que o liquidificador ficou em um regime mais estável na velocidade 4, ficando com barulho e som notavelmente mais baixo do que nas outras condições, o que acarretou em um erro menor.

3.3 Análise de dados no Blynk

Durante os experimentos, testou-se os comandos disponibilizados no blynk para observar a sua atuação que foram observados no monitor serial do Arduino IDE conforme a Figura 18.

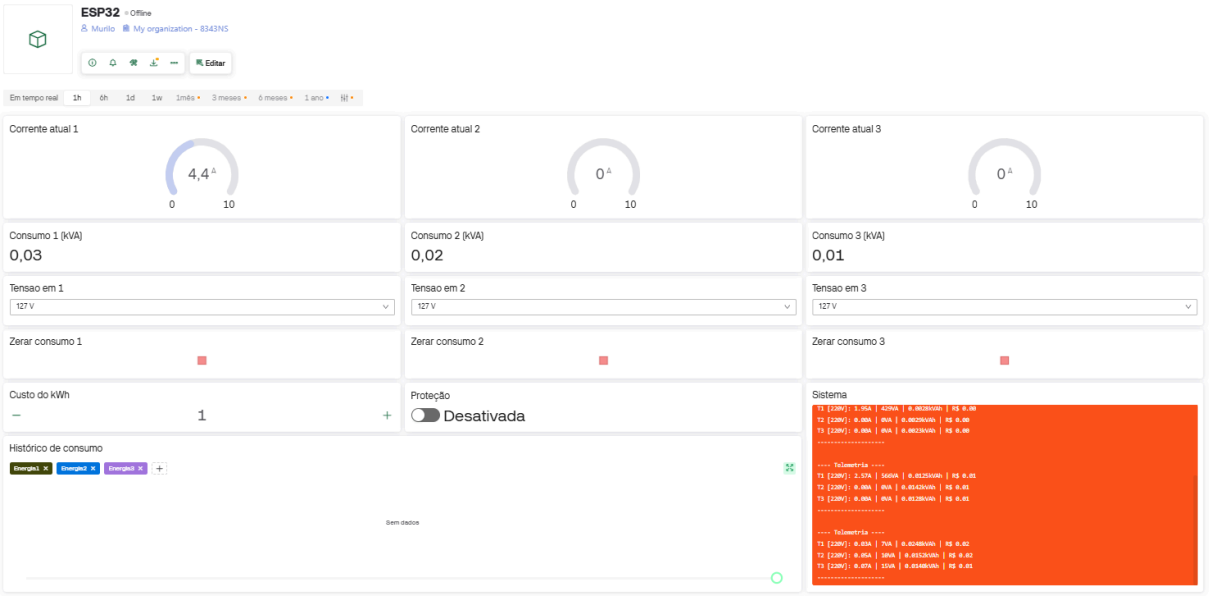
Figura 18 - Monitor serial com alterações feitas pela nuvem



Fonte: Elaborado pelo autor

Então, após os testes e comandos, a dashboard do Blynk é mostrada na Figura 19.

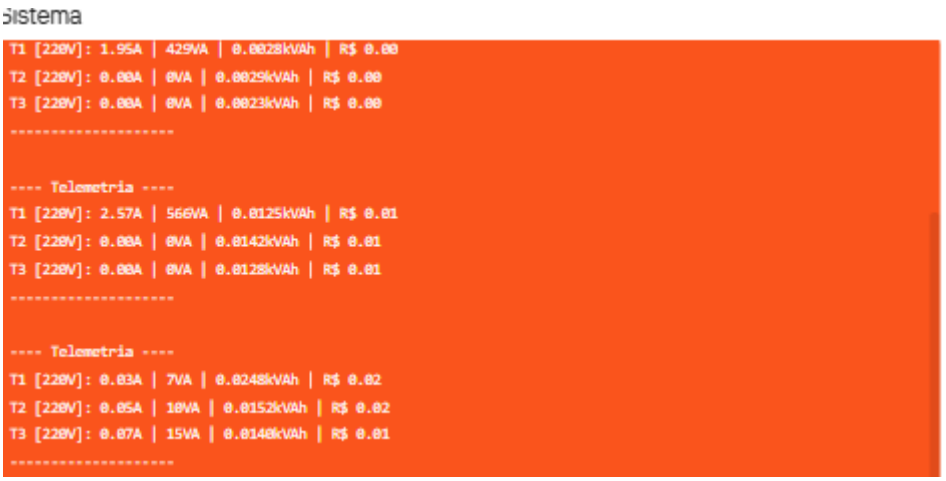
Figura 19 - Dashboard do Blynk



Fonte: Elaborado pelo autor

Como é possível observar na Figura 18, a ESP32 conectou-se ao Blynk, salvou informações na *flash* ao comando de zerar consumo, ativou e desativou a proteção e, por fim, salvou aumentou o valor da tarifa. Na Figura 20, é possível observar o terminal da Figura 19 com o devido zoom.

Figura 20- Terminal do sistema Blynk



Fonte: Elaborado pelo autor

4. CONCLUSÃO

O trabalho dedicou-se ao desenvolvimento e validação de um sistema embarcado baseado no microcontrolador ESP32 para o controle, seccionamento e monitoramento remoto de tomadas em ambiente industrial. A motivação central foi a mitigação de sobrecargas causadas pelo uso simultâneo de equipamentos de alta potência em galpões industriais de acordo com os princípios da norma NBR 5410 e a indústria 4.0.

A utilização do Arduino IDE como plataforma de código aberto e dos componentes de baixo custo para a montagem de um protótipo de teste permitiram a comprovação da lógica de operação de um ambiente industrial de forma segura e constante ao longo dos testes.

Os resultados obtidos afirmam que o objetivo geral foi atingido, uma vez que o protótipo funcional foi capaz de gerenciar o fornecimento de energia de forma autônoma, garantindo que apenas uma carga opere por vez por meio de um intertravamento físico. As medições de energia realizadas com um liquidificador residencial obtiveram um erro de até 5% do protótipo em relação a medições com multímetros dadas como referencial.

A integração com a plataforma da Blynk foi capaz de fornecer uma interface de gestão remota de fácil uso. A flexibilidade da configuração da tensão nominal e o custo tarifário por meio da interface local de uma tela OLED e da Blynk permite a sua adaptação a diferentes redes elétricas sem necessidade de programação complexa do usuário.

Ressalta-se que, embora os resultados sejam promissores, os equipamentos do protótipo não refletem fielmente os dispositivos corretos para uso em ambiente industrial. O sensor de efeito Hall ACS712 foi suscetível a ruídos magnéticos, exigindo um tratamento via *software* média de valores e *threshold* para evitar a atuação incorreta dos relés. Outras melhorias possíveis ao projeto seriam relacionadas ao uso de sensores de tensão e cálculos de fatores de potência.

Portanto, o sistema desenvolvido apresenta-se como uma solução de baixo custo em relação aos CLPs tradicionais. A união do seccionamento a relé com o processamento digital do ESP32 e a plataforma de IoT da Blynk foram uma alternativa eficaz para a segurança da instalação elétrica, demonstrando o potencial de inovação das soluções da indústria 4.0.

5. REFERÊNCIAS BIBLIOGRÁFICAS

ALEXANDER, C. K.; SADIKU, M. N. O. **Fundamentos de circuitos elétricos**. 5. ed. Porto Alegre: AMGH, 2013.

ARANA-LANDÍN, G.; LANDETA-MANZANO, B.; HERAS-SAIZARBITORIA, I.; BERASATEGI-VAQUERO, L. The contribution of lean management—Industry 4.0 technologies to improving energy efficiency. **Energies**, [S. l.], v. 16, n. 5, p. 2124, 2023. Disponível em: https://www.mdpi.com/1996-1073/16/5/2124.

ARUN, M.; RAMASUBRAMANIAN, S.; PRADEEP, R. J.; KUMAR, S. S.; PRABHAKARAN, S. Internet of things and deep learning-enhanced monitoring for energy efficiency in older buildings. **Case Studies in Thermal Engineering**, [S. l.], v. 61, p. 104867, 2024. Disponível em: https://www.sciencedirect.com/science/article/pii/S2214157X24008980.

BABIUCH, M.; FOLTÝNEK, P.; SMUTNÝ, P. Using the ESP32 microcontroller for data processing. *In*: INTERNATIONAL CARPATHIAN CONTROL CONFERENCE (ICCC), 20., 2019, Krakow. **Proceedings [...]**. Piscataway: IEEE, 2019. p. 1-6.

BLYNK. **Launch connected products with low-code speed and custom-built quality**. c2025. Disponível em: https://www.blynk.io/. Acesso em: 1 jul. 2025.

BOYLESTAD, R. L. **Introdução à análise de circuitos**. 12. ed. São Paulo: Pearson Education do Brasil, 2012.

CREDER, H. **Instalações elétricas**. 16. ed. Rio de Janeiro: LTC, 2016.

DE FREITAS, C. A.; BARBOSA, A. F.; FREITAS, A. A evolução da segurança no trabalho aplicada na manutenção industrial 4.0. **REMIPE-Revista de Micro e Pequenas Empresas e Empreendedorismo da Fatec Osasco**, Osasco, v. 6, n. 2, p. 229-251, 2020.

DE LIMA CANEPPELE, F.; CANEPPELE, C. E.; GABRIEL FILHO, L. R. A.; SERAPHIM, O. J. **Análise da incidência de mortes por choques elétricos notificados no SUS no período 2009-2013**. 2013. Tese (Doutorado em Agronomia) – Faculdade de Ciências Agronômicas, Universidade Estadual Paulista, Botucatu, 2013.

ESPRESSIF SYSTEMS. **ESP32 Series Datasheet**: versão 4.0. Xangai: Espressif Systems, 2023. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. Acesso em: 1 jul. 2025.

KURNIAWAN, A. **Internet of Things Projects with ESP32**: build exciting and powerful IoT projects using the all-new Espressif ESP32. [S. l.]: Packt Publishing Ltd, 2019.

MISCHIANTI, R. **DOIT ESP32 DEV KIT v1**: high resolution pinout and specs. 2025. Disponível em: https://mischianti.org/doit-esp32-dev-kit-v1-high-resolution-pinout-and-specs/. Acesso em: 1 jul. 2025.

UPESY. **ESP32 Pinout Reference**: which GPIO pins should you use? 2025. Disponível em: https://www.upesy.com. Acesso em: 1 jul. 2025.

APÊNDICE I - CÓDIGO DO ESP32

```
#define BLYNK_TEMPLATE_ID "TMPL2t7tlhPyj"
#define BLYNK_TEMPLATE_NAME "Monitor de energia"
#define BLYNK_AUTH_TOKEN "PItvX1ccJCqRW_NjwCYRr2036MizhSZu"

#define WIFI_SSID "Wifi"
#define WIFI_PASS "Wifii"

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_ADS1X15.h>
#include <Preferences.h>
#include <math.h>
#include <WiFi.h>
#include <BlynkSimpleEsp32.h>

// PINOUT
const int relayPins[3][3] = {
  { 2,  4,  5  },
  {16, 17, 18},
  {19, 23, 25}
};
#define SDA_PIN 21
#define SCL_PIN 22

// Display / ADC / NVS
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
Adafruit_ADS1115 ads;
Preferences prefs;

// Blynk
BlynkTimer blynkTimer;
WidgetTerminal terminal(V60);
const int VP_CURR[3]   = {V0, V1, V2};
const int VP_ENERGY[3] = {V10, V11, V12};
const int VP_VOLT[3]   = {V30, V31, V32};
const int VP_COST      = V40;
const int VP_PROT      = V50;
const int VP_TERM      = V60;
```

```
// Medição
const float LSB = 0.0001875F;
const float sensitivity = 0.066F;
float thresholdA = 0.25F;
int calibSamples = 1000;
int recalSamples = 200;
const int RMS_SAMPLES = 75;

// Timers
unsigned long sampleIntervalMs = 500;
unsigned long lastSampleTime = 0;
unsigned long saveIntervalMs = 60000UL;
unsigned long lastSaveTime = 0;
unsigned long telemetryIntervalMs = 180000UL;
unsigned long blynkReconnectIntervalMs = 30000UL;
unsigned long lastBlynkTry = 0;

// Estado
enum State { IDLE, ACTIVE };
State state = IDLE;
int activeIndex = -1;
unsigned long lastRecal = 0;
unsigned long minRecalIntervalMs = 20000UL;
bool calibrating = false;
bool initialCalibrated = false;

unsigned long checkInterval = 10000UL;
unsigned long lastCheckTime = 0;
float refCurrent = 0.0F;

// Dados de medição
float sensorOffsetRaw[3] = {0.0F, 0.0F, 0.0F};
float currents[3] = {0.0F, 0.0F, 0.0F};

// Energia e configurações persistentes
float energy_kvah[3] = {0.0F, 0.0F, 0.0F};
int outletVoltage[3] = {127, 127, 127};
bool disable_protection = false;
float CUSTO_KVAH = 0.0F; // custo por kVAh (moeda por kVAh)

// WiFi debug
unsigned long wifiStatusInterval = 15000UL;
unsigned long lastWifiStatusPrint = 0;
```

```
// Evitar falsa ativação após mudanças de estado no relé
unsigned long settleDelayMs = 500UL; // tempo para aguardar após mudança nos relés
bool waitingForSettle = false;
unsigned long settleStart = 0;
int lastAppliedActiveIndex = -2;
bool lastAppliedDisableProtection = false;

// Prototipos
void calibrateAll(int samples);
void loadPersistent();
void savePersistent();
void handleSerial();
void applyRelays();
void updateRelaysIfNeeded();
float readCurrentA(int ch);
void showStatusSerial();
void sendTelemetry();
void blynkSendAllValues();
void onResetButtonPress(int idx);
void updateOutletVoltage(int idx, int v);
void setProtection(bool disabled);
void tryBlynkReconnectIfNeeded();
void printWiFiStatus();
const char* wifiStatusToString(int s);

// Funções

void calibrateAll(int samples) {
  if (calibrating) return;
  calibrating = true;
  Serial.println(">>> Iniciando recalibracao...");
  if (display.width() > 0) {
    display.clearDisplay();
    display.setCursor(0,0);
    display.println("Recalibrando...");
    display.display();
  }
  for (int ch = 0; ch < 3; ch++) {
    long sum = 0;
    for (int i = 0; i < samples; i++) {
      sum += ads.readADC_SingleEnded(ch);
      delay(1);
    }
    sensorOffsetRaw[ch] = float(sum) / samples;
    Serial.printf("Offset Canal %d = %.2f contagens\n", ch+1, sensorOffsetRaw[ch]);
```

```
  }
  lastRecal = millis();
  calibrating = false;
  initialCalibrated = true;
  Serial.println("<<< Recalibracao finalizada.");
  delay(50);
}

void loadPersistent() {
  prefs.begin("energy", true);
  for (int i = 0; i < 3; i++) {
    energy_kvah[i] = prefs.getFloat(String("kvah"+String(i)).c_str(), 0.0F);
    outletVoltage[i] = prefs.getInt(String("v"+String(i)).c_str(), outletVoltage[i]);
  }
  CUSTO_KVAH = prefs.getFloat("cost_kvah", 0.0F);
  disable_protection = prefs.getBool("prot_disabled", false);
  prefs.end();
  Serial.println("--- Dados carregados da flash ---");
  for (int i=0;i<3;i++) Serial.printf("T%d: %.6f kVAh | %dV\n", i+1, energy_kvah[i], outletVoltage[i]);
  Serial.printf("Tarifa carregada: %.4f\n", CUSTO_KVAH);
}

void savePersistent() {
  prefs.begin("energy", false);
  for (int i = 0; i < 3; i++) {
    prefs.putFloat(String("kvah"+String(i)).c_str(), energy_kvah[i]);
    prefs.putInt(String("v"+String(i)).c_str(), outletVoltage[i]);
  }
  prefs.putFloat("cost_kvah", CUSTO_KVAH);
  prefs.putBool("prot_disabled", disable_protection);
  prefs.end();
  lastSaveTime = millis();
  Serial.println("Dados salvos na flash.");
}

float readCurrentA(int ch) {
  if (!initialCalibrated) {
    Serial.printf("AVISO: leitura solicitada antes de calibracao (canal %d). Retornando 0.\n", ch+1);
    return 0.0F;
  }
  float sumOfSquares = 0.0F;
  for (int i = 0; i < RMS_SAMPLES; i++) {
    float sample = ads.readADC_SingleEnded(ch);
    float instVoltage = (sample - sensorOffsetRaw[ch]) * LSB;
    sumOfSquares += (instVoltage * instVoltage);
```

```
  }
  float meanSquare = sumOfSquares / RMS_SAMPLES;
  float rmsVoltage = sqrt(meanSquare);
  float rmsCurrent = rmsVoltage / sensitivity;
  return rmsCurrent;
}

void applyRelays() {
  if (disable_protection) {
    for (int t=0; t<3; t++)
      for (int k=0; k<3; k++)
        digitalWrite(relayPins[t][k], HIGH);
    return;
  }
  for (int t=0; t<3; t++){
    bool shouldBeOn = (state == IDLE) || (t == activeIndex);
    for (int k=0; k<3; k++){
      digitalWrite(relayPins[t][k], shouldBeOn ? HIGH : LOW);
    }
  }
}

// Atualiza relés apenas quando necessário e inicia o tempo de "settle"
void updateRelaysIfNeeded() {
  if (activeIndex != lastAppliedActiveIndex || disable_protection != lastAppliedDisableProtection) {
    if (disable_protection) {
      for (int t=0; t<3; t++)
        for (int k=0; k<3; k++)
          digitalWrite(relayPins[t][k], HIGH);
    } else {
      for (int t=0; t<3; t++){
        bool shouldBeOn = (state == IDLE) || (t == activeIndex);
        for (int k=0; k<3; k++){
          digitalWrite(relayPins[t][k], shouldBeOn ? HIGH : LOW);
        }
      }
    }
    lastAppliedActiveIndex = activeIndex;
    lastAppliedDisableProtection = disable_protection;
    waitingForSettle = true;
    settleStart = millis();
    Serial.printf("Relays atualizados (active=%d, prot=%d). Aguardando %lums para estabilizar.\n",
activeIndex, disable_protection, settleDelayMs);
  }
}
```

```
void showStatusSerial() {
  Serial.println("\n=== STATUS ===");
  Serial.printf("Estado: %s | Tomada Ativa: %d\n", state==IDLE?"OCIOSO":"ATIVO", activeIndex!=-1 ?
activeIndex+1 : 0);
  for (int i=0;i<3;i++){
    float S_VA = (outletVoltage[i] == 380) ? sqrtf(3.0F) * outletVoltage[i] * currents[i] : outletVoltage[i]
* currents[i];
    float price = energy_kvah[i] * CUSTO_KVAH;
    Serial.printf("T%d [%dV]: I=%.3f A | S=%.1f VA | E=%.6f kVAh | Preco=R$ %.2f\n", i+1,
outletVoltage[i], currents[i], S_VA, energy_kvah[i], price);
  }
  Serial.printf("Tarifa: %.4f | Protecao Desativada: %s\n", CUSTO_KVAH, disable_protection ? "SIM" :
"NAO");
  Serial.println("=============\n");
}

void handleSerial() {
  if (Serial.available()) {
    String s = Serial.readStringUntil('\n');
    s.trim();
    s.toUpperCase();
    if (s == "SHOW") {
      showStatusSerial();
    } else if (s == "WIFI") {
      printWiFiStatus();
    } else if (s.startsWith("RESET ALL")) {
      for (int i=0;i<3;i++) energy_kvah[i]=0.0F;
      savePersistent();
      Serial.println("TODAS energias zeradas.");
    } else if (s == "SAVE") {
      savePersistent();
      Serial.println("Salvo.");
    } else if (s == "RECAL") {
      if (!calibrating && millis() - lastRecal > 1000UL) calibrateAll(recalSamples);
      Serial.println("Recalibracao solicitada.");
    } else if (s.startsWith("FORCE ")) {
      int idx = s.substring(6).toInt();
      if (idx >= 1 && idx <= 3) {
        activeIndex = idx - 1;
        state = ACTIVE;
        refCurrent = currents[activeIndex];
        lastCheckTime = millis();
        Serial.printf("FORCE: Tomada %d ativada via comando.\n", idx);
        updateRelaysIfNeeded();
```

```
    } else if (idx == 0) {
      state = IDLE;
      activeIndex = -1;
      Serial.println("FORCE: Voltando para IDLE.");
      updateRelaysIfNeeded();
    } else {
      Serial.println("Uso: FORCE 1 | FORCE 2 | FORCE 3 | FORCE 0");
    }
   } else if (s.startsWith("COST ")) {
    // permitir ajuste rápido do custo via Serial: ex: COST 0.85
    float v = s.substring(5).toFloat();
    if (v >= 0.0) {
      CUSTO_KVAH = v;
      savePersistent();
      Serial.printf("Tarifa atualizada via serial: %.4f\n", CUSTO_KVAH);
    } else {
      Serial.println("Uso: COST <valor>");
    }
   } else {
    Serial.println("Comandos: SHOW | WIFI | RECAL | SAVE | RESET ALL | FORCE N | COST x.xx");
   }
  }
}

// Blynk callbacks
BLYNK_WRITE(V20) { if (param.asInt()==1) onResetButtonPress(0); }
BLYNK_WRITE(V21) { if (param.asInt()==1) onResetButtonPress(1); }
BLYNK_WRITE(V22) { if (param.asInt()==1) onResetButtonPress(2); }
BLYNK_WRITE(V30) { updateOutletVoltage(0, param.asInt()); }
BLYNK_WRITE(V31) { updateOutletVoltage(1, param.asInt()); }
BLYNK_WRITE(V32) { updateOutletVoltage(2, param.asInt()); }

BLYNK_WRITE(V40) {
  float val = param.asFloat();
  if (val >= 0.0) {
    CUSTO_KVAH = val;
    savePersistent();
    Serial.printf("Nova tarifa: %.4f\n", CUSTO_KVAH);
    if (Blynk.connected()) {
      terminal.printf("Tarifa salva: %.4f\n", CUSTO_KVAH);
      terminal.flush();
    }
  }
}
```

```
BLYNK_WRITE(V50) {
  setProtection(param.asInt() == 1);
}

BLYNK_WRITE(V60) {
  String s = param.asStr();
  s.trim();
  if (s.length() > 0) {
    Serial.printf("Terminal: %s\n", s.c_str());
    if (s.equalsIgnoreCase("SHOW")) {
      showStatusSerial();
      terminal.println("--- Status ---");
      for(int i=0;i<3;i++){
        float price = energy_kvah[i] * CUSTO_KVAH;
        terminal.printf("T%d: %.3fA | %.4fkVAh | R$ %.2f\n", i+1, currents[i], energy_kvah[i], price);
      }
      terminal.flush();
    } else if (s.equalsIgnoreCase("RESET ALL")) {
      for (int i=0;i<3;i++) energy_kvah[i]=0.0F;
      savePersistent();
      if (Blynk.connected()) { terminal.println("Energias zeradas."); terminal.flush(); }
    } else {
      if (Blynk.connected()) { terminal.println("Comando desconhecido."); terminal.flush(); }
    }
  }
}

void onResetButtonPress(int idx) {
  if (idx < 0 || idx > 2) return;
  energy_kvah[idx] = 0.0F;
  savePersistent();
  Serial.printf("Reset T%d\n", idx+1);
  if (Blynk.connected()) {
    terminal.printf("T%d zerada.\n", idx+1);
    terminal.flush();
    Blynk.virtualWrite(VP_ENERGY[idx], String(energy_kvah[idx], 6));
  }
}

void updateOutletVoltage(int idx, int v) {
  if (idx<0 || idx>2) return;
  if (v==127 || v==220 || v==380) {
    outletVoltage[idx] = v;
    savePersistent();
    Serial.printf("T%d -> %dV\n", idx+1, v);
```

```
    if (Blynk.connected()) { terminal.printf("T%d configurada %dV\n", idx+1, v); terminal.flush(); }
  }
}

void setProtection(bool disabled) {
  disable_protection = disabled;
  savePersistent();
  Serial.printf("Protecao Desativada: %s\n", disable_protection ? "SIM" : "NAO");
  if (Blynk.connected()) { terminal.printf("Protecao %s\n", disable_protection ? "DESATIVADA" :
"ATIVADA"); terminal.flush(); Blynk.virtualWrite(VP_PROT, disable_protection ? 1 : 0); }
  updateRelaysIfNeeded();
}

void sendTelemetry() {
  if (!Blynk.connected()) return;
  for (int i=0;i<3;i++){
    float sendCurr = currents[i];
    if (!disable_protection && state == ACTIVE && i != activeIndex) sendCurr = 0.0F;
    Blynk.virtualWrite(VP_CURR[i], String(sendCurr, 3));
    Blynk.virtualWrite(VP_ENERGY[i], String(energy_kvah[i], 6));
    Blynk.virtualWrite(VP_VOLT[i], outletVoltage[i]);
  }
  Blynk.virtualWrite(VP_COST, String(CUSTO_KVAH, 4));
  Blynk.virtualWrite(VP_PROT, disable_protection ? 1 : 0);

  // envia relatório detalhado para o Terminal do app
  terminal.println("\n---- Telemetria ----");
  for (int i=0;i<3;i++){
    float sendCurr = currents[i];
    if (!disable_protection && state == ACTIVE && i != activeIndex) sendCurr = 0.0F;
    float S_VA = (outletVoltage[i] == 380) ? sqrtf(3.0F) * outletVoltage[i] * sendCurr : outletVoltage[i] *
sendCurr;
    float price = energy_kvah[i] * CUSTO_KVAH;
    terminal.printf("T%d [%dV]: %.2fA | %.0fVA | %.4fkVAh | R$ %.2f\n", i+1, outletVoltage[i],
sendCurr, S_VA, energy_kvah[i], price);
  }
  terminal.println("--------------------");
  terminal.flush();
}

void blynkSendAllValues() {
  if (!Blynk.connected()) return;
  for (int i=0;i<3;i++){
    float sendCurr = currents[i];
    if (!disable_protection && state == ACTIVE && i != activeIndex) sendCurr = 0.0F;
```

```
    Blynk.virtualWrite(VP_CURR[i], String(sendCurr, 3));
    Blynk.virtualWrite(VP_ENERGY[i], String(energy_kvah[i], 6));
    Blynk.virtualWrite(VP_VOLT[i], outletVoltage[i]);
  }
  Blynk.virtualWrite(VP_COST, String(CUSTO_KVAH, 4));
  Blynk.virtualWrite(VP_PROT, disable_protection ? 1 : 0);


}

void tryBlynkReconnectIfNeeded() {
  unsigned long now = millis();
  if (now - lastBlynkTry < blynkReconnectIntervalMs) return;
  lastBlynkTry = now;
  if (WiFi.status() == WL_CONNECTED && !Blynk.connected()) {
    Serial.println("Tentando reconectar ao Blynk...");
    Blynk.connect();
    if (Blynk.connected()) Serial.println("Blynk reconectado!");
    else Serial.println("Falha na reconexao Blynk.");
  }
}

void printWiFiStatus() {
  int s = WiFi.status();
  Serial.printf("WiFi status: %s (%d)\n", wifiStatusToString(s), s);
  if (s == WL_CONNECTED) {
    Serial.printf("IP: %s\n", WiFi.localIP().toString().c_str());
    Serial.printf("RSSI: %d dBm\n", WiFi.RSSI());
  }
}

const char* wifiStatusToString(int s) {
  switch (s) {
    case WL_NO_SHIELD: return "NO_SHIELD";
    case WL_IDLE_STATUS: return "IDLE_STATUS";
    case WL_NO_SSID_AVAIL: return "NO_SSID_AVAIL";
    case WL_SCAN_COMPLETED: return "SCAN_COMPLETED";
    case WL_CONNECTED: return "CONNECTED";
    case WL_CONNECT_FAILED: return "CONNECT_FAILED";
    case WL_CONNECTION_LOST: return "CONNECTION_LOST";
    case WL_DISCONNECTED: return "DISCONNECTED";
    default: return "UNKNOWN";
  }
}
```

```
// setup / loop
void setup() {
  Serial.begin(115200);
  delay(10);
  Serial.println("\n--- INICIANDO MONITOR DE ENERGIA TCC v3 ---");

  for (int t=0; t<3; t++)
    for (int k=0; k<3; k++) { pinMode(relayPins[t][k], OUTPUT); digitalWrite(relayPins[t][k], HIGH); }

  Wire.begin(SDA_PIN, SCL_PIN);

  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println("ERRO: OLED nao encontrado!");
  } else {
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0,0);
    display.println("Iniciando...");
    display.display();
  }

  ads.begin(0x48);
  ads.setGain(GAIN_TWOTHIRDS);
  ads.setDataRate(RATE_ADS1115_860SPS);
  Serial.println("ADC iniciado a 860 SPS.");

  loadPersistent();

  delay(200);
  calibrateAll(calibSamples);
  lastSampleTime = millis();
  lastSaveTime = millis();
  lastCheckTime = millis();

  Serial.printf("Tentando conectar ao WiFi: %s\n", WIFI_SSID);
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);

  unsigned long startWifi = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - startWifi < 5000UL) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
```

```
  printWiFiStatus();

  Blynk.config(BLYNK_AUTH_TOKEN);
  tryBlynkReconnectIfNeeded();

  updateRelaysIfNeeded();

  blynkTimer.setInterval(telemetryIntervalMs, sendTelemetry);
  blynkTimer.setInterval(5000L, blynkSendAllValues);
  blynkTimer.setInterval(blynkReconnectIntervalMs, tryBlynkReconnectIfNeeded);
}

void loop() {
  unsigned long now = millis();

  handleSerial();
  Blynk.run();
  blynkTimer.run();

  // Wifi status debug
  if (now - lastWifiStatusPrint >= wifiStatusInterval) {
    lastWifiStatusPrint = now;
    printWiFiStatus();
  }

  // Se estamos aguardando estabilização e o tempo já passou, desativa waitingForSettle
  if (waitingForSettle && (now - settleStart >= settleDelayMs)) {
    waitingForSettle = false;
    Serial.println("Settling terminado. Voltando a ler sensores.");
    lastSampleTime = 0; // força leitura imediata
  }

  // Atualiza relés se necessário (quando estado ou proteção mudam)
  updateRelaysIfNeeded();

  if (now - lastSampleTime >= sampleIntervalMs) {
    lastSampleTime = now;

    if (waitingForSettle) {
      Serial.println("Aguardando estabilidade (skip leitura)...");
      // skip leitura e acumulo
    } else {
      unsigned long elapsedMs = sampleIntervalMs;
      float dt_hours = elapsedMs / 3600000.0F;
```

```
    // 1) leitura das correntes (True RMS)
    for (int ch=0; ch<3; ch++){
     currents[ch] = readCurrentA(ch);
    }

    // 2) acumula energia com base nas correntes medidas reais
    for (int ch=0; ch<3; ch++) {
     float S_VA = (outletVoltage[ch] == 380) ? sqrtf(3.0F) * (float)outletVoltage[ch] * currents[ch] :
(float)outletVoltage[ch] * currents[ch];
     energy_kvah[ch] += (S_VA * dt_hours) / 1000.0F;
    }

    // 3) máquina de estados
    if (!disable_protection) {
     if (state == IDLE) {
      for (int ch=0; ch<3; ch++){
       if (currents[ch] > thresholdA) {
        activeIndex = ch;
        state = ACTIVE;
        refCurrent = currents[ch];
        lastCheckTime = now;
        Serial.printf("-> ATIVACAO T%d (%.2f A)\n", ch+1, currents[ch]);
        updateRelaysIfNeeded(); // aplica mudança e entra em settle
        break;
       }
      }
      if (!calibrating && initialCalibrated && (now - lastRecal > minRecalIntervalMs)) {
       calibrateAll(recalSamples);
      }
     } else {
      if (now - lastCheckTime >= checkInterval) {
       float cur = readCurrentA(activeIndex);
       float delta = fabs(cur - refCurrent);
       Serial.printf("Monitorando T%d: Ref=%.2fA | Atual=%.2fA | Delta=%.2fA\n", activeIndex+1,
refCurrent, cur, delta);
       if (cur < thresholdA) {
        Serial.printf("-> T%d desligada. IDLE.\n", activeIndex+1);
        state = IDLE;
        activeIndex = -1;
        updateRelaysIfNeeded(); // aplica mudança e entra em settle
        if (!calibrating && (now - lastRecal > minRecalIntervalMs)) calibrateAll(recalSamples);
       } else {
        refCurrent = cur;
        lastCheckTime = now;
       }
```

```
        }
       }
      } else {
       state = IDLE;
       activeIndex = -1;
      }
     }

     // 5) display: quando tomada ativa, mostra corrente, energia e preço acumulado
     if (display.width() > 0) {
      display.clearDisplay();
      display.setCursor(0,0);
      if (waitingForSettle) {
       display.println("AJUSTANDO...");
      } else if (!disable_protection && state == ACTIVE && activeIndex >= 0) {
       float price = energy_kvah[activeIndex] * CUSTO_KVAH;
       display.printf("TOMADA %d ATIVA\n", activeIndex+1);
       display.printf("I: %.2f A\n", currents[activeIndex]);
       display.printf("E: %.4f kVAh\n", energy_kvah[activeIndex]);
       display.printf("Preco: R$ %.2f\n", price);
      } else {
       display.println(disable_protection ? "MODO LIVRE" : "AGUARDANDO USO...");
      }
      display.setCursor(0,45);
      display.printf("T1:%.3f T2:%.3f", energy_kvah[0], energy_kvah[1]);
      display.setCursor(0,55);
      display.printf("T3:%.3f kVAh", energy_kvah[2]);
      display.display();
     }

     // 6) salvamento periódico
     if (now - lastSaveTime >= saveIntervalMs) savePersistent();
    } // fim amostragem

    delay(5);
}
```