UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
Engenharia Elétrica – Ênfase em Sistemas de Energia e Automação

Gabriel do Prado Arthur

AUTOMAÇÃO RESIDENCIAL E DE IRRIGAÇÃO RURAL COM MONITORAMENTO E CONTROLE REMOTO VIA INTERNET UTILIZANDO ESP32

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos da Universidade de São Paulo como requisito parcial para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Rogério Andrade Flauzino

São Carlos
2025

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Arthur, Gabriel
A788a AUTOMAÇÃO RESIDENCIAL E DE IRRIGAÇÃO RURAL COM MONITORAMENTO E CONTROLE REMOTO VIA INTERNET UTILIZANDO ESP32 / Gabriel Arthur; orientador Rogério Andrade Flauzino. São Carlos, 2025.

Monografia (Graduação em Engenharia Elétrica com ênfase em Sistemas de Energia e Automação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2025.

1. ESP32. 2. MQTT. 3. automação. 4. IoT. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Nome: Gabriel do Prado Arthur

Título: "Automação residencial e de irrigação rural com monitoramento e controle remoto via internet utilizando ESP32"

Trabalho de Conclusão de Curso defendido e aprovado em 05 / 12 / 2025,

com NOTA 9,5 **(** Nove, cinco **), pela Comissão Julgadora:**

Prof. Associado Rogério Andrade Flauzino - Orientador SEL/EESC/USP

Prof. Dr. Fábio Romano Lofrano Dotto - SEL/EESC/USP

Eng. Ivan Talão Martins - Doutorando EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Professor Associado José Carlos de Melo Vieira Júnior

Gabriel do Prado Arthur

AUTOMAÇÃO RESIDENCIAL E DE IRRIGAÇÃO RURAL COM MONITORAMENTO E CONTROLE REMOTO VIA INTERNET UTILIZANDO ESP32

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos da Universidade de São Paulo como requisito parcial para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Rogério Andrade Flauzino

São Carlos
2025

Sumário

Resumo

Este trabalho projeta e implementa um sistema de automação residencial e rural baseado no ESP32 (DevKit V1), com monitoramento e controle remoto via internet. O projeto possui caráter educacional, orientado ao aprendizado prático de automação e IoT. No ambiente urbano, o sistema acionou corretamente as três lâmpadas por servidor web embarcado, aplicativo MQTT e botão local; no ambiente rural, três lâmpadas, uma válvula de irrigação e os sensores de umidade do solo e temperatura operaram de forma estável, exceto pelo deslocamento de 3 horas observado nas agendas automáticas. A comunicação utilizou Wi-Fi e o protocolo MQTT sobre TLS, com visualização dos dados no SCADA-LTS e em aplicativo móvel. Os testes demonstraram aquisição consistente de temperatura e umidade, e baixo custo operacional do sistema em modo automático, estimado em R$ 25,61/mês (R$ 307,29/ano). A metodologia compreendeu definição de requisitos, projeto eletrônico, desenvolvimento de firmware, integração com o *broker* e validação funcional. São apresentadas recomendações de segurança, incluindo o uso de TLS no MQTT e boas práticas de gestão de credenciais.
Palavras-chave: ESP32; MQTT; automação residencial; automação rural; internet das coisas (IoT).

Abstract

This work designs and implements a residential and rural automation system based on the ESP32 (DevKit V1), providing internet-enabled monitoring and remote control. Educational in scope, the project serves as a hands-on learning platform in automation and IoT. In the urban environment, the system successfully actuated three lamps via embedded web server, MQTT-based mobile application, and local push button; in the rural environment, three lamps, one irrigation valve, and soil-moisture and temperature sensors operated stably, except for a systematic 3-hour offset observed in automatic scheduling. Communication relies on Wi-Fi and the MQTT protocol over TLS, with data visualization in SCADA-LTS dashboards and a mobile application. Experimental tests showed reliable actuation and consistent acquisition of temperature and humidity, as well as a low operating cost in automatic mode, estimated at BRL 25.61 per month (BRL 307.29 per year). The methodology comprises requirements definition, electronic design, firmware development, MQTT broker integration, and functional validation. Security recommendations are provided, focusing on the use of TLS in MQTT and proper credential management.
Keywords: ESP32; MQTT; home automation; rural automation; Internet of Things (IoT).

Capítulo 1

Introdução

1.1 Contextualização

O avanço das tecnologias digitais tem alterado profundamente a forma como residências e propriedades rurais são gerenciadas. A combinação entre maior demanda por conforto, segurança e eficiência energética cria um cenário em que soluções manuais passam a ser insuficientes para lidar com rotinas cada vez mais complexas. Sistemas de automação permitem padronizar tarefas, reduzir erros humanos e disponibilizar informações em tempo real para o usuário, que passa a ter maior controle sobre o consumo de energia, o funcionamento de equipamentos e a supervisão de ambientes mesmo à distância. Nesse contexto, a automação deixa de ser um diferencial restrito a instalações de alto custo e se torna uma ferramenta de apoio à gestão cotidiana em diferentes perfis de usuários.

No meio rural, essa necessidade se torna ainda mais evidente quando se considera o manejo de água e energia em atividades agrícolas. Pequenas e médias propriedades frequentemente dependem de deslocamentos presenciais para acionar bombas, abrir ou fechar válvulas e verificar condições de solo e clima, o que consome tempo, combustível e recursos financeiros. A ausência de monitoramento sistemático da umidade do solo e da temperatura pode resultar em irrigações desnecessárias ou tardias, com impactos diretos sobre produtividade, desperdício de água e custo da energia elétrica. A integração entre sensoriamento, controle automático e supervisão remota surge, assim, como uma alternativa para tornar o uso desses recursos mais racional, previsível e alinhado às restrições econômicas típicas de ambientes não industrializados.

A automação residencial tem evoluído significativamente nas últimas décadas, impulsionada pelo desenvolvimento de redes sem fio, sensores inteligentes e microcontroladores embarcados. Nesse contexto, Gill, Yang, Yao e Lu (2009) destacam que a arquitetura de sistemas domésticos modernos tende a migrar de soluções centralizadas e com lógica fixa para modelos distribuídos, baseados em redes de sensores sem fio (WSNs), capazes de coletar, processar e transmitir informações de forma autônoma e descentralizada.

Segundo os autores, tecnologias como ZigBee desempenham papel fundamental nesse processo, por oferecerem uma solução de comunicação de baixo consumo energético, estrutura de rede em malha (mesh) e alta escalabilidade. A escolha do protocolo ZigBee deve-se à sua eficiência em aplicações que requerem monitoramento contínuo, como o controle de iluminação, sensores de presença, temperatura e segurança em ambientes residenciais. Esses recursos tornam a tecnologia adequada para ambientes em que o acesso à energia ou à internet pode ser limitado ou instável.

O artigo também enfatiza que, ao integrar sensores e atuadores com módulos de comunicação ZigBee, é possível construir uma infraestrutura doméstica capaz de se adaptar às preferências dos usuários e de operar de forma automatizada, com mínima intervenção humana. Essa flexibilidade permite que o sistema execute tarefas rotineiras, como acionar luzes ao detectar presença ou desligar equipamentos com base em horários predefinidos, elevando o nível de conforto e segurança do ambiente.

Gill, Yang, Yao e Lu (2009) dão ênfase à importância de adotar uma abordagem modular e de baixo custo no desenvolvimento de sistemas de automação residencial, especialmente para países em desenvolvimento. O uso de microcontroladores acessíveis, combinados a protocolos de comunicação eficientes como ZigBee, representa uma alternativa viável à automação comercial tradicional, que geralmente é restrita a usuários com maior poder aquisitivo. Essa democratização tecnológica possibilita a aplicação da automação em diferentes camadas sociais, ampliando seu impacto social e econômico. Dando continuidade, o panorama de Sriskanthan, Tan e Karande (2011) descreve arquiteturas típicas de automação centradas na interação entre smartphone e microcontrolador, e compara Bluetooth, ZigBee, GSM/SMS, Wi-Fi/Internet e EnOcean quanto a custo, alcance, consumo de energia e adequação a aplicações de baixa latência. Os autores ressaltam que a seleção tecnológica deve equilibrar simplicidade de implantação com requisitos de disponibilidade e tempo de resposta, favorecendo soluções baseadas em IP quando a interface remota e a integração com serviços em nuvem são prioritárias. Essa leitura sustenta a priorização da conectividade Wi-Fi em projetos que exigem visualização web e comando remoto contínuo.

À luz desse panorama, este trabalho adota a plataforma ESP32 (DevKit V1) com conectividade Wi-Fi e o protocolo MQTT como eixo de comunicação, combinando a leveza do modelo *publish/subscribe* para telemetria e comandos. No arranjo proposto, o nó urbano aciona três lâmpadas; o nó rural aciona três lâmpadas e uma válvula de irrigação e realiza leituras contínuas de umidade do solo e temperatura. Os dados são publicados em tópicos MQTT e exibidos em interface web, permitindo acompanhamento em tempo real e registro para análise. Considerando robustez e segurança, emprega-se MQTT sobre TLS com verificação do certificado do *broker* no cliente ESP32.

1.2 Objetivos

1.2.1 Objetivo Geral

Projetar, implementar e validar um sistema integrado de automação residencial e de irrigação rural, baseado na plataforma ESP32, com supervisão e comando remotos via internet por meio do protocolo MQTT, contemplando o controle de iluminação e da válvula de irrigação, a aquisição de umidade do solo e temperatura, a disponibilização de interface web para operação e a adoção de práticas de segurança (TLS e gestão de credenciais).

1.2.2 Objetivos Específicos

- Projetar a arquitetura de hardware para controle de lâmpadas e válvula de irrigação com ESP32;
- Implementar comunicação sem fio via Wi-Fi (2,4 GHz) utilizando o protocolo MQTT sobre TLS (porta 8883), com autenticação no *broker* e validação de certificado (CA) no cliente;
- Integrar os sensores de umidade do solo e de temperatura ambiente no ESP32 02(ESP32 da zona rural), implementar o envio periódico das leituras ao *broker* MQTT e disponibilizar visualização remota da temperatura e da umidade em *dashboard* web (SCADA-LTS) e aplicativo móvel;
- Validar o sistema, testando confiabilidade e funcionalidade;
- Avaliar o impacto da solução em termos de economia de energia.

1.3 Metodologia

A metodologia adotada neste trabalho é aplicada e experimental, com foco no desenvolvimento prático de um sistema funcional de automação. O projeto será dividido em etapas, iniciando-se com a definição dos requisitos e a seleção dos componentes de hardware e software. Em seguida, será realizada a montagem do circuito eletrônico com a plataforma ESP32, sensores de umidade do solo, sensor de temperatura e módulos relé para acionamento de lâmpadas e válvula de irrigação.

A programação será desenvolvida utilizando a IDE Arduino e o VS Code, com implementação de protocolos de comunicação como MQTT e TLS para transmissão de dados entre os dispositivos e o servidor na nuvem(Broker). Para o monitoramento remoto, será desenvolvida uma interface web hospedada no ESP32 destinado ao ambiente residencial. Adicionalmente, será disponibilizado uma conexão em um aplicativo móvel que agregará os dados publicados pelos nós ESP32 (residencial e rural), permitindo ao usuário visualizar,

em tempo real, o estado dos sensores e executar comandos de acionamento sobre os atuadores.

O sistema será dividido entre dois módulos ESP32: um dedicado à automação residencial (controle de três lâmpadas) e outro à automação rural (três lâmpadas, uma válvula de irrigação, um sensor de umidade e um sensor de temperatura). Cada módulo será testado individualmente em bancada, seguido de testes integrados simulando o ambiente real de operação.

1.4 Estrutura do Trabalho

Este trabalho está organizado em seis capítulos, além de apêndices que reúnem materiais complementares.

O **Capítulo 1** apresenta a introdução, com a contextualização do tema, a motivação do estudo, os objetivos geral e específicos, a metodologia adotada em nível macro e a organização do texto.

O **Capítulo 2** traz a revisão bibliográfica, discutindo trabalhos relacionados à automação residencial e rural, Internet das Coisas (IoT).

O **Capítulo 3** aborda a fundamentação teórica necessária para o entendimento do projeto, incluindo conceitos de automação, redes de computadores, protocolos de comunicação, arquitetura do ESP32 e noções de segurança em sistemas conectados.

No **Capítulo 4** é apresentada a metodologia do projeto, detalhando os materiais empregados, a arquitetura proposta, o diagrama do sistema, a configuração do hardware e o desenvolvimento do software.

O **Capítulo 5** reúne os resultados e discussões, apresentando os testes realizados, os dados obtidos e a análise crítica quanto à eficiência, funcionalidade e confiabilidade da solução implementada.

o **Capítulo 6** apresenta as conclusões do trabalho, sintetizando os principais resultados, destacando as contribuições do projeto, apontando as limitações encontradas e indicando sugestões para trabalhos futuros.

Os **Apêndices A, B, C e D** reúnem os códigos-fonte e arquivos de configuração mais extensos do projeto (módulos de controle, comunicação e automação), de forma a não sobrecarregar o corpo principal do texto.

Capítulo 2

Revisão bibliográfica

O avanço recente da Internet das Coisas (IoT) tem favorecido arquiteturas de automação residencial baseadas em sensores distribuídos, conectividade *wireless* e serviços em nuvem. Em um estudo voltado para domicílios no contexto do Oriente Médio, Al-Kuwari et al(2018) propõem uma plataforma de automação que combina sensores de temperatura, umidade e presença com atuadores conectados a uma rede Wi-Fi doméstica. O sistema utiliza um *gateway* central para coletar os dados, aplicar regras simples de automação e disponibilizar ao usuário uma interface de monitoramento e comando remoto. Os autores destacam a viabilidade de soluções de baixo custo, baseadas em protocolos leves, para controlar múltiplas cargas em tempo real.

Na mesma linha, Pravalika e Prasad (2019) desenvolvem um sistema de monitoramento residencial e acionamento de dispositivos utilizando o microcontrolador ESP32 como nó principal. A proposta explora o uso da conectividade Wi-Fi integrada para publicar medidas ambientais e receber comandos de acionamento a partir de uma aplicação móvel. O trabalho reforça o papel do ESP32 como plataforma adequada para integrar sensores, relés e interface com o usuário em um único módulo, o que reduz a quantidade de hardware auxiliar e simplifica a instalação em residências já em uso.

Enquanto esses trabalhos se concentram na automação residencial, Aghenta e Iqbal (2019) avançam para uma arquitetura de supervisão mais próxima de sistemas SCADA, também com foco em baixo custo. Os autores descrevem um sistema em que o ESP32 atua como *gateway* e unidade remota (RTU), publicando dados de temperatura, umidade, pressão e luminosidade via protocolo MQTT para um servidor local baseado em Raspberry Pi, onde são executados o *broker*, o Node-RED e o banco de dados. A interface supervisória é construída na própria ferramenta de fluxos, permitindo monitorar os sinais em tempo real e acionar cargas remotamente.

Do ponto de vista de integração de sensores e serviços, Lekić e Gardašević (2018) analisam o uso do Node-RED como plataforma de orquestração de dados em aplicações de IoT. O estudo mostra como sensores heterogêneos podem ser conectados ao Node-RED por diferentes protocolos, com destaque para MQTT, e como a ferramenta facilita a criação de

dashboards e rotinas de processamento por meio de blocos gráficos. Os resultados indicam que o Node-RED reduz o esforço de desenvolvimento da camada de aplicação e favorece a integração com serviços de banco de dados e nuvem

A questão da segurança em arquiteturas de supervisão baseadas em nuvem é discutida por Sajid, Abbas e Saleem (2016). Os autores realizam uma revisão do estado da arte em sistemas SCADA assistidos por IoT, identificando ameaças como interceptação de mensagens, falsificação de comandos e ataques de negação de serviço. O estudo enfatiza a necessidade de empregar criptografia de ponta a ponta, autenticação robusta e segmentação de redes para mitigar riscos, especialmente quando protocolos leves como MQTT são expostos à internet pública. Essas recomendações fundamentam o uso de canais TLS, credenciais específicas para o *broker* e separação entre rede local e acesso externo no desenvolvimento do sistema apresentado neste TCC.

Em síntese, os trabalhos analisados convergem na utilização de microcontroladores conectados, protocolos leves como MQTT e ferramentas de orquestração como Node-RED para construir soluções de automação residencial e de supervisão de baixo custo. A contribuição deste TCC se apoia nessas evidências ao integrar, em uma mesma arquitetura, um nó urbano e um nó rural baseados em ESP32, combinando automação de iluminação, controle de irrigação e monitoramento de variáveis ambientais, com supervisão remota via internet e foco em escalabilidade e segurança.

Na literatura de agricultura de precisão e irrigação inteligente, destacam-se ainda propostas que utilizam sensores de umidade do solo e temperatura para subsidiar decisões de manejo hídrico. Nessas soluções, o solo é monitorado continuamente, e os dados são enviados a um servidor ou plataforma em nuvem, que pode aplicar limiares fixos ou algoritmos mais sofisticados para decidir sobre o acionamento de válvulas e bombas. O objetivo recorrente é reduzir o consumo de água e energia, ao mesmo tempo em que se mantém a umidade em faixas adequadas ao desenvolvimento das culturas, aproximando o processo de um controle orientado por dados, em contraste com práticas baseadas apenas em observação visual ou experiência empírica do operador.

Em relação a esse conjunto de trabalhos, o presente TCC se insere como uma aplicação híbrida que combina automação residencial e irrigação rural em uma mesma infraestrutura de supervisão. Ao utilizar um nó dedicado à área rural, equipado com sensores de umidade do solo e temperatura e com capacidade de comandar uma válvula de irrigação, o sistema explora os princípios de monitoramento contínuo e controle remoto discutidos na literatura de irrigação inteligente, mas integrados ao ecossistema de automação residencial e à pilha tecnológica baseada em ESP32, MQTT, Node-RED e SCADA-LTS. Essa integração reforça a relevância do monitoramento de irrigação como foco central do trabalho, conectando-o diretamente às tendências atuais de IoT aplicada ao campo.

Capítulo 3

Fundamentação Teórica

3.1 Internet das Coisas (IoT)

A Internet das Coisas (IoT) pode ser entendida como o ecossistema em que objetos físicos dotados de identificação, sensoriamento, processamento e conectividade passam a interagir entre si e com serviços em nuvem para entregar funcionalidades úteis ao usuário final, nesse contexto, cada dispositivo publica dados do ambiente, recebe comandos e coopera com outros para compor serviços de maior valor. A literatura destaca quatro pilares recorrentes: *coisas* (dispositivos), *comunicação* (redes e protocolos), *computação* (processamento local e em nuvem) e *serviços* (aplicações e integrações). Do ponto de vista arquitetural, é comum organizar um sistema IoT em camadas: percepção (sensores e atuadores), rede (meios físicos e protocolos de enlace/roteamento), transporte/aplicação (TCP/IP e protocolos como HTTP, MQTT ou CoAP) e serviços (armazenamento, análise e interfaces). Essa separação ajuda a isolar responsabilidades: a camada de percepção adquire dados e executa comandos, a camada de rede garante entrega, a camada de aplicação define semântica das mensagens e a de serviços agrega visualização, persistência e automações.

O protocolo MQTT tem ganhado destaque em cenários de monitoramento e controle por sua leveza e modelo *publish/subscribe*. Em vez de endereçar dispositivos ponto a ponto, os nós publicam mensagens em *tópicos*, e assinantes recebem apenas o que lhes interessa. QoS selecionável, *keep-alive* e *last-will* contribuem para resiliência em redes sem fio. Em sistemas conectados à internet pública, recomenda-se o uso de TLS para confidencialidade e integridade, e adicionalmente, autenticação robusta no *broker*. Em ambientes embarcados, boas práticas incluem armazenar credenciais de forma segura, validar o certificado da autoridade (CA) e, quando aplicável, empregar mecanismos do hardware para proteção do firmware Bahga e Madisetti (2015).

3.2 Automação

A ideia de fazer dispositivos operarem sozinhos é antiga. Na Antiguidade, Herão de Alexandria descreveu mecanismos capazes de abrir portas de templos automaticamente ao acender o fogo no altar, explorando variações de pressão de ar e água; são exemplos de autômatos com lógica puramente mecânica descritos em seus tratados *Pneumatica* e *Automata* (Herão de Alexandria). Na tradição islâmica medieval, al-Jazarī catalogou, em 1206, dispositivos hidráulicos e autômatos musicais no *Compêndio de dispositivos engenhosos*, combinando potência hidráulica, temporização e sequências mecânicas para executar tarefas de forma autônoma (al-Jazarī, 1206).

Nos séculos XVIII e XIX, a automação ganha duas inflexões decisivas. A primeira é a programabilidade: o tear de Jacquard (1804–1805) usa cartões perfurados encadeados para tecer padrões complexos sem intervenção constante do operador, antecipando o controle por instruções discretas que mais tarde influenciaria a própria computação (Jacquard, 1804–1805). A segunda é o controle por realimentação: James Watt aplica, em 1788, o governador centrífugo ao motor a vapor para regular a velocidade automaticamente, fechando o ciclo entre medição e atuação e estabelecendo um marco do controle automático na Revolução Industrial (Watt, 1788).

No século XX, a automação industrial incorpora eletrônica e controle programável. Em 1961, o *Unimate* entra em operação numa linha da General Motors, tornando-se o primeiro robô industrial em serviço: um manipulador capaz de repetir sequências perigosas e repetitivas com precisão, apontando para a robótica de manufatura (Devol e Engelberger, 1961). Os mesmos princípios que atravessam essa trajetória — sequenciamento de ações, instruções explícitas, realimentação e segurança operacional — aparecem aqui em escala embarcada. No sistema proposto, nós com ESP32 executam o ciclo senso–decisão–ação: sensores fornecem dados, a lógica organiza comandos e estados por tópicos MQTT, e os atuadores realizam as tarefas de iluminação e irrigação de forma previsível.

3.3 Plataforma ESP32

A plataforma ESP32 tem se consolidado como base de projetos embarcados conectados, graças à combinação de processador de 32 bits, Wi-Fi 802.11 b/g/n, Bluetooth clássico/BLE e um conjunto amplo de periféricos (GPIOs, ADCs, DACs, PWM, SPI, I^2C e UART). Segundo a própria documentação técnica da fabricante, o chip foi projetado para aplicações de Internet das Coisas com modos finos de economia de energia, operação em 2,4 GHz até 150 Mb/s e integração que reduz a quantidade de componentes externos (Espressif Systems, 2021). Esses recursos permitem construir nós compactos, de baixo custo e com conectividade nativa, o que atende diretamente às demandas deste trabalho.

Do ponto de vista histórico, o ESP32 sucede o ESP8266 e marca a transição, na família

da Espressif, de soluções "Wi-Fi com microcontrolador auxiliar" para um *System-on-Chip* mais completo, com CPU dual-core Tensilica LX6, coprocessador de baixo consumo e blocos analógicos integrados. Essa evolução incorporou BLE, ampliou o número de GPIOs e trouxe periféricos que normalmente exigiam circuitos externos, tornando o projeto de placas e produtos finais mais simples. O principal motivo prático de adoção é a relação custo-benefício: em uma única placa, o ESP32 oferece conectividade Wi-Fi e BLE integrada, clock de até 240 MHz, SRAM na casa de centenas de kilobytes e periféricos suficientes para a maioria das aplicações de automação. Em comparação com plataformas clássicas baseadas em AVR, como Arduino Uno/Nano (8 bits, 16 MHz, SRAM na ordem de kilobytes e sem rede nativa), o ESP32 entrega mais processamento, mais memória e conectividade embutida, reduzindo a necessidade de módulos adicionais e, portanto, o custo total do sistema. Vale destacar ainda que a velocidade de desenvolvimento é favorecida por uma comunidade ampla, documentação extensa e exemplos oficiais. A combinação de hardware acessível, conectividade embarcada, desempenho superior às placas 8-bit tradicionais e um ecossistema maduro explica por que o ESP32 se tornou a escolha preferencial em projetos de IoT e automação nos últimos anos (Espressif Systems, 2021).

3.4 Segurança em Sistemas Embarcados

No plano da aplicação, o protocolo MQTT atende bem a dispositivos embarcados por combinar modelo *publish/subscribe* com baixa sobrecarga de rede. A organização por tópicos permite separar telemetria, comandos e estados, enquanto o controle de qualidade de serviço (QoS) e o *last will* ajudam a lidar com links instáveis. Em termos de segurança, a prática corrente é autenticar clientes no *broker* e restringir permissões por tópico, evitando publicação ou assinatura indevida. Uma apresentação estruturada desses elementos, organizados em camadas, papéis e padrões de troca de mensagens, é apresentada no manual didático de Bahga e Madisetti (2015), que também discute a integração do MQTT com serviços em nuvem e painéis de visualização.

A proteção do canal é garantida pelo protocolo TLS, sucessor do SSL. De forma geral, o TLS negocia algoritmos, autentica o servidor por meio de certificados X.509 e estabelece chaves de sessão. A cifragem simétrica protege o tráfego de dados, enquanto a criptografia assimétrica é utilizada para a troca de chaves e para a autenticação. Esse conjunto de mecanismos assegura, de maneira eficiente, os três pilares clássicos da segurança da informação: confidencialidade, integridade e autenticidade. Tanenbaum (2011) descreve essa combinação de técnicas, como o uso de chaves públicas, certificados, *handshake* e HMAC, como a base prática para o estabelecimento de sessões seguras em redes públicas.

Na prática embarcada, as duas camadas se complementam: o MQTT organiza a comunicação por tópicos entre sensores e atuadores, enquanto o TLS impede interceptação e adulteração do conteúdo. A combinação recomendada é autenticar cada dispositivo no

broker, validar a cadeia de certificação do servidor e manter políticas mínimas de acesso por tópico. Com isso, reduz-se a superfície de ataque sem penalizar a latência de forma relevante, preservando o caráter leve do MQTT ao mesmo tempo em que se herda a base criptográfica madura do TLS (Bahga e Madisetti, 2015; Tanenbaum, 2011).

Capítulo 4

Metodologia do Projeto

4.1 Materiais e Equipamentos

Os itens listados foram selecionados para compor uma arquitetura IoT completa, do processamento local à conectividade externa, passando por sensoriamento e atuação. O **ESP32 (DevKit V1)** cumpre o papel de nó de borda, concentrando processamento e conectividade Wi-Fi para telemetria e comandos. O **RTC DS3231** provê temporização precisa e estável ao longo do dia, requisito para agendamentos, e também atua como sensor de temperatura. A malha de **sensoriamento** inclui o **sensor capacitivo de umidade do solo**, a **camada de atuação** é realizada pelos **módulos de relé de 4 canais**, que oferecem comutação elétrica isolada para cargas de iluminação e para a **válvula solenóide** do sistema de irrigação. A **infraestrutura de rede** é atendida pelo **roteador Mercusys (MW301R)** para a LAN local e pelo **ELSYS AmpliMax**, que integra modem e antena direcionais para levar conectividade 3G/4G a áreas com cobertura limitada, disponibilizando Ethernet ao ESP32 02(ESP 32 rural). O **cabeamento** é composto por 80m de fio de 1,5mm², que asseguram a conexão elétrica dos módulos, até as lâmpadas e a válvula, e por um cabo de rede RJ45 de 30,m, responsável por interligar o ELSYS AmpliMax ao roteador Mercusys (MW301R). Para prototipagem e testes, as **protoboards**, o **kit de jumpers**, os **pushbuttons** e o **kit de LEDs** facilitam montagem, depuração e sinalização de estados; as **fontes ajustáveis para protoboard** alimentam os circuitos de baixa tensão com 3,3/5 V. Em conjunto, esses materiais viabilizam um sistema híbrido (residencial e rural) capaz de monitorar variáveis ambientais, acionar cargas e operar rotinas automáticas com sincronização temporal e acesso remoto seguro via MQTT, conforme resumido na Tabela 4.1.

Tabela 4.1: Materiais e custos do protótipo

Item	**Quantidade**	**Preço total (R$)**
ESP32 DevKit V1	2	77,98
Fontes ajustáveis para protoboard	2	17,94
Push-buttons	3	5,00
RTC DS3231	2	78,00
Sensor capacitivo de umidade do solo	1	3,89
Cabo 1,5 mm^2 (energia)	80 m	104,00
Válvula solenóide	1	20,44
Módulos relé 4 canais	2	100,00
Cabo de rede RJ45	30 m	22,56
Roteador Mercusys MW301R	1	78,00
Elsys AmpliMax	1	679,90
Kit de 10 LEDs coloridos	1	10,00
Kit de jumpers	1	30,00
Protoboards	2	24,12
Total		**1.251,83**

4.2 Diagrama do Sistema

A arquitetura proposta é composta por dois nós ESP32 fisicamente separados: um nó *urbano* (“ESP32 01”) instalado na residência da cidade e um nó *rural* (“ESP32 02”) instalado no sítio, a aproximadamente 7 km de distância. Ambos controlam três lâmpadas; o nó rural acrescenta a automação da irrigação e a telemetria de umidade do solo e temperatura. A conectividade local é feita via Wi-Fi; o transporte de dados e o controle remoto utilizam o protocolo MQTT com canal seguro (TLS), permitindo supervisão e por meio de um serviço de *dashboard*/SCADA.

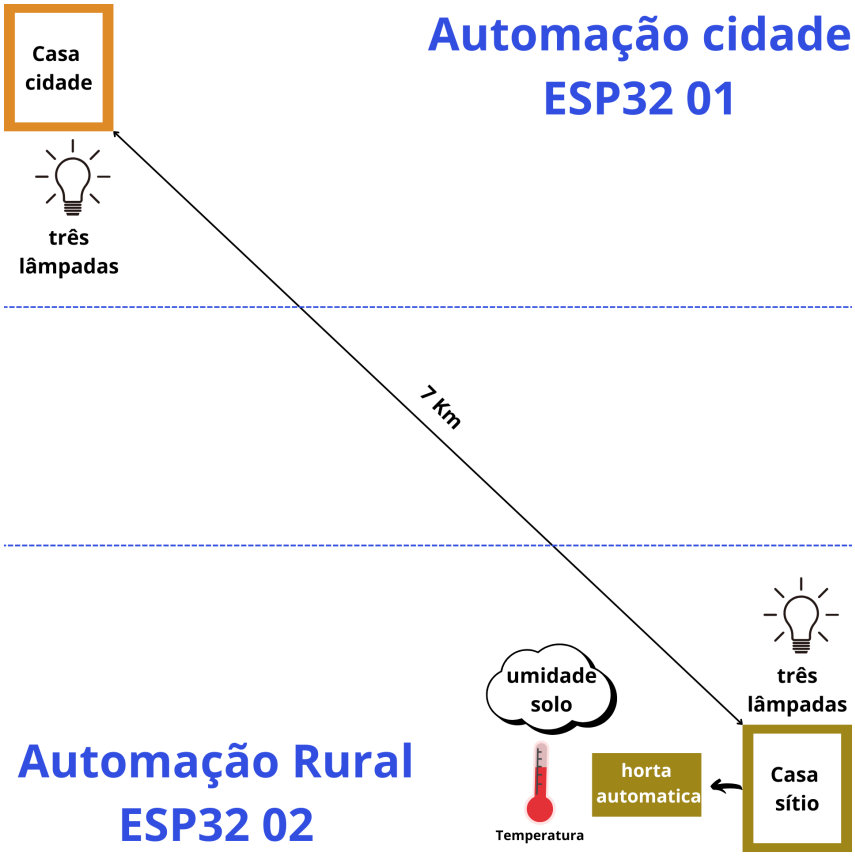


Figura 4.1: Topologia física: nó urbano (ESP32 01) e nó rural (ESP32 02)

O diagrama da Figura 4.1 evidencia os dois domínios: no lado urbano, o ESP32 01 aciona três lâmpadas; no lado rural, o ESP32 02 aciona três lâmpadas e a válvula solenóide da horta. Sensores de temperatura complementam o contexto ambiental. A sincronização temporal é garantida por RTC (DS3231), essencial para registros e rotinas programadas.

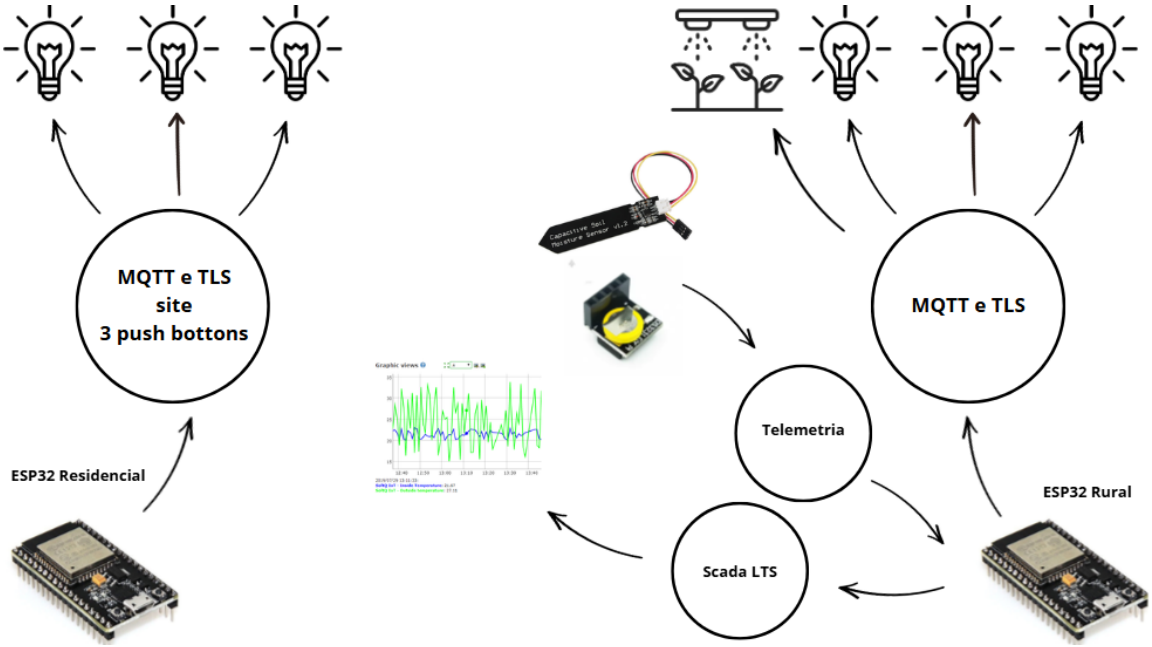


Figura 4.2: Fluxo lógico de telemetria e controle do projeto

A Figura 4.2 detalha o ciclo de dados:

1. **Aquisição local:** o ESP32 02 lê a umidade do solo (sensor capacitivo) e a temperatura; ambos os nós mantêm relógio estável com o DS3231 para carimbar eventos.

2. **Publicação MQTT:** as leituras e estados são publicados em tópicos hierárquicos (por exemplo, `tcc/sitio/umidade`, `tcc/cidade/lampadas/estado`). Níveis de QoS e retenção podem ser ajustados conforme a criticidade do dado.

3. ***Broker* e segurança:** a sessão ocorre sobre TLS na porta segura do serviço, com verificação da CA pelo cliente, assegurando confidencialidade, integridade e autenticidade dos pontos.

4. **Supervisão e histórico:** o SCADA-LTS/*dashboard* assina os tópicos de interesse para fazer os gráficos séries e armazenar histórico.

5. **Atuação:** os ESP32 consomem os comandos e acionam os módulos de relé (iluminação e válvula), respeitando lógicas locais como *modo automático*, e janelas de horário.

4.3 Configuração do Hardware

4.3.1 Protótipo urbano (ESP32-01): acionamento local e remoto

O protótipo urbano integra um **ESP32 DevKit V1** em protoboard, um módulo de alimentação dedicado só para ele com 5V, botões de teste e um **módulo de relés de 4 canais** alimentado por 5V para comandar as cargas residenciais (lâmpadas). A montagem valida a arquitetura do sistema: leitura de entradas locais, lógica de controle embarcada e comutação das saídas, preparando a integração com a infraestrutura de nuvem via *Wi-Fi*/MQTT.

Composição do protótipo.

- **ESP32 DevKit V1**: controlador principal, responsável pelo ciclo de leitura das entradas, temporizações e publicação/assinatura de tópicos MQTT.
- **Protoboard com alimentação**: fornece trilhos de 5 V/3,3 V para lógica e periféricos; LED de diagnóstico para verificação rápida do estado.
- **Botões de teste**: permitem acionar localmente as saídas e validar o *firmware* sem a interface remota.
- **Módulo de relés (4 canais)**: estágio de potência para o acionamento das cargas; recebe sinais de nível lógico do ESP32 e isola a comutação das lâmpadas.
- **Rede de desacoplamento**: capacitores para estabilidade dos sinais durante chaveamentos.
- **RTC DS3231**: relógio de tempo real com sensor de temperatura integrado.

Funcionamento: no *firmware*, os botões locais e os comandos recebidos por MQTT atualizam o estado das saídas digitais. O módulo de relés comuta as três lâmpadas previstas para o nó urbano.

Observações de integração e segurança.

- Manter o aterramento comum entre a lógica e o módulo de relés, respeitando a isolação recomendada pelo fabricante do módulo de relés.
- Separar fisicamente os condutores de baixa tensão dos condutores das cargas de rede.

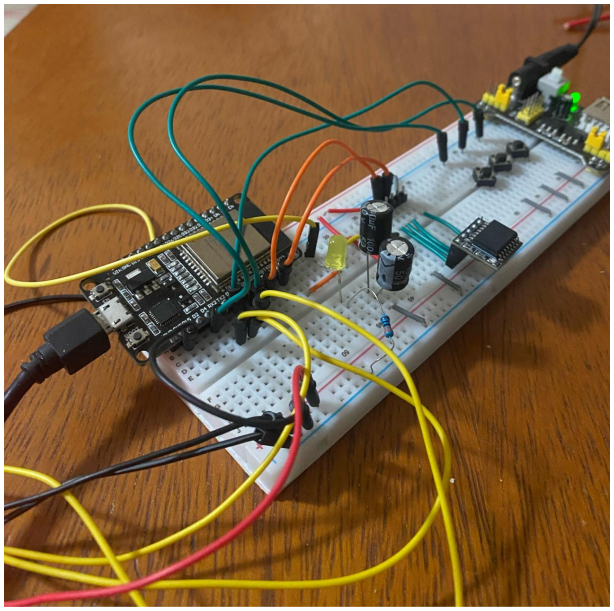


Figura 4.3: ESP32 01 na protoboard, LED que indica modo automatico.

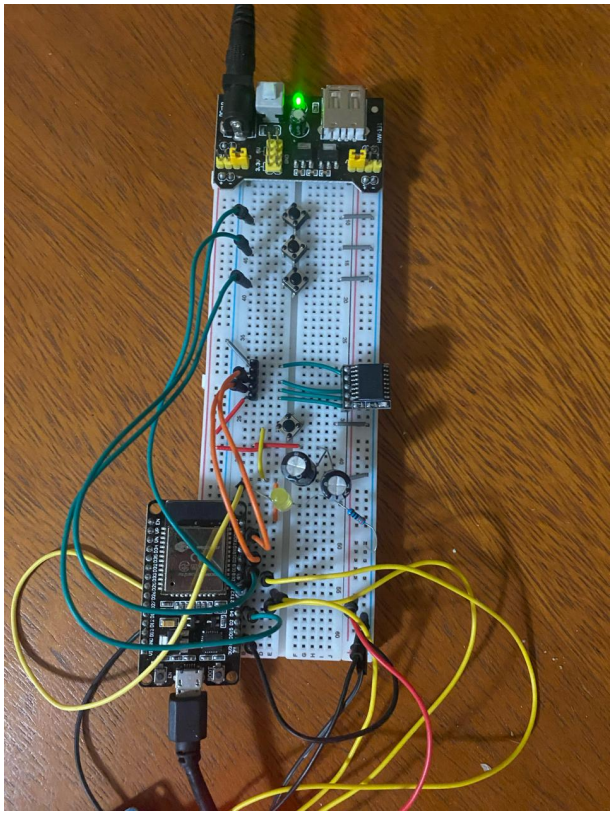


Figura 4.4: Vista superior do arranjo: ESP32 01, fileira de botões de teste e periféricos conectados ao barramento.

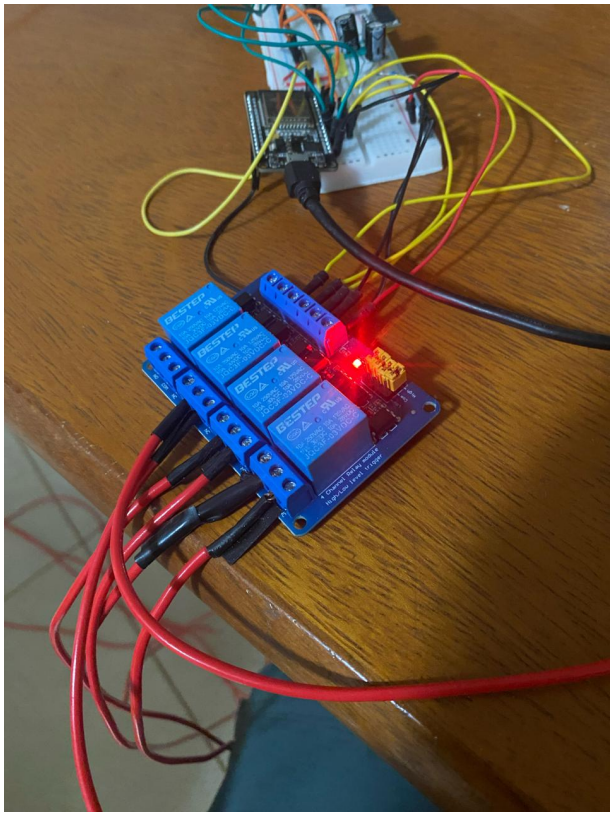


Figura 4.5: Módulo de relés interligado ao ESP32 01.

4.3.2 Protótipo rural (ESP32-02): irrigação e iluminação

O nó rural implementa um **ESP32 DevKit V1** em protoboard conectado a um **módulo de relés de 4 canais**, preparado para o acionamento das cargas do sítio (três lâmpadas e uma válvula solenoide). A comunicação de controle é feita via *Wi-Fi*/MQTT (com TLS quando ativado), integrando-se ao fluxo Node-RED.

Composição do protótipo.

- **ESP32 DevKit V1**: unidade de controle, responsável por ler comandos remotos (MQTT) e aplicar rotinas locais de segurança (temporizações e estados).
- **Protoboard**: fornece trilhos de 5 V/3,3 V para lógica e periféricos;
- **Módulo de relés (4 canais)**: estágio de comutação das cargas de campo (válvula e iluminação), isolando a lógica do lado de potência.
- **Cabeamento de bancada**: organização que prioriza separação entre sinais de controle e condutores de carga, facilitando medições e depuração.
- **RTC DS3231**: relógio de tempo real com sensor de temperatura integrado; fornece base de tempo estável para agendamentos e carimbo de tempo das leituras. Comuni-

cação via I^2C (SDA/SCL) em 3,3 V, com bateria de reserva (CR2032) para manter a hora em caso de falta de energia.

- **Sensor de umidade do solo (HW390)**: módulo de três pinos conectado pelo *fio amarelo* ao sinal analógico do ESP32 (saída `AO` do módulo), com **GND** ao terra comum e **VCC** em **3,3 V** (tensão de operação). Utilizado para estimar o teor de umidade do solo.

- **Roteador WiFi Mercusys**: atua como ponto de acesso local para o nó rural, fornecendo a rede WiFi utilizada pelo ESP32 e pelo notebook/smartphone de supervisão. Realiza a função de *gateway* entre a LAN do protótipo e a Internet, encaminhando o tráfego MQTT/TLS até o servidor remoto.

- **Amplimax**: equipamento de comunicação de longa distância instalado na área rural, responsável por estabelecer o enlace 4G com a operadora celular e criar a rede IP que atende o nó agrícola (ESP32). Opera como roteador de borda, garantindo conectividade estável para o envio de telemetria e o recebimento de comandos remotos.

Funcionamento: O *firmware* do ESP32 recebe comandos do *broker* MQTT para abrir/fechar a válvula e acionar a iluminação. Este nó está preparado para integrar sensores de **umidade do solo** e **temperatura**.

Integração e segurança.

- **Elétrica**: manter terra comum entre a lógica e o módulo de relés, respeitando a isolação do fabricante;

- **Comunicação**: quando em produção, utilizar MQTT com TLS (porta 8883), autenticação de usuário/senha e certificado válido.

A Figura 4.6 apresenta o ESP32-02 montado em protoboard e interligado ao módulo de relés, enquanto a Figura 4.7 detalha o estágio de comutação das cargas. A Figura 4.8 ilustra o sensor de umidade do solo HW-390 utilizado nos ensaios e as Figuras 4.9 e 4.10 mostram a válvula de irrigação empregada no protótipo. Por ultimo as Figuras 4.11 e 4.12 mostram respectivamente o roteador mercusys e o amplimax instalado

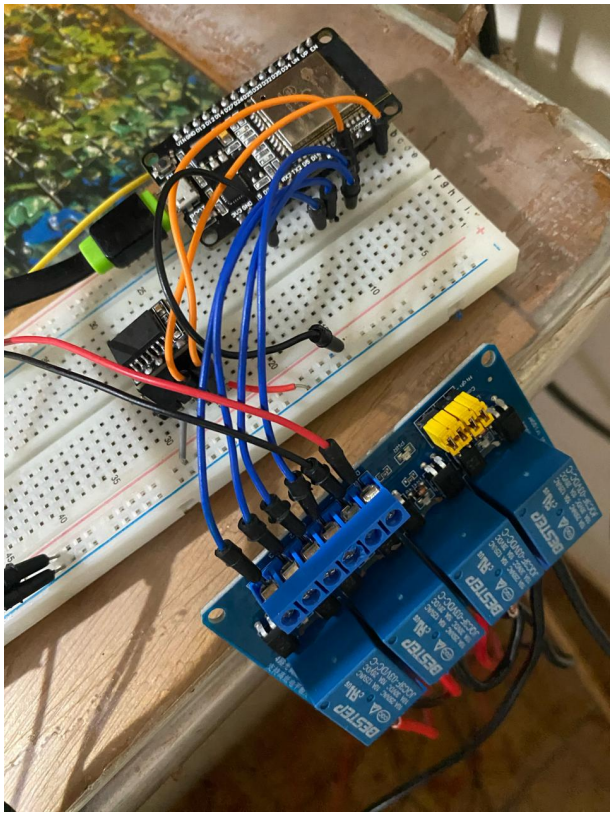


Figura 4.6: ESP32 02 em protoboard interligado ao módulo de relés.

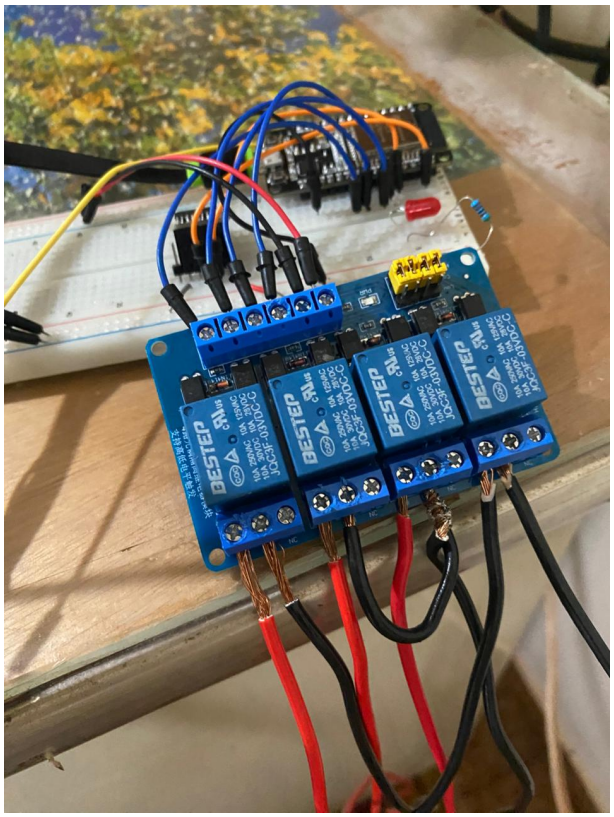


Figura 4.7: Detalhe do módulo de relés (4 canais) com retornos de carga.

Figura 4.8: Sensor de umidade do solo (HW390).

Figura 4.9: Válvula de irrigação Vista Frontal

Figura 4.10: Válvula de irrigação Vista Lateral

Figura 4.11: Amplimax instalado

Figura 4.12: Roteador Mercusys configurado.

4.3.3 Configuração do roteador para encaminhamento de porta segura (TLS)

Para permitir a comunicação remota entre os módulos ESP32 e o servidor MQTT instalado na máquina virtual, foi necessário realizar a configuração do roteador da operadora Vivo, responsável pela rede local do sistema. O objetivo foi liberar a **porta 8883/TCP**, utilizada pelo protocolo **MQTT sobre TLS (MQTTS)**, garantindo a criptografia e a integridade dos dados trafegados entre os dispositivos e o servidor.

Procedimento de configuração: O acesso ao roteador foi feito por meio do navegador, digitando o endereço IP local do equipamento. Em seguida, acessou-se o menu **"Configurações → Rede Local"**, onde o sistema exige autenticação (Figura 4.13). Após o login com o nome de usuário e senha do administrador, foi aberta a aba **"Encaminhamento de Porta"**, responsável por redirecionar conexões externas a dispositivos específicos da rede interna.

Na tela de encaminhamento, foi criada a regra para o serviço `MQTT_TLS`, utilizando o protocolo **TCP**, com a **porta externa e interna 8883**, e o **endereço IP interno 192.168.15.60**, correspondente à máquina virtual onde o *broker* Mosquitto está hospedado (Figura 4.14). Dessa forma, todo pacote destinado à porta 8883 do roteador é

automaticamente redirecionado para o servidor MQTT dentro da rede local.

Importância da configuração: Esse procedimento é essencial para o funcionamento da comunicação remota no projeto, pois:

- possibilita que os dispositivos ESP32 (urbano e rural) publiquem e recebam mensagens MQTT através da internet;
- mantém a camada de segurança do TLS ativa, evitando interceptações e adulterações no tráfego de dados;
- permite a integração entre o ambiente físico de automação e o sistema de supervisão hospedado na nuvem.

Boas práticas de segurança.

- Utilizar senha de administrador forte no roteador para impedir acesso não autorizado às configurações.
- Manter a porta 8883 exclusiva para o tráfego do serviço MQTT seguro.
- Monitorar o *log* de conexões no roteador e no servidor Mosquitto para detectar acessos suspeitos.

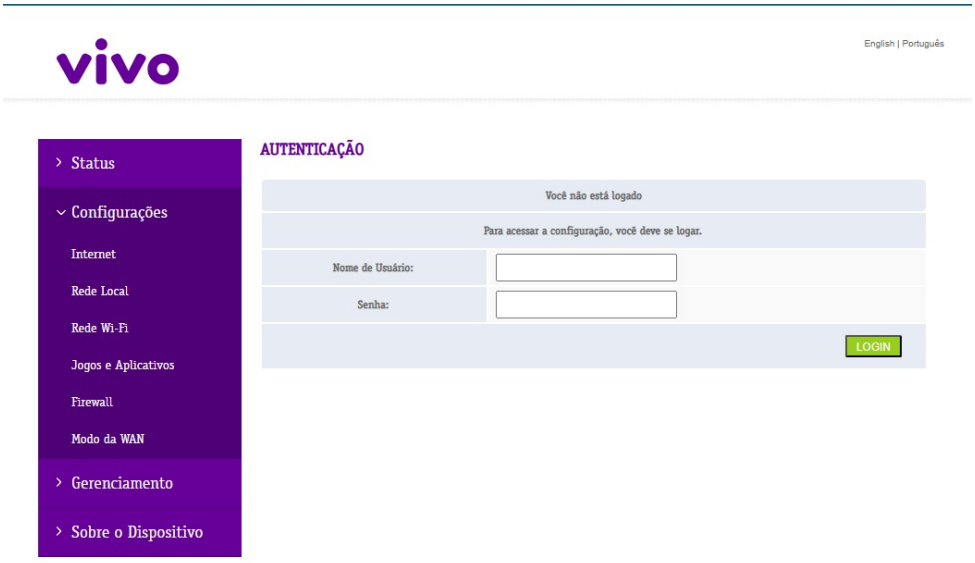


Figura 4.13: Tela de autenticação do roteador Vivo antes do acesso às configurações de rede.

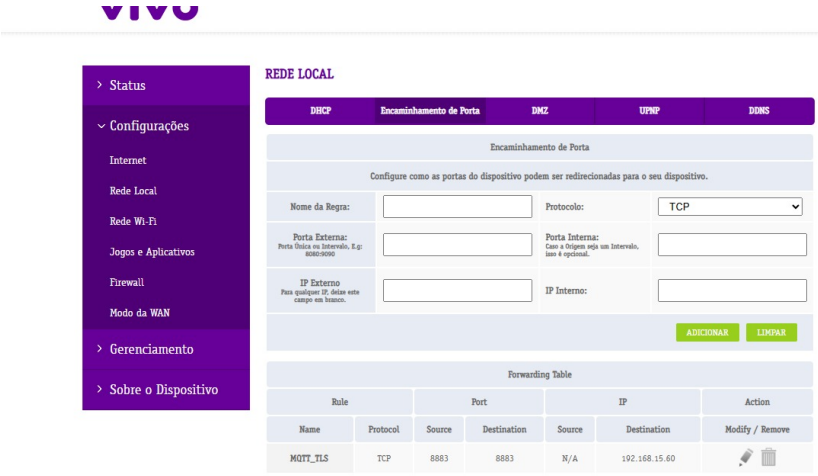


Figura 4.14: Criação da regra de encaminhamento da porta 8883/TCP para o servidor interno (IP 192.168.15.60).

4.4 Programação dos Módulos

4.4.1 Firmware e servidor web do nó urbano (ESP32 01)

O nó urbano é implementado com um microcontrolador ESP32 DevKit V1, responsável pelo acionamento de três lâmpadas instaladas em ambiente residencial. O firmware desenvolvido integra, em um único dispositivo, a conexão Wi-Fi, a comunicação MQTT segura com o *broker*, o controle local por botões físicos e um servidor web embarcado para acionamento via navegador, o módulo DS3231 é utilizado como relógio de tempo real (RTC) para viabilizar o modo automático baseado em janela horária.

O ESP32 opera como estação Wi-Fi, conectando-se ao roteador doméstico por meio da função WiFi.begin(). Após o estabelecimento do enlace na camada de enlace/rede, o firmware cria um cliente TLS (WiFiClientSecure) e inicializa a biblioteca PubSubClient para comunicação MQTT na porta 8883. O certificado da autoridade certificadora do *broker* é armazenado no código em formato PEM e carregado em tempo de execução com setCACert(), permitindo a autenticação do servidor e a criptografia de todos os *frames* MQTT. A sessão MQTT ainda é protegida por autenticação com usuário e senha, de modo que somente dispositivos autorizados conseguem publicar e assinar tópicos do projeto.

A organização dos tópicos MQTT segue a hierarquia tcc/esp01/casa/. Foram definidos quatro tópicos de comando, destinados a receber mensagens de aplicações externas (aplicativo MQTT no smartphone), a saber: tcc/esp01/casa/l1, l2, l3 e auto. De forma complementar, o nó publica seu estado em tópicos de telemetria do tipo tcc/esp01/casa/state/varivel, permitindo que qualquer cliente reconstrua o estado atual das saídas digitais. Todas as publicações de estado utilizam a opção *retain*, de maneira que o último valor permaneça armazenado no *broker* e seja reenviado automaticamente a

novos clientes assinantes. Configura-se uma mensagem de status (tcc/esp01/casa/status com os valores ‘‘online’’/‘‘offline’’), que indica aos supervisórios quando o nó urbano perde a conexão com o servidor.

O mapeamento de hardware associa os GPIOs do ESP32 aos três relés (LAMP1, LAMP2 e LAMP3), aos botões de comando local e a um LED indicador do modo automático. O firmware mantém variáveis booleanas para representar o estado de cada lâmpada (estadoLamp1, estadoLamp2 e estadoLamp3) e quando uma mensagem MQTT é recebida, a função de *callback* interpreta o *payload* textual (‘‘on’’, ‘‘off’’, ‘‘1’’, ‘‘0’’, etc.), atualiza as variáveis internas e aciona imediatamente os respectivos GPIOs com digitalWrite(). Em seguida, a função publishState() publica o novo estado em todos os tópicos de telemetria, garantindo a sincronização entre o nó físico, o painel MQTT.

Para suportar o modo automático, o ESP32 é conectado a um módulo RTC DS3231 via barramento I^2C. O relógio é inicializado no *setup* e, a cada iteração do laço principal, a função dentroJanela_BRT_1830a0500() consulta o horário em formato BRT. Quando o modo automático está habilitado e o horário corrente encontra-se na janela 18h30--5h00, o firmware força as três lâmpadas para o estado ligado; fora dessa janela, as saídas são desligadas. Assim, mesmo na ausência de conexão com a nuvem, o nó urbano mantém um comportamento previsível, alinhado a uma rotina de iluminação residencial noturna.

Vale destacar ainda que além da interface MQTT, o ESP32 disponibiliza um servidor HTTP embarcado na porta 80, responsável pela página de acionamento ilustrada na Figura 4.15. A função relay_wifi() monitora se há clientes conectados ao objeto WiFiServer e processa manualmente as requisições recebidas. Quando a linha de requisição contém GET /lamp1/on, /lamp1/off e comandos equivalentes para as demais saídas, o firmware altera o estado interno da lâmpada correspondente e atualiza o LED de modo automático quando aplicável.

O conteúdo HTML é gerado dinamicamente e enviado ao navegador linha a linha via httpclient.println(). A página utiliza uma folha de estilo CSS simples, com a classe .par1 definindo o formato dos botões e as classes .btOn e .btOff diferenciando visualmente os estados ‘‘ligar’’ e ‘‘desligar’’. O cabeçalho exibe o título ‘‘Servidor de acionamento’’ e, abaixo, são mostrados o estado textual de cada lâmpada e um botão correspondente. Quando a saída está desligada, o botão aparece em verde com o texto ‘‘*lamp X turn on*’’; quando está ligada, o botão passa para cinza com o texto ‘‘*lamp X turn off*’’, permitindo ao usuário alternar o estado com um único clique. Um bloco adicional apresenta o estado do modo automático e um botão dedicado para sua ativação ou desativação.

No laço principal (`loop()`), o firmware gerencia, a cada iteração, a seleção entre controle manual e automático, além dos serviços de comunicação. Primeiro, lê-se o botão dedicado ao modo automático (`BOTAO_AUTO`) com uma janela mínima de antirruído (`debounce`) baseada em `millis()`; quando pressionado, a variável `modoAutomatico` é alternada e o LED de indicação (`LED_MODO_AUTO`) é atualizado. Em seguida, se o modo automático estiver

desativado, o código verifica os três botões locais (`BOTAO1-3`). Um botão pressionado força o respectivo estado da lâmpada (`estadoLampX`) para `true`; quando solto, caso o espelho lógico proveniente das interfaces remotas (`lampX_status`) esteja em `false`, a saída é mantida desligada. Após computar esses estados, os GPIOs dos relés (`LAMP1-3`) são atualizados com `digitalWrite()`, garantindo que o estado interno reflita o hardware.

Por fim, o laço assegura a robustez da conectividade. Se o Wi-Fi cair, uma tentativa de reconexão é feita a cada 5 s e o laço retorna cedo, evitando processamento inútil de MQTT/HTTP sem rede. Quando a interface está ativa, o cliente MQTT é mantido com `mqtt.loop()` e, se necessário, é restabelecido por `mqttReconnect()`, que refaz as assinaturas e republica os estados. Em todas as iterações, o servidor HTTP embarcado é atendido pela chamada `relay_wifi()`, que processa requisições `GET` para ligar/desligar cada lâmpada e o modo automático, também invocando `publishState()` após qualquer mudança. Os códigos completos estão no Apêndice A.

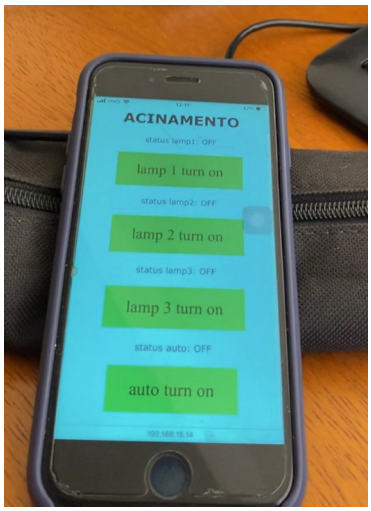


Figura 4.15: servidor de acionamento ESP32 01.

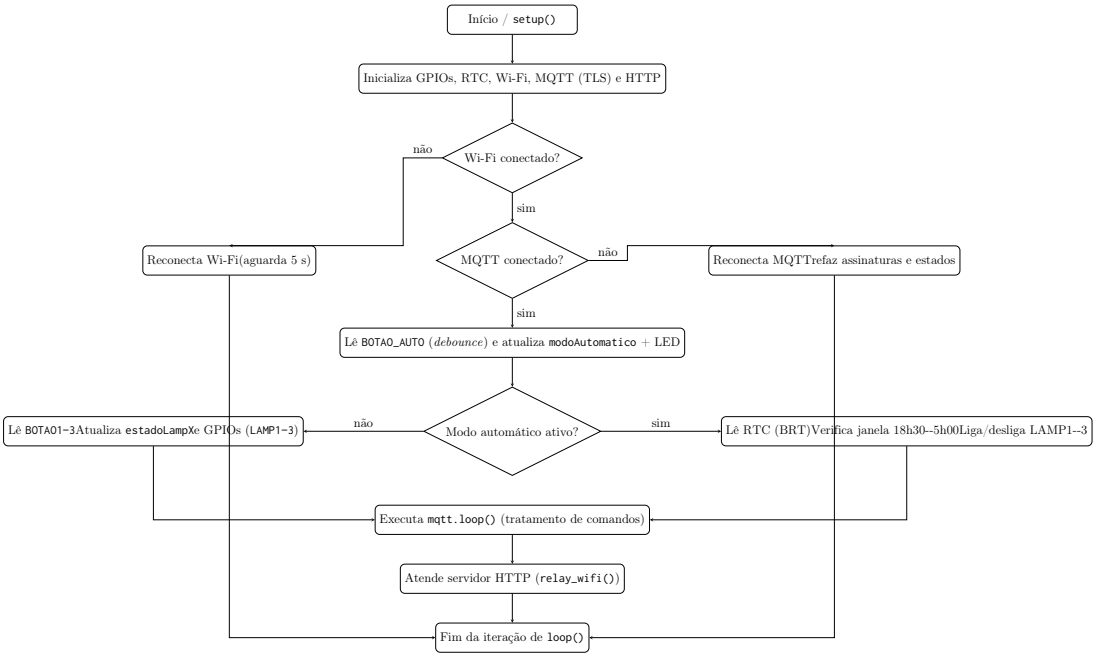


Figura 4.16: Fluxograma simplificado da lógica do nó urbano (ESP32 01).

4.4.2 Firmware do nó rural (ESP32 02)

4.4.3 Módulo de comunicação Wi-Fi do nó rural

A conectividade sem fio do nó rural foi encapsulada em um componente específico denominado `wifi`, responsável por inicializar a interface de rede do ESP32 em modo estação, gerenciar o processo de conexão ao roteador do sítio e disponibilizar para o restante do firmware uma sinalização simples de estado da rede. O código-fonte completo desse componente encontra-se no Apêndice B, identificado como "Módulo de Wi-Fi (componente `wifi`)".

O arquivo `CMakeLists.txt` do componente declara que o módulo `wifi` é composto pelo arquivo `connect.c`, expõe seu cabeçalho na própria pasta (`INCLUDE_DIRS "."`) e depende diretamente de bibliotecas do ESP-IDF relacionadas à pilha de rede, sistema e registro de eventos, tais como `esp_wifi`, `esp_netif`, `esp_event`, `nvs_flash` e `freertos`. Essa organização permite que o mesmo componente de Wi-Fi seja reutilizado em diferentes nós da automação.

A interface pública do módulo é definida em `connect.h`. Esse cabeçalho declara três funções principais: `wifi_init()`, `wifi_connect_sta()` e `wifi_disconnect()`. A primeira prepara a infraestrutura de rede e registra os tratadores de evento; a segunda estabelece a conexão em modo estação com um ponto de acesso Wi-Fi, recebendo como parâmetros o *SSID*, a senha e um tempo máximo de espera; a terceira encerra de forma ordenada a conexão.

A implementação dessas funções encontra-se em `connect.c`. Logo no início, é declarada

a variável global `volatile bool wifiOnline`, utilizada por outras tarefas do sistema para saber se o nó está efetivamente conectado e com endereço IP válido. Esse arquivo também define um *event group* (`wifi_events`) e dois bits de sincronização: `CONNECTED_GOT_IP` e `DISCONNECTED`. Esses bits são acionados pelos tratadores de evento conforme o estado da conexão evolui.

A função `wifi_event_handler()` atua como ponto central de tratamento dos eventos gerados pelo Wi-Fi e pela pilha de IP. Quando o módulo entra em *WIFI_EVENT_STA_START*, a função inicia a tentativa de conexão ao roteador. Em *WIFI_EVENT_STA_CONNECTED*, registra-se o sucesso de associação física ao ponto de acesso, ainda sem IP. Em caso de *WIFI_EVENT_STA_DISCONNECTED*, o código decodifica o motivo da desconexão, registra mensagens de log e sinaliza o bit de desconexão no *event group*. Já o evento *IP_EVENT_STA_GOT_IP* indica que o nó recebeu um endereço IP do roteador; nesse momento, a variável `wifiOnline` é ajustada para `true` e o bit `CONNECTED_GOT_IP` é acionado, liberando as tarefas que dependem de conectividade para prosseguir.

A função `wifi_init()` executa a configuração inicial da pilha de rede. Nela são chamados, em sequência, os procedimentos de inicialização do `esp_netif`, a criação do *loop* de eventos padrão, a inicialização do driver de Wi-Fi e o registro da própria `wifi_event_handler()` tanto para eventos de Wi-Fi quanto para o evento de obtenção de IP. Também é definido que as configurações de Wi-Fi serão mantidas em RAM e criado o *event group* utilizado para sincronização com a função de conexão.

A função `wifi_connect_sta()` é responsável por, de fato, conectar o ESP32 ao roteador do sítio. Ela cria a interface padrão em modo estação, preenche a estrutura `wifi_config_t` com o *SSID* e a senha informados, configura o dispositivo em modo `WIFI_MODE_STA`, aplica a configuração (`esp_wifi_set_config()`) e inicia o Wi-Fi com `esp_wifi_start()`. Em seguida, a função bloqueia em `xEventGroupWaitBits()`, aguardando até que um dos bits `CONNECTED_GOT_IP` ou `DISCONNECTED` seja acionado, dentro de um tempo limite estabelecido pelo parâmetro *timeout*. Caso o IP seja obtido a tempo, a função devolve `ESP_OK`; em caso de falha ou estouro de tempo, retorna `ESP_FAIL`.

A função `wifi_disconnect()` oferece um encerramento explícito da conexão, chamando `esp_wifi_disconnect()` e `esp_wifi_stop()`. Embora o nó rural permaneça normalmente conectado para suportar a recepção contínua de comandos MQTT e o envio periódico de dados, essa função permite desligar a interface sem fio em cenários de teste, depuração ou economia de energia.

4.4.4 Arquitetura principal e módulo de temporização do nó rural

O firmware do nó rural (ESP32 02) foi organizado como um componente do ESP-IDF, de forma a agrupar em um único módulo o arquivo principal da aplicação (`main.c`), a lógica de comunicação MQTT, o agendador de tarefas e o driver do RTC externo DS3231.

O arquivo `CMakeLists.txt` associado a esse componente declara, por meio da instrução `idf_component_register`, que o conjunto de fontes é composto pelos arquivos `main.c`, `MQTT.c`, `ds3231.c` e `scheduler.c`. Esse mesmo arquivo define que os cabeçalhos do componente se encontram na própria pasta (`INCLUDE_DIRS "."`) e especifica as dependências internas e externas, como os módulos de Wi-Fi, JSON, MQTT, drivers de hardware e pilha TCP/IP. Com isso, o ESP-IDF é capaz de compilar o nó rural como um bloco coeso, reaproveitável e com suas bibliotecas de suporte claramente declaradas. O código-fonte completo desse componente encontra-se reunido no Apêndice B.

Dentro dessa arquitetura, o par de arquivos `ds3231.h` e `ds3231.c` implementa o módulo responsável pela contagem de tempo e pela medição de temperatura com o circuito integrado DS3231, conectado ao ESP32 por meio do barramento I^2C. O cabeçalho `ds3231.h` define o endereço I^2C do dispositivo (`0x68`) e os registradores utilizados para acesso à informação de tempo e de temperatura, também declara as funções públicas do módulo: inicialização do RTC, leitura e escrita da data e hora, e leitura da temperatura.

A implementação em `ds3231.c` encapsula a lógica de acesso ao hardware. Inicialmente, são declaradas funções auxiliares para conversão entre o formato BCD (*Binary Coded Decimal*), utilizado pelo DS3231 para armazenar segundos, minutos, horas e demais campos de data, e valores inteiros comuns. Em seguida, são definidas duas rotinas de leitura e escrita genéricas sobre o barramento I^2C, responsáveis por montar os comandos com início, endereço do escravo, registrador alvo e finalização da transação.

A função `ds3231_init()` recebe como parâmetros a porta I^2C a ser utilizada, os pinos SDA e SCL e a frequência de operação do barramento. Ela configura o ESP32 como mestre I^2C, associa os pinos físicos, instala o driver de comunicação e realiza uma leitura de teste no registrador de segundos. Essa leitura inicial serve como verificação simples de que o DS3231 está presente e respondendo corretamente no barramento, permitindo que a aplicação registre um erro caso o módulo de relógio em tempo real esteja desconectado ou com defeito.

Para leitura da hora e da data, a função `ds3231_get_time()` lê em sequência os sete registradores que armazenam segundos, minutos, horas, dia da semana, dia, mês e ano. Os valores em BCD são convertidos para a estrutura `struct tm`, utilizada pelas funções padrão de tempo da linguagem C, incluindo o tratamento do formato de 12 ou 24 horas e os ajustes de índice de mês e ano. Essa função é usada, posteriormente, pelo agendador de tarefas do nó rural para calcular o horário atual em minutos desde a meia-noite e decidir, por exemplo, quando ligar ou desligar automaticamente as lâmpadas e a irrigação.

4.4.5 Lógica principal e agendamento do nó rural

A organização da lógica principal do nó rural foi estruturada a partir de um pequeno conjunto de arquivos que se complementam: `general.h`, `main.c`, `scheduler.c` e

scheduler.h. Esses arquivos reúnem a configuração física do nó (mapeamento de GPIOs, parâmetros de sensores e tópicos MQTT), o fluxo de inicialização do firmware, a criação das tarefas do sistema e a lógica de agendamento automático de lâmpadas e irrigação. O código-fonte completo dessa estrutura pode ser consultado no Apêndice B.

O arquivo general.h concentra as definições que descrevem o mundo físico e lógico do nó rural. Nele são declarados o nome da rede Wi-Fi e a respectiva senha utilizados para conectar o ESP32 ao roteador do sítio, os GPIOs associados às três lâmpadas e à saída de irrigação, e também de parâmetros de operação do conversor analógico-digital utilizados para leitura do sensor de umidade do solo. O arquivo também define dois pontos de calibração para o solo seco e úmido, que são empregados em uma interpolação linear para converter o valor cru do ADC em porcentagem de umidade. Ainda em general.h, é estabelecido um namespace coerente de tópicos MQTT para o nó rural, separando claramente comandos (tcc/esp02/rural/l1, /irrig, /auto, /irr_auto) de tópicos de estado (tcc/esp02/rural/state/), além da declaração da variável wifiOnline, utilizada por outras partes do firmware para saber se há conectividade disponível.

A função principal de inicialização encontra-se em main.c. Nesse arquivo, a rotina setup() executa a sequência de preparação do nó: inicializa a NVS, ajusta o nível de detalhamento dos logs, chama wifi_init() e, em seguida, wifi_connect_sta() para estabelecer a conexão à rede local. Após garantir a conectividade básica, o firmware inicia o processo de sincronização de horário via SNTP, configura o fuso horário do sistema para UTC-3 e invoca ds3231_init() para habilitar o uso do RTC externo. Com isso, mesmo que a conexão com a internet seja perdida posteriormente, o nó passa a contar com uma base de tempo local estável para acionar as agendas automáticas.

O mesmo arquivo main.c contém a função io_init(), responsável por configurar os GPIOs associados às lâmpadas e à irrigação como saídas digitais, assegurando que todas as cargas iniciem desligadas no momento do boot. Na função app_main(), após a chamada a setup() e io_init(), são criadas as tarefas FreeRTOS que compõem o comportamento do nó rural: uma tarefa dedicada ao controle MQTT (MQTTControlTask), o agendador automático (SchedulerTask), a tarefa de envio de dados (MQTTSenderTask) e duas tarefas de aquisição das grandezas monitoradas (taskTemperatureQueue e taskHumidityQueue). Essa divisão em tarefas especializadas permite que o nó execute, em paralelo, a recepção de comandos, o acionamento de cargas, a leitura de sensores e o envio de telemetria, sem que uma função interfira diretamente na responsividade das demais.

Já a lógica de agendamento automático está concentrada nos arquivos scheduler.c e scheduler.h. O cabeçalho scheduler.h expõe apenas o protótipo da função SchedulerTask(), enquanto a implementação em scheduler.c utiliza o horário fornecido pelo RTC DS3231 para calcular os minutos decorridos desde a meia-noite e, a partir disso, decidir se o sistema deveria estar em um período de lâmpadas ligadas ou de irrigação ativa. Funções auxiliares como minutes_since_midnight() e in_window() encapsulam esses cálculos, incluindo o

caso em que uma janela de funcionamento cruza a meia-noite (por exemplo, das 18h30 até as 5h do dia seguinte).

A tarefa `SchedulerTask()` é executada com período de um segundo e avalia continuamente duas janelas de funcionamento: a irrigação, configurada entre 9h00 e 9h10, e a iluminação, configurada das 18h30 até as 5h00 da manhã seguinte. Quando o modo automático de irrigação está habilitado, a tarefa compara o horário atual com a janela definida e liga ou desliga a saída de irrigação por meio da função `set_irrig()`, que também atualiza o tópico MQTT de estado correspondente. De forma análoga, quando o modo automático de iluminação está ativo, a tarefa comanda o conjunto de três lâmpadas simultaneamente, utilizando `set_all_lamps()` para refletir o estado físico das cargas e publicar o estado atualizado nos tópicos de *state*.

4.4.6 Módulo de comunicação MQTT e formatação de dados

A comunicação com o broker MQTT, bem como o empacotamento e o envio dos dados de sensores, foi organizada no conjunto formado por `MQTT.h`, `mqtt_cert.h` e `MQTT.c`. Esses arquivos definem tanto a interface de alto nível usada pelas demais partes do firmware (tarefas de leitura de sensores, agendador e função principal) quanto os detalhes da conexão segura via TLS, da assinatura de tópicos de comando e da publicação periódica dos valores de temperatura e umidade do nó rural. O código-fonte completo desses arquivos está reunido no Apêndice B.

O cabeçalho `MQTT.h` descreve a interface pública do módulo de comunicação. Nele são definidas duas estruturas de dados de uso recorrente: `MqttQueueFloat_t`, utilizada como fila de envio para o cliente MQTT de telemetria, e `SysDataFloat_t`, empregada como buffer circular interno para armazenar séries temporais de leituras de sensores. Cada estrutura guarda um rótulo (`tag`), o identificador lógico (`local`), o valor de ponto flutuante medido e um carimbo de tempo em segundos. O arquivo também declara constantes que determinam o tamanho máximo dessas filas e reúne as variáveis globais que representam o estado das cargas e dos modos de operação do nó (`lamp1_state`, `auto_mode`, `irrig_state`, `auto_irrig`), além do identificador da tarefa responsável pelo envio MQTT. Por fim, `MQTT.h` expõe os protótipos das funções de inicialização de tempo, armazenamento em fila, tarefas de leitura de sensores e tarefas de comunicação, permitindo que o restante do firmware utilize o módulo sem conhecer os detalhes de implementação.

O arquivo `mqtt_cert.h` armazena o certificado da autoridade certificadora utilizado para verificar a identidade do broker MQTT na conexão segura. O certificado é incluído no firmware como uma cadeia em formato PEM e é associado aos campos de configuração do cliente MQTT responsáveis pela verificação do servidor remoto. Ao utilizar a URI `mqtts://` e preencher o parâmetro `broker.verification.certificate` com esse certificado, o nó rural passa a estabelecer uma sessão TLS em que o broker é autenticado antes da troca de

dados. Essa abordagem protege os comandos e as leituras de telemetria contra interceptação e alteração durante o tráfego na rede, reforçando a segurança da automação rural.

A implementação central do módulo encontra-se em `MQTT.c`. Esse arquivo começa com funções auxiliares para sincronização de horário via SNTP e conversão de carimbos de tempo para o formato ISO 8601, que serão utilizados na construção dos JSON enviados ao backend. Também são definidos um manipulador genérico de eventos MQTT e uma função `payload_is_on()`, que interpreta mensagens de comando simples (`"on"`, `"1"`, `"true"`) e as converte em valores booleanos. A tarefa `MQTTControlTask()` cria o cliente MQTT dedicado ao controle, configurado com o endereço do broker, credenciais de acesso e certificado TLS. Uma vez conectado, esse cliente assina os tópicos de comando das lâmpadas, da irrigação e dos modos automáticos. Sempre que uma mensagem é recebida nesses tópicos, a função `mqtt_control_event()` identifica qual carga está sendo comandada, atualiza as variáveis de estado correspondentes, aciona os GPIOs físicos por meio de funções internas e publica, nos tópicos de *state*, o novo estado lógico com a flag de retenção ativada.

O mesmo arquivo também implementa o caminho da telemetria. As tarefas `taskHumidityQueue()` e `taskTemperatureQueue()`, descritas anteriormente, preenchem os buffers circulares com as leituras mais recentes de umidade do solo e temperatura. A tarefa `MQTTSenderTask()` observa essas filas e, sempre que há dados novos e conectividade Wi-Fi disponível, copia as amostras mais recentes para o vetor `generalDataQueue`, marcando que há dados a enviar. Em seguida, essa tarefa notifica a função `MQTTSender()`, que cria um segundo cliente MQTT, dedicado apenas à publicação, reutilizando o mesmo certificado de confiança. Esse cliente constrói dois tipos de payload: um formato textual compacto, no padrão `{TAG:VAL:TIMESTAMP,...}`, publicado no tópico `tcc/esp02/rural/raw` para depuração em ferramentas como o MQTTBox, e dois objetos JSON separados, com valor numérico e *timestamp* em ISO 8601, enviados aos tópicos `tcc/esp02/rural/temp` e `tcc/esp02/rural/hum`, compatíveis com os fluxos de processamento no backend.

Além do envio de dados, `MQTT.c` também contém rotinas para leitura pontual de valores assinados e para gerenciamento das filas. A função `storeFloatQueue()` implementa uma fila circular de tamanho fixo, descartando as medições mais antigas quando o buffer está cheio e registrando cada nova amostra com um carimbo de tempo obtido a partir do relógio DS3231. A função `convertData()` realiza o parse de mensagens recebidas em formato textual, extraindo o valor numérico mesmo quando o payload possui prefixos ou outros campos auxiliares. Por fim, após a confirmação de publicação (*ACK* do broker), o módulo limpa as estruturas internas, zera os índices e ajusta as variáveis de controle para evitar acumulação de dados redundantes na memória.

Em conjunto, `MQTT.h`, `mqtt_cert.h` e `MQTT.c` estruturam o subsistema de comunicação do nó rural de forma a combinar controle e envio periódico de telemetria, sempre sobre uma conexão cifrada e autenticada.

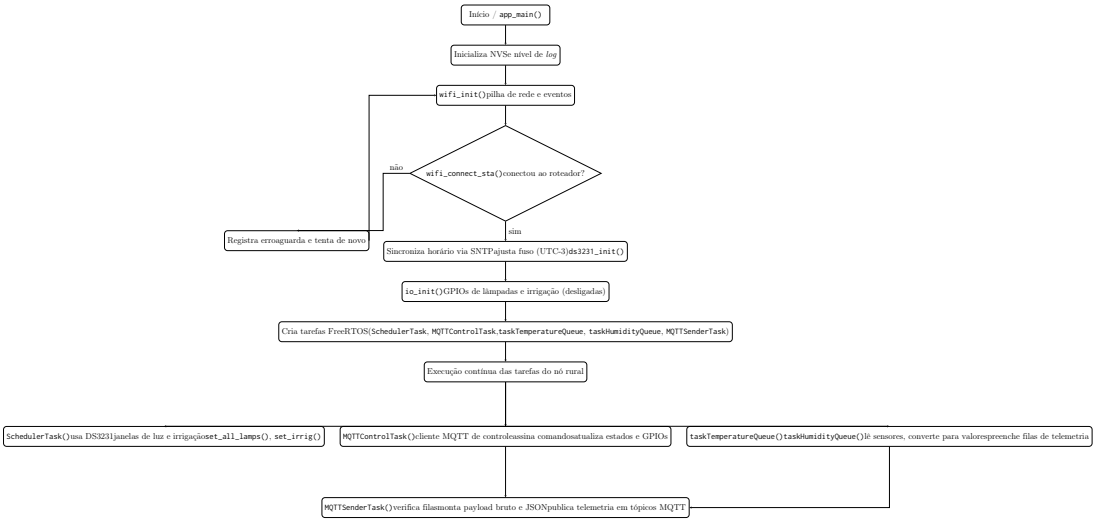


Figura 4.17: Fluxograma simplificado da lógica de inicialização e tarefas do nó rural (ESP32 02).

4.5 Aplicativo Mobile(IoT MQTT Panel)

O Aplicativo utilizado neste trabalho foi o *IoT MQTT Panel*. Este aplicativo transforma o smartphone em cliente MQTT para supervisão e comando dos nós *Casa* (urbano, ESP32-01) e *Sítio* (rural, ESP32-02). As Figuras 4.18 a 4.23 mostram a criação da conexão segura com o broker e os painéis utilizados no projeto.

Lista de conexões: Na tela *Connections* (Figura 4.18) o usuário visualiza, cria e gerencia conexões MQTT. O item `IOT` indica a conexão já cadastrada, os botões flutuantes permitem importar/exportar configurações e criar novas conexões.

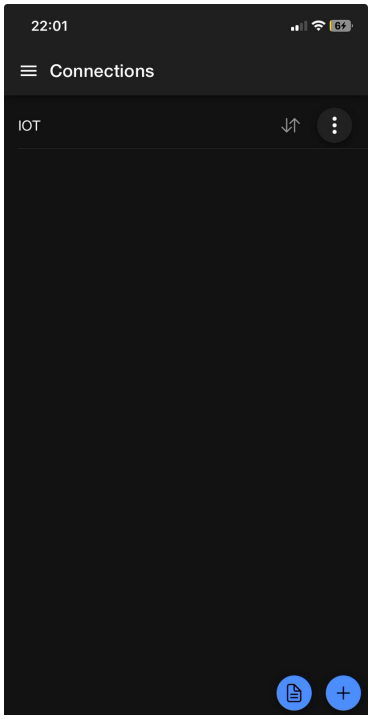


Figura 4.18: Tela *Connections* do IoT MQTT Panel com a conexão `IOT`.

Edição da conexão segura: Na tela *Edit Connection* (Figura 4.19) são definidos: *Connection name* (`IOT`), *Client ID* (`iphone-gabriel`), *Broker address* (IP público do servidor Mosquitto), *Port* (`8883`) e *Network protocol* (`TCP-SSL`). A seção *Manage SSL configuration* armazena certificado/ chaves TLS. Em *Add Dashboard* vinculam-se os painéis *Casa* e *Sítio*.

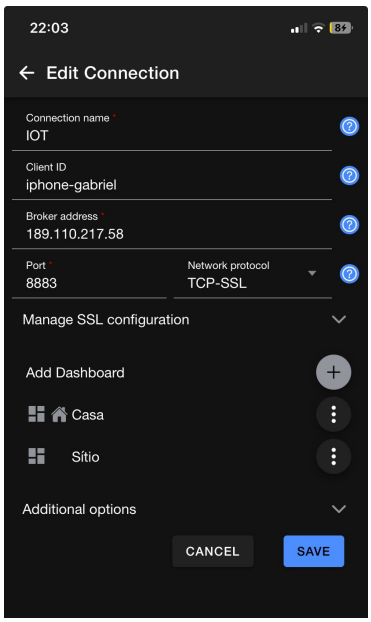


Figura 4.19: Edição da conexão MQTT segura e associação dos dashboards.

Painel do nó urbano (*Casa*): A Figura 4.20 apresenta o painel *Casa*, com quatro widgets: *lâmpada 1*, *lâmpada 2*, *lâmpada 3* e *modo automático*. Esses controles publicam comandos MQTT para as três cargas de iluminação do nó urbano e para a comutação entre operação manual e a rotina automática (18:30–05:00, conforme firmware).

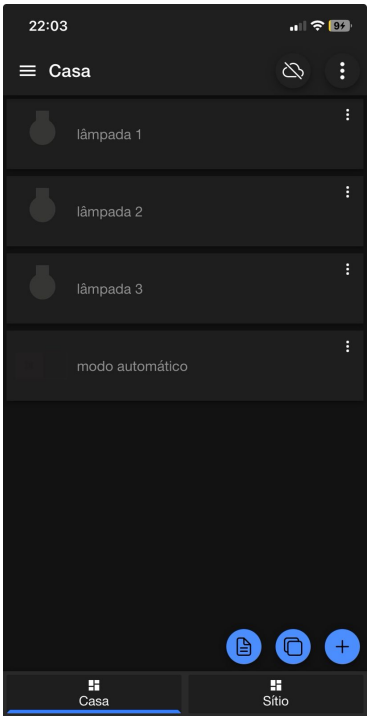


Figura 4.20: Painel *Casa* com widgets de comando.

Painel do nó rural (*Sítio*): A Figura 4.21 mostra o painel *Sítio* com seis widgets: *lâmpada 1*, *lâmpada 2*, *lâmpada 3*, *modo automático*, *irrigação* (acionamento manual da válvula) e *irrigação automática* (rotina diária 09:00–09:10). Na parte inferior, o gráfico *Temperatura* exibe dados publicados pelo sensor do nó rural.

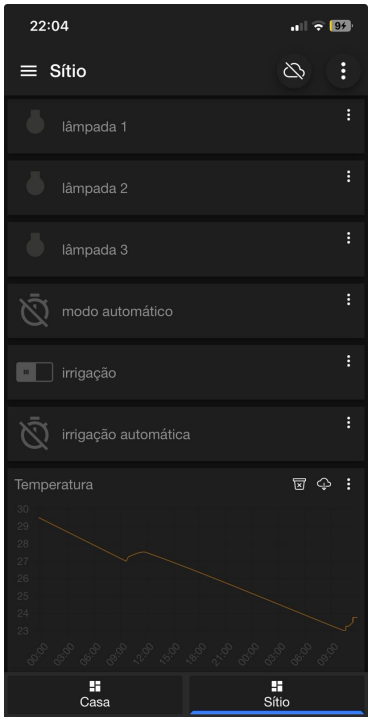


Figura 4.21: Painel *Sítio* com controles de iluminação e irrigação e gráfico de temperatura.

Gráficos de temperatura e umidade ilustrativos no Aplicativo: A Figura 4.22 foca o monitoramento feito pelos dois gráficos de linha. Nos gráficos é possível ver a *Temperatura* (°C) e a *Umidade* (%), permitindo acompanhar a evolução temporal. Os dados apresentados nos gráficos são dados de testes.

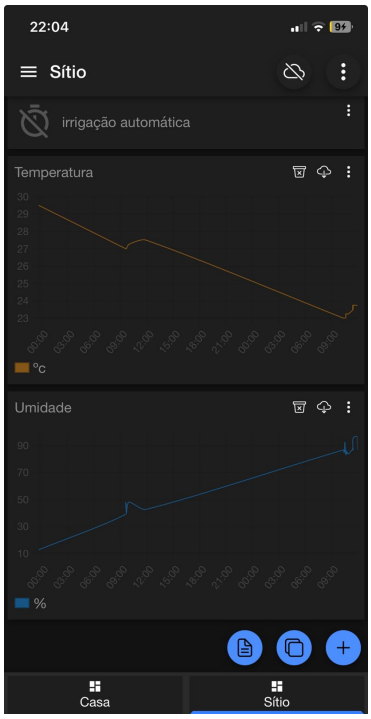


Figura 4.22: Monitoramento no painel *Sítio*: gráficos de temperatura e umidade.

Operação ativa com broker conectado. Por fim, a Figura 4.23 registra o painel em operação, com conexão estabelecida ao broker (ícone de nuvem em destaque). Observam-se os mesmos widgets de comando e o gráfico *Temperatura* com leituras em tempo real, evidenciando a troca contínua de mensagens MQTT entre smartphone e ESP32.

Figura 4.23: Painel durante operação: conexão ativa com o broker e gráfico de temperatura.

4.6 Envio de Dados para a Nuvem

4.6.1 Configuração e Explicação do MQTTBox

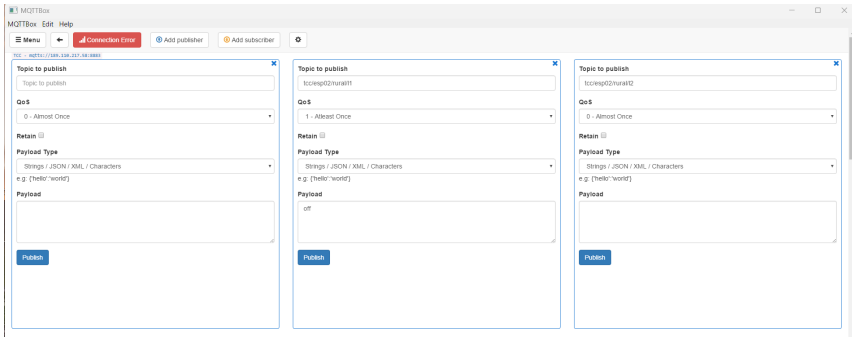
Esta seção descreve a configuração do cliente *MQTTBox* para envio de dados ao broker MQTT hospedado em máquina virtual. Na Figura 4.24a, observa-se que a conexão ainda não foi estabelecida, pois a máquina virtual (VM) no Oracle VirtualBox permanece desligada. Consequentemente, o endereço IP público e a respectiva porta do broker não respondem até que a VM seja inicializada e as regras de rede (NAT/port forwarding) estejam ativas.

A Figura 4.24b apresenta os parâmetros definidos no cliente, que serão efetivos assim que a VM estiver operacional e o broker acessível.

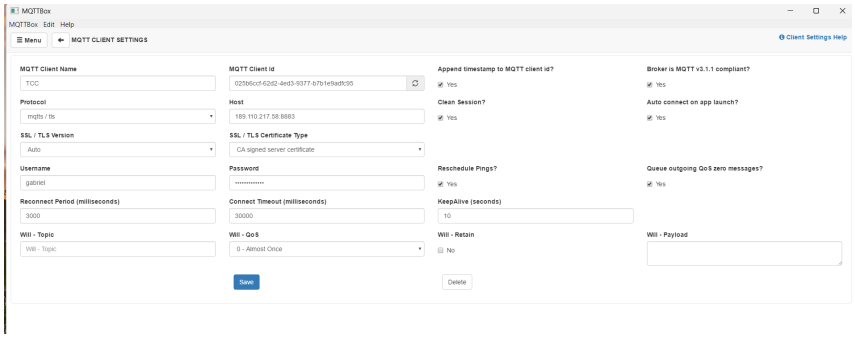
- **MQTT Client Name:** identificador único do cliente no broker.
- **Protocol (MQTTS/TLS):** habilita sessão segura com autenticação e criptografia.
- **SSL/TLS Version:** versão mínima/negociada do TLS utilizada na sessão.
- **Username e Password:** credenciais de autenticação do cliente no broker.
- **Certificate Type:** tipo de certificado para validar o servidor (por exemplo, CA do broker).
- **Host:** IP público ou FQDN do broker; indisponível enquanto a VM estiver desligada.

Observação1: O endereço IP utilizado na configuração do servidor corresponde ao *IP público* da conexão com a internet, distinto do IP interno atribuído à máquina virtual ou ao roteador local. Esse valor pode ser obtido por meio de serviços de consulta, como o site `whatismyip.com`. O uso do IP público é essencial, pois o ESP32 poderá estar conectado em outra rede Wi-Fi e, portanto, precisará acessar o servidor pela internet a partir desse endereço.

Observação2: O certificado da autoridade certificadora (CA) utilizado pelo broker MQTT foi gerado diretamente na máquina virtual Linux por meio do utilitário `openssl`. Em um diretório dedicado (por exemplo, `~/certs`), foram executados os comandos `mkdir -p /certs && cd /certs`, `openssl genrsa -out ca.key 2048` e, em seguida, `openssl req -x509 -new -nodes -key ca.key -sha256 -days 365 -out ca.crt`. O arquivo `ca.crt` resultante corresponde ao certificado público da CA e deve ser copiado para o código do ESP32, permitindo que o cliente MQTT valide a identidade do servidor durante o estabelecimento da conexão TLS.



(a) Cliente *MQTTBox* sem conexão, com a VM ainda desligada.



(b) Parâmetros de conexão: nome do cliente, protocolo TLS, versão, credenciais, certificado e host.

4.6.2 Configuração da Máquina Virtual

As Figuras 4.25 e 4.26 apresentam os detalhes da configuração da máquina virtual CentOS10 no Oracle VirtualBox. A VM é nomeada como *"CentOS10"* e executa um sistema operacional Linux, utilizando a opção *Red Hat 64-bit* como sistema convidado. A memória principal foi configurada para 2048 MB, com dois processadores atribuídos. A ordem de boot inclui disquete, unidade óptica e disco rígido, e a aceleração de hardware permanece habilitada.

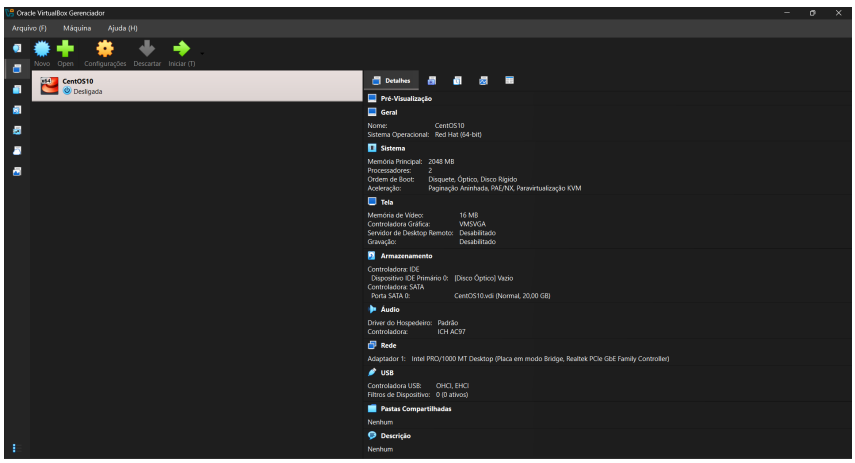


Figura 4.25: Configurações gerais da máquina virtual CentOS10 no VirtualBox.

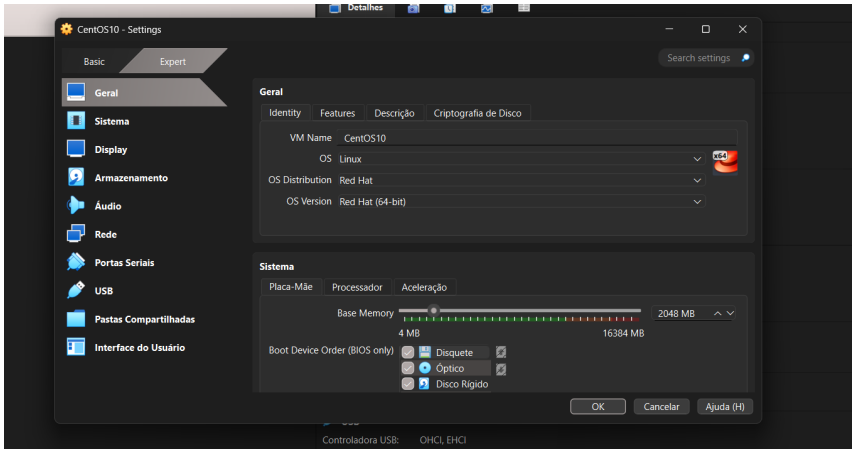


Figura 4.26: Detalhes adicionais da configuração de sistema da VM, incluindo memória, placa-mãe e dispositivos de boot.

4.6.3 Configuração e Ativação da Máquina Virtual

Após o primeiro acesso à máquina virtual com o usuário `root` e a senha definida na instalação, foi realizada a configuração inicial do ambiente na distribuição Linux, incluindo a instalação do Docker por meio do gerenciador de pacotes. Esse mesmo usuário é utilizado para criar e ajustar os contêineres necessários ao projeto (broker MQTT, banco de dados e Scada-LTS), que serão apresentados nas figuras mostradas na sequência. Ao iniciar a máquina virtual, como ilustrado na Figura 4.27, o sistema solicitará as credenciais de login. Utilize o usuário `root` e a senha correspondente para acessar o sistema.

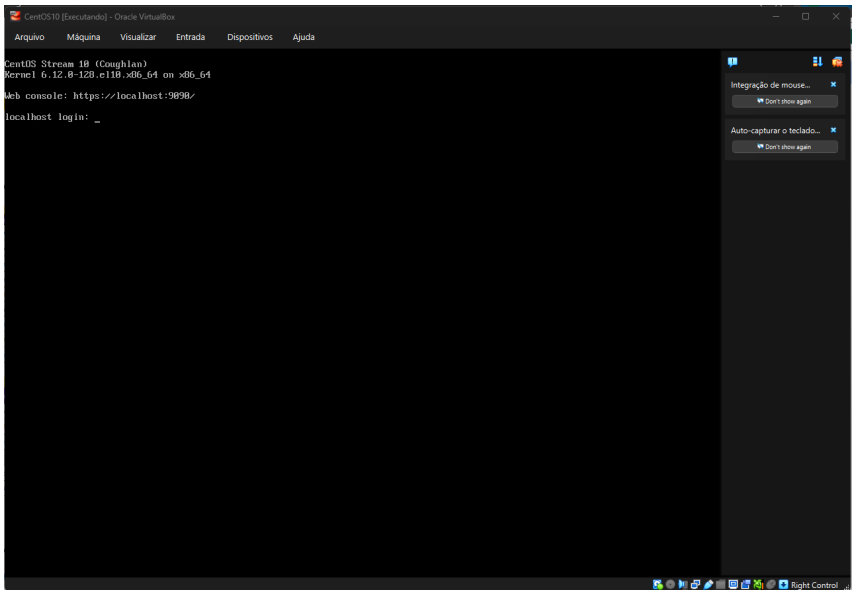


Figura 4.27: Tela de login da máquina virtual CentOS10.

Em seguida, execute o comando `nmtui` para acessar a interface de configuração de rede. Escolha a opção "Edit a connection", conforme demonstrado na Figura 4.28.

Após selecionar "Edit a connection", a tela de configuração de rede será exibida

Figura 4.28: Interface do `nmtui` para edição de conexões.

(Figura 4.29). Nessa tela, ajuste o endereço IP, o gateway e o servidor DNS, marcando a opção "Available to all users"antes de confirmar.

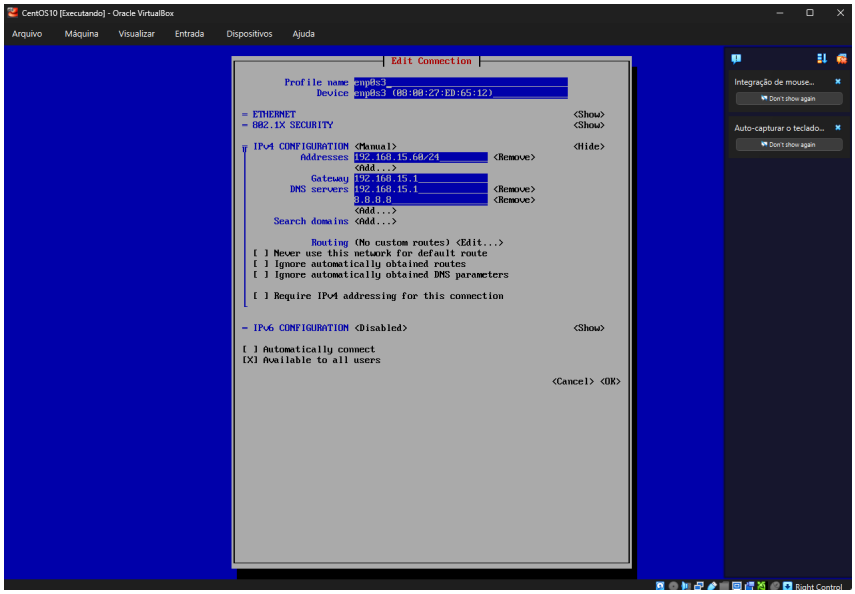


Figura 4.29: Ajuste das configurações de rede na VM.

Dessa forma, a máquina virtual estará configurada e pronta para uso.

Com as configurações de rede ajustadas, retornamos ao menu principal do `nmtui`, como ilustrado na Figura 4.28. Nesse menu, selecione a opção "Activate a connection"para ativar a conexão recém-configurada. A Figura 4.30 demonstra o processo de ativação.

Ao prosseguir e confirmar a ativação, conforme a Figura 4.30, a máquina virtual estará finalmente conectada e pronta para ser utilizada.

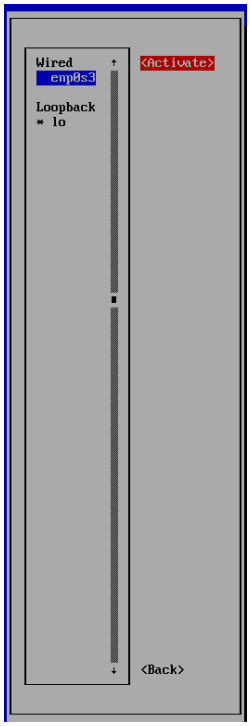


Figura 4.30: Confirmação da ativação da conexão na máquina virtual.

Na etapa final, conforme ilustrado na Figura 4.31, o cliente MQTTBox aparece devidamente conectado ao broker, indicando que a máquina virtual foi iniciada com sucesso e as configurações foram aplicadas.

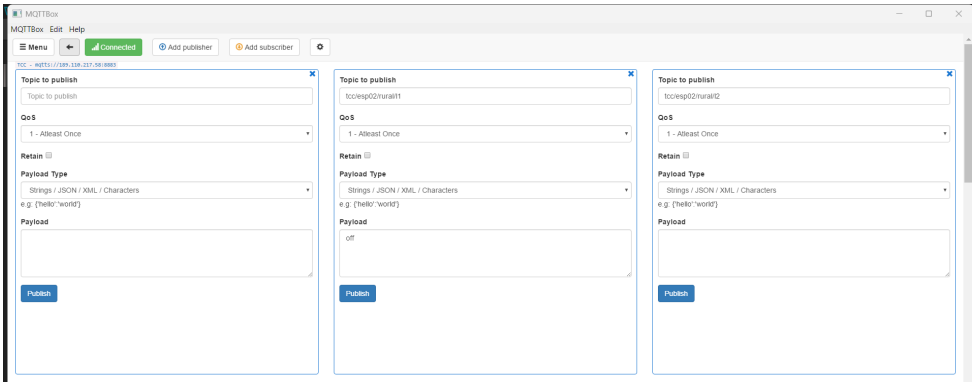


Figura 4.31: Estado de conexão estabelecida no MQTTBox.

As figuras subsequentes mostram a configuração dos *publishers* e *subscribers*, respectivamente, para os tópicos de controle e monitoramento definidos no projeto.

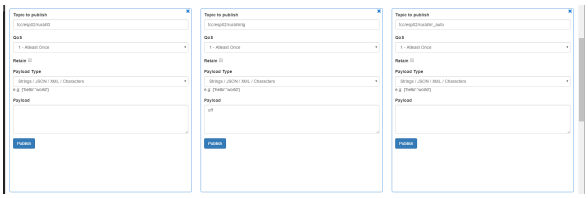


Figura 4.32: Configuração de publishers e subscribers do projeto parte 1.

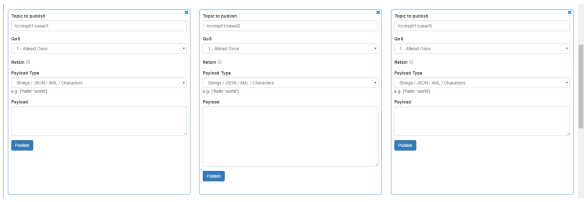


Figura 4.33: Configuração de publishers e subscribers do projeto parte 2.

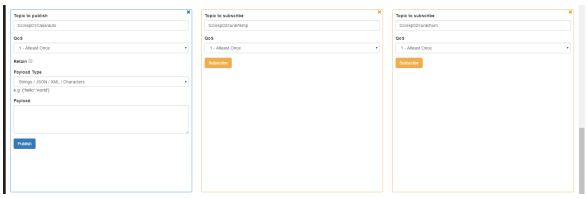


Figura 4.34: Configuração de publishers e subscribers do projeto parte 3.

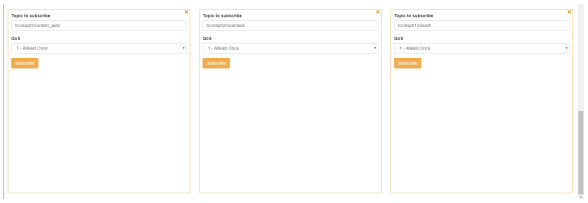


Figura 4.35: Configuração de publishers e subscribers do projeto parte 4.

Essas imagens ilustram os tópicos configurados para publicação e assinatura de mensagens entre os dispositivos ESP32.

Caso o *MQTTBox* não estabeleça conexão, deve-se acessar a máquina virtual e executar o **Comando 01**, responsável por inicializar o serviço do broker e verificar seu estado. Esse procedimento está ilustrado na Figura 4.36.

Listing 4.1: Comando 01 para reativação do serviço do broker e verificação do contêiner.

```
1 docker start scadalts-mqtt-1
2 docker ps
```

A primeira instrução inicia o contêiner `scadalts-mqtt-1`; a segunda lista os contêineres em execução, permitindo confirmar o estado do serviço após a tentativa de reconexão (vide Figura 4.36).

```
[root@localhost ~]# docker start scadalts-mqtt-1
docker ps
scadalts-mqtt-1
CONTAINER ID   IMAGE                       COMMAND                  CREATED
 STATUS                    PORTS
                                  NAMES
e2212f13dcd7   eclipse-mosquitto:2         "/docker-entrypoint.…"   13 days ago
 Up 1 second               0.0.0.0:1883->1883/tcp, [::]:1883->1883/tcp, 0.0.0.0:8
883->8883/tcp, [::]:8883->8883/tcp   scadalts-mqtt-1
f668fedf4062   scadalts/scadalts:master    "catalina.sh run"        2 weeks ago
 Up 2 minutes              0.0.0.0:8082->8080/tcp, [::]:8082->8080/tcp
                                  scadalts-scadalts-1
ab5a865b14de   mysql:5.7                   "docker-entrypoint.s…"   2 weeks ago
 Up 2 minutes (healthy)    3306/tcp, 33060/tcp
                                  scadalts-database-1
d839bab60d95   nodered/node-red:latest     "./entrypoint.sh"        4 weeks ago
 Up 2 minutes (healthy)    0.0.0.0:1880->1880/tcp, [::]:1880->1880/tcp
                                  mynodered
```

Figura 4.36: Tela de referência do procedimento de contingência para reconexão do *MQTTBox*.

4.6.4 Fluxo de Dados e Integração com o Node-RED

O *Node-RED* foi utilizado como plataforma intermediária para o tratamento e roteamento das mensagens publicadas pelo *broker* MQTT na máquina virtual. Essa ferramenta permitiu integrar o sistema de automação desenvolvido com os bancos de dados do *SCADA-LTS* e com as interfaces de visualização em tempo real.

A Figura 4.37 apresenta o fluxo criado no ambiente do *Node-RED*, responsável por receber os dados enviados pelos módulos ESP32, realizar o processamento das informações e repassá-las ao banco de dados *MySQL*. Cada nó do fluxo foi configurado de acordo com a função desempenhada:

- **MQTT In:** subscreve aos tópicos definidos para leitura de temperatura e umidade, recebendo as mensagens publicadas pelos dispositivos ESP32;
- **Function Nodes:** convertem as mensagens JSON recebidas em estruturas compatíveis com os comandos SQL utilizados pelo banco de dados;
- **MySQL Out:** executa as instruções de inserção (`INSERT`) no banco `sensorData`, garantindo o armazenamento contínuo dos valores coletados;
- **Debug Nodes:** exibem o status de transmissão e eventuais erros de comunicação durante o teste do fluxo.

Esse arranjo garante a interoperabilidade entre o nível de campo (ESP32 e sensores) e o nível de supervisão (SCADA-LTS e dashboards), permitindo tanto o envio quanto a leitura dos dados em tempo real. O uso do *Node-RED* simplifica o desenvolvimento de integrações complexas por meio de blocos visuais e fluxos lógicos, vale a pena perceber que o Node-RED pode oferecer alta flexibilidade para futuras expansões do sistema de automação.

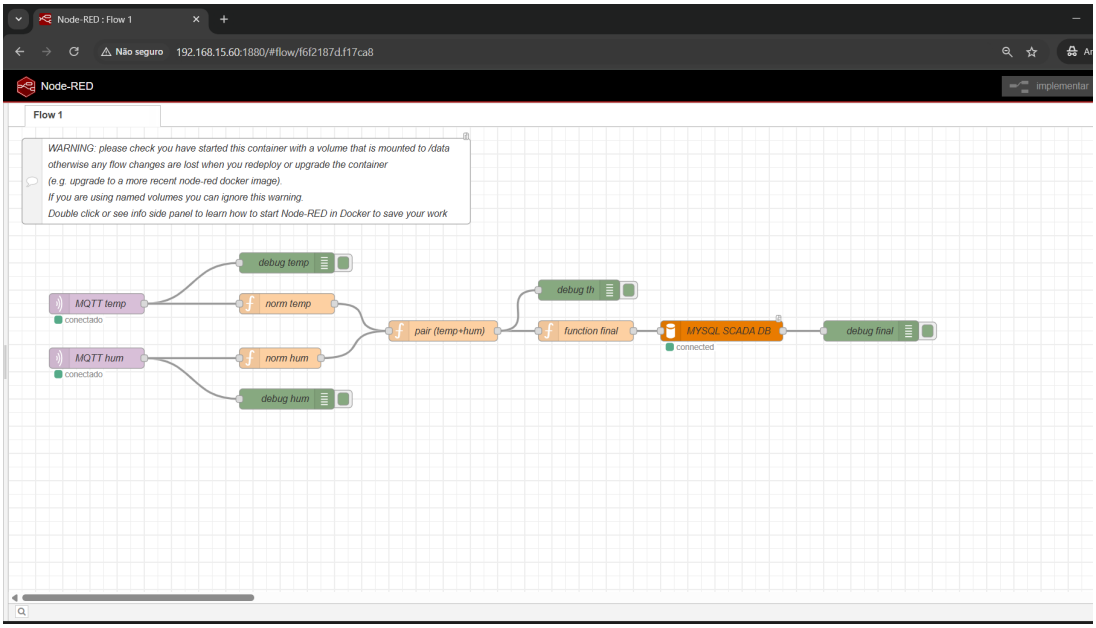


Figura 4.37: Fluxo de integração do *Node-RED* entre o broker MQTT e o banco de dados *MySQL*.

A configuração dos blocos *mqtt in* e do nó de *broker* no Node-RED que viabilizam a recepção de dados de temperatura e umidade publicados pelo ESP32 via MQTT com autenticação e TLS (porta 8883). As Figuras 4.38 a 4.42 registram as telas de configuração adotadas.

4.6.5 Assinatura do tópico de temperatura

O nó ***mqtt in*** para temperatura foi configurado para assinar um único tópico, com decodificação automática de JSON e QoS 1, conforme a Figura 4.38. Os principais campos são:

- **Servidor**: *Broker* ;
- **Ação**: *Assinar um tópico único*;
- **Tópico**: `tcc/esp02/rural/temp`;
- **QoS**: 1 (entrega pelo menos uma vez);
- **Saída**: *um objeto JSON analisado sintaticamente.*

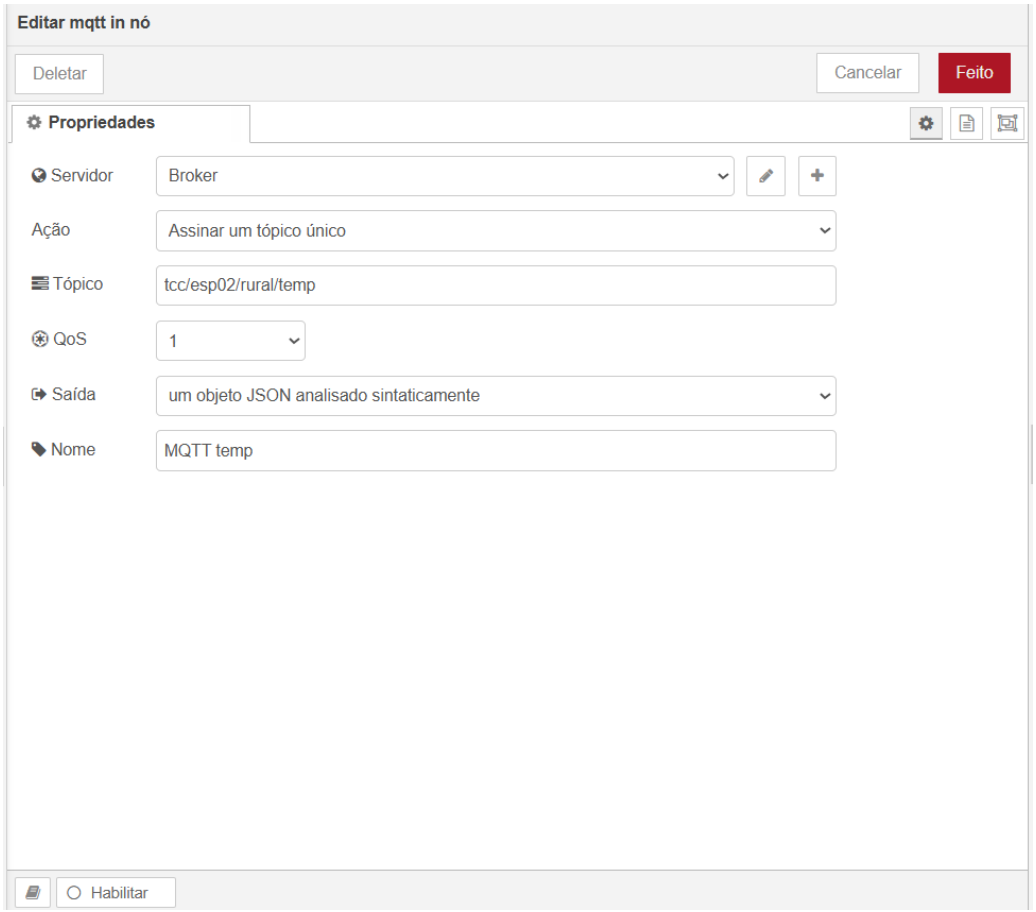


Figura 4.38: Nó *mqtt in* para temperatura.

4.6.6 Assinatura do tópico de umidade

De forma análoga, o nó ***mqtt in*** para umidade assina o tópico `tcc/esp02/rural/hum`, com QoS 1 e saída em JSON (Figura 4.39). Os campos espelham a configuração anterior, alterando apenas o nome e o tópico.

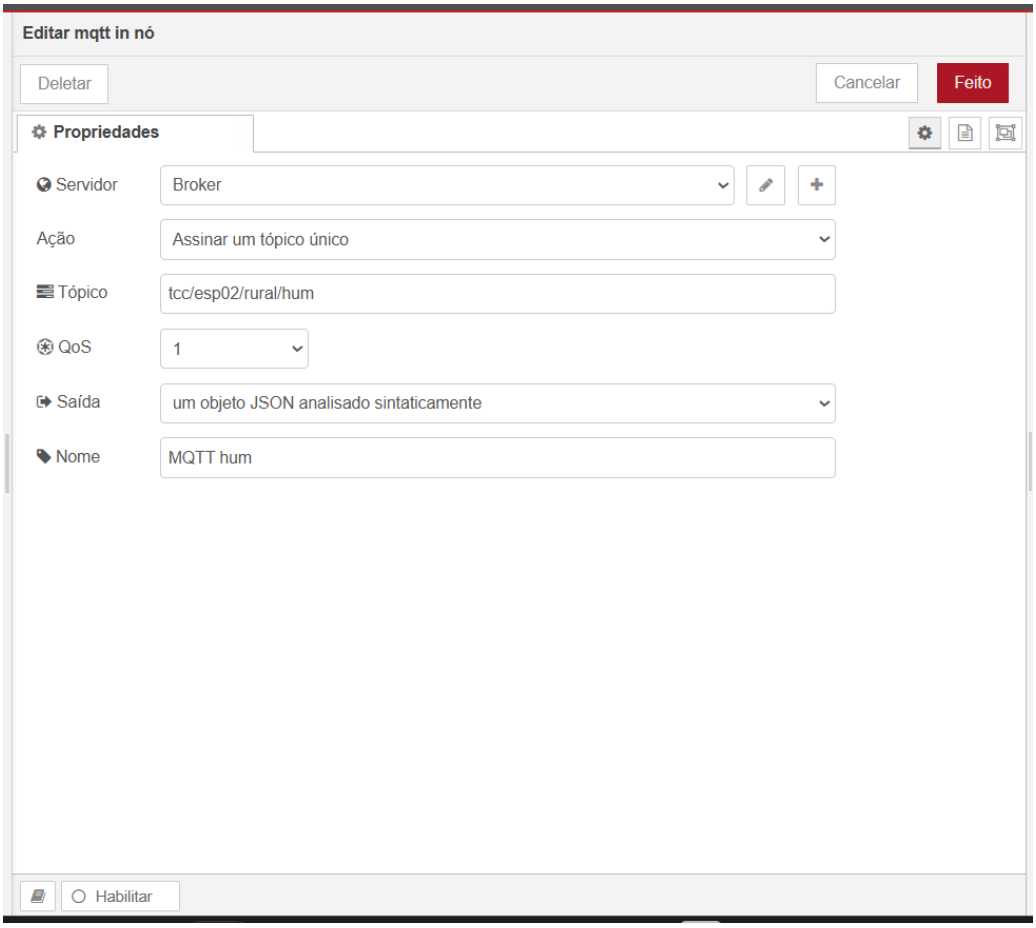


Figura 4.39: Nó *mqtt in* para umidade.

4.6.7 Configuração do *broker* MQTT

O nó ***mqtt-broker*** concentra os parâmetros de conexão, segurança e sessão.

Parâmetros de conexão

A Figura 4.40 mostra a aba *Conexão* que vale tanto para o nó de temperatura quanto para o nó de umidade. Foi utilizado o IP público do broker na porta segura 8883, com reconexão automática e uso de TLS habilitado:

- **Servidor**: `189.110.217.58`;
- **Porta**: `8883` (MQTT sobre TLS);
- **Protocolo**: `MQTT V3.1.1`;
- **Conectar automaticamente**: habilitado;
- **Usar TLS**: habilitado, associado a uma configuração TLS (vide Seção 4.6.8).
- **Keep-alive**: 60 s;

- **Sessão limpa**: habilitada.

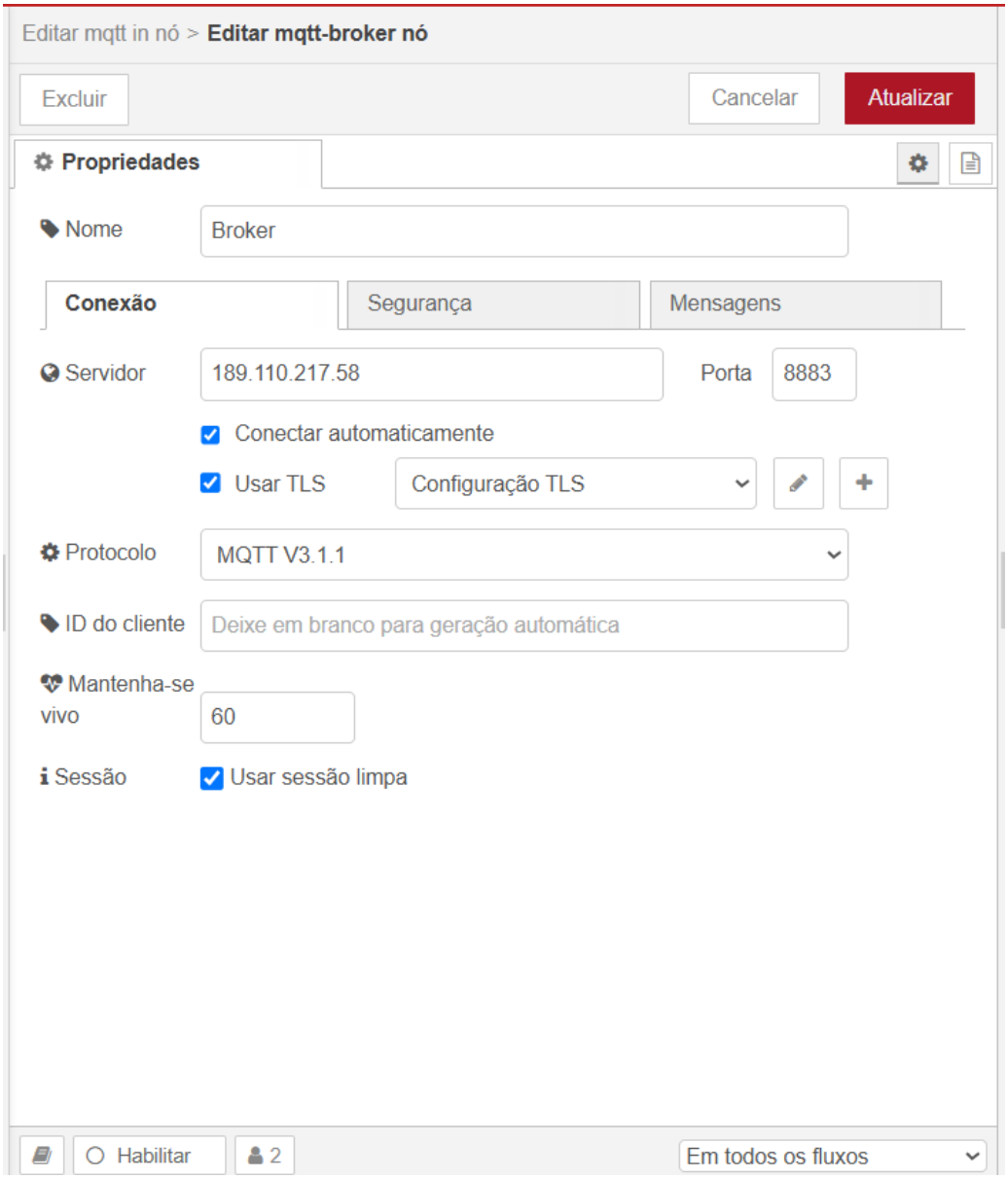


Figura 4.40: Nó *mqtt-broker* (Conexão).

Credenciais de autenticação

A aba *Segurança* que vale tanto para o nó de temperatura quanto para o nó de umidade(Figura 4.41) define as credenciais do cliente MQTT:

- **Nome de usuário**: `gabriel`;
- **Senha**: cadastrada no broker (campo oculto na interface).

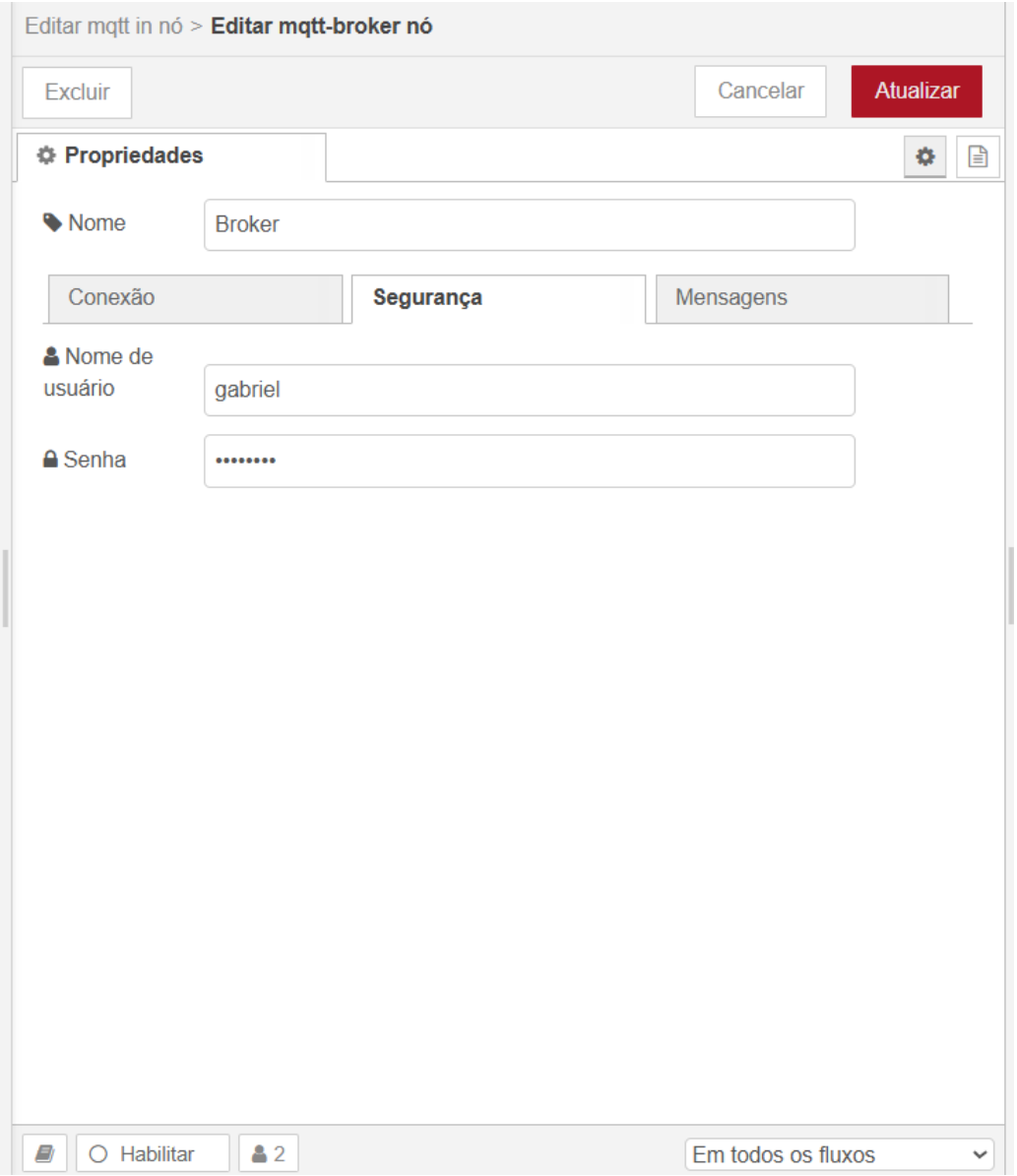


Figura 4.41: Nó *mqtt-broker* (Segurança).

4.6.8 Configuração TLS

A Figura 4.42 documenta o nó ***tls-config*** associado ao broker. Os itens essenciais são:

- **Certificado CA**: arquivo `server.crt` carregado para validar o certificado apresentado pelo broker;
- **Verifique o certificado do servidor**: habilitado, assegurando validação do lado cliente;
- **Certificado/Chave privada do cliente**: não utilizados nesta configuração (autenticação mútua não requerida).

Com essa configuração, a sessão MQTT opera cifrada (TLS), com verificação do certificado do servidor e autenticação por usuário e senha.

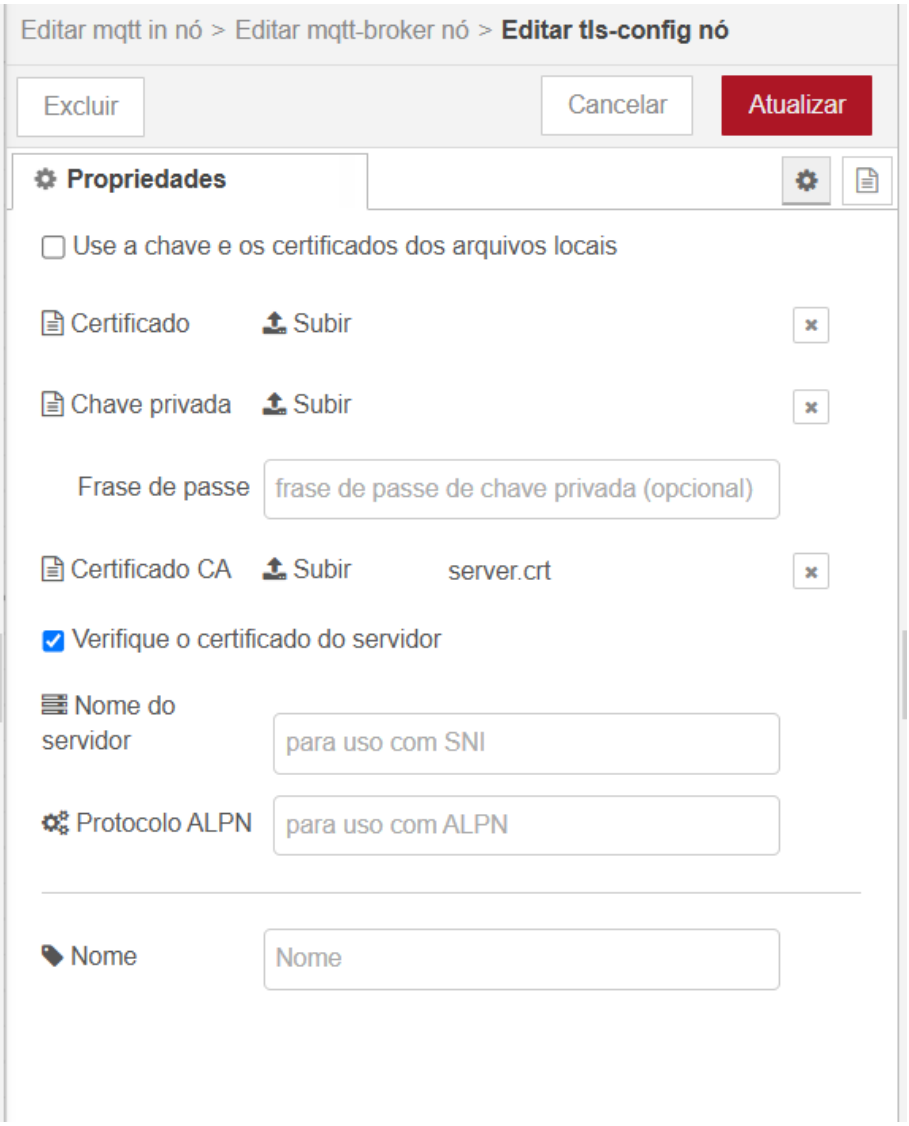


Figura 4.42: Nó *tls-config*: CA `server.crt` carregada.

4.6.9 Resumo operacional dos blocos MQTT

Com o *broker* definido (Seção 4.6.7) e a camada TLS ativa (Seção 4.6.8), os nós *mqtt in* das Figuras 4.38 e 4.39 consomem, com QoS 1, os tópicos:

- `tcc/esp02/rural/temp` (temperatura, JSON);
- `tcc/esp02/rural/hum` (umidade, JSON).

A opção de saída como *objeto JSON* permite que as funções subsequentes do fluxo manipulem diretamente campos como `msg.payload.temp` e `msg.payload.hum`, reduzindo a necessidade de *parsing* manual e evitando erros de conversão.

4.6.10 Funções do Fluxo Node-RED: Normalização, Pareamento e Inserção

Esta seção descreve, em ordem de execução, as funções JavaScript do Node-RED utilizadas no fluxo: *norm temp*, *norm hum*, *pair (temp+hum)* e *function final*. Para cada função, apresentam-se a captura de configuração no editor do Node-RED e a listagem do código empregado.

4.6.11 Função *norm temp*

A Figura 4.43 mostra o editor do nó ***function*** *norm temp*. A função recebe a mensagem publicada no tópico de temperatura e:

1. realiza *parse* opcional de JSON quando o payload chega como string;
2. extrai a temperatura (aceita `temperature`, `temp` ou número direto);
3. obtém o carimbo de tempo `timestamp`/`ts` (ou gera um `ISO 8601`);
4. valida o valor numérico (`Number.isFinite`);
5. padroniza a saída em `msg.topic='temp'` e `msg.payload={val, ts}`.

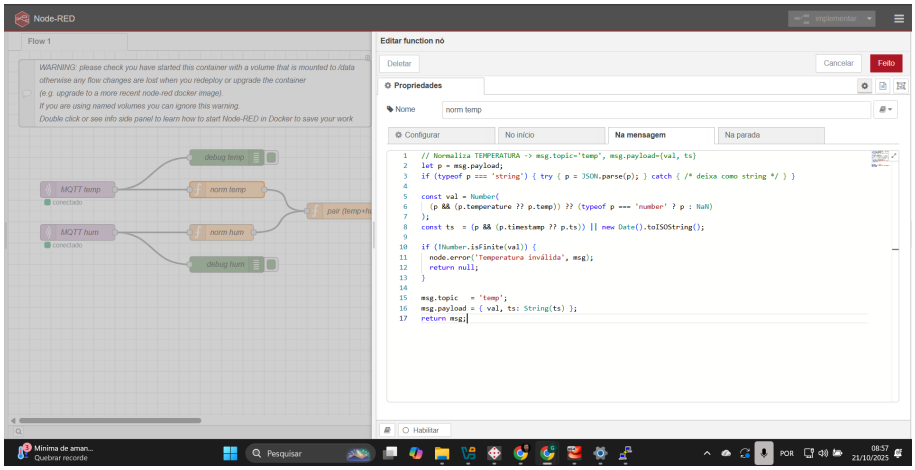


Figura 4.43: Nó *norm temp* no editor do Node-RED.

A implementação completa da função *norm temp* encontra-se no Apêndice C.

4.6.12 Função *norm hum*

A Figura 4.44 apresenta o nó ***function*** *norm hum*. Ele aplica a mesma estratégia de saneamento para a variável de umidade, aceitando chaves `humidity`/`hum` ou número direto, e produz como saída `msg.topic='hum'` e `msg.payload={val, ts}`.

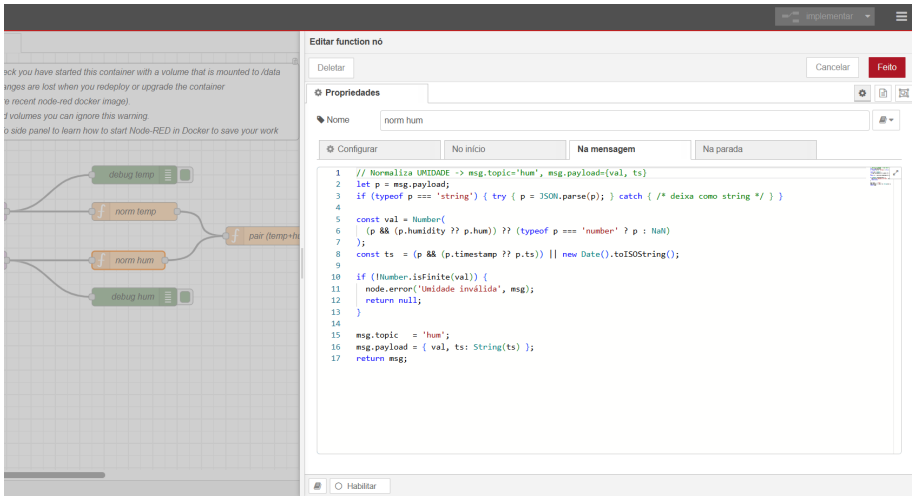


Figura 4.44: Nó *norm hum* no editor do Node-RED.

A função *norm hum* é apresentada integralmente no Apêndice C.

4.6.13 Função *pair (temp+hum)*

A Figura 4.45 exibe o nó ***function*** *pair (temp+hum)*. Ele mantém, no contexto do nó, a última leitura de cada variável e só emite uma mensagem combinada quando os *timestamps* de temperatura e umidade diferem em no máximo 15 s. Leituras vencidas são expiradas para evitar pareamentos defasados.

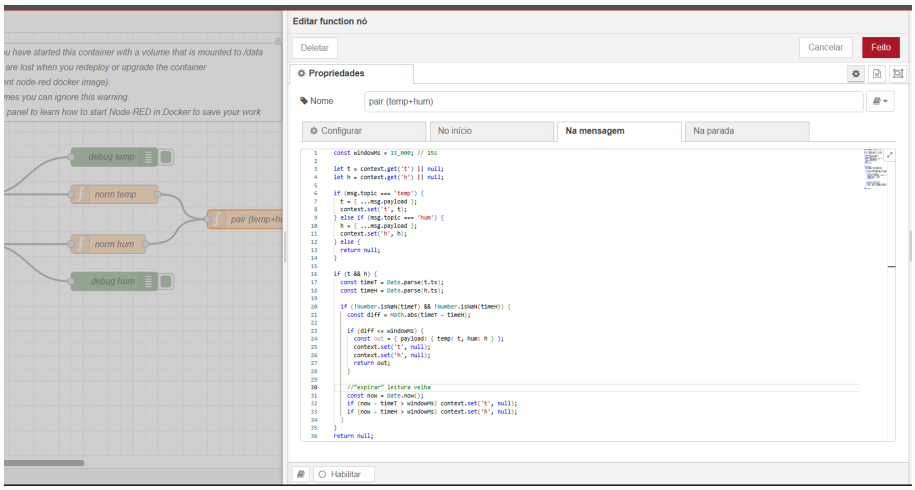


Figura 4.45: Nó *pair (temp+hum)*: pareamento com janela temporal de 15 s.

O código responsável pelo pareamento entre temperatura e umidade pode ser consultado no Apêndice C.

4.6.14 Função *function final*

A Figura 4.46 mostra o nó ***function*** *function final*, que recebe o par {`temp`, `hum`} e prepara a inserção parametrizada no MySQL. A função escolhe um *timestamp* (prioriza o

da temperatura), converte para o formato `YYYY-MM-DD HH:MM:SS` e preenche `msg.topic` com o `INSERT` e `msg.payload` com os valores. O código está na Listagem 19.

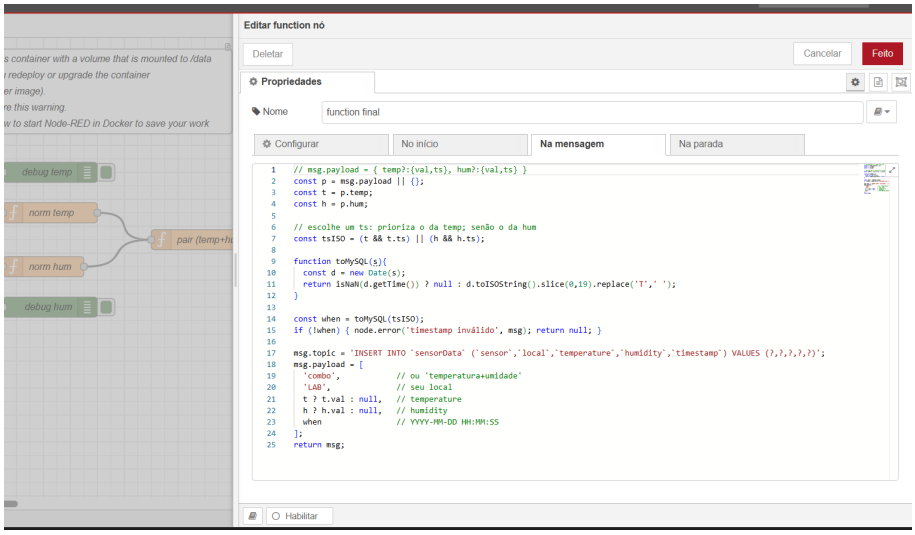


Figura 4.46: Nó *function final*: preparação do `INSERT` parametrizado no MySQL.

A função *function final*, que monta o comando `INSERT` para a tabela `sensorData`, está documentada no Apêndice C.

4.6.15 Pipeline MySQL → SCADA-LTS (configuração passo a passo)

1) Acesso ao phpMyAdmin (login)

O *phpMyAdmin* foi instalado na máquina virtual Linux como um serviço definido no arquivo `docker-compose.yml`, integrado ao contêiner do MySQL e exposto para acesso via navegador a partir do host pelo endereço `192.168.15.60/mysql/`.

A Figura 4.47 mostra a tela de autenticação do *phpMyAdmin*. Neste ponto, utiliza-se o usuário e senha definidos na instalação (na VM), com o idioma ajustado para *Português (Brasil)*. Após a autenticação, navega-se até o servidor MySQL para criar (ou conferir) o banco e a tabela do projeto.

Figura 4.47: Tela de login do *phpMyAdmin*.

2) Criação/Verificação do banco `SCADA` e da tabela `sensorData`

A Figura 4.48 apresenta a visão do banco `SCADA` no *phpMyAdmin*, já contendo a tabela `sensorData`. Caso precise criar do zero, utilize o script no apêndice D.

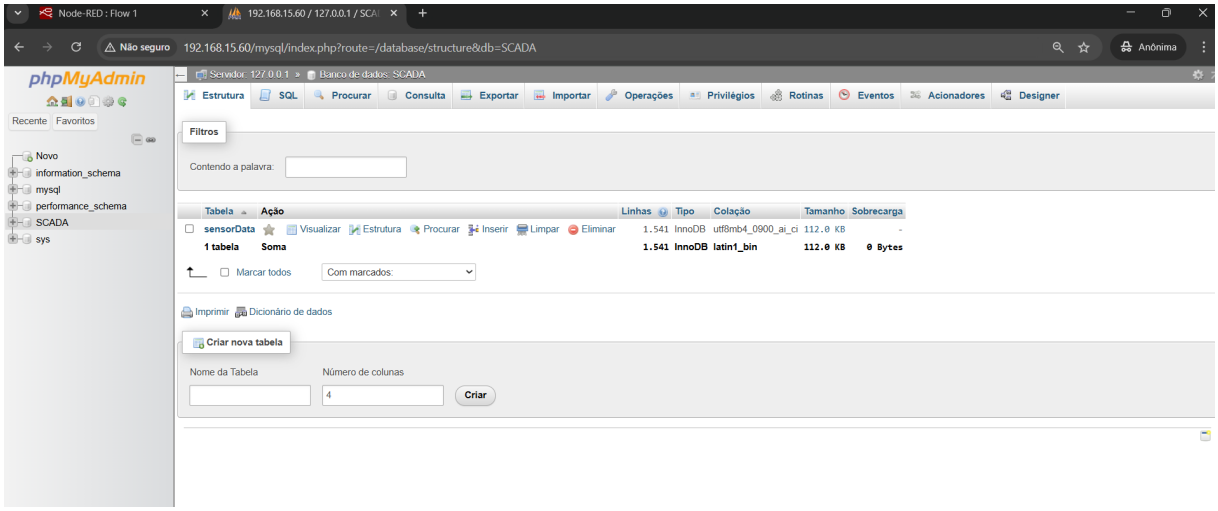


Figura 4.48: Banco `SCADA` no *phpMyAdmin* com a tabela `sensorData`.

Observação prática. O Node-RED insere linhas nesta tabela por meio do nó `mysql` usando `INSERT` (vide função *function final* descrita em seção anterior), preenchendo `sensor`, `local`, `temperature`, `humidity` e `timestamp`.

3) Instalação do *SCADA-LTS*

O *SCADA-LTS* foi instalado na máquina virtual Linux por meio de um contêiner Docker definido no arquivo `docker-compose.yml`, apontando para o banco de dados MySQL e expondo a porta de acesso HTTP na VM(8082). Após a inicialização dos contêineres, a

interface web foi acessada via navegador usando o IP da VM, com autenticação inicial pelas credenciais padrão do sistema: usuário `admin` e senha `admin`.

4) Criação dos *Data Sources* SQL no SCADA-LTS

A Figura 4.49 mostra dois *Data Sources* do tipo `SQL`, chamados `hum_rural` e `temp_rural`, ambos conectados ao banco `SCADA` via JDBC. O *connection string* utiliza o `com.mysql.cj.jdbc.Driver` e define parâmetros de codificação e tempo limite.

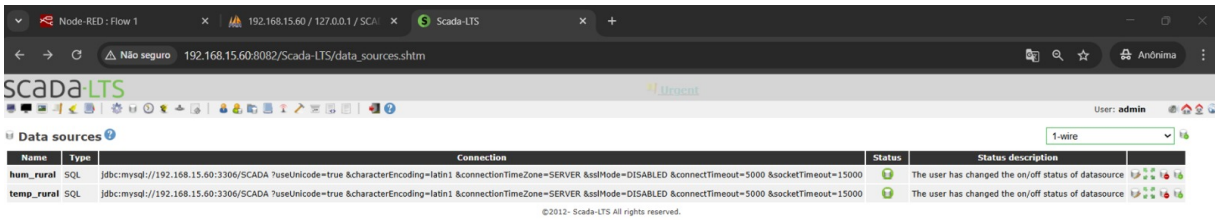


Figura 4.49: *SCADA-LTS*: lista de *Data Sources* `hum_rural` e `temp_rural` (tipo `SQL`).

5) Data Source `hum_rural`: consulta e ponto de medição

A Figura 4.50 detalha a configuração do `hum_rural`. O período de atualização está em 10 s e a **consulta** retorna sempre o último registro de umidade (`humidity`) de `sensorData`, mapeando colunas para o SCADA-LTS conforme: `valueCol` (valor numérico) e `timeCol` (carimbo de tempo). O *Point Humidity* está configurado como `Numeric`, com *Column name* = `valueCol` e *Time override column* = `timeCol`.

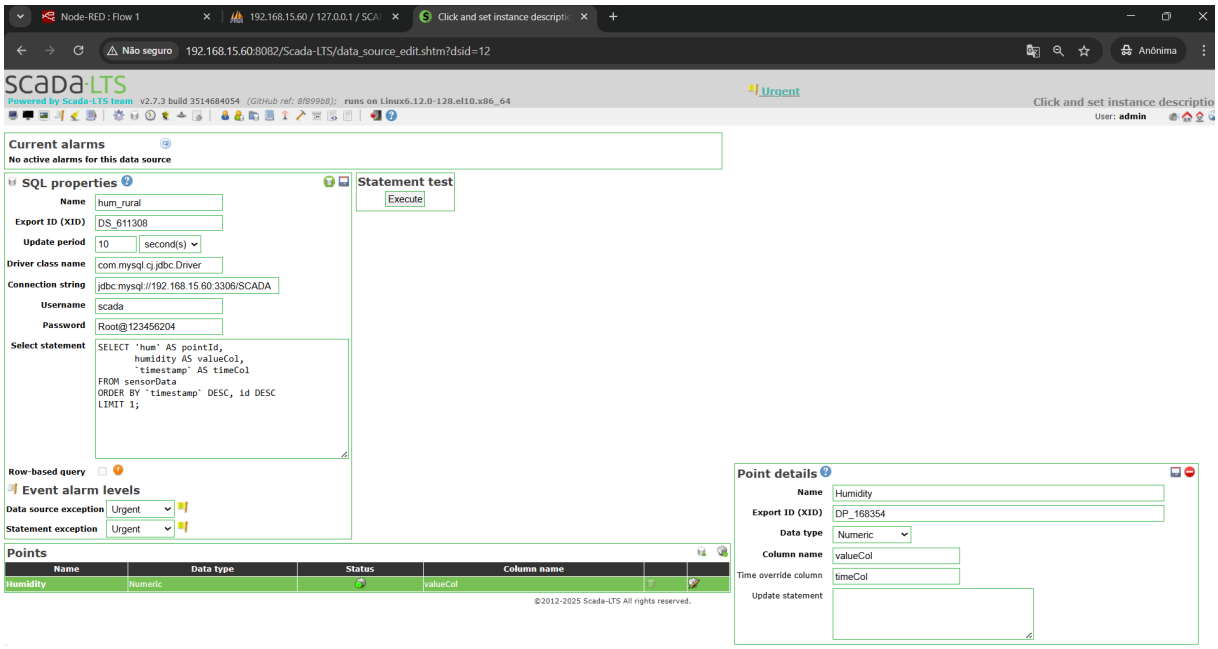


Figura 4.50: *SCADA-LTS* → `hum_rural`: propriedades SQL, *Select Statement* e *Point details*.

6) Data Source `temp_rural`: consulta e ponto de medição

De forma análoga, a Figura 4.51 apresenta o `temp_rural`, cuja consulta obtém a última temperatura (`temperature`) registrada. O *Point Temperatura* também é `Numeric`, com *Column name* = `valueCol` e *Time override column* = `timeCol`.

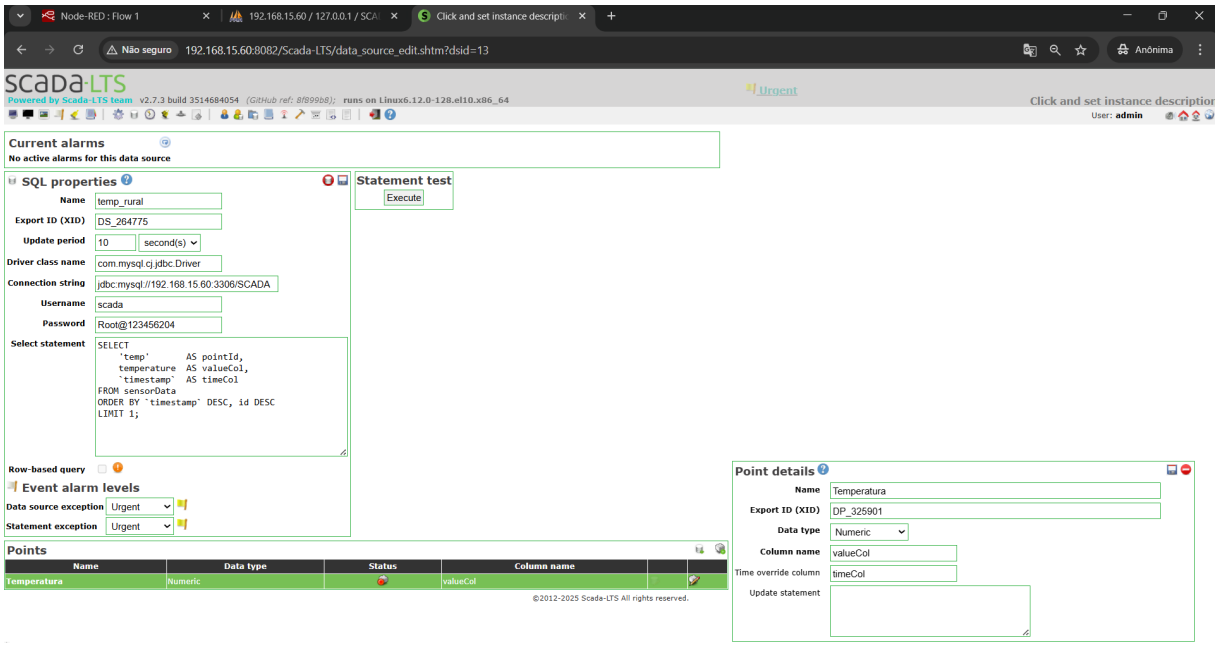


Figura 4.51: *SCADA-LTS* → `temp_rural`: propriedades SQL, *Select Statement* e *Point details*.

Validação e notas de segurança

Com os *Data Sources* habilitados e **conectados** (ícone verde em *Status* na Figura 4.49), os *Points* passam a refletir, a cada 10 s, o último par {`valor`, `timestamp`} gravado na tabela `sensorData`. Recomenda-se:

- criar um usuário dedicado no MySQL (por exemplo, `scada`) com permissões mínimas de `SELECT` sobre `SCADA.sensorData`;
- manter o fuso horário do `connectionTimeZone` e do sistema alinhados para evitar *drift* nos painéis e alarmes;
- testar as consultas pelo botão *Execute* (*Statement test*) nas telas dos *Data Sources* (Figuras 4.50 e 4.51).

4.6.16 Resultado final: gráfico em tempo real no SCADA-LTS (Modern Watch List)

A Figura 4.52 sintetiza o funcionamento completo da arquitetura implementada --- do ESP32 ao supervisório --- exibindo a série temporal de **temperatura** atualizada em

tempo real no *Modern Watch List* do *SCADA-LTS*. Os dados percorrem o pipeline *ESP32 → MQTT (TLS) → Node-RED → MySQL → SCADA-LTS*, onde:

- os nós *norm temp* e *norm hum* padronizam as medidas;
- o nó *pair (temp+hum)* garante sincronismo por janela temporal;
- o nó *function final* realiza o `INSERT` parametrizado na tabela `SCADA.sensorData`;
- os *Data Sources* SQL `temp_rural` e `hum_rural` (atualização a cada 10 s) publicam os pontos no SCADA-LTS.

O gráfico apresenta navegação temporal interativa (controle deslizante superior), *tooltips* por amostra e a curva de tendência da variável monitorada ao longo do dia, evidenciando a variação térmica típica: resfriamento na madrugada seguido de aquecimento progressivo no período diurno.

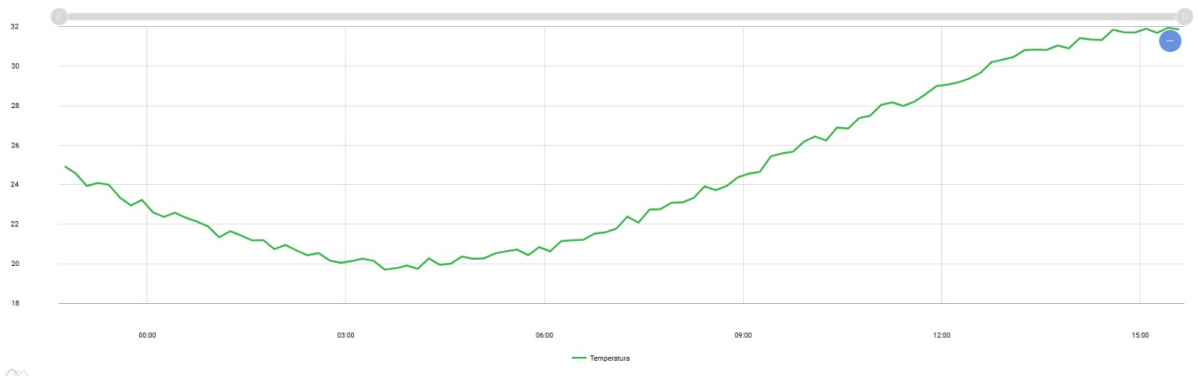


Figura 4.52: *SCADA-LTS Modern Watch List*: série de temperatura atualizada em tempo real a partir da tabela `sensorData`.

Os scripts SQL completos utilizados na configuração do *pipeline* MySQL → SCADA-LTS foram reunidos no Apêndice D, de forma a não sobrecarregar o texto principal.

Capítulo 5

Resultados e Discussões

5.1 Testes Realizados

Esta seção registra os ensaios funcionais executados nos nós urbano (ESP32 01) e rural (ESP32 02), abrangendo controle por servidor web, aplicativo MQTT no smartphone e acionamento local por *push button*, além da operação do modo automático (agendas). A Tabela 5.1 sintetiza os resultados.

Tabela 5.1: Síntese dos testes funcionais por nó, canal de comando e alvo

ID	Nó	Canal	Alvo	Resultado/Observação
1	ESP01	Servidor web	Lâmpada 1	OK: comutação executada
2	ESP01	Servidor web	Lâmpada 2	OK: comutação executada
3	ESP01	Servidor web	Lâmpada 3	OK: comutação executada
4	ESP01	Servidor web	Modo automático (lâmpadas)	OK: ativação/desativação funcionou
5	ESP01	MQTT (smartphone)	Lâmpada 1	OK: comutação executada
6	ESP01	MQTT (smartphone)	Lâmpada 2	OK: comutação executada
7	ESP01	MQTT (smartphone)	Lâmpada 3	OK: comutação executada
8	ESP01	MQTT (smartphone)	Modo automático (lâmpadas)	OK: ativação/desativação funcionou
9	ESP01	*Push button*	Lâmpada 1	OK: comutação local
10	ESP01	*Push button*	Lâmpada 2	OK: comutação local
11	ESP01	*Push button*	Lâmpada 3	OK: comutação local
12	ESP01	*Push button*	Modo automático (lâmpadas)	OK: ativação/desativação funcionou
13	ESP02	MQTT (smartphone)	Lâmpada 1	OK: comutação executada
14	ESP02	MQTT (smartphone)	Lâmpada 2	OK: comutação executada
15	ESP02	MQTT (smartphone)	Lâmpada 3	OK: comutação executada
16	ESP02	MQTT (smartphone)	Modo automático (lâmpadas)	**Parcial**: agenda liga com **deslocamento de 3 h**
17	ESP02	MQTT (smartphone)	Irrigação (manual)	OK: comutação executada
18	ESP02	MQTT (smartphone)	Modo automático (irrigação)	**Parcial**: agenda liga com **deslocamento de 3 h**

5.2 Análise dos Resultados

Controle distribuído e coerência de estados

No nó urbano (ESP01), o controle por servidor web, aplicativo MQTT e *push button* apresentou comportamento consistente, com comutação correta das três lâmpadas e ativação do modo automático. A redundância de canais de comando reforça a disponibilidade operacional: em ausência temporária de rede, o acionamento local permanece funcional, e quando a rede está presente, o controle remoto é efetivo.

No nó rural (ESP02), as comutações por aplicativo MQTT funcionaram para lâmpadas e irrigação. A ativação do modo automático (lâmpadas e irrigação) também foi bem-sucedida, porém com **deslocamento de 3 horas** em relação ao horário planejado.

Anomalia de agendamento: deslocamento de 3 horas

O **offset de 3 h** observado em agendas do ESP02 é típico de *mismatch* de fuso/relógio entre os elementos do sistema. Hipóteses prováveis:

- ESP32 sem ajuste de *timezone* após sincronização SNTP/NTP (relógio interno em UTC e agenda interpretada como local).
- RTC externo descalibrado ou gravado em horário local enquanto o software espera UTC (ou vice-versa).

Registro: a anomalia não impediu a comutação; o sistema executou as agendas, porém em horário deslocado. A correção será tratada na seção de trabalhos corretivos (Capítulo de Conclusões/Trabalhos Futuros), alinhando todos os componentes para o fuso `America/Sao_Paulo` e padronizando o armazenamento em UTC com exibição local.

Observou-se, durante os ensaios, que o ESP32-02 registrava no SCADA-LTS *timestamps* coerentes com o horário configurado na máquina virtual, isto é, os gráficos de temperatura, umidade do solo e estados de válvula apresentavam o tempo correto. No entanto, as agendas automáticas de iluminação e irrigação eram executadas com um deslocamento de aproximadamente 3 horas em relação ao horário esperado. Isso indica que a base de tempo utilizada para o registro histórico no supervisório estava alinhada ao fuso adotado pelo servidor, enquanto a lógica de agendamento local no nó rural utilizava uma referência distinta (por exemplo, UTC sem ajuste de fuso ou RTC configurado em horário diferente), gerando o descompasso entre o horário "visto" nos gráficos e o momento efetivo de acionamento das cargas.

Séries temporais e evidências visuais (SCADA-LTS)

Os gráficos desta seção têm a finalidade de evidenciar, de forma ilustrativa, as funcionalidades implementadas no sistema até o nível de integração com o Scada-LTS.

Temperatura em tempo real (LIVE): A Figura 5.1 apresenta o registro de temperatura em modo *LIVE* no *Modern Watch List*. Observa-se uma única curva em verde, associada à temperatura, variando aproximadamente de 22,5 ℃ no início da manhã para um pico em torno de 30--31 ℃ no início da tarde, seguido de resfriamento gradual até cerca de 25 ℃ próximo das 18:00. A evolução ao longo do período exibido (aproximadamente de 07:00 a 18:00) é suave, com pequenas flutuações pontuais.

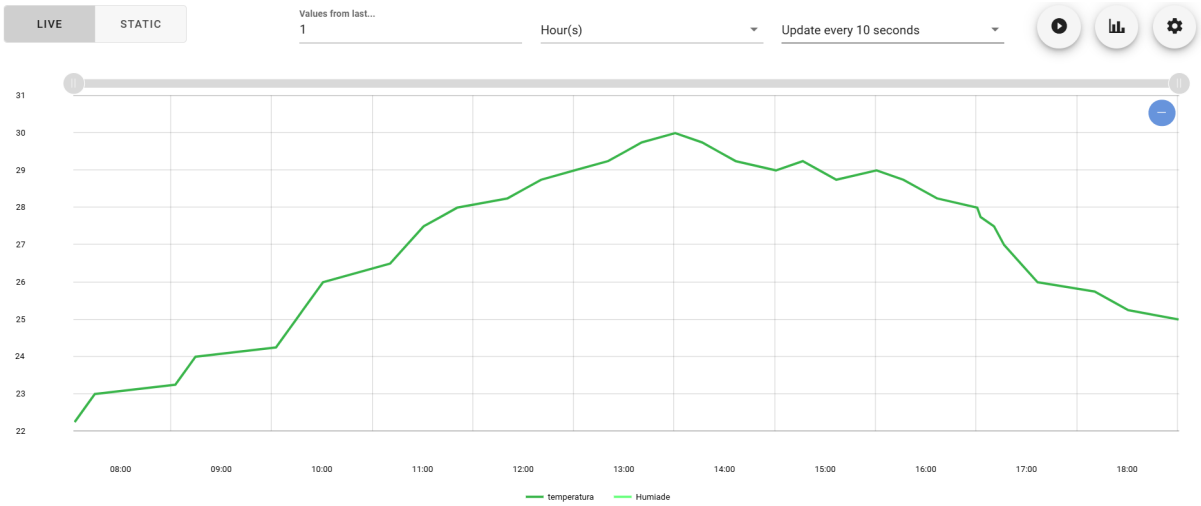


Figura 5.1: Temperatura em tempo real no SCADA-LTS (Modern Watch List).

Umidade em tempo real (LIVE): Durante o acompanhamento em tempo real da umidade no nó rural, a Figura 5.2 exibe a série da variável *hum_rural1* em modo *LIVE* no *Modern Watch List*.

Figura 5.2: Umidade relativa no nó rural em modo LIVE no SCADA-LTS (Modern Watch List).

Temperatura e umidade diárias (24 h, sobrepostas). A Figura 5.3 apresenta, em modo *LIVE* no *Modern Watch List*, as séries de temperatura e umidade ao longo do dia. As duas variáveis são mostradas em verde, em faixas distintas de valores: a temperatura varia de aproximadamente 26 °C no final da manhã até um máximo em torno de 29--30 °C próximo de 13:00, decrescendo gradualmente para cerca de 25 °C no fim da tarde. A umidade relativa permanece concentrada na faixa de 56--59 %, com oscilações suaves ao redor de um patamar quase constante, sem tendência marcada de aumento ou redução ao longo do intervalo observado.

Figura 5.3: Temperatura e umidade no SCADA-LTS (Watch List, 24 h).

Temperatura diária (histórico): A Figura 5.4 mostra a evolução da temperatura ao longo do dia no módulo de gráfico do Watch List do SCADA-LTS, ou seja, é a reconstrução do gráfico na faixa de tempo selecionada.

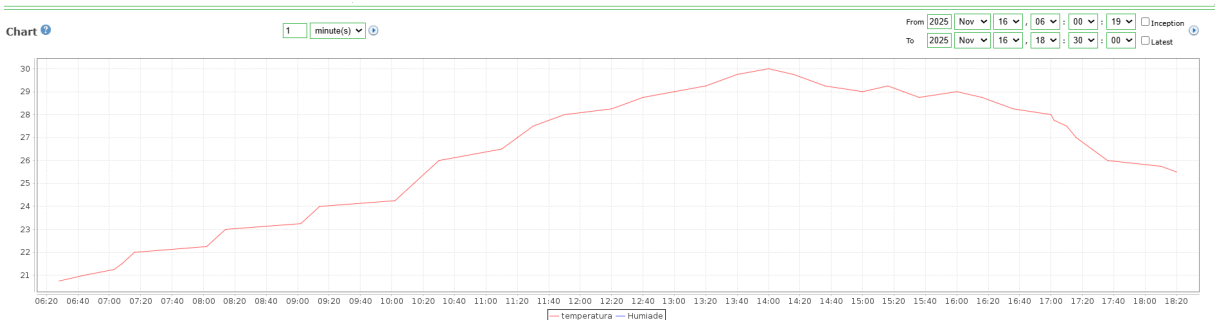


Figura 5.4: Temperatura diária no SCADA-LTS (Watch List, 24 h).

Umidade diária (histórico). A Figura 5.5 mostra a evolução da umidade ao longo do dia no módulo de gráfico Watch List do SCADA-LTS, ou seja, é a reconstrução do gráfico na faixa de tempo selecionada.

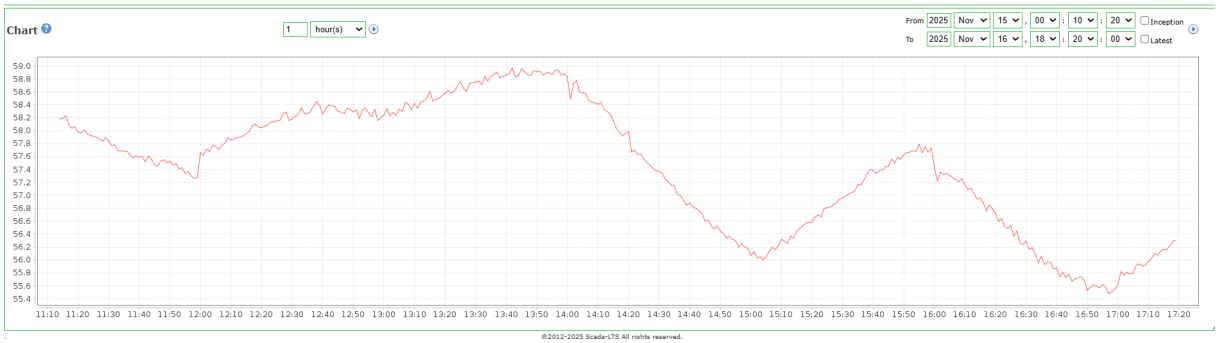


Figura 5.5: Umidade diária no SCADA-LTS (Watch List, 24 h).

Temperatura e umidade diárias (histórico): A Figura 5.6 apresenta, a partir de 06:00, as séries de temperatura e umidade registradas no módulo de gráfico do Watch List do SCADA-LTS. A curva de temperatura (em vermelho) inicia o período em torno de 20--21 °C, eleva-se de forma quase contínua ao longo da manhã, passando pela faixa de 24--27 °C entre aproximadamente 09:00 e 11:00, e atinge um máximo em torno de 30 °C por volta de 14:00. Em seguida, observa-se leve decréscimo, encerrando o intervalo em cerca de 27--28 °C. A curva de umidade relativa (em azul) aparece na faixa de 55--59 %, iniciando próximo de 55--56 % por volta das 11:00 às 11:30, subindo até aproximadamente 59--60 % próximo de 14:00 e retornando gradualmente para cerca de 56--57 % no final do período, caracterizando oscilações moderadas em torno de um patamar quase constante(desconsiderar dados antes do horário 6:00, pois eram apenas dados de teste).

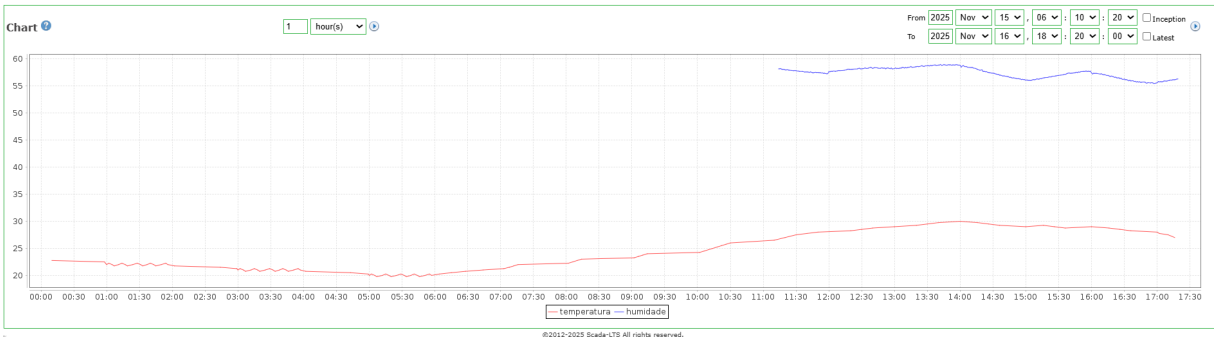


Figura 5.6: Umidade diária no SCADA-LTS (Watch List, 24 h).

5.3 Eficiência Energética e Confiabilidade

5.3.1 Metodologia de estimativa

A energia foi estimada a partir da potência ativa e do tempo de acionamento obtido nos *logs* de estados/comandos. Adotou-se

$$P \approx V_{\text{rms}}\, I_{\text{rms}}\, \cos\varphi \qquad \text{e} \qquad E\,[\text{kWh}] = \frac{P\,[\text{W}] \cdot t\,[\text{h}]}{1000}.$$

As janelas fixas do modo automático consideradas na consolidação foram: iluminação diária das 18:30 às 05:00 e irrigação das 09:00 às 09:10. O *overhead* (nós ESP32, relés e conversores) foi contabilizado conforme detalhado nas premissas da planilha *Cálculo da energia*.

5.3.2 Premissas utilizadas (planilha *Cálculo da energia*)

- **ESP32 (dissipação no regulador)**: queda de $\Delta V \approx 1{,}7\,\text{V}$ com $I \approx 0{,}5\,\text{A}$ por nó, logo $P_{\text{LDO}} \approx 0{,}85\,\text{W} = 0{,}00085\,\text{kW}$.

- **Relés 5 V**: potência por canal $P \approx 0{,}36\,\mathrm{W}$ (bobina 5 V). Na casa foram considerados 3 relés e no sítio 4 relés.
- **Válvula solenóide 3/4'' (DN20), 110 V AC, NC**: potência típica de 20–25 W quando energizada (valor de referência de 22 W na planilha).
- **Iluminação**: casa com duas lâmpadas frias de 15 W e uma lâmpada amarela de 9 W (total 39 W); sítio com duas lâmpadas frias de 15 W e um conjunto tubular com duas de 18 W (total 66 W).

5.3.3 Resultados consolidados (modo automático)

Tabela 5.2: Estimativa de potência, energia mensal e custo por item (cenário de modo automático).

Item	Entrada (V)	Saída (V)	ΔV (V)	Corrente (A)	Potência (kW)	Energia (kWh)	Custo (R$)
ESP32-01 funcionamento	5	3,3	--	0,5	0,00165	0,0495	0,03
ESP32-01 energia dissipada	5	3,3	1,7	0,5	0,000850	0,612	0,43
ESP32-02 funcionamento	5	3,3	--	0,5	0,00165	0,0495	0,03
ESP32-02 energia dissipada	5	3,3	1,7	0,5	0,000850	0,612	0,43
4 relés (casa)	--	--	--	--	0,001440	1,0368	0,73
4 relés (sítio)	--	--	--	--	0,001440	1,0368	0,73
Válvula solenóide (modo automático)	--	--	--	--	0,022	0,11	0,08
3 lâmpadas (casa, modo automático)	--	--	--	--	0,039	12,285	8,60
3 lâmpadas (sítio, modo automático)	--	--	--	--	0,066	20,79	14,55
						TOTAL mês	**25,61**
						TOTAL ano	**307,29**

5.3.4 Fontes das potências nominais (links de referência)

Tabela 5.3: Potências nominais e links utilizados na planilha.

Item	Quantidade	Potência (W)	Link
Lâmpada fria (casa)	2	15	Kian A60 15 W 6500 K
Lâmpada amarela (casa)	1	9	Kian 9 W 3000 K (ML)
Lâmpada fria (sítio)	2	15	Kian A60 15 W 6500 K
Lâmpada fria tubular (sítio, T8)	2	18	T8 18 W 6500 K 120 cm
Born relé (SRD-05VDC-SL-C)	8	0,36	Songle SRD-05VDC-SL-C
Válvula solenóide 3/4'' 110 V AC	1	22	Plastic Solenoid Valve

5.3.5 Explicações complementares

1. **Fonte do nó ESP32:** o ESP32 não regula *corrente*, apenas *tensão*. Assim, mesmo que a fonte forneça 1 A em 5 V, o microcontrolador consome apenas o necessário

(tipicamente 80–250 mA, podendo chegar a ≈500 mA com Wi-Fi ativo). Corrente disponível acima disso não causa problema; o risco está em exceder a tensão de entrada, não a capacidade de corrente.

2. **Válvula solenóide 3/4'' 110 V AC:** bobinas AC dessa família (DN20, corpo plástico, NC) operam tipicamente na faixa de 20–25 W (ex.: especificação comercial de *20 VA* para modelo plástico 3/4'').

   Esse valor deve ser considerado no dimensionamento de fonte, fiação, relé/driver e dissipação térmica.

5.4 Adoção do ESP32

A opção pelo microcontrolador ESP32, em detrimento de plataformas clássicas como o *Arduino Uno* ou *Mega*, está diretamente ligada às exigências de conectividade e processamento do projeto. O ESP32 integra, em um único componente, interfaces Wi-Fi e Bluetooth, eliminando a necessidade de módulos adicionais para acesso à rede e reduzindo custo, espaço físico e complexidade de cabeamento O ESP32 dispõe de maior capacidade de memória, frequência de operação mais elevada.

5.5 Adoção do SCADA-LTS

Optou-se pela utilização do *SCADA-LTS* em vez de plataformas em nuvem como ThingSpeak, Firebase ou Blynk porque, além de se aproximar do ambiente de supervisão industrial que se deseja simular no protótipo, atende ao caráter educacional do projeto, permitindo ao autor ter contato direto com ferramentas e conceitos mais próximos da realidade da automação industrial.

5.6 Adoção do RTC DS3231 e do sensor HW-390

A adoção conjunta do RTC DS3231 e do sensor de umidade do solo HW-390, em vez de sensores convencionais como o DHT11 ou DHT22, está ligada à necessidade de medições mais consistentes e úteis para o contexto de automação proposto. O DS3231 fornece não apenas uma base de tempo em tempo real, independente de conexão à Internet ou do estado da rede elétrica, como também dispõe de um sensor interno de temperatura, permitindo associar cada leitura a um carimbo de tempo confiável. Isso facilita o registro histórico, a comparação entre dias e a implementação de rotinas de agendamento no protótipo. Já o HW-390 foi escolhido por ser um sensor capacitivo de umidade do solo, mais adequado para monitoramento.

Capítulo 6

Conclusão

6.1 Síntese dos Resultados

Esta seção sintetiza os principais achados experimentais do sistema de automação residencial (ESP32-01) e de irrigação rural (ESP32-02), considerando comando local e remoto, agendas automáticas, telemetria/visualização e custos energéticos.

Atuação e controle

- **ESP32-01 (nó urbano)**: as três formas de comando de iluminação (servidor web embarcado, aplicativo MQTT no smartphone e *push button* local) operaram corretamente. O *modo automático* executou as janelas programadas conforme esperado, mantendo a redundância entre comando remoto e contingência local.
- **ESP32-02 (nó rural)**: o comando de lâmpadas via MQTT apresentou funcionamento consistente. O *modo automático* funcionou *parcialmente*, registrando **deslocamento de 3 horas** na execução das agendas. Esse mesmo deslocamento ocorreu ao agendar pelo aplicativo MQTT no smartphone e também na irrigação automática. A irrigação via MQTT (*on demand*) funcionou corretamente.

Telemetria e visualização

- **Modern WatchList**: exibiu temperatura e umidade;
- **WatchList (padrão)**: apresentou séries em uma determinada faixa de tempo para temperatura e umidade separadamente e, posteriormente, ambas as variáveis no **mesmo gráfico** operando.

Eficiência energética e custo

- Com base na planilha *Cálculo da energia* e na Tabela 5.2, o cenário de operação em *modo automático* apresentou **baixo custo operacional**: **R$ 25,61/mês** e **R$ 307,29/ano** (incluindo cargas principais e *overhead* de nós/relés/fonte). Esse resultado indica viabilidade econômica para operação contínua.

Síntese

Em conjunto, os resultados confirmam a **viabilidade técnica** do controle distribuído: servidor web e MQTT atenderam ao comando remoto, enquanto o *push button* garantiu contingência local. A visualização dos gráficos no *Modern WatchList* para um **monitoramento consolidado** com séries e gráfico unificado de temperatura e umidade teve resultado positivo. Do ponto de vista energético, o sistema apresentou **consumo e custo mensais reduzidos**, compatíveis com a proposta de automação acessível. O principal desvio observado foi o **deslocamento de 3 horas** nas agendas do nó rural (ESP32-02) e nos agendamentos via smartphone.

6.2 Limitações do Projeto

1. **Dependência de infraestrutura local e ponto único de falha:** o *broker* MQTT foi executado em máquina virtual (PC pessoal). A disponibilidade do sistema depende do computador estar ligado e da VM estar íntegra, criando um ponto único de falha e ausência de alta disponibilidade.
2. **Instabilidade da supervisão:** o SCADA-LTS apresentou oscilações (ex.: *Modern WatchList* deixou de atualizar algumas vezes), o que impactou a observabilidade contínua.O *WatchList* também apresentou alguns problemas na apresntação dos gráficos(Os gráficos não apareciam mesmo selecionando a faixa de tempo correta e fazendo as configurações corretas).
3. **Desalinhamento temporal nas agendas:** no ESP32-02 e no agendamento via aplicativo MQTT no smartphone observou-se **deslocamento de 3 horas** na execução do modo automático, evidenciando problemas de sincronização de tempo/fuso ao longo da cadeia (nó, servidor e interface).
4. **Critério de irrigação predominantemente temporal:** a irrigação automática foi baseada em horário fixo (09:00–09:10). Na ausência de inibição por chuva ou solo já úmido, há risco de acionamento desnecessário em condições climáticas adversas, a situação de controle não foi implementada por conta da limitação da verba do projeto.
5. **Medição energética indireta:** a estimativa de energia/custo baseou-se em potências

nominais e tempos de acionamento (Tabela 5.2), sem medição elétrica dedicada. Optou-se por esse caminho por conta da insuficiência de verba do projeto.

6. **Eficiência da alimentação dos nós:** o uso de regulador linear (LDO) com $\Delta V \approx 1{,}7\,\mathrm{V}$ e $I \approx 0{,}5\,\mathrm{A}$ implica dissipação térmica relevante no regime contínuo, reduzindo eficiência e margem térmica em ambientes quentes.
7. **Robustez elétrica e física do atuador/comutação:** faltam elementos de proteção e de supressão de surtos/transientes (fusíveis, MOV/TVS, snubber em cargas indutivas AC), bem como encapsulamento com grau de proteção adequado (IP) para operação em ambiente rural.
8. **Manutenibilidade e escalabilidade:** a arquitetura atual exige intervenção manual em caso de falhas de componentização (relés, válvulas, cabeamento) e não contempla mecanismos de *auto-recovery* ou *backup/restore* automatizados de configuração e dados.
9. **Controle de tomada:** não foi implementado um ponto de tomada com acionamento remoto via MQTT, em função das limitações orçamentárias do projeto.
10. **Sistema de controle por umidade do solo:** não foi implementado o acionamento automático da válvula de irrigação a partir do sensor de umidade do solo via MQTT, também devido às restrições de orçamento na fase de prototipagem (falta de cabos até a válvula de irrigação).

6.3 Sugestões para Trabalhos Futuros

A partir das limitações mapeadas e dos resultados experimentais, são propostos os seguintes desdobramentos:

1. **Supervisão mais estável e observável:** migrar do SCADA-LTS para solução mais robusta (p.ex., SCADA-BR).
2. **Alta disponibilidade e independência do PC:** hospedar o *broker* em dispositivo dedicado 24/7 (Raspberry Pi/NUC) ou VPS; empregar *watchdogs*, reinício automático e *bridging* entre *brokers*; proteger a energia com UPS.
3. **Sincronização de tempo ponta a ponta:** padronizar armazenamento e agendamento em UTC, reforçar SNTP/RTC (DS3231) como fonte de tempo; incluir testes automatizados para impedir o deslocamento de 3 horas.
4. **Irrigação orientada a dados ambientais:** integrar sensor de chuva e o sensor de umidade do solo como condição de habilitação (com histerese); incorporar previsão do tempo/ET_0 para *skip* de irrigação desnecessária e ajuste dinâmico do tempo de válvula.

5. **Medição elétrica dedicada e validação metrológica:** incluir módulos de medição (p.ex., PZEM-004T/HLW8012 para CA; INA219 para CC), calibrar, estimar incerteza e comparar com o método indireto, refinando a Tabela 5.2.

6. **Alimentação mais eficiente e robusta:** substituir LDO por conversor *buck* (5 V→3,3 V) para aumentar eficiência e reduzir aquecimento; revisar dimensionamento térmico/ventilação e adicionar proteção a transientes.

7. **Endurecimento de segurança:** habilitar mTLS (certificados cliente), *Secure Boot*, *Flash Encryption*, OTA autenticado, rotação de credenciais e segmentação de rede (VLAN IoT) com ACLs e firewall no *host* do *broker*.

8. **Resiliência *offline* e retomada:** implementar *store-and-forward* no ESP32 para eventos/telemetria durante indisponibilidade do *broker*, além de lógica de reexecução de agendas perdidas após reconexão.

Referências Bibliográficas

GILL, K.; YANG, S.; YAO, F.; LU, X. A ZigBee-Based Home Automation System. *IEEE Transactions on Consumer Electronics*, v. 55, n. 2, p. 422--430, 2009. DOI: 10.1109/TCE.2009.5174403.

SRISKANTHAN, C.; TAN, D. T.; KARANDE, A. An Overview of Home Automation Systems. *International Journal of Computer Applications*, vol. 19, no. 2, 2011.

AL-KUWARI, M.; RAMADAN, A.; ISMAEL, Y.; AL-SUGHAIR, L.; GASTLI, A.; BENAMMAR, M. Smart-home automation using IoT-based sensing and monitoring platform. In: *2018 12th IEEE International Conference on Compatibility, Power Electronics and Power Engineering (CPE-POWERENG)*. Doha, 2018. p. 1--6. DOI: 10.1109/CPE.2018.8372548.

PRAVALIKA, V.; PRASAD, C. R. Internet of things based home monitoring and device control using ESP32. *International Journal of Recent Technology and Engineering (IJRTE)*, v. 8, n. 1S4, p. 58--62, 2019.

AGHENTA, L. O.; IQBAL, M. T. Low-Cost, Open Source IoT-Based SCADA System Design Using Thinger.IO and ESP32 Thing. *Electronics*, v. 8, n. 8, art. 822, 2019. DOI: 10.3390/electronics8080822.

LEKIĆ, M.; GARDAŠEVIĆ, G. IoT sensor integration to Node-RED platform. In: *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*. East Sarajevo: IEEE, 2018. p. 1--5. DOI: 10.1109/INFOTEH.2018.8345544.

SAJID, A.; ABBAS, H.; SALEEM, K. Cloud-assisted IoT-based SCADA systems security: a review of the state of the art and future challenges. *IEEE Access*, v. 4, p. 1375--1384, 2016. DOI: 10.1109/ACCESS.2016.2549047.

BAHGA, Arshdeep; MADISETTI, Vijay. *Internet of Things: A Hands-On Approach*. Hyderabad: Universities Press, 2015. ISBN 978-8173719547.

HERON OF ALEXANDRIA. *The Pneumatics of Hero of Alexandria: From the Original Greek*. Translated by Joseph George Greenwood; edited by Bennet Woodcroft. London: Taylor, Walton and Maberly, 1851.

AL-JAZARĪ, Ibn al-Razzāz. *The Book of Knowledge of Ingenious Mechanical Devices (Kitāb fī marifat al-hiyal al-handasiyya)*. Translated and annotated by Donald R. Hill. Dordrecht: D. Reidel Publishing Company, 1974.

COLUMBIA UNIVERSITY. The Jacquard Loom. Computing History, Columbia University, s.d. Disponível em: <https://www.columbia.edu/cu/computinghistory/jacquard.html>. Acesso em: 19 nov. 2025.

BENNETT, Stuart. *A History of Control Engineering 1800--1930*. London: Peter Peregrinus Ltd., 1979.

DEVOL, George C. Programmed Article Transfer. U.S. Patent 2,988,237, 13 jun. 1961.

NOF, Shimon Y. (ed.). *Handbook of Industrial Robotics*. 2. ed. New York: John Wiley & Sons, 1999.

Espressif Systems. *ESP32 Series Datasheet, Version 3.5*. 2021. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>

TANENBAUM, Andrew S.; WETHERALL, David J. *Redes de Computadores*. 5. ed. São Paulo: Pearson, 2011.

APÊNDICE A -- Firmware e página web do nó urbano (ESP32 01)

Este apêndice apresenta o código completo do nó urbano (ESP32 01), incluindo o firmware responsável pela conexão Wi-Fi, comunicação MQTT, controle das lâmpadas e servidor web, bem como a folha de estilos CSS utilizada na interface HTTP embarcada.

Firmware principal do ESP32 01

Listing 1: Firmware principal do nó urbano (ESP32 01)

```
// ========================================================================
// --- Bibliotecas Auxiliares ---
#include <WiFi.h> //inclui biblioteca WiFi
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
#include <Wire.h>
#include <RTClib.h>

// ========================================================================
// Broker
const char* mqtt_host = "179.145.53.227";
const uint16_t mqtt_port = 8883;
unsigned long lastWiFiAttempt = 0;

const char* mqtt_user = "gabriel";
const char* mqtt_pass = "...";

static const char mqtt_ca_cert_pem[] PROGMEM = R"PEM(
-----BEGIN CERTIFICATE-----
MIIDsTCCApmgAwIBAgIUVnoWnkW08
.
.
.
```

```
.
J1cwAsIhrfKNc9QaSMboHEz3knK6822jEFTCDgH4+nRD6ROD2A==
-----END CERTIFICATE-----
)PEM";

WiFiClientSecure wifimqttTLS;
PubSubClient mqtt(wifimqttTLS);

// ============================================================================
// Topicos de comando (subscribe)
const char* TOPIC_L1_CMD = "tcc/esp01/casa/l1";
const char* TOPIC_L2_CMD = "tcc/esp01/casa/l2";
const char* TOPIC_L3_CMD = "tcc/esp01/casa/l3";
const char* TOPIC_AUTO_CMD = "tcc/esp01/casa/auto";

RTC_DS3231 rtc;
// ============================================================================
// Tpicos de estado (publish)
const char* TOPIC_L1_STATE = "tcc/esp01/casa/state/l1";
const char* TOPIC_L2_STATE = "tcc/esp01/casa/state/l2";
const char* TOPIC_L3_STATE = "tcc/esp01/casa/state/l3";
const char* TOPIC_AUTO_STATE = "tcc/esp01/casa/state/auto";


// ============================================================================
// --- Mapeamento de Hardware ---
#define LAMP1 16
#define LAMP2 17
#define LAMP3 18

#define BOTAO1 4
#define BOTAO2 5
#define BOTAO3 19
#define BOTAO_AUTO 23 // Botao para ativar/desativar o modo automatico

#define LED_MODO_AUTO 27 // LED que indica se o modo automatico esta ativado

#define RTC_EM_UTC true

// ============================================================================
// --- Constantes Auxiliares ---
```

```
65 const char *ssid = "Gabriel"; //atribuir nome da rede WiFi
66 const char *password = "senhadowifi123"; //atribuir senha da rede
67
68 // ========================================================================
69 // --- Objetos ---
70 WiFiServer server(80); //define a porta que o servidor ir utilizar
71
72 // ========================================================================
73 // --- Prottipo das Funes ---
74 void relay_wifi(); //function para gerar web server e controlar os rels
75
76 // ========================================================================
77 // --- Variveis Globais ---
78 String header;
79
80 //manual
81 bool estadoLamp1 = false;
82 bool estadoLamp2 = false;
83 bool estadoLamp3 = false;
84
85
86 // Auxiliar variables to store the current output state
87 bool lamp1_status = false;
88 bool lamp2_status = false;
89 bool lamp3_status = false;
90 bool auto_status = false;
91
92 bool agendamentoAtivo = false; // espelha se "as lmpadas deveriam estar ligadas"
       ↪ pela janela de tempo
93 bool ultimoAgendamento = false; // para detectar transies
94
95 // Variveis de controle dos botes
96 static bool lastBotao1 = HIGH, lastBotao2 = HIGH, lastBotao3 = HIGH;
97 static unsigned long lastDebounce1 = 0, lastDebounce2 = 0, lastDebounce3 = 0;
98 const unsigned long debounce = 200;
99
100 // Current time
101 unsigned long tempoAtual1 = 0;
102 // Previous time
103 unsigned long tempogravado = 0;
104 // Define timeout time in milliseconds (example: 2000ms = 2s)
```

```
105 const long timeoutTime = 2000;
106
107 // ============================================================================
108
109 bool modoAutomatico = false; // flag do modo automtico
110
111 unsigned long debounceDelay = 10;
112 unsigned long lastPressAuto = 0;
113
114 // Controle do ciclo automtico
115 unsigned long tempoAtual = 0;
116 unsigned long tempoCiclo = 0;
117 bool ledAutoLigado = false;
118
119 // ============================================================================
120 //RTC-DS3231
121
122 static DateTime toBRT(const DateTime& t) {
123   if (RTC_EM_UTC) return t - TimeSpan(0, 3, 0, 0);
124   return t;
125 }
126
127 static bool dentroJanela_BRT_1830a0500() {
128   DateTime nowBRT = toBRT(rtc.now());
129   int h = nowBRT.hour();
130   int m = nowBRT.minute();
131   int s = nowBRT.second();
132
133  // Se for entre 18:30 (inclusive) e 23:59:59 ou entre 00:00 e 04:59:59
134   if (h > 18) {
135     // qualquer hora depois de 18:00 j  automaticamente dentro
136     return true;
137   }
138   if (h == 18 && m >= 30) {
139     // exatamente das 18:30 em diante
140     return true;
141   }
142   // agora trata o intervalo aps a meia-noite
143   if (h < 5) {
144     return true;
145   }
```

```
146   // se estivermos exatamente h==5, no inclumos minuto 0
147   // normal h<5 j cobre at 4:59
148   return false;
149 }
150
151 // ========================================================================
152
153 void setup() {
154   Serial.begin(115200); //inicializa Serial em 115200 baud rate
155
156   pinMode(LAMP1, OUTPUT);
157   pinMode(LAMP2, OUTPUT);
158   pinMode(LAMP3, OUTPUT);
159   pinMode(LED_MODO_AUTO, OUTPUT);
160
161   pinMode(BOTAO1, INPUT_PULLUP);
162   pinMode(BOTAO2, INPUT_PULLUP);
163   pinMode(BOTAO3, INPUT_PULLUP);
164   pinMode(BOTAO_AUTO, INPUT_PULLUP);
165
166   Wire.begin(21, 22);
167   if (!rtc.begin()) {
168     Serial.println("ERRO: DS3231 no encontrado (0x68). Confira fiao.");
169   }
170
171   rtc.adjust(DateTime(2025, 10, 25, 10, 48, 00));
172
173   DateTime now = rtc.now(); // leitura cannica da RTClib
174   if (now.year() < 2020) {
175     Serial.println("RTC invalido; ajuste uma vez com rtc.adjust(...)");
176     //rtc.adjust(DateTime(2025, 10, 6, 10, 26, 00)); // exemplo de ajuste pontual
177   }
178
179   // Sincroniza o estado inicial do modo automtico com a janela 1011 BRT
180   ultimoAgendamento = dentroJanela_BRT_1830a0500();
181
182   digitalWrite(LAMP1, LOW);
183   digitalWrite(LAMP2, LOW);
184   digitalWrite(LAMP3, LOW);
185   digitalWrite(LED_MODO_AUTO, LOW);
186
```

```
187   Serial.println(); //
188     Serial.print("Conectando-se a "); //
189     Serial.println(ssid); //
190     WiFi.begin(ssid, password); //inicializa WiFi, passando o nome da rede e a
          ↪ senha
191
192     while(WiFi.status() != WL_CONNECTED) //aguarda conexo (WL_CONNECTED  uma
          ↪ constante que indica sucesso na conexo)
193     {
194       delay(741); //
195       Serial.print("."); //vai imprimindo pontos at realizar a conexo...
196     }
197
198     Serial.println(""); //mostra WiFi conectada
199     Serial.println("WiFi conectada"); //
200     Serial.println("Endereo de IP: "); //
201     Serial.println(WiFi.localIP()); //mostra o endereco IP
202
203     server.begin(); //inicializa o servidor web
204
205
206 // --- MQTT
207 wifimqttTLS.setCACert(mqtt_ca_cert_pem); //carrega o certificado CA para
      ↪ WIFIClientSecure
208 mqtt.setServer(mqtt_host, mqtt_port);
209 mqtt.setCallback(mqttCallback);
210 mqtt.setKeepAlive(60);
211 mqtt.setSocketTimeout(20);
212 mqttReconnect(); // faz a primeira conexo e assina os tpicos
213 publishState(); // publica o estado atual com retain
214 }
215
216 // ========================================================================
217 // --- MQTT: publicar estados com retain = true ---
218 void publishState() {
219   mqtt.publish(TOPIC_L1_STATE, estadoLamp1 ? "on" : "off", true);
220   mqtt.publish(TOPIC_L2_STATE, estadoLamp2 ? "on" : "off", true);
221   mqtt.publish(TOPIC_L3_STATE, estadoLamp3 ? "on" : "off", true);
222   mqtt.publish(TOPIC_AUTO_STATE, modoAutomatico ? "on" : "off", true);
223 }
224
```

```
225 // --- MQTT: callback de mensagens recebidas ---
226 void mqttCallback(char* topic, byte* payload, unsigned int length) {
227   String msg;
228   for (unsigned int i = 0; i < length; i++) msg += (char)payload[i];
229   msg.toLowerCase();
230 
231   bool isOn = (msg == "on" || msg == "1" || msg == "true");
232 
233   if (String(topic) == TOPIC_L1_CMD) { estadoLamp1 = isOn; lamp1_status = isOn;
      ↪ digitalWrite(LAMP1, estadoLamp1); }
234   else if (String(topic) == TOPIC_L2_CMD) { estadoLamp2 = isOn; lamp2_status =
      ↪ isOn; digitalWrite(LAMP2, estadoLamp2); }
235   else if (String(topic) == TOPIC_L3_CMD) { estadoLamp3 = isOn; lamp3_status =
      ↪ isOn; digitalWrite(LAMP3, estadoLamp3); }
236   else if (String(topic) == TOPIC_AUTO_CMD) { modoAutomatico = isOn; auto_status =
      ↪ isOn; digitalWrite(LED_MODO_AUTO, modoAutomatico); }
237 
238   publishState(); // sempre que mudar algo, publica o novo estado (retain)
239 }
240 
241 
242 // --- MQTT: reconectar e refazer subscriptions ---
243 void mqttReconnect() {
244   while (!mqtt.connected()) {
245     // ClientId nico para evitar briga no broker
246     String id = "esp32-esp01-casa-" + String((uint32_t)ESP.getEfuseMac(), HEX);
247 
248     // LWT: se o ESP cair sem DISCONNECT, o broker publica "offline"
249     const char* willTopic = "tcc/esp01/casa/status";
250     const char* willPayload = "offline";
251     int willQoS = 1;
252     bool willRetain = true;
253 
254     // connect(clientId, username, password, willTopic, willQoS, willRetain,
        ↪ willMessage, cleanSession)
255     if (mqtt.connect(id.c_str(),
256                      mqtt_user, mqtt_pass,
257                      willTopic, willQoS, willRetain, willPayload, true)) {
258 
259       // Marca presena para quem assina status
260       mqtt.publish("tcc/esp01/casa/status", "online", true);
```

```
261
262       // Reassina seus comandos
263       mqtt.subscribe(TOPIC_L1_CMD);
264       mqtt.subscribe(TOPIC_L2_CMD);
265       mqtt.subscribe(TOPIC_L3_CMD);
266       mqtt.subscribe(TOPIC_AUTO_CMD);
267
268       // Reenvia os estados (retain) para sincronizar painel/SCADA
269       publishState();
270
271     } else {
272       Serial.printf("MQTT falhou, state=%d. Tentando de novo...\n", mqtt.state());
273       delay(1000);
274     }
275   }
276 }
277
278
279
280 // ========================================================================
281 void loop() {
282
283   tempoAtual = millis();
284   // --- Boto que ativa/desativa o modo automtico ---
285   if (digitalRead(BOTAO_AUTO) == LOW && tempoAtual - lastPressAuto >
        ↪ debounceDelay) {
286     modoAutomatico = !modoAutomatico;
287     lastPressAuto = tempoAtual;
288     while (digitalRead(BOTAO_AUTO) == LOW); // espera soltar
289   }
290
291   // LED indicador do modo automtico
292   digitalWrite(LED_MODO_AUTO, modoAutomatico);
293
294
295
296 if (!modoAutomatico) {
297   if (digitalRead(BOTAO1) == LOW) {
298     estadoLamp1 = true;
299   } else if (lamp1_status == false) {
300     estadoLamp1 = false;
```

```
301   }
302
303   if (digitalRead(BOTAO2) == LOW) {
304     estadoLamp2 = true;
305   } else if (lamp2_status == false) {
306     estadoLamp2 = false;
307   }
308
309   if (digitalRead(BOTAO3) == LOW) {
310     estadoLamp3 = true;
311   } else if (lamp3_status == false) {
312     estadoLamp3 = false;
313   }
314 }
315
316
317
318 else {
319
320 }
321
322
323   digitalWrite(LAMP1, estadoLamp1);
324   digitalWrite(LAMP2, estadoLamp2);
325   digitalWrite(LAMP3, estadoLamp3);
326
327 // --- CONTROLE AUTOMTICO POR JANELA HORRIA ---
328 if (modoAutomatico) {
329   agendamentoAtivo = dentroJanela_BRT_1830a0500();
330
331   // **Se dentro da janela, mesmo que no tenha mudado, ligue as luzes**
332   if (agendamentoAtivo) {
333     // no momento em que estiver na janela e modo automtico
334     estadoLamp1 = estadoLamp2 = estadoLamp3 = true;
335   } else {
336     // fora da janela, desligue
337     estadoLamp1 = estadoLamp2 = estadoLamp3 = false;
338   }
339
340   // Aplique e publique sempre que estiver no modo automtico
341   digitalWrite(LAMP1, estadoLamp1);
```

```
342   digitalWrite(LAMP2, estadoLamp2);
343   digitalWrite(LAMP3, estadoLamp3);
344   publishState();
345
346   // atualiza histrico para prximas comparaes
347   ultimoAgendamento = agendamentoAtivo;
348 } else {
349   // MODO MANUAL: seu cdigo existente
350 }
351
352
353
354
355   if (WiFi.status() != WL_CONNECTED) {
356     if (millis() - lastWiFiAttempt > 5000) {
357       WiFi.begin(ssid, password);
358       lastWiFiAttempt = millis();
359     } //chama function para controle dos rels por wifi
360     // sem Wi-Fi no adianta processar MQTT/HTTP
361     return;
362   }
363
364   if (!mqtt.connected()) {
365     mqttReconnect();
366   }
367   mqtt.loop(); // TEM que rodar o tempo todo
368
369   // TELEMETRIA: imprime no Serial a cada 30 s
370   static uint32_t tLog = 0;
371   if (millis() - tLog >= 30000) {
372     tLog = millis();
373     DateTime nowBRT = toBRT(rtc.now()); // usa function utilitria
374     Serial.printf("[BRT] %04d-%02d-%02d %02d:%02d:%02d | janela18:30-5:00=%s |
          ↪ modoAuto=%d | L1=%d L2=%d L3=%d\n",
375       nowBRT.year(), nowBRT.month(), nowBRT.day(),
376       nowBRT.hour(), nowBRT.minute(), nowBRT.second(),
377       dentroJanela_BRT_1830a0500() ? "ON" : "OFF",
378       modoAutomatico,
379       estadoLamp1, estadoLamp2, estadoLamp3
380     );
381   }
```

```
382
383  relay_wifi();
384 }
385
386 void relay_wifi()
387 {
388
389   WiFiClient httpclient = server.available(); //verifica se existe um cliente
        ↪ conectado com dados a serem transmitidos
390
391   if(httpclient) //existe um cliente?
392   {
393     tempoAtual1 = millis();
394     tempogravado = tempoAtual1; //armazena tempo atual
395     Serial.println("Novo cliente definido"); //informa por serial
396     String currentLine = ""; //string para aguardar entrada de dados do cliente
397
398     while(httpclient.connected() && tempoAtual1 - tempogravado <= timeoutTime)
          ↪ //executa enquanto cliente conectado
399     {
400       tempoAtual1 = millis(); //atualiza tempo atual
401       mqtt.loop();
402       if(httpclient.available()) //existem dados do cliente a serem lidos?
403       { //sim
404         char c = httpclient.read(); //salva em c
405         Serial.write(c); //imprime via serial
406         header += c; //acumula dados do cliente em header
407
408         if (c == '\n') // um caractere de nova linha?
409         { //sim
410
411           if (currentLine.length() == 0) //se final da mensagem...
412           {
413
414             httpclient.println("HTTP/1.1 200 OK"); //HTTP sempre inicia com este
                  ↪ cdigo de resposta
415             httpclient.println("Content-type:text/html");
416             httpclient.println(); //imprime nova linha
417
418             // Controle das Sadas do ESP32:
419             if(header.indexOf("GET /lamp1/on") >= 0) //liga Rel 1
```

```
420             {
421               lamp1_status = true; //atualiza status
422               estadoLamp1 = true; //ativa sada
423               publishState();
424             } //end if lamp1 ON
425
426             else if(header.indexOf("GET /lamp1/off") >= 0) //desliga Rel 1
427             {
428               lamp1_status = false; //atualiza status
429               estadoLamp1 = false; //desativa sada
430               publishState();
431             } //end else if lamp1 OFF
432
433             else if(header.indexOf("GET /lamp2/on") >= 0) //liga Rel 2
434             {
435               lamp2_status = true; //atualiza status
436               estadoLamp2 = true; //ativa sada
437               publishState();
438             } //end else if lamp2 ON
439
440             else if(header.indexOf("GET /lamp2/off") >= 0) //desliga Rel 2
441             {
442               lamp2_status = false; //atualiza status
443               estadoLamp2 = false; //desativa sada
444               publishState();
445             } //end if lamp2 OFF
446
447             else if(header.indexOf("GET /lamp3/on") >= 0) //liga Rel 3
448             {
449               lamp3_status = true; //atualiza status
450               estadoLamp3 = true; //ativa sada
451               publishState();
452             } //end else if lamp3 ON
453
454             else if(header.indexOf("GET /lamp3/off") >= 0) //desliga Rel 3
455             {
456               lamp3_status = false; //atualiza status
457               estadoLamp3 = false; //desativa sada
458               publishState();
459             } //end if lamp3 OFF
460
```

```
461             else if(header.indexOf("GET /auto/on") >= 0) //liga automtico
462             {
463               auto_status = true;
464               modoAutomatico = true; //atualiza status
465               digitalWrite(LED_MODO_AUTO, HIGH); //ativa sada
466               publishState();
467             } //end else if auto ON
468
469             else if(header.indexOf("GET /auto/off") >= 0) //desliga automtico
470             {
471               auto_status = false;
472               modoAutomatico = false; //atualiza status
473               digitalWrite(LED_MODO_AUTO, LOW); //desativa sada
474               publishState();
475             } //end else if auto OFF
476
477
478             //Gera a pgina HTML
479
480             httpclient.println("<!DOCTYPE html><html>");
481             httpclient.println("<head><meta name=\"viewport\"
                    ↪ content=\"width=device-width, initial-scale=1\">");
482             httpclient.println("<link rel=\"icon\" href=\"data:,\">");
483
484             httpclient.println("<style>html { font-family: Verdana; margin: 0px
                    ↪ auto; text-align: center; background-color: #00FFFF;}");
485             httpclient.println(".par1 { border: none; color: #000000; padding: 20px
                    ↪ 40px;");
486             httpclient.println("text-decoration: none; font-size: 30px; margin:
                    ↪ 5px; cursor: pointer; font-family: Tahoma;}");
487             httpclient.println(".btOn { background-color: #2fb04d; }");
488             httpclient.println(".btOff { background-color: #616161; }");
489             httpclient.println("</style>");
490             httpclient.println("<title>SERVIDOR DE ACINAMENTO</title></head>");
491
492             httpclient.println("<body><h1>SERVIDOR DE ACINAMENTO</h1>");
493
494             //Imprime status atual do lamp 1
495             httpclient.println("<p>status lamp1: " + String(lamp1_status ? "ON" :
                    ↪ "OFF") + "</p>");
496
```

```
497           //Gera o boto conforme o status do Rel 1
498           if(!lamp1_status)
499             httpclient.println("<p><a href=\"/lamp1/on\"><button class=\"par1
                  ↪ btOn\">lamp 1 turn on</button></a></p>");
500           else
501             httpclient.println("<p><a href=\"/lamp1/off\"><button class=\"par1
                  ↪ btOff\">lamp 1 turn off</button></a></p>");
502
503           //Imprime status atual do lamp 2
504           httpclient.println("<p>status lamp2: " + String(lamp2_status ? "ON" :
                ↪ "OFF") + "</p>");
505
506           //Gera o boto conforme o status do Rel 2
507           if(!lamp2_status)
508             httpclient.println("<p><a href=\"/lamp2/on\"><button class=\"par1
                  ↪ btOn\">lamp 2 turn on</button></a></p>");
509           else
510             httpclient.println("<p><a href=\"/lamp2/off\"><button class=\"par1
                  ↪ btOff\">lamp 2 turn off</button></a></p>");
511
512           //Imprime status atual do lamp 3
513           httpclient.println("<p>status lamp3: " + String(lamp3_status ? "ON" :
                ↪ "OFF") + "</p>");
514
515           //Gera o boto conforme o status do Rel 3
516           if(!lamp3_status)
517             httpclient.println("<p><a href=\"/lamp3/on\"><button class=\"par1
                  ↪ btOn\">lamp 3 turn on</button></a></p>");
518           else
519             httpclient.println("<p><a href=\"/lamp3/off\"><button class=\"par1
                  ↪ btOff\">lamp 3 turn off</button></a></p>");
520
521           //Imprime status atual do auto
522           httpclient.println("<p>status auto: " + String(auto_status ? "ON" :
                ↪ "OFF") + "</p>");
523
524           //Gera o boto conforme o status do auto
525           if(!auto_status)
526             httpclient.println("<p><a href=\"/auto/on\"><button class=\"par1
                  ↪ btOn\">auto turn on</button></a></p>");
527           else
```

```
528             httpclient.println("<p><a href=\"/auto/off\"><button class=\"par1
                    ↪ btOff\">auto turn off</button></a></p>");
529
530
531           httpclient.println("</body></html>");
532           httpclient.println();
533           break;
534         }
535
536         else currentLine = "";
537
538       } //end if c
539
540       else if (c != '\r')
541         currentLine += c; //adiciona caractere como parte da mensage
542
543
544     } //end if client.available
545
546   } //end while client.connected
547
548   header = ""; //limpa header
549
550   httpclient.stop(); //finaliza conexo
551   Serial.println("Cliente desconectado"); //
552   Serial.println(""); //
553
554 } //end if client
555
556
557 } //end relay_wifi
```

Folha de estilos CSS da página web (ESP32 01)

Listing 2: Folha de estilos CSS da interface web do ESP32 01

```
1 /* Documento CSS */
2
3 html {
4     font-family: Verdana;
5     margin: 0px auto;
```

```
    text-align: center;
    background-color: #00FFFF;
}

.par1 {
    border: none;
    color: #000000;
    padding: 20px 40px;
    text-decoration: none;
    font-size: 30px;
    margin: 5px;
    cursor: pointer;
    font-family: Tahoma;
}

.bton {
    background-color: #2fb04d;
}

.btoff {
    background-color: #616161;
}
```

Apêndice B -- Firmware do nó rural (ESP32 02)

Este apêndice apresenta os principais arquivos de firmware utilizados no nó rural (ESP32 02), organizados por módulo: Wi-Fi, temporização com o RTC DS3231, lógica geral de hardware, tarefas de agendamento, envio e recebimento de dados via MQTT e função principal da aplicação.

B.1 Módulo de Wi-Fi (componente `wifi`)

B.1.1 `CMakeLists.txt` do componente `wifi`

Listing 3: Arquivo CMakeLists.txt do componente de Wi-Fi

```
idf_component_register(
    SRCS "connect.c"
    INCLUDE_DIRS "."
    REQUIRES esp_wifi esp_netif esp_event nvs_flash freertos esp_system log
)
```

B.1.2 Arquivo `connect.h`

Listing 4: Cabeçalho do módulo de conexão Wi-Fi (connect.h)

```
#ifndef __CONNECT_H
#define __CONNECT_H

#include "esp_err.h"
#include "esp_wifi.h"

void wifi_init(void);
esp_err_t wifi_connect_sta(const char * ssid, const char * pwd, int timeout);
void wifi_disconnect(void);

```

```
11 #endif
```

B.1.3 Arquivo `connect.c`

Listing 5: Implementação do módulo de conexão Wi-Fi (`connect.c`)

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdbool.h>
4  #include "esp_log.h"
5  #include "freertos/FreeRTOS.h"
6  #include "freertos/task.h"
7  #include "esp_netif.h"
8  #include "freertos/event_groups.h"
9  #include "esp_event.h"
10 #include "connect.h"
11
12 #define TAG "WIFI"
13
14 volatile bool wifiOnline = false;
15
16 esp_netif_t *wifi_netif;
17
18 static EventGroupHandle_t wifi_events;
19 static const int CONNECTED_GOT_IP = BIT0;
20 static const int DISCONNECTED = BIT1;
21
22 char *get_wifi_err(uint8_t errcode)
23 {
24     switch (errcode)
25     {
26     case WIFI_REASON_UNSPECIFIED:
27         return "WIFI_REASON_UNSPECIFIED";
28     case WIFI_REASON_AUTH_EXPIRE:
29         return "WIFI_REASON_AUTH_EXPIRE";
30     case WIFI_REASON_AUTH_LEAVE:
31         return "WIFI_REASON_AUTH_LEAVE";
32     case WIFI_REASON_ASSOC_EXPIRE:
33         return "WIFI_REASON_ASSOC_EXPIRE";
34     case WIFI_REASON_ASSOC_TOOMANY:
35         return "WIFI_REASON_ASSOC_TOOMANY";
36     case WIFI_REASON_NOT_AUTHED:
```

```
37          return "WIFI_REASON_NOT_AUTHED";
38      case WIFI_REASON_NOT_ASSOCED:
39          return "WIFI_REASON_NOT_ASSOCED";
40      case WIFI_REASON_ASSOC_LEAVE:
41          return "WIFI_REASON_ASSOC_LEAVE";
42      case WIFI_REASON_ASSOC_NOT_AUTHED:
43          return "WIFI_REASON_ASSOC_NOT_AUTHED";
44      case WIFI_REASON_DISASSOC_PWRCAP_BAD:
45          return "WIFI_REASON_DISASSOC_PWRCAP_BAD";
46      case WIFI_REASON_DISASSOC_SUPCHAN_BAD:
47          return "WIFI_REASON_DISASSOC_SUPCHAN_BAD";
48      case WIFI_REASON_BSS_TRANSITION_DISASSOC:
49          return "WIFI_REASON_BSS_TRANSITION_DISASSOC";
50      case WIFI_REASON_IE_INVALID:
51          return "WIFI_REASON_IE_INVALID";
52      case WIFI_REASON_MIC_FAILURE:
53          return "WIFI_REASON_MIC_FAILURE";
54      case WIFI_REASON_4WAY_HANDSHAKE_TIMEOUT:
55          return "WIFI_REASON_4WAY_HANDSHAKE_TIMEOUT";
56      case WIFI_REASON_GROUP_KEY_UPDATE_TIMEOUT:
57          return "WIFI_REASON_GROUP_KEY_UPDATE_TIMEOUT";
58      case WIFI_REASON_IE_IN_4WAY_DIFFERS:
59          return "WIFI_REASON_IE_IN_4WAY_DIFFERS";
60      case WIFI_REASON_GROUP_CIPHER_INVALID:
61          return "WIFI_REASON_GROUP_CIPHER_INVALID";
62      case WIFI_REASON_PAIRWISE_CIPHER_INVALID:
63          return "WIFI_REASON_PAIRWISE_CIPHER_INVALID";
64      case WIFI_REASON_AKMP_INVALID:
65          return "WIFI_REASON_AKMP_INVALID";
66      case WIFI_REASON_UNSUPP_RSN_IE_VERSION:
67          return "WIFI_REASON_UNSUPP_RSN_IE_VERSION";
68      case WIFI_REASON_INVALID_RSN_IE_CAP:
69          return "WIFI_REASON_INVALID_RSN_IE_CAP";
70      case WIFI_REASON_802_1X_AUTH_FAILED:
71          return "WIFI_REASON_802_1X_AUTH_FAILED";
72      case WIFI_REASON_CIPHER_SUITE_REJECTED:
73          return "WIFI_REASON_CIPHER_SUITE_REJECTED";
74      case WIFI_REASON_INVALID_PMKID:
75          return "WIFI_REASON_INVALID_PMKID";
76      case WIFI_REASON_BEACON_TIMEOUT:
77          return "WIFI_REASON_BEACON_TIMEOUT";
```

```
78      case WIFI_REASON_NO_AP_FOUND:
79          return "WIFI_REASON_NO_AP_FOUND";
80      case WIFI_REASON_AUTH_FAIL:
81          return "WIFI_REASON_AUTH_FAIL";
82      case WIFI_REASON_ASSOC_FAIL:
83          return "WIFI_REASON_ASSOC_FAIL";
84      case WIFI_REASON_HANDSHAKE_TIMEOUT:
85          return "WIFI_REASON_HANDSHAKE_TIMEOUT";
86      case WIFI_REASON_CONNECTION_FAIL:
87          return "WIFI_REASON_CONNECTION_FAIL";
88      case WIFI_REASON_AP_TSF_RESET:
89          return "WIFI_REASON_AP_TSF_RESET";
90      case WIFI_REASON_ROAMING:
91          return "WIFI_REASON_ROAMING";
92      }
93      return "WIFI_REASON_UNSPECIFIED";
94  }
95
96  void wifi_event_handler(void *arg, esp_event_base_t event_base,
97                          int32_t event_id, void *event_data)
98  {
99      if (event_base == WIFI_EVENT) {
100         switch (event_id)
101         {
102         case WIFI_EVENT_STA_START:
103             ESP_LOGI(TAG, "Conectando...");
104             wifiOnline = false;
105             esp_wifi_connect();
106             break;
107         case WIFI_EVENT_STA_CONNECTED:
108             wifiOnline = false;
109             ESP_LOGI(TAG, "Conectado com sucesso...");
110             break;
111         case WIFI_EVENT_STA_DISCONNECTED:
112         {
113             wifi_event_sta_disconnected_t *disc =
114                 (wifi_event_sta_disconnected_t *)event_data;
115             wifiOnline = false;
116             char *err = get_wifi_err(disc->reason);
117             if (disc->reason != WIFI_REASON_ASSOC_LEAVE)
118                 ESP_LOGE(TAG, "Desconectado %s", err);
```

```
119             else
120                 ESP_LOGI(TAG, "Desconectado...");
121 
122             xEventGroupSetBits(wifi_events, DISCONNECTED);
123             break;
124         }
125         }
126     } else if (event_base == IP_EVENT) {
127         if (event_id == IP_EVENT_STA_GOT_IP) {
128             ESP_LOGI(TAG, "IP obtido com exito");
129             wifiOnline = true;
130             xEventGroupSetBits(wifi_events, CONNECTED_GOT_IP);
131         }
132     }
133 }
134 
135 void wifi_init(void)
136 {
137     wifiOnline = false;
138     wifi_init_config_t wifiCfg = WIFI_INIT_CONFIG_DEFAULT();
139     ESP_ERROR_CHECK(esp_netif_init());
140     ESP_ERROR_CHECK(esp_event_loop_create_default());
141     ESP_ERROR_CHECK(esp_wifi_init(&wifiCfg));
142     ESP_ERROR_CHECK(esp_event_handler_register(
143         WIFI_EVENT, ESP_EVENT_ANY_ID, wifi_event_handler, NULL));
144     ESP_ERROR_CHECK(esp_event_handler_register(
145         IP_EVENT, IP_EVENT_STA_GOT_IP, wifi_event_handler, NULL));
146     ESP_ERROR_CHECK(esp_wifi_set_storage(WIFI_STORAGE_RAM));
147 
148     wifi_events = xEventGroupCreate();
149 }
150 
151 esp_err_t wifi_connect_sta(const char *ssid, const char *pwd, int timeout)
152 {
153     wifi_netif = esp_netif_create_default_wifi_sta();
154 
155     wifi_config_t wifiCfg = {0};
156     memset(&wifiCfg, 0, sizeof(wifi_config_t));
157     strncpy((char *)wifiCfg.sta.ssid, ssid,
158             sizeof(wifiCfg.sta.ssid) - 1);
159     strncpy((char *)wifiCfg.sta.password, pwd,
```

```
160             sizeof(wifiCfg.sta.password) - 1);
161
162     esp_wifi_set_mode(WIFI_MODE_STA);
163     esp_wifi_set_config(WIFI_IF_STA, &wifiCfg);
164     esp_wifi_start();
165
166     EventBits_t evnt_result = xEventGroupWaitBits(
167         wifi_events,
168         CONNECTED_GOT_IP | DISCONNECTED,
169         pdTRUE,
170         pdFALSE,
171         pdMS_TO_TICKS(timeout)
172     );
173
174     if (evnt_result & CONNECTED_GOT_IP) {
175         return ESP_OK;
176     }
177     return ESP_FAIL;
178 }
179
180 void wifi_disconnect(void)
181 {
182     esp_wifi_disconnect();
183     esp_wifi_stop();
184 }
```

B.2 Arquivo principal `main.c` e `CMakeLists.txt` do projeto

Listing 6: Arquivo principal da aplicação (`main.c`)

```
1  #include <stdio.h>
2  #include <time.h>
3  #include "connect.h"
4  #include "esp_log.h"
5  #include "scheduler.h"
6  #include "general.h"
7  #include "MQTT.h"
8  #include "ds3231.h"
9  #include "freertos/FreeRTOS.h"
10 #include "freertos/task.h"
11 #include "freertos/event_groups.h"
```

```c
#include "freertos/queue.h"
#include "esp_event.h"
#include "nvs_flash.h"
#include "driver/gpio.h"
#include "esp_wifi.h"
#include "mqtt_cert.h"

static const char *TAG = "APP";

static esp_err_t wifi_connect(void)
{
    esp_err_t err = wifi_connect_sta(WIFI_SSID, WIFI_PASS, 10000);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "Conexao Wi-Fi falhou");
        return err;
    }
    ESP_LOGI(TAG, "Conexao Wi-Fi ativa");
    return ESP_OK;
}

static void setup(void)
{
    ESP_ERROR_CHECK(nvs_flash_init());

    esp_log_level_set("*", ESP_LOG_INFO);
    esp_log_level_set("SOIL", ESP_LOG_WARN);

    wifi_init();
    ESP_ERROR_CHECK( esp_wifi_set_ps(WIFI_PS_NONE) );
    ESP_ERROR_CHECK(wifi_connect());

    time_sync_start();

    setenv("TZ", "<+03>-3", 1);
    tzset();

    esp_err_t err = ds3231_init(I2C_NUM_0, GPIO_NUM_21,
                                GPIO_NUM_22, 100000);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "DS3231 init falhou: %s", esp_err_to_name(err));
    }
```

```
53 }
54
55 static void io_init(void) {
56     gpio_config_t io = {
57         .pin_bit_mask = (1ULL<<LAMP1_GPIO) |
58                         (1ULL<<LAMP2_GPIO) |
59                         (1ULL<<LAMP3_GPIO) |
60                         (1ULL<<IRR_GPIO),
61         .mode = GPIO_MODE_OUTPUT,
62         .pull_up_en = 0,
63         .pull_down_en = 0,
64         .intr_type = GPIO_INTR_DISABLE
65     };
66     gpio_config(&io);
67     gpio_set_level(LAMP1_GPIO, 0);
68     gpio_set_level(LAMP2_GPIO, 0);
69     gpio_set_level(LAMP3_GPIO, 0);
70     gpio_set_level(IRR_GPIO, 0);
71 }
72
73 void app_main(void)
74 {
75     setup();
76     io_init();
77
78     xTaskCreatePinnedToCore(MQTTControlTask, "MQTTControlTask",
79                             6*4096, NULL, 6, NULL, 0);
80     xTaskCreatePinnedToCore(SchedulerTask, "SchedulerTask",
81                             3*4096, NULL, 5, NULL, 0);
82     BaseType_t ok = xTaskCreate(MQTTSenderTask, "MQTTSenderTask",
83                                 5*4096, NULL, 5, &mqttTaskHandle);
84     xTaskCreate(taskTemperatureQueue, "taskTemperatureQueue",
85                 configMINIMAL_STACK_SIZE * 5, NULL, 5, NULL);
86     xTaskCreate(taskHumidityQueue, "taskHumidityQueue",
87                 configMINIMAL_STACK_SIZE * 5, NULL, 5, NULL);
88     if (ok != pdPASS) {
89         ESP_LOGE(TAG, "Falha ao criar MQTTSenderTask.");
90     }
91 }
```

Listing 7: Arquivo CMakeLists.txt do projeto principal (`main`)

```
idf_component_register(SRCS "main.c" "MQTT.c" "ds3231.c" "scheduler.c"
    INCLUDE_DIRS "."
    PRIV_REQUIRES wifi json mqtt driver lwip
    REQUIRES wifi mqtt json nvs_flash esp_event esp_adc
)
```

B.3 Módulo DS3231 (RTC e temperatura)

B.3.1 Arquivo **ds3231.h**

Listing 8: Cabeçalho do módulo DS3231

```
#pragma once
#include "driver/i2c.h"
#include <time.h>

#define DS3231_ADDR 0x68
#define DS3231_REG_SEC 0x00
#define DS3231_TEMP_MSB 0x11
#define DS3231_TEMP_LSB 0x12

esp_err_t ds3231_init(i2c_port_t port, gpio_num_t sda,
                      gpio_num_t scl, uint32_t freq_hz);
esp_err_t ds3231_get_time(struct tm *out_tm);
esp_err_t ds3231_set_time(const struct tm *in_tm);
esp_err_t ds3231_get_temperature(float *out_celsius);
```

B.3.2 Arquivo **ds3231.c**

Listing 9: Implementação do módulo DS3231

```
#include <time.h>
#include <stdbool.h>
#include "ds3231.h"
#include "esp_log.h"

static i2c_port_t s_port;

static uint8_t bcd2bin(uint8_t v) { return (v & 0x0F) + ((v >> 4) * 10); }
static uint8_t bin2bcd(uint8_t v) { return ((v / 10) << 4) | (v % 10); }

```

```
static esp_err_t i2c_wr(uint8_t reg, const uint8_t *data, size_t len) {
    i2c_cmd_handle_t cmd = i2c_cmd_link_create();
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (DS3231_ADDR<<1) | I2C_MASTER_WRITE, true);
    i2c_master_write_byte(cmd, reg, true);
    if (len) i2c_master_write(cmd, (uint8_t*)data, len, true);
    i2c_master_stop(cmd);
    esp_err_t err = i2c_master_cmd_begin(s_port, cmd, pdMS_TO_TICKS(100));
    i2c_cmd_link_delete(cmd);
    return err;
}

static esp_err_t i2c_rd(uint8_t reg, uint8_t *data, size_t len) {
    i2c_cmd_handle_t cmd = i2c_cmd_link_create();
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (DS3231_ADDR<<1) | I2C_MASTER_WRITE, true);
    i2c_master_write_byte(cmd, reg, true);
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (DS3231_ADDR<<1) | I2C_MASTER_READ, true);
    i2c_master_read(cmd, data, len, I2C_MASTER_LAST_NACK);
    i2c_master_stop(cmd);
    esp_err_t err = i2c_master_cmd_begin(s_port, cmd, pdMS_TO_TICKS(100));
    i2c_cmd_link_delete(cmd);
    return err;
}

esp_err_t ds3231_init(i2c_port_t port, gpio_num_t sda,
                      gpio_num_t scl, uint32_t freq_hz) {
    s_port = port;
    i2c_config_t cfg = {
        .mode = I2C_MODE_MASTER,
        .sda_io_num = sda,
        .scl_io_num = scl,
        .sda_pullup_en = GPIO_PULLUP_DISABLE,
        .scl_pullup_en = GPIO_PULLUP_DISABLE,
        .master.clk_speed = freq_hz ? freq_hz : 100000
    };
    ESP_ERROR_CHECK(i2c_param_config(s_port, &cfg));
    ESP_ERROR_CHECK(i2c_driver_install(s_port, I2C_MODE_MASTER,
                                       0, 0, 0));
    uint8_t ping;
```

```
52      return i2c_rd(DS3231_REG_SEC, &ping, 1);
53  }
54
55  esp_err_t ds3231_get_time(struct tm *out_tm) {
56      uint8_t b[7];
57      esp_err_t err = i2c_rd(DS3231_REG_SEC, b, sizeof(b));
58      if (err != ESP_OK) return err;
59
60      out_tm->tm_sec = bcd2bin(b[0] & 0x7F);
61      out_tm->tm_min = bcd2bin(b[1] & 0x7F);
62      uint8_t hr = b[2];
63      if (hr & 0x40) {
64          uint8_t h12 = bcd2bin(hr & 0x1F);
65          out_tm->tm_hour = (hr & 0x20) ? (h12 % 12) + 12 : (h12 % 12);
66      } else {
67          out_tm->tm_hour = bcd2bin(hr & 0x3F);
68      }
69      out_tm->tm_wday = (b[3] & 0x07) - 1;
70      out_tm->tm_mday = bcd2bin(b[4] & 0x3F);
71      out_tm->tm_mon = bcd2bin(b[5] & 0x1F) - 1;
72      out_tm->tm_year = bcd2bin(b[6]) + 100;
73      return ESP_OK;
74  }
75
76  esp_err_t ds3231_set_time(const struct tm *in_tm) {
77      uint8_t b[7];
78      b[0] = bin2bcd(in_tm->tm_sec);
79      b[1] = bin2bcd(in_tm->tm_min);
80      b[2] = bin2bcd(in_tm->tm_hour);
81      b[3] = bin2bcd(in_tm->tm_wday + 1);
82      b[4] = bin2bcd(in_tm->tm_mday);
83      b[5] = bin2bcd(in_tm->tm_mon + 1);
84      b[6] = bin2bcd(in_tm->tm_year - 100);
85      return i2c_wr(DS3231_REG_SEC, b, sizeof(b));
86  }
87
88  esp_err_t ds3231_get_temperature(float *out_celsius) {
89      uint8_t msb, lsb;
90      esp_err_t err = i2c_rd(DS3231_TEMP_MSB, &msb, 1);
91      if (err != ESP_OK) return err;
92      err = i2c_rd(DS3231_TEMP_LSB, &lsb, 1);
```

```
93      if (err != ESP_OK) return err;
94      int8_t whole = (int8_t)msb;
95      float frac = (lsb >> 6) * 0.25f;
96      *out_celsius = whole + frac;
97      return ESP_OK;
98  }
```

B.4 Arquivo `general.h`

Listing 10: Definições gerais de hardware e tópicos MQTT (`general.h`)

```
1  #ifndef __GENERAL_H
2  #define __GENERAL_H
3
4  #include <stdbool.h>
5  #include "driver/gpio.h"
6
7  #define WIFI_SSID "Wi-Fi Sitio"
8  #define WIFI_PASS "senhadowifisitio"
9
10 #define LAMP1_GPIO GPIO_NUM_16
11 #define LAMP2_GPIO GPIO_NUM_17
12 #define LAMP3_GPIO GPIO_NUM_18
13 #define IRR_GPIO GPIO_NUM_19
14
15 #define IRR_ON_MS 8000
16 #define IRR_OFF_MS 4000
17
18 #define SOIL_ADC_UNIT ADC_UNIT_1
19 #define SOIL_ADC_CHANNEL ADC_CHANNEL_6
20 #define SOIL_ATTEN ADC_ATTEN_DB_12
21 #define SOIL_BITWIDTH ADC_BITWIDTH_12
22
23 #define SOIL_RAW_DRY 1860
24 #define SOIL_RAW_WET 950
25
26 #define MQTT_NS_RURAL "tcc/esp02/rural"
27 #define TOPIC_L1_CMD_RURAL MQTT_NS_RURAL "/l1"
28 #define TOPIC_L2_CMD_RURAL MQTT_NS_RURAL "/l2"
29 #define TOPIC_L3_CMD_RURAL MQTT_NS_RURAL "/l3"
30 #define TOPIC_AUTO_CMD_RURAL MQTT_NS_RURAL "/auto"
```

```
#define TOPIC_L1_STATE_RURAL MQTT_NS_RURAL "/state/l1"
#define TOPIC_L2_STATE_RURAL MQTT_NS_RURAL "/state/l2"
#define TOPIC_L3_STATE_RURAL MQTT_NS_RURAL "/state/l3"
#define TOPIC_AUTO_STATE_RURAL MQTT_NS_RURAL "/state/auto"

#define TOPIC_IRR_CMD_RURAL MQTT_NS_RURAL "/irrig"
#define TOPIC_IRR_STATE_RURAL MQTT_NS_RURAL "/state/irrig"
#define TOPIC_IRR_AUTO_CMD_RURAL MQTT_NS_RURAL "/irr_auto"
#define TOPIC_IRR_AUTO_STATE_RURAL MQTT_NS_RURAL "/state/irr_auto"

extern volatile bool wifiOnline;

#endif
```

B.5 Protocolo MQTT

B.5.1 Arquivo `MQTT.h`

Listing 11: Cabeçalho do módulo MQTT (`MQTT.h`)

```
#pragma once

#include "freertos/FreeRTOS.h"
#include "freertos/event_groups.h"
#include "freertos/task.h"
#include <math.h>
#include <time.h>

#define SensorQueueLength 100
#define generalDataQueueLength 10

typedef struct t_MqttQueueFloat {
    char tag[30];
    char local[30];
    float val;
    long long int timestamp;
} MqttQueueFloat_t;

typedef struct t_SysDataFloat {
    char local[30];
    float val;
```

```
22      long long int timestamp;
23  } SysDataFloat_t;
24
25  extern TaskHandle_t mqttTaskHandle;
26  extern bool lamp1_state, lamp2_state, lamp3_state, auto_mode;
27  extern bool irrig_state, auto_irrig;
28
29  #define NETWORK_CONNECTED BIT1
30  #define MQTT_CONNECTED BIT2
31  #define MQTT_PUBLISHED BIT3
32  #define MQTT_SUBSCRIBED BIT4
33  #define MQTT_ERROR BIT5
34
35  extern int humidityIndexProcess;
36  extern int temperatureIndexProcess;
37
38  extern MqttQueueFloat_t generalDataQueue[generalDataQueueLength];
39  extern SysDataFloat_t humidityData[SensorQueueLength];
40  extern SysDataFloat_t temperatureData[SensorQueueLength];
41  extern SysDataFloat_t subscribedData;
42
43  void time_sync_start(void);
44  void epoch_to_iso8601_utc(time_t t, char out[21]);
45  void MQTTSenderTask(void *args);
46  float convertData(char data[], int lenght);
47  void getSubscribed(void);
48  int storeFloatQueue(float data, char local[], SysDataFloat_t *internalData);
49  void taskHumidityQueue(void *args);
50  void taskTemperatureQueue(void *args);
51  void MQTTControlTask(void *args);
52  void publish_state_all(void);
53  void IrrigationAutoTask(void *args);
54  void publish_irrig_state(void);
```

B.5.2 Arquivo `MQTT.c`

Listing 12: Implementação do módulo MQTT do nó rural (`MQTT.c`).

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <time.h>
4  #include <stdlib.h>
```

```c
#include "MQTT.h"
#include "connect.h"
#include "mqtt_client.h"
#include "cJSON.h"
#include "general.h"
#include "esp_log.h"
#include "driver/gpio.h"
#include "esp_adc/adc_oneshot.h"
#include "esp_sntp.h"
#include "ds3231.h"
#include "mqtt_cert.h"

static adc_oneshot_unit_handle_t s_adc = NULL;
static void set_lamp(gpio_num_t gpio, bool on);

int humidityIndexProcess = 0;
int temperatureIndexProcess = 0;

float retSubscribedValue = 0;
static bool COMSTATUS = 0;
bool dataToSend = 0;
bool humSended = 0;
bool tmpSended = 0;
bool lamp1_state=false, lamp2_state=false, lamp3_state=false, auto_mode=false;
bool irrig_state=false, auto_irrig=false;

static long long now_epoch_ds3231(void);


// --- SNTP helpers ---
void time_sync_start(void) {
    esp_sntp_setoperatingmode(ESP_SNTP_OPMODE_POLL);
    esp_sntp_setservername(0, "pool.ntp.org");
    esp_sntp_setservername(1, "time.google.com");
    sntp_set_sync_mode(SNTP_SYNC_MODE_SMOOTH);
    esp_sntp_init();
    // Espera simples (mx ~5s) pela primeira sync
    for (int i = 0; i < 100 && esp_sntp_get_sync_status() == SNTP_SYNC_STATUS_RESET
        ↪ ; ++i) {
        vTaskDelay(pdMS_TO_TICKS(100));
    }
```

```
45 }
46
47 void epoch_to_iso8601_utc(time_t t, char out[21]) {
48     struct tm tm_utc;
49     gmtime_r(&t, &tm_utc);
50     strftime(out, 21, "%Y-%m-%dT%H:%M:%SZ", &tm_utc);
51 }
52
53
54 static bool payload_is_on(const char* p, int len) {
55     // aceita on/1/true (case-insensitive)
56     if (!p || len<=0) return false;
57     if (len==1 && (p[0]=='1')) return true;
58     if (len==4 && (p[0]=='t'||p[0]=='T')) return true; // true
59     if (len==2 && (p[0]=='o'||p[0]=='O')) return true; // on
60     return false;
61 }
62
63 static esp_mqtt_client_handle_t ctrl_client = NULL;
64 MqttQueueFloat_t generalDataQueue[generalDataQueueLength] = {0};
65 SysDataFloat_t humidityData[SensorQueueLength] = {0};
66 SysDataFloat_t temperatureData[SensorQueueLength] = {0};
67 SysDataFloat_t subscribedData = {0};
68
69 #define TAG "MQTT"
70 #define BASE_TOPIC "tcc"
71
72 #define TOPIC_TEMP "tcc/esp02/rural/temp"
73 #define TOPIC_HUM "tcc/esp02/rural/hum"
74 #define TOPIC_RAW "tcc/esp02/rural/raw"
75
76 TaskHandle_t mqttTaskHandle = NULL;
77
78 void mqtt_event_handler_cb(esp_mqtt_event_handle_t event_data){
79     switch (event_data->event_id){
80         case MQTT_EVENT_CONNECTED:
81             ESP_LOGI(TAG, "MQTT CONECTADO");
82             xTaskNotify(mqttTaskHandle, MQTT_CONNECTED, eSetValueWithOverwrite);
83             break;
84         case MQTT_EVENT_DISCONNECTED:
85             ESP_LOGI(TAG, "MQTT DESCONECTADO");
```

```
86              break;
87          case MQTT_EVENT_SUBSCRIBED:
88              ESP_LOGI(TAG, "MQTT ASSINADO, msg_if=%d", event_data->msg_id);
89              break;
90          case MQTT_EVENT_UNSUBSCRIBED:
91              break;
92          case MQTT_EVENT_PUBLISHED:
93              ESP_LOGI(TAG, "MQTT PUBLICADO, msg_id=%d", event_data->msg_id);
94              xTaskNotify(mqttTaskHandle, MQTT_PUBLISHED, eSetValueWithOverwrite);
95              break;
96          case MQTT_EVENT_DATA:
97              ESP_LOGI(TAG, "DADO LIDO EM SUBSCRICAO");
98              ESP_LOGI("MQTT", "TOPICO=%.*s", event_data->topic_len, event_data->topic
                    ↪ );
99              ESP_LOGI("MQTT", "DADO=%.*s", event_data->data_len, event_data->data);
100             retSubscribedValue = convertData(event_data->data, event_data->data_len)
                    ↪ ;
101             xTaskNotify(mqttTaskHandle, MQTT_SUBSCRIBED, eSetValueWithOverwrite);
102             break;
103         case MQTT_EVENT_ERROR:
104             ESP_LOGI(TAG, "ERRO GERAL");
105             xTaskNotify(mqttTaskHandle, MQTT_ERROR, eSetValueWithOverwrite);
106             break;
107         default:
108             ESP_LOGI(TAG, "OUTRO EVENTO - id:%d", event_data->event_id);
109             break;
110     }
111 }
112
113 static void mqtt_control_event(void* handler_args, esp_event_base_t base, int32_t
        ↪ eid, void* event_data) {
114     esp_mqtt_event_handle_t e = (esp_mqtt_event_handle_t)event_data;
115     switch (eid) {
116     case MQTT_EVENT_CONNECTED:
117         esp_mqtt_client_subscribe(ctrl_client, TOPIC_L1_CMD_RURAL, 1);
118         esp_mqtt_client_subscribe(ctrl_client, TOPIC_L2_CMD_RURAL, 1);
119         esp_mqtt_client_subscribe(ctrl_client, TOPIC_L3_CMD_RURAL, 1);
120         esp_mqtt_client_subscribe(ctrl_client, TOPIC_AUTO_CMD_RURAL, 1);
121         esp_mqtt_client_subscribe(ctrl_client, TOPIC_IRR_CMD_RURAL, 1);
122         esp_mqtt_client_subscribe(ctrl_client, TOPIC_IRR_AUTO_CMD_RURAL, 1);
123         esp_mqtt_client_subscribe(ctrl_client, TOPIC_TEMP, 2);
```

```
124         esp_mqtt_client_subscribe(ctrl_client, TOPIC_HUM, 2);
125         publish_state_all(); // publica estado atual ao conectar (retain)
126         publish_irrig_state();
127         break;
128
129     case MQTT_EVENT_DATA: {
130         // topic e data NO so null-terminated
131         const char* t = e->topic; int tlen = e->topic_len;
132         const char* p = e->data; int plen = e->data_len;
133         bool is_on = payload_is_on(p, plen);
134
135         if (tlen == (int)strlen(TOPIC_TEMP) && strncmp(t, TOPIC_TEMP, tlen) == 0) {
136             ESP_LOGI(TAG, "JSON recebido em %.*s: %.*s", tlen, t, plen, p);
137             return;
138           }
139         if (tlen == (int)strlen(TOPIC_HUM) && strncmp(t, TOPIC_HUM, tlen) == 0) {
140             ESP_LOGI(TAG, "JSON recebido em %.*s: %.*s", tlen, t, plen, p);
141             return;
142           }
143
144         if (tlen == strlen(TOPIC_L1_CMD_RURAL) && strncmp(t, TOPIC_L1_CMD_RURAL,
                ↪ tlen)==0) {
145             lamp1_state = is_on; set_lamp(LAMP1_GPIO, lamp1_state);
146             esp_mqtt_client_publish(ctrl_client, TOPIC_L1_STATE_RURAL, lamp1_state?"
                    ↪ on":"off", 0, 1, true);
147         } else if (tlen == strlen(TOPIC_L2_CMD_RURAL) && strncmp(t,
                ↪ TOPIC_L2_CMD_RURAL, tlen)==0) {
148             lamp2_state = is_on; set_lamp(LAMP2_GPIO, lamp2_state);
149             esp_mqtt_client_publish(ctrl_client, TOPIC_L2_STATE_RURAL, lamp2_state?"
                    ↪ on":"off", 0, 1, true);
150         } else if (tlen == strlen(TOPIC_L3_CMD_RURAL) && strncmp(t,
                ↪ TOPIC_L3_CMD_RURAL, tlen)==0) {
151             lamp3_state = is_on; set_lamp(LAMP3_GPIO, lamp3_state);
152             esp_mqtt_client_publish(ctrl_client, TOPIC_L3_STATE_RURAL, lamp3_state?"
                    ↪ on":"off", 0, 1, true);
153         } else if (tlen == strlen(TOPIC_AUTO_CMD_RURAL) && strncmp(t,
                ↪ TOPIC_AUTO_CMD_RURAL, tlen)==0) {
154             auto_mode = is_on;
155             esp_mqtt_client_publish(ctrl_client, TOPIC_AUTO_STATE_RURAL, auto_mode?"
                    ↪ on":"off", 0, 1, true);
```

```
156         } else if (tlen == strlen(TOPIC_IRR_CMD_RURAL) && strncmp(t,
                ↪ TOPIC_IRR_CMD_RURAL, tlen)==0) {
157             irrig_state = is_on; set_lamp(IRR_GPIO, irrig_state);
158             publish_irrig_state();
159         } else if (tlen == strlen(TOPIC_IRR_AUTO_CMD_RURAL) && strncmp(t,
                ↪ TOPIC_IRR_AUTO_CMD_RURAL, tlen)==0) {
160             auto_irrig = is_on;
161             esp_mqtt_client_publish(ctrl_client, TOPIC_IRR_AUTO_STATE_RURAL,
                    ↪ auto_irrig ? "on" : "off", 0, 1, true);
162             if (!auto_irrig) {
163                 set_lamp(IRR_GPIO, irrig_state);
164                 publish_irrig_state();
165             }
166         }
167 
168         break;
169     }
170     default:
171         break;
172     }
173 }
174 
175 void MQTTControlTask(void *args){
176     const esp_mqtt_client_config_t cfg = {
177     .broker.address.uri = "mqtts://179.145.53.227:8883",
178     .credentials.client_id = "esp32-esp02-rural",
179     .credentials.username = "gabriel",
180     .credentials.authentication.password = "root123456204",
181     .broker.verification.certificate = mqtt_ca_cert_pem,
182     .session.keepalive = 30,
183     .network.disable_auto_reconnect = false
184     };
185     ctrl_client = esp_mqtt_client_init(&cfg);
186     esp_mqtt_client_register_event(ctrl_client, ESP_EVENT_ANY_ID,
            ↪ mqtt_control_event, NULL);
187     esp_mqtt_client_start(ctrl_client);
188 
189     while (1) {
190 
191         vTaskDelay(pdMS_TO_TICKS(200));
192     }
```

```
193 }
194
195 static void mqtt_event_handler(void* event_handler_arg, esp_event_base_t event_base
        ↪ , int32_t event_id, void* event_data){
196     mqtt_event_handler_cb(event_data);
197 }
198
199 static void set_lamp(gpio_num_t gpio, bool on) {
200     gpio_set_level(gpio, on ? 1 : 0);
201 }
202
203 void MQTTSender(MqttQueueFloat_t *sensorFloatReading, bool subscribe){
204     uint32_t command = 0;
205 const esp_mqtt_client_config_t mqttConfig = {
206         .broker.address.uri = "mqtts://179.145.53.227:8883",
207         .broker.verification.certificate = (const char*)mqtt_ca_cert_pem,
208         .credentials.username = "gabriel",
209         .credentials.authentication.password = "root123456204",
210         .credentials.client_id = "esp32-esp02-rural-sender",
211         .session.keepalive = 60,
212         .network.disable_auto_reconnect = false
213     };
214     esp_mqtt_client_handle_t client = NULL;
215
216     while(1){
217         char local[30];
218         char dataBuff[15];
219         char outBuff[1000];
220         char tag[30];
221         char timestamp[15];
222         xTaskNotifyWait(0,0,&command, portMAX_DELAY);
223         switch (command)
224         {
225             case NETWORK_CONNECTED:
226                 COMSTATUS = 0;
227                 client = esp_mqtt_client_init(&mqttConfig);
228                 esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID,
                        ↪ mqtt_event_handler, client);
229                 esp_mqtt_client_start(client);
230                 ESP_LOGI(TAG, "ONLINE");
231                 break;
```

```
232             case MQTT_CONNECTED:
233                 COMSTATUS = 0;
234                 if(subscribe){
235                   strcpy(local,sensorFloatReading[0].local);
236                   esp_mqtt_client_subscribe(client, local, 1);
237                 }else {
238                     // ---------- MONTA O PAYLOAD ANTIGO PARA O VISOR ----------
239                     // usa as variveis j declaradas acima (outBuff, tag, dataBuff,
                            ↪ timestamp)
240                     // zera o buffer
241                     outBuff[0] = '\0';
242                     strcat(outBuff, "{");
243 
244                     // contador para vrgula apenas entre pares vlidos
245                     int wrote = 0;
246                     for (int i = 0; i < generalDataQueueLength; i++) {
247                         if (sensorFloatReading[i].timestamp == 0) continue; // slot
                                ↪ vazio
248 
249                         // monta "TAG:VAL:TIMESTAMP"
250                         strcpy(tag, sensorFloatReading[i].tag);
251                         sprintf(dataBuff, "%.2f", sensorFloatReading[i].val);
252                         sprintf(timestamp, "%lld", sensorFloatReading[i].timestamp);
253 
254                         if (wrote > 0) strcat(outBuff, ",");
255                         strcat(outBuff, tag);
256                         strcat(outBuff, ":");
257                         strcat(outBuff, dataBuff);
258                         strcat(outBuff, ":");
259                         strcat(outBuff, timestamp);
260                         wrote++;
261                     }
262                     strcat(outBuff, "}");
263 
264                     // PUBLICA O PAYLOAD ANTIGO (para o visor do MQTTBox)
265                     esp_mqtt_client_publish(client, TOPIC_RAW, outBuff, 0 /* strlen
                            ↪ auto */, 2, false);
266 
267                     // ---------- MONTA E PUBLICA OS JSONs PARA O BACKEND ----------
268                     const MqttQueueFloat_t *hum = NULL, *tmp = NULL;
269                     for (int i = 0; i < generalDataQueueLength; i++) {
```

```
270                     if (sensorFloatReading[i].timestamp == 0) continue;
271                     if (strncmp(sensorFloatReading[i].tag, "HUM", 3) == 0) hum =
                            ↪ &sensorFloatReading[i];
272                     if (strncmp(sensorFloatReading[i].tag, "TMP", 3) == 0) tmp =
                            ↪ &sensorFloatReading[i];
273                 }
274
275                 char jsonBuf[160];
276
277                 if (tmp) {
278                     char iso_temp[21];
279                     epoch_to_iso8601_utc((time_t)tmp->timestamp, iso_temp);
280                     snprintf(jsonBuf, sizeof(jsonBuf),
281                            "{\"temperature\": %.2f, \"timestamp\": \"%s\"}", tmp
                                ↪ ->val, iso_temp);
282                     esp_mqtt_client_publish(client, TOPIC_TEMP, jsonBuf, 0, 1,
                            ↪ false);
283                 }
284
285                 if (hum) {
286                     char iso_hum[21];
287                     epoch_to_iso8601_utc((time_t)hum->timestamp, iso_hum);
288                     snprintf(jsonBuf, sizeof(jsonBuf),
289                            "{\"humidity\": %.2f, \"timestamp\": \"%s\"}", hum->
                                ↪ val, iso_hum);
290                     esp_mqtt_client_publish(client, TOPIC_HUM, jsonBuf, 0, 1,
                            ↪ false);
291                 }
292
293             }
294             break;
295         case MQTT_PUBLISHED:
296             ESP_LOGI(TAG, "PARANDO");
297         if (client) {
298                 esp_mqtt_client_stop(client);
299                 esp_mqtt_client_destroy(client);
300                 client = NULL;
301             }
302             COMSTATUS = 1;
303             return;
304         case MQTT_SUBSCRIBED:
```

```
305                 sensorFloatReading[0].val = retSubscribedValue;
306                 if (client) {
307                     ESP_ERROR_CHECK( esp_mqtt_client_stop(client) );
308                     ESP_ERROR_CHECK( esp_mqtt_client_destroy(client) );
309                     client = NULL;
310                 }
311                  COMSTATUS = 1;
312                  return;
313             case MQTT_ERROR:
314                 ESP_LOGE(TAG, "ERRO DE CONEXAO");
315                 if (client) {
316                     esp_mqtt_client_stop(client);
317                     esp_mqtt_client_destroy(client);
318                     client = NULL;
319                 }
320                 COMSTATUS = 0;
321             default:
322                 break;
323         }
324     }
325 }
326
327 void publish_state_all(void) {
328     if (!ctrl_client) return;
329     esp_mqtt_client_publish(ctrl_client, TOPIC_L1_STATE_RURAL, lamp1_state?"on":"
        ↪ off", 0, 1, true);
330     esp_mqtt_client_publish(ctrl_client, TOPIC_L2_STATE_RURAL, lamp2_state?"on":"
        ↪ off", 0, 1, true);
331     esp_mqtt_client_publish(ctrl_client, TOPIC_L3_STATE_RURAL, lamp3_state?"on":"
        ↪ off", 0, 1, true);
332     esp_mqtt_client_publish(ctrl_client, TOPIC_AUTO_STATE_RURAL, auto_mode?"on":"
        ↪ off", 0, 1, true);
333 }
334
335 void publish_irrig_state(void) {
336     if (!ctrl_client) return;
337     esp_mqtt_client_publish(ctrl_client, TOPIC_IRR_STATE_RURAL, irrig_state?"on":"
        ↪ off", 0, 1, true);
338 }
339
340 static void soil_adc_init_once(void){
```

```
341     static bool inited = false;
342     if (inited) return;
343 
344     adc_oneshot_unit_init_cfg_t unit_cfg = {
345         .unit_id = SOIL_ADC_UNIT,
346     };
347     ESP_ERROR_CHECK(adc_oneshot_new_unit(&unit_cfg, &s_adc));
348 
349     adc_oneshot_chan_cfg_t chan_cfg = {
350         .bitwidth = SOIL_BITWIDTH,
351         .atten = SOIL_ATTEN,
352     };
353     ESP_ERROR_CHECK(adc_oneshot_config_channel(s_adc, SOIL_ADC_CHANNEL, &chan_cfg))
        ↪ ;
354     inited = true;
355 }
356 
357 // Mediana simples de N leituras para reduzir rudo
358 static int read_soil_raw_multisample(void){
359     const int N = 15;
360     int v[N];
361     for (int i = 0; i < N; i++){
362         ESP_ERROR_CHECK(adc_oneshot_read(s_adc, SOIL_ADC_CHANNEL, &v[i]));
363     }
364     // insertion sort
365     for (int i = 1; i < N; i++){
366         int key = v[i], j = i - 1;
367         while (j >= 0 && v[j] > key){ v[j+1] = v[j]; j--; }
368         v[j+1] = key;
369     }
370     return v[N/2];
371 }
372 
373 // Mapeia leitura para % de umidade (0..100)
374 static float soil_percent_from_raw(int raw){
375     float dry = SOIL_RAW_DRY, wet = SOIL_RAW_WET;
376     if (dry < wet){ float t = dry; dry = wet; wet = t; } // por via das dvidas
377     float pct = (dry - raw) / (dry - wet);
378     if (pct < 0) pct = 0;
379     if (pct > 1) pct = 1;
380     return pct * 100.0f;
```

```
}


void MQTTSenderTask(void *args){
    mqttTaskHandle = xTaskGetCurrentTaskHandle();
    ESP_LOGI(TAG, "INICIADO SENDER TASK");

    while (1)
    {
        static int index = 0;
        index = 0;

        ESP_LOGI(TAG, "INICIANDO REDE WIFI");
        ESP_ERROR_CHECK(esp_wifi_start());

        while(!wifiOnline){
            vTaskDelay(20);
        }

        if (humidityIndexProcess > 0 && wifiOnline){
            strcpy(generalDataQueue[index].tag, humidityData[humidityIndexProcess
                ↪ -1].local);
            strcpy(generalDataQueue[index].local, "tcc/esp02/rural/hum");
            generalDataQueue[index].val = humidityData[humidityIndexProcess-1].val;
            generalDataQueue[index].timestamp = humidityData[humidityIndexProcess
                ↪ -1].timestamp;
            index++;
            dataToSend = 1;
            humSended = 1;
        }

        if (temperatureIndexProcess > 0 && wifiOnline){
            strcpy(generalDataQueue[index].tag, temperatureData[
                ↪ temperatureIndexProcess-1].local);
            strcpy(generalDataQueue[index].local, "tcc/esp02/rural/temp");
            generalDataQueue[index].val = temperatureData[temperatureIndexProcess
                ↪ -1].val;
            generalDataQueue[index].timestamp = temperatureData[
                ↪ temperatureIndexProcess-1].timestamp;
            index++;
            dataToSend = 1;
```

```
417             tmpSended = 1;
418         }
419         vTaskDelay(pdMS_TO_TICKS(30000));
420 
421          if(dataToSend == 1){
422             xTaskNotify(mqttTaskHandle, NETWORK_CONNECTED, eSetValueWithOverwrite);
423             MQTTSender(generalDataQueue,0);
424             dataToSend=0;
425         }
426 
427         if (COMSTATUS == 1){
428             if(humidityIndexProcess > 0 && humSended ==1){
429                 humidityData[humidityIndexProcess-1].timestamp = 0;
430                 humidityData[humidityIndexProcess-1].val = 0;
431                 humidityIndexProcess--;
432                 humSended = 0;
433             }
434 
435             if(temperatureIndexProcess > 0 && tmpSended ==1){
436                 temperatureData[temperatureIndexProcess-1].timestamp = 0;
437                 temperatureData[temperatureIndexProcess-1].val = 0;
438                 temperatureIndexProcess--;
439                 tmpSended = 0;
440             }
441 
442             for(int i = 0; i<generalDataQueueLength; i++){
443                 strcpy(generalDataQueue[i].tag, " ");
444                 strcpy(generalDataQueue[i].local, " ");
445                 generalDataQueue[i].val=0;
446                 generalDataQueue[i].timestamp=0;
447             }
448         }
449         vTaskDelay(pdMS_TO_TICKS(30000));
450     }
451 }
452 
453 void getSubscribed(){
454     MqttQueueFloat_t temp[1];
455 
456     strcpy(subscribedData.local, "tcc/esp02/rural/temp");
457     xTaskNotify(mqttTaskHandle, NETWORK_CONNECTED, eSetValueWithOverwrite);
```

```
458     strcpy(temp[0].tag, " ");
459     strcpy(temp[0].local, " ");
460     temp[0].val = 0;
461     temp[0].timestamp=0;
462     strcpy(temp[0].local, subscribedData.local);
463     MQTTSender(temp, 1);
464     subscribedData.val = temp[0].val;
465     printf("Valor lido: %f \n\r", subscribedData.val);
466 }
467 
468 int storeFloatQueue(float data, char local[], SysDataFloat_t *internalData){
469     int indexLocal = 0;
470 
471     if (internalData[SensorQueueLength-1].timestamp != 0){
472         for(int i=0; i<(SensorQueueLength-1); i++){
473             internalData[i].val = internalData[i+1].val;
474             internalData[i].timestamp = internalData[i+1].timestamp;
475             internalData[i].timestamp = internalData[i+1].timestamp;
476             strcpy(internalData[i].local,internalData[i+1].local);
477             indexLocal = SensorQueueLength-1;
478         }
479     }else{
480         for(int i=0; i<SensorQueueLength;i++){
481             if(internalData[i].timestamp == 0) {
482                 indexLocal = i;
483                 break;
484             }
485         }
486 
487     }
488     internalData[indexLocal].val = data;
489     internalData[indexLocal].timestamp = now_epoch_ds3231();
490     strcpy(internalData[indexLocal].local,local);
491 
492     return(indexLocal+1);
493 }
494 
495 void taskHumidityQueue(void *args){
496     humidityIndexProcess = 0;
497     soil_adc_init_once();
498 
```

```
499     while(1){
500         int raw = read_soil_raw_multisample();
501         float hum_pct = soil_percent_from_raw(raw); // 0..100 %
502 
503         humidityIndexProcess = storeFloatQueue(hum_pct, "HUM01", humidityData);
504         vTaskDelay(pdMS_TO_TICKS(2000));
505     }
506 }
507 
508 void taskTemperatureQueue(void *args){
509     temperatureIndexProcess = 0;
510 
511     while(1){
512         float tC = 0.0f;
513         if (ds3231_get_temperature(&tC) == ESP_OK) {
514             temperatureIndexProcess = storeFloatQueue(tC, "TMP01", temperatureData);
515         }
516         vTaskDelay(pdMS_TO_TICKS(2000)); // RTC atualiza
517     }
518 }
519 
520 
521 
522 float convertData(char data[], int lenght){
523     bool negative = 0;
524     bool dot = 0;
525     int factorMult = 0;
526     int factorDiv = 0;
527     int factorDivAux = 0;
528     float valor = 0;
529 
530 
531   for(int i=9; i<lenght-1; i++){
532     if(data[i]=='.'){
533       dot=1;
534     }
535 
536     if (data[i] =='-'){
537       negative = 1;
538     }else{
539         if (dot==0){
```

```
540           factorMult++;
541         }else{
542           factorDiv++;
543         }
544       }
545     }
546     //Se no achou o . ento precisa subtrair um para ajustar o comprimento
547     if(dot==0){
548       factorMult--;
549     }
550     dot = 0;
551
552     factorDivAux = factorDiv-1;
553
554     for(int i=9; i<lenght-1; i++){
555       if(data[i]=='.'){
556         dot=1;
557       }else{
558         if (data[i] > 47 && data[i] < 58){
559           if(dot == 0){
560             factorMult--;
561              valor = (float)(((int)data[i]) -48) * pow(10,factorMult) + valor;
562           }
563           if(dot == 1){
564              valor = (float)(((int)data[i]-48)) / pow(10,factorDiv-factorDivAux) +
                   ↪ valor;
565              factorDivAux--;
566           }
567         }
568       }
569     }
570
571     if(negative){
572       valor = -valor;
573     }
574
575     return valor;
576   }
577
578   static long long now_epoch_ds3231(void){
579         struct tm t;
```

```
580         if (ds3231_get_time(&t) == ESP_OK) {
581             // TZ=UTC0 -> mktime() devolve epoch UTC
582             return (long long) mktime(&t);
583         }
584      // fallback: se o RTC falhar, devolve 0 (ou um erro)
585     return (0);
586 }
```

B.5.3 Arquivo `mqtt_cert.h`

Listing 13: Certificado da autoridade certificadora (`mqtt_cert.h`)

```
1  #ifndef MQTT_CERT_H
2  #define MQTT_CERT_H
3
4  static const char mqtt_ca_cert_pem[] =
5      "-----BEGIN CERTIFICATE-----\n"
6      "MIIDsTCCApmgAwIBAgIUVnoWnkW08LN+bo8boufYy4Ap8agwDQYJKoZIhvcNAQEL\n"
7      /* linhas intermediarias do certificado omitidas para brevidade */
8      "J1cwAsIhrfKNc9QaSMboHEz3knK6822jEFTCDgH4+nRD6ROD2A==\n"
9      "-----END CERTIFICATE-----\n";
10
11 #endif // MQTT_CERT_H
```

B6 Arquivo `scheduler.h` e `scheduler.c`

Listing 14: Cabeçalho do agendador de tarefas (`scheduler.h`)

```
1  #pragma once
2  #ifdef __cplusplus
3  extern "C" {
4  #endif
5
6  void SchedulerTask(void *args);
7
8  #ifdef __cplusplus
9  }
10 #endif
```

Listing 15: Implementação do agendador de iluminação e irrigação (`scheduler.c`)

```c
#include <stdbool.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "esp_log.h"
#include "ds3231.h"
#include "general.h"
#include "MQTT.h"

static const char *TAG = "SCHED";

static int minutes_since_midnight(void) {
    struct tm t = {0};
    if (ds3231_get_time(&t) != ESP_OK) {
        ESP_LOGW(TAG, "RTC falhou; assumindo 00:00");
        return 0;
    }
    return t.tm_hour * 60 + t.tm_min;
}

static bool in_window(int now_min, int start_min, int end_min) {
    if (start_min <= end_min) {
        return (now_min >= start_min) && (now_min < end_min);
    } else {
        return (now_min >= start_min) || (now_min < end_min);
    }
}

static void set_all_lamps(bool on) {
    lamp1_state = lamp2_state = lamp3_state = on;
    gpio_set_level(LAMP1_GPIO, on);
    gpio_set_level(LAMP2_GPIO, on);
    gpio_set_level(LAMP3_GPIO, on);
    publish_state_all();
}

static void set_irrig(bool on) {
    irrig_state = on;
    gpio_set_level(IRR_GPIO, on);
    publish_irrig_state();
}
```

```
42
43 void SchedulerTask(void *args) {
44     const TickType_t period = pdMS_TO_TICKS(1000);
45
46     const int IRR_START = 9*60 + 0;
47     const int IRR_END = 9*60 + 10;
48     const int LMP_START = 18*60 + 30;
49     const int LMP_END = 5*60;
50
51     for (;;) {
52         int now = minutes_since_midnight();
53
54         if (auto_irrig) {
55             bool want_irrig = in_window(now, IRR_START, IRR_END);
56             if (want_irrig != irrig_state) {
57                 set_irrig(want_irrig);
58             }
59         }
60
61         if (auto_mode) {
62             bool want_lamps = in_window(now, LMP_START, LMP_END);
63             bool group_on = lamp1_state || lamp2_state || lamp3_state;
64             if (want_lamps != group_on) {
65                 set_all_lamps(want_lamps);
66             }
67         }
68
69         vTaskDelay(period);
70     }
71 }
```

APÊNDICE C -- Códigos das funções Node-RED

Este apêndice apresenta os códigos completos dos nós *function* utilizados no Node-RED, descritos na Seção 3.5. As listagens a seguir correspondem, respectivamente, às funções *norm temp*, *norm hum*, *pair (temp+hum)* e *function final*, associadas às subseções 3.5.11, 3.5.12, 3.5.13 e 3.5.14 do texto principal.

Função *norm temp*

Listing 16: Função de normalização da temperatura (`norm temp`)

```
// Normaliza TEMPERATURA -> msg.topic='temp', msg.payload={val, ts}
let p = msg.payload;
if (typeof p === 'string') {
  try {
    p = JSON.parse(p);
  } catch {
    // deixa como string
  }
}

const val = Number(
  (p && (p.temperature ?? p.temp)) ?? (typeof p === 'number' ? p : NaN)
);
const ts = (p && (p.timestamp ?? p.ts)) || new Date().toISOString();

if (!Number.isFinite(val)) {
  node.error('Temperatura invlida', msg);
  return null;
}

msg.topic = 'temp';
msg.payload = { val, ts: String(ts) };
```

```
return msg;
```

Função *norm hum*

Listing 17: Função de normalização da umidade (`norm hum`)

```
// Normaliza UMIDADE -> msg.topic='hum', msg.payload={val, ts}
let p = msg.payload;
if (typeof p === 'string') {
  try {
    p = JSON.parse(p);
  } catch {
    // deixa como string
  }
}

const val = Number(
  (p && (p.humidity ?? p.hum)) ?? (typeof p === 'number' ? p : NaN)
);
const ts = (p && (p.timestamp ?? p.ts)) || new Date().toISOString();

if (!Number.isFinite(val)) {
  node.error('Umidade invlida', msg);
  return null;
}

msg.topic = 'hum';
msg.payload = { val, ts: String(ts) };
return msg;
```

Função *pair (temp+hum)*

Listing 18: Função de pareamento de temperatura e umidade (`pair (temp+hum)`)

```
const windowMs = 15_000; // 15s

let t = context.get('t') || null;
let h = context.get('h') || null;

if (msg.topic === 'temp') {
```

```
  t = { ...msg.payload };
  context.set('t', t);
} else if (msg.topic === 'hum') {
  h = { ...msg.payload };
  context.set('h', h);
} else {
  return null;
}

if (t && h) {
  const timeT = Date.parse(t.ts);
  const timeH = Date.parse(h.ts);

  if (!Number.isNaN(timeT) && !Number.isNaN(timeH)) {
    const diff = Math.abs(timeT - timeH);

    if (diff <= windowMs) {
      const out = { payload: { temp: t, hum: h } };
      context.set('t', null);
      context.set('h', null);
      return out;
    }

    // opcional: expirar leitura velha para no empacar
    const now = Date.now();
    if (now - timeT > windowMs) context.set('t', null);
    if (now - timeH > windowMs) context.set('h', null);
  }
}
return null;
```

Função *function final*

Listing 19: Função de montagem do comando SQL para inserção em `sensorData` (`function final`)

```
// msg.payload = { temp?:{val,ts}, hum?:{val,ts} }
const p = msg.payload || {};
const t = p.temp;
const h = p.hum;

```

```
// escolhe um ts: prioriza o da temp; seno o da hum
const tsISO = (t && t.ts) || (h && h.ts);

function toMySQL(s) {
  const d = new Date(s);
  return isNaN(d.getTime())
    ? null
    : d.toISOString().slice(0, 19).replace('T', ' ');
}

const when = toMySQL(tsISO);
if (!when) {
  node.error('timestamp invlido', msg);
  return null;
}

msg.topic = 'INSERT INTO `sensorData`
    ↪ (`sensor`,`local`,`temperature`,`humidity`,`timestamp`) VALUES (?,?,?,?,?)';
msg.payload = [
  'combo', // ou 'temperatura+umidade'
  'LAB', // seu local
  t ? t.val : null, // temperature
  h ? h.val : null, // humidity
  when // YYYY-MM-DD HH:MM:SS
];
return msg;
```

APÊNDICE D -- Pipeline MySQL → SCADA-LTS

Este apêndice apresenta os códigos em SQL utilizados na configuração do *pipeline* entre o banco de dados MySQL (tabela `sensorData`) e o SCADA-LTS, conforme descrito na Seção 3.5.15. As listagens a seguir correspondem, respectivamente, à criação/verificação do banco e da tabela (*passo 2*), e às consultas associadas aos *Data Sources* `hum_rural` (*passo 4*) e `temp_rural` (*passo 5*).

Criação/verificação do banco **SCADA** e da tabela **sensorData**

Listing 20: Criação/verificação do banco `SCADA` e da tabela `sensorData`

```sql
CREATE DATABASE IF NOT EXISTS SCADA
  DEFAULT CHARACTER SET utf8mb4
  COLLATE utf8mb4_0900_ai_ci;
USE SCADA;

CREATE TABLE IF NOT EXISTS sensorData (
  id INT AUTO_INCREMENT PRIMARY KEY,
  sensor VARCHAR(30) NOT NULL,
  `local` VARCHAR(50) NOT NULL,
  temperature DECIMAL(5,2) NULL,
  humidity DECIMAL(5,2) NULL,
  `timestamp` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB;
```

Data Source **hum_rural**: consulta e ponto de medição

Listing 21: *Select Statement* do `hum_rural`

```sql
SELECT 'hum' AS pointId,
       humidity AS valueCol,
```

```
3       `timestamp` AS timeCol
4 FROM sensorData
5 ORDER BY `timestamp` DESC, id DESC
6 LIMIT 1;
```

Data Source `temp_rural`: consulta e ponto de medição

Listing 22: *Select Statement* do `temp_rural`

```
1 SELECT
2     'temp' AS pointId,
3     temperature AS valueCol,
4     `timestamp` AS timeCol
5 FROM sensorData
6 ORDER BY `timestamp` DESC, id DESC
7 LIMIT 1;
```