RELÉ TEMPORIZADOR MULTIFUNCIONAL, CONFIGURADO COM APP VIA BLUETOOTH

Ariel S S Ribeiro, Rogério L Nascimento
Instituto Federal de Santa Catarina
Campus Jaraguá do Sul – Rau – Bacharelado em Engenharia Elétrica
e-mail: ariel.s21@aluno.ifsc.edu.br, rogerio.nascimento@ifsc.edu.br
Trabalho de Conclusão de Curso – 28/08/2025

***Resumo* – Este artigo apresenta o desenvolvimento de um relé temporizador multifuncional configurável por meio de aplicativo móvel utilizando comunicação *Bluetooth Low Energy (BLE)*. O sistema proposto é composto por uma placa de circuito baseada no microcontrolador ESP32, alimentada por um módulo HLK-PM03, associada a um sensor de tensão ZMPT101B e a um módulo relé de dois canais. Foram implementados cinco modos de temporização distintos, incluindo retardo na energização, retardo na desenergização, operação cíclica com início ligado, operação cíclica com início desligado e a função estrela-triângulo, permitindo flexibilidade de aplicação em diferentes cenários industriais e acadêmicos. O *firmware* foi desenvolvido em linguagem C na plataforma Arduino, enquanto o aplicativo móvel foi implementado em *Flutter*, possibilitando a configuração remota. Os testes realizados confirmaram a confiabilidade do protótipo, evidenciando sua aplicabilidade no ensino e em aplicações práticas de acionamentos elétricos e automação industrial.**

***Palavras-chave* – Relé temporizador, Bluetooth, ESP32, Automação, Flutter.**

I. INTRODUÇÃO

Em diferentes segmentos da indústria e da sociedade há uma demanda crescente por dispositivos que auxiliem no controle de processos e parâmetros operacionais. Entre os recursos utilizados para essa finalidade, destacam-se os relés temporizadores, cuja aplicação varia desde o acionamento de sistemas de iluminação até a coordenação de processos industriais complexos. Contudo, configurações inadequadas podem ocasionar atrasos, falhas ou até mesmo interrupções totais de funcionamento, gerando prejuízos técnicos e econômicos [1].

Com a diversificação dos tipos de processos e a adoção de protocolos de comunicação sem fio em diversas esferas de automação, torna-se relevante o desenvolvimento de um relé temporizador capaz de integrar múltiplas funções em um único dispositivo. Além de contribuir para a redução de custos de instalação e manutenção, um componente multifuncional, configurável via aplicativo, possibilita ajustes de maneira mais intuitiva, dinâmica e remota [2].

Nesse contexto, observa-se um movimento ascendente de soluções que exploram a conectividade *Bluetooth* para a integração com dispositivos móveis. Tais soluções refletem a Indústria 4.0 e a Internet das Coisas (IoT), em que a comunicação imediata entre máquinas e usuários amplia a capacidade de monitoramento e controle de sistemas. A crescente popularização de *smartphones* reforça a importância de disponibilizar interfaces amigáveis para a programação de temporizadores, atendendo às necessidades de agilidade e flexibilidade na indústria e em aplicações residenciais e comerciais [3].

Trabalhos anteriores abordam o desenvolvimento de relés temporizadores digitais a partir de microcontroladores de baixo custo, mostrando que a adoção de circuitos eletrônicos simplificados não apenas eleva a precisão do controle de tempos como também possibilita a expansão das funcionalidades, tais como modos de operação cíclica, retardo de desligamento e acionamentos escalonados [4]. Adicionalmente, estudos sobre a implementação de redes sem fio em ambientes industriais reforçam o potencial de protocolos como o *BLE* , que mantém consumo reduzido de energia e garante segurança na comunicação entre dispositivos [5].

Além disso, pesquisas indicam que a interface de configuração via aplicativo móvel tende a melhorar a experiência do usuário, já que elimina a necessidade de manipular fisicamente botões ou interfaces restritas no próprio dispositivo [6]. Por meio de aplicativos, o operador pode monitorar diversas variáveis em tempo real, personalizando modos de temporização, o que amplia a aplicabilidade do produto.

Considerando-se esses fatores, este trabalho propõe o desenvolvimento de um relé temporizador multifuncional configurável via *Bluetooth*, com foco na implementação de diferentes modos de temporização e na facilidade de uso proporcionada pela interface gráfica em um aplicativo móvel. O dispositivo é capaz de realizar funções de retardo na energização, retardo na desenergização, operação cíclica com início ligado, operação cíclica com início desligado e partida estrela-triângulo para motores de indução trifásicos.

Além do caráter acadêmico, ressalta-se o potencial de transformação deste projeto em um produto comercial. Um único equipamento, ao reunir diversas funções em um

mesmo hardware, pode substituir a necessidade de múltiplos relés temporizadores específicos. Dessa forma, indústrias não precisariam manter diferentes modelos em estoque para reposição, reduzindo custos logísticos e de manutenção, ao mesmo tempo em que ampliam a flexibilidade de aplicação em campo.

O dispositivo, portanto, busca unir robustez, versatilidade e praticidade, apresentando-se como uma solução viável tanto para aplicações industriais quanto para contextos comerciais e residenciais.

A execução deste projeto baseia-se em competências desenvolvidas ao longo das unidades curriculares como Circuitos Elétricos, que fornecem fundamentos para análise e dimensionamento dos componentes; Programação, necessária para o desenvolvimento do *firmware* e do aplicativo de configuração; Microcontroladores, crucial para a lógica de controle embarcada; Eletrônica Digital, que embasa a arquitetura do dispositivo; Instrumentação Eletrônica, fundamental para aquisição e condicionamento de sinais; Automação Industrial, que orienta a integração do dispositivo em processos produtivos, assegurando confiabilidade e eficiência; e Acionamentos Industriais, onde se aplica efetivamente o estudo e a elaboração de comandos elétricos, garantindo a utilidade prática do projeto na operação de chaves de partida para acionamento de motores elétricos.

A Seção II apresenta a fundamentação teórica sobre os modos de temporização, tecnologias de relés e protocolo de comunicação BLE. Na Seção III são descritos os métodos e materiais empregados para o desenvolvimento do protótipo, contemplando *hardware*, *firmware* e aplicativo móvel. Por fim, na Seção IV são discutidos os resultados obtidos e apresentadas as conclusões, juntamente com sugestões para trabalhos futuros.

II. FUNDAMENTAÇÃO TEÓRICA

Esta seção aborda os conceitos essenciais para a compreensão do projeto, iniciando pela definição e aplicação de relés temporizadores. Em seguida, são detalhadas as tecnologias de software e hardware empregadas, como o protocolo de comunicação BLE e o *framework Flutter* para o desenvolvimento da interface móvel, que constituem a base para a implementação do sistema proposto.

A. Relés temporizadores

Relés são dispositivos eletromecânicos ou eletrônicos que funcionam como interruptores controlados eletricamente, permitindo o acionamento ou desligamento de cargas em circuitos de potência. Os relés temporizadores, em particular, incorporam funcionalidades para controle temporal, tais como retardo na energização, retardo na desenergização e operação cíclica programável, entre outras. Suas aplicações vão desde sistemas de iluminação até controle de sistemas *HVAC* (aquecimento, ventilação e ar-condicionado), incluindo ainda a coordenação de partidas de motores industriais, como no caso da partida estrela-triângulo, que reduz a corrente inicial em motores trifásicos [7].

No âmbito deste projeto, o equipamento desenvolvido, denominado Relé Temporizador Multifuncional (RTM), é continuamente alimentado por uma rede elétrica monofásica de 220 V_{CA}, aplicada aos bornes **F** e **A2**. A inicialização da contagem de tempo requer a aplicação de um sinal de comando, igualmente de 220 V_{CA}, no borne **A1**. O RTM é constituído por dois relés com contato reversível (**RL1** e **RL2**), conforme ilustrado na Figura 1.

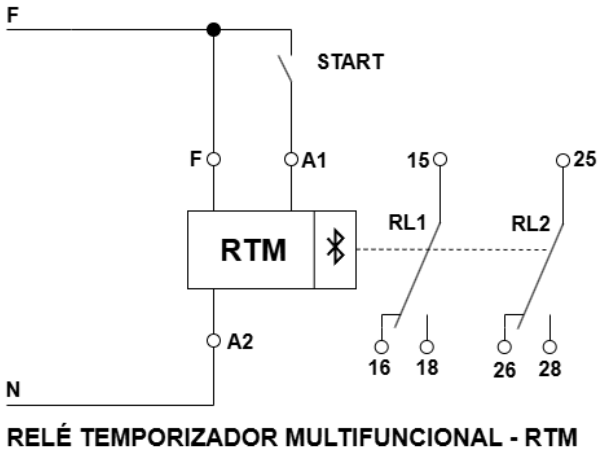

Fig. 1. Diagrama esquemático do Relé Temporizador Multifuncional.

Neste projeto, o **RTM** será implementado com cinco modos de operação.

1) ***Relé Temporizador Multifuncional com retardo na energização (RTM-RE):*** ao receber o sinal de comando no borne **A1**, o dispositivo inicia a contagem de tempo (T) previamente programada no aplicativo . Após o término do intervalo de tempo configurado, os relés **RL1** e **RL2** realizam a comutação de seus contatos. A Figura 2 apresenta o diagrama temporal funcional correspondente a este modo de operação.

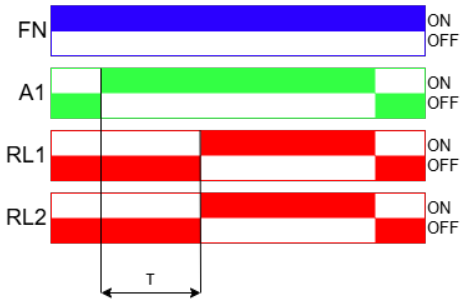

Fig. 2. Diagrama temporal do RTM-RE

A Figura 3 apresenta a aplicação do relé temporizador multifuncional do tipo retardo na energização (RTM-RE) em um circuito elétrico de comando, destinado ao acionamento sequencial e temporizado de dois motores trifásicos, por meio

dos contatores **K1** e **K2**. O circuito de comando é constituído pelo fusível **F1**, responsável pela proteção contra curto-circuito; pelos relés térmicos **FT1** e **FT2**, que asseguram a proteção dos motores contra sobrecarga; pelo botão de impulso **S0**, utilizado para o desligamento dos dois motores; pelo botão de impulso **S1**, empregado na partida do sistema; e pelo relé temporizador multifuncional RTM-RE. Ressalta-se que este circuito de comando atua sobre o circuito de força dos motores, o qual não está representado neste trabalho. O referido circuito de força é composto por três fusíveis, destinados à proteção contra curto-circuito; por um relé térmico, responsável pela proteção contra sobrecarga; e por um contator, que desempenha a função de chaveamento para ligar e desligar o motor.

O funcionamento ocorre da seguinte forma: ao pressionar o botão **S1**, a bobina do contator **K1** é energizada, acionando o primeiro motor **(M1)**. Mesmo após a liberação do botão **S1**, o contator **K1** permanece energizado por meio do contato auxiliar 13-14, conhecido como contato de retenção ou selo. Simultaneamente, o borne **A1** do RTM-RE recebe o sinal de partida e inicia a contagem do tempo previamente configurado no aplicativo (APP). Após transcorrido o intervalo programado, o contato auxiliar **15-18** do RTM-RE é comutado, promovendo a energização da bobina do contator **K2**, responsável pelo acionamento do segundo motor **(M2)**. Os dois motores são desligados juntos, ao pressionar o botão **S0** ou por uma sobrecarga em qualquer um dos dois motores, através dos contatos auxiliares dos relés térmicos **FT1** e **FT2**.

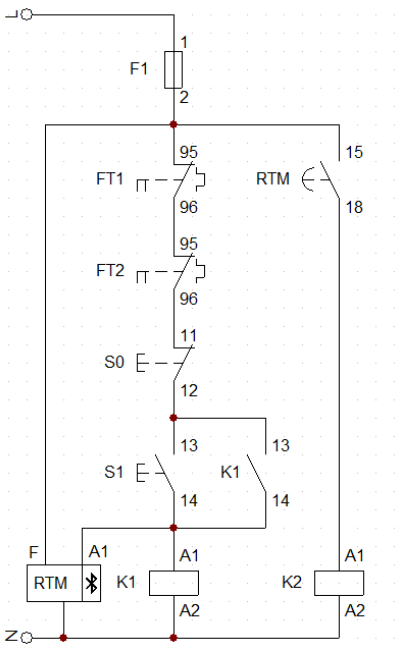

Fig. 3. Diagrama de ligação do RTM-RE e RTM-RD.

2) ***Relé Temporizador Multifuncional com Retardo na desenergização (RTM-RD)****:* ao receber o sinal de comando no borne **A1**, o dispositivo de imediato realiza a comutação dos relés **RL1** e **RL2**. Quando o borne **A1** deixa de receber o sinal de comando, inicia-se a contagem de tempo (T) previamente programada no aplicativo. Após o término do intervalo de tempo configurado, os relés **RL1** e **RL2** são desligados, retornando seus contatos à condição inicial. A Figura 4 apresenta o diagrama temporal funcional correspondente a este modo de operação.

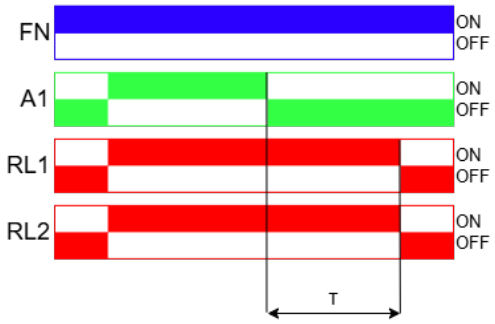

Fig. 4. Diagrama temporal do RTM-RD

O circuito apresentado na Figura 3 também pode ser empregado como aplicação do relé temporizador multifuncional no modo retardo na desenergização (RTM-RD). Nessa configuração, o funcionamento ocorre conforme descrito a seguir: ao acionar o botão de impulso **S1**, a bobina do contator **K1** é energizada, promovendo o acionamento do primeiro motor **(M1)**. Mesmo após a liberação do botão **S1**, o contator K1 permanece energizado por meio do contato auxiliar 13-14, denominado contato de retenção ou selo. Simultaneamente, o borne **A1** do RTM-RD recebe o sinal de comando e, de forma imediata, realiza a comutação do contato auxiliar **15-18**, o que resulta na energização do contator **K2** e, consequentemente, no acionamento simultâneo dos motores **M1** e **M2**.

Quando o botão de impulso S0 é pressionado, o primeiro motor (M1) é desligado instantaneamente, interrompendo a alimentação do borne A1 do RTM-RD. A partir desse momento, inicia-se a contagem do tempo previamente programado. Após o término do intervalo configurado, o contato auxiliar 15-18 do RTM-RD é desativado, desligando a bobina do contator K2 e, por consequência, promovendo a parada do segundo motor (M2).

3) ***Relé Temporizador Multifuncional Cíclico, início ligado (RTM-CL)****:* ao receber o sinal de comando no borne **A1**, o dispositivo realiza imediatamente a comutação dos relés **RL1** e **RL2**, iniciando a contagem de tempo (T) previamente programada no aplicativo. A partir desse ponto, os relés **RL1** e **RL2** são acionados de forma cíclica, alternando entre os estados de ligado e desligado, até que o sinal de comando no borne **A1** seja interrompido. A Figura 5 apresenta o diagrama temporal funcional correspondente a este modo de operação.

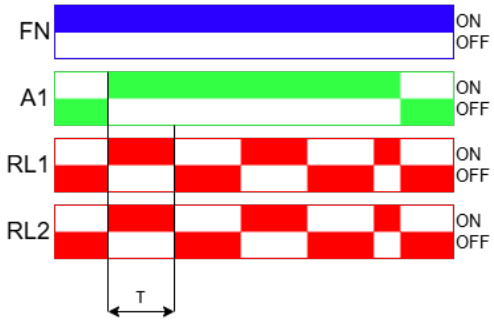

Fig. 5. Diagrama temporal do RTM-CL

4) ***Relé Temporizador Multifuncional cíclico com início desligado (RTM-CD):*** ao receber o sinal de comando no borne **A1**, o dispositivo inicia a contagem de tempo (T) previamente programada no aplicativo. A partir desse ponto, os relés **RL1** e **RL2** são acionados de forma cíclica, alternando entre os estados de desligado e ligado, até que o sinal de comando no borne **A1** seja interrompido. A Figura 6 apresenta o diagrama funcional correspondente a este modo de operação.

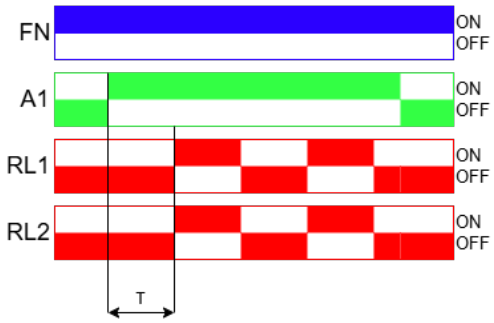

Fig. 6. Diagrama temporal do RTM-CD

A Figura 7 apresenta uma aplicação do relé temporizador multifuncional (RTM) configurado nos modos **RTM-CL** e **RTM-CD**. Nesse arranjo, as bobinas solenoides **Y1** e **Y2** são acionadas de forma alternada pelo RTM, conforme o tempo previamente programado por meio do aplicativo. Esses solenoides atuam diretamente sobre um cilindro pneumático, também denominado pistão pneumático, o qual é responsável pelo comando de uma válvula pneumática do tipo **5/2 vias** (cinco vias e duas posições). Ressalta-se que o respectivo circuito pneumático não está incluído neste trabalho.

Quando o borne **A1** do dispositivo é energizado por meio da chave de comando **S1**, o relé temporizador **RTM** inicia a comutação periódica de seus contatos. Durante o primeiro intervalo de tempo, o contato **15–18** fecha, enquanto o contato **25–26** é aberto, energizando a válvula solenoide **Y1**, responsável pelo avanço do atuador pneumático. Em seguida, ao término do período configurado, ocorre a abertura do contato **15–18** e o fechamento do contato **25–26**, que energiza a válvula solenoide **Y2**, promovendo o recuo do atuador.

Esse ciclo se repete continuamente enquanto o borne **A1** permanecer alimentado, garantindo o movimento alternado do pistão. A diferença entre os modos Relé Temporizador Multifuncional cíclico com início ligado (RTM-CL) e RTM-CD está apenas na condição inicial: no primeiro, o sistema inicia com a saída ligada (**Y1** energizada), enquanto no segundo o início ocorre com a saída desligada (**Y2** inativa).

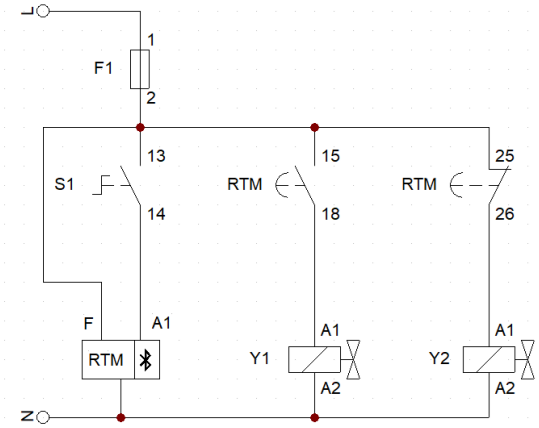

Fig. 7. Diagrama de ligação do RTM-CL e RTM-CD.

5) ***Relé Temporizador Multifuncional Estrela-Triangulo (RTM-ET):*** Ao receber o sinal de comando no borne **A1**, o dispositivo realiza imediatamente a comutação do relé **RL1**, acionando o circuito de força da chave de partida estrela-triângulo para conectar o motor de indução trifásico na configuração estrela. Simultaneamente, inicia-se a contagem de tempo (T) previamente programada no aplicativo. Para esse tipo de chave de partida, o tempo programado deve corresponder ao intervalo necessário para que o motor, sob carga, atinja aproximadamente 90% da sua rotação nominal. Após a conclusão desse período, o relé **RL1** é desligado, iniciando-se uma contagem de 100 ms, após a qual o relé **RL2** é acionado, promovendo a conexão do motor na configuração triângulo no circuito de força. A Figura 8 apresenta o diagrama funcional correspondente a este modo de operação.

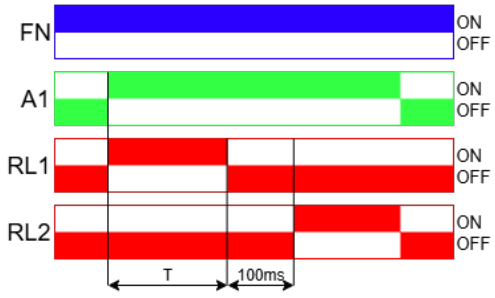

Fig. 8. Diagrama temporal do RTM-ET

A Figura 9 apresenta o diagrama de ligação do relé temporizador multifuncional (RTM) configurado para o modo

estrela-triângulo, amplamente utilizado na partida de motores de indução trifásicos. O circuito de comando é formado pelos contatores **K1**, **K2** e **K3**, além do botão de partida **S1**, botão de desligamento **S0**, relé térmico **FT1** e o próprio RTM.

Ao acionar o botão de impulso *S1*, o borne **A1** do relé temporizador multifuncional (RTM) é energizado, ocasionando a comutação imediata do contato **15-18**. Esse contato alimenta a bobina do contator **K3**, responsável pelo fechamento das bobinas do motor de indução trifásico na configuração estrela. Em seguida, a bobina do contator **K1** é energizada por meio do contato **13-14** de **K3**, estabelecendo a ligação do motor em estrela. Os contatos auxiliares **13-14** e **43-44** de **K1** atuam como circuito de retenção, assegurando a manutenção do sistema em operação.

Concomitantemente, inicia-se a contagem do tempo previamente ajustado no aplicativo. Após o término desse intervalo, o contato **15-18** do RTM é aberto, desligando o contator **K3**. Decorrido um tempo de comutação fixo de **100 ms**, ocorre a mudança do contato **25-28** do RTM, energizando a bobina do contator **K2**. Nesse estágio, realiza-se a conexão do motor de indução trifásico em triângulo, condição na qual o equipamento permanece em regime contínuo de operação até que o botão de desligamento **S0** seja acionado ou ocorra uma condição de sobrecarga, momento em que o relé térmico **FT1** entra em atuação, interrompendo o funcionamento do sistema.

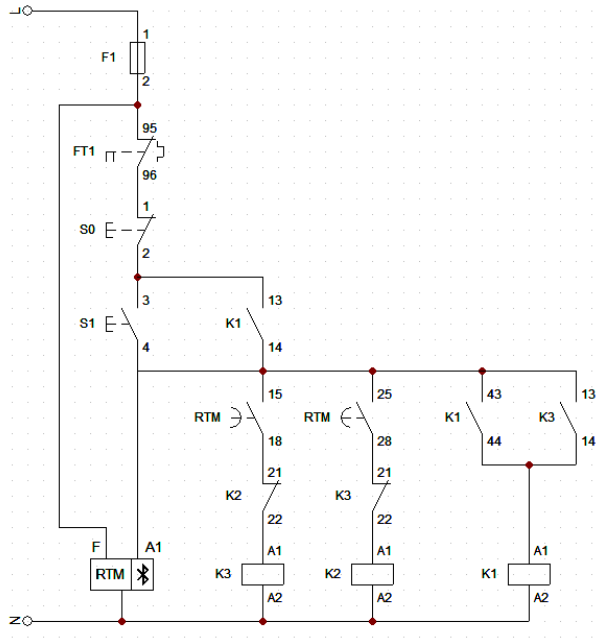

Fig. 9. Diagrama de ligação do RTM no modo Estrela-Triangulo.

O estudo aprofundado desses modos de operação e das tecnologias de controle é essencial para garantir a integração eficaz entre hardware e software, resultando em um sistema funcional, robusto e com boa experiência de uso.

B. Tecnologias Habilitadoras

A implementação de um relé temporizador multifuncional e conectado remotamente depende da integração de diferentes tecnologias de hardware e software, detalhadas a seguir.

1) Microcontrolador: Microcontroladores são circuitos integrados capazes de executar instruções programadas e interagir com periféricos externos, desempenhando papel central em sistemas embarcados e de automação. Em comparação com computadores de uso geral, os microcontroladores possuem recursos mais restritos, porém otimizados para controle em tempo real e baixo consumo de energia. São amplamente utilizados em aplicações industriais, residenciais e acadêmicas, integrando sensores, atuadores e interfaces de comunicação em um único dispositivo [8].

Entre os microcontroladores mais difundidos em prototipagem e desenvolvimento encontram-se as famílias PIC, ARM Cortex, Arduino (baseado em ATmega/ARM) e a linha ESP (Espressif Systems). Cada plataforma oferece diferentes níveis de desempenho, consumo e conectividade, permitindo adequação às necessidades específicas de cada projeto.

Neste projeto, o microcontrolador *ESP32* foi escolhido devido à sua combinação de baixo custo, baixo consumo energético e integração nativa de *Wi-Fi* e *BLE*. Sua arquitetura flexível e ampla adoção em sistemas embarcados consolidam sua posição como escolha robusta para aplicações *IoT* e automação [8].

O ESP32 é compatível com diferentes ambientes de desenvolvimento, como Arduino IDE, ESP-IDF e MicroPython, o que amplia sua flexibilidade de uso em projetos embarcados [9].

2) Tecnologia Bluetooth Low Energy (BLE): O protocolo *BLE* foi projetado para comunicação sem fio com alta eficiência energética, especialmente apropriado para dispositivos *IoT* e sistemas embarcados. Ao contrário do *Bluetooth* clássico, o *BLE* utiliza pacotes menores e intervalos ajustáveis, garantindo eficiência energética e baixa latência. Sua operação típica alcança até 100 metros em espaços abertos, sendo amplamente compatível com *smartphones*, o que facilita sua aplicação em interfaces remotas de configuração [10].

A segurança no *BLE* é garantida por criptografia *AES-128* e mecanismos de autenticação, prevenindo interferências e acessos não autorizados. A utilização do *BLE* no projeto deve-se à facilidade de integração com o microcontrolador *ESP32* e sua ampla adoção em plataformas móveis.

Além disso, a comunicação por *BLE* tornou-se padrão em dispositivos *IoT* e *wearables*. Estudos técnicos indicam que, embora o desempenho real dependa de variáveis como configuração e ambiente, o *BLE* é amplamente valorizado por sua eficiência energética e adequação a comunicações em redes curtas [11].

3) Desenvolvimento de Aplicativo com Flutter: O *Flutter* é um *framework open-source* desenvolvido pelo Google, que permite a criação de aplicativos multiplataforma (*iOS* e *Android*) a partir de uma única base de código escrita em *Dart* [12]. O *Flutter* oferece desempenho próximo ao das soluções nativas e permite interfaces altamente personalizáveis e intuitivas, devido à sua arquitetura baseada em *widgets*.

Embora o *Flutter* não possua suporte nativo a protocolos de comunicação sem fio como o *BLE*, a comunidade disponibiliza pacotes consolidados que estendem suas funcionalidades. Entre eles, destaca-se o *flutter_blue* (e sua variação *flutter_blue_plus*), que provê abstrações de alto nível para operações típicas de comunicação: varredura de dispositivos, emparelhamento, leitura e escrita de características, além do recebimento de notificações assíncronas. Essa integração simplifica a troca de dados entre o aplicativo e o microcontrolador *ESP32*, reduzindo a complexidade de implementação e assegurando a confiabilidade [13].

Dessa forma, o uso de bibliotecas complementares permite ao *Flutter* integrar-se a sistemas embarcados por meio do *BLE*, tornando-o adequado para aplicações que requerem configuração remota, monitoramento em tempo real e interfaces móveis multiplataforma.

C. Arquitetura do Sistema

O sistema do relé temporizador multifuncional desenvolvido neste trabalho utiliza como núcleo o microcontrolador *ESP32*, programado em linguagem *C*, e integra-se a um aplicativo móvel desenvolvido em *Flutter*. A comunicação entre ambos é realizada por meio do protocolo *BLE*, visando o baixo consumo energético e ampla compatibilidade com dispositivos móveis.

Essa combinação tecnológica possibilita a criação de uma solução confiável, configurável remotamente e de fácil uso, atendendo a requisitos típicos de aplicações em automação residencial, comercial e industrial. O *firmware* em linguagem *C* assegura controle direto sobre o *hardware*, resultando em maior confiabilidade e desempenho determinístico, aspectos essenciais em sistemas de temporização de relés [9].

A arquitetura geral do sistema é organizada em três camadas principais, conforme representado na Figura 10.

- **Camada Potência AC**: Circuito de potência responsável pela alimentação, leitura de sinais e acionamento dos relés.
- **Camada de controle**: *Firmware* em linguagem *C* no *ESP32*, responsável por interpretar comandos recebidos via aplicativo e gerenciar os diferentes modos de temporização.
- **Camada de interface**: Aplicativo desenvolvido com *Flutter*, responsável pela configuração e monitoramento do sistema.

A comunicação entre aplicativo e microcontrolador ocorre por meio de um protocolo customizado sobre *BLE*, no qual o aplicativo envia parâmetros de configuração e recebe

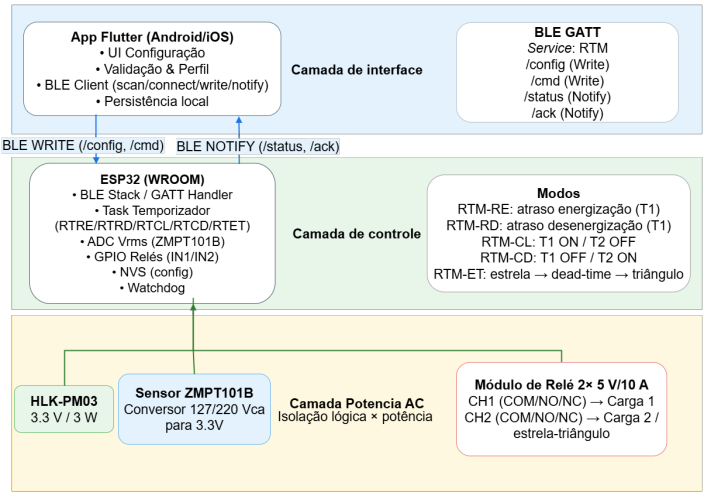

Fig. 10. Diagrama de blocos da arquitetura do relé temporizador.

confirmações e atualizações de estado do *ESP32*. Dessa forma, a integração entre hardware, firmware e interface móvel resulta em um sistema robusto, de fácil configuração e adequado a diferentes cenários de aplicação.

III. METODOLOGIA

Esta seção descreve, de forma reprodutível, os materiais e os métodos empregados no desenvolvimento do relé temporizador multifuncional configurável via *BLE*. A abordagem segue o ciclo *especificar* → *projetar* → *implementar* → *validar*, cobrindo requisitos, *hardware*, *firmware*, aplicativo móvel, prototipagem e testes.

A. Especificações de Requisitos

1) Requisitos funcionais:

RF1 Implementar os cinco modos operacionais apresentados na Fundamentação Teórica.

RF2 Permitir configuração remota de tempos no intervalo de 1 segundo – 1.728.000, equivalente a 20 dias, via BLE, com resolução mínima de **1 s**.

RF3 Oferecer interface gráfica intuitiva (Android / iOS) para seleção de modo, ajuste de parâmetros e monitoramento em tempo real.

RF4 Armazenar localmente a última configuração (EEPROM do ESP32) e restaurá-la no próximo ciclo de energização.

2) Requisitos não funcionais:

RN1 Tensão de alimentação primária: 100 V – 240V AC, 50/60 Hz.

RN2 Tensão lógica máxima: 3.3 V (limite do ESP32).

RN3 Consumo de corrente do sistema, sem carga, inferior a 200 miliampère.

RN4 Alcance mínimo de comunicação BLE: 5 m em ambiente industrial.

B. Projeto de Hardware

1) Alimentação: Para adequar a linha de 220 V_{CA} aos 3.3 V_{DC} exigidos pelo ESP32, utilizou-se o módulo **HLK-PM03** (3,3 V, 1 A, 3 W) — isolado, compacto e certificado para aplicações de baixa potência em placas de circuito impresso. Esse módulo fornece a tensão necessária tanto ao microcontrolador quanto aos periféricos lógicos do sistema.

*2) Condicionamento da entrada **A1**:* O monitoramento do sinal de comando em **A1** foi implementado por meio do sensor de tensão **ZMPT101B**, específico para medições em redes CA. Esse módulo incorpora um transformador de isolamento de alta precisão e circuito de condicionamento, permitindo a detecção de presença de tensão de forma segura e galvanicamente isolada do restante da lógica.

A saída do **ZMPT101B**, devidamente ajustada via potenciômetro de sensibilidade, é conectada ao pino *GPIO34* do ESP32. Esse pino é de uso exclusivo como entrada (*input only*) e não possui resistores internos de *pull-up* ou *pull-down*, de modo que foi necessário empregar um resistor externo de *pull-down* para garantir nível lógico estável. Dessa forma, o microcontrolador é capaz de identificar com precisão as transições de energização e desenergização do comando **A1**.

3) Acionamento dos relés: O acionamento das cargas foi implementado com o módulo **Relé 2 Canais 5 V / 10 A**, projetado para controle direto de dispositivos em corrente alternada até 250 V ou em corrente contínua até 30 V, ambos com capacidade de 10 A. Cada canal do módulo corresponde a um contato **SPDT**[1], disponibilizando terminais de saída **NO**, **NC** e **COM**.

Entre as principais características do módulo, destacam-se:

- **Tensão de operação**: 3.3 V-5 V, com consumo típico de 15–20 mA por canal.
- **Capacidade de comutação**: até 10 A em 250 V_{AC} ou 30 V_{DC}.
- **Isolação**: acionamento por **optoacoplador**, garantindo separação entre o domínio lógico (ESP32) e a carga de potência.
- **Interface de comando**: sinais digitais de 3.3 V - 5 V aplicados aos pinos *IN1* e *IN2*, compatíveis com níveis lógicos do ESP32 mediante ajuste de interface.
- **Tempo de resposta**: entre 5 e 10 ms.
- **Recursos adicionais**: LEDs indicadores de status em cada canal.

Esse módulo substitui a necessidade de circuitos discretos de acionamento com MOSFETs e diodos de roda-livre, uma vez que já integra a proteção contra transientes e a isolação elétrica, simplificando o projeto e aumentando a confiabilidade do sistema.

[1]"Single Pole, Double Throw": um pólo (contato comum) que pode ser comutado entre duas vias — *Normally Closed* (NC) e *Normally Open* (NO).

4) Interface de sinalização por LEDs: A sinalização luminosa foi simplificada para fornecer apenas as informações essenciais ao usuário, sem sobrecarregar a interface. Foram definidos cinco indicadores principais:

- **LEDs de modo (amarelo e azul)**: utilizados em combinação para representar os cinco modos operacionais do relé temporizador, conforme mapeamento:
  1. Apenas azul aceso — Modo 1 (RTM-RE).
  2. Azul piscando — Modo 2 (RTM-RD).
  3. Apenas amarelo aceso — Modo 3 (RTM-CL).
  4. Amarelo piscando — Modo 4 (RTM-CD).
  5. Azul e amarelo piscando simultaneamente — Modo 5 (RTM-ET).
- **LED de entrada**: indica o estado do sinal de comando em **A1**, acendendo sempre que a entrada estiver energizada.
- **LED de contagem**: pisca durante a temporização ativa, auxiliando no diagnóstico do funcionamento do temporizador.
- **LED Azul do ESP32**: aproveitado para indicar status da conexão **BLE**; apagado quando desconectado e aceso fixo quando conectado.
- **LEDs dos relés**: próprios do módulo de relé de 2 canais, acendem quando cada contato é energizado, permitindo inspeção visual imediata do estado das saídas.

C. Projeto de Firmware

O *firmware* foi desenvolvido em linguagem C na **IDE Arduino** (núcleo ESP32 v3.1.2), utilizando o paradigma de máquina de estados finitos para o controle dos diferentes modos de temporização. A implementação foi organizada em módulos independentes, favorecendo a manutenção, a escalabilidade e a clareza do código.

1) Arquitetura do Sistema: A arquitetura adota uma divisão em três camadas principais:

- **Camada de abstração de *hardware* (HAL)** – encapsula funções básicas como configuração de pinos *GPIO*, acionamento dos relés, leitura do sensor de tensão ZMPT101B via ADC interno e inicialização da interface *Bluetooth*.
- **Camada de controle (Core)** – implementa a máquina de estados responsável pela execução dos cinco modos de operação: retardo na energização, retardo na desenergização, operação cíclica com início ligado, operação cíclica com início desligado e estrela-triângulo.
- **Camada de serviços (Service)** – gerencia comandos recebidos via Bluetooth, calibração do sensor, validação de parâmetros e armazenamento das configurações persistentes na memória não volátil (*Preferences*).

2) Máquina de Estados: O controle é orientado a eventos de entrada, representados pela detecção de tensão AC no sensor ZMPT101B e por comandos recebidos via Bluetooth. Cada modo de operação corresponde a um estado distinto, conforme ilustrado no diagrama da Figura 11.

Os temporizadores foram implementados com a função *millis()*, que oferece resolução de 1 ms. Para intervalos de longa duração (até 20 dias), os tempos são armazenados em segundos, prevenindo estouro de variáveis de 32 bits. Além disso, a leitura do sensor ZMPT101B aplica cálculo de valor eficaz (RMS) combinado com média móvel, reduzindo o impacto de ruídos.

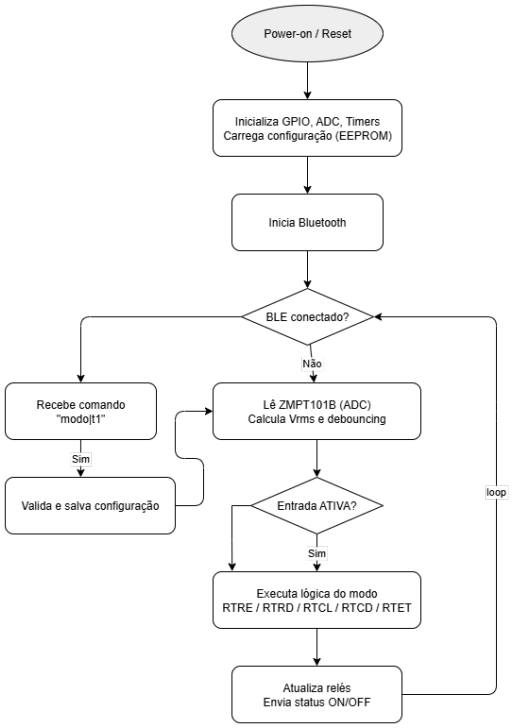

Fig. 11. Fluxo de execução do *firmware*

Descrição do Fluxo. O diagrama da Figura 11 ilustra o ciclo de execução do *firmware*. Após o *reset* ou energização, o sistema inicializa periféricos (*GPIO*, *ADC*, temporizadores) e carrega as configurações armazenadas na EEPROM. Em seguida, o módulo *Bluetooth* é ativado para receber comandos de configuração.

Se a conexão BLE não estiver ativa, o *firmware* mantém a leitura contínua do sensor ZMPT101B, realizando o cálculo de *Vrms* e *debouncing*. Quando um comando é recebido, ele é inicialmente validado e, em seguida, salvo, possibilitando a execução de um dos cinco modos de operação. O núcleo lógico, então, atualiza os relés e transmite periodicamente o status de saída (*ON/OFF*) para o aplicativo.

Tal abordagem proporciona segurança, uma vez que o dispositivo mantém sua operação mesmo na ausência de conexão via *Bluetooth*, retornando automaticamente ao último modo de configuração recebido.

3) Protocolo de Comunicação BLE: A comunicação com o aplicativo foi implementada utilizando a biblioteca nativa *BluetoothSerial*, dispensando dependências externas. O pareamento utiliza o modo “Just Works”, adequado ao perfil de uso do dispositivo.

O método de associação *Just Works* é um procedimento simplificado do *Bluetooth Low Energy*, no qual a conexão entre os dispositivos ocorre sem necessidade de código ou senha. Esse modo prioriza a praticidade, utilizando apenas criptografia básica, sendo indicado para aplicações de baixo risco e ambientes controlados.

Os comandos seguem um formato simplificado baseado em strings delimitadas pelo caractere ”|”:

`"modo|tempo|tempo"`

Onde *modo* corresponde ao identificador numérico do algoritmo de temporização e *tempo* define o intervalo em segundos. Por exemplo, o comando `"3|120"` ativa o modo cíclico com início ligado, alternando o estado dos relés a cada 120 segundos.

O protocolo prevê três tipos de mensagens:

- **Comando** – enviado pelo aplicativo (`"modo|tempo|tempo"`).
- **Resposta** – confirmação de configuração (`"OK"` ou `"ERR"`).
- **Notificação** – atualização em tempo real (`"ON"`, `"OFF"` ou mensagens de calibração).

4) Exemplo de Implementação: O código 1 apresenta um trecho representativo da rotina de recepção de comandos, evidenciando a simplicidade do protocolo adotado.

```
void processarComandosRecebidos() {
  if (SerialBT.available()) {
    String cmd = SerialBT.readStringUntil('\n');
    if (processarConfiguracao(cmd)) {
      salvarConfiguracao();
      iniciarModo();
      SerialBT.println("OK");
    } else {
      SerialBT.println("ERR");
    }
  }
}
```

Código 1: Rotina de processamento de comando Bluetooth.

Essa arquitetura em camadas garante baixo acoplamento, reduzindo a complexidade do código e permitindo a expansão futura com novas funcionalidades, como calibração dinâmica do sensor ou ajustes remotos de parâmetros de detecção.

O código-fonte completo do firmware do ESP32 encontra-se no Apêndice A.

D. Desenvolvimento do Aplicativo

O aplicativo foi desenvolvido em **Flutter 3.29** com linguagem **Dart 3**, oferecendo suporte nativo para Android a partir da

versão 12 e iOS a partir da versão 15. O padrão de arquitetura adotado foi o *Model–View–ViewModel* (MVVM), permitindo separar a camada de interface da lógica de apresentação e do acesso a dados. Essa abordagem favoreceu a testabilidade, manutenção e evolução futura do sistema.

1) Arquitetura: A aplicação foi estruturada em três camadas principais:

- **View** – interface construída em Material 3, com layout responsivo utilizando *LayoutBuilder* e *MediaQuery*, garantindo adaptação a diferentes tamanhos de tela.
- **ViewModel** – classes baseadas em *ChangeNotifier*, responsáveis por expor estados reativos, como conexão Bluetooth, parâmetros válidos e status do relé. O gerenciamento é feito via *provider*, assegurando a atualização da interface em tempo real.
- **Model** – serviço BLE implementado em *RelayTimerService*, que compõe a string de comando no formato "`modo|t1|t2`", envia ao dispositivo e recebe notificações assíncronas. Essa classe foi implementada como um **Singleton**, garantindo que apenas uma instância gerencie toda a comunicação BLE, evitando conflitos de conexão e uso concorrente de recursos.

Além disso, foi definido um **Provider global de Bluetooth**, centralizando a lógica de conexão, reconexão, envio de comandos e recepção de notificações. Essa abordagem simplifica o fluxo do aplicativo, pois qualquer tela pode acessar o estado atualizado da conexão sem precisar replicar código.

2) Persistência: A persistência local foi implementada com `SharedPreferences`, utilizada para armazenar o `deviceId` do último dispositivo conectado, bem como os últimos parâmetros de configuração (`modo`, `t1`, `t2`). Com isso, ao reiniciar o aplicativo, o usuário não precisa reconfigurar manualmente os tempos ou selecionar novamente o dispositivo, aumentando a usabilidade.

Além disso, sempre que o aplicativo se conecta ao *ESP32*, o microcontrolador envia o modo atualmente em execução, garantindo que a interface apresente em tela o estado real do sistema no momento da conexão.

O uso de `SharedPreferences` em conjunto com o padrão `Singleton` no serviço BLE garante que essas informações estejam sempre disponíveis em qualquer ponto do app de forma consistente, sem risco de múltiplas instâncias conflitantes.

3) Pacotes Flutter: O desenvolvimento do aplicativo exigiu a utilização de pacotes complementares ao *framework Flutter*, responsáveis por expandir suas funcionalidades nativas e facilitar a integração com o microcontrolador. Esses pacotes permitem desde o gerenciamento de permissões do sistema operacional até a implementação de conexões sem fio e mecanismos de persistência de dados.

Em particular, destaca-se o uso do `flutter_blue_plus`, responsável por viabilizar a comunicação *BLE* com o microcontrolador *ESP32*. Essa biblioteca fornece recursos de varredura de dispositivos, emparelhamento e troca de dados via serviços e características GATT, possibilitando tanto o envio de comandos (*write*) quanto a recepção de notificações assíncronas (*notify*).

O pacote *permission_handler* foi empregado para tratar permissões de acesso, garantindo que o aplicativo obtenha autorização do usuário para utilização do *Bluetooth* e dos serviços de localização, requisito obrigatório em sistemas *Android* e *iOS*. Já o *provider* viabiliza o gerenciamento de estado de forma reativa, simplificando a injeção de dependências e a comunicação entre camadas da aplicação, enquanto o *shared_preferences* assegura o armazenamento local de parâmetros de configuração, preservando dados entre sessões de uso.

Outros pacotes desempenham papéis de suporte à experiência do usuário, como o *fluttertoast*, empregado para a exibição de notificações rápidas; o *vibration*, que fornece retorno tátil para confirmar ações; o *flutter_material_pickers*, utilizado na criação de seletores intuitivos de opções em formato de *bottom sheet*; e o *wheel_chooser*, que implementa um seletor rotativo para a definição dos tempos de temporização do relé.

Os principais pacotes utilizados estão resumidos na Tabela I.

TABELA I
Pacotes Flutter utilizados no aplicativo

Pacote (v)	Função no app
`flutter_blue_plus 1.35.3`	Escaneamento e conexão BLE (`write/notify`).
`permission_handler 11.4.0`	Solicitação de permissões para BLE e localização.
`provider 6.1.4`	Gerenciamento de estado reativo e injeção de dependência.
`shared_preferences 2.5.3`	Armazenamento de dispositivo e parâmetros configurados.
`fluttertoast 8.2.12`	Exibição de notificações rápidas (OK/Erro).
`vibration 3.1.3`	Feedback tátil após operações válidas.
`flutter_material_pickers 3.7.0`	Seletor de modo em *bottom sheet*.
`wheel_chooser 1.1.2`	Controle rotativo para definição de tempos.

4) Fluxo de navegação: O fluxo de navegação foi projetado para ser direto e intuitivo, conduzindo o usuário desde a busca do dispositivo Bluetooth até a configuração e monitoramento em tempo real do relé temporizador.

Tela de varredura e conexão – realiza a busca por dispositivos Bluetooth, listando separadamente os já pareados e os novos. O usuário pode atualizar a lista e selecionar o relé multifuncional identificado pelo nome definido no *firmware*.

A Figura 12 mostra a tela inicial do aplicativo durante a varredura.

Tela principal de configuração e monitoramento – após a conexão bem-sucedida, são exibidos o modo ativo, os tempos configurados e o estado atual da saída. O usuário pode selecionar entre os modos de operação.

Fig. 12. Tela de pareamento Bluetooth

A Figura 13 apresenta a tela principal do aplicativo em funcionamento. Na parte superior, são exibidas as informações de status do RTM, incluindo o modo de operação ativo — neste caso, RTM-RE, configurado com tempo de 10 segundos — e o estado atual do relé, indicado como ligado.

Logo abaixo, encontra-se a lista de modos de operação disponíveis (RTM-RE, RTM-RD, RTM-CL, RTM-CD, RTM-ET e modo manual), permitindo ao usuário selecionar rapidamente a função desejada.

Na sequência, o aplicativo apresenta o diagrama temporal correspondente ao modo selecionado, facilitando a compreensão visual do comportamento do relé durante o ciclo de acionamento.

Mais abaixo, é disponibilizado o seletor de tempo no formato de *spinner*, permitindo ajustar horas, minutos e segundos de forma intuitiva. Por fim, o botão **Salvar** realiza o envio das configurações para o dispositivo, garantindo que os parâmetros definidos sejam aplicados imediatamente.

Disponibilidade do código. O código-fonte do aplicativo está nos Apêndices B–D:

- **Apêndice B — *BluetoothProvider***: conexão *BLE*, envio (*write*) e notificações (`notify`).
- **Apêndice C — *HomeController***: valida entradas, monta `'modo|t1|t2'` e realiza persistência básica.
- **Apêndice D — *HomePage***: interface principal (Material 3)

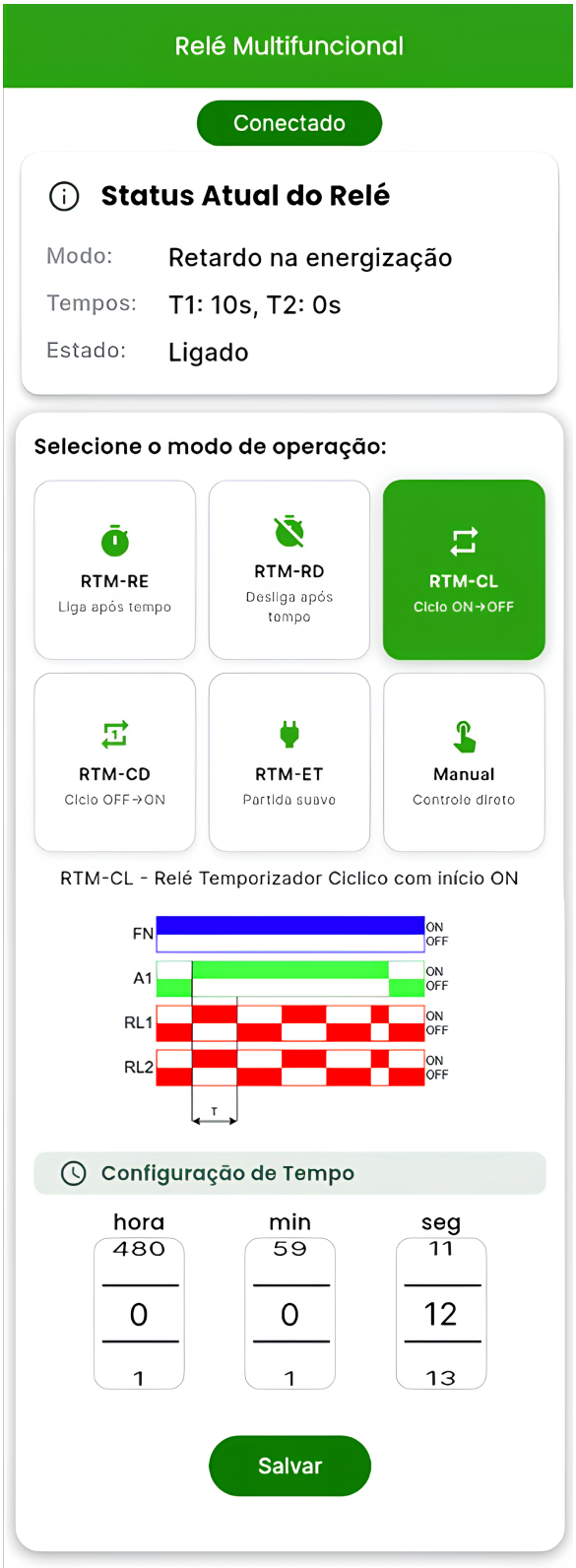

Fig. 13. Tela inicial do app

e integração reativa via *provider*.

E. Processo de Prototipagem

A prototipagem foi desenvolvida sobre uma placa fenolite perfurada do tipo ilhada com dimensões de 10 por 10 centímetros, que possibilita a montagem de circuitos de forma prática e flexível, sem a necessidade de confeccionar um circuito impresso exclusivo. Esse recurso foi escolhido por oferecer maior versatilidade na fase de testes, permitindo alterações rápidas de layout e substituição de componentes sempre que necessário.

Foram empregados os seguintes módulos: o ESP32 DevKit v1, que centraliza o processamento e a comunicação via *BLE*; o módulo HLK-PM03, responsável por converter a tensão da rede elétrica de 110 ou 220 V_{CA} alternados para 3,3V em corrente contínua; o sensor de tensão ZMPT101B, utilizado para identificar a presença de sinal no ponto de comando; e o módulo relé de dois canais, destinado à comutação de cargas. Adicionalmente, foram utilizados um conector borne KRE de duas vias para a entrada de energia, uma barra de pinos fêmea de quarenta vias passo 2,54 milímetros ângulo de 180 graus para facilitar a conexão com o ESP32, além de jumpers para a interligação dos módulos.

O módulo HLK-PM03 fornece saída de 3,3V, que deve ser conectada diretamente ao pino 3V3 do ESP32, e não ao pino VIN. Isso ocorre porque o pino VIN do ESP32 está associado a um regulador de tensão projetado para entradas próximas a 5 volts. Caso o HLK-PM03 fosse ligado ao VIN, haveria dupla regulação, resultando em queda de tensão abaixo do valor necessário ao funcionamento estável do microcontrolador. Dessa forma, a ligação ao pino 3V3 assegura alimentação adequada e confiável ao sistema.

Na organização do *firmware* e do *hardware*, definiu-se o uso específico de cada porta do ESP32 conforme a função do componente. O GPIO34 foi escolhido para a leitura do sensor ZMPT101B por se tratar de um pino exclusivo de entrada analógica, sem resistores internos de pull-up ou pull-down, o que garante medições estáveis após a adição de resistor externo de referência. Os GPIOs digitais foram destinados ao acionamento dos canais do módulo relé, uma vez que são capazes de fornecer os níveis lógicos compatíveis com o acionamento por optoacopladores. Para os LEDs indicadores, foram selecionados GPIOs de uso geral, permitindo sinalização clara dos modos de operação, do estado da entrada de comando e da conexão Bluetooth.

Cada LED foi ligado em série a um resistor de 220 ohms, valor definido para limitar a corrente de operação a aproximadamente 6 miliampères, considerando a alimentação de 3,3V do ESP32 e a queda de tensão direta típica de 2 volts em LEDs comuns. Essa configuração assegura boa intensidade luminosa sem sobrecarregar as saídas digitais do microcontrolador, aumentando a durabilidade dos LEDs e preservando a confiabilidade do sistema.

Durante a montagem, buscou-se manter clara a separação entre os circuitos de baixa tensão de 3,3V e a seção de potência composta pela rede elétrica e pelos relés, de forma a reduzir interferências eletromagnéticas e garantir maior segurança. Após a soldagem, todas as ligações foram inspecionadas visualmente e verificadas com testes de continuidade, assegurando a correspondência entre o diagrama de referência e a montagem física.

A Figura 14 apresenta o diagrama esquemático que orientou a construção do protótipo, enquanto a Figura 15 mostra a fotografia da montagem final realizada na placa fenolite perfurada.

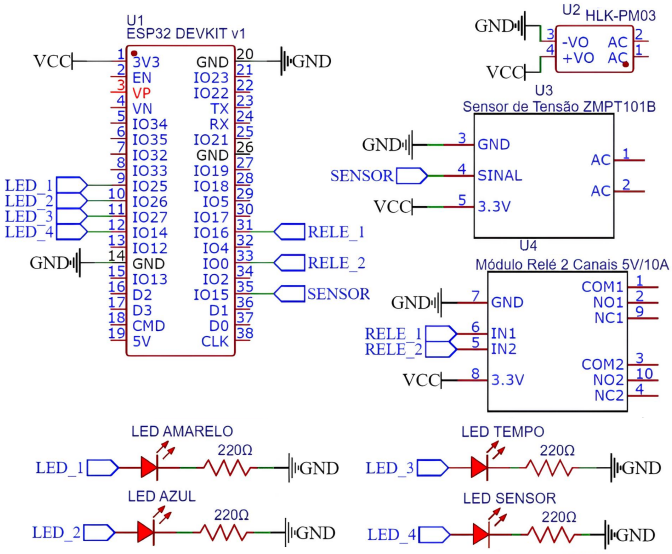

Fig. 14. Diagrama esquemático do sistema prototipado.

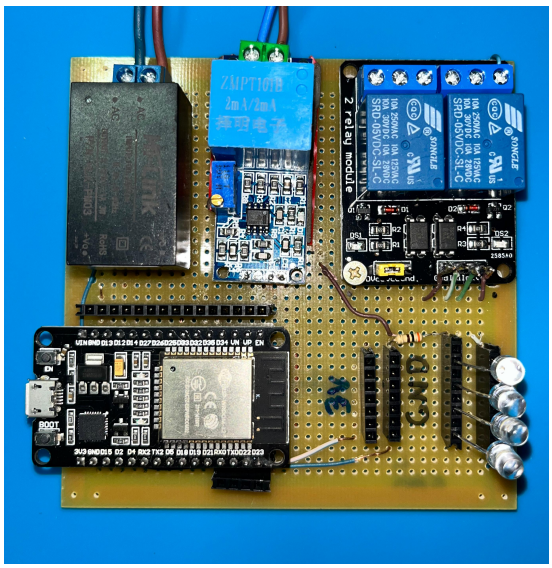

Fig. 15. Protótipo final.

Dessa forma, a prototipagem possibilitou a transição do projeto conceitual para uma implementação física funcional,

permitindo validar a integração entre *hardware* e *firmware* antes da realização dos ensaios experimentais.

F. Procedimentos de Teste

Esta subseção descreve passo a passo a validação de cada modo do relé temporizador, bem como o uso dos LEDs para diagnóstico durante os ensaios. O protocolo de testes parte do esquema elétrico e da lógica de *firmware* já definidos no trabalho.

1) Preparação e segurança:

- Conferir ligações, continuidade e isolamento entre baixa tensão e potência.
- Confirmar a alimentação do ESP32 a partir do HLK-PM03 em 3,3V ligados no pino 3V3 do módulo, e não em VIN. O pino VIN é destinado a entrada próxima de cinco volts devido ao regulador de tensão. O HLK-PM03 é o conversor especificado para obter 3,3V a partir da rede de cem a duzentos e quarenta volts.
- Preparar uma carga de teste simples, por exemplo uma lâmpada em 220 V, conectada aos terminais do relé no contato normalmente aberto e comum. Os LEDs do módulo de relé indicam o estado de cada saída e auxiliam a inspeção visual.
- Ligar A1 à fase por meio de um botão de partida e A2 ao neutro. Esta é a entrada de comando usada nos modos temporizados.
- Conectar o aplicativo via Bluetooth e enviar comandos no formato `modo|tempo`, por exemplo `3|120` para o modo cíclico com início ligado e cento e vinte segundos. O *firmware* responde com confirmações e notificações. O LED do ESP32 indica desconectado apagado e conectado aceso.

2) Diagnóstico por LEDs durante os testes:

- LED de modo azul e amarelo informa o modo ativo segundo o mapeamento: azul aceso para RTM-RE, azul piscando para RTM-RD, amarelo aceso para RTM-CL, amarelo piscando para RTM-CD, os dois acesos para RTM-ET.
- LED de entrada aceso quando A1 está energizada.
- LED de contagem piscando durante a temporização.
- LEDs do módulo de relé acesos quando cada contato é acionado.

3) Procedimento base para cada ensaio:

- Definir tempos curtos para facilitar a conferência.
- Acionar A1 para iniciar o modo.
- Cronometrar os intervalos e observar a sequência de LEDs e o estado da carga.
- Conferir no aplicativo a atualização de estado ligado e desligado no monitor.

4) Validação por modo:

RTM-RE Selecionar o modo RTM-RE no aplicativo e confirmar LED de modo azul aceso. Definir tempo T igual a cinco segundos. Acionar A1 e iniciar a cronometragem. Verificar LED de contagem piscando durante o tempo T. Ao término, os dois relés fecham e a carga acende. Ao retirar A1, ambos abrem imediatamente.

RTM-RD Selecionar RTM-RD e confirmar LED de modo azul piscando. Definir T igual a cinco segundos. Acionar A1 e observar fechamento imediato dos relés e carga ligada. Desligar A1, iniciar a cronometragem e verificar que os relés permanecem fechados por T. Ao fim de T, ambos abrem e a carga desliga.

RTM-CL Selecionar RTM-CL e confirmar LED de modo amarelo aceso. Definir T igual a três segundos. Acionar A1 e observar a sequência repetitiva: relés fechados durante Ton e abertos durante Toff enquanto A1 permanecer ativa. O LED de contagem pisca nos intervalos temporizados.

RTM-CD Selecionar RTM-CD e confirmar LED de modo amarelo piscando. Definir T inicial igual a dois segundos. Acionar A1 e verificar que os relés permanecem abertos no primeiro Toff, fecham por Ton em seguida e repetem o ciclo até A1 ser removido.

RTM-ET Selecionar RTM-ET e confirmar os dois LEDs acesos. Definir tempo TY igual a quatro segundos. Acionar A1 e verificar fechamento imediato de RL1 que representa a configuração estrela durante TY. Ao fim de TY, RL1 abre. Após 100ms, RL2 fecha e mantém a carga em configuração triângulo até A1 ser removido. Em nenhum momento RL1 e RL2 podem permanecer fechados ao mesmo tempo.

5) Observações complementares:

- A faixa de tempos configuráveis e a resolução estão de acordo com os requisitos funcionais do projeto.
- O sistema deve manter operação mesmo sem conexão Bluetooth, retornando à configuração quando necessário. Esse comportamento pode ser observado ao forçar a perda de conexão durante um ciclo de temporização.

Com estes procedimentos, cada modo foi validado quanto à sequência temporal, aos estados de saída e ao diagnóstico por LEDs, validando a conformidade ao projeto proposto.

IV. RESULTADOS E DISCUSSÕES

A implementação do protótipo do RTM configurável via aplicativo móvel demonstrou a viabilidade técnica da proposta. O sistema, composto por uma placa de circuito com microcontrolador ESP32, módulo de alimentação HLK-PM03, sensor ZMPT101B para medição de tensão e módulo relé de 2 canais 5V/10A, atendeu aos requisitos definidos na etapa de especificação.

A. Resultados Obtidos

O dispositivo apresentou funcionamento adequado em todos os modos de operação implementados: RTM-RE, RTM-RD, RTM-CL, RTM-CD e RTM-ET. Em cada modo, a sequência de acionamento foi realizada conforme os tempos programados no aplicativo, garantindo previsibilidade e confiabilidade no controle de cargas.

Durante os testes, verificou-se que a comunicação entre o ESP32 e o aplicativo desenvolvido em Flutter, por meio da tecnologia BLE, apresentou estabilidade satisfatória com latência baixa o suficiente para ser imperceptível ao usuário. A interface do aplicativo permitiu configurar com facilidade os tempos de operação, mantendo a usabilidade intuitiva. Observou-se ainda que o alcance da comunicação se manteve confiável em distâncias de até aproximadamente 10 metros entre o microcontrolador e o dispositivo móvel, sem perdas de desempenho em um ambiente residencial.

O sensor de tensão ZMPT101B também foi integrado com sucesso, viabilizando a medição de grandezas elétricas em tempo real. Essa funcionalidade contribui para ampliar a aplicabilidade do sistema em cenários de monitoramento, diferenciando-se dos temporizadores tradicionais disponíveis no mercado.

B. Discussões

Os resultados confirmam que o protótipo desenvolvido pode substituir relés temporizadores comerciais, oferecendo maior versatilidade e praticidade por meio da configuração via *smartphone*. A utilização de plataformas abertas (Arduino/C e Flutter) também favorece expansões e personalizações, possibilitando a adaptação do dispositivo a diferentes contextos de aplicação.

Algumas limitações foram identificadas. A alimentação baseada no módulo HLK-PM03, embora funcional, não representa a solução mais adequada para produção em larga escala devido ao seu custo elevado . Além disso, a dependência exclusiva do BLE restringe o alcance do sistema a curtas distâncias. Também se observou que a interface do aplicativo poderia ser aprimorada para fornecer feedback visual mais detalhado do estado dos relés em tempo real.

Ressalta-se ainda que o protótipo não foi testado em ambientes industriais, nos quais a presença de grandes cargas pode gerar ruídos e interferências eletromagnéticas. Ensaios futuros em ambientes industriais serão fundamentais para validar a robustez do sistema frente a ruídos eletromagnéticos e cargas elevadas, assegurando sua confiabilidade em condições reais de aplicação.

Ainda assim, os experimentos mostraram que a proposta cumpre os objetivos iniciais de oferecer um dispositivo compacto, reconfigurável e de fácil utilização, sendo uma alternativa competitiva aos temporizadores convencionais.

C. Perspectivas Futuras

Melhorias possíveis incluem:

- **Adição de novos modos de temporização**, ampliando a flexibilidade de aplicação.
- **Integração com IoT** para permitir o controle remoto via internet, expandindo a usabilidade para cenários de automação residencial e industrial.
- **Implementação de proteções adicionais contra sobrecorrente e sobretensão**, aumentando a confiabilidade e segurança do sistema.
- **Criação de uma PCB dedicada mais compacta**, otimizando o espaço físico e reduzindo custos de produção.
- **Sincronização dinâmica de modos entre ESP32 e aplicativo**, de forma que, ao se conectar, o microcontrolador envie ao app os modos de operação disponíveis. Isso eliminaria a dependência de atualizações frequentes no aplicativo, tornando o sistema mais escalável e independente de versões específicas do software móvel.

V. CONCLUSÃO

O RTM configurável via aplicativo móvel cumpriu os objetivos propostos, demonstrando-se uma solução viável, reconfigurável e de fácil utilização. Os testes realizados confirmaram o funcionamento adequado em todos os modos de operação, com comunicação estável via BLE e integração bem-sucedida do sensor de tensão ZMPT101B, diferenciando-se dos temporizadores tradicionais.

Apesar das limitações identificadas, como o uso do módulo HLK-PM03, de custo elevado e dimensões pouco adequadas para produção em larga escala, e a restrição do alcance de comunicação, o protótipo se mostrou funcional e competitivo em relação a soluções comerciais.

O trabalho abre caminho para aplicações práticas tanto em ambientes residenciais quanto industriais. Ensaios futuros em fábricas, sujeitos a cargas elevadas e ruídos eletromagnéticos, serão fundamentais para validar a robustez do sistema em condições reais de operação. Além disso, perspectivas de melhoria incluem a criação de uma PCB dedicada, a integração com IoT e a adição de novos modos de temporização, ampliando a versatilidade do dispositivo.

REFERÊNCIAS

[1] L. Boggia, R. Cardozo, C. Fuentes, *Automação Industrial*, Rede e-Tec Brasil, Universidade Federal de Santa Maria (UFSM), Colégio Técnico Industrial de Santa Maria, Santa Maria, RS, 2010, inclui tópicos sobre sensores, acionamento de cargas e portas lógicas.

[2] A. L. C. Ramos, J. E. L. d. Santos, *Sistema integrado de automação residencial com comunicação sem fio*, B.S. thesis, Universidade Tecnológica Federal do Paraná, 2015.

[3] Bluetooth SIG, "Bluetooth Core Specification v5.3", `https://www.bluetooth.com/specifications/`, accessed: 2025-01-20, 2022.

[4] S. J. Chapman, *Electric Machinery Fundamentals*, 5 ed., McGraw-Hill, New York, 2016.

[5] A. Barua, M. A. Al Alamin, M. S. Hossain, E. Hossain, "Security and privacy threats for bluetooth low energy in iot and wearable devices: A comprehensive survey", *IEEE Open Journal of the Communications Society*, vol. 3, 2022.

[6] D. Gorecky, M. Schmitt, M. Loskyll, D. Zühlke, "Human-machine-interaction in the industry 4.0 era", *in 2014 12th IEEE international conference on industrial informatics (INDIN)*, Ieee, 2014.

[7] WEG, "Relés Eletrônicos – Linha Modular: Aplicações e funções temporizadas", `https://static.weg.net/medias/downloadcenter/hb4/h1d/WEG-reles-temporizadores-protetores-e-de-nivel-50009830-catalogo-portugues-br-dc.pdf`, 2021.

[8] S. D. Kalamaras, M.-A. Tsitsimpikou, C. A. Tzenos, A. A. Lithourgidis, D. S. Pitsikoglou, T. A. Kotsopoulos, "A low-cost IoT system based on the ESP32 microcontroller for efficient monitoring of a pilot anaerobic biogas reactor", *Applied Sciences*, 2025.

[9] D. Hercog, "Design and implementation of ESP32-based IoT devices", *Sensors (MDPI)*, 2023.

[10] J. Tosi, "Performance evaluation of Bluetooth Low Energy", *IEEE (via PubMed)*, 2017.

[11] G. Koulouras, S. Katsoulis, F. Zantalis, "Evolution of Bluetooth Technology: BLE in the IoT Ecosystem", *Sensors*, 2025.

[12] L. Lovrić, M. Fischer, N. Röderer, A. Wünsch, "Evaluation of the Cross-Platform Framework Flutter Using the Example of a Cancer Counselling App", pp. 135–142, 2023.

[13] A. Kaczmarczyk, "Performance comparison of native and hybrid Android . . . ", *Energies*, 2022.

Apêndice
Firmware ESP32: https://github.com/arielsam567/TCC-Engenharia-Eletrica---ESP32-Arduino/tree/v2
App Flutter: https://github.com/arielsam567/TCC-Engenharia-Eletrica-Flutter
Apêndice A

```
#include "BluetoothSerial.h"
#include <Preferences.h>
#include <driver/adc.h>
#include "esp_adc_cal.h"
#include <math.h>

/*
 * ========================================
 * SISTEMA DE DEBUG CONFIGURÁVEL
 * ========================================
 *
 * Para DESATIVAR todos os Serial.println de debug:
 * - Altere a constante DEBUG_ENABLED para false
 * - Não é necessário remover nenhum código
 *
 * Para ATIVAR novamente:
 * - Altere DEBUG_ENABLED para true
 *
 * Exemplo:
 * const bool DEBUG_ENABLED = false;  // Debug desativado
 * const bool DEBUG_ENABLED = true;   // Debug ativado
 *
 * ========================================
 */

// Verifica se o Bluetooth está realmente disponível no microcontrolador
#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to enable it
#endif

// Cria um objeto para controlar o Bluetooth Serial
BluetoothSerial SerialBT;

// Definição dos pinos
const int entrada = 34;     // GPIO34 (entrada analógica do ZMPT101B)
const int rele1 = 25;      // GPIO25 (relé 1)
const int rele2 = 32;      // GPIO32 (relé 2)
const int ledBluetooh = 2;       // GPIO2 (relé 3) - Controlado automaticamente pelo status do Bluetooth

// Definição dos pinos dos LEDs indicadores
const int ledAzul = 23;     // GPIO23 - LED Azul (Modos 1, 2 e 5)
const int ledAmarelo = 22;  // GPIO22 - LED Verde (Modos 3, 4 e 5)
const int ledVermelho = 19; // GPIO21 - LED Vermelho (Indicador de tensão AC)
const int ledBranco = 18;   // GPIO18 - LED Branco (Indicador de contagem de tempo e operação cíclica)

const char* btName = "RELÉ MULTIFUNCIONAL - TCC ";

// Configurações do sensor ZMPT101B - ADC ESP32
const adc1_channel_t ZMPT101B_CHANNEL = ADC1_CHANNEL_6; // GPIO34
```

```
const adc_atten_t ZMPT101B_ATTEN = ADC_ATTEN_DB_11;
const adc_bits_width_t ZMPT101B_WIDTH = ADC_WIDTH_BIT_12;
const int ZMPT101B_SAMPLES = 2000;         // Número de amostras para cálculo RMS
const float ZMPT101B_THRESHOLD_VRMS = 15.0f; // Threshold em Volts RMS para detectar tensão AC
const float ZMPT101B_MODULE_MEASURED_VRMS = 1.663f; // Tensão medida no pino OUT (Vrms)
const float ZMPT101B_NETWORK_VOLTAGE = 220.0f; // Tensão da rede (V)

// Variável para caracterização do ADC
esp_adc_cal_characteristics_t adc_chars;

// Estados da máquina de estados
enum Estados {
  MODO_1 = 1,      // Retardo na energização
  MODO_2 = 2,      // Retardo na desenergização
  MODO_3 = 3,      // Cíclico com início ligado
  MODO_4 = 4,      // Cíclico com início desligado
  MODO_5 = 5,      // Partida estrela-triângulo
  MODO_6 = 6       // Alteração via comando bluetooh
};

// Estrutura para configuração dos relés
struct ConfigReles {
  int modo;
  unsigned long tempo1;  // em segundos (usando sempre tempo1)
};

// ========================================
// CONSTANTES DO SISTEMA
// ========================================

// Valores default para configuração
const int DEFAULT_MODO = 1;
const unsigned long DEFAULT_TEMPO1 = 300; // 5 minutos em segundos

// Valores de validação para modo
const int MIN_MODO = 1;
const int MAX_MODO = 5;

// Valores de validação para tempo
const unsigned long MAX_TEMPO = 1728000; // 20 dias em segundos
const unsigned long SEGUNDOS_POR_DIA = 86400; // Segundos em um dia

// Tempos de controle
const unsigned long TEMPO_STATUS_AUTOMATICO = 3000; // 3 segundos para envio automático de status
const unsigned long TEMPO_LOG_MODO = 5000; // 5 segundos para log do modo
const unsigned long DELAY_LOOP = 50; // Delay do loop principal em milissegundos
const unsigned long CONVERSOR_SEGUNDOS = 1000; // Conversor de millis() para segundos

// Configurações de comunicação
const unsigned long VELOCIDADE_SERIAL = 115200; // Velocidade do Serial em bauds

// Valores de controle
const int VALOR_INVALIDO = -1; // Valor para indicar erro ou inválido
const int VALOR_NAO_ENCONTRADO = -1; // Valor para indicar que não foi encontrado
```

```
const unsigned long VALOR_INICIAL = 0; // Valor inicial para variáveis de tempo

// Tempo de transição estrela-triângulo (em milissegundos)
const unsigned long TEMPO_TRANSICAO_ESTRELA_TRIANGULO = 150;

// Configurações dos LEDs
const unsigned long TEMPO_PISCA_LED = 500; // Tempo de piscada dos LEDs em milissegundos
const unsigned long TEMPO_PISCA_LED_BRANCO_RAPIDO = 200; // Tempo de piscada rápida do LED branco (200ms)
const unsigned long TEMPO_PISCA_LED_BRANCO_LENTO = 500;  // Tempo de piscada lenta do LED branco (500ms)

// Controle de debug - altere para false para desativar todos os Serial.println
const bool DEBUG_ENABLED = true;

// ========================================
// VARIÁVEIS GLOBAIS
// ========================================

bool deviceConnected = false;
bool oldDeviceConnected = false;
String comandoRecebido = "";

Estados estadoAtual = (Estados)DEFAULT_MODO; // Inicializa com modo padrão
ConfigReles config;
unsigned long tempoInicio = 0;
unsigned long tempoAtual = 0;
bool relesLigados = false;
bool modoEstrela = true; // true = estrela, false = triangulo

// Variáveis para controle de alteração manual
bool relesLigadosAnterior = false;
bool modoEstrelaAnterior = true;
unsigned long ultimaAlteracaoManual = 0;

// Variáveis para controle da transição estrela-triângulo (MODO 5)
bool transicaoEstrelaTrianguloEmAndamento = false;
unsigned long tempoInicioTransicao = 0;

// Variáveis para validação da entrada (anti-ruído)
const int VALIDACAO_ENTRADA_COUNT = 3; // Número de leituras consecutivas necessárias
int contadorEntradaAtiva = 0;
int contadorEntradaInativa = 0;
bool entradaValidada = false; // Estado validado da entrada

// Variável para controle de mudança de status da entrada
bool entradaAtivaAnterior = false;

// Variáveis para controle de status automático
unsigned long tempoConexao = 0;
bool statusEnviado = false;

// Variáveis para controle de tempo no modo 1
bool temporizadorModo1Iniciado = false;

// Variáveis para controle dos LEDs
```

```
unsigned long ultimoTempoPiscaLed = 0;
bool estadoPiscaLed = false;

// Variáveis para controle do LED branco
unsigned long ultimoTempoPiscaLedBranco = 0;
bool estadoPiscaLedBranco = false;

// Variável para controlar estado atual do relé1
bool rele1Atual = false;

Preferences preferences;

// Declarações de funções (forward declarations)
void iniciarModo();
void executarMaquinaEstados();
void ligarRele(bool ligar);
void ligarReleEstrela();
void ligarReleTriangulo();
void enviarResposta(String resposta);
void enviarNotificacao(String notificacao);
bool processarConfiguracao(String comando);
void salvarConfiguracao();
void carregarConfiguracao();
void restaurarEstadoSalvo();
void verificarConexaoBluetooth();
void processarComandosRecebidos();
void verificarAlteracaoManual();
void enviarNotificacaoAlteracaoManual(bool novoEstado, String tipoAlteracao);
void enviarStatusAutomatico();
void verificarStatusEntrada();
void calibrarZMPT101B();
bool ajustarThresholdZMPT101B(String comando);
void debugPrint(String mensagem);
void controlarLEDs();
void configurarLEDs();
void controlarLEDBranco();

void setup() {
  Serial.begin(VELOCIDADE_SERIAL);
  debugPrint("=== INICIANDO RELAY TIMER ===");

  // Configuração do ADC para ZMPT101B
  debugPrint("Configurando ADC para ZMPT101B...");
  adc1_config_width(ZMPT101B_WIDTH);
  adc1_config_channel_atten(ZMPT101B_CHANNEL, ZMPT101B_ATTEN);
  esp_adc_cal_characterize(ADC_UNIT_1, ZMPT101B_ATTEN, ZMPT101B_WIDTH, 1100, &adc_chars);
  delay(100);
  debugPrint("ADC configurado com sucesso");

  // Configuração dos pinos
  // GPIO34 é entrada analógica por padrão, não precisa de pinMode
  pinMode(rele1, OUTPUT);
  pinMode(rele2, OUTPUT);
  pinMode(ledBluetooh, OUTPUT);
```

```
  // Configuração dos LEDs indicadores
  configurarLEDs();

  // Inicializar relés desligados
  digitalWrite(rele1, HIGH);
  digitalWrite(rele2, HIGH);
  rele1Atual = true; // Relé1 HIGH = desligado
  digitalWrite(ledBluetooh, LOW); // LED Bluetooth inicia desligado (sem conexão)

  // Inicializar variável de controle da entrada
  entradaAtivaAnterior = validarEntrada();
  debugPrint("Status inicial da entrada: " + String(entradaAtivaAnterior ? "ATIVA" : "INATIVA"));

  // Carregar configuração salva
  debugPrint("Iniciando carregamento da configuração");
  carregarConfiguracao();

  // Inicializar Bluetooth
  SerialBT.begin(btName);

  // Verificar status inicial do Bluetooth e configurar LED
  deviceConnected = SerialBT.hasClient();
  digitalWrite(ledBluetooh, deviceConnected ? HIGH : LOW);
  debugPrint("Status inicial Bluetooth: " + String(deviceConnected ? "CONECTADO" : "DESCONECTADO"));

  debugPrint("Relay Timer iniciado - Modo " + String(config.modo) + " - T1: " + String(config.tempo1) + "s");
  debugPrint("Estado atual após inicialização: " + String(estadoAtual));
}

void loop() {
  // Verificar conexão Bluetooth
  verificarConexaoBluetooth();

  // Processar comandos recebidos
  processarComandosRecebidos();

  // Verificar alterações manuais nos relés
  verificarAlteracaoManual();

  // Executar máquina de estados
  executarMaquinaEstados();

  // Controlar LEDs indicadores
  controlarLEDs();

  // delay(DELAY_LOOP); // pausa para estabilidade
}

void verificarConexaoBluetooth() {
  // Verificar se há dispositivos conectados
  bool conectado = SerialBT.hasClient();

  if (conectado != deviceConnected) {
```

```
    deviceConnected = conectado;

    if (deviceConnected) {
      // Ligar LED Bluetooth quando conectar
      digitalWrite(ledBluetooh, HIGH);
      debugPrint("Bluetooth conectado - LED Bluetooth ligado");

      // Inicializar controle de status automático
      tempoConexao = millis();
      statusEnviado = false;

      enviarNotificacao("CONECTADO");
    } else {
      // Desligar LED Bluetooth quando desconectar
      digitalWrite(ledBluetooh, LOW);
      debugPrint("Bluetooth desconectado - LED Bluetooth desligado");

      // Resetar controle de status automático
      statusEnviado = false;
    }

    oldDeviceConnected = deviceConnected;
  }

  // Verificar se deve enviar status automático
  if (deviceConnected && !statusEnviado && (millis() - tempoConexao) >= TEMPO_STATUS_AUTOMATICO) {
    debugPrint("Enviando status automático após conexão Bluetooth");
    enviarStatusAutomatico();
    statusEnviado = true;
  }
}

void processarComandosRecebidos() {
  // Se há dados disponíveis no Bluetooth
  if (SerialBT.available()) {
    char c = SerialBT.read();

    // Adicionar caractere ao comando
    comandoRecebido += c;

    // Verificar se o comando termina com _END
    if (comandoRecebido.endsWith("_END")) {
      // Comando completo recebido - remover _END
      String comandoProcessado = comandoRecebido.substring(0, comandoRecebido.length() - 4);
      debugPrint("Comando recebido: " + comandoProcessado);

      if (comandoProcessado == "X" || comandoProcessado == "6") {
        // Comando para alterar estado dos relés
        debugPrint("Comando de controle de relés recebido: " + comandoProcessado);
        if (relesLigados) {
          ligarRele(false); // Desliga os relés
        } else {
          ligarRele(true);  // Liga os relés
        }
```

```
      enviarResposta("OK");
    } else if (comandoProcessado == "CALIBRAR") {
      // Comando para calibrar o sensor ZMPT101B
      debugPrint("Comando de calibração recebido");
      calibrarZMPT101B();
      enviarResposta("CALIBRACAO_INICIADA");
    } else if (ajustarThresholdZMPT101B(comandoProcessado)) {
      // Comando para ajustar threshold do ZMPT101B
      debugPrint("Comando de ajuste de threshold processado");
      // A resposta já é enviada na função ajustarThresholdZMPT101B
    } else if (processarConfiguracao(comandoProcessado)) {
      // Comando de configuração válido
      debugPrint("Comando de configuração válido - aplicando alterações");
      salvarConfiguracao();
      debugPrint("Restaurando estado baseado na nova configuração");
      restaurarEstadoSalvo(); // Restaurar estado baseado na nova configuração
      debugPrint("Iniciando modo com nova configuração");
      iniciarModo(); // Aplicar a nova configuração imediatamente
      enviarResposta("OK");
      debugPrint("Configuração aplicada com sucesso");
    } else {
      // Comando inválido ou bloqueado por segurança
      debugPrint("Configuração não pôde ser aplicada");
    }

    comandoRecebido = ""; // Limpar comando
  }
 }
}

bool processarConfiguracao(String comando) {
  debugPrint("Processando configuração: " + comando);

  // Verificar se é comando de configuração (formato: "modo|tempo1|tempo1")
  int pipe1 = comando.indexOf('|');
  int pipe2 = comando.indexOf('|', pipe1 + 1);

  if (pipe1 == VALOR_INVALIDO || pipe2 == VALOR_INVALIDO) {
    debugPrint("Formato inválido: " + comando);
    return false;
  }

  int modo = comando.substring(0, pipe1).toInt();
  unsigned long t1 = comando.substring(pipe1 + 1, pipe2).toInt();
  unsigned long t1_aux = comando.substring(pipe2 + 1).toInt();

  debugPrint("Valores extraídos - Modo: " + String(modo) + ", T1: " + String(t1) + ", T1_aux: " + String(t1_a

  // Validar modo
  if (modo < MIN_MODO || modo > MAX_MODO) {
    debugPrint("Modo inválido: " + String(modo) + " (deve ser entre " + String(MIN_MODO) + " e " + String(MAX
    return false;
  }
```

```
  // Validar tempos (máximo 20 dias)
  if (t1 > MAX_TEMPO || t1_aux > MAX_TEMPO) {
    debugPrint("Tempo muito longo: máximo " + String(MAX_TEMPO / SEGUNDOS_POR_DIA) + " dias");
    return false;
  }

  // VERIFICAÇÃO DE SEGURANÇA: Não permitir alterar modo se entrada estiver ativa
  bool entradaAtiva = validarEntrada();
  debugPrint("Status VALIDADO da entrada durante validação: " + String(entradaAtiva ? "ATIVA" : "INATIVA"));

  if (entradaAtiva) {
    debugPrint("Não é possível alterar modo com entrada ativa!");

    // Enviar mensagem de erro via Bluetooth
    if (deviceConnected) {
      enviarResposta("ERR: Entrada ativa. Desligue a entrada para alterar o modo.");
      enviarNotificacao("ALERTA: Modo não pode ser alterado com entrada ativa!");
    }

    return false;
  }

  // Verificar se está tentando alterar para um modo diferente
  if (modo != config.modo) {
    debugPrint("Alterando modo: " + String(config.modo) + " → " + String(modo));
  }

  debugPrint("Configuração atual antes da alteração - Modo: " + String(config.modo) + ", T1: " + String(confi

  config.modo = modo;
  config.tempo1 = t1;

  debugPrint("Configuração válida - Modo: " + String(modo) + ", T1: " + String(t1) + "s");
  debugPrint("Nova configuração definida - Modo: " + String(config.modo) + ", T1: " + String(config.tempo1));

  return true;
}

void salvarConfiguracao() {
  debugPrint("Salvando configuração na EEPROM - Modo: " + String(config.modo) + ", T1: " + String(config.temp

  preferences.begin("relaytimer", false);
  preferences.putInt("modo", config.modo);
  preferences.putULong("tempo1", config.tempo1);
  preferences.end();

  debugPrint("Configuração salva na EEPROM com sucesso");

  // Verificar se foi salva corretamente
  preferences.begin("relaytimer", true);
  int modoSalvo = preferences.getInt("modo", VALOR_NAO_ENCONTRADO);
  unsigned long tempoSalvo = preferences.getULong("tempo1", VALOR_INICIAL);
  preferences.end();
```

```
  debugPrint("Verificação - Modo salvo: " + String(modoSalvo) + ", Tempo salvo: " + String(tempoSalvo));
}

void carregarConfiguracao() {
  debugPrint("Iniciando carregamento da configuração da EEPROM");

  preferences.begin("relaytimer", true);
  config.modo = preferences.getInt("modo", DEFAULT_MODO);
  config.tempo1 = preferences.getULong("tempo1", DEFAULT_TEMPO1);
  preferences.end();

  debugPrint("Configuração carregada da EEPROM - Modo: " + String(config.modo) + ", T1: " + String(config.tem

  // Restaurar o estado atual baseado na configuração carregada
  debugPrint("Chamando restaurarEstadoSalvo() para aplicar configuração carregada");
  restaurarEstadoSalvo();
}

void restaurarEstadoSalvo() {
  debugPrint("Restaurando estado salvo - Modo configurado: " + String(config.modo));

  // Se há uma configuração válida salva, restaurar o estado
  if (config.modo >= MIN_MODO && config.modo <= MAX_MODO) {
    estadoAtual = (Estados)config.modo;
    debugPrint("Estado restaurado para MODO " + String(estadoAtual));

    // Configurar estado inicial dos relés baseado no modo restaurado
    switch (estadoAtual) {
      case MODO_1: // Retardo na energização - inicia desligado
        relesLigados = false;
        debugPrint("MODO 1 configurado - relés iniciam DESLIGADOS");
        break;
      case MODO_2: // Retardo na desenergização - inicia ligado
        relesLigados = true;
        modoEstrela = false; // Inicializar como false para MODO 2
        debugPrint("MODO 2 configurado - relés iniciam LIGADOS");
        break;
      case MODO_3: // Cíclico com início ligado
        relesLigados = true;
        debugPrint("MODO 3 configurado - relés iniciam LIGADOS");
        break;
      case MODO_4: // Cíclico com início desligado
        relesLigados = false;
        debugPrint("MODO 4 configurado - relés iniciam DESLIGADOS");
        break;
      case MODO_5: // Partida estrela-triângulo - inicia desligado
        relesLigados = false;
        modoEstrela = true;
        debugPrint("MODO 5 configurado - relés iniciam DESLIGADOS (modo estrela)");
        break;
    }

    // Atualizar estados anteriores para detecção de alteração manual
    relesLigadosAnterior = relesLigados;
```

```
    modoEstrelaAnterior = modoEstrela;

    debugPrint("Estados anteriores atualizados - relesLigados: " + String(relesLigados) + ", modoEstrela: " +
  } else {
    // Se não há configuração, usar modo padrão
    estadoAtual = (Estados)DEFAULT_MODO;
    relesLigados = false;
    modoEstrela = true;
    debugPrint("Usando MODO " + String(DEFAULT_MODO) + " como padrão - configuração inválida");
  }
}

void iniciarModo() {
  debugPrint("Iniciando modo - Configuração atual: modo=" + String(config.modo) + ", tempo1=" + String(config

  estadoAtual = (Estados)config.modo;
  tempoInicio = millis();
  tempoAtual = 0;

  debugPrint("Estado atual definido para: " + String(estadoAtual));

  // Configurar estado inicial baseado no modo (consistente com restaurarEstadoSalvo)
  switch (estadoAtual) {
    case MODO_1: // Retardo na energização - inicia desligado
      relesLigados = false;
      modoEstrela = true;
      debugPrint("MODO 1 selecionado - relés iniciam DESLIGADOS");
      break;
    case MODO_2: // Retardo na desenergização - inicia ligado
      relesLigados = true;
      modoEstrela = false; // Inicializar como false para MODO 2
      debugPrint("MODO 2 selecionado - relés iniciam LIGADOS");
      break;
    case MODO_3: // Cíclico com início ligado
      relesLigados = true;
      modoEstrela = true;
      debugPrint("MODO 3 selecionado - relés iniciam LIGADOS");
      break;
    case MODO_4: // Cíclico com início desligado
      relesLigados = false;
      modoEstrela = true;
      debugPrint("MODO 4 selecionado - relés iniciam DESLIGADOS");
      break;
    case MODO_5: // Partida estrela-triângulo - inicia desligado
      relesLigados = false;
      modoEstrela = true;
      debugPrint("MODO 5 selecionado - relés iniciam DESLIGADOS (modo estrela)");
      break;
  }

  // Inicializar estados anteriores para detecção de alteração manual
  relesLigadosAnterior = relesLigados;
  modoEstrelaAnterior = modoEstrela;
```

```
  debugPrint("Iniciando modo " + String(config.modo) + " - Estado inicial: " + String(relesLigados ? "LIGADO"

  // Verificar se a entrada está ativa antes de configurar estado inicial
  bool entradaAtiva = validarEntrada();
  debugPrint("Status VALIDADO da entrada: " + String(entradaAtiva ? "ATIVA" : "INATIVA"));

  if (!entradaAtiva) {
    // Entrada desacionada - todos os modos iniciam com relés desligados
    ligarRele(false);
    debugPrint("Entrada inativa - relés desligados");
    return;
  }

  // Aplicar estado inicial baseado no modo (apenas quando entrada ativa)
  ligarRele(relesLigados);
  debugPrint("Entrada ativa - aplicando estado inicial do modo " + String(config.modo));
}

void executarMaquinaEstados() {
  // Validar entrada com anti-ruído
  bool entradaAtiva = validarEntrada();

  // Log apenas quando o status validado da entrada mudar
  if (entradaAtiva != entradaAtivaAnterior) {
    unsigned long timestamp = millis() / CONVERSOR_SEGUNDOS;
    debugPrint("Mudança de status VALIDADO da entrada: " + String(entradaAtiva ? "ATIVA" : "INATIVA") + " (se
    entradaAtivaAnterior = entradaAtiva;

    // Resetar temporizador quando entrada mudar de status
    if (estadoAtual == MODO_1) {
      temporizadorModo1Iniciado = false;
      tempoInicio = millis();
      tempoAtual = 0;
    }
  }

  tempoAtual = (millis() - tempoInicio) / CONVERSOR_SEGUNDOS; // converter para segundos

  // Log do modo atual sendo executado (a cada 5 segundos para não poluir o log)
  static unsigned long ultimoLogModo = 0;
  if (millis() - ultimoLogModo >= TEMPO_LOG_MODO) {
    debugPrint("Executando MODO " + String(estadoAtual) + " - Tempo atual: " + String(tempoAtual) + "s");
    ultimoLogModo = millis();
  }

  switch (estadoAtual) {
    case MODO_1: // Retardo na energização
      if (!entradaAtiva) {
        // Entrada desacionada - desligar relés imediatamente
        if (relesLigados) {
          unsigned long timestamp = millis() / 1000;
          debugPrint("MODO 1: Entrada desacionada - desligando relés imediatamente (segundo " + String(timest
          ligarRele(false);
          // Resetar temporizador para próxima operação
```

```
      temporizadorModo1Iniciado = false;
      tempoInicio = millis();
      tempoAtual = 0;
    }
  } else if (entradaAtiva && !relesLigados) {
    // Entrada acionada e relés desligados - controlar temporizador
    if (!temporizadorModo1Iniciado) {
      // Primeira vez que entrada foi acionada - iniciar temporizador
      tempoInicio = millis();
      tempoAtual = 0;
      temporizadorModo1Iniciado = true;
            unsigned long timestamp = millis() / CONVERSOR_SEGUNDOS;
  debugPrint("MODO 1: Entrada acionada - iniciando temporizador de " + String(config.tempo1) + "s (segund
    } else if (tempoAtual >= config.tempo1) {
      // Tempo atingido - ligar relés
                  unsigned long timestamp = millis() / CONVERSOR_SEGUNDOS;
        debugPrint("MODO 1 CONCLUÍDO - Relés ligados após " + String(config.tempo1) + "s (segundo " + Str
      ligarRele(true);
    }
  }
  break;

case MODO_2: // Retardo na desenergização
  if (!entradaAtiva) {
    // Entrada desacionada - iniciar contagem para desligamento
    if (relesLigados && !modoEstrela) { // Usar modoEstrela como flag de controle
      // Primeira vez que entrada foi desacionada - iniciar temporizador
      modoEstrela = true; // Marcar que temporizador foi iniciado
      tempoInicio = millis();
      tempoAtual = 0;
      debugPrint("MODO 2: Entrada desacionada - iniciando contagem para desligamento em " + String(config
    } else if (relesLigados && modoEstrela && tempoAtual >= config.tempo1) {
      // Tempo de retardo atingido - desligar relés
      debugPrint("MODO 2 CONCLUÍDO - Relés desligados após " + String(config.tempo1) + "s");
      ligarRele(false);
      estadoAtual = MODO_2; // Permanecer no MODO_2
    }
  } else if (entradaAtiva) {
    // Entrada acionada - ligar relés imediatamente e resetar temporizador
    if (!relesLigados) {
      debugPrint("MODO 2: Entrada acionada - ligando relés imediatamente");
      ligarRele(true);
      modoEstrela = false; // Resetar flag de controle
      tempoInicio = millis();
      tempoAtual = 0;
    }
  }
  break;

case MODO_3: // Cíclico com início ligado
  if (!entradaAtiva) {
    // Entrada desacionada - desligar relés imediatamente e interromper ciclo
    if (relesLigados) {
      debugPrint("MODO 3: Entrada desacionada - desligando relés imediatamente");
```

```
      ligarRele(false);
      // Resetar ciclo para próxima ativação
      tempoInicio = millis();
      tempoAtual = 0;
    }
  } else if (entradaAtiva) {
    // Entrada acionada - controlar ciclo
    if (relesLigados && tempoAtual >= config.tempo1) {
      // Tempo T1 atingido - desligar relés e iniciar contagem para ligar
      debugPrint("MODO 3: Desligando relés após " + String(config.tempo1) + "s");
      ligarRele(false);
      tempoInicio = millis();
      tempoAtual = 0;
      debugPrint("MODO 3: Relés desligados, aguardando " + String(config.tempo1) + "s para ligar");
    } else if (!relesLigados && tempoAtual >= config.tempo1) {
      // Tempo T1 atingido - ligar relés e reiniciar ciclo
      debugPrint("MODO 3: Ligando relés após " + String(config.tempo1) + "s");
      ligarRele(true);
      tempoInicio = millis();
      tempoAtual = 0;
      debugPrint("MODO 3: Relés ligados, aguardando " + String(config.tempo1) + "s para desligar");
    }
  }
  break;

case MODO_4: // Cíclico com início desligado
  if (!entradaAtiva) {
    // Entrada desacionada - desligar relés imediatamente e interromper ciclo
    if (relesLigados) {
      debugPrint("MODO 4: Entrada desacionada - desligando relés imediatamente");
      ligarRele(false);
      // Resetar ciclo para próxima ativação
      tempoInicio = millis();
      tempoAtual = 0;
    }
  } else if (entradaAtiva) {
    // Entrada acionada - controlar ciclo
    if (!relesLigados && tempoAtual >= config.tempo1) {
      // Tempo T1 atingido - ligar relés e iniciar contagem para desligar
      debugPrint("MODO 4: Ligando relés após " + String(config.tempo1) + "s");
      ligarRele(true);
      tempoInicio = millis();
      tempoAtual = 0;
      debugPrint("MODO 4: Relés ligados, aguardando " + String(config.tempo1) + "s para desligar");
    } else if (relesLigados && tempoAtual >= config.tempo1) {
      // Tempo T1 atingido - desligar relés e reiniciar ciclo
      debugPrint("MODO 4: Desligando relés após " + String(config.tempo1) + "s");
      ligarRele(false);
      tempoInicio = millis();
      tempoAtual = 0;
      debugPrint("MODO 4: Relés desligados, aguardando " + String(config.tempo1) + "s para ligar");
    }
  }
  break;
```

```
    case MODO_5: // Partida estrela-triângulo
      if (!entradaAtiva) {
        // Entrada desacionada - desligar relés imediatamente e resetar transição
        if (relesLigados) {
          debugPrint("MODO 5: Entrada desacionada - desligando relés imediatamente");
          ligarRele(false);
          modoEstrela = true; // Reset para modo estrela
          transicaoEstrelaTrianguloEmAndamento = false; // Cancelar transição em andamento
          tempoInicio = millis();
          tempoAtual = 0;
        }
      } else if (entradaAtiva && !relesLigados && !transicaoEstrelaTrianguloEmAndamento) {
        // Entrada acionada e relés desligados - iniciar modo estrela
        debugPrint("MODO 5: Entrada acionada - iniciando modo estrela");
        ligarReleEstrela();
        tempoInicio = millis();
        tempoAtual = 0;
      } else if (entradaAtiva && modoEstrela && !transicaoEstrelaTrianguloEmAndamento && tempoAtual >= config
        // Iniciar transição para triângulo após tempo configurado
        debugPrint("→MODO 5: Iniciando transição para triângulo após " + String(config.tempo1) + "s");
        debugPrint("Desligando Relé 1 para transição");

        // Iniciar transição - desligar Relé 1
        digitalWrite(rele1, HIGH);
        transicaoEstrelaTrianguloEmAndamento = true;
        tempoInicioTransicao = millis();
        debugPrint("Transição iniciada - aguardando " + String(TEMPO_TRANSICAO_ESTRELA_TRIANGULO) + "ms");
      } else if (entradaAtiva && transicaoEstrelaTrianguloEmAndamento) {
        // Verificar se tempo de transição foi atingido
        unsigned long tempoTransicao = millis() - tempoInicioTransicao;
        if (tempoTransicao >= TEMPO_TRANSICAO_ESTRELA_TRIANGULO) {
          // Tempo de transição atingido - completar transição
          debugPrint("MODO 5: Transição concluída - ligando modo triângulo");

          // Ligar Relé 2 (modo triângulo)
          ligarReleTriangulo();
          modoEstrela = false;
          transicaoEstrelaTrianguloEmAndamento = false;
          tempoInicio = millis();
          tempoAtual = 0;
          debugPrint("Modo triângulo ativado com sucesso");
        }
      }
      break;
  }
}

void verificarAlteracaoManual() {
  // Verificar se houve alteração manual nos relés
  if (relesLigados != relesLigadosAnterior) {
    unsigned long timestamp = millis() / CONVERSOR_SEGUNDOS;
    debugPrint("Alteração manual detectada (segundo " + String(timestamp) + "):");
    debugPrint("   Estado anterior: " + String(relesLigadosAnterior ? "LIGADO" : "DESLIGADO"));
```

```
    debugPrint("   Estado atual: " + String(relesLigados ? "LIGADO" : "DESLIGADO"));

    if (deviceConnected) {
      enviarNotificacaoAlteracaoManual(relesLigados, "RELE");
    }

    relesLigadosAnterior = relesLigados;
    ultimaAlteracaoManual = millis();
  }

  // Verificar alteração no modo estrela-triângulo (apenas para modo 5)
  if (estadoAtual == MODO_5 && modoEstrela != modoEstrelaAnterior) {
    debugPrint("Alteração manual detectada (Modo 5):");
    debugPrint("   Modo anterior: " + String(modoEstrelaAnterior ? "ESTRELA" : "TRIÂNGULO"));
    debugPrint("   Modo atual: " + String(modoEstrela ? "ESTRELA" : "TRIÂNGULO"));

    if (deviceConnected) {
      enviarNotificacaoAlteracaoManual(modoEstrela, "ESTRELA_TRIANGULO");
    }

    modoEstrelaAnterior = modoEstrela;
    ultimaAlteracaoManual = millis();
  }
}

void enviarNotificacaoAlteracaoManual(bool novoEstado, String tipoAlteracao) {
  unsigned long timestamp = millis() / CONVERSOR_SEGUNDOS; // timestamp em segundos
  String notificacao = "MANUAL|" + String(timestamp) + "|" + tipoAlteracao + "|" + String(novoEstado ? "ON" :

  debugPrint("Notificação de alteração manual: " + notificacao);
  debugPrint("Detalhes - Tipo: " + tipoAlteracao + ", Estado: " + String(novoEstado ? "ON" : "OFF") + ", Time

  enviarNotificacao(notificacao);
}

void ligarRele(bool ligar) {
  debugPrint("Função ligarRele chamada - Parâmetro: " + String(ligar ? "LIGAR" : "DESLIGAR"));
  debugPrint("Estado anterior dos relés: " + String(relesLigados ? "LIGADO" : "DESLIGADO"));

  relesLigados = ligar;

  if (ligar) {
    digitalWrite(rele1, LOW);
    digitalWrite(rele2, LOW);
    rele1Atual = false; // Relé1 LOW = ligado
    // Não alterar saida3 (porta 2) - ela é controlada pelo status do Bluetooth
    unsigned long timestamp = millis() / CONVERSOR_SEGUNDOS;
    debugPrint("Relés ligados (GPIO25 e GPIO32) - segundo " + String(timestamp));
    enviarNotificacao("ON");
  } else {
    digitalWrite(rele1, HIGH);
    digitalWrite(rele2, HIGH);
    rele1Atual = true; // Relé1 HIGH = desligado
    // Não alterar saida3 (porta 2) - ela é controlada pelo status do Bluetooth
```

```
    unsigned long timestamp = millis() / CONVERSOR_SEGUNDOS;
    debugPrint("Relés desligados (GPIO25 e GPIO32) - segundo " + String(timestamp));
    enviarNotificacao("OFF");
  }

  debugPrint("Estado atual dos relés: " + String(relesLigados ? "LIGADO" : "DESLIGADO"));
}

void ligarReleEstrela() {
  // Modo estrela: apenas relé 1 ligado
  digitalWrite(rele2, HIGH);
  digitalWrite(rele1, LOW);
  rele1Atual = false; // Relé1 LOW = ligado
  // Não alterar saida3 (porta 2) - ela é controlada pelo status do Bluetooth
  relesLigados = true;
  debugPrint("Modo estrela ativado - Relé 1 ligado, Relé 2 desligado");
  enviarNotificacao("ON");
}

void ligarReleTriangulo() {
  // Modo triângulo: apenas relé 2 ligado
  digitalWrite(rele2, LOW);
  digitalWrite(rele1, HIGH);
  rele1Atual = true; // Relé1 HIGH = desligado
  // Não alterar saida3 (porta 2) - ela é controlada pelo status do Bluetooth
  relesLigados = true;
  debugPrint("Modo triângulo ativado - Relé 1 desligado, Relé 2 ligado");
  enviarNotificacao("ON");
}

void enviarResposta(String resposta) {
  if (deviceConnected) {
    SerialBT.println(resposta);
    debugPrint("Resposta enviada via Bluetooth: " + resposta);
  } else {
    debugPrint("Resposta não enviada - Bluetooth desconectado: " + resposta);
  }
}

void enviarNotificacao(String notificacao) {
  if (deviceConnected) {
    SerialBT.println(notificacao);
    debugPrint("Notificação enviada via Bluetooth: " + notificacao);
  } else {
    debugPrint("Notificação não enviada - Bluetooth desconectado: " + notificacao);
  }
}

void enviarStatusAutomatico() {
  debugPrint("Preparando status automático - Modo atual: " + String(estadoAtual) + ", Config modo: " + String

  // Verificar e informar status da entrada
  verificarStatusEntrada();
```

```
  // Determinar estado atual dos relés
  String estadoReles = "DESLIGADO";
  if (relesLigados) {
    if (estadoAtual == MODO_5 && !modoEstrela) {
      estadoReles = "TRIANGULO";
    } else if (estadoAtual == MODO_5 && modoEstrela) {
      estadoReles = "ESTRELA";
    } else {
      estadoReles = "LIGADO";
    }
  }

  // Determinar nome do modo - sempre mostrar o modo atual
  String nomeModo = "DESCONHECIDO";
  switch (estadoAtual) {
    case MODO_1: nomeModo = "RETARDO_ENERGIZACAO"; break;
    case MODO_2: nomeModo = "RETARDO_DESENERGIZACAO"; break;
    case MODO_3: nomeModo = "CICLICO_INICIO_LIGADO"; break;
    case MODO_4: nomeModo = "CICLICO_INICIO_DESLIGADO"; break;
    case MODO_5: nomeModo = "ESTRELA_TRIANGULO"; break;
    case MODO_6: nomeModo = "CONTROLE_MANUAL"; break;
    default: nomeModo = "MODO_" + String(estadoAtual); break;
  }

  // Criar string de status
  String status = "STATUS|" + nomeModo + "|" + String(config.tempo1) + "|" + String(config.tempo1) + "|" + es

  debugPrint("Enviando status automático: " + status);
  debugPrint("Detalhes - Estado atual: " + String(estadoAtual) + ", Config modo: " + String(config.modo) + ",

  enviarNotificacao(status);
}

void verificarStatusEntrada() {
  bool entradaAtiva = validarEntrada();

  debugPrint("Verificando status VALIDADO da entrada: " + String(entradaAtiva ? "ATIVA" : "INATIVA"));

  if (deviceConnected) {
    if (entradaValidada) {
      enviarNotificacao("INFO: Entrada ATIVA - Modo não pode ser alterado");
    } else {
      enviarNotificacao("INFO: Entrada INATIVA - Modo pode ser alterado");
    }
  }

  debugPrint("Status da entrada: " + String(entradaAtiva ? "ATIVA" : "INATIVA") + " | Validada: " + String(en
}

// Função para validar entrada com anti-ruído usando ZMPT101B com cálculo RMS
bool validarEntrada() {
  // Calcular tensão RMS usando método correto para tensão AC
  long sum_mV = 0;
```

```
// Primeira passada: calcular média
for (int i = 0; i < ZMPT101B_SAMPLES; i++) {
  int raw = adc1_get_raw(ZMPT101B_CHANNEL);
  uint32_t mv = esp_adc_cal_raw_to_voltage(raw, &adc_chars);
  sum_mV += (long)mv;
}

double mean_mV = (double)sum_mV / (double)ZMPT101B_SAMPLES;
double meanV = mean_mV / 1000.0;

// Segunda passada: calcular RMS
double sq = 0.0;
int lastRaw = 0;
for (int i = 0; i < ZMPT101B_SAMPLES; i++) {
  int raw = adc1_get_raw(ZMPT101B_CHANNEL);
  lastRaw = raw;
  uint32_t mv = esp_adc_cal_raw_to_voltage(raw, &adc_chars);
  double v = (double)mv / 1000.0;
  double d = v - meanV;
  sq += d * d;
}

double Vrms_raw = sqrt(sq / (double)ZMPT101B_SAMPLES);
double calib = ZMPT101B_NETWORK_VOLTAGE / ZMPT101B_MODULE_MEASURED_VRMS;
double Vrms_network = Vrms_raw * calib;

// Determinar se há tensão AC baseado no threshold RMS
bool leituraAtual = Vrms_network > ZMPT101B_THRESHOLD_VRMS;

// Log da leitura RMS (a cada 10 verificações para não poluir o log)
static int contadorLog = 0;
contadorLog++;
if (contadorLog >= 10) {
  debugPrint("ZMPT101B - Vrms_rede: " + String(Vrms_network, 2) + "V | Vrms_modulo: " + String(Vrms_raw, 4)
  contadorLog = 0;
}

if (leituraAtual) {
  // Entrada lida como ativa
  contadorEntradaAtiva++;
  contadorEntradaInativa = 0; // Reset contador inativo

  if (contadorEntradaAtiva >= VALIDACAO_ENTRADA_COUNT && !entradaValidada) {
    // Entrada validada como ativa
    entradaValidada = true;
    debugPrint("Entrada VALIDADA como ATIVA após " + String(VALIDACAO_ENTRADA_COUNT) + " leituras consecuti
    return true;
  }
} else {
  // Entrada lida como inativa
  contadorEntradaInativa++;
  contadorEntradaAtiva = 0; // Reset contador ativo

  if (contadorEntradaInativa >= VALIDACAO_ENTRADA_COUNT && entradaValidada) {
```

```
      // Entrada validada como inativa
      entradaValidada = false;
      debugPrint("Entrada VALIDADA como INATIVA após " + String(VALIDACAO_ENTRADA_COUNT) + " leituras consecu
      return false;
    }
  }

  // Retorna o estado validado anterior (não mudou ainda)
  return entradaValidada;
}

// Função para calibrar o sensor ZMPT101B
void calibrarZMPT101B() {
  debugPrint("Iniciando calibração do ZMPT101B...");
  debugPrint("  Certifique-se de que NÃO há tensão AC na entrada!");

  // Usar o mesmo método da função validarEntrada para consistência
  long sum_mV = 0;
  const int amostrasCalibracao = 2000;

  for (int i = 0; i < amostrasCalibracao; i++) {
    int raw = adc1_get_raw(ZMPT101B_CHANNEL);
    uint32_t mv = esp_adc_cal_raw_to_voltage(raw, &adc_chars);
    sum_mV += (long)mv;

    if (i % 500 == 0) {
      debugPrint("Calibração: " + String(i) + "/" + String(amostrasCalibracao));
    }
  }

  double mean_mV = (double)sum_mV / (double)amostrasCalibracao;
  double meanV = mean_mV / 1000.0;

  // Calcular RMS para calibração
  double sq = 0.0;
  for (int i = 0; i < amostrasCalibracao; i++) {
    int raw = adc1_get_raw(ZMPT101B_CHANNEL);
    uint32_t mv = esp_adc_cal_raw_to_voltage(raw, &adc_chars);
    double v = (double)mv / 1000.0;
    double d = v - meanV;
    sq += d * d;
  }

  double Vrms_raw = sqrt(sq / (double)amostrasCalibracao);
  double Vrms_network = Vrms_raw * (ZMPT101B_NETWORK_VOLTAGE / ZMPT101B_MODULE_MEASURED_VRMS);

  debugPrint("Calibração concluída:");
  debugPrint("   Média sem tensão: " + String(meanV, 4) + "V");
  debugPrint("   Vrms módulo: " + String(Vrms_raw, 4) + "V");
  debugPrint("   Vrms rede estimada: " + String(Vrms_network, 2) + "V");
  debugPrint("   Threshold atual: " + String(ZMPT101B_THRESHOLD_VRMS) + "V");

  // Sugerir novo threshold baseado na calibração
  float novoThreshold = Vrms_network + 5.0; // Threshold = tensão calibrada + margem de segurança
```

```
  debugPrint("   Threshold sugerido: " + String(novoThreshold, 1) + "V");

  // Enviar resultado da calibração via Bluetooth se conectado
  if (deviceConnected) {
    String resultadoCalibracao = "CALIBRACAO|" + String(Vrms_network, 2) + "|" + String(novoThreshold, 1);
    enviarNotificacao(resultadoCalibracao);
  }
}

// Função para ajustar threshold do ZMPT101B via comando
bool ajustarThresholdZMPT101B(String comando) {
  // Formato: "THRESHOLD|valor" (valor em Volts RMS)
  if (comando.startsWith("THRESHOLD|")) {
    float novoThreshold = comando.substring(10).toFloat();

    if (novoThreshold > 0.0 && novoThreshold < 300.0) { // Threshold entre 0V e 300V RMS
      // Aqui você pode implementar a lógica para salvar o novo threshold
      // Por enquanto, apenas loga a alteração
      debugPrint("Threshold ZMPT101B alterado para: " + String(novoThreshold, 1) + "V RMS");

      if (deviceConnected) {
        enviarResposta("THRESHOLD_ALTERADO|" + String(novoThreshold, 1) + "V");
      }
      return true;
    } else {
      debugPrint("Threshold inválido: " + String(novoThreshold, 1) + "V");
      if (deviceConnected) {
        enviarResposta("ERR: Threshold deve estar entre 0.1V e 300V RMS");
      }
      return false;
    }
  }
  return false;
}

void debugPrint(String mensagem) {
  if (DEBUG_ENABLED) {
    Serial.println(mensagem);
  }
}

// ========================================
// FUNÇÕES DE CONTROLE DOS LEDs
// ========================================

// Função para configurar os pinos dos LEDs
void configurarLEDs() {
  pinMode(ledAzul, OUTPUT);
  pinMode(ledAmarelo, OUTPUT);
  pinMode(ledVermelho, OUTPUT);
  pinMode(ledBranco, OUTPUT);

  // Inicializar todos os LEDs desligados
```

```
  digitalWrite(ledAzul, LOW);
  digitalWrite(ledAmarelo, LOW);
  digitalWrite(ledVermelho, LOW);
  digitalWrite(ledBranco, LOW);

  // Inicializar estado do relé1 (HIGH = desligado)
  rele1Atual = true;

  debugPrint("LEDs configurados: Azul(GPIO23), Amarelo(GPIO22), Vermelho(GPIO21), Branco(GPIO18)");
}

// Função principal para controlar os LEDs baseado no modo atual
void controlarLEDs() {
  // Controlar LED Vermelho (indicador de tensão AC)
  bool tensaoAC = validarEntrada();
  digitalWrite(ledVermelho, tensaoAC ? HIGH : LOW);

  // Controlar LEDs Azul e Amarelo baseado no modo atual
  switch (estadoAtual) {
    case MODO_1: // Retardo na energização
      digitalWrite(ledAzul, HIGH);      // LED Azul sempre aceso
      digitalWrite(ledAmarelo, LOW);    // LED Amarelo sempre desligado
      break;

    case MODO_2: // Retardo na desenergização
      // LED Azul piscando
      if (millis() - ultimoTempoPiscaLed >= TEMPO_PISCA_LED) {
        estadoPiscaLed = !estadoPiscaLed;
        digitalWrite(ledAzul, estadoPiscaLed ? HIGH : LOW);
        ultimoTempoPiscaLed = millis();
      }
      digitalWrite(ledAmarelo, LOW);    // LED Amarelo sempre desligado
      break;

    case MODO_3: // Cíclico com início ligado
      digitalWrite(ledAzul, LOW);       // LED Azul sempre desligado
      digitalWrite(ledAmarelo, HIGH);   // LED Amarelo sempre aceso
      break;

    case MODO_4: // Cíclico com início desligado
      digitalWrite(ledAzul, LOW);       // LED Azul sempre desligado
      // LED Amarelo piscando
      if (millis() - ultimoTempoPiscaLed >= TEMPO_PISCA_LED) {
        estadoPiscaLed = !estadoPiscaLed;
        digitalWrite(ledAmarelo, estadoPiscaLed ? HIGH : LOW);
        ultimoTempoPiscaLed = millis();
      }
      break;

    case MODO_5: // Partida estrela-triângulo
      digitalWrite(ledAzul, HIGH);      // LED Azul sempre aceso
      digitalWrite(ledAmarelo, HIGH);   // LED Amarelo sempre aceso
      break;
```

```
    default:
      // Modo inválido - todos os LEDs desligados
      digitalWrite(ledAzul, LOW);
      digitalWrite(ledAmarelo, LOW);
      digitalWrite(ledVermelho, LOW);
      digitalWrite(ledBranco, LOW);
      break;
  }

  // Controlar LED Branco baseado no modo atual
  controlarLEDBranco();
}

// Função para controlar especificamente o LED branco
void controlarLEDBranco() {
  // Obter estado atual da entrada
  bool entradaAtiva = validarEntrada();

  switch (estadoAtual) {
    case MODO_1: // Retardo na energização
      // LED Branco pisca apenas durante contagem de tempo para ligar os relés (200ms)
      if (temporizadorModo1Iniciado && !relesLigados && (millis() - tempoInicio) < (config.tempo1 * 1000)) {
        if (millis() - ultimoTempoPiscaLedBranco >= TEMPO_PISCA_LED_BRANCO_RAPIDO) {
          estadoPiscaLedBranco = !estadoPiscaLedBranco;
          digitalWrite(ledBranco, estadoPiscaLedBranco ? HIGH : LOW);
          ultimoTempoPiscaLedBranco = millis();
        }
      } else {
        digitalWrite(ledBranco, LOW); // Desligado quando não está contando tempo
      }
      break;

    case MODO_2: // Retardo na desenergização
      // LED Branco pisca apenas durante contagem de tempo para desligar os relés (200ms)
      if (relesLigados && (millis() - tempoInicio) < (config.tempo1 * 1000)) {
        if (millis() - ultimoTempoPiscaLedBranco >= TEMPO_PISCA_LED_BRANCO_RAPIDO) {
          estadoPiscaLedBranco = !estadoPiscaLedBranco;
          digitalWrite(ledBranco, estadoPiscaLedBranco ? HIGH : LOW);
          ultimoTempoPiscaLedBranco = millis();
        }
      } else {
        digitalWrite(ledBranco, LOW); // Desligado quando não está contando tempo
      }
      break;

    case MODO_3: // Cíclico com início ligado
      // LED Branco pisca de 0.5s em 0.5s apenas quando entrada estiver ativa (true)
      if (entradaAtiva) {
        if (millis() - ultimoTempoPiscaLedBranco >= TEMPO_PISCA_LED_BRANCO_LENTO) {
          estadoPiscaLedBranco = !estadoPiscaLedBranco;
          digitalWrite(ledBranco, estadoPiscaLedBranco ? HIGH : LOW);
          ultimoTempoPiscaLedBranco = millis();
        }
      } else {
```

```
      digitalWrite(ledBranco, LOW); // Desligado quando entrada não estiver ativa
    }
    break;

  case MODO_4: // Cíclico com início desligado
    // LED Branco pisca de 0.5s em 0.5s apenas quando entrada estiver ativa (true)
    if (entradaAtiva) {
      if (millis() - ultimoTempoPiscaLedBranco >= TEMPO_PISCA_LED_BRANCO_LENTO) {
        estadoPiscaLedBranco = !estadoPiscaLedBranco;
        digitalWrite(ledBranco, estadoPiscaLedBranco ? HIGH : LOW);
        ultimoTempoPiscaLedBranco = millis();
      }
    } else {
      digitalWrite(ledBranco, LOW); // Desligado quando entrada não estiver ativa
    }
    break;

  case MODO_5: // Partida estrela-triângulo
    // LED Branco pisca apenas quando relé1 estiver LOW e entrada estiver ativa (200ms)
    if (rele1Atual == false && entradaAtiva) { // rele1Atual == false significa relé1 LOW (ligado)
      if (millis() - ultimoTempoPiscaLedBranco >= TEMPO_PISCA_LED_BRANCO_RAPIDO) {
        estadoPiscaLedBranco = !estadoPiscaLedBranco;
        digitalWrite(ledBranco, estadoPiscaLedBranco ? HIGH : LOW);
        ultimoTempoPiscaLedBranco = millis();
      }
    } else {
      digitalWrite(ledBranco, LOW); // Desligado quando não atende às condições
    }
    break;

  default:
    digitalWrite(ledBranco, LOW); // Desligado para modo inválido
    break;
 }
}
```

Apêndice B

```
import 'dart:async';
import 'dart:convert';
import 'dart:typed_data'; // Added for Uint8List

import 'package:flutter/material.dart';
import 'package:flutter_bluetooth_serial/flutter_bluetooth_serial.dart';
import 'package:permission_handler/permission_handler.dart';
import 'package:tcc_eng_eletrica/models/bluetooh_model.dart';
import 'package:tcc_eng_eletrica/models/rele_model.dart';
import 'package:tcc_eng_eletrica/services/storage.dart';
import 'package:tcc_eng_eletrica/utils/utils.dart';
import 'package:tcc_eng_eletrica/config/commands.dart';
import 'package:vibration/vibration.dart';

class BluetoothProvider extends ChangeNotifier {
```

```
BluetoothConnection? connection;
String idConnected = '';
bool isConnected = false;
bool loading = true;
bool scanningNewDevices = false;
List<BluetoothModel> devices = [];
List<BluetoothModel> newDevices = [];
StreamSubscription? streamAce;
StreamSubscription? _discoveryStream;
Timer? _scanTimer;
String? lastDevice;
final LocalStorage _storage = LocalStorage();
bool connecting = false;
String aux = '';
String lastAux = '';
String lastMessage = '';

// Campos para sincronização automática
ReleModel? statusAtual;
Timer? _statusTimer;
bool aguardandoStatus = false;

BluetoothProvider() {
  _init();
}

Future<void> searchDevices() async {
  await _startDiscovery();
}

Future<void> startScanningNewDevices({int scanDurationSeconds = 30}) async {
  await _startNewDevicesDiscovery(scanDurationSeconds);
}

Future<void> stopScanningNewDevices() async {
  await _stopNewDevicesDiscovery();
}

@override
void dispose() {
  disconnect();
  _discoveryStream?.cancel();
  _scanTimer?.cancel();
  _statusTimer?.cancel();
  super.dispose();
}

Future<bool> checkBluetoothStatus() async {
  try {
    // Verificar permissões Bluetooth básicas
    final PermissionStatus bluetoothStatus =
        await Permission.bluetooth.status;
    if (bluetoothStatus.isDenied) {
      final PermissionStatus requestedStatus =
```

```
        await Permission.bluetooth.request();
    if (!requestedStatus.isGranted) {
      debugPrint('Permissão Bluetooth negada');
      return false;
    }
  }

  // Verificar permissão de localização (necessária para Android 6+)
  final PermissionStatus locationStatus = await Permission.location.status;
  if (locationStatus.isDenied) {
    final PermissionStatus requestedLocationStatus =
        await Permission.location.request();
    if (!requestedLocationStatus.isGranted) {
      debugPrint('Permissão de localização negada');
      return false;
    }
  }

  // Verificar permissões específicas do Android 12+ (API 31+)
  try {
    // Verificar BLUETOOTH_CONNECT
    final PermissionStatus connectStatus =
        await Permission.bluetoothConnect.status;
    if (connectStatus.isDenied) {
      final PermissionStatus requestedConnectStatus =
          await Permission.bluetoothConnect.request();
      if (!requestedConnectStatus.isGranted) {
        debugPrint('Permissão BLUETOOTH_CONNECT negada');
        return false;
      }
    }

    // Verificar BLUETOOTH_SCAN
    final PermissionStatus scanStatus =
        await Permission.bluetoothScan.status;
    if (scanStatus.isDenied) {
      final PermissionStatus requestedScanStatus =
          await Permission.bluetoothScan.request();
      if (!requestedScanStatus.isGranted) {
        debugPrint('Permissão BLUETOOTH_SCAN negada');
        return false;
      }
    }
  } catch (e) {
    debugPrint('Erro ao verificar permissões Bluetooth específicas: $e');
    // Se não conseguir verificar essas permissões, pode ser Android mais antigo
    // Continuar com a verificação básica
  }

  // Verificar se o Bluetooth está ligado usando FlutterBluetoothSerial
  try {
    final bool? isEnabled = await FlutterBluetoothSerial.instance.isEnabled;
    if (isEnabled == null || !isEnabled) {
      debugPrint('Bluetooth não está ligado');
```

```
        // Em vez de usar requestEnable que pode causar erro, vamos retornar false
        // e deixar o usuário ligar manualmente
        return false;
      }

      return true;
    } catch (e) {
      debugPrint('Erro ao verificar status do Bluetooth: $e');
      return false;
    }
  } catch (e) {
    debugPrint('Erro ao verificar permissões: $e');
    return false;
  }
}

/// Verifica o status do Bluetooth e retorna uma mensagem descritiva
Future<Map<String, dynamic>> getBluetoothStatusInfo() async {
  try {
    // Verificar permissões básicas
    final PermissionStatus bluetoothStatus =
        await Permission.bluetooth.status;
    final PermissionStatus locationStatus = await Permission.location.status;

    if (bluetoothStatus.isDenied) {
      return {
        'ready': false,
        'message': 'Permissão Bluetooth básica necessária',
        'action': 'Solicitar permissão Bluetooth'
      };
    }

    if (locationStatus.isDenied) {
      return {
        'ready': false,
        'message': 'Permissão de localização necessária',
        'action': 'Solicitar permissão de localização'
      };
    }

    // Verificar permissões específicas do Android 12+
    try {
      final PermissionStatus connectStatus =
          await Permission.bluetoothConnect.status;
      if (connectStatus.isDenied) {
        return {
          'ready': false,
          'message': 'Permissão BLUETOOTH_CONNECT necessária',
          'action': 'Solicitar permissão BLUETOOTH_CONNECT',
          'subMessage':
              'Esta permissão é necessária para conectar a dispositivos Bluetooth'
        };
      }
```

```
      final PermissionStatus scanStatus =
          await Permission.bluetoothScan.status;
      if (scanStatus.isDenied) {
        return {
          'ready': false,
          'message': 'Permissão BLUETOOTH_SCAN necessária',
          'action': 'Solicitar permissão BLUETOOTH_SCAN',
          'subMessage':
              'Esta permissão é necessária para descobrir dispositivos Bluetooth'
        };
      }
    } catch (e) {
      debugPrint('Erro ao verificar permissões Bluetooth específicas: $e');
      // Se não conseguir verificar essas permissões, pode ser Android mais antigo
      // Continuar com a verificação básica
    }

    // Verificar se Bluetooth está ligado
    final bool? isEnabled = await FlutterBluetoothSerial.instance.isEnabled;
    if (isEnabled == null || !isEnabled) {
      return {
        'ready': false,
        'message': 'Bluetooth está desligado',
        'action': 'Abrir Configurações',
        'subMessage':
            'Toque para abrir as configurações do sistema e ligar o Bluetooth manualmente'
      };
    }

    return {
      'ready': true,
      'message': 'Bluetooth está funcionando',
      'action': 'Pronto para usar'
    };
  } catch (e) {
    return {
      'ready': false,
      'message': 'Erro ao verificar Bluetooth',
      'action': 'Verificar configurações'
    };
  }
}

Future<void> _startDiscovery() async {
  try {
    if (devices.isNotEmpty) {
      devices.clear();
    }

    // Verificar status do Bluetooth antes de iniciar a descoberta
    final bool bluetoothReady = await checkBluetoothStatus();
    if (!bluetoothReady) {
      debugPrint('Bluetooth não está pronto para uso');
      loading = false;
```

```
      notifyListeners();
      return;
    }

    debugPrint('Iniciando descoberta de dispositivos Bluetooth...');
    await FlutterBluetoothSerial.instance
        .getBondedDevices()
        .then((bondedDevices) {
      for (final element in bondedDevices) {
        debugPrint(
            'Dispositivo encontrado: ${element.name} (${element.address})');
        devices.add(
          BluetoothModel(
            name: element.name ?? 'Dispositivo Desconhecido',
            id: element.address,
          ),
        );
      }
      debugPrint('Total de dispositivos encontrados: ${devices.length}');

      loading = false;
      notifyListeners();
    });
  } catch (error) {
    debugPrint('Erro durante a descoberta de dispositivos: $error');
    loading = false;
    notifyListeners();
  }
}

Future<void> _startNewDevicesDiscovery([int scanDurationSeconds = 30]) async {
  if (scanningNewDevices) {
    debugPrint('Descoberta de novos dispositivos já em andamento.');
    return;
  }

  scanningNewDevices = true;
  newDevices.clear();
  notifyListeners();

  debugPrint('Iniciando descoberta de novos dispositivos Bluetooth...');
  _discoveryStream =
      FlutterBluetoothSerial.instance.startDiscovery().listen((result) {
    if (result.device.name != null && result.device.name!.isNotEmpty) {
      debugPrint(
          'Dispositivo novo encontrado: ${result.device.name} (${result.device.address})');
      final newDevice = BluetoothModel(
        name: result.device.name!,
        id: result.device.address,
      );

      // Verificar se o dispositivo já existe nas listas
      bool existsInDevices =
          devices.any((device) => device.id == newDevice.id);
```

```
      bool existsInNewDevices =
          newDevices.any((device) => device.id == newDevice.id);

      if (!existsInDevices && !existsInNewDevices) {
        newDevices.add(newDevice);
        debugPrint(
            'Dispositivo adicionado à lista de novos dispositivos: ${newDevice.name}');
        notifyListeners();
      }
    }
  });

  // Timer para parar o escaneamento automaticamente após o tempo especificado
  _scanTimer = Timer(Duration(seconds: scanDurationSeconds), () {
    debugPrint(
        'Timer de escaneamento expirado após $scanDurationSeconds segundos. Parando descoberta...');
    _stopNewDevicesDiscovery();
  });
}

Future<void> _stopNewDevicesDiscovery() async {
  await _discoveryStream?.cancel();
  _discoveryStream = null;
  _scanTimer?.cancel();
  _scanTimer = null;
  scanningNewDevices = false;
  notifyListeners();
}

void reloadDevices() {
  loading = true;
  notifyListeners();
  devices = [];
  _startDiscovery();
}

void clearNewDevices() {
  newDevices.clear();
  notifyListeners();
}

// Verifica se o escaneamento está ativo e retorna o tempo restante
bool get isScanning => scanningNewDevices;

// Retorna o tempo restante do escaneamento em segundos (para UI)
int get remainingScanTime {
  if (_scanTimer == null || !scanningNewDevices) return 0;
  // Como não temos acesso direto ao tempo restante do Timer,
  // vamos usar uma aproximação baseada no estado
  return 30; // Valor padrão para simplificar
}

Future<void> disconnect() async {
  await connection?.close();
```

```
  await connection?.finish();
  streamAce?.cancel();
  _limparTimerStatus();
  statusAtual = null;
  idConnected = '';
  isConnected = false;
  notifyListeners();
  if (await Vibration.hasVibrator() == true) {
    Vibration.vibrate(pattern: [200, 200, 200, 200]);
  }
}

Future<void> _setAsConnected(String id) async {
  isConnected = true;
  idConnected = id;
  notifyListeners();
  if (await Vibration.hasVibrator() == true) {
    Vibration.vibrate(duration: 300);
  }
}

Future<bool> connectDevice(String address) async {
  try {
    saveLastConnected(address);
    connecting = true;
    notifyListeners();

    // Tentar conexão com retry
    bool connected = false;
    int retryCount = 0;
    const int maxRetries = 3;

    while (!connected && retryCount < maxRetries) {
      try {
        debugPrint('Tentativa de conexão ${retryCount + 1} para $address');

        // Criar conexão com timeout
        connection = await BluetoothConnection.toAddress(address);

        // Verificar se a conexão foi estabelecida
        if (connection != null) {
          // Aguardar um pouco para a conexão se estabilizar
          await Future.delayed(const Duration(milliseconds: 500));
          connected = true;
          debugPrint(
              'Conexão estabelecida com sucesso na tentativa ${retryCount + 1}');
          break;
        }
      } catch (e) {
        retryCount++;
        debugPrint('Tentativa $retryCount falhou: $e');

        // Aguardar antes da próxima tentativa
        if (retryCount < maxRetries) {
```

```
        await Future.delayed(Duration(seconds: retryCount * 2));
      }

      // Limpar conexão anterior se existir
      if (connection != null) {
        try {
          await connection!.close();
          await connection!.finish();
        } catch (cleanupError) {
          debugPrint('Erro ao limpar conexão anterior: $cleanupError');
        }
        connection = null;
      }
    }
  }

  if (!connected) {
    throw Exception('Falha na conexão após $maxRetries tentativas');
  }

  // Configurar listeners para a conexão estabelecida
  connection!.input!.listen((data) {
    debugPrint('Data incoming: ${ascii.decode(data)}');
    aux += ascii.decode(data);

    // Verificar se recebeu string de status
    if (ascii.decode(data).contains('STATUS|')) {
      _processarStatusRecebido(ascii.decode(data));
    }

    // Verificar se recebeu notificação de estado (ON/OFF)
    if (ascii.decode(data).contains('ON') ||
        ascii.decode(data).contains('OFF')) {
      _processarNotificacaoEstado(ascii.decode(data));
    }

    if (ascii.decode(data).contains('!')) {
      debugPrint('Disconnecting by local host');
    }
  }).onDone(() {
    debugPrint('Disconnected by remote request');
    // Executar disconnect de forma assíncrona sem aguardar
    disconnect();
  });

  await _setAsConnected(address);
  connecting = false;
  notifyListeners();

  // Iniciar timer para aguardar status automático do ESP32
  _iniciarTimerStatus();

  return true;
} catch (exception) {
```

```
    debugPrint('ERRO na conexão: $exception');

    // Limpar estado de conexão
    if (connection != null) {
      try {
        await connection!.close();
        await connection!.finish();
      } catch (cleanupError) {
        debugPrint('Erro ao limpar conexão: $cleanupError');
      }
      connection = null;
    }

    disconnect();
    connecting = false;
    notifyListeners();

    return false;
  }
}

Future<void> sendString(String comando) async {
  // Validação do comando antes do envio
  // if (!Commands.isValidCommand(comando)) {
  //   debugPrint('ERRO: Comando inválido: "$comando"');
  //   throw Exception(
  //       'Comando inválido: "$comando". Comandos válidos: ${Commands.validCommands.join(', ')}');
  // }

  // // Validação do formato do comando
  // if (!Commands.isValidCommandFormat(comando)) {
  //   debugPrint('ERRO: Formato inválido para o comando: "$comando"');
  //   throw Exception('Formato inválido para o comando: "$comando"');
  // }

  comando = '${comando}_END';
  try {
    // Verificar se a conexão existe e está ativa
    if (connection == null) {
      debugPrint('ERRO: Tentativa de envio sem conexão ativa');
      throw Exception('Conexão Bluetooth não está ativa');
    }

    if (!isConnected) {
      debugPrint('ERRO: Tentativa de envio sem estar conectado');
      throw Exception('Dispositivo não está conectado');
    }

    // Verificar se a conexão ainda está válida
    if (connection!.isConnected == false) {
      debugPrint('ERRO: Conexão Bluetooth perdida');
      throw Exception('Conexão Bluetooth perdida');
    }
```

```
    debugPrint('Enviando comando: "$comando"');
    debugPrint('Tamanho do comando: ${comando.length} caracteres');

    // Converter string para bytes e enviar
    final Uint8List bytes = Uint8List.fromList(ascii.encode(comando));
    debugPrint('Bytes a serem enviados: $bytes');

    // Adicionar dados ao buffer de saída
    connection!.output.add(bytes);

    // Aguardar todos os dados serem enviados
    await connection!.output.allSent;

    debugPrint('Comando enviado com sucesso: "$comando"');

    // Aguardar um pequeno delay para garantir o envio
    await Future.delayed(const Duration(milliseconds: 100));
  } catch (e) {
    debugPrint('ERRO ao enviar comando "$comando": $e');

    // Se houve erro de envio, tentar reconectar
    if (e.toString().contains('Conexão Bluetooth perdida') ||
        e.toString().contains('connection')) {
      debugPrint('Tentando reconectar devido a erro de conexão...');
      await disconnect();
      // Notificar o usuário sobre o problema
      notifyListeners();
    }

    // Re-lançar o erro para ser tratado pela camada superior
    rethrow;
  }
}

bool hasLastConnected() {
  return lastDevice != null;
}

Future<void> saveLastConnected(String device) async {
  lastDevice = device;
  _storage.saveLastConnected(device);
}

void _init() {
  lastDevice = _storage.getLastConnected();
  notifyListeners();
}

void connectLastDevice() {
  connectDevice(lastDevice!);
}

void startToReceiveData(int milliseconds) {
  if (aux.isEmpty) {
```

```
    Future.delayed(Duration(milliseconds: milliseconds), () {
      debugPrint('RECEIVED $aux');
      lastAux = aux;
      showMessageSuccess(lastAux);
      notifyListeners();
      aux = '';
    });
  }
}

String get lastData => lastAux;

// Método para obter informações sobre comandos válidos (útil para debug e manutenção)
Map<String, String> getValidCommandsInfo() {
  Map<String, String> commandsInfo = {};
  for (String command in Commands.validCommands) {
    commandsInfo[command] = Commands.getCommandDescription(command);
  }
  return commandsInfo;
}

// Método para validar um comando sem enviá-lo
bool validateCommand(String comando) {
  return Commands.isValidCommand(comando) &&
      Commands.isValidCommandFormat(comando);
}

// Função para processar string de status recebida
void _processarStatusRecebido(String data) {
  try {
    debugPrint('Processando status recebido: $data');

    // Limpar timer se ainda estiver ativo
    _statusTimer?.cancel();
    aguardandoStatus = false;

    // Procurar por padrão STATUS| na string completa
    String statusString = '';
    if (data.contains('STATUS|')) {
      int startIndex = data.indexOf('STATUS|');
      int endIndex = data.indexOf('\n', startIndex);
      if (endIndex == -1) endIndex = data.length;
      statusString = data.substring(startIndex, endIndex).trim();
    }

    if (statusString.isNotEmpty) {
      // Criar modelo de relé a partir da string de status
      statusAtual = ReleModel.fromStatusString(statusString);

      debugPrint(
          'Status processado: Modo ${statusAtual?.modo}, T1: ${statusAtual?.tempo1}, T2: ${statusAtual?.tem

      notifyListeners();
    }
```

```
  } catch (e) {
    debugPrint('Erro ao processar status: $e');
  }
}

// Função para processar notificações de estado (ON/OFF)
void _processarNotificacaoEstado(String data) {
  try {
    debugPrint('Processando notificação de estado: $data');

    if (statusAtual != null) {
      bool novoEstado = data.contains('ON');
      statusAtual = statusAtual!.copyWith(estadoAtual: novoEstado);
      debugPrint('Estado atualizado: ${novoEstado ? 'Ligado' : 'Desligado'}');
      notifyListeners();
    }
  } catch (e) {
    debugPrint('Erro ao processar notificação de estado: $e');
  }
}

// Função para iniciar timer de aguardar status
void _iniciarTimerStatus() {
  aguardandoStatus = true;
  notifyListeners();

  _statusTimer = Timer(const Duration(seconds: 3), () {
    debugPrint('Timer de status expirado - aguardando dados do ESP32');
    // O ESP32 deve enviar o status automaticamente
  });
}

// Função para limpar timer de status
void _limparTimerStatus() {
  _statusTimer?.cancel();
  aguardandoStatus = false;
  notifyListeners();
}

// Getter para verificar se está aguardando status
bool get isAguardandoStatus => aguardandoStatus;

// Getter para obter status atual
ReleModel? get getStatusAtual => statusAtual;

// Método para atualizar o status do relé (chamado pelo HomeController)
void atualizarStatusRele(ReleModel novoStatus) {
  statusAtual = novoStatus;
  debugPrint(
      'Status do relé atualizado pelo HomeController: ${novoStatus.modoDescricao}');
  notifyListeners();
}

// Função para testar a conexão enviando um comando de teste
```

```
  Future<bool> testConnection() async {
    try {
      if (!isConnected || connection == null) {
        debugPrint('Teste de conexão: Não conectado');
        return false;
      }

      debugPrint('Testando conexão...');

      // Enviar comando de teste
      await sendString('TESTE');

      // Aguardar um pouco para ver se há resposta
      await Future.delayed(const Duration(milliseconds: 500));

      debugPrint('Teste de conexão: Sucesso');
      return true;
    } catch (e) {
      debugPrint('Teste de conexão falhou: $e');
      return false;
    }
  }

  // Função para obter informações detalhadas da conexão
  Map<String, dynamic> getConnectionInfo() {
    return {
      'isConnected': isConnected,
      'hasConnection': connection != null,
      'connectionValid': connection?.isConnected ?? false,
      'idConnected': idConnected,
      'lastData': lastAux,
      'statusAtual': statusAtual != null
          ? {
              'modo': statusAtual!.modo,
              'tempo1': statusAtual!.tempo1,
              'tempo2': statusAtual!.tempo2,
              'estadoAtual': statusAtual!.estadoAtual,
            }
          : null,
    };
  }
}
```

Apêndice C

```
import 'package:flutter/material.dart';
import 'package:tcc_eng_eletrica/config/commands.dart';
import 'package:tcc_eng_eletrica/config/reles_config.dart';
import 'package:tcc_eng_eletrica/models/rele_model.dart';
import 'package:tcc_eng_eletrica/providers/bt_provider.dart';
import 'package:tcc_eng_eletrica/services/storage.dart';
import 'package:tcc_eng_eletrica/utils/utils.dart';
```

```
class HomeController extends ChangeNotifier {
  final List<ReleModel> reles = RelesConfig.reles;
  ReleModel releSelected = RelesConfig.reles[0];
  final BluetoothProvider bt;
  final LocalStorage _storage = LocalStorage();
  final List<FixedExtentScrollController> controllerTimer = [];
  int dias = 0;
  int horas = 0;
  int minutos = 0;
  int segundos = 0;
  bool showImage = true;

  final List<String> list = [];

  HomeController(this.bt) {
    initReles();
    _initStorageAndLoadConfig();
  }

  Future<void> _initStorageAndLoadConfig() async {
    await LocalStorage.init();
    _loadSavedConfig();
  }

  @override
  void dispose() {
    super.dispose();
    bt.dispose();
  }

  Future<void> sendOptToEsp32() async {
    if (!bt.isConnected) {
      showMessageError('Bluetooth não conectado');
      return;
    }

    int time = getTimeInSeconds();
    if (time == 0) {
      showMessageError('Tempo não pode ser 0');
      return;
    }

    try {
      String textToSend;
      int tempo2 = 0;
      if (releSelected.model == 6) {
        // Modo 6: apenas controle manual, sem tempo
        textToSend = '${releSelected.model}|0|0_END';
        debugPrint('Enviando configuração modo manual: $textToSend');
      } else {
        tempo2 = time;

        textToSend = '${releSelected.model}|$time|$tempo2';
        debugPrint('Enviando configuração com tempo: $textToSend');
```

```
    }

    await bt.sendString(textToSend);
    // espera 3 segundos e verifica se bt.lastMessage contem OK
    await Future.delayed(const Duration(seconds: 3));
    debugPrint('bt.lastMessage: -${bt.lastMessage}-');
    if (bt.lastMessage.contains('OK') || 1 == 1) {
      showMessageSuccess('Configuração enviada com sucesso!');
      saveConfig(); // Salva a configuração no storage
      showMessageSuccess('Configuração enviada com sucesso!');
      debugPrint('Configuração enviada com sucesso');

      // Atualizar o status do relé selecionado com base nos dados enviados
      _atualizarStatusReleSelecionado(time, tempo2);
    } else {
      showMessageError('Erro ao enviar comando: ${bt.lastMessage}');
    }

    // Aguardar um pouco para a configuração ser processada
    await Future.delayed(const Duration(milliseconds: 500));

    // Sistema inicia automaticamente após configuração
    // Não é necessário comando START separado
  } catch (e) {
    debugPrint('Erro ao enviar valores: $e');
    showMessageError('Erro ao enviar comando: $e');
  }
}

// Método para atualizar o status do relé selecionado
void _atualizarStatusReleSelecionado(int tempo1, int tempo2) {
  // Criar um novo modelo de relé com os dados atualizados
  final novoStatus = ReleModel(
    title: releSelected.title,
    model: releSelected.model,
    url: releSelected.url,
    modo: releSelected.model,
    tempo1: tempo1,
    tempo2: tempo2,
    estadoAtual: false, // Estado inicial sempre desligado
  );

  // Atualizar o status no provider Bluetooth
  bt.atualizarStatusRele(novoStatus);

  debugPrint(
      'Status do relé atualizado: Modo ${novoStatus.modo}, T1: ${novoStatus.tempo1}, T2: ${novoStatus.tempo
}

void setMode(String title, int index) {
  print('setMode: $title index: $index');
  releSelected = reles[index];
  saveConfig(); // Salva no storage
  notifyListeners();
```

```
  debugPrint('rele ${reles[index].url} index: $index');
}

void initReles() {
  releSelected = reles[0];
  for (final element in reles) {
    list.add(element.title);
  }
}

void setTime(int time, String type) {
  if (type == 'dia') {
    dias = time;
  } else if (type == 'hora') {
    horas = time;
  } else if (type == 'min') {
    minutos = time;
  } else if (type == 'seg') {
    segundos = time;
  }
  saveConfig(); // Salva no storage
  notifyListeners();
}

int getTimeInMilliseconds() {
  int time = 0;
  time += dias * 86400000;
  time += horas * 3600000;
  time += minutos * 60000;
  time += segundos * 1000;
  return time;
}

// Novo método para obter tempo em segundos (formato esperado pelo ESP32)
int getTimeInSeconds() {
  int time = 0;
  time += dias * 86400; // dias em segundos
  time += horas * 3600; // horas em segundos
  time += minutos * 60; // minutos em segundos
  time += segundos; // segundos
  return time;
}

Future<void> changeStatusRele() async {
  if (!bt.isConnected) {
    showMessageError('Bluetooth não conectado');
    return;
  }

  try {
    await bt.sendString(Commands.createChangeReleStatusCommand());
    showMessageSuccess('Comando enviado com sucesso!');
  } catch (e) {
    debugPrint('Erro ao enviar comando: $e');
```

```
      showMessageError('Erro ao enviar comando: $e');
    }
  }

  // void toggleImage() {
  //   showImage = !showImage;
  //   notifyListeners();
  // }

  Future<void> _loadSavedConfig() async {
    final savedConfig = _storage.getModeConfig();
    // mode = savedConfig['mode'] ?? 1;
    try {
      releSelected = reles.firstWhere(
        (element) => element.model == savedConfig['mode'],
        orElse: () =>
            reles[0], // Fallback para o primeiro relé se não encontrar
      );
    } catch (e) {
      // Se houver algum erro, usa o primeiro relé como padrão
      releSelected = reles[0];
    }
    dias = savedConfig['dias'] ?? 0;
    horas = savedConfig['horas'] ?? 0;
    minutos = savedConfig['minutos'] ?? 0;
    segundos = savedConfig['segundos'] ?? 0;
    final times = [dias, horas, minutos, segundos];
    for (var i = 0; i < reles.length; i++) {
      print('times[i]: ${times[i]}');
      controllerTimer.add(FixedExtentScrollController(initialItem: times[i]));
    }
    notifyListeners();
  }

  Future<void> saveConfig() async {
    _storage.saveModeConfig(releSelected.model, dias, horas, minutos, segundos);
  }
}
```

Apêndice D

```
import 'package:direct_select_flutter/direct_select_container.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:tcc_eng_eletrica/config/strings.dart';
import 'package:tcc_eng_eletrica/config/themes/colors.dart';
import 'package:tcc_eng_eletrica/pages/home/home_controller.dart';
import 'package:tcc_eng_eletrica/providers/bt_provider.dart';
import 'package:tcc_eng_eletrica/widgets/bt_connect.dart';
import 'package:tcc_eng_eletrica/widgets/card_selector.dart';
import 'package:tcc_eng_eletrica/widgets/status_display.dart';
import 'package:wheel_chooser/wheel_chooser.dart';
import 'package:google_fonts/google_fonts.dart';
```

```
class HomePage extends StatelessWidget {
  const HomePage({super.key});

  @override
  Widget build(BuildContext context) {
    final btProvider = Provider.of<BluetoothProvider>(context);
    return ChangeNotifierProvider<HomeController>(
      create: (context) => HomeController(btProvider),
      child: Consumer<HomeController>(
        builder: (context, controller, _) {
          return Scaffold(
            appBar: AppBar(
              title: const Text(Strings.appName),
              centerTitle: true,
              backgroundColor: MyColors.secondary,
            ),
            body: DirectSelectContainer(
              child: SingleChildScrollView(
                child: Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: Column(
                    children: [
                      //BOTOES CONECTAR BT
                      Row(
                        mainAxisAlignment: MainAxisAlignment.center,
                        children: [
                          Center(
                            child: SizedBox(
                              height: 30,
                              child: FilledButton(
                                style: ButtonStyle(
                                  backgroundColor: WidgetStateProperty.all(
                                      getButtonColor(controller)),
                                  padding: WidgetStateProperty.all(
                                    const EdgeInsets.symmetric(
                                      horizontal: 24,
                                      vertical: 2,
                                    ),
                                  ),
                                ),
                                onPressed: () {
                                  if (controller.bt.isConnected) {
                                    controller.bt.disconnect();
                                  } else {
                                    showModalToConnectBluetooth(
                                        controller, context);
                                  }
                                },
                                child: Text(
                                  getStringButton(controller),
                                  textAlign: TextAlign.center,
                                  style: GoogleFonts.inter(
                                    fontSize: 14,
                                    color: getTextButtonColor(controller),
```

```
                ),
              ),
            ),
          ),
        ),
        if (hasLastConnection(controller)) ...[
          const SizedBox(width: 20),
          SizedBox(
            height: 30,
            width: 30,
            child: AnimatedRotation(
              duration: const Duration(seconds: 5),
              turns: controller.bt.connecting ? 8 : 0,
              child: Material(
                shape: const CircleBorder(),
                color: controller.bt.connecting
                    ? Colors.black12
                    : MyColors.secondary,
                child: IconButton(
                  padding: const EdgeInsets.all(0),
                  onPressed: () =>
                      controller.bt.connectLastDevice(),
                  icon: const Icon(
                    Icons.refresh,
                    color: Colors.white,
                    size: 18,
                  ),
                ),
              ),
            ),
          )
        ]
      ],
    ),

    // WIDGET DE STATUS ATUAL
    const StatusDisplay(),

    const SizedBox(height: 8),

    //CARD DE SELEÇÃO DE MODO
    Container(
      width: double.infinity,
      decoration: BoxDecoration(
        color: Colors.white,
        borderRadius: BorderRadius.circular(16),
        boxShadow: [
          BoxShadow(
            color: Colors.grey,
            blurRadius: 8,
            offset: const Offset(0, 2),
          ),
        ],
      ),
```

```
child: Padding(
  padding: const EdgeInsets.all(12.0),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      CardSelector(
        data: controller.reles,
        selectedIndex: controller.reles
            .indexOf(controller.releSelected),
        onChange: (value, index) =>
            controller.setMode(value.title, index),
        modelFromEsp32:
            controller.bt.getStatusAtual?.model,
      ),
      AnimatedSwitcher(
        duration: const Duration(milliseconds: 300),
        child: Column(
          children: [
            if (controller.showImage &&
                controller.releSelected.model != 6) ...[
              Column(
                children: [
                  const SizedBox(height: 8),
                  Text(
                    controller.releSelected.title,
                    textAlign: TextAlign.center,
                    style: GoogleFonts.inter(
                      fontSize: 12,
                      fontWeight: FontWeight.w400,
                    ),
                  ),
                  // const SizedBox(height: 6),
                  Image.asset(
                    controller.releSelected.url,
                    width: MediaQuery.of(context)
                            .size
                            .width *
                        0.7,
                  ),
                ],
              ),
            ],

            // Só mostra os controles de tempo se NÃO for o modo 6
            if (controller.releSelected.model != 6) ...[
              // Título da seção de tempo
              Container(
                width: double.infinity,
                padding: const EdgeInsets.symmetric(
                    vertical: 4, horizontal: 16),
                decoration: BoxDecoration(
                  color: MyColors.primary
                      .withValues(alpha: 0.1),
                  borderRadius:
```

```
            BorderRadius.circular(8),
        ),
        child: Row(
          children: [
            Icon(
              Icons.schedule,
              color: MyColors.primary,
              size: 20,
            ),
            const SizedBox(width: 8),
            Text(
              'Configuração de Tempo',
              style: GoogleFonts.poppins(
                color: MyColors.primary,
                fontSize: 13,
                fontWeight: FontWeight.w600,
              ),
            ),
          ],
        ),
      ),
    ),
    const SizedBox(height: 2),

    // Mensagem informativa quando não conectado
    if (!controller.bt.isConnected && 1 != 1)
      Container(
        width: double.infinity,
        padding: const EdgeInsets.all(16),
        margin:
            const EdgeInsets.only(bottom: 20),
        decoration: BoxDecoration(
          color: Colors.orange.shade50,
          borderRadius:
              BorderRadius.circular(12),
          border: Border.all(
            color: Colors.orange.shade200,
            width: 1,
          ),
        ),
        child: Row(
          children: [
            Icon(
              Icons.info_outline,
              color: Colors.orange.shade700,
              size: 24,
            ),
            const SizedBox(width: 12),
            Expanded(
              child: Text(
                'Conecte-se ao dispositivo para configurar o tempo',
                style: GoogleFonts.inter(
                  color:
                      Colors.orange.shade700,
                  fontSize: 14,
```

```
                    fontWeight: FontWeight.w500,
                  ),
                ),
              ),
            ],
          ),
        ),

      // Controles de tempo
      if (controller.controllerTimer.length ==
          4) ...[
        Row(
          mainAxisAlignment:
              MainAxisAlignment.spaceEvenly,
          children: [
            scrollWidget(
              controller,
              'dia',
              19,
              controller.controllerTimer[0],
            ),
            scrollWidget(
              controller,
              'hora',
              23,
              controller.controllerTimer[1],
            ),
            scrollWidget(
              controller,
              'min',
              59,
              controller.controllerTimer[2],
            ),
            scrollWidget(
              controller,
              'seg',
              59,
              controller.controllerTimer[3],
            ),
          ],
        ),
      ],
      const SizedBox(height: 20),
    ],
    // if (controller.releSelected.model != 6) ...[
    //   Row(
    //     mainAxisAlignment:
    //         MainAxisAlignment.spaceEvenly,
    //     children: [
    //       FilledButton.icon(
    //         style: ButtonStyle(
    //           backgroundColor:
    //               WidgetStateProperty.all(
    //                   MyColors.primary),
```

```
//           padding: WidgetStateProperty.all(
//             const EdgeInsets.symmetric(
//               horizontal: 16,
//               vertical: 12,
//             ),
//           ),
//         ),
//         onPressed: () =>
//             controller.toggleImage(),
//         icon: Icon(
//           controller.showImage
//               ? Icons.visibility_off
//               : Icons.visibility,
//           color: Colors.white,
//         ),
//         label: Text(
//           controller.showImage
//               ? 'Ocultar Imagem'
//               : 'Mostrar Imagem',
//           style: const TextStyle(
//             color: Colors.white,
//             fontSize: 12,
//           ),
//         ),
//       ),
//       // Só mostra o botão Salvar se NÃO for o modo 6 E estiver conect
if (controller.bt.isConnected) ...[
  FilledButton(
    style: ButtonStyle(
      backgroundColor:
          WidgetStateProperty.all(
              MyColors.success),
      padding: WidgetStateProperty.all(
        const EdgeInsets.symmetric(
          horizontal: 32,
          vertical: 10,
        ),
      ),
    ),
    onPressed: () =>
        controller.sendOptToEsp32(),
    child: const Text(
      'Salvar',
      style: TextStyle(
        color: Colors.white,
        fontSize: 14,
      ),
    ),
  ),
],
//     ],
//   ),
// ],
],
```

```
                            ),
                          ),
                        ],
                      ),
                    ),
                  ),
                  const SizedBox(height: 24),

                  //BOTÃO MUDAR ESTADO
                  if (controller.bt.isConnected &&
                      controller.releSelected.model == 6)
                    FilledButton(
                      style: ButtonStyle(
                        backgroundColor:
                            WidgetStateProperty.all(MyColors.success),
                        padding: WidgetStateProperty.all(
                          const EdgeInsets.symmetric(
                            horizontal: 32,
                            vertical: 16,
                          ),
                        ),
                      ),
                      onPressed: () => controller.changeStatusRele(),
                      child: Text(
                        'MUDAR ESTADO',
                        style: GoogleFonts.inter(
                          color: Colors.white,
                          fontSize: 16,
                        ),
                      ),
                    ),
                ],
              ),
            ),
          ),
        ),
      );
    },
  ),
);
}

Widget scrollWidget(HomeController controller, String title, int maxValue,
    FixedExtentScrollController controllerTimer,
    {step = 1}) {
  final enableEdit = controller.bt.isConnected;

  return Padding(
    padding: const EdgeInsets.symmetric(vertical: 6),
    child: Column(
      children: [
        Text(
          title,
          style: GoogleFonts.poppins(
```

```
      fontSize: 14,
      fontWeight: FontWeight.w500,
    ),
  ),
  Container(
    decoration: BoxDecoration(
      borderRadius: BorderRadius.circular(10),
      border: Border.all(
        color: controller.bt.isConnected
            ? Colors.black12
            : Colors.grey.shade300,
      ),
    ),
    height: 100,
    width: 60,
    child: Stack(
      children: [
        Positioned(
          top: 30,
          left: 5,
          child: Container(
            width: 50,
            height: 1.3,
            color: controller.bt.isConnected
                ? Colors.black
                : Colors.grey.shade400,
          ),
        ),
        Positioned(
          bottom: 30,
          left: 5,
          child: Container(
            width: 50,
            height: 1.3,
            color: controller.bt.isConnected
                ? Colors.black
                : Colors.grey.shade400,
          ),
        ),
        WheelChooser.integer(
          onValueChanged:
              enableEdit ? (i) => controller.setTime(i, title) : null,
          maxValue: maxValue,
          minValue: 0,
          step: step,
          controller: controllerTimer,
          isInfinite: true,
        ),
        if (!enableEdit)
          Positioned(
            top: 0,
            left: 0,
            child: Container(
              width: 100,
```

```
                    height: 100,
                    decoration: BoxDecoration(
                      borderRadius: BorderRadius.circular(10),
                      color: Colors.black.withValues(alpha: 0.1),
                      border: Border.all(
                        color: Colors.grey.shade300,
                      ),
                    ),
                    child: const Center(),
                  ),
                ),
              ],
            ),
          ),
        ],
      ),
    );
}

Future<void> showModalToConnectBluetooth(
    HomeController controller, context) async {
  await showModalBottomSheet(
    context: context,
    isScrollControlled: true,
    constraints: BoxConstraints(
      maxHeight: MediaQuery.of(context).size.height * 0.8,
    ),
    backgroundColor: Colors.white,
    builder: (context) {
      return ConnectBluetooth(ct: controller);
    },
  );
}

bool hasLastConnection(controller) {
  return controller.bt.hasLastConnected() && !controller.bt.isConnected;
}

Color getButtonColor(HomeController controller) {
  if (controller.bt.isConnected) {
    return MyColors.success;
  } else {
    if (hasLastConnection(controller)) {
      return MyColors.secondary.withValues(alpha: 0.9);
    }
    return MyColors.secondary;
  }
}

String getStringButton(HomeController controller) {
  return controller.bt.isConnected ? Strings.desconectar : Strings.conectar;
}

Color getTextButtonColor(HomeController controller) {
```

```
    return controller.bt.isConnected ? Colors.white : Colors.white;
  }
}
```