



INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS
CAMPUS ITUMBIARA
BACHARELADO EM ENGENHARIA ELÉTRICA

KELDSO ALVES LOPES

SISTEMA IOT PARA SUPERVISÃO E CONTROLE DE INVERSORES COM
APLICAÇÃO EM MICRORREDES

ITUMBIARA

2025



INSTITUTO FEDERAL
Goiás

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
SISTEMA INTEGRADO DE BIBLIOTECAS

TERMO DE AUTORIZAÇÃO PARA DISPONIBILIZAÇÃO NO REPOSITÓRIO DIGITAL DO IFG - ReDi IFG

Com base no disposto na Lei Federal nº 9.610/98, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia de Goiás, a disponibilizar gratuitamente o documento no Repositório Digital (ReDi IFG), sem ressarcimento de direitos autorais, conforme permissão assinada abaixo, em formato digital para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IFG.

Identificação da Produção Técnico-Científica

- | | |
|--|---|
| ☐ Tese | ☐ Artigo Científico |
| ☐ Dissertação | ☐ Capítulo de Livro |
| ☐ Monografia – Especialização | ☐ Livro |
| ☒ TCC - Graduação | ☐ Trabalho Apresentado em Evento |
| ☐ Produto Técnico e Educacional - Tipo: _____ | |

Nome Completo do Autor: Keldson Alves Lopes

Matrícula: 20201040070140

Título do Trabalho: Sistema IOT para Supervisão e Controle de Inversores com Aplicação em Microrredes

Autorização - Marque uma das opções

1. (X) Autorizo disponibilizar meu trabalho no Repositório Digital do IFG (acesso aberto);
2. () Autorizo disponibilizar meu trabalho no Repositório Digital do IFG somente após a data ____/____/____ (Embargo);
3. () Não autorizo disponibilizar meu trabalho no Repositório Digital do IFG (acesso restrito).

Ao indicar a opção **2 ou 3**, marque a justificativa:

- () O documento está sujeito a registro de patente.
() O documento pode vir a ser publicado como livro, capítulo de livro ou artigo.
() Outra justificativa: _____

DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA

O/A referido/a autor/a declara que:

- i. o documento é seu trabalho original, detém os direitos autorais da produção técnico-científica e não infringe os direitos de qualquer outra pessoa ou entidade;
- ii. obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autor/a, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia de Goiás os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;
- iii. cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia de Goiás.

Itumbiara-GO, 14/05/2025.

Local

Data

Documento assinado digitalmente



KELSON ALVES LOPES

Data: 19/05/2025 13:02:34-0300

Verifique em <https://validar.iti.gov.br>

Assinatura do Autor e/ou Detentor dos Direitos Autorais

KELDSO ALVES LOPES

**SISTEMA IOT PARA SUPERVISÃO E CONTROLE DE INVERSORES COM
APLICAÇÃO EM MICRORREDES**

Trabalho de Conclusão de Curso apresentado à Banca Examinadora do Curso de Bacharelado em Engenharia Elétrica, em cumprimento às exigências legais como requisito parcial à obtenção do título de Engenheiro Eletricista.

Área de Concentração do Trabalho: Engenharias, Engenharia Elétrica, Sistemas Elétricos de Potência.

Prof. Ghunter Paulo Viajante, Dr. (IFG) - Orientador

ITUMBIARA

2025

Dados Internacionais de Catalogação na Publicação (CIP)

L864s Lopes, Keldson Alves
 Sistema IoT para supervisão e controle de inversores com
 aplicação em microrredes. / Keldson Alves Lopes -- Itumbiara,
 2025.
 175p. : il.

 Trabalho de conclusão de curso (Bacharelado em Engenharia
 Elétrica) - Instituto Federal de Educação, Ciência e Tecnologia
 de Goiás, Câmpus Itumbiara, 2025.

 Orientador: Prof. Dr. Ghunter Paulo Viajante.

 1. Energia elétrica - Racionamento. 2. Microcontroladores. I.
 Título. II. Viajante, Gunter Paulo. III. Instituto Federal de
 Educação, Ciência e Tecnologia de Goiás.

CDD 333.914

Instituto Federal de Educação, Ciência e Tecnologia de Goiás
Biblioteca Maria Gabriela Pacheco Pardey / Câmpus Itumbiara
Bibliotecário: Rosiane Gonçalves de Lima CRB1/1684



INSTITUTO FEDERAL
Goiás

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS
CÂMPUS ITUMBIARA

Termo de Aprovação

Keldson Alves Lopes

Sistema IoT para supervisão e controle de inversores com aplicação em microrredes

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Engenharia Elétrica do Instituto Federal de Educação, Ciência e Tecnologia de Goiás – Campus Itumbiara, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia Elétrica.

Trabalho de Conclusão de Curso defendido e aprovado, em 13 de março de 2025, pela banca examinadora constituída pelos seguintes membros:

BANCA EXAMINADORA

Prof. Dr. Ghunter Paulo Viajante - Orientador

Instituto Federal de Goiás, Campus Itumbiara.

(Assinado Eletronicamente).

Prof. Dr. Marcelo Escobar de Oliveira - Membro interno

Instituto Federal de Goiás, Campus Itumbiara.

(Assinado Eletronicamente).

Profa. Dr. Eric Nery Chaves - membro interno

Instituto Federal de Goiás, Campus Itumbiara.

(Assinado Eletronicamente).

Itumbiara – GO

2025

Documento assinado eletronicamente por:

- **Marcelo Escobar de Oliveira**, PROFESSOR ENS BASICO TECN TECNOLOGICO, em 13/03/2025 21:46:26.
- **Eric Nery Chaves**, PROFESSOR ENS BASICO TECN TECNOLOGICO, em 13/03/2025 21:34:25.
- **Ghunter Paulo Viajante**, PROFESSOR ENS BASICO TECN TECNOLOGICO, em 13/03/2025 21:32:22.

Este documento foi emitido pelo SUAP em 24/02/2025. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse <https://suap.ifg.edu.br/autenticar-documento/> e forneça os dados abaixo:

Código Verificador: 621522

Código de Autenticação: 67b46872e5



Instituto Federal de Educação, Ciência e Tecnologia de Goiás
Avenida Furnas, nº 55, 55, Bairro Village Imperial, ITUMBIARA / GO, CEP 75524-010
(64) 2103-5612 (ramal: 5612)

DEDICATÓRIA

Dedico este trabalho aos meus pais e professores, que sempre demonstraram o quanto a educação pode ser transformadora.

AGRADECIMENTOS

Agradeço em primeiro lugar, aos meus pais e irmã, Solange Alves de Sena, Flávio Marques Lopes, e Ketally Alves Lopes, por todo apoio incondicional durante toda a minha vida.

Agradeço também ao professor orientador Dr. Ghunter Paulo Viajante pela ajuda, confiança e incentivo durante essa trajetória acadêmica.

Agradeço também aos colegas e professores do iTec-lab, especialmente ao professor Dr. Josemar Alves dos Santos Junior e aos colegas Yuri Dias Paraiba Cirino e Eryc Dias Medeiros Silva, pelos auxílios que contribuíram diretamente para o desenvolvimento deste trabalho.

RESUMO

A implementação de microrredes se trata de uma alternativa eficiente para aumento da confiabilidade no fornecimento de energia elétrica, tendo em vista que elas são compostas por sistemas de geração distribuída, de armazenamento de energia e cargas, que pode operar conectada à rede elétrica ou de forma totalmente isolada. Com isso, elas também são consideradas sustentáveis, já que utilizam de fontes renováveis, como energia fotovoltaica, eólica ou biomassa, e possuem a vantagem de implementação mais ágil, de forma a adiar, ou evitar, investimentos em grandes centrais geradoras. Ademais, para que as microrredes possam operar de maneira adequada, faz-se necessário a implementação de ferramentas para a supervisão e o controle dos seus componentes. Neste sentido, este trabalho apresenta o desenvolvimento de um sistema de baixo custo para supervisão e controle de inversores fotovoltaicos e híbridos, visando a aplicação em microrredes. Para isso, é proposto a implementação de um sistema microcontrolador em uma rede industrial (Modbus), de forma a permitir uma comunicação eficiente deste dispositivo, com os inversores de uma microrrede instalada no Instituto Federal de Goiás (IFG) – Campus Itumbiara. Assim, os dados obtidos nessa comunicação são tratados e enviados para interfaces específicas, através de redes Modbus e MQTT, que por sua vez, viabilizam as funcionalidades de supervisão e controle desses equipamentos. Neste caso, através da utilização desses recursos, torna-se possível realizar, na microrrede, a limitação de potência ativa, controle de reativos e do despacho de energia armazenada em baterias, contribuindo para aplicações no que envolve o controle das demandas de consumo e injeção, fator de potência e confiabilidade em sistemas de geração intermitente.

Palavras-chaves: microrrede, supervisão, controle, iot, microcontrolador, modbus, mqtt, node-red.

ABSTRACT

The implementation of microgrids is an efficient alternative for increasing the reliability of electricity supply, considering that they are composed of distributed generation systems, energy storage, and loads, which can operate connected to the power grid or completely isolated. As a result, they are also considered sustainable since they utilize renewable sources, such as photovoltaic, wind, or biomass energy, and they have the advantage of faster implementation, in order to postpone or avoid investments in large generating plants. Moreover, for microgrids to operate properly, the implementation of tools for the supervision and control of their components is necessary. In this sense, this work presents the development of a low-cost system for the supervision and control of photovoltaic and hybrid inverters, aimed at application in microgrids. For this purpose, the implementation of a microcontroller system in an industrial network (Modbus) is proposed, in order to allow efficient communication between this device and the inverters of a microgrid installed at the Federal Institute of Goiás (IFG) – Itumbiara Campus. Thus, the data obtained from this communication are processed and sent to specific interfaces, via Modbus and MQTT networks, which in turn enable the supervision and control functionalities of these devices. In this case, through the use of these resources, it becomes possible to perform in the microgrid the limitation of active power, reactive control, and the dispatch of energy stored in batteries, contributing to applications involving the control of consumption and injection demands, power factor, and reliability in intermittent generation systems.

Keywords: *microgrid, monitoring, control, iot, microcontroller, modbus, mqtt, node-red.*

LISTA DE FIGURAS

Figura 1 - Esquemático do sistema completo de supervisão e controle dos inversores	21
Figura 2 - Esquemático de uma microrrede	23
Figura 3 - Conversor de um estágio	28
Figura 4 - Conversor de dois estágios	28
Figura 5 - Estrutura das funções e etapas de controle de uma microrrede	32
Figura 6 - Microrrede do Laboratório de Fontes Renováveis	34
Figura 7 - Região interna do rack de baterias	38
Figura 8 - Diagrama de conexão das baterias	39
Figura 9 - Esquema de conexão dos componentes da microrrede	40
Figura 10 - Esquemático das redes Modbus implementadas na microrrede	43
Figura 11 - Modbus Serial UDP	45
Figura 12 - Conexão dos dispositivos em uma rede Modbus RTU no padrão RS485	46
Figura 13 - Tipos de mensagem da comunicação Modbus TCP	48
Figura 14 - Empacotamento da mensagem na comunicação Modbus TCP/IP	48
Figura 15 - Representação da rede Modbus TCP/IP estabelecida	50
Figura 16 - Ativação do ponto de acesso Wi-Fi do inversor fotovoltaico	51
Figura 17 - Interface web de configuração do inversor fotovoltaico	52
Figura 18 - Interface de configuração de rede do inversor fotovoltaico	53
Figura 19 - Definição de IP fixo do inversor fotovoltaico	54
Figura 20 - Interface de configuração da comunicação do inversor fotovoltaico	54
Figura 21 - Amostra da tabela de registradores do inversor fotovoltaico	55
Figura 22 - Exemplo de correção de escala para registradores do inversor fotovoltaico	56
Figura 23 - Interface de configuração do Unity ID do inversor fotovoltaico	57
Figura 24 - Ponto de acesso Wi-Fi gerado pelo inversor híbrido	58
Figura 25 - Interface de configuração web do inversor fotovoltaico	59
Figura 26 - Interface de configuração Wi-Fi do inversor híbrido	59
Figura 27 - Configuração de IP estático do inversor híbrido	60
Figura 28 - Interface de configuração da comunicação Modbus do inversor híbrido	61
Figura 29 - Amostra da Tabela de Registradores do Inversor Híbrido	62
Figura 30 - Amostra da tabela de mapeamento dos Unity IDs do Inversor Híbrido	62
Figura 31 - Microcontrolador modelo ESP32 utilizado no trabalho	64
Figura 32 - Fluxograma da função de conexão Wi-Fi do microcontrolador	66

Figura 33 - Ilustração da estrutura "dados" utilizado no trabalho	67
Figura 34 - Ilustração da utilização de um vetor de estruturas	67
Figura 35 - Fluxograma da função "addReg"	68
Figura 36 - Fluxograma da comunicação Modbus do Microcontrolador	70
Figura 37 - Fluxograma da função de leitura na comunicação Modbus	71
Figura 38 - Fluxograma da função de escrita na comunicação Modbus	72
Figura 39 - Exemplo de implementação de uma rede MQTT	75
Figura 40 - Arquivo de configuração do Mosquitto Broker	76
Figura 41 - Configuração da porta de operação do broker	77
Figura 42 - Configuração inicial para autenticação de usuário do broker	77
Figura 43 - Acesso da pasta de instalação padrão do <i>broker</i> via terminal	78
Figura 44 - Definição de usuário e senha de acesso do broker	78
Figura 45 - Arquivo de autenticação do <i>broker</i> configurado	79
Figura 46 - Validação do arquivo de autenticação de acesso ao broker	79
Figura 47 - Configuração do Firewall do Windows para liberação da porta de comunicação do broker	80
Figura 48 - Configuração da inicialização automática do broker	81
Figura 49 - Exemplo de nomeação dos tópicos da comunicação MQTT	83
Figura 50 - Fluxograma da comunicação MQTT do Microcontrolador	84
Figura 51 - Função de <i>callback</i> da comunicação MQTT do microcontrolador	85
Figura 52 - Fluxograma da função de conexão ao servidor MQTT	86
Figura 53 - Formato da mensagem a ser publicada na rede MQTT	86
Figura 54 - Função para publicação de dados via comunicação MQTT	87
Figura 55 - Comando de instalação do Node-RED	89
Figura 56 - Comando de inicialização do servidor Node-Red	90
Figura 57 - Ambiente de programação do Node-RED	91
Figura 58 - Definição da senha de acesso ao editor do Node-RED	91
Figura 59 - Definição de usuário e senha de acesso ao editor do Node-RED	92
Figura 60 - Interface de autenticação do editor do Node-RED	92
Figura 61 - Inserção do nó " <i>mqtt in</i> " no Node-RED	93
Figura 62 - Edição do nó " <i>mqtt in</i> " no Node-RED	94
Figura 63 - Configuração do <i>broker</i> de conexão do nó " <i>mqtt in</i> "	95
Figura 64 - Configuração do nó " <i>mqtt in</i> " validado	95
Figura 65 - Utilização do nó " <i>function</i> " no Node-RED	96

Figura 66 - Configuração do nó " <i>function</i> " do Node-RED	97
Figura 67 - Fluxograma do algoritmo para correção de escala no Node-RED	98
Figura 68 - Implementação do nó " <i>link out</i> " no Node-RED	98
Figura 69 - Configuração do nó " <i>link out</i> " no Node-RED	99
Figura 70 - Identificação do fluxo criado	100
Figura 71 - Amostra do fluxo final da comunicação MQTT no Node-RED	100
Figura 72 - Acesso a configuração para inserção de nós	101
Figura 73 - Instalação dos nós do Node-RED Dashboard	102
Figura 74 - Nós para Dashboard instalados	102
Figura 75 - Inserção do nó " <i>link in</i> "	103
Figura 76 - Configuração do nó " <i>text</i> " no Node-RED	104
Figura 77 - Configuração de grupos para a interface web do Node-RED Dashboard	105
Figura 78 - Configuração de guias para a interface web do Node-RED Dashboard	105
Figura 79 - Configuração do nó " <i>chart</i> " no Node-RED	106
Figura 80 - Processo de abertura da interface de supervisão configurada	107
Figura 81 - Interface de supervisão inicial configurada	107
Figura 82 - Configuração do nó " <i>form</i> " no Node-RED	108
Figura 83 - Interface gerada pelo nó " <i>form</i> " no <i>dashboard</i>	109
Figura 84 - Mensagem de resposta do nó " <i>form</i> " no Node-RED	109
Figura 85 - Amostra dos registradores para limitação da potência de saída do inversor fotovoltaico	110
Figura 86 - Fluxo para a interface de limitação de potência do inversor fotovoltaico	111
Figura 87 - Fluxo para interface de configuração do modo de operação do inversor híbrido	112
Figura 88 - Fluxo para interface de programação horária de limitação de potência do inversor fotovoltaico	114
Figura 89 - Fluxograma do algoritmo de verificação do nó " <i>Verificação do horário</i> "	115
Figura 90 - Instalação do banco de dados MySQL no XAMPP	117
Figura 91 - Painel de controle do XAMPP	117
Figura 92 - Interface de gerenciamento do banco de dados MySQL	118
Figura 93 - Adição de conta de usuário para acesso ao banco de dados MySQL	118
Figura 94 - Configuração do usuário de acesso ao banco de dados via Node-RED	120
Figura 95 - Interface para criação de um novo banco de dados	121
Figura 96 - Esquema do banco de dados implementado	121
Figura 97 - Criação de uma nova tabela para o banco de dados	121

Figura 98 - Especificação das colunas das tabelas do banco de dados	122
Figura 99 - Criação de uma nova tabela no banco de dados "bd_monitoramento"	123
Figura 100 - Estrutura básica de utilização do nó "mysql"	124
Figura 101 - Configuração do nó "mysql" no Node-RED	125
Figura 102 - Fluxo para o registro de dados no banco de dados via Node-RED	125
Figura 103 - Fluxograma para operação de registro no banco de dados via Node-RED	127
Figura 104 - Primeira parte do fluxo para consulta histórica via Node-RED	128
Figura 105 - Configuração do nó "dropdown" da interface de consulta histórica no Node-RED	129
Figura 106 - Configuração do nó "text input" da interface de consulta histórica utilizada no Node-RED	130
Figura 107 - Fluxo completo da interface de consulta histórica via Node-RED	130
Figura 108 - Configuração do botão de confirmação da interface de consulta histórica do Node-RED	131
Figura 109 - Fluxograma do fluxo da interface de consulta histórica via Node-RED	132
Figura 110 - IHM Samkoon EA-043	133
Figura 111 - Tela de supervisão geral da IHM	134
Figura 112 - Tela dos gráficos de potência da IHM	135
Figura 113 - Definição da linguagem do software SATOOL	136
Figura 114 - Confirmação da instalação dos drivers da IHM	136
Figura 115 - Ambiente do software SATOOL	137
Figura 116 - Configuração do projeto básico para interface da IHM	138
Figura 117 - Configuração do protocolo de comunicação da IHM	138
Figura 118 - Configuração da Tela 1 da IHM	139
Figura 119 - Configuração da comunicação da IHM com o ESP32	139
Figura 120 - Configuração do padrão RS232 na IHM	140
Figura 121 - Configurações adicionais da comunicação Modbus da IHM	141
Figura 122 - Configuração do <i>Numeric Display</i> da IHM	142
Figura 123 - Posicionamento do <i>Numeric Display</i>	144
Figura 124 - Configuração da tela de visão geral da IHM	145
Figura 125 - Configuração básica do <i>Static Picture</i>	145
Figura 126 - Configuração geral do Trend chart	146
Figura 127 - Botão de gatilho para funcionamento do gráfico	148
Figura 128 - Alocação do botão de gatilho para a camada inferior	148

Figura 129 - Configuração das linhas do gráfico	149
Figura 130 - Configuração das grades do gráfico	149
Figura 131 - Vista final da Tela de gráficos da IHM	150
Figura 132 - Configuração do botão de avanço de tela	151
Figura 133 - Configuração do botão de retorno de tela	152
Figura 134 - Download do projeto para a IHM	153
Figura 135 - Fluxograma da rotina de comunicação Modbus RTU do ESP32	155
Figura 136 - Representação da alocação de dados nos registradores Modbus	156
Figura 137 - Fluxograma da função att_Reg_ESP	156
Figura 138 - Função "floatToRegisters"	158
Figura 139 - Portas de comunicação serial da IHM (a) e do ESP32 (b)	159
Figura 140 - Padrão de conexão do conector DB9 para Modbus RS232	160
Figura 141 - Conversor TTL para RS232	161
Figura 142 - Esquema de ligação do microcontrolador com a IHM	162
Figura 143 - Guia de supervisão geral dos inversores	165
Figura 144 - Guia de supervisão e controle do inversor fotovoltaico	166
Figura 145 - Configuração da limitação de potência ativa do inversor fotovoltaico	167
Figura 146 - Operação do inversor fotovoltaico com a potência de injeção limitada	168
Figura 147 - Configuração do fator de potência (a) indutivo e (b) capacitivo	169
Figura 148 - Operação do inversor com fator de potência (a) indutivo e (b) capacitivo	169
Figura 149 - Configuração de desconexão do inversor fotovoltaico	170
Figura 150 - Resultado da configuração de desconexão do inversor fotovoltaico	170
Figura 151 - Guia específica para a supervisão e controle do inversor híbrido	171
Figura 152 - Configuração do despacho de energia armazenada considerando (a) consumo máximo da microrrede e (b) injeção mínima na rede	172
Figura 153 - Reposta do sistema a configuração positiva no despacho das baterias	173
Figura 154 - Resposta do sistema a configuração negativa no despacho das baterias	174
Figura 155 - Configuração do status do inversor híbrido	175
Figura 156 - Configuração da permissão de carregamento e injeção do inversor híbrido	176
Figura 157 - Interface de programação horária dos inversores	176
Figura 158 - Resposta da programação de despacho da energia armazenada nas baterias ...	177
Figura 159 - Resposta da programação de limitação de injeção do inversor fotovoltaico	178
Figura 160 - Resposta da programação de alteração do fator de potência do inversor fotovoltaico	178

Figura 161 - Interface para a configuração da consulta histórica dos dados dos inversores .	179
Figura 162 - Configuração de consulta na interface de dados históricos	180
Figura 163 - Consulta histórica para os dados de potência da microrrede	181
Figura 164 - Consulta histórica para os dados de corrente da microrrede	181
Figura 165 - Consulta histórica para os dados de tensão da microrrede	182
Figura 166 - Protótipo do dispositivo de supervisão local	183

LISTA DE TABELAS

Tabela 1 - Descrição dos componentes da microrrede	35
Tabela 2 - Componentes da estrutura utilizada no algoritmo	67
Tabela 3 - Endereçamento utilizado no <i>Monitor Address</i>	143
Tabela 4 - Endereço dos registradores inseridos no microcontrolador	157
Tabela 5 - Resumo das funcionalidades do sistema de supervisão e controle	164

LISTA DE ABREVIATURAS E SIGLAS

A_I	<i>Auto_Increment</i>
ADU	<i>Application Data Unit</i>
ANEEL	Agência Nacional de Energia Elétrica
APIs	<i>Application Programming Interface</i>
BD	Banco de Dados
BMS	<i>Battery Manegement System</i>
BMU	<i>Battery Management Unity</i>
CA	Corrente Alternada
CC	Corrente Contínua
CLPs	Controladores Lógico Programáveis
DEC	Duração Equivalente de Interrupção por Unidade Consumidora
EMS	<i>Energy Management System</i>
EPE	Empresa de Pesquisa Energética
FEC	Frequência Equivalente de Interrupção por Unidade Consumidora
FP	Fator de Potência
GD	Geração Distribuída
GPIOs	<i>General Purpose Input Output</i>
HFAC	<i>High Frequency Alternating Current</i>
IBM	<i>International Business Machines Corporation</i>
IHM	Interface Homem-Máquina
IoT	<i>Internet of Things</i>
IP	<i>Internet Protocol</i>
ISO	<i>International Standards Organization</i>
JSON	<i>JavaScript Object Notation</i>
LFAC	<i>Line Frequency Alternating Current</i>
M2M	<i>Machine to Machine</i>
MBAP Header	<i>Modbus Application Protocol Header</i>
MQTT	<i>Message Queuing Telemetry Transport</i>
OSI	<i>Open Systems Interconnection</i>
PDE	Plano Decenal de Expansão de Energia
Prodist	Procedimento de Distribuição
REDs	Recursos Energéticos Distribuídos
SAE	Sistemas de Armazenamento de Energia
SOC	<i>State Of Charge</i>
UART	<i>Universal Asynchronous Receiver-Transmitter</i>
UDP	Unidade de Dados de Protocolo

SUMÁRIO

1 CAPÍTULO I - INTRODUÇÃO	19
2 CAPÍTULO II - MICRORREDES	23
2.1 CONCEITOS	23
2.1.1 Cargas	23
2.1.2 Recursos Energéticos Distribuídos	24
2.1.3 Elementos de controle	25
2.2 TOPOLOGIAS DE MICRORREDES	26
2.2.1 Grandezas Elétricas	26
2.2.2 Aplicação das Microrredes	26
2.2.3 Tipo dos Conversores Eletrônicos	27
2.2.4 Tipo de Controle e Supervisão	28
2.2.5 Conexão dos RED	29
2.2.6 Quantidade de Fases	29
2.3 REQUISITOS E NÍVEIS DE CONTROLE	30
2.4 MODOS DE OPERAÇÃO	32
2.5 MICRORREDE DE ESTUDO	34
2.5.1 Módulos e Inversor Fotovoltaico	35
2.5.2 Inversor Híbrido	36
2.5.3 Sistema de armazenamento de energia em banco de baterias	37
2.5.4 Característica de operação do sistema	39
3 CAPÍTULO III - COMUNICAÇÃO COM OS INVERSORES	42
3.1 REDES INDUSTRIAIS	42
3.1.1 Protocolo Modbus	44
3.1.1.1 Modbus Serial	44
3.1.1.2 Modbus TCP/IP	47
3.2 IMPLEMENTAÇÃO DA REDE DE COMUNICAÇÃO DOS INVERSORES	50
3.2.1 Comunicação com o Inversor Fotovoltaico	51
3.2.1.1 Pré configuração do equipamento	51
3.2.1.2 Relação de endereço dos registradores	55
3.2.2 Comunicação com o inversor híbrido	58
3.2.2.1 Pré configuração do equipamento	58
3.2.2.2 Relação de endereço dos registradores	61

3.2.3 PROGRAMAÇÃO DA PLATAFORMA MICROCONTROLADA	63
3.2.3.1 Conexão com a rede Wi-Fi e Estrutura de dados	65
3.2.3.2 Comunicação com os inversores	69
4 CAPÍTULO IV - COMPARTILHAMENTO DE DADOS	74
4.1 PROTOCOLO MQTT	74
4.1.1 Implementação da rede MQTT	75
4.2 PROGRAMAÇÃO DA PLATAFORMA MICROCONTROLADA	81
5 CAPÍTULO V - INTERFACE WEB	88
5.1 NODE-RED	88
5.1.1 Instalação do servidor Node-RED	89
5.1.2 Vinculação com o <i>broker</i> MQTT	93
5.1.2.1 Formatação e Direcionamento das Mensagens Recebidas	96
5.1.3 Desenvolvimento da Interface de Supervisão e Controle	101
5.1.3.1 Implementação dos Recursos de Supervisão	103
5.1.3.2 Implementação dos recursos de controle	108
5.1.3.3 Implementação dos recursos de programação horária	113
5.1.4 Desenvolvimento da interface de consulta histórica	116
5.1.4.1 Instalação e Modelagem do Banco de Dados	116
5.1.4.2 Inserção de registros no banco de dados a partir do Node-RED	123
5.1.4.3 Requisição de registros do banco de dados a partir do Node-RED	127
6 CAPÍTULO VI - INTERFACE DE MONITORAMENTO LOCAL	133
6.1 CONFIGURAÇÃO DA IHM	134
6.1.1 Instalação do software de configuração da IHM	135
6.1.2 Configurações básicas da IHM	136
6.1.3 Configuração da tela de supervisão geral da IHM	141
6.1.4 Configuração do gráfico dinâmico	145
6.1.5 Configuração da navegação de telas	150
6.2 PROGRAMAÇÃO DA PLATAFORMA MICROCONTROLADA	153
6.3 CONEXÃO DO MICROCONTROLADOR COM A IHM	159
7 CAPÍTULO VII - RESULTADOS E DISCUSSÃO	163
7.1 GUIA WEB DE SUPERVISÃO GERAL	164
7.2 GUIA WEB DE SUPERVISÃO E CONTROLE DO INVERSOR FOTOVOLTAICO	165
7.2.1 Ajuste da limitação de potência	166
7.2.2 Ajuste do fator de potência de operação	168

7.2.3 Ajuste da conexão do inversor fotovoltaico	169
7.3 GUIA DE SUPERVISÃO E CONTROLE DO INVERSOR HÍBRIDO	170
7.3.1 Ajuste do despacho de energia armazenada	172
7.3.2 Ajuste do status do inversor híbrido	175
7.3.3 Ajuste das permissões de operação do inversor híbrido	175
7.4 GUIA DE CONFIGURAÇÃO PROGRAMÁVEL DOS INVERSORES	176
7.5 GUIA DE CONSULTAS HISTÓRICAS	179
7.6 INTERFACE DE SUPERVISÃO LOCAL	182
8 CAPÍTULO VII - CONSIDERAÇÕES FINAIS	184
REFERÊNCIAS	186
APÊNDICE A - CÓDIGO DO MICROCONTROLADOR ESP32	191
APÊNDICE B - FUNÇÃO PARA CORREÇÃO DE FATOR DE ESCALA NO NODE-RED	218
APÊNDICE C - CÓDIGOS PARA OS NÓS DE FUNÇÃO DA INTERFACE DE CONTROLE DA LIMITAÇÃO DE POTÊNCIA ATIVA DO INVERSOR FOTOVOLTAICO	219
APÊNDICE D - CÓDIGOS PARA OS NÓS DE FUNÇÃO DA INTERFACE DE CONTROLE DO MODO DE OPERAÇÃO DO INVERSOR HÍBRIDO	221
APÊNDICE E - CÓDIGOS PARA OS NÓS DE FUNÇÃO DA INTERFACE DE PROGRAMAÇÃO DE INJEÇÃO DO INVERSOR FOTOVOLTAICO	222
APÊNDICE F - CÓDIGOS PARA OS NÓS DE FUNÇÃO DA INTERFACE DE REGISTRO DE INFORMAÇÕES NO BANCO DE DADOS MYSQL	226
APÊNDICE G - CÓDIGOS PARA OS NÓS DE FUNÇÃO DA INTERFACE DE CONSULTA HISTÓRICA NO BANCO DE DADOS MYSQL	227

1 CAPÍTULO I - INTRODUÇÃO

Anualmente, a Agência Nacional de Energia Elétrica (ANEEL), disponibiliza um relatório de desempenho das distribuidoras no fornecimento de energia elétrica para os consumidores, tendo como base dois indicadores, o FEC e o DEC. Estes indicadores estão relacionados, respectivamente, com a frequência e a duração das interrupções na distribuição de energia por unidade consumidora, de forma a não considerar a influência de eventos climáticos externos [1].

Assim, com a elaboração deste ranking entre as concessionárias de todo o país, a ANEEL visa garantir um serviço de melhor qualidade para os consumidores, tendo em vista que quanto menor for o registro desses indicadores, maior será a confiabilidade no fornecimento das distribuidoras.

Ademais, como apresentado em [1], estes indicadores não obtiveram um resultado satisfatório para o estado de Goiás no ano de 2023, tendo em vista que a concessionária responsável pelo fornecimento de energia para a região, ficou em último lugar no ranking nacional. Esta situação representa uma queda de duas posições em relação aos anos de 2021 e 2022, de forma a demonstrar que a confiabilidade no fornecimento de energia elétrica no estado de Goiás, se trata de um tópico discutível para a região [2][3].

Como possíveis justificativas para essa situação, pode-se destacar a precariedade das linhas de distribuição, a sobrecarga de transformadores, e o ajuste incorreto de sistemas de proteção, que, em conjunto, ocasionam na interrupção do fornecimento de energia por parte das concessionárias.

De maneira complementar, tendo como base o Plano Decenal de Expansão de Energia (PDE), que é realizado pela Empresa de Pesquisa Energética (EPE), e apresenta as perspectivas da expansão do setor de energia, estima-se um aumento médio anual de 3,4% no consumo de energia elétrica até o ano de 2034 [4]. Dessa forma, com o aumento da demanda no fornecimento de energia elétrica, faz-se necessário garantir a confiabilidade no fornecimento dessa energia, tendo em vista que, em alguns casos, a distribuidora pode não garantir a melhor qualidade para o fornecimento deste serviço.

Assim, a implementação de fontes de Geração Distribuída (GD) pode ser destacada como uma alternativa viável, baseando-se na premissa de que esta solução é definida como a implementação de fontes geradoras mais próximas dos consumidores, de maneira a evitar grandes investimentos que envolvem a implementação de centrais geradoras e linhas de

transmissão. Além disso, a utilização de fontes de GD pode auxiliar na redução das perdas nas linhas de transmissão e distribuição, já que a energia produzida é consumida quase que instantaneamente no local. Por último, tem-se ainda um aumento no fornecimento ininterrupto de eletricidade, tendo em vista a diversidade de fontes que podem compor este tipo de geração [5].

Uma outra alternativa para auxiliar na maior confiabilidade no fornecimento de energia elétrica, se trata da implementação de microrredes, que podem aproveitar de uma maneira mais eficiente, os recursos energéticos disponíveis em uma determinada instalação. Segundo [6], uma microrrede pode ser compreendida como sistema integrado de GD e cargas que pode operar junto à rede elétrica, fornecendo energia a rede de distribuição, ou de forma isolada, cumprindo localmente e, em menor escala, as funções do sistema centralizado. Dessa forma, esta tipologia de sistema se torna uma alternativa sustentável, uma vez que sua implementação é mais rápida, podendo adiar ou evitar investimento em grandes centrais geradoras, além de utilizar de fontes renováveis de energia, como a fotovoltaica, eólica ou biomassa [7].

De forma complementar, sistemas de armazenamento de energia (SAE) em baterias também são comumente relacionados às microrredes, principalmente em sistemas isolados baseados em fontes intermitentes de energia, como no caso da fotovoltaica e eólica. Isso ocorre devido a rápida resposta deste tipo de sistema de armazenamento a perturbações que podem ocorrer na microrrede, tanto em relação a aumento e redução de geração, quanto a aumento e redução de carga, fazendo com que seja necessário controlar o despacho de energia em um curto período de tempo [8]. Assim, a implementação deste tipo de tecnologia em conjunto com fontes de GD e cargas, ocasiona em uma estrutura básica para microrredes, em que esses componentes passam a ser chamados de Recursos Energéticos Distribuídos (REDs).

Portanto, este trabalho tem como objetivo o desenvolvimento de um sistema de baixo custo para supervisionar e controlar a operação de inversores fotovoltaicos e híbridos, sendo aplicado em uma microrrede composta por sistemas de geração fotovoltaica e armazenamento em baterias. Para atingir esse objetivo, utilizou-se da implementação de uma plataforma microcontrolada em uma rede industrial, de forma a viabilizar a comunicação entre este dispositivo e os inversores que compõem uma microrrede instalada no Instituto Federal de Educação, Ciência e Tecnologia de Goiás (IFG) - Campus Itumbiara. Uma visão geral dos conceitos que envolvem uma microrrede pode ser encontrada no capítulo 2, assim como o detalhamento dos componentes que compõem a microrrede objeto deste estudo.

Com a implementação bem sucedida da comunicação entre esses dispositivos, as informações obtidas foram apresentadas em uma interface web, acessível por qualquer dispositivo com acesso a rede do laboratório, em que é possível utilizar das ferramentas de controle da microrrede. De maneira complementar, os dados gerais de operação também foram apresentados em uma Interface Homem-Máquina (IHM), seguindo as particularidades ilustradas na Figura 1, em que é possível visualizar um esquemático geral do funcionamento do sistema proposto neste trabalho.

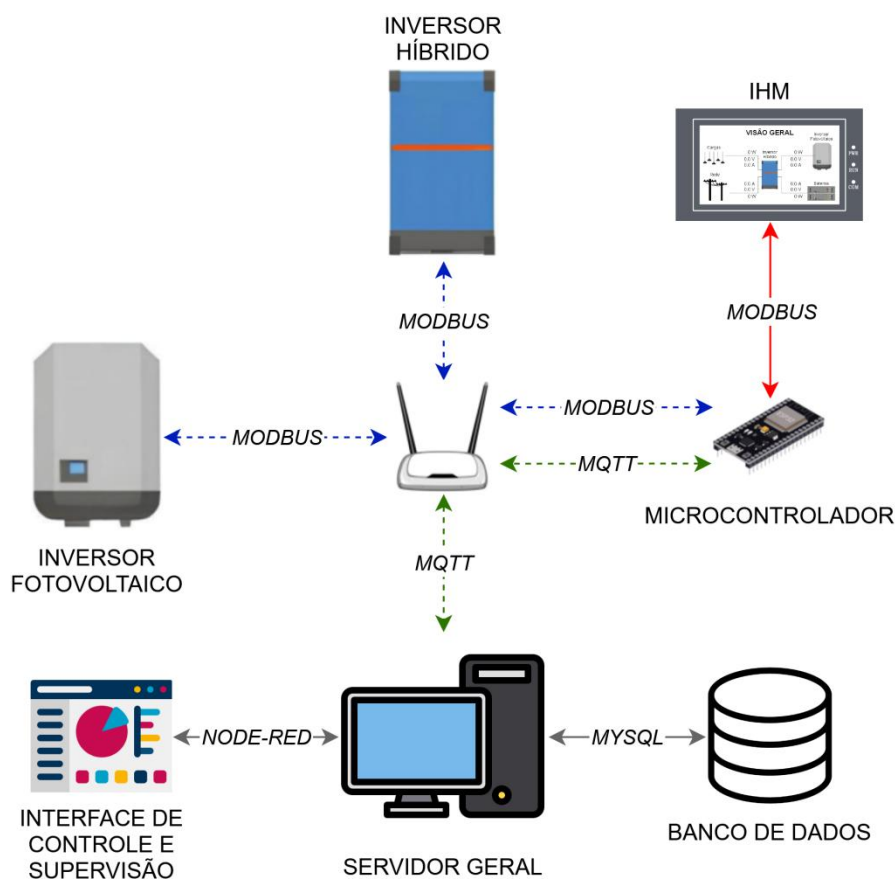


Figura 1 - Esquemático do sistema completo de supervisão e controle dos inversores
Fonte: autoria própria

Na Figura 1, nota-se a total dependência da plataforma microcontrolada em estabelecer a comunicação com os demais componentes do sistema por meio de um rede Wi-Fi, de forma que a partir disso, fosse possível viabilizar a implementação de uma rede Modbus TCP para a comunicação com os inversores da microrrede, conforme descrito detalhadamente no capítulo 3 deste trabalho.

Além disso, é através desta mesma conexão com o roteador que o dispositivo realiza o compartilhamento dos dados da comunicação com os inversores para um servidor MQTT, que

neste caso, se encontra instalado em um computador representado como “Servidor Geral” na Figura 1. Assim, todo procedimento de configuração desta funcionalidade se encontra apresentado no capítulo 4.

De maneira complementar, o capítulo 5 destaca o processo de desenvolvimento da interface web para a supervisão e controle dos inversores, juntamente com a implementação de um banco de dados local. Com isso, tem-se como objetivo o armazenamento das informações de interesse no que diz respeito a operação deste sistema, de forma que seja necessário a utilização de servidores Node-RED e MYSQL no computador descrito como “Servidor Geral” na Figura 1.

Em relação a configuração da IHM, esta foi descrita no capítulo 6 deste trabalho, que, neste caso, necessitou da implementação de uma rede Modbus RS232 para viabilizar a troca de informações entre este dispositivo e a plataforma microcontrolada.

Já no capítulo 7, é realizado a apresentação dos resultados que foram obtidos com a operação do sistema completo, no que diz respeito a supervisão em tempo real e histórica de parâmetros como tensão, corrente, potência, e também das funcionalidades de controle de injeção de potência, reativos e despacho das baterias.

Por fim, o capítulo 8 demonstra as conclusões obtidas durante as fases de desenvolvimento e teste deste trabalho, havendo, ainda, a indicação de como este pode ser aproveitado em trabalhos futuros.

Com isso, tendo em vista toda a conectividade implementada entre os dispositivos na Figura 1, há, então, a caracterização da aplicação IoT (*Internet of Things*) neste trabalho, tendo em vista que este termo está relacionado com a capacidade de um conjunto de dispositivos realizar a troca de informações entre si.

2 CAPÍTULO II - MICRORREDES

2.1 CONCEITOS

Os tópicos a seguir apresentam os principais conceitos no que diz respeito aos elementos que se encontram presentes no estudo de microrredes, como a caracterização de cargas e recursos energéticos distribuídos nesta tipologia de instalação. De maneira complementar, também é demonstrado uma visão geral a respeito dos elementos de controle encontrados em uma microrrede.

2.1.1 Cargas

Como apresentado anteriormente, as microrredes se tratam de instalações compostas por um conjunto de cargas e sistemas de geração distribuídas, que podem operar conectados ou isolados da rede de distribuição local, geralmente em baixa ou média tensão. Na Figura 2, é possível observar o esquemático de uma microrrede composta por sistemas de GD, armazenamento em baterias e um conjunto de cargas.

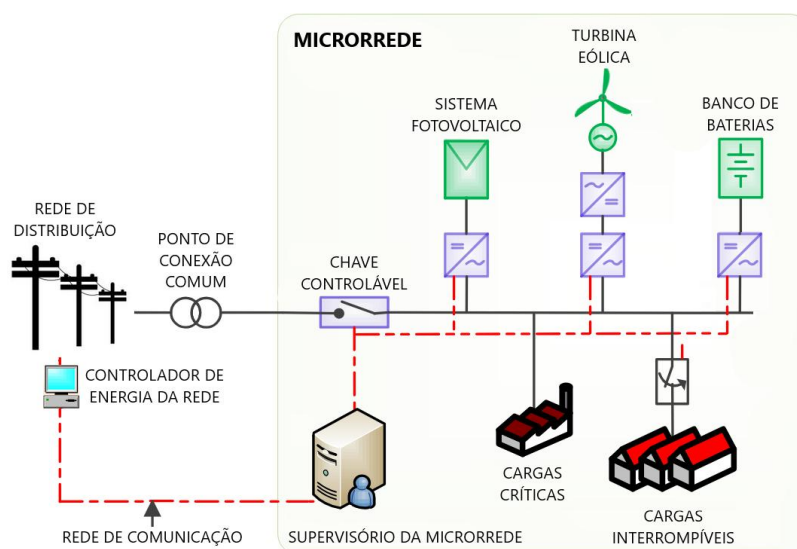


Figura 2 - Esquemático de uma microrrede
 Fonte: Adaptado de [9]

Segundo [5] (apud IEE, 2018, p. 43), essas cargas podem ser conceituadas em 6 diferentes tipos, sendo elas as cargas críticas, prioritárias, controláveis, interrompíveis, de desvios e despejo, e, por fim, as cargas indefinidas.

Primeiramente, as cargas críticas são aquelas que devem ser atendidas em todos os modos operacionais da microrrede, não podendo ser desenergizadas, como no caso de equipamentos de TI e sistemas de segurança e controle.

Em segundo lugar, as carga prioritárias são definidas como aquelas que podem ser desligadas quando houver a necessidade, mas que em grande parte da sua operação devem permanecer com suprimento de energia elétrica, como no caso dos sistemas de iluminação.

De forma complementar, as cargas controláveis permitem ajustes no próprio consumo, até mesmo interrupções periódicas na operação. Um exemplo mais comum deste tipo de equipamento, seriam os sistemas de climatização, como ar condicionado e climatizadores.

As cargas interrompíveis, por outro lado, podem ser desligadas a qualquer momento, de forma a priorizar o atendimento das cargas críticas ou prioritárias. Como exemplo, destaca-se sistemas de armazenamento térmico, bombas de água, e o carregamento das baterias.

Em relação as cargas de desvio ou despejo, estas serão utilizadas quando houver a necessidade de absorver o excesso de geração na microrrede, tendo em vista aspectos técnicos e econômicos.

Por fim, ainda existem as cargas indefinidas, sendo aquelas que não se encaixam em nenhuma das definições apresentadas anteriormente, possuindo, assim, características bem específicas.

2.1.2 Recursos Energéticos Distribuídos

Ainda no contexto das microrredes, como a apresentado na Figura 2, um outro conceito importante está relacionado com os Recursos Energéticos Distribuídos (REDs), que podem compreender tanto os sistemas de GD, quanto os sistemas de armazenamento de energia (SAE) [5][8].

Neste sentido, as fontes de GD podem ser separadas em despacháveis e não despacháveis, sendo que no primeiro caso, caracterizam-se aquelas fontes que podem ser controladas pelos sistemas da microrrede, podendo, dessa forma, sofrer alteração quanto a sua capacidade de geração de energia, e os limites de tempo para entrar ou sair de operação. Já no

caso das fontes não despacháveis, destacam-se aquelas que não possuem, por natureza, um controle elevado da sua capacidade de geração, como acontece no caso das fontes intermitentes, como a fotovoltaica e eólica.

Em relação aos SAE, estes são os responsáveis por manter o equilíbrio na microrrede, através do armazenamento de energia no momento de alta geração local, mas que também auxilia nas situações em que se faz necessário suprir uma determinada demanda, seja na operação conectada à rede de distribuição, mas especialmente nos momentos de operação isolada dela.

2.1.3 Elementos de controle

Por fim, um último aspecto de suma importância acerca da operação das microrredes, está relacionada com os elementos de controle do sistema, que podem ser destacados através do sistema de gerenciamento de energia (EMS do inglês *Energy Management System*), em conjunto com seus respectivos elementos complementares [5].

O EMS é responsável pelo controle da microrrede, através do gerenciamento dos modos de operação (conectado ou isolado), dos REDs, das cargas, e dos parâmetros elétricos da instalação, como tensão, frequência e fator de potência (FP). Além disso, ele fornece dados sobre a operação da instalação, de forma a garantir total monitoramento de todas as características da microrrede. Para isso, faz-se necessário a implementação de alguns elementos complementares, como sensores, atuadores, sistemas e redes de comunicação, de forma que seja possível integrar hardware e software e garantir tanto a supervisão quanto o controle da microrrede de maneira confiável.

Destaca-se, como exemplo, o conceito apresentado na Lei 14.300 de janeiro de 2022 [10], que une todas essas características apresentadas, fortalecendo ainda mais a sua implementação futura, tendo em vista todo aspecto legal vigente. Nesta lei é definido que as fontes de geração fotovoltaicas de até 3 MW podem ser consideradas fontes despacháveis, desde que possuam instalados, em conjunto, um SAE em baterias com capacidade mínima equivalente a 20% da capacidade de geração mensal do sistema de GD. Além disso, faz-se necessário a implementação de um controlador que permita a configuração do despacho de energia de maneira local ou remota.

Portanto, a operação das microrredes exige uma integração eficiente entre os sistemas que a compõem, fazendo com que o uso dessas tecnologias não só possibilite uma operação confiável, mas também atenda às exigências legais para uma implementação sustentável e eficiente.

2.2 TOPOLOGIAS DE MICRORREDES

Atualmente, as microrredes podem ser classificadas considerando seis grandes aspectos, como as grandezas elétricas que elas se baseiam, a sua respectiva aplicação, a topologia dos conversores eletrônicos utilizados, tipo de controle e supervisão aplicados, tipo dos RED e a quantidade de fases da instalação [5][11][12].

2.2.1 Grandezas Elétricas

Segundo [11], as microrredes podem ser baseadas em corrente contínua (CC), corrente alternada (CA), ou também híbridas (CA e CC). Além disso, as microrredes baseadas em CA podem ser classificadas de acordo com a sua frequência de operação, sendo elas em alta frequência quando operam a 400 ou 500 Hz (HFAC - *High Frequency Alternating Current*), ou em frequência industrial na operação de 50 ou 60 Hz (LFAC - *Line Frequency Alternating Current*).

2.2.2 Aplicação das Microrredes

Quanto a aplicação das microrredes, os autores de [13] definem que elas podem ser separadas em microrredes utilitárias, microrredes comerciais e industriais, e as microrredes isoladas.

A primeira situação é destacada como uma microrrede inserida no contexto de um alimentador de distribuição, fazendo com que parte de uma cidade, como um bairro, pudesse operar de maneira isolada conforme a necessidade da concessionária e dos consumidores. Além disso, esta tipologia ainda poderia fornecer um controle adicional quanto a parâmetros que envolvem a queda de tensão, perdas e o nível de reativos para os setores próximos da microrrede, de forma a aumentar a qualidade da energia distribuída [13].

Em relação as microrredes comerciais e industriais, estas estão relacionadas com as instalações comerciais, residenciais e industriais que possuem cargas críticas e sensíveis, de forma a exigir altos níveis de qualidade e confiabilidade no fornecimento de energia. Neste sentido, as microrredes são aplicadas de forma a prevenir interrupções, garantir um índice mínimo de qualidade da energia, e ainda fornecer alternativas para gerenciamento dos recursos destes locais, como o controle da demanda de uma instalação [13].

Já as microrredes isoladas podem ser descritas como aquelas que foram desenvolvidas para atender locais remotos, que não estão conectados aos sistemas de transmissão e distribuição centralizados. Assim, esta tipologia depende, em grande parte, de fontes como usinas fotovoltaicas, pequenas centrais hidrelétricas, aerogeradores e geradores movidos à gás ou combustível, além de sistemas de armazenamento de energia, de forma que seja possível garantir o suprimento de energia por longos períodos de tempo. Com base nisso, entende-se que o dimensionamento dos REDs, SAE e das cargas devem ser meticulosamente realizados nesta tipologia, pensando sempre em manter uma determinada reserva para contingências, tendo em vista o contexto de isolamento destes locais [13].

2.2.3 Tipo dos Conversores Eletrônicos

Os conversores eletrônicos utilizados nos RED, como os inversores fotovoltaicos ou controladores de carga de baterias, são classificados na microrrede de acordo com o número de estágios (ou conversões), que ocorrem no processamento da energia do local.

Os conversores de energia de um estágio, esquematizado na Figura 3, realizam apenas uma conversão de CC para CA, como no caso dos inversores fotovoltaicos, ou uma conversão de CA para CC, como no caso dos carregadores de baterias [11].

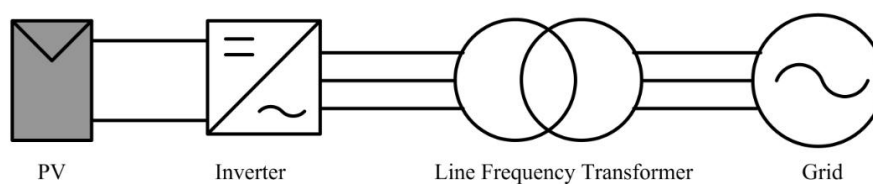


Figura 3 - Conversor de um estágio
Fonte: [11]

Já os conversores de dois estágios geralmente utilizam de um conversor CC-CC ou CA-CC para as fontes de GD, em cascata com um conversor CC-CA, de forma a processar a energia de maneira que ela possa ser utilizada em uma rede convencional de energia em corrente alternada. Esta última configuração permite um maior controle sobre a energia gerada pelas fontes, principalmente das intermitentes, de forma a extrair a máxima produção de energia delas [11]. Na Figura 4 é apresentada uma representação deste tipo de conversor de energia.

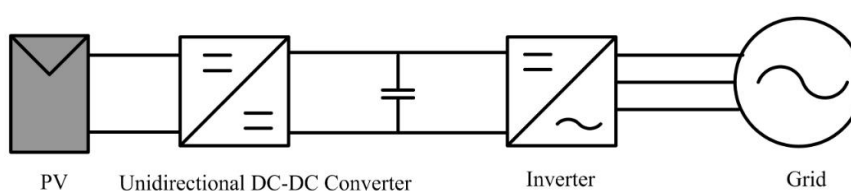


Figura 4 - Conversor de dois estágios
Fonte: [11]

2.2.4 Tipo de Controle e Supervisão

Levando em consideração as possíveis arquiteturas de controle e supervisão que podem ser implementadas em uma microrrede, duas vertentes são destacadas por [11], sendo elas do tipo centralizada ou descentralizada.

Basicamente, no controle centralizado, um controlador geral é implementado para gerenciar a microrrede com base nos protocolos internos, nos dados disponibilizados pelas cargas, pelos RED, e pelos sensores existentes na instalação. Com base nestes dados, este controlador realiza a configurações dos parâmetros operacionais para cada um destes componentes, de forma a manter a operação da microrrede de maneira integrada, comandada por um único dispositivo [11].

Já em relação ao controle descentralizado, este ocorre na aplicação de sistemas multiagentes, que permitem que cada unidade dos RED possuam um determinado nível de autonomia para manter o controle da microrrede. Para isso, faz-se necessário a implementação de softwares e redes de comunicação específicos, de forma a garantir a confiabilidade na operação de um sistema mais complexo [11].

2.2.5 Conexão dos RED

De acordo com [11], os recursos de geração de energia em uma microrrede podem ser classificados de acordo com a maneira que são conectados a rede, seja através de unidades rotativas, ou por meio de acoplamento eletrônico.

No primeiro caso, é utilizado uma máquina rotativa convencional, como no caso dos geradores à combustão interna, ou turbinas eólicas de velocidade fixa, sendo que, para o pleno funcionamento destes equipamentos, faz-se necessário manter a velocidade de rotação, assim como a frequência da rede, em níveis aceitáveis [11].

Para o acoplamento com equipamentos eletrônicos, estes são realizados através dos conversores de potência, de forma que eles possam se ajustar as características existentes na rede. Este é o caso dos inversores utilizados em sistemas fotovoltaicos, turbinas eólicas de velocidade variável e banco de baterias [11].

2.2.6 Quantidade de Fases

Em se tratando de microrredes do tipo CA, elas podem ser construídas com base em sistemas de tensão monofásicos ou trifásicos, a depender da quantidade de potência a ser utilizada na instalação [5].

2.3 REQUISITOS E NÍVEIS DE CONTROLE

A norma IEEE Std. 2030.7, é a responsável por estabelecer os requisitos mínimos para o controle de microrredes, além de descrever este tipo de instalação, estabelecer requisitos funcionais e a estrutura das funções de controle da microrrede [14].

Em relação ao sistema de controle de uma microrrede, tem-se que:

o sistema de controle de uma microrrede deve contemplar um conjunto de funções de controle que tornam a microrrede auto gerenciável e capaz de operar de forma autônoma ou conectada à rede de distribuição, com capacidade de desligamento ou de conexão à rede para troca de energia e alimentação de sistemas ancilares e auxiliares. Este sistema deve possuir funções de controle em tempo real e de gerenciamento de energia ([5] apud IEEE Std. 2030.7, p. 45).

Assim, é definido em [5], que estas funções devem ser executadas durante as operações isoladas e conectadas da microrrede, de forma a garantir uma transição automática da microrrede para o modo ilhado na ocorrência de anormalidades na rede de distribuição, ou conforme o planejamento interno do gerenciador da instalação, sendo este um tipo de ilhamento intencional planejado. De maneira complementar, durante a operação isolada, faz-se necessário manter o balanço interno de potência na microrrede, de forma a garantir que a demanda das cargas seja atendida de acordo com a geração disponível.

Além disso, é esperado que estas funções permitam a ressincronização e reconexão da microrrede com a rede de distribuição local, de forma a transitar do modo isolado para o conectado. Neste modo, é esperado que o sistema forneça alternativas para o gerenciamento da energia, levando em consideração a geração e consumo de potência ativa e reativa, a alimentação de sistemas auxiliares, e por fim, o suporte à rede de distribuição local, de forma a participar na operação e no mercado de energia, em um contexto de microrrede utilitária, quando for aplicável [5].

Em relação aos requisitos técnicos, assim como as funções do sistema de controle de uma microrrede, é destacado que estes variam conforme sua estrutura, localização e objetivos técnicos e financeiros. Neste caso, a estrutura depende dos dispositivos disponíveis, como geradores, conversores e controladores, enquanto a localização é definida por fatores econômicos e pela disponibilidade de fontes de energia. Já os objetivos são determinados pelo perfil dos consumidores e pelas cargas atendidas, visando benefícios como qualidade de energia, confiabilidade e economia [5] apud IEEE Std. 2030.7.

Ademais, é previsto que alguns requisitos gerais devem ser atendidos, principalmente no que diz respeito ao PCC. De acordo com [5] apud IEEE Std. 2030.7, é esperado que as

microrredes apresentem meios para detecção de ilhamento, para a garantia de operação continuada, mesmo durante perturbações de tensão e frequência, para a manutenção da qualidade de energia, e para o controle da geração e consumo de potência reativa e ativa.

Ao considerar as questões operacionais, a rede de distribuição deve visualizar a microrrede como uma instalação única e controlável, fazendo com que seja necessário a implementação de um sistema de controle que permita esta situação. Este é o caso da implementação dos sistemas de controle centralizados e descentralizados [5] apud IEEE Std. 2030.7.

Por fim, é definido uma estrutura funcional das funções de controle de uma microrrede, sendo ela dividida em 4 níveis, sendo que os níveis inferiores estão subordinados as funções dos níveis superiores [5] apud IEEE Std. 2030.7.

O primeiro nível está relacionado com o controle direto dos dispositivos que compõem a microrrede, de forma a estabelecer os controles relacionados aos níveis de tensão, frequência, reativos, energia armazenada, controle das cargas, detecção de ilhamento e proteção contra falhas [5].

No segundo nível, existem as funções de controle de área local, sendo executadas pelos sistemas de gerenciamento de energia da microrrede, de forma a comandar as características que envolvem o estado e as sequências lógicas das operações internas, o gerenciamento das cargas e dos RED, as ações de ressincronização e o registro de perturbações [5].

Em relação ao terceiro nível, este está vinculado as funções de controle do sistema de supervisão da microrrede, através da previsão climática, da demanda e da geração, mas que também envolvem a visualização, o gerenciamento, e a previsão de dados. Estas funções visam ser utilizadas como forma de otimizar a operação da microrrede, auxiliando em aspectos como o gerenciamento de energia ativa e reativa, o controle de emergências, a suavização da geração e a coordenação dos sistemas de proteção [5].

Por fim, o quarto nível oferece funções de controle interativo com a rede comercial, de forma que estas sejam executadas por operadores do sistema de distribuição ou transmissão. Assim, elas fornecem soluções para a integração da rede com microrredes adjuntas, assim como o controle das redes de distribuição, a implementação de contratos de comercialização, e os sistemas de gerenciamento e supervisão das linhas de distribuição [5].

Dessa forma, estes níveis de funções de controle podem ser organizados em três etapas de controle, de acordo com a sua posição hierárquica, e o domínio de tempo de ação para cada nível, conforme apresentado na Figura 5. Assim, “todo sistema de controle deve possuir

funções que proporcionem a execução de ações de controle apropriadas para cada um dos modos de operação da microrrede” ([5], p. 50), sendo estes modos de operação relacionados ao funcionamento isolado ou conectado em regime permanente, e também durante o período de transição deles.

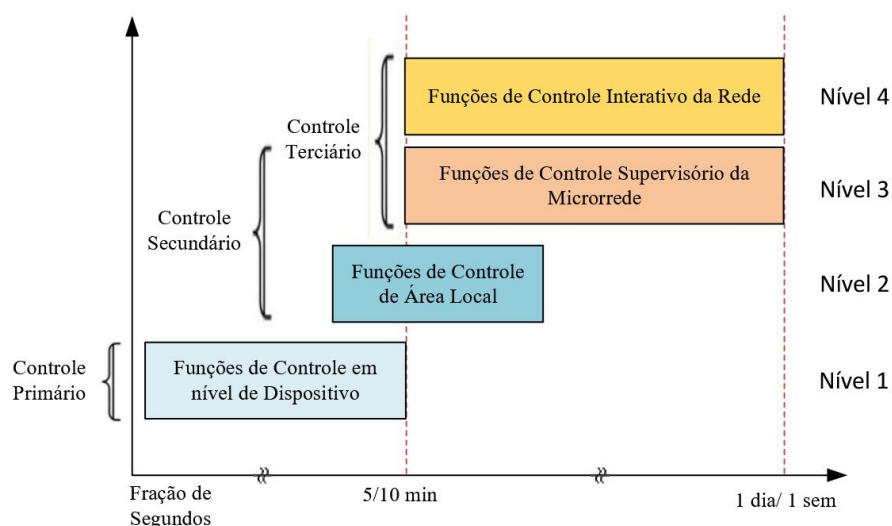


Figura 5 - Estrutura das funções e etapas de controle de uma microrrede
Fonte: [5]

2.4 MODOS DE OPERAÇÃO

Segundo [5] apud IEEE Std 1547.4-2011, uma microrrede pode operar em 3 modos distintos, sendo eles o modo conectado a rede elétrica local, o modo ilhado ou isolado da rede elétrica, e, por último, o modo de transição.

No primeiro modo, em que a microrrede é conectada a rede de distribuição local, as características que envolvem a frequência e tensão de operação da instalação são determinadas pela rede de distribuição, fazendo com que a microrrede tenha que se adequar a estes parâmetros de uma maneira solidamente estabelecida. No caso do Brasil, a ANEEL regula estes parâmetros de acordo com o documento intitulado Procedimento de Distribuição (Prodinst), mais especificamente no módulo 3, que trata da conexão ao sistema de distribuição de energia elétrica. Dessa forma, tem-se a necessidade de que as unidades geradoras possuam meios de se sincronizar com a rede de distribuição, de forma que microrrede consiga injetar e

consumir energia conforme a sua necessidade, respeitando os contratos estabelecidos com a concessionária de distribuição.

No modo de operação isolada, toda conexão existente com a rede de distribuição é cessada, fazendo com que a microrrede continue operando apenas com os RED existentes na instalação. Para isso, é necessário manter um controle constante entre a geração e a demanda das cargas, de forma a manter o sistema estável e seguro durante essa operação. Além disso, é importante que pelo menos um dos RED tenha a capacidade de formar uma rede com características de tensão e frequência que permitam a conexão dos demais componentes geradores de energia, garantindo, assim, a continuidade no fornecimento de eletricidade à instalação [5].

Por fim, o modo de transição consiste em um momento temporário, em que se altera do modo conectado para o modo isolado, e vice-versa. Essas transições, quando realizadas do modo conectado para isolado, são conhecidas como ilhamento, e podem ocorrer de forma intencional, quando é elaborado um plano de desconexão da rede, ou não intencional, quando ocorre uma falta no fornecimento da rede de distribuição. Este último geralmente representa um risco para a microrrede, tendo em vista que se trata de uma desconexão inesperada e os níveis de geração e demanda da instalação podem não estar adequados para a continuidade da operação [5]. Assim, existe uma dependência na rapidez da resposta das unidades geradoras, que podem não ser suficientes quando se tratam de máquinas rotativas, mas que geralmente são supridas por SAE em baterias, até que esses geradores estejam prontos para manter o ilhamento em regime permanente.

Em relação a transição do modo isolado para o conectado, esta é chamada de reconexão, e pode ocorrer sem o desligamento prévio das fontes internas da microrrede, denominada de reconexão sincronizada, ou com o desligamento momentâneo de todas as fontes e cargas da instalação. Dessa forma, com a conexão reestabelecida através de uma chave no PCC, a rede volta a fornecer os parâmetros de sincronização para as fontes geradoras, de forma que a microrrede possa se estabelecer novamente [5].

Com base nisso, entende-se que este processo de transição entre estes modos deve ocorrer de maneira controlada e segura, tendo em vista que durante o ilhamento, deve-se assegurar que não se tenha qualquer fornecimento de energia a rede de distribuição, de forma a garantir que não haja nenhum risco aos possíveis profissionais que podem vir a realizar manutenções nas linhas de distribuição. De forma complementar, entende-se que na reconexão, seja ela sincronizada ou não, os procedimentos sejam realizados adequadamente,

de forma que, ao se conectar com a rede elétrica, não haja a sensibilização dos sistemas de proteção da instalação e da rede.

2.5 MICRORREDE DE ESTUDO

No IFG Campus Itumbiara, encontra-se instalado, no laboratório de Fontes Renováveis, uma microrrede de 3 kW, composta por um sistema de geração distribuída por módulos fotovoltaicos, um sistema de armazenamento de energia em banco de baterias, e um conjunto de cargas prioritárias, referentes ao sistema de iluminação do laboratório. Na Figura 6 é possível visualizar os componentes desta microrrede instalados, cujo objetivo é possibilitar estudos e o desenvolvimento de trabalhos relacionados ao contexto de fontes renováveis de geração de energia, sistemas de armazenamento, extração, análise e controle de dados. Assim, essa estrutura serve de base para a integração de diferentes tecnologias e dispositivos, permitindo a simulação de cenários reais e a avaliação de soluções para a otimização do uso de recursos energéticos e o aumento da confiabilidade no fornecimento de energia elétrica. Uma descrição mais detalhada acerca dos componentes, pode ser observada na Tabela 1.



Figura 6 - Microrrede do Laboratório de Fontes Renováveis
Fonte: Autoria própria

Tabela 1 - Descrição dos componentes da microrrede

Componente	Potência total
Inversor Fotovoltaico Fronius Primo 3.0-1	3 kVA
Módulos BYD335PHK-36 (335 Wp)	4,02 kWp
Inversor Híbrido Victron Multiplus II 48/3000/35-32	3 kVA
Baterias BYD B-Plus 2.5 - U3A1-50P-A (2,5 kWh)	5 kWh
Lâmpadas LED - OSRAM G13 6500K (20W)	0,48 kW

Fonte: autoria própria

2.5.1 Módulos e Inversor Fotovoltaico

Por volta do ano de 2012, o número de sistemas fotovoltaicos como fontes de geração distribuída cresceu de maneira expressiva, alcançando mais de 26 GW de potência instalada no ano de 2023, em mais de 2,3 milhões de unidades geradoras pelo país [15]. Isso ocorreu devido à diversos fatores, como a regulamentação do sistema de compensação de energia, o avanço das tecnologias, e a redução dos preços dos equipamentos, que vinculados ao grande potencial energético do país para este tipo de fonte de geração de energia, permitiu o aproveitamento de terras sem uma função definida anteriormente, além do melhor aproveitamento de espaços ocupados por galpões comerciais, industriais e residências.

Esta tecnologia se baseia no efeito fotovoltaico para a produção de energia elétrica, que depende, basicamente, da utilização de um material semicondutor devidamente tratado, para a captação da radiação eletromagnética emitida pelo sol. Assim, este material, comumente fabricado e comercializado em módulos, consegue realizar a conversão dessa radiação solar em tensão elétrica, que pode ser aproveitada para manter certos equipamentos operando [16].

Porém, esta tensão gerada pode apresentar grande variabilidade, fazendo com que seja necessário a implementação de conversores eletrônicos para realizar a normalização tanto da

tensão, quanto da corrente, de forma a permitir, assim, a conexão de equipamentos. Esses conversores podem ser os controladores de carga, no caso da utilização de sistemas fotovoltaicos com baterias e cargas baseadas em CC, mas também podem ser os inversores, para permitir o suprimento das cargas em CA. Estes últimos equipamentos ainda podem ser autônomos (*off-grid*) ou interativos (*on-grid*), sendo o primeiro utilizado para instalações isoladas, e o segundo para unidades que estejam conectadas a rede de distribuição. Ainda há aqueles inversores conhecidos como híbridos, que realizam ambas as conexões, e permitem tanto a operação isolada, quanto conectada à rede [16].

No caso deste trabalho, tem-se instalado um sistema fotovoltaico interativo, com doze módulos fotovoltaicos de 335 Wp, e um inversor 3 kW de potência máxima de saída, de forma que se tenha um conjunto de seis módulos conectado em cada uma das entradas disponíveis do inversor. Assim, este inversor fica encarregado de converter a energia gerada pelos módulos, de forma que ela possa ser tanto utilizada pelas cargas locais, quanto injetada na rede de distribuição, o que faz com que ele tenha a capacidade de operar em uma tensão fase-neutro de 220 V, a uma frequência de 60 Hz na região em que o campus se encontra. Além disso, estes equipamentos são projetados para se desligarem quando ocorrer a perda da fonte de tensão de referência, que no caso da GD se trata da rede elétrica de distribuição, fazendo com que o dispositivo entre em um modo de segurança que impede a injeção de corrente na rede, e dessa forma, venha a causar um possível acidente.

2.5.2 Inversor Híbrido

Como destacado anteriormente, a implementação de um inversor se faz necessária quando se possui a necessidade de se converter corrente contínua para alternada, de forma a manter o suprimento de energia uma determinada instalação baseada em CA. Atualmente, os sistemas de GD que possuem desta tipologia de equipamento estão restringidos a sincronia com a rede de distribuição, de forma que na ocorrência de uma falta na distribuição, a operação do sistema deve ser imediatamente interrompida até que a normalidade se reestabeleça.

Dessa maneira, o inversor híbrido surge como uma alternativa para essa problemática, de forma que ele consiga manter a operação tanto em paralelo com a rede local, quanto de maneira isolada. Para isso, faz-se necessário que este equipamento apresente uma série de

certificações quanto a sua desconexão completa com a rede nos momento de operação isolada, além da compatibilidade de operação com banco de baterias, de forma que a energia armazenada por estes, seja utilizada para manter a estabilidade da instalação quando operando de maneira isolada. Assim, este tipo de equipamento permite que as baterias sejam carregadas durante um momento de maior geração de uma fonte fotovoltaica, por exemplo, de forma que ela só seja utilizada em um outro momento, seja para manter a energia da instalação durante longos períodos, ou para suportar as cargas momentaneamente, enquanto um gerador rotativo é acionado.

No caso do inversor híbrido destacado na Tabela 1, além de possuir estas características, ele ainda realiza automaticamente as ações de controle que envolvem os modos de transição da microrrede, tanto no ilhamento, quanto na ressincronização, além do gerenciamento no balanço interno de energia quando operando de maneira isolada. Em outras palavras, este equipamento garante que, na ocorrência de uma falta, as cargas serão imediatamente atendidas pelo conjunto de baterias, e a chave conectada as portas de entrada da rede serão automaticamente desconectadas, de forma a impedir qualquer injeção de energia enquanto o suprimento da rede de distribuição não for normalizado.

Além disso, este inversor ainda utiliza da energia armazenada nas baterias para formar uma forma de onda de tensão na sua saída que pode ser utilizado como referência para a ressincronização de inversores fotovoltaicos que estejam conectados a ela. Isso permite que, mesmo com a falta da rede de distribuição, os inversores fotovoltaicos possam continuar operando, desde que estejam conectados na saída do inversor híbrido.

2.5.3 Sistema de armazenamento de energia em banco de baterias

Como apresentado anteriormente, para que seja possível manter a microrrede operando de forma isolada, a aplicação de um SAE baseado em baterias se torna uma alternativa atrativa, devido a sua rápida resposta a essas perturbações, e a alta controlabilidade fornecida pelo seu componente eletrônico de acoplamento.

Assim, diversos tipos de baterias para armazenamento de energia estão disponíveis no mercado, como no caso das baterias de chumbo ácido, níquel-hidreto metálico, e as baterias de íons de lítio. Esta última tecnologia, em que as baterias apresentadas na Tabela 1 são baseadas, foi inicialmente utilizada na década de 1990 em dispositivos eletrônicos, como

celulares, notebooks, computadores e reprodutores de música. Por possuir uma maior densidade de energia e de potência, além de um ciclo de vida superior, e um peso mais reduzido, estas baterias rapidamente se estabeleceram no mercado, mantendo a utilização em componentes eletrônicos, mas também servindo de base para a implementação dos veículos elétricos [17]. Isso se deve ao fato de que, ao comparar as baterias citadas anteriormente, sendo todas do mesmo tamanho, tem-se a baseada em íons de lítio como aquela que possui maior capacidade de armazenar energia e de fornecer potência em relação as demais, além de ter uma duração da vida útil superior, e representar menos peso para as estruturas.

No caso do SAE utilizado neste trabalho, as baterias estão dispostas em dois racks, sendo instalado um único módulo de 2,5 kWh de capacidade em cada um deles, conforme apresentado na Figura 7, totalizando assim, 5 kWh de capacidade total. De maneira complementar, cada rack possui capacidade de suportar uma expansão de mais 3 módulos idênticos ao instalado atualmente, de forma a totalizar 10 kWh por rack. Além disso, estes módulos possuem 51,2 V de tensão nominal, fazendo com que a conexão de cada módulo com o inversor seja em paralelo, de forma que a tensão total do sistema não seja superior a tensão máxima suportada pelo inversor híbrido.

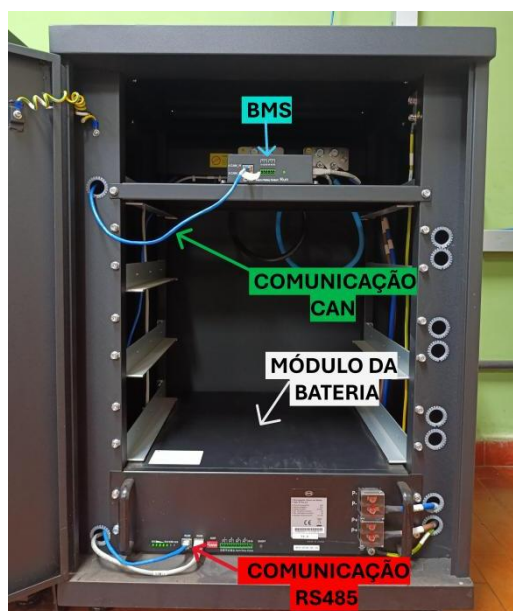


Figura 7 - Região interna do rack de baterias
Fonte: autoria própria

De maneira complementar, o rack ainda possui um sistema de gerenciamento da bateria (BMS, do inglês *Battery Manegement System*), que fornece funções de monitoramento, controle e proteção ao sistema. Para isso, cada módulo deve ser conectado ao barramento de

uma rede Modbus RTU, no padrão RS485, de forma que este barramento seja integrado a unidade de gerenciamento da bateria (BMU, do inglês *Battery Management Unity*). Esta unidade, por sua vez, deve ser conectada ao inversor através de uma rede CAN, conforme apresentado na Figura 8, de maneira que o conversor consiga manter o monitoramento das baterias, supervisionar o estado de carga do conjunto (SOC - do inglês *State Of Charge*), verificar a ocorrência de erros, e comunicar os limites de carregamento e descarregamento das baterias [18][19]. O manual para a confecção do cabo a ser utilizado nesta comunicação, via protocolo CAN, pode ser encontrado em [20].

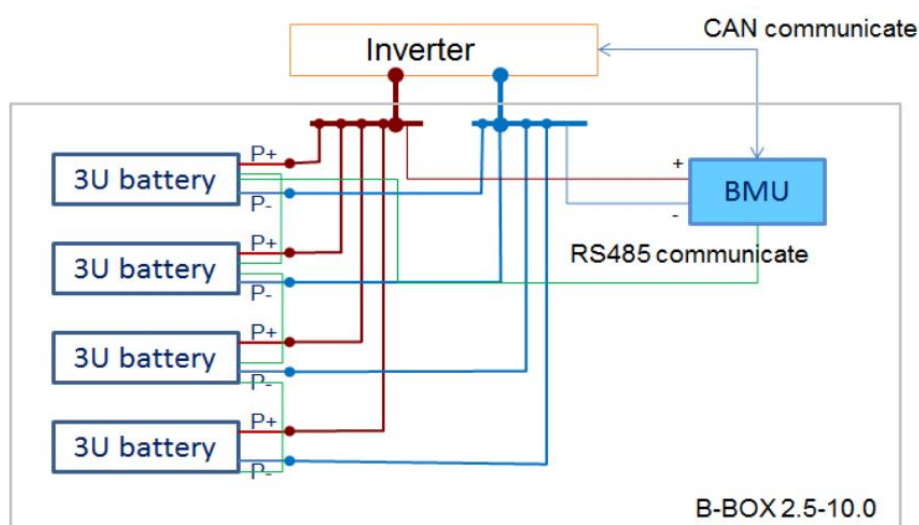


Figura 8 - Diagrama de conexão das baterias
Fonte: [19]

No caso deste trabalho, apesar de cada rack possuir uma unidade de gerenciamento disponível, juntamente com um módulo de bateria, realizou-se a conexão dos módulos em apenas um BMU, de forma que os dois módulos ficassem restritos a mesma unidade de gerenciamento, mesmo que eles se encontrem em racks distintos.

2.5.4 Característica de operação do sistema

Com base nessas informações, na Figura 9 é apresentado o esquema de conexão dos equipamentos descritos anteriormente. Nesta figura, nota-se que o sistema fotovoltaico se encontra instalado em conjunto com as cargas locais, ao mesmo tempo em que ambas estão

conectadas na saída do inversor híbrido. Isso permite que a energia gerada pelos módulos fotovoltaicos seja primeiramente consumida por essas cargas, e que o excedente seja utilizado pelo inversor/carregador para carregar as baterias caso elas estejam descarregadas. Caso o contrário, ela será injetada na rede elétrica da instalação, desde que a microrrede esteja operado no modo conectado. A energia armazenada nas baterias, por sua vez, apenas será utilizada como *back-up*, nos momentos que não houver conexão com a rede de distribuição, e nenhuma outra fonte possuir mais capacidade de fornecer energia para as cargas.

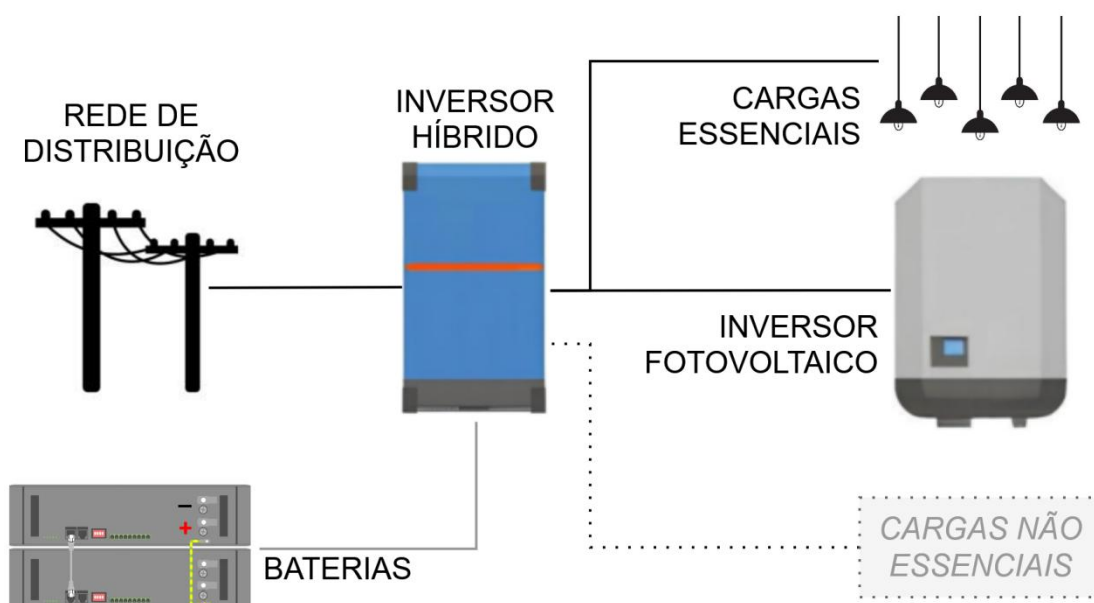


Figura 9 - Esquema de conexão dos componentes da microrrede
Fonte: autoria própria

Dessa forma, nos momentos em que há necessidade de operar de maneira isolada, o inversor híbrido é capaz de formar uma rede de sincronização para o inversor fotovoltaico, de forma que este consiga manter a injeção da energia gerada para as cargas locais. Ademais, é válido destacar que, durante esta operação, faz-se necessário elaborar e manter métodos para o controle dessa injeção do inversor fotovoltaico, tendo em vista algumas situações operacionais que envolvem o gerenciamento automático das cargas na microrrede.

A primeira dessas situações, ocorre quando a geração total é superior a demanda das cargas, e as baterias se encontram totalmente carregadas, fazendo com que se tenha um excedente de geração na instalação. No caso de não haver cargas de desvio para equilibrarem esta relação, o inversor híbrido começa a elevar a frequência na portas de saída de energia, como uma maneira de indicar a necessidade de reduzir a geração local. Esta frequência é incrementada gradualmente até atingir um nível que causa a desconexão automática por

sobrefrequência do inversor fotovoltaico, fazendo com que as baterias sejam imediatamente acionadas para suprir a demanda das cargas. Após isso, o inversor fotovoltaico inicia seu procedimento de reconexão, e caso nenhuma atitude seja tomada, esse processo irá se repetir assim que as baterias estiverem com 100% de carga.

A segunda situação ocorre nos momentos em que não se tem geração do sistema fotovoltaico, ou que ela se encontra reduzida, seja por baixa irradiação solar, ou por limitação da injeção da fonte. Nesta caso, as baterias suprem a demanda da instalação até atingir seu limite de carga, e se até este momento nenhuma outra fonte for despachada, ou se o as cargas não forem reduzidas, a frequência na saída do inversor híbrido será gradativamente decrementada, de forma a informar a necessidade de despachar mais energia para as cargas, ou reduzi-las, até que ela atinja um nível mínimo, ocasionando na desativação de equipamentos como inversores e geradores rotativos por subfrequência.

No caso de a conexão com a rede ser reestabelecida, o inversor híbrido é reconectado a rede de maneira sincronizada, sem a necessidade de desconectar momentaneamente as cargas e as fontes geradoras. A partir disso, o gerenciamento automático da energia passa a contar com a rede de distribuição, de forma que o excedente de energia possa voltar a ser injetado nela, e que a demanda interna também possa ser suprida pela rede, seja para abastecer as cargas ou as baterias.

Dessa forma, tem-se caracterizado todo o processo automático de funcionamento da microrrede instalada no laboratório, fazendo com que o objetivo deste trabalho seja fornecer meios alternativos para realizar a comunicação com os inversores apresentados, de forma a permitir tanto a supervisão, quanto o controle da operação destes equipamentos. Assim, busca-se maneiras de realizar o controle do inversor fotovoltaico para ambos os modos de operação, além da possibilidade de realizar o despacho da energia armazenada nas baterias durante a operação conectada.

3 CAPÍTULO III - COMUNICAÇÃO COM OS INVERSORES

3.1 REDES INDUSTRIAIS

De acordo com [21], quando dois ou mais computadores são capazes de trocar informações entre si, ocorre ali o estabelecimento de uma rede de computadores, tendo em vista que eles estão interconectados por um meio de transmissão específico, como fios de cobre, fibra óptica, ou micro-ondas e ondas de rádio. De maneira complementar, ainda há utilização do modelo cliente-servidor como base para o uso dessas redes, em que um computador definido como cliente, solicita informações a um outro computador caracterizado como servidor, e este será o responsável por atender essa solicitação. Como exemplo, tem-se as aplicações web da internet, em que um servidor oferece acesso a páginas web para um cliente que pode, inclusive, estar alocado em outro continente.

Este conceito de rede de computadores ainda pode ser ampliado ao meio industrial, como definido em [22], em que as redes industriais são apresentadas como aquelas responsáveis por interligarem os computadores aos CLPs (Controladores Lógicos Programáveis). Isso permite um controle mais preciso dos processos que envolvem determinada instalação, de forma que possa ser aplicado tanto em ambientes industriais, quanto em escritórios e setores bancários.

Além de um meio para realizar a troca de dados entre os dispositivos, seja ele físico ou não, essas redes ainda necessitam de um protocolo bem estabelecido de comunicação, de forma que a requisição enviada pelo cliente, seja entendida pelo servidor. Para isso, a ISO (*International Standards Organization*), desenvolveu um modelo base de protocolo baseado em camadas, definido como modelo OSI (*Open Systems Interconnection*), de forma a padronizar o projeto de redes de comunicação a partir de 1977 [22]. Dessa forma, tendo como referência o modelo OSI, cada fabricante seria capaz de criar o seu próprio conjunto de regras para a comunicação, estabelecendo, assim, os diferentes protocolos de comunicação existentes.

Outras redes de comunicação bastante comuns seriam aquelas baseadas na transmissão serial de dados, que consistem, basicamente, na transmissão de informação através de sinais elétricos encadeados. Assim, torna-se possível formar um conjunto de bits que são interpretados como uma determinada informação, que por sua vez, dependem do tipo de protocolo e padrão de comunicação utilizados. Os exemplos mais comuns dessa tipologia de

comunicação pode ser encontrado nos protocolos UART (*Universal Asynchronous Receiver-Transmitter*) e Modbus, através dos padrões TTL e RS232, respectivamente.

No caso das microrredes, tem-se o ambiente perfeito para a implementação dessas redes e protocolos de comunicação industrial, tendo em vista todo conjunto de equipamentos que necessitam de operar de forma conjunta e harmônica para garantir o pleno funcionamento da instalação. Por se tratarem, muitas vezes, de equipamentos distintos, de fabricantes diferentes, faz-se necessário integrar estes dispositivos em suas respectivas redes de comunicação padronizadas, e depois a um servidor, de forma que seja possível realizar comunicação individual com cada um desses equipamentos, e, assim, entender seus aspectos de funcionamento.

Nesta situação, para a realização deste trabalho, implementou-se no laboratório apresentado na Figura 6, duas redes baseadas no protocolo Modbus, sendo uma utilizando o padrão TCP/IP, e a outra o padrão RS232. Assim a primeira dessas redes se tornou a responsável pela comunicação dos inversores apresentados na Figura 9 com o microcontrolador, de forma que a troca de informações entre eles ocorram através de Wi-Fi ou cabos de Ethernet. Já a segunda rede, foi utilizada exclusivamente para a conexão deste mesmo microcontrolador com uma IHM, visando o objetivo de transmitir os dados da primeira rede, para a outra interface, utilizado um meio unicamente físico para este fim. Neste sentido, na Figura 10 é destacada uma visão geral a respeito das redes Modbus implementadas na microrrede objeto de estudo.

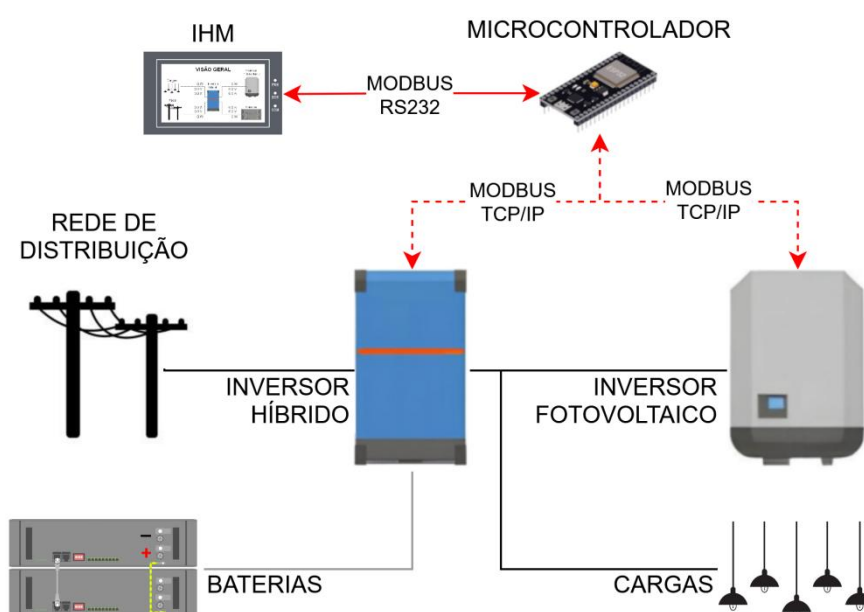


Figura 10 - Esquemático das redes Modbus implementadas na microrrede
Fonte: autoria própria

3.1.1 Protocolo Modbus

O protocolo Modbus, como destacado em [23], foi desenvolvido pela Modicon em 1979 com o propósito de viabilizar a comunicação cliente-servidor entre dispositivos inteligentes. Desde então, este protocolo se tornou amplamente adotado como padrão na indústria devido à sua versatilidade e simplicidade, tendo como um dos principais fatores que impulsionaram sua popularidade, a capacidade de integrar dispositivos de diferentes fabricantes de forma eficiente e padronizada. Além disso, sua independência de taxas de licenciamento, por se tratar de um software livre, contribuiu para sua disseminação global, fazendo com que ele seja uma escolha prática e acessível para sistemas de automação industrial.

De maneira geral, o protocolo Modbus pode ser implementado de acordo com a característica da rede, ou barramento, em que se deseja realizar a troca de informações. Dessa forma, surgem aquelas redes baseadas na transmissão serial entre dispositivos, e outra na arquitetura TCP/IP [24].

3.1.1.1 Modbus Serial

Na utilização de uma rede Modbus baseada na transmissão serial de dados, os dispositivos vinculados à ela podem ser conectados a um barramento no padrão RS232 ou RS485. O primeiro padrão é utilizado na comunicação direta do servidor com apenas um cliente, em redes de até 20 metros, de forma que a informação seja transmitida através de um sinal elétrico esperado entre -25 V e 25 V [25]. Enquanto isso, o segundo padrão é implementado na troca de informações de vários clientes com um único servidor, em distâncias superiores a 1000 metros, graças a utilização de um sinal diferencial que pode variar de -7 V até 12 V [26].

O padrão serial ainda pode ser dividido de acordo com o modo de transmissão das mensagens, que podem ser o RTU ou o ASCII. A diferença entre eles ocorre no encapsulamento e na decodificação da mensagem, sendo que, no primeiro caso, cada conjunto

de 4 bits corresponde a um caractere hexadecimal, enquanto no segundo esse mesmo conjunto corresponde a um caractere ASCII [27].

Assim, nesse padrão de comunicação serial, apenas um dispositivo pode operar como cliente por barramento, de forma que até 247 dispositivos possam ser vinculados a ele como servidores, quando operando no padrão RS485. Dessa forma, para manter a comunicação eficiente com todos esses dispositivos, a comunicação sempre é iniciada pelo cliente, em um processo chamado de transação, de forma que os servidores nunca transmitam dados sem a requisição dele, e nem realizem a troca de informações entre si. Essa transação, por sua vez, consiste em duas mensagens, sendo elas a requisição do servidor, e a resposta do cliente [27].

De maneira complementar, cada servidor deve ser endereçado individualmente, com valores entre 1 e 247 [27], de forma que o cliente consiga identificar aqueles em que se faz necessário iniciar uma transação. Com isso, o cliente se torna capaz de iniciar a troca de informação com um determinado servidor por vez, obedecendo uma Unidade de Dados de Protocolo (UDP) para a formatação da mensagem, que pode ser representada na Figura 11.

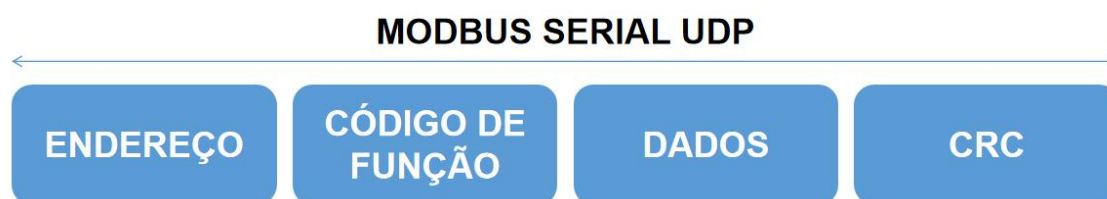


Figura 11 - Modbus Serial UDP
Fonte: adaptado de [27]

Na Figura 11, ao iniciar uma transação o campo destacado como “Endereço” é preenchido pelo cliente com o endereço do servidor de interesse, enquanto o campo “Código de Função” armazena o código que destaca a ação a ser realizada por este servidor no recebimento da mensagem. Esses códigos estão relacionados com o acesso a leitura e escrita de dados, que, por padrão, em sua maioria, podem ser vinculados aos códigos 03 e 06, respectivamente [24].

De maneira complementar, o campo de “Dados” pode armazenar um conjunto variável de informações, a depender do momento da transação, como no caso da requisição de dados do cliente ao servidor, ou da resposta do servidor a essa requisição. No momento da requisição, este campo armazena as informações complementares ao código de função estabelecido, como o endereço do registrador de interesse no servidor, a quantidade de registradores a serem manipulados, e o tamanho total ocupado por este campo, em bytes. Caso

seja uma operação de escrita no servidor, a informação a ser configurada também será enviada neste campo. Por outro lado, no processo de resposta do servidor ao cliente, os dados requisitados serão formatados neste campo, assim como os demais dados solicitados, de forma que o cliente possa extraí-los de maneira adequada [24].

Por fim, o campo “CRC” é utilizado como referência para a verificação de erros, tanto no processo de requisição, quanto na resposta ao cliente, de forma a manter a comunicação da rede operando com o máximo de confiabilidade possível nos dados compartilhados [27].

Com base nisso, para que seja possível implementar uma rede Modbus no modo Serial, ainda se faz necessário realizar a conexão física dos equipamentos, de acordo com o padrão estabelecido em [27]. A forma mais simples deste padrão, conhecida como *half-duplex*, consiste, basicamente, em conectar o cliente ao barramento dos servidores, que por sua vez é implementado tendo como base dois circuitos. Estes circuitos, por sua vez, são descritos como terminais A e B, que na prática estão vinculados com a transação dos dados através de um sinal elétrico diferencial, de forma que a conexão dos componentes da rede seja feita conforme exemplificado na Figura 12. Nesta figura, nota-se a presença de dois resistores “LT”, instalados nas extremidades do barramento como uma alternativa para suprimir possíveis interferências no sinal diferencial.

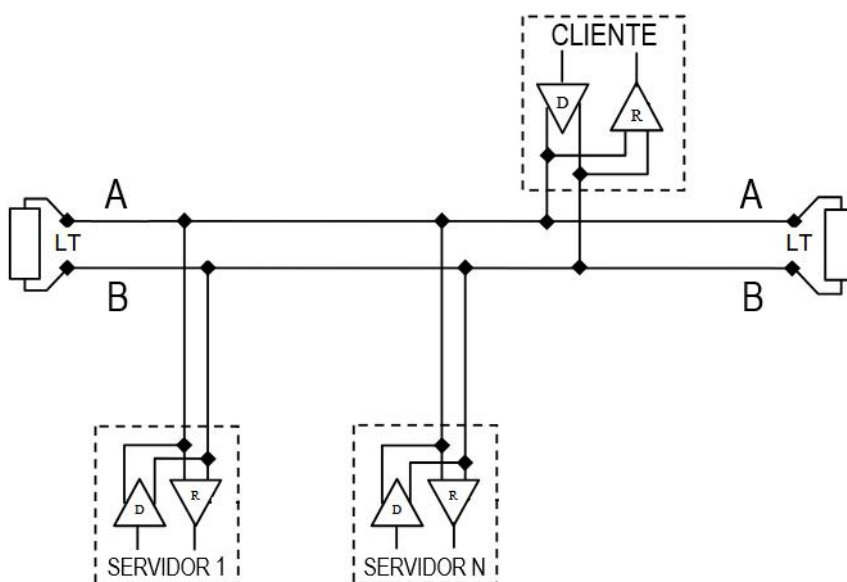


Figura 12 - Conexão dos dispositivos em uma rede Modbus RTU no padrão RS485
Fonte: adaptado de [27]

No caso de uma rede Modbus RTU no padrão RS232, a mesma metodologia apresentada na Figura 12 é aplicada para este tipo de protocolo, porém, a conexão do cliente é feita diretamente em um único servidor.

Portanto, para que uma rede Modbus Serial seja implementada de maneira geral, faz-se necessário garantir que os dispositivos da rede estejam corretamente conectados, seja através do padrão RS232 ou RS485. Além disso, durante a configuração de software desta rede, faz-se necessário conhecer o endereçamento de todos os servidores, assim como a correta utilização dos códigos de função, e natureza dos registradores de interesse para a comunicação.

3.1.1.2 Modbus TCP/IP

Segundo [28], TCP/IP pode ser definido como um conjunto de protocolos que auxiliar na formação da internet, através da estruturação da comunicação entre computadores e servidores. Para isso, faz-se necessário a junção de dois protocolos, sendo eles o Protocolo de Controle e Transmissão (*TCP* do inglês *Transmission Control Protocol*) e o Protocolo de Internet (do inglês *Internet Protocol - IP*). Assim, de maneira geral, o protocolo TCP é encarregado pelo tratamento de dados durante a transmissão de uma informação, enquanto o IP se torna responsável por identificar os dispositivos vinculados a essa transmissão.

Com base nisso, o Modbus TCP/IP se trata de uma adaptação do protocolo Modbus para a aplicação sobre o padrão Ethernet, de forma a utilizar a arquitetura TCP/IP como base para o seu funcionamento. De maneira complementar, diferentemente do padrão serial, o Modbus TCP/IP permite a implementação de mais de um cliente na rede, assim como a ocorrência de transações simultâneas devido a robustez da arquitetura TCP/IP [29]. Assim, essa versão oferece uma alternativa prática e econômica para a troca de dados entre dispositivos, tendo em vista a integração com redes sem fio e a Internet, possibilitando uma comunicação ampla e global entre diferentes equipamentos.

De forma geral, uma rede que utiliza o protocolo Modbus TCP/IP funciona seguindo o mesmo modelo de comunicação cliente-servidor entre os dispositivos, em que, neste caso, a mensagem passa por quatro etapas distintas de processamento, como ilustrado na Figura 13.

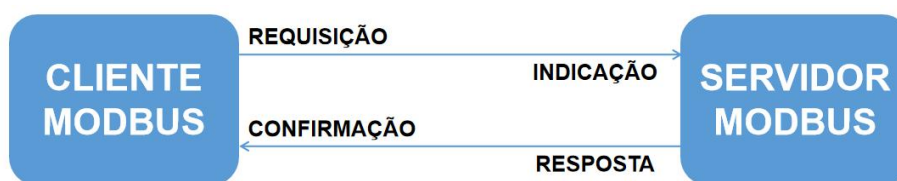


Figura 13 - Tipos de mensagem da comunicação Modbus TCP
 Fonte: autoria própria

Com base na Figura 13, o processo para início da comunicação entre cliente e servidor consiste na etapa conhecida como Requisição, em que o cliente deve iniciar o envio de uma mensagem ao servidor. Ao receber essa mensagem, o servidor a processa durante a etapa de Indicação, realizando as operações necessárias para retornar a mensagem ao cliente na fase de Resposta. Após receber a Resposta, o cliente verifica a transação por meio do procedimento de Confirmação. Caso não sejam identificados erros em nenhuma das etapas, o processo é concluído com sucesso [29].

Adicionalmente, para que a requisição iniciada pelo cliente seja realizada corretamente, é fundamental que certas condições sejam atendidas, permitindo ao servidor emitir uma resposta condizente com o solicitado. Nesse contexto, é essencial compreender a estrutura da mensagem, definida pelo conceito de Unidade de Dados de Aplicação Modbus (ADU, do inglês Application Data Unit). O processo de empacotamento da ADU, ilustrado na Figura 14, é uma etapa crucial e fica a cargo do cliente durante a fase inicial da requisição no protocolo de comunicação Modbus TCP/IP.



Figura 14 - Empacotamento da mensagem na comunicação Modbus TCP/IP
 Fonte: autoria própria

Conforme ilustrado na Figura 14, a primeira seção da ADU corresponde ao cabeçalho do protocolo de aplicação Modbus, conhecido como *MBAP Header (Modbus Application Protocol Header)*, conforme descrito em [29]. Este cabeçalho contém os Identificadores de Transação, Protocolo e Unidade, além do tamanho total da mensagem em bytes. Assim, o Identificador de Transação pode ser descrito como o responsável por determinar se a mensagem se refere a uma Requisição ou Resposta na rede, enquanto o Identificador de Protocolo indica a qual protocolo a mensagem está associada. No caso do Modbus TCP/IP, o

valor 0 é reservado exclusivamente para identificar as mensagens desse protocolo em específico. Por último, o Identificador de Unidade (Unity ID) possibilita a comunicação entre cliente e servidor, utilizando uma sub-rede serial baseada no protocolo Modbus e uma rede Ethernet TCP/IP [29].

O “Código de Função”, assim como na versão serial do protocolo, tem a função de especificar o propósito da transação, podendo ser tanto a leitura quanto a escrita de dados. Conforme descrito em [24], existe um mapeamento completo dos códigos associados ao protocolo Modbus, definidos com base nos modelos de dados disponibilizados por este padrão. No entanto, as funções mais frequentemente utilizadas na troca de informações entre dispositivos são a 03 (*Read Holding Registers*) e a 06 (*Write Single Registers*), pois permitem, de maneira eficiente, a leitura e a escrita de dados armazenados em registradores de 16 bits.

Por fim, o campo de “Dados” é responsável por carregar as informações relevantes durante a transação, da mesma maneira que o padrão serial deste protocolo. Assim, quando a operação envolve a escrita de registros no servidor (código de função 06), o cliente preenche este campo com o valor a ser enviado durante a Requisição, juntamente com as informações adicionais para a operação. Em contrapartida, em uma operação de leitura (código de função 03), o cliente utiliza desse espaço apenas para as informações adicionais do código de função, de forma que o servidor seja o responsável por inserir os dados de interesse durante a operação de Resposta.

Com base nisso, a principal diferença do Modbus TCP/IP para o RTU consiste na identificação dos clientes e servidores da rede, que no caso da versão implementada sobre o padrão Ethernet, depende da configuração do IP dos dispositivos na rede. Neste sentido, compreende-se, então, que o protocolo TCP se torna o responsável por tratar da transmissão dos dados que compõem o ADU, que por padrão são encaminhados através da porta 502. Enquanto isso, o endereço dos dispositivos são gerenciados pelo protocolo IP, que em conjunto com o TCP, fazem com que os dados de requisição de cliente cheguem ao servidor correto durante uma transação do protocolo Modbus TCP/IP.

Portanto, para que uma rede Modbus TCP/IP seja adequadamente implementada, faz-se necessário garantir que os dispositivos da rede estejam corretamente conectados a um roteador ou *switch*, seja por meio de um cabo Ethernet, ou via Wi-Fi. Dessa forma, torna-se possível identificar o IP dos clientes e servidores da rede, de forma a permitir a correta parametrização dos dispositivos que compõem essa tipologia de rede Modbus. Portanto, de maneira análoga a comunicação serial, faz-se necessário o conhecimento do endereçamento

de todos os clientes e servidores de interesse na rede, assim como os códigos de função permitidos, e a natureza dos registradores de interesse.

3.2 IMPLEMENTAÇÃO DA REDE DE COMUNICAÇÃO DOS INVERSORES

Tendo em vista a compatibilidade da comunicação Modbus TCP/IP dos inversores da microrrede apresentada na Figura 9, realizou-se a implementação desta tipologia de rede industrial no laboratório em que os dispositivos se encontram instalados, com o objetivo de permitir a leitura e escrita de parâmetros de operação desses equipamentos. Para isso, utilizou-se de uma plataforma microcontrolada, modelo ESP32 operando como cliente, de forma a ocasionar na formação de uma rede semelhante a presente na Figura 15. Nessa figura, nota-se que a conexão entre os dispositivos ocorreu através de uma rede sem fio, por meio de um roteador instalado no laboratório, viabilizado pela utilização do protocolo Modbus TCP/IP.

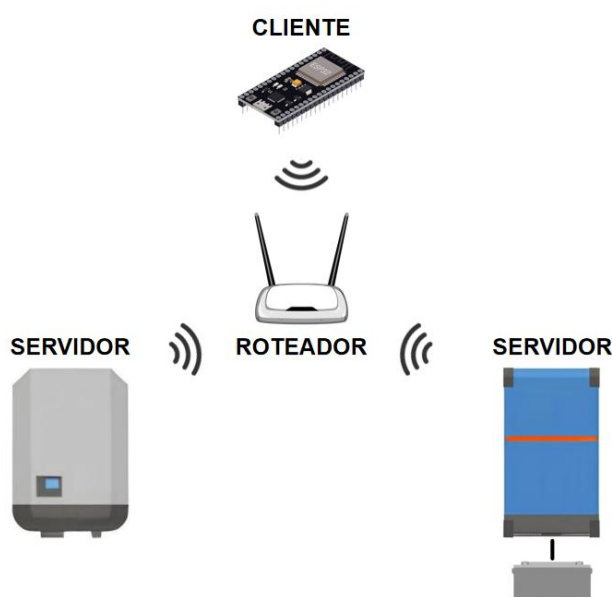


Figura 15 - Representação da rede Modbus TCP/IP estabelecida
Fonte: autoria própria

Nesse contexto, o microcontrolador desempenhou o papel de cliente na rede Modbus, sendo o responsável por realizar solicitações diretamente ao Inversor Híbrido e ao Inversor Fotovoltaico, em relação aos dados de interesse para a supervisão e controle do local. Porém, para estabelecer essa comunicação com os inversores, fez-se necessário realizar certas

configurações com cada um dos dispositivos, de forma a garantir que eles estivessem conectados a rede de comunicação da microrrede. Além disso, tornou-se essencial a identificação do endereço de IP dos dispositivos na rede, da porta destinada à comunicação Modbus TCP/IP, dos registros de memória que contêm os dados de interesse, e das Unity IDs internas desses equipamentos. De forma complementar, também se fez necessário realizar toda a programação do microcontrolador, de forma que ele conseguisse acessar a rede Modbus, e realizar toda a troca de informações com os inversores.

3.2.1 Comunicação com o Inversor Fotovoltaico

3.2.1.1 Pré configuração do equipamento

Para que fosse possível iniciar a comunicação com o inversor fotovoltaico por meio do protocolo Modbus TCP/IP, algumas configurações prévias foram realizadas. A primeira dessas configurações consistia em acessar o painel de configuração do equipamento, que, no caso deste trabalho, esteve disponível em uma interface web acessível com base no endereço de IP do dispositivo na rede.

Desta forma, esta primeira configuração foi realizada ao acionar o ponto de acesso Wi-Fi padrão do equipamento, seguindo as informações disponibilizadas no manual do fabricante, disponibilizado em [30]. Este procedimento para acionar o ponto de acesso consistiu em navegar pelo *display* do equipamento, seguindo o caminho *Setup* > P. de Acesso Wi-Fi > Ativar Wi-Fi AP, até que a interface se assemelhasse a visível na Figura 16.



Figura 16 - Ativação do ponto de acesso Wi-Fi do inversor fotovoltaico
Fonte: autoria própria

Após isso, realizou-se o acesso à rede gerada pelo equipamento através de um outro dispositivo, que neste caso poderia ser um celular ou notebook. As informações a respeito do nome da rede e da senha a ser utilizada foram apresentadas na interface do inversor, conforme destacado na Figura 16.

Após essa conexão, foi acessado o endereço de IP do dispositivo (192.168.250.181) no navegador do dispositivo utilizado para a realização dessa conexão, mas que também poderia ter sido utilizado endereço “http://datamanager”, de forma que o site acessado se tornasse semelhante ao presente na Figura 17.

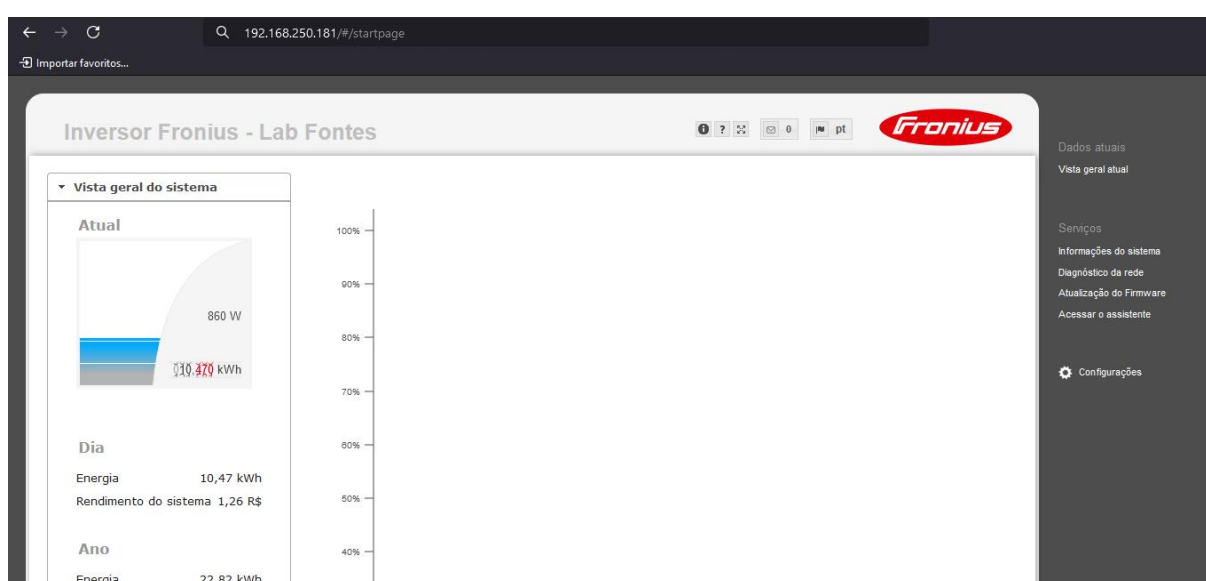


Figura 17 - Interface web de configuração do inversor fotovoltaico
Fonte: autoria própria

Com o acesso a essa interface web de configuração garantido, o próximo passo consistiu em conectar o inversor na rede Wi-Fi que foi utilizada como base para a comunicação Modbus. Nesta situação, o IP do dispositivo na rede foi configurado como estático, de forma a tornar possível que a comunicação entre ele e o microcontrolador prosseguisse de maneira constante, em uma única configuração. Para isso, utilizou-se do caminho Configurações > Rede, que pôde ser iniciado com base na interface apresentada na Figura 17, de forma que a interface final alcançada seja a vista na Figura 18.

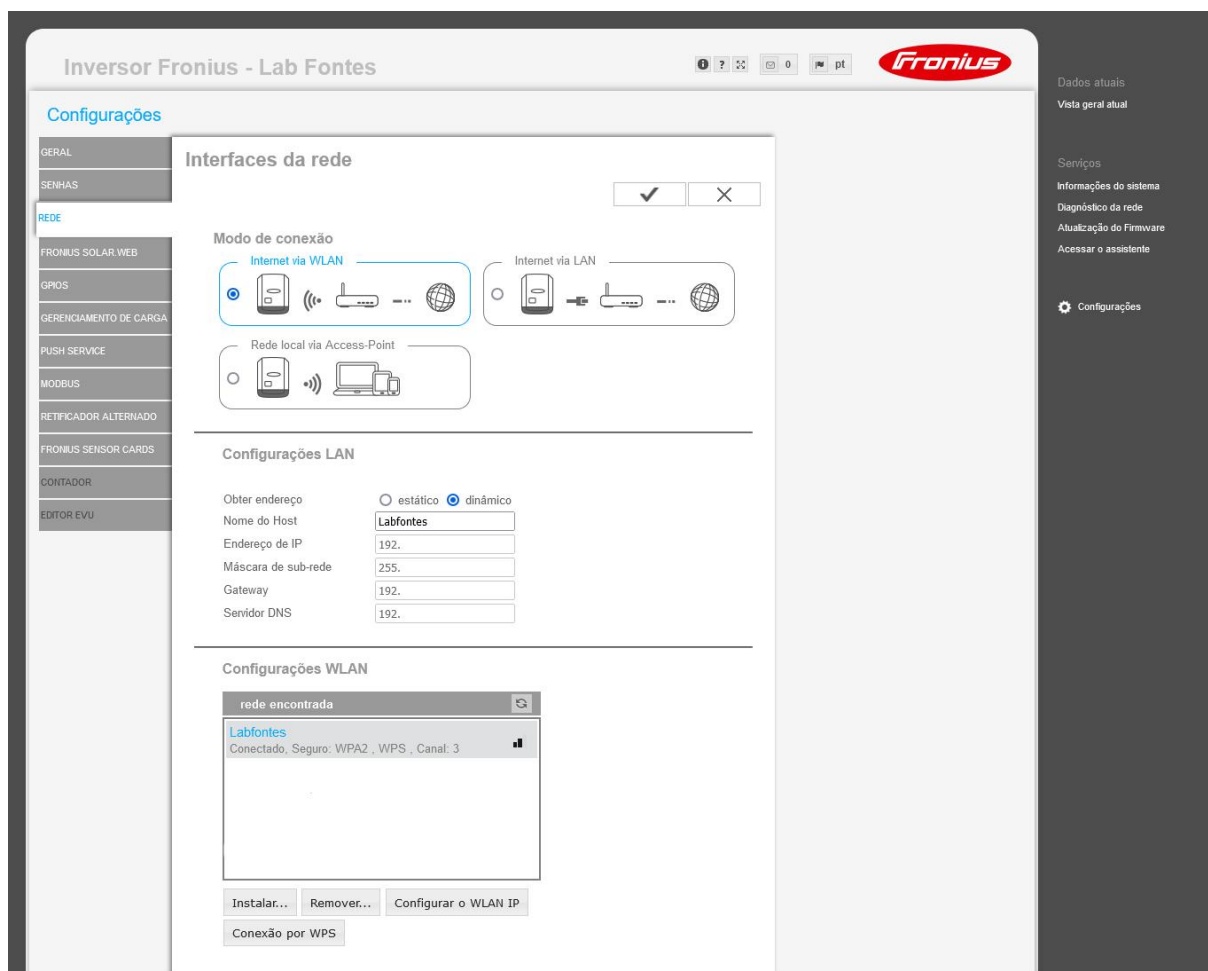


Figura 18 - Interface de configuração de rede do inversor fotovoltaico
Fonte: autoria própria

Com base nisso, as configurações que foram realizadas utilizaram como referência os dados presentes na Figura 18, de forma que o campo “Modo de Conexão” fosse definido para “Internet via WLAN”, enquanto que no campo “Configurações WLAN”, fossem realizadas as configurações referentes a seleção da rede de interesse. Para isso, seguiu-se com a utilização do botão “Instalar...”, de forma que, uma nova interface se tornasse apresentável ao usuário, requisitando o preenchimento da senha da rede de interesse, uma vez que a rede “Labfontes” já havia sido selecionada.

Com essa conexão realizada, ainda na mesma interface de “Configurações WLAN”, realizou-se a definição do IP estático do inversor. Para isso, bastou selecionar o botão “Configurar o WLAN IP”, de forma que uma janela de configuração semelhante a ilustrada na Figura 19 fosse apresentada. Neste caso, alterou-se o campo “Obter endereço” para estático, de forma que os demais campos ficassem livres para a realização da alteração. Dessa forma, o campo “Endereço de IP” foi preenchido com o IP desejado para o inversor fotovoltaico na rede, de maneira que os demais parâmetros estivessem em conformidade com as

configurações do roteador utilizado, ou de acordo com as informações definidas pelo administrador da rede.

Configurar o WLAN IP

Obter endereço ☒ estático ☐ dinâmico

Nome do Host: Labfontes

Endereço de IP: 192.

Máscara de sub-rede: 255.

Gateway: 192.

Servidor DNS: 192.

OK Interromper

Figura 19 - Definição de IP fixo do inversor fotovoltaico
Fonte: autoria própria

Com o acesso estabelecido, o próximo passo consistiu em configurar o inversor fotovoltaico para operar com a comunicação Modbus TCP/IP, que neste caso, pôde ser acessada através do caminho Configurações > Modbus. Dessa forma, na Figura 20 é apresentada a interface final que foi encontrada para este caminho, e nesta figura, pode-se perceber, que alguns parâmetros já haviam sido configurados para viabilizar essa comunicação. Neste sentido, as informações visíveis na Figura 20 se tratam daquelas que foram utilizadas na configuração do sistema do inversor, de forma a viabilizar a comunicação Modbus TCP/IP do equipamento.

Inversor Fronius - Lab Fontes

Configurações

Modbus

Saída de dados através do Modbus ☒ desligado ☒ tcp ☐ rtu

Porta Modbus: 502

String Control Address-Offset: 101

Sunspec Model Type: ☐ float ☒ int + SF

Modo Demo: ☐

Controle do retificador alternado através do Modbus: ☒

Limitar o controle: ☐

Prioridades de controle

1. Controle através do Modbus
2. Receptor do sinal do controle remoto
3. Redução dinâmica de potência

Legenda:
1 ... prioridade alta
2 ... prioridade média
3 ... prioridade baixa

Aviso: uma alteração das prioridades de controle somente é possível no menu do editor EVU com a senha de serviço.

Figura 20 - Interface de configuração da comunicação do inversor fotovoltaico
Fonte: autoria própria

Com base nesta última configuração, concluiu-se a pré configuração do inversor fotovoltaico da microrrede objeto de estudo, de forma que, a partir de então, este equipamento tenha a capacidade se conectar a rede Modbus TCP/IP.

3.2.1.2 Relação de endereço dos registradores

Com a configuração de rede e comunicação do inversor fotovoltaico realizada, o próximo passo consistiu na obtenção da relação dos registradores passíveis de leitura e escrita do equipamento, via rede Modbus. Esses registradores basicamente representam um endereço numérico da memória do inversor, em que se é possível acessar determinada informação a respeito da operação do dispositivo. Por este motivo, é de suma importância que os fabricantes dos equipamentos disponibilizem esta lista, e no caso deste trabalho, toda a documentação a respeito desses registradores para o inversor fotovoltaico pôde ser encontrado em [31]. Além disso, o manual acerca da comunicação Modbus do equipamento também pôde ser encontrado em [32].

Com base nisso, na Figura 21 é apresentado uma amostra da tabela de registradores do inversor fotovoltaico instalado na microrrede, podendo perceber um padrão de informações contidas na figura, em que cada linha corresponde a um único parâmetro do inversor, e suas respectivas informações de acesso estão descritas em cada coluna.

Start	End	Size	R/W	Functioncodes	Name	Description	Type	Units	Scale Factor	Range of values
40072	40072	1	R	0x03	A	AC Current	uint16	A	A_SF	
40076	40076	1	R	0x03	A_SF		sunssf			auto-scaled on A
40094	40095	2	R	0x03	WH	AC Energy	acc32	Wh	WH_SF	
40232	40232	1	RW	0x03 0x06 0x10	Conn	Enumerated valued. Connection control.	enum16			0: Disconnected 1: Connected

Figura 21 - Amostra da tabela de registradores do inversor fotovoltaico

Fonte: Adaptado de [31]

Assim, a primeira e segunda coluna da tabela mostrada na Figura 21, guardam os endereços a serem utilizados na comunicação Modbus, sendo que cada endereço armazena um

conjunto de dados de 2 bytes, ou 16 bits. Quando a informação armazenada necessitar de maior espaço para armazenamento, como no caso do registro 40094, o registro imediatamente posterior é utilizado para guardar o restante dessa informação, de forma que a primeira coluna indique o registro 40094 como início, por exemplo, e o fim como o registro 40095, de forma a ocupar um espaço total (coluna 3) de 2 registradores, ou 4 bytes (32 bits). Como exemplo, o registro 40094 e 40095 devem ser interpretados como se fossem um único registrador que armazena uma informação de 32 bits.

De forma complementar, a 4ª e 5ª coluna indicam qual código de função pode ser utilizado no registro de interesse, sendo que R corresponde a leitura (*Read*) e RW a leitura e escrita (*Read Write*). Por sua vez, estes correspondem, respectivamente aos códigos de função 03 e 06 na comunicação Modbus.

As demais colunas fornecem informações adicionais, como o nome do parâmetro de operação ao qual aquele registro corresponde, uma breve descrição sobre ele, o tipo de dado ao qual ele corresponde, a unidade de medida da informação armazenada, o fator de escala para correção do parâmetro, e o intervalo possível ao qual este dado está restrito, respectivamente.

Assim, no caso da primeira linha na Figura 21, destaca-se que o registro 40072, com tamanho equivalente a um único registrador, corresponde a leitura da corrente na saída do inversor fotovoltaico, e que a informação armazenada ali deve ser interpretada como um dado do tipo inteiro de 16 bits sem sinal. Esta informação deve ser indicada com unidade de medida em Amperes, e que deve ser corrigida com um fator de escala que está armazenado em outro registro, se tratando, então, de um fator de escala dinâmico, e por isso está armazenado em outro registrador, cujo nome corresponde a “A_SF”, e possui endereço 40076. Este fator de escala é indicado como sendo do tipo “sunssf”, e deve ser interpretado como um dado do tipo inteiro com sinal de 16 bits [32]. Na prática, ele armazena um número que deve ser utilizado como o expoente na base 10 a ser multiplicado pelo registro ao qual ele representa o fator de escala. Na Figura 22 é apresentado um exemplo de cálculo desse parâmetro.

Example calculation: (Model 160): 1_DCW = 10000, DCW_SF = -1 -> Power = 10000 x 10⁽⁻¹⁾ = 1000

Figura 22 - Exemplo de correção de escala para registradores do inversor fotovoltaico
 Fonte: [32]

Além disso, uma última informação se faz necessária, que se trata da identificação do *unity ID* a ser utilizada na configuração com o inversor fotovoltaico. Esta configuração,

basicamente, corresponde a identificação do equipamento na rede, de forma que se torne possível identificar inversores específicos em uma instalação que possui diversos dispositivos do mesmo modelo operando.

No caso do inversor fotovoltaico deste trabalho, essa informação a respeito do *Unity ID* do equipamento pôde ser verificada, ou configurada, no *display* do equipamento, conforme apresentado em [32]. Neste manual é destacado que o *Unity ID* deste dispositivo corresponde ao mesmo número de identificação do inversor, com a única exceção do ID de número 0, que neste caso, está relacionado ao *Unity ID* de número 100. Dessa forma, para que fosse possível acessar a interface de configuração desta identificação no inversor fotovoltaico, navegou-se pelo caminho *Setup* > *DATCOM* > Num. do Inversor, até que a tela apresentada se assemelhasse a visível na Figura 23. Assim, conforme apresentado nesta figura, o *Unity ID* utilizado para a comunicação com o inversor fotovoltaico se tratou do número 1.



Figura 23 - Interface de configuração do Unity ID do inversor fotovoltaico
Fonte: Autoria própria

Com base nessas informações, tem-se, como exemplo, que para uma aplicação se tornar capaz de realizar a leitura da corrente na saída do inversor fotovoltaico, faz-se necessário realizar a configuração do cliente Modbus para manter a comunicação com o dispositivo, que por sua vez, possui o IP configurado de acordo com os procedimentos apresentados na Figura 17, e o Unity ID conforme na Figura 23. Após isso, deve-se utilizar a função de leitura de múltiplos registradores (código de função 03) com o endereço 40072, de forma a armazenar a resposta obtida como um dado do tipo inteiro de 16 bits sem sinal. Logo em seguida deve-se realizar a leitura do parâmetro 40076, que corresponde ao seu respectivo fator de escala, e armazenar essa informação como um dado do tipo inteiro de 16 bits com sinal. Por fim, realiza-se o cálculo do valor final da corrente de acordo com o método apresentado na Figura 22, ao multiplicar o valor lido no registro 40072 pelo expoente em base

10 do valor lido no registro 40076. Dessa forma, obtém-se a leitura do valor final da corrente na saída durante a operação do inversor fotovoltaico.

3.2.2 Comunicação com o inversor híbrido

3.2.2.1 Pré configuração do equipamento

De maneira análoga ao inversor fotovoltaico, o inversor híbrido também necessitou de configurações prévias que permitiram a troca de dados entre ele e demais dispositivos em uma comunicação Modbus TCP/IP. Estas configurações, do mesmo modo que na configuração do inversor fotovoltaico, também puderam ser acessadas via interface web, mas de uma maneira um pouco distinta do outro equipamento.

No caso do inversor híbrido, existe um ponto de acesso Wi-Fi que permanece operando junto com o equipamento, conforme apresentado na Figura 24, identificado pelos 5 primeiro caracteres que correspondem a “venus”. Assim, as informações a respeito do complemento do restante do nome da rede, e também da senha a ser utilizada nesta conexão, puderam ser encontradas em uma etiqueta colada na lateral do equipamento, de forma fosse possível estabelecer a conexão de outro dispositivo com esta rede, como um celular ou notebook.

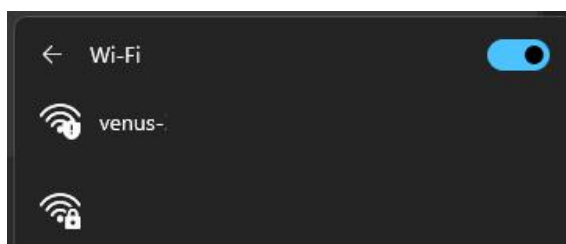


Figura 24 - Ponto de acesso Wi-Fi gerado pelo inversor híbrido
Fonte: autoria própria

Com a conexão realizada com a rede de interesse, e para prosseguir com os procedimentos de configuração do inversor híbrido, fez-se necessário acessar, via navegador, o endereço “venus.local”, de forma que a interface apresentada se tornasse parecida com a visível na Figura 25.

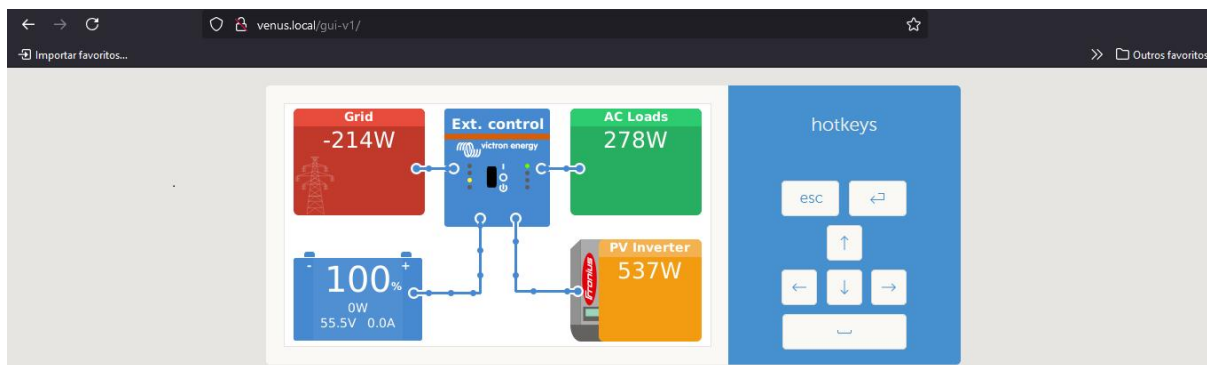


Figura 25 - Interface de configuração web do inversor fotovoltaico
 Fonte: autoria própria

Após o acesso a interface de configuração web do equipamento, destacada na Figura 25, o próximo passo consistiu na conexão do dispositivo com a rede Wi-Fi em que a comunicação Modbus TCP/IP com o inversor fotovoltaico foi implementada. Para isso o caminho *Menu > Settings > Wi-Fi > Wi-Fi Networks* foi utilizado, de forma que a interface web se localizasse como na Figura 26.

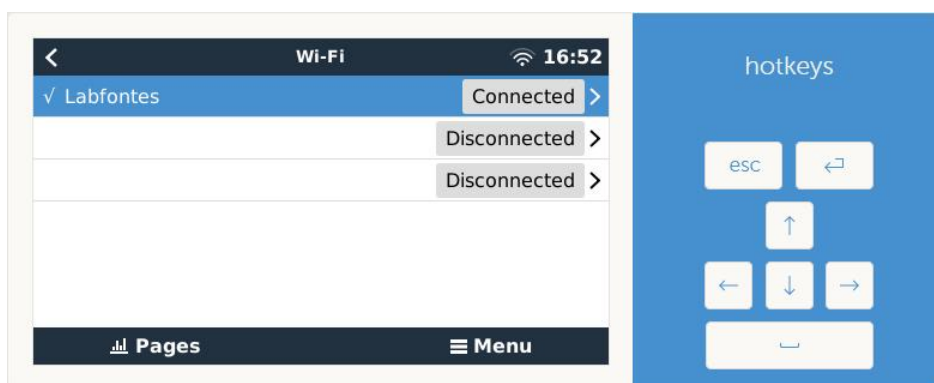


Figura 26 - Interface de configuração Wi-Fi do inversor híbrido
 Fonte: autoria própria

Na Figura 26, nota-se que todas as redes disponíveis para conexão do equipamento foram listadas, restando, assim, apenas escolher a rede de interesse e seguir com a configuração, de forma que as informações contidas na Figura 27 fossem apresentadas.

Nas interface de configuração apresentada na Figura 27, fez-se necessário definir a senha da rede no campo “*Password*”, a configuração de IP para “manual” no campo “*IP configuration*”, e o novo endereço de IP fixo do dispositivo campo “*IP address*”. As demais informações que envolvem a configuração da *Netmask*, do *Gateway* e do *DNS Server* foram preenchidas automaticamente após a realização da conexão do dispositivo.

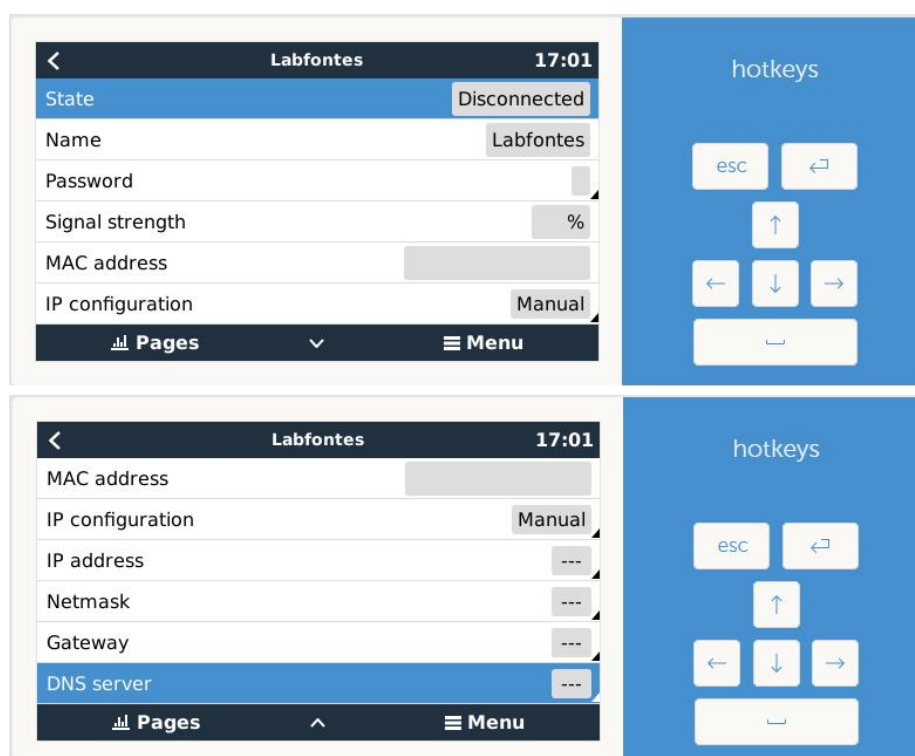


Figura 27 - Configuração de IP estático do inversor híbrido
 Fonte: autoria própria

Com esta configuração de conexão do inversor híbrido realizada, a interface de configuração do equipamento pôde ser novamente acessada, através da utilização do novo IP do equipamento no navegador de internet. Destaca-se, em tempo, que a partir desta configuração, a informação a respeito do IP do equipamento se tornou fixa no *display* físico do equipamento, desde que a rede em que ele foi conectado estivesse disponível. Outra maneira de acessar essa interface de configuração, seguindo o mesmo raciocínio, foi através da utilização do endereço “venus.local” novamente, ao invés de utilizar do endereço de IP diretamente no navegador.

Após este acesso, realizou-se a configuração do equipamento para que ele operasse com compatibilidade a comunicação Modbus TCP/IP. Para isso, o caminho *Settings > Services > Modbus TCP > Enable Modbus/TCP* foi seguido, de forma que a interface alcançada fosse visível como na Figura 28.

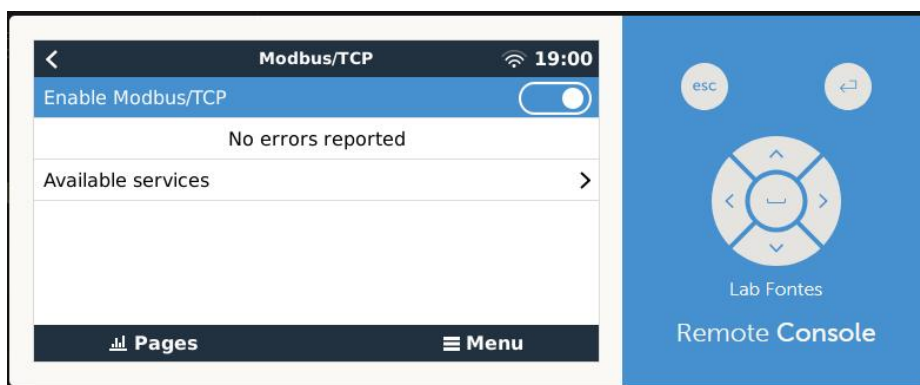


Figura 28 - Interface de configuração da comunicação Modbus do inversor híbrido
Fonte: autoria própria

Dessa forma, realizou-se a pré configuração do inversor híbrido na microrrede objeto de estudo deste trabalho, de forma que, a partir de então, este equipamento tenha a capacidade de se comunicar com outros dispositivos via protocolo Modbus TCP/IP.

3.2.2.2 Relação de endereço dos registradores

De maneira análoga ao procedimento de verificação dos registradores do inversor fotovoltaico, o mesmo foi realizado para o inversor híbrido. No caso do equipamento que foi utilizado neste trabalho, essas informações acerca dos endereços, assim como do *Unity ID* puderam ser encontradas no site do fabricante, ao acessar a opção “*Modbus-TCP register list*” [33].

Assim, na Figura 29 é apresentado uma amostra dos dados extraídos dessa tabela, de forma que na primeira coluna, tenha-se o campo reservado para o nome do serviço ao qual o respectivo registrador está vinculado, sendo esta uma informação necessária para se identificar o *Unity ID* deste registro. Isso se deve ao fato de que no mesmo arquivo indicado em [33], existe uma outra tabela que estabelece a relação entre os equipamentos conectados ao inversor híbrido, e a respectiva *Unity ID* necessária para interagir com esses dados, conforme apresentado na Figura 30. De maneira complementar, os *Unity IDs* disponíveis para o inversor híbrido puderam ser acessados na interface “*Available services*”, apresentada na Figura 28.

NOTE: Use unit-id 100 for the com.victronenergy.system data, for more information see FAQ.						ModbusTCP Register list		
dbus-service-name	description	Address	Type	Scalefactor	Range	writable	dbus-unit	Remarks
com.victronenergy.vebus	Input voltage phase 1	3	uint16	10	0 to 6553.5	no	V AC	
com.victronenergy.battery	Battery power	258	int16	1	-32768 to 32767	no	W	
com.victronenergy.settings	ESS control loop setpoint	2700	int16	1	-32768 to 32767	yes	W	ESS Mode 2 - Setpoint for the ESS control-loop in the CCGX. The control-loop will increase/decrease the Multi charge/discharge power to get the grid reading to this setpoint

Figura 29 - Amostra da Tabela de Registradores do Inversor Híbrido
Fonte: Adaptado de [1.30]

Unit ID	/DeviceInstance	Remark
100	0	VE.Can device instance 0 and all registers listed as com.victronenergy.system
225	512	CAN-bus BMS
228	275	EasySolar-II/Multiplus-II GX VE.Bus port (ttyS3)

Figura 30 - Amostra da tabela de mapeamento dos Unity IDs do Inversor Híbrido
Fonte: Adaptado de [1.30]

Com base nisso, observou-se que, para acessar as informações dos registradores cuja a primeira coluna na Figura 29 correspondessem ao endereço “com.victronenergy.system”, ou então a “com.victronenergy.settings”, o *Unity ID* de número 100 destacado na Figura 30 deveria ser utilizado. Assim, de forma genérica, identificou-se que essa *Unity ID* estava relacionada aos registradores que abrangem algum aspecto de configuração do sistema interno do inversor híbrido, como no caso do controle do despacho da energia armazenada nas baterias. Por outro lado, ao considerar os endereços “com.victronenergy.vebus” e “com.victronenergy.battery”, correspondentes aos *Unity IDs* 228 e 225, respectivamente, percebeu-se que eles estavam associados a todos os dados de operação do inversor híbrido, e das baterias.

Dessa forma, diferentemente do inversor fotovoltaico, foi necessário se atentar as múltiplas *Unity IDs* vinculadas ao inversor híbrido, pois, caso contrário, não se tornava possível realizar as operações de leitura e escrita neste equipamento, mesmo que as outras configurações estivessem corretas.

Em relação a segunda coluna na Figura 29, é possível encontrar uma breve descrição sobre o dado armazenado naquele registrador, que por sua vez, tem o endereço apresentado logo em seguida, na terceira coluna, seguido pelo tipo de dado armazenado, na quarta coluna. Na mesma figura, a coluna 5 apresenta o fator de escala do registrador, sendo que o valor armazenado ali representa o denominador de uma fração. Nesta fração, o numerador dela deve ser substituído pelo valor armazenado no seu respectivo registrador, de forma que a divisão deste por aquele, resulte no valor final da informação de interesse. Assim, diferentemente da

metodologia de correção de escala do inversor fotovoltaico, o inversor/carregador possui como característica a utilização de um fator de escala fixo para cada registrador, de forma a tornar o processo de leitura de um parâmetro mais simplificado. De forma complementar, a coluna 6 destaca o intervalo possível para o dado armazenado em cada registrador, levando em consideração a sua respectiva correção de escala.

Além disso, a sétima coluna apresenta a informação referente a possibilidade de escrita daquele parâmetro, de forma que, em caso positivo, torna-se possível de utilizar os códigos de função 03 e 06. Ademais, em caso negativo, apenas as operações que envolvem o código 03 podem ser realizados com este registro em específico. Por fim, as duas últimas colunas descrevem a unidade de medida da informação armazenada, e algumas informações importantes sobre aquele registro.

Dessa forma, para que uma aplicação seja capaz realizar a leitura da potência das baterias conectadas ao inversor híbrido, faz-se necessário realizar a configuração do cliente Modbus para iniciar a comunicação com o dispositivo que possui o IP configurado de acordo com os procedimento que resultaram na Figura 27, e o Unity ID conforme na Figura 30. Após isso, deve-se utilizar a função de leitura de múltiplos registros (código de função 03) com o endereço 258, de forma a armazenar a resposta obtida como um dado do tipo inteiro de 16 bits como sinal. Por fim, realiza-se o cálculo do valor final da potência de acordo com a divisão do valor armazenado anteriormente com o seu respectivo fator de escala, destacado na Figura 29. Dessa maneira, tem-se estabelecido o procedimento de obtenção do valor final da potência de operação das baterias conectadas ao inversor híbrido, que no caso deste trabalho, foi ampliado para todas as informações passíveis de leitura do inversor fotovoltaico.

3.2.3 PROGRAMAÇÃO DA PLATAFORMA MICROCONTROLADA

Como apresentado na Figura 15, uma plataforma microcontrolada modelo ESP32 DevKit de 39 pinos foi utilizada como cliente da rede Modbus, de forma que ele se tornasse o responsável por iniciar as operações de leitura e escrita dos dados de operação dos dispositivos que compõem a microrrede.

De maneira geral, os microcontroladores podem ser descritos como dispositivos eletrônicos cuja operação é definida por códigos gravados em sua memória, necessitando apenas de uma fonte de alimentação externa para funcionar. Esses componentes integram

CPU, memórias PROM e RAM, terminais de entrada e saída, além de outros periféricos, variando em recursos, tamanho e formato conforme o fabricante. Dessa forma, a construção desses dispositivos possui características que fornecem tecnologia compacta e integrada, fazendo com que o uso dela seja mais simples e econômica em diversas aplicações [34].

Entre os diversos modelos disponíveis no mercado, o ESP32, desenvolvido pela Espressif, destaca-se por ser projetado para aplicações como dispositivos móveis, *wearables* e IoT. Este equipamento possui Wi-Fi integrado, eliminando a necessidade de adaptadores, e pode ser alimentado via USB, dispensando o uso de fontes externas. Além disso, oferece conectividade Bluetooth e outras especificações técnicas que ampliam sua versatilidade [35], de forma a se tornar o equipamento perfeito para ser aplicado neste trabalho. Na Figura 31 é possível visualizar a estrutura da plataforma microcontrolada utilizada no desenvolvimento deste projeto.

Para realizar a configuração deste tipo de equipamento, de forma que ele consiga realizar as funções desejadas, diversos ambientes de programação podem ser utilizados, sendo que, no caso deste trabalho, o editor Arduino IDE foi utilizado. Inicialmente, este ambiente de desenvolvimento foi criado para dar suporte aos hardwares *open source* da empresa, baseados na CPU AVR [36], porém, rapidamente ele pôde ser aplicado para outros microcontroladores, tendo em vista a facilidade de programação neste ambiente, através da utilização da linguagem C/C++.

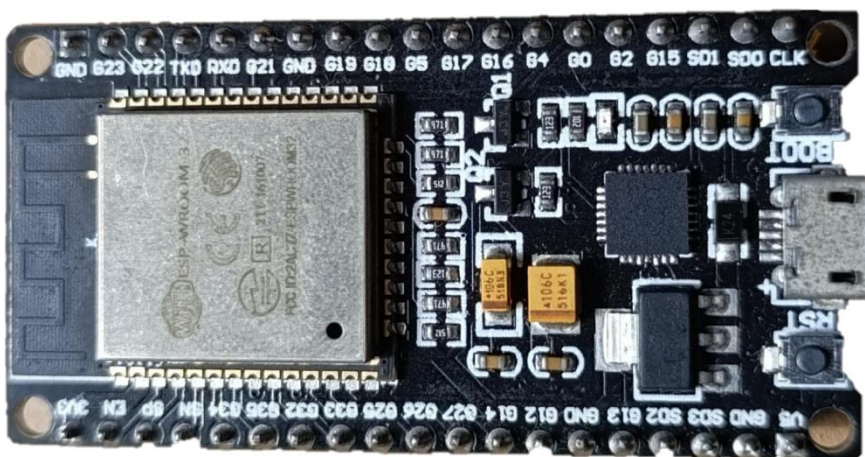


Figura 31 - Microcontrolador modelo ESP32 utilizado no trabalho
Fonte: autoria própria

3.2.3.1 Conexão com a rede Wi-Fi e Estrutura de dados

Com base nisso, para que fosse possível integrar o microcontrolador apresentado na Figura 31 na rede Modbus TCP do laboratório, foi necessário a utilização dos recursos de conexão Wi-Fi da plataforma. Assim, tornou-se possível realizar a conexão deste dispositivo na rede formada pelo roteador, assim como a comunicação direta com os equipamentos configurados nas seções 3.2.1 e 3.2.2.

Para isso, utilizou-se a biblioteca para conexão Wi-Fi, denominada “WiFi.h”, que pode ser encontrada em [37], e que permite que os microcontroladores, no geral, consigam se conectar a uma rede local, seja através de recursos próprios da plataforma, ou através de módulos externos. No caso do microcontrolador apresentado na Figura 31, tem-se a compatibilidade de fábrica com este tipo de operação, de forma a tornar mais facilitado a implementação do microcontrolador na rede de comunicação Modbus sem fio.

Dessa forma, na Figura 32 é apresentado o fluxograma do algoritmo utilizado para a conexão na rede Wi-Fi do laboratório, de forma que, ao iniciar a operação do microcontrolador, ele fosse capaz de se conectar, de maneira automática, na rede Wi-Fi configurada. Para isso, utilizou-se, como referência, os parâmetros como nome (SSID) e senha da rede sem fio do laboratório, que, por sua vez, foram vinculados apenas uma vez ao código do programa no ambiente do Arduino IDE, conforme apresentado no APÊNDICE A. Na decorrência de alteração de qualquer um desses parâmetros, faz-se necessário realizar a edição do código da plataforma microcontrolada, de forma a configurar os novos parâmetros de conexão de rede ao dispositivo.

Ademais, antes de efetivamente iniciar a comunicação com os inversores via rede Wi-Fi, fez-se necessário, primeiramente, garantir que o microcontrolador conseguisse armazenar todas as variáveis de leitura e escrita dos dispositivos de maneira correta. Como destacado na Figura 21 e na Figura 29, cada registrador do equipamento possui um conjunto complexo de informações que devem ser observadas para conseguir efetivamente interpretar o dado armazenado, como a quantidade de registros, o fator de escala, e o tipo de dado armazenado a ser lido ou escrito na operação. Dessa forma, foi necessário garantir que o algoritmo desenvolvido tivesse capacidade para realizar todas as operações necessárias para manter a confiabilidade no manejo dos dados que envolviam a comunicação entre os dispositivos que compõem a rede Modbus do laboratório.

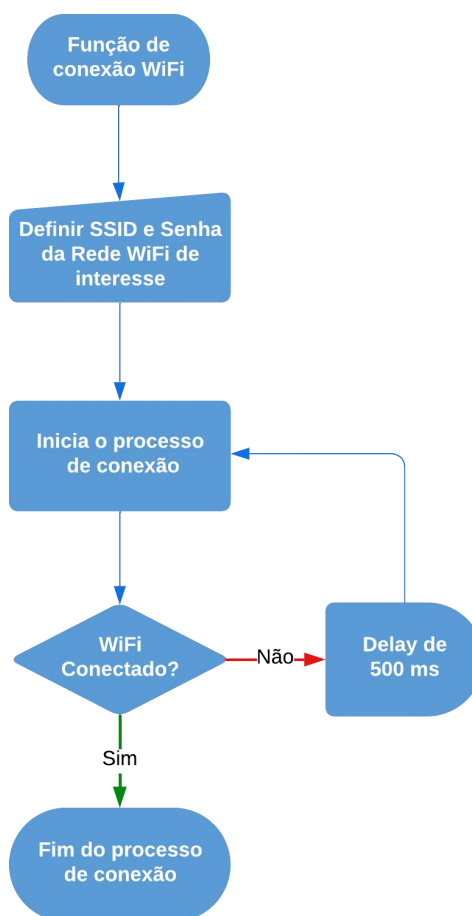


Figura 32 - Fluxograma da função de conexão Wi-Fi do microcontrolador
Fonte: autoria própria

Para que este objetivo fosse alcançado, optou-se pela utilização de Estruturas no algoritmo deste trabalho, tendo em vista que, de maneira simplificada, elas representam variáveis, ou estruturas, que são compostas por diversas outras variáveis de tipologias distintas, conhecidas como componentes [38]. Assim, tornou-se possível, como exemplo, a definição de uma estrutura do tipo “dados”, que pôde possuir como componente, uma variável que correspondesse ao endereço do registrador como um número inteiro sem sinal. Ao mesmo tempo, essa mesma estrutura também possuía um outro componente vinculado a ela, de maneira independente da anterior, como uma variável que indica se este mesmo registrador pode ou não adquirir um fator de escala dinâmico, através da lógica booleana. Um exemplo ilustrativo da declaração dessa estrutura do tipo “dados” pode ser encontrado na Figura 33.

```

estrutura dados {
    componente1 (inteiro sem sinal);
    componente2 (booleano);
};

```

Figura 33 - Ilustração da estrutura "dados" utilizado no trabalho
Fonte: autoria própria

Com base nisso, foi desenvolvido um vetor de estruturas para armazenar os dados dos registradores de cada inversor, seguindo o exemplo apresentado na Figura 34. Assim, este vetor foi baseado em uma estrutura do tipo “dados”, que, por sua vez, possuía como componentes os parâmetros apresentados na Tabela 2. Dessa forma, cada posição deste vetor de estruturas correspondia a uma linha da tabela de registradores dos equipamentos, apresentadas na Figura 21 e na Figura 29.

Tabela 2 - Componentes da estrutura utilizada no algoritmo

Componente	Tipo	Descrição
registro	uint16	Armazena o endereço do registrador
qtd_Reg	uint16	Quantidade de registradores em que a informação está armazenada
FS_Dinamico	bool	<i>True</i> : O respectivo registro possui fator de escala dinâmico <i>False</i> : O respectivo registro não possui fator de escala dinâmico
FS	int16	Armazena o fator de escala como expoente de base 10 (Figura 22)
UNITY_ID	uint8	Unity ID ao qual o registrador está vinculado
escrita	bool	<i>True</i> : Função de leitura e escrita permitida <i>False</i> : Apenas função de leitura permitida
valor[2]	uint16	Armazena a informação da leitura ou escrita do registrador
sinal	bool	<i>True</i> : A informação armazenada pode adquirir valor negativo <i>False</i> : A informação armazenada possui apenas valor positivo

Fonte: autoria própria

```

dados vetorDados[3] = [

|                                 |
|---------------------------------|
| componente1 (inteiro sem sinal) |
| componente2 (booleano)          |
| componente1 (inteiro sem sinal) |
| componente2 (booleano)          |
| componente1 (inteiro sem sinal) |
| componente2 (booleano)          |

];

```

Figura 34 - Ilustração da utilização de um vetor de estruturas
Fonte: autoria própria

De maneira complementar, para que o microcontrolador tivesse a capacidade de armazenar essas informações, foi desenvolvido uma função dentro dessa estrutura, denominada de “addReg”, que por sua vez, tinha como objetivo a adição de novos componentes a estrutura de interesse. Assim, o algoritmo utilizado por essa função pode ser encontrado na Figura 35, e consiste, basicamente, em receber os dados dos parâmetros apresentados na Tabela 2, e armazená-los em seu respectivo componente na estrutura.

Destaca-se, em tempo, que apesar de os dois inversores apresentarem metodologias distintas para cálculo da correção de escala, apenas a metodologia apresentada na Figura 22 foi utilizada. Isso ocorreu devido ao fato de que essas duas metodologias ocasionavam em um padrão diferente para armazenar a informação de fator de escala no componente “FS” apresentado na Tabela 2, tendo em vista a existência de um vetor de estruturas para o inversor fotovoltaico, e outra para o inversor híbrido. Dessa forma, como uma maneira de padronizar o algoritmo para correção de escala dos dados de ambos os equipamentos, estabeleceu-se que aquela metodologia apresentada na Figura 22 seria a metodologia de referência. Assim, a informação apresentada na Figura 29, no que diz respeito ao fator de escala do registrador, foi convertida para o padrão presente na Figura 22, de forma que o mesmo cálculo pudesse ser realizado tanto para as informações do inversor fotovoltaico, quanto para o inversor híbrido.

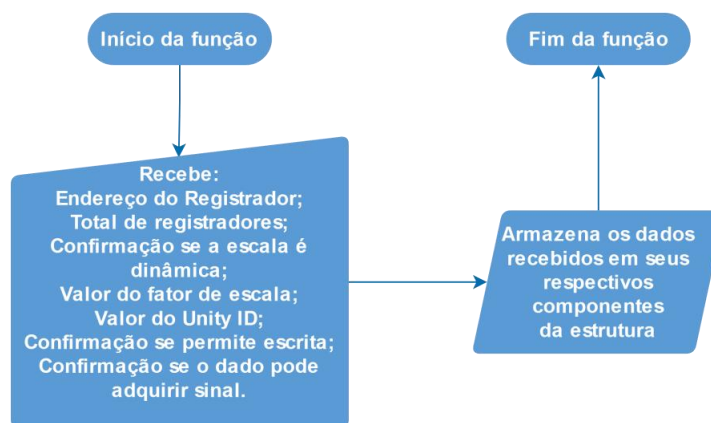


Figura 35 - Fluxograma da função "addReg"
Fonte: autoria própria

Portanto, os dados de cada registrador foram preenchidos manualmente através da utilização das funções “def_Reg_FR” e “def_Reg_VE”, que podem ser visualizadas no código completo do microcontrolador, destacado no APÊNDICE A. Assim, essas duas funções foram implementadas com o objetivo de adicionar os dados dos registradores do inversor fotovoltaico e do inversor híbrido, respectivamente, através da utilização da função “addReg”.

3.2.3.2 Comunicação com os inversores

Com a estrutura de armazenamento das informações da comunicação com os inversores definida, restou-se, a partir disso, implementar o algoritmo que permitisse realizar a transação de dados entre estes equipamentos e o microcontrolador. Para isso, foi utilizado a biblioteca “ModbusIP_ESP8266.h”, que permite que uma plataforma microcontrolada, de maneira geral, se comunique via rede Modbus, seja no padrão serial ou TCP, como cliente ou servidor. Os arquivos de instalação para esta biblioteca, assim como toda documentação necessária para utilização da mesma, podem ser encontrados em [39].

Com base na documentação da biblioteca instalada, o algoritmo do fluxograma destacado na Figura 36 foi desenvolvido, que por sua vez, apresenta um resumo do código utilizado para manter o microcontrolador operando como cliente na rede Modbus TCP/IP implementada no laboratório. Com a utilização do código deste fluxograma, que pode ser encontrado no APÊNDICE A, tornou-se possível o estabelecimento da troca de dados entre o microcontrolador e os dois inversores. Assim foi possível utilizar do estabelecimento desta comunicação para atualizar os componentes da estrutura desenvolvida com base nos dados presentes na Tabela 2.

Tendo em vista as informações que foram apresentadas na Figura 36, destaca-se as funções “ler_mb” e “esc_mb”, que representam, respectivamente, as funções criadas no algoritmo para realizar os procedimentos de leitura e escrita de dados na comunicação com os inversores. Para a utilização de ambas as funções, foi necessário a utilização das informações de identificação dos dispositivo cuja a conexão se fazia necessária, como o IP deles na rede, o *Unity ID*, e os registradores de interesse. Além disso, as estruturas utilizadas para armazenar os dados de leitura e escrita também foram utilizados como parâmetros para essas funções, de forma que o processo de alocação dos dados dessa comunicação ocorresse de maneira mais direta. Assim, na Figura 37 é descrito o algoritmo da função criada para realizar a leitura de registradores na comunicação Modbus. Por outro lado, na Figura 38 é destacado a função para o processo de escrita desses parâmetros.

No caso da função de leitura do fluxograma apresentado na Figura 37, esta foi desenvolvida tendo em vista uma limitação da biblioteca disponibilizada em [39], em que o procedimento de leitura de múltiplos registradores só podia retornar um conjunto máximo de informações referentes a 99 registros. Neste sentido, os parâmetros de registrador inicial e registrador final necessários para a utilização da função, foram utilizados para fornecer a

informação necessária para que a função conseguisse realizar mais de uma transação nas situações em que se fizesse necessário ler mais de 99 registradores. Assim, o código implementado nesta função, permitiu que um conjunto de vários registradores fossem lidos em quantas transações se tornassem necessárias, ao mesmo tempo que as informações obtidas nessa comunicação sejam armazenadas nos seus respectivos componentes da estrutura declarada.

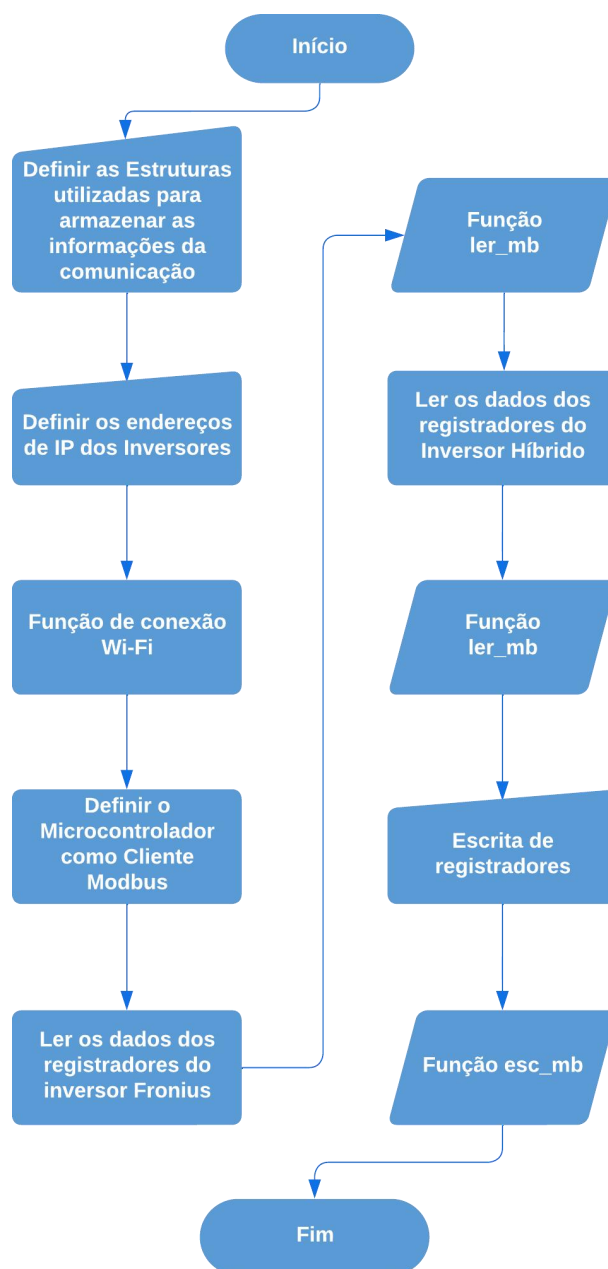


Figura 36 - Fluxograma da comunicação Modbus do Microcontrolador
Fonte: autoria própria

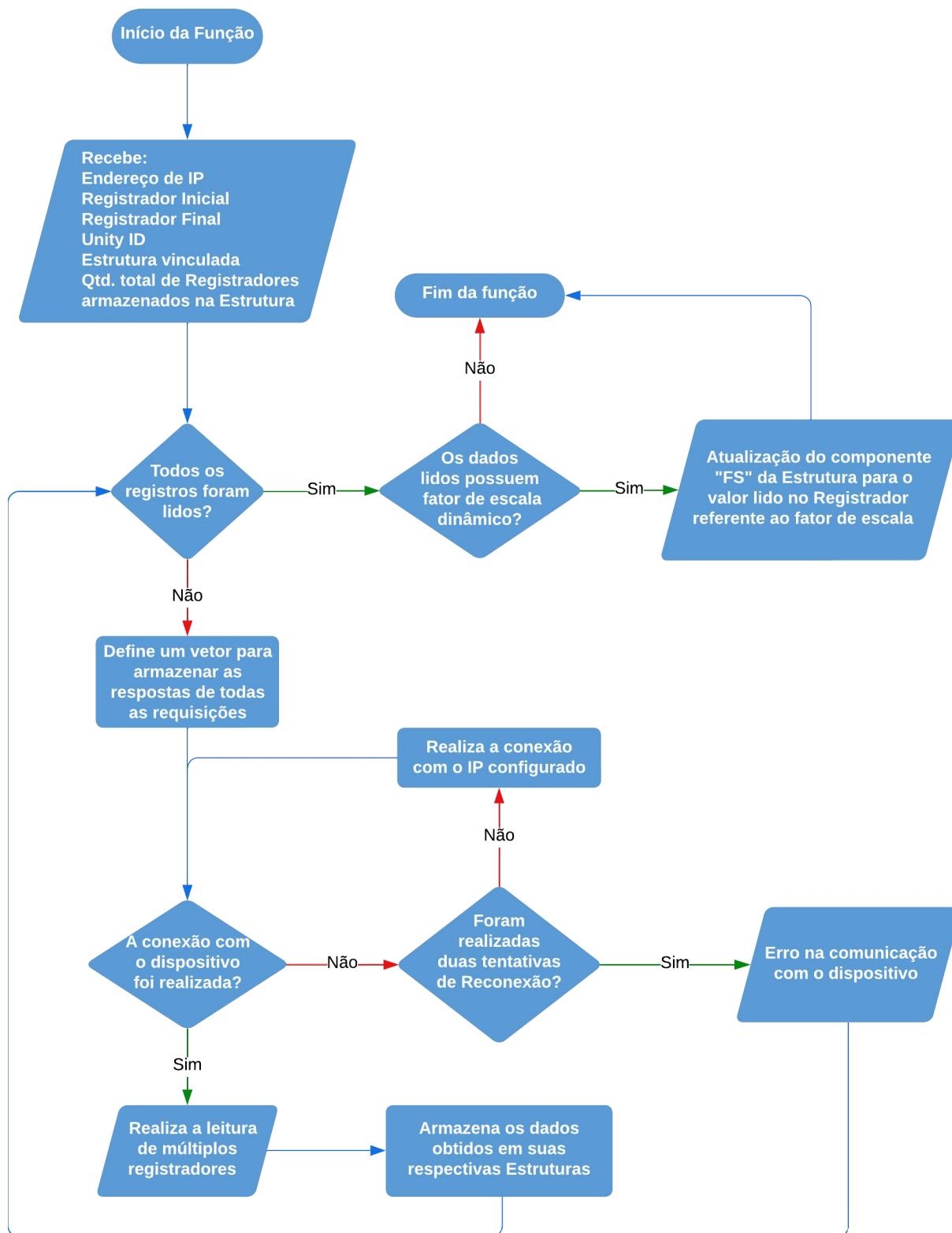


Figura 37 - Fluxograma da função de leitura na comunicação Modbus
Fonte: autoria própria

Já em relação a função de escrita, ilustrada na Figura 38, foi desenvolvido um procedimento para a escrita de um único registrador por vez, tendo em vista que o número total de registradores que se teve a necessidade de realizar esta operação, foi naturalmente

mais reduzido. Com isso, o código implementado teve a capacidade de realizar a vinculação do endereço do registrador fornecido, com a estrutura ao qual a sua respectiva informação foi armazenada, tendo como base o endereço de IP do dispositivo. Assim, o algoritmo se tornou capaz de diferenciar entre a estrutura vinculada ao inversor híbrido, e aquela vinculada ao inversor fotovoltaico, ao mesmo tempo que também conseguisse verificar se o registrador fornecido realmente possuía compatibilidade com a função de escrita na comunicação Modbus. Caso esta situação fosse positiva, o processo inverso da metodologia de correção de escala para cada equipamento era utilizado, de forma que a informação de interesse fosse escrita de maneira adequada no respectivo inversor.

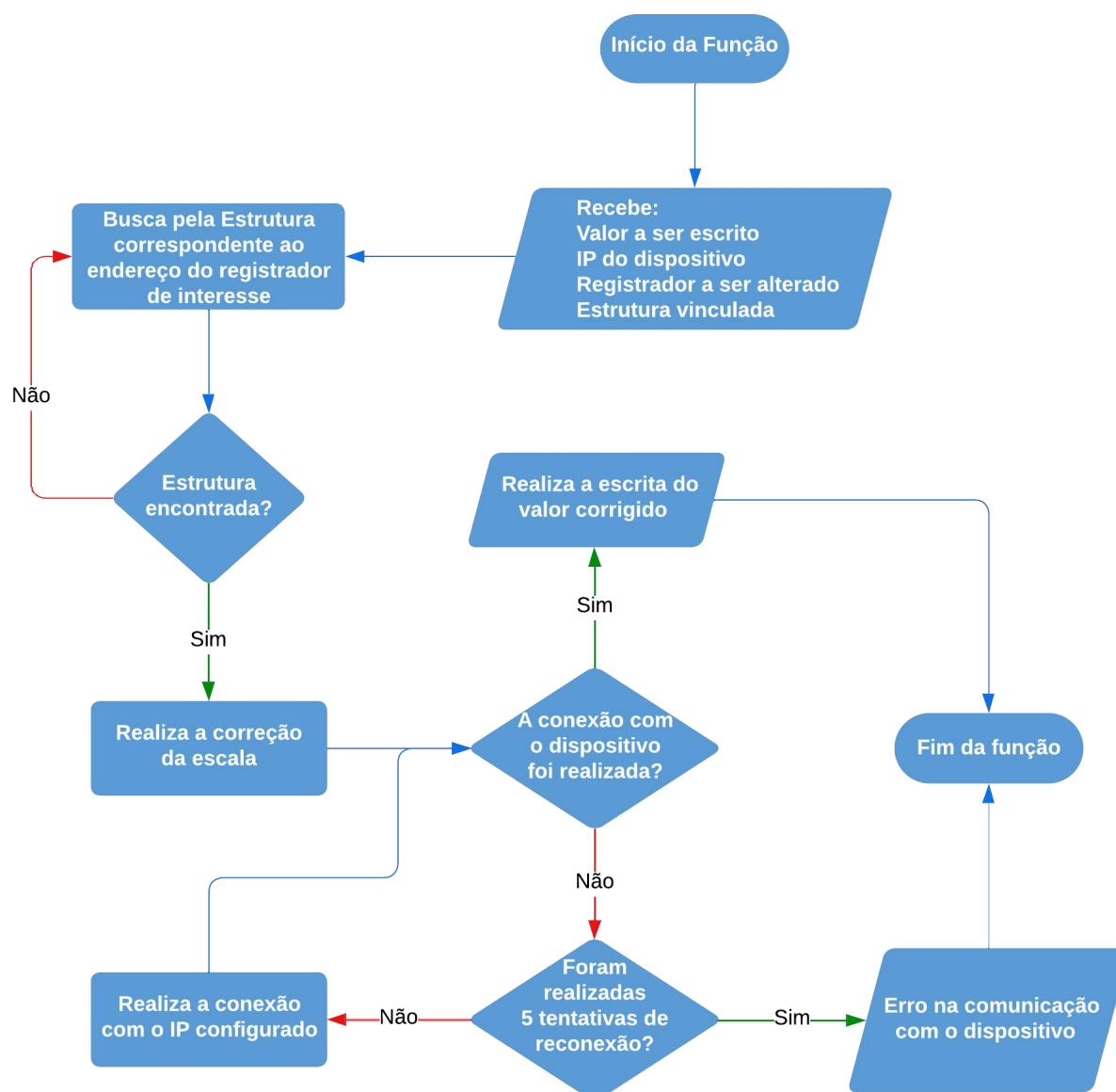


Figura 38 - Fluxograma da função de escrita na comunicação Modbus
 Fonte: autoria própria

Portanto, com base nas informações apresentadas, foram realizadas todas as configurações necessárias para que o microcontrolador conseguisse operar como cliente na rede Modbus TCP/IP. Com essas configurações, ele se torna capaz de realizar as operações de leitura e escrita de múltiplos registradores tanto do inversor fotovoltaico, quanto do inversor híbrido, através de uma rede sem fio, e assim, armazenar esses dados na própria memória do dispositivo.

4 CAPÍTULO IV - COMPARTILHAMENTO DE DADOS

Com base nos dados obtidos com a comunicação dos inversores que compõem a microrrede, através da integração do microcontrolador com a rede Modbus TCP/IP, a próxima etapa deste trabalho consistiu na apresentação dessas informações em uma interface para a interação com o usuário. Para isso, adotou-se, primeiramente, o protocolo de comunicação MQTT como método para a transferência dos dados obtidos na comunicação com os inversores, de forma que esses dados se tornassem disponíveis para acesso posterior da interface de supervisão e controle. Assim, este capítulo apresenta um panorama geral a respeito dos procedimentos utilizados para estabelecer uma rede MQTT no laboratório em que a microrrede se encontra operando.

4.1 PROTOCOLO MQTT

De acordo com [40], o MQTT (*Message Queuing Telemetry Transport*) foi desenvolvido pela IBM (*International Business Machines Corporation*) na década de 1990, como um protocolo de comunicação entre máquinas (M2M), visando uma solução leve e simples para esta aplicação. Atualmente, ele é amplamente utilizado em aplicações de Internet das Coisas (IoT), podendo operar com dois tipos principais de dispositivos, sendo eles os clientes e o *broker*. Os primeiros são os dispositivos responsáveis por enviar e receber dados, enquanto o segundo atua como um servidor intermediário entre os dispositivos que enviam e recebem essas informações, de forma a organizar estas mensagens em tópicos específicos. Assim, o *broker* apenas permite que uma mensagem seja transmitida de um cliente a outro, caso eles estejam inscritos no mesmo tópico em que a mensagem foi inicialmente publicada.

Além disso, dentro desse modelo de comunicação, os clientes ainda podem ser caracterizados como *publishers*, nas situações em que eles realizam o envio de dados ao *broker*, ou como *subscribers*, nos momentos que recebem e processam informações específicas recebidas do servidor. De maneira complementar, ainda pode haver aqueles dispositivos que operam das duas maneiras, hora publicando informações em tópicos, mas também processando informações advindas deles. Neste sentido, na Figura 39 é exemplificado o funcionamento de um sistema submetido ao protocolo MQTT, em que os

dispositivos são divididos entre clientes e *broker*, e que no caso daqueles, podem ainda serem caracterizados como *publisher* e/ou *subscribers*.

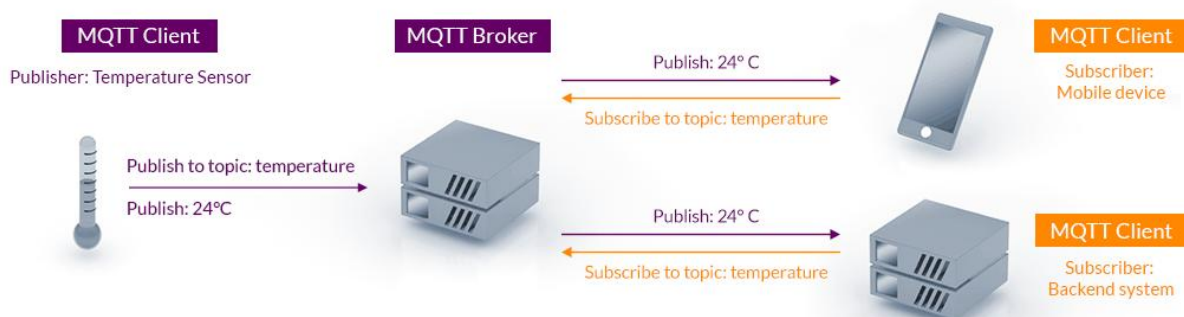


Figura 39 - Exemplo de implementação de uma rede MQTT

Fonte: [59]

Portanto, entende-se que, para realizar a publicação e leitura de uma determinada informação extraída de um sistema, deve-se, primeiramente, garantir a conexão de todos os dispositivos de interesse com o *broker* da rede MQTT. Após isso, deve-se estabelecer um tópico comum a esses dois dispositivos, de forma que o *broker* compreenda que toda informação publicada nesse tópico, seja repassada para os dispositivos que aguardam essa informação.

4.1.1 Implementação da rede MQTT

No contexto deste trabalho, implementou-se um *broker* local em um computador existente no laboratório de Fontes Renováveis, de forma que o *broker* se tornasse acessível via rede Wi-Fi. Dessa forma, o microcontrolador passou a ter a capacidade de publicar, em tópicos específicos, os dados coletados da comunicação Modbus, assim como receber determinados conjuntos de parâmetros que estão relacionados as operações de escrita de registradores nos inversores. Posteriormente, esses dados encaminhados via rede MQTT foram acessados pela interface de interação do usuário, que por sua vez, se tornou o responsável pela apresentação das informações da microrrede, assim como do recebimento de comandos externos.

Para viabilizar essa funcionalidade, utilizou-se do software *Open-Source* Eclipse Mosquitto, que disponibiliza a instalação de um *broker* gratuito, com suporte para as versões 5.0, 3.1.1 e 3.1 do protocolo MQTT [41]. O arquivo executável para a instalação pode ser encontrado em [42], sendo que, no caso deste trabalho, a versão para Windows de 64 bits foi utilizada. É válido destacar a necessidade de se conhecer o caminho utilizado para a instalação do *broker*, tendo em vista que esta pasta deve ser posteriormente acessada para a realização de algumas configurações.

Após seguir com a instalação padrão do arquivo executável, a próxima etapa consistiu em definir a porta de operação em que o *broker* deve aguardar pelo recebimento de qualquer informação. Para isso, foi acessado o arquivo “mosquitto.conf”, disponibilizado na pasta definida para a instalação, conforme apresentado na Figura 40. Neste arquivo, obteve-se o acesso a todas as configurações de operação do *broker*, de forma que a parametrização desse servidor tenha ocorrido através da edição das linhas de código presentes no próprio arquivo. Para isso, utilizou-se de um outro software específico para a realização da edição deste tipo de arquivo, que neste contexto, tratou-se do “Notepad++”, disponibilizado para download em [43].

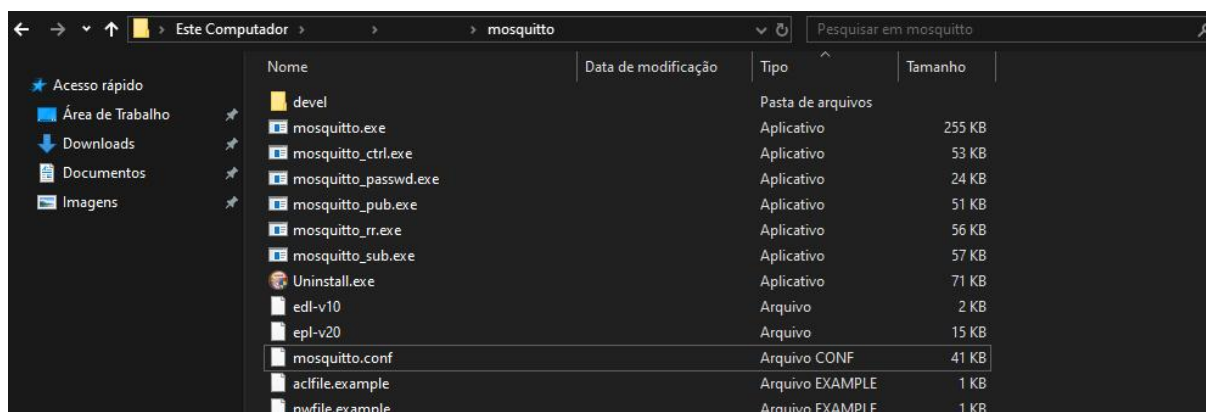
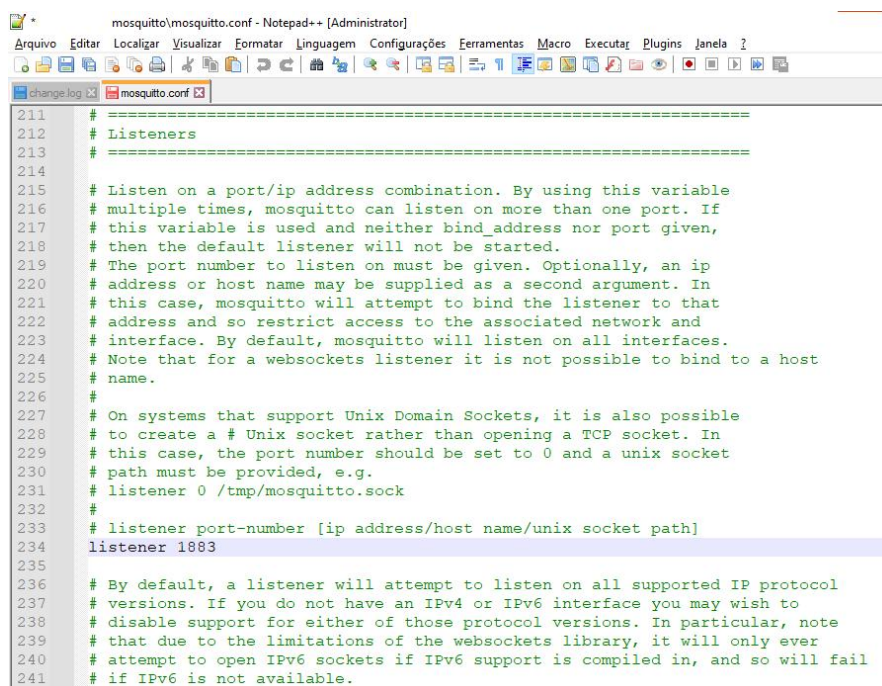


Figura 40 - Arquivo de configuração do Mosquitto Broker

Fonte: autoria própria

Dessa maneira, após realizar o acesso do arquivo de configuração do *broker* com o software Notepad++, navegou-se até a linha 234, e a alterou para a opção “*listener 1883*”, conforme ilustrado na Figura 41, de forma que a definição da porta 1883 tenha sido realizada como porta padrão para a operação do *broker*.



```

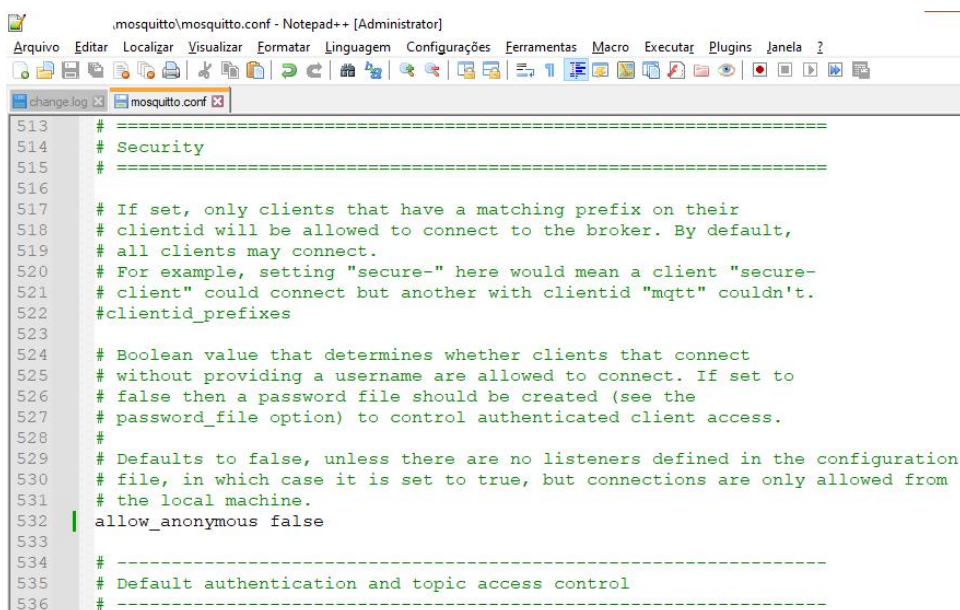
211 # =====
212 # Listeners
213 # =====
214
215 # Listen on a port/ip address combination. By using this variable
216 # multiple times, mosquitto can listen on more than one port. If
217 # this variable is used and neither bind_address nor port given,
218 # then the default listener will not be started.
219 # The port number to listen on must be given. Optionally, an ip
220 # address or host name may be supplied as a second argument. In
221 # this case, mosquitto will attempt to bind the listener to that
222 # address and so restrict access to the associated network and
223 # interface. By default, mosquitto will listen on all interfaces.
224 # Note that for a websockets listener it is not possible to bind to a host
225 # name.
226 #
227 # On systems that support Unix Domain Sockets, it is also possible
228 # to create a # Unix socket rather than opening a TCP socket. In
229 # this case, the port number should be set to 0 and a unix socket
230 # path must be provided, e.g.
231 # listener 0 /tmp/mosquitto.sock
232 #
233 # listener port-number [ip address/host name/unix socket path]
234 listener 1883
235
236 # By default, a listener will attempt to listen on all supported IP protocol
237 # versions. If you do not have an IPv4 or IPv6 interface you may wish to
238 # disable support for either of those protocol versions. In particular, note
239 # that due to the limitations of the websockets library, it will only ever
240 # attempt to open IPv6 sockets if IPv6 support is compiled in, and so will fail
241 # if IPv6 is not available.

```

Figura 41 - Configuração da porta de operação do broker

Fonte: autoria própria

Com a configuração realizada na Figura 41, a próxima etapa consistiu na definição de um usuário e de uma senha para que os clientes conseguissem acessar o broker. Para isso, realizou-se, primeiramente, a alteração da linha 532 do arquivo “mosquitto.conf” para a configuração “*allow_anonymous false*”, conforme estabelecido na Figura 42.



```

513 # =====
514 # Security
515 # =====
516
517 # If set, only clients that have a matching prefix on their
518 # clientid will be allowed to connect to the broker. By default,
519 # all clients may connect.
520 # For example, setting "secure-" here would mean a client "secure-
521 # client" could connect but another with clientid "mqtt" couldn't.
522 #clientid_prefixes
523
524 # Boolean value that determines whether clients that connect
525 # without providing a username are allowed to connect. If set to
526 # false then a password file should be created (see the
527 # password_file option) to control authenticated client access.
528 #
529 # Defaults to false, unless there are no listeners defined in the configuration
530 # file, in which case it is set to true, but connections are only allowed from
531 # the local machine.
532 allow_anonymous false
533
534 # -----
535 # Default authentication and topic access control
536 # -----

```

Figura 42 - Configuração inicial para autenticação de usuário do broker

Fonte: autoria própria

Assim, a etapa seguinte para a implementação da autenticação do *broker* ocorreu com a criação de um arquivo contendo as informações de usuário e senha de acesso para o servidor MQTT. Para que isso fosse possível, fez-se necessário a utilização do *prompt* de comando do Windows como administrador, de forma que o processo consistisse, inicialmente, em navegar até a pasta de instalação padrão do *broker*, de acordo com o apresentado na Figura 43.

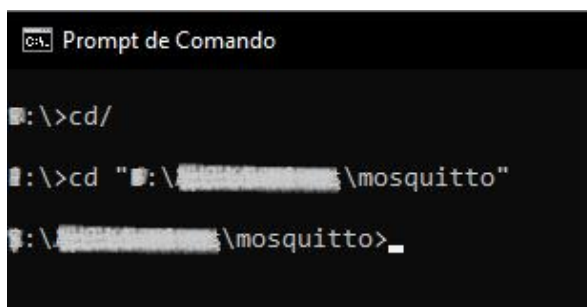


Figura 43 - Acesso da pasta de instalação padrão do *broker* via terminal
Fonte: autoria própria

Após o acesso da pasta via terminal de comando, criou-se um arquivo para armazenar as informações de usuário e senha de acesso ao *broker*, através do comando “type nul > arq_senha”. Assim, tornou-se possível a criação de um arquivo com o nome de “arq_senha”, que por sua vez, foi utilizado para armazenar as informações de acesso ao *broker*, mas que, de maneira complementar, pode ser configurado com outros nomes, de acordo com o desejado pelo usuário.

Dessa forma, após a criação do arquivo, executou-se, logo em seguida, o arquivo de configuração de senhas do *broker*, através do comando “mosquitto_passwd.exe -c arq_senha usuario”. Neste comando o nome utilizado em “usuario”, correspondeu ao efetivo nome do usuário de acesso, e após a confirmação da execução do comando, uma nova linha foi apresentada para a inserção da senha de acesso para este perfil. Posteriormente, foi solicitado confirmação dessa senha no campo “*Reenter password*”, conforme destacado na Figura 44.

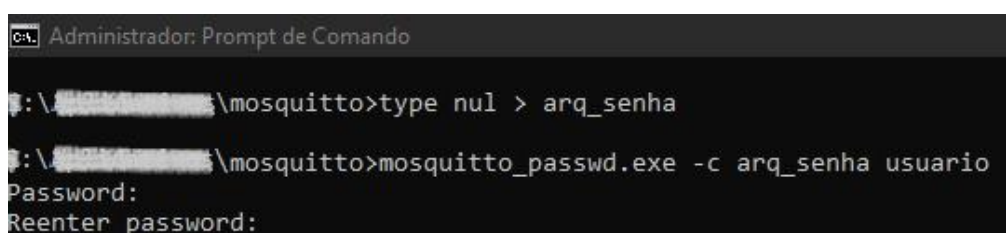
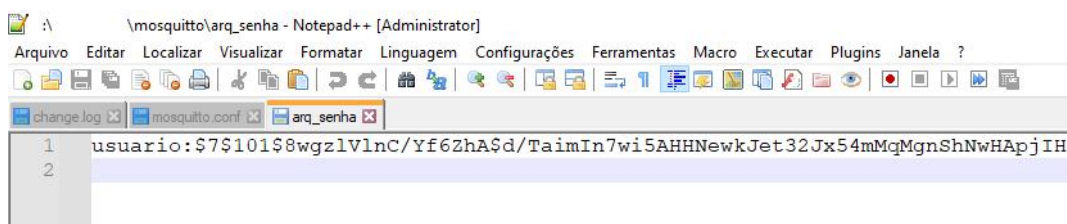


Figura 44 - Definição de usuário e senha de acesso do *broker*
Fonte: autoria própria

Após a confirmação da senha de acesso ao *broker*, foi realizado a verificação do arquivo “*arq_senha*” com o mesmo editor utilizado para a parametrização do arquivo “*mosquitto.conf*”. Dessa forma, foi possível visualizar que o processo ocorreu adequadamente, tendo em vista o ilustrado na Figura 45, em que se nota a criação de um usuário chamado “*usuario*”, ao mesmo tempo que se tem um conjunto de caracteres vinculados a ele. Na prática, este conjunto de caracteres representou a criptografia da senha definida na Figura 44.



```

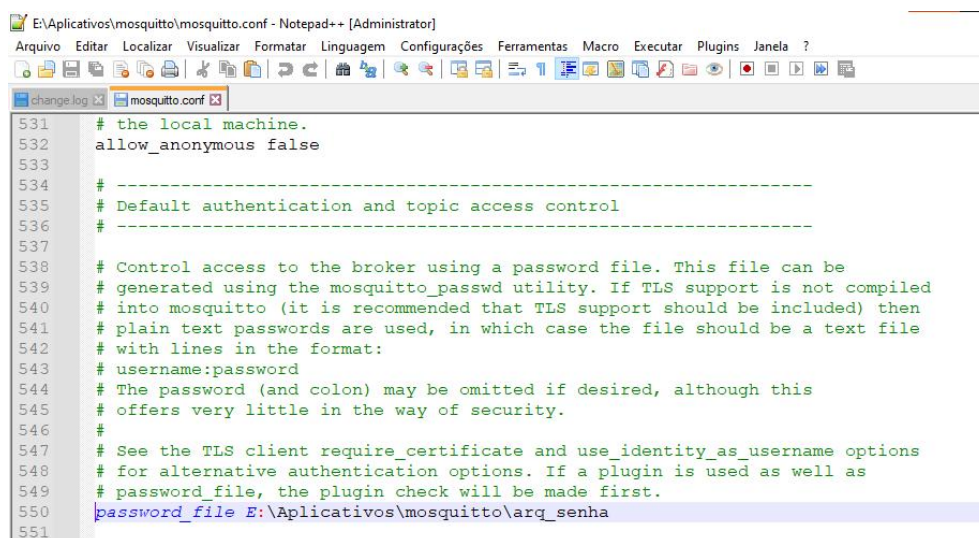
1 usuario:$7$101$8wgz1VlnC/Yf6ZhA$d/TaimIn7wi5AHHNewkJet32Jx54mMqMgnShNwHApjIH
2

```

Figura 45 - Arquivo de autenticação do *broker* configurado

Fonte: autoria própria

Por último, para que fosse possível validar a configuração de autenticação na Figura 45, realizou-se, novamente, o acesso ao arquivo de configuração do *broker* (*mosquitto.conf*), e dessa vez foi realizado a edição da linha 550. Com essa edição, definiu-se o caminho em que se encontra o arquivo com as informações a respeito do usuário e sua respectiva senha de acesso (*arq_senha*), que foi apresentado na Figura 45. Neste sentido, na Figura 46 é destacado a configuração final do arquivo “*mosquitto.conf*”, com todas essas alterações realizadas.



```

531 # the local machine.
532 allow_anonymous false
533
534 # -----
535 # Default authentication and topic access control
536 # -----
537
538 # Control access to the broker using a password file. This file can be
539 # generated using the mosquitto_passwd utility. If TLS support is not compiled
540 # into mosquitto (it is recommended that TLS support should be included) then
541 # plain text passwords are used, in which case the file should be a text file
542 # with lines in the format:
543 # username:password
544 # The password (and colon) may be omitted if desired, although this
545 # offers very little in the way of security.
546 #
547 # See the TLS client require_certificate and use_identity_as_username options
548 # for alternative authentication options. If a plugin is used as well as
549 # password_file, the plugin check will be made first.
550 password_file E:\Aplicativos\mosquitto\arq_senha
551

```

Figura 46 - Validação do arquivo de autenticação de acesso ao *broker*

Fonte: autoria própria

Em relação aos arquivos de configuração do *broker*, os procedimentos de parametrização do servidor foram encerrados. Porém, para garantir o correto funcionamento deste servidor, fez-se necessário realizar mais duas configurações na máquina em que o *broker* foi instalado.

A primeira dessas configurações esta relacionada com a liberação da porta 1883, parametrizada anteriormente na Figura 41. Assim, para acessar essa funcionalidade, foi realizado a busca por “*Windows Defender Firewall com Segurança Avançada*” na barra de pesquisa do Windows. Com a interface aberta, direcionou-se até as opções “Regras de Entrada > Nova regra...” até que a tela de configuração se assemelhasse a presente na Figura 47. Após isso, as seguintes configurações foram selecionadas: “Porta > Avançar > TCP > Portas locais específicas > 1883 > Avançar > Permitir conexão > Avançar > Domínio > Particular > Público > Avançar > Nome: MQTT > Concluir”. Dessa forma, garantiu-se não haver impeditivo por parte das políticas do Firewall, para que outros dispositivos se conectassem ao servidor MQTT que foi configurado para operar na máquina.

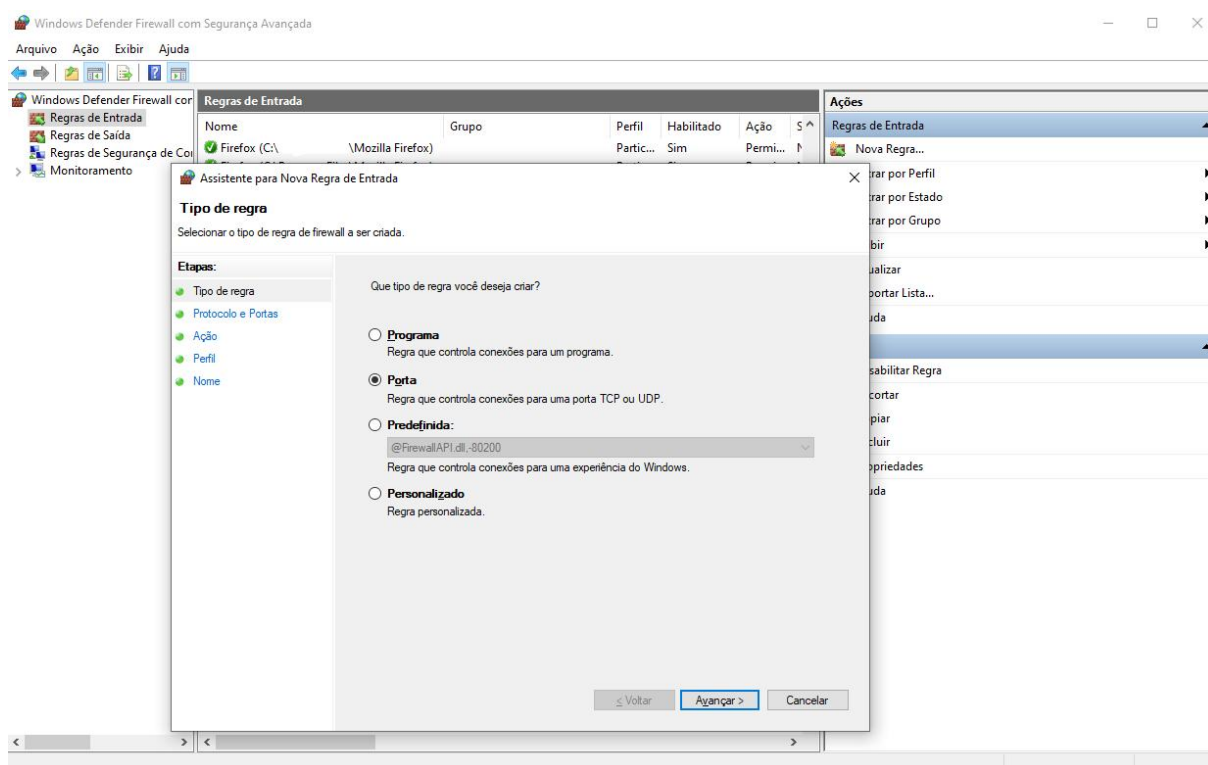


Figura 47 - Configuração do Firewall do Windows para liberação da porta de comunicação do broker
Fonte: autoria própria

Por fim, a segunda configuração pôde ser realizada ao acessar a aba de “Serviços”, via barra de pesquisas do Windows, de forma que fosse possível configurar o *broker* para iniciar

automaticamente sua operação assim que a máquina fosse ligada, sem a necessidade de realizar este procedimento manualmente. Assim, após acessar a interface de Serviços do Windows, procurou-se pelo Nome “Mosquitto Broker”, e ao encontrá-lo, foi realizado a configuração da opção “Tipo de Inicialização” para Automático, conforme apresentado na Figura 48.

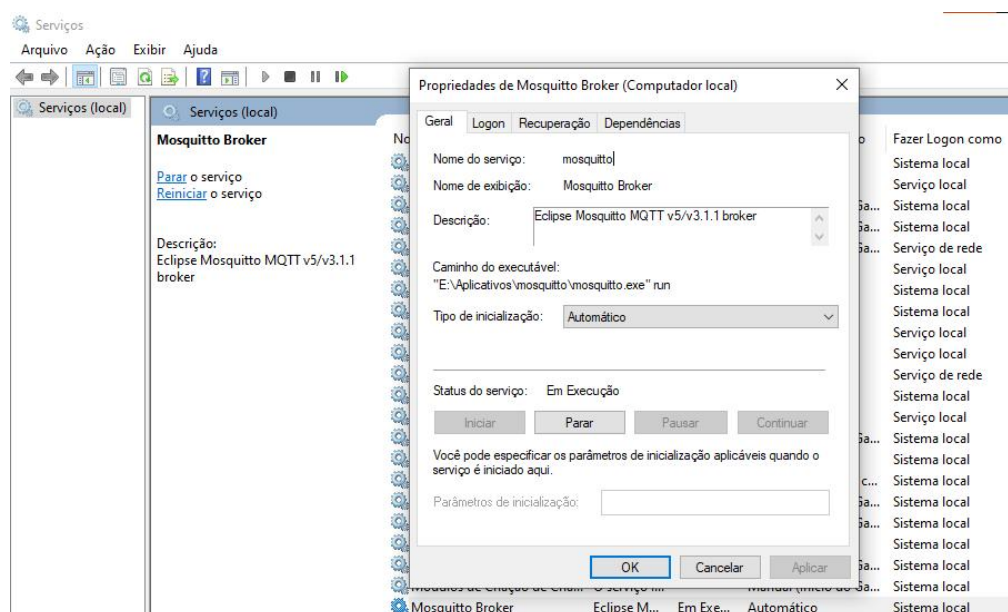


Figura 48 - Configuração da inicialização automática do broker
Fonte: autoria própria

Dessa forma, foram realizadas todas as configurações necessárias para permitir a operação do *broker* na rede, de forma que, a partir disso, os clientes consigam se conectar ao servidor, e assim, realizar a troca de informações entre si.

4.2 PROGRAMAÇÃO DA PLATAFORMA MICROCONTROLADA

Como destacado na seção anterior, teve-se o objetivo de integrar o microcontrolador apresentado na Figura 15 em uma rede MQTT, de forma que ele se torne um cliente nesta rede, e opere tanto como *publisher*, quanto *subscriber*. Assim, nos momentos em que o microcontrolador enviasse os dados de leitura da comunicação com os inversores ao *broker*, o dispositivo seria caracterizado como *publisher*. Por outro lado, quando um sinal externo fosse enviado aos tópicos em que o microcontrolador se encontrasse inscrito, ele passaria a ser

caracterizado como *subscriber*, uma vez que ele aguardaria o recebimento desses dados para executar uma determinada função.

Neste sentido, para que fosse possível implementar o microcontrolador na rede MQTT, utilizou-se da biblioteca “Pubsubclient.h”, disponível em [44], que possibilita que microcontroladores, no geral, estabeleçam conexão com um determinado *broker*. Após essa conexão, o ESP32 utilizado neste trabalho se tornou capaz de se inscrever nos tópicos de interesse, de forma que ele conseguisse receber informações de outros dispositivos que também estivessem vinculados a estes mesmos tópicos. De maneira complementar, a plataforma microcontrolada também se tornou capaz de publicar dados específicos em seus respectivos tópicos, de forma que outros dispositivos conectados ao *broker* tivessem acesso a essas informações.

Dessa maneira, caso fosse necessário alterar o fator de potência de operação do inversor fotovoltaico, ou ler a capacidade de carga das baterias no inversor híbrido, um comando deveria ser enviado ao *broker*, em um tópico em que o microcontrolador fosse capaz de operar. Ao receber essa informação, o microcontrolador deveria ser capaz de compreender o tipo de operação a ser realizada, a qual dos inversores ela está relacionada, e qual o registrador de interesse para esta operação.

Assim, para que fosse possível identificar esses parâmetros referentes ao tipo de operação a ser realizada, do inversor e do registrador de interesse, definiu-se um padrão para a nomeação dos tópicos, de forma que fosse possível realizar a identificação desses parâmetros citados apenas com a designação do nome do tópico. Esse padrão que foi utilizado pode ser visualizado nos exemplos apresentados na Figura 49, em que os dois primeiros caracteres foram utilizados para gerar a correspondência entre o equipamento de origem, e o dado a ser publicado. De maneira complementar, os próximos três caracteres foram definidos para identificar se aquele tópico deve ser tratado como leitura (“_L_”), ou escrita (“_E_”) de registradores, tendo em vista que, na prática, o primeiro se trata do envio de dados armazenados no microcontrolador ao *broker*, e o segundo do recebimento de mensagens de outros dispositivos ao ESP32. Por fim, os últimos caracteres foram utilizados para identificar o registrador em que aquela informação teve origem na comunicação Modbus, de forma que, ao relacionar essa informação com aquelas presentes na Figura 21 e na Figura 29, tenha-se a correta utilização e configuração desses dados.

Equipamento	Código Equipamento	Leitura (L) ou Escrita (E)	Endereço do Registrador	Nome do tópico (MQTT)
Inv. Fotovoltaico	FR	L	40071	FR_L_40071
Inv. Fotovoltaico	FR	L	40232	FR_L_40232
Inv. Fotovoltaico	FR	E	40232	FR_E_40232
Inv. Híbrido	VE	L	3	VE_L_3
Inv. Híbrido	VE	L	56	VE_L_56
Inv. Híbrido	VE	E	56	VE_E_56

Figura 49 - Exemplo de nomeação dos tópicos da comunicação MQTT

Fonte: autoria própria

Tendo como base esses princípios, para que o microcontrolador fosse capaz de se conectar ao *broker* configurado neste trabalho, foi necessário garantir que o ESP32 estivesse conectado na mesma rede da máquina em que o servidor MQTT foi instalado e iniciado a operação. Assim, tornou-se possível de realizar a identificação do IP desta máquina, que de maneira direta, corresponde ao IP do *broker* na rede.

Além disso fez-se necessário conhecer os dados de autenticação do *broker*, como usuário e senha, que foram configurados de acordo com o procedimento apresentado na Figura 46, assim como a porta padrão em que a rede MQTT se encontra operando, destacado na Figura 41. Com essas informações, obteve-se todos os dados necessários para realizar a programação do microcontrolador, tendo em vista o código apresentado no APÊNDICE A, que foi resumido de acordo com o algoritmo do fluxograma destacado na Figura 50.

Em relação ao fluxograma apresentado na Figura 50, deve-se destacar as funções “callback_MQTT”, “reconnectMQTT” e “pub_MQTT”, que foram funções essenciais para o funcionamento da comunicação entre o microcontrolador e o *broker*.

A primeira dessas funções, “callback_MQTT”, foi a responsável por tratar das mensagens publicadas nos tópicos em que o microcontrolador foi inscrito, de forma que qualquer nova informação recebida nestes tópicos, fosse utilizada como gatilho para que ocorresse a execução automática desta função. Assim, o algoritmo dessa função foi implementado visando a resposta do microcontrolador ao receber qualquer informação dos tópicos de interesse, e no caso deste trabalho, essa situação esteve relacionada com a operação de escrita dos registradores dos inversores na comunicação Modbus. Dessa forma, no contexto deste trabalho, esta função somente foi executada quando houve a incidência de novas mensagens em tópicos que possuem os caracteres “_E_” como identificadores, obedecendo o padrão apresentado na Figura 49. Por sua vez, as informações contidas nessas mensagens correspondiam ao valor de alteração na operação dos inversores, e por este motivo, a função “esc_mb”, apresentada no fluxograma ilustrado na Figura 38, também foi utilizada

nesta função. Com base nisso, na Figura 51 é apresentado o fluxograma de funcionamento do algoritmo utilizado na função de *callback* da comunicação MQTT.

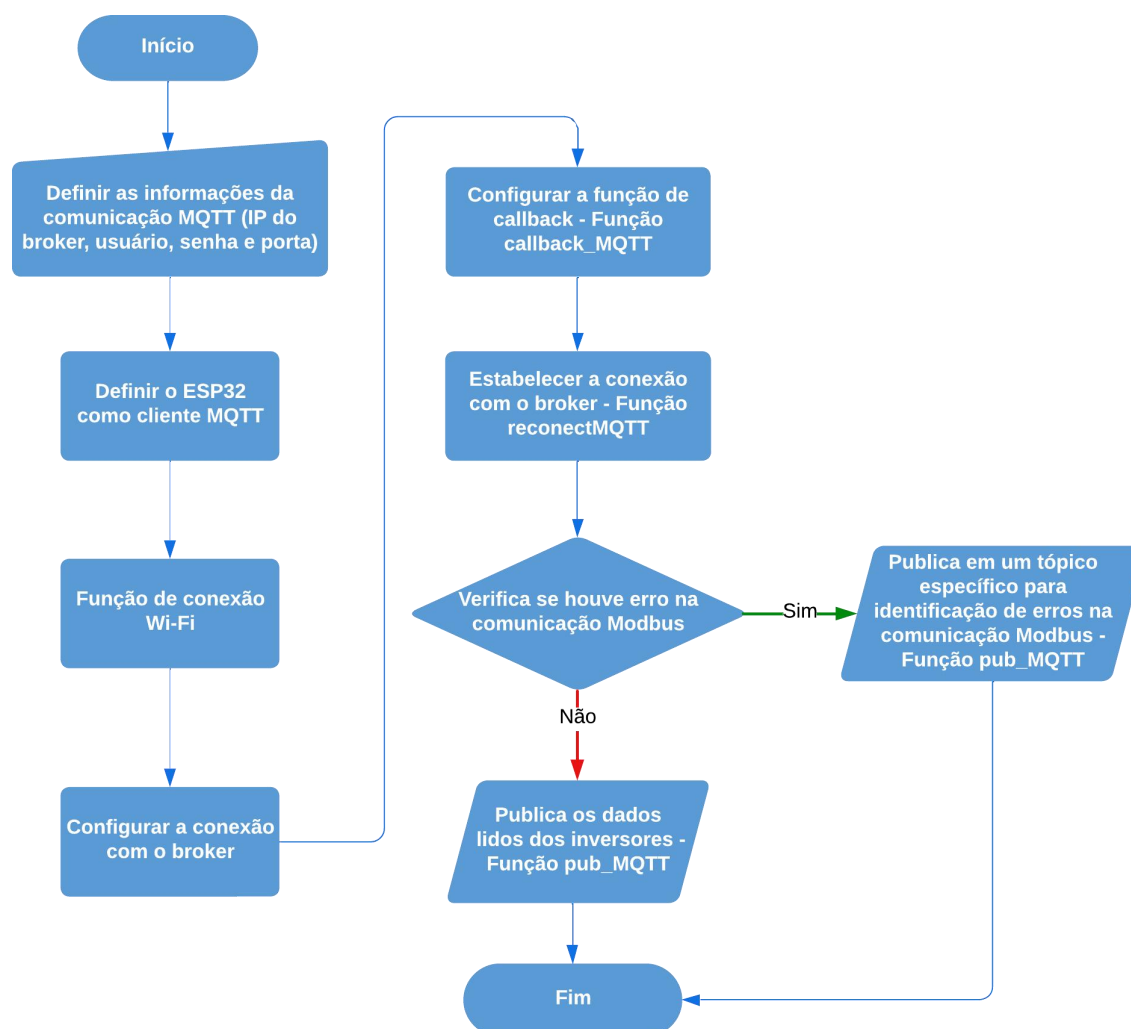


Figura 50 - Fluxograma da comunicação MQTT do Microcontrolador
 Fonte: autoria própria

Em segundo lugar, a função “reconnectMQTT” se tornou a responsável por efetivamente realizar a conexão do microcontrolador com o *broker*, via protocolo MQTT. Neste contexto, as informações referentes ao IP do servidor, porta de comunicação utilizada, e os dados de autenticação foram utilizados, seguindo o algoritmo apresentado no fluxograma destacado na Figura 52. Nesta função, também foi realizado a inscrição do dispositivo em todos os tópicos relacionados a escrita de dados dos inversores, de forma a garantir que a função “callback_MQTT” fosse disparada sempre que houvesse uma nova mensagem nesses tópicos.

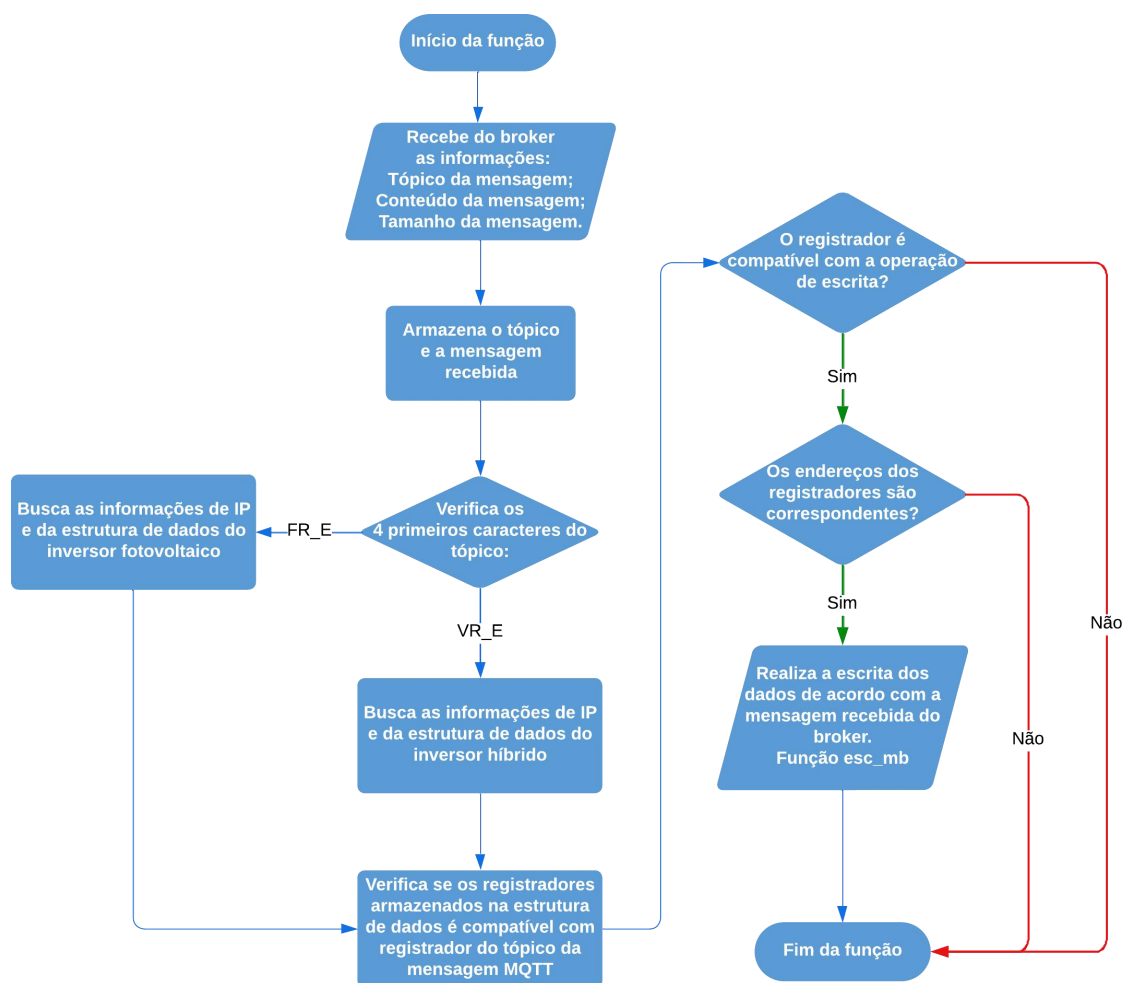


Figura 51 - Função de *callback* da comunicação MQTT do microcontrolador
Fonte: autoria própria

Por fim, tem-se a função “pub_MQTT”, que teve como encargo a publicação efetiva dos dados obtidos na comunicação Modbus, em seus respectivos tópicos na comunicação MQTT. Para isso, ela utilizou de todas as informações armazenadas nas estruturas de dados, de forma que os tópicos dessas informações fossem nomeados em conformidade com o padrão apresentado na Figura 49, e os dados encaminhados obedecendo uma outra formatação. Esta outra formatação dos dados para publicação nos tópicos ocorreu de acordo com padrão destacado na Figura 53, em que foi utilizado de um separador, no caso o caractere “#”, para dividir a mensagem em dois campos. O primeiro campo, que antecede o caractere separador, correspondia ao valor extraído diretamente da leitura via comunicação Modbus com os inversores, sem que tenha sido realizado o ajuste de escala deste parâmetro. Já o segundo campo, após o caractere separador, apresentava o respectivo fator de escala desse registrador, de forma que, com a composição completa da mensagem, qualquer outra interface fosse capaz de realizar a conversão deste valor, tendo em vista a metodologia apresentada na Figura 22. Assim, este padrão foi estabelecido devido a maior facilidade para realizar este tratamento dos

dados na plataforma utilizada para a interface do sistema, conforme é apresentado no tópico 5.1.2.1.

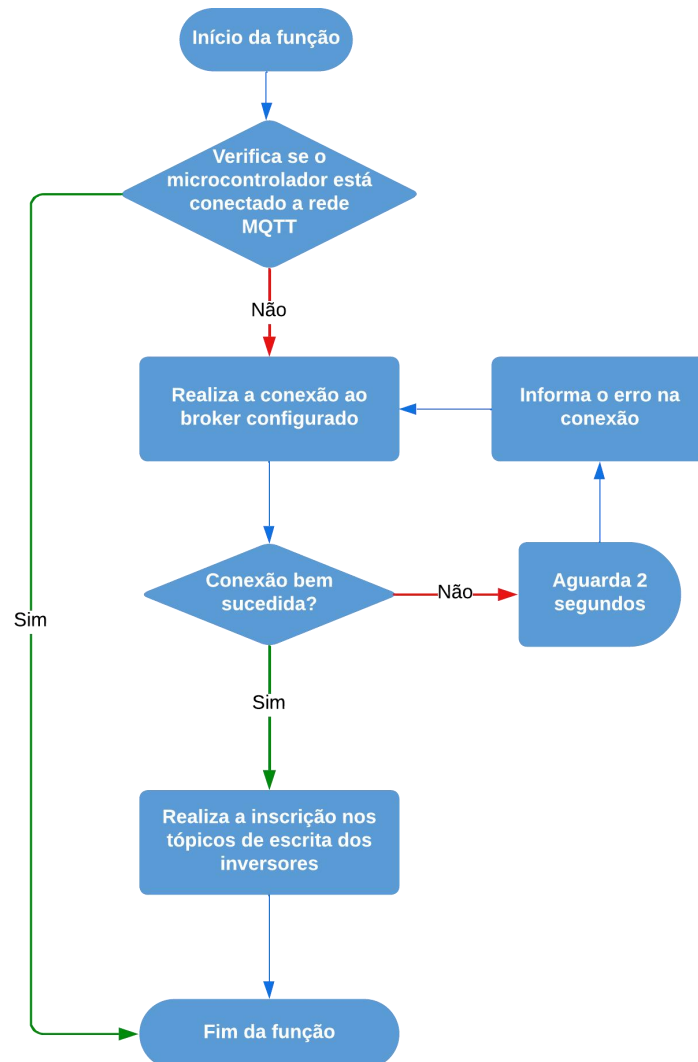


Figura 52 - Fluxograma da função de conexão ao servidor MQTT
Fonte: autoria própria

Valor	Caractere Separador	Fator de escala
10	#	-1
Mensagem final: 10#-1		

Figura 53 - Formato da mensagem a ser publicada na rede MQTT
Fonte: autoria própria

Com base nos métodos das formatações dos tópicos e das mensagens, que foram apresentados na Figura 49 e na Figura 53, o algoritmo utilizado para a função de publicação dos dados na rede MQTT pode ser visualizado no fluxograma ilustrado na Figura 54. Com essas configurações, o microcontrolador se tornou capaz de publicar todos os dados que foram

armazenados na estrutura que foi configurada como parâmetro desta função. Com isso, ocasionou-se na disponibilidade de acesso a essas informações em qualquer dispositivo que estivesse conectado ao *broker* da rede MQTT. Assim, a próxima etapa do trabalho consistiu em utilizar esses dados disponíveis no servidor MQTT para compor uma interface web que permitisse a um usuário interagir com essas informações de maneira simples e dinâmica.

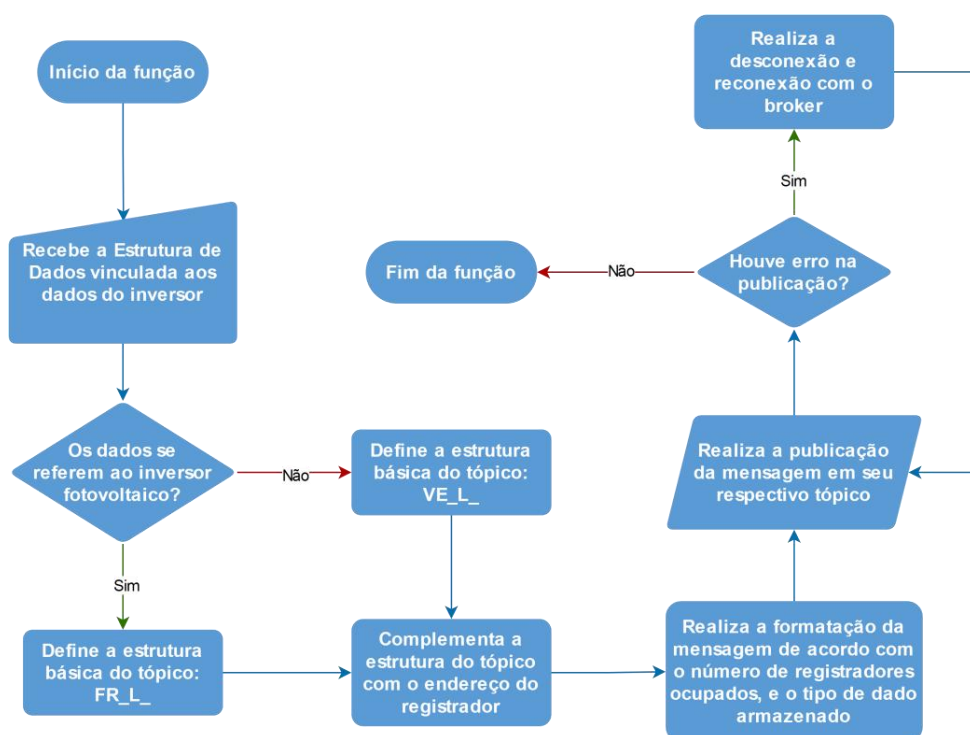


Figura 54 - Função para publicação de dados via comunicação MQTT

Fonte: autoria própria

5 CAPÍTULO V - INTERFACE WEB

Com a implementação da rede MQTT para a transferência dos dados da microrrede realizada, a próxima etapa do trabalho consistiu no desenvolvimento de uma interface web que fosse capaz de interagir com essas informações de maneira funcional. Para isso, foi utilizado o software Node-RED, de forma que a interface desenvolvida tivesse a capacidade de extrair os dados da rede MQTT, e apresentar essas informações ao usuário por meio de recursos gráficos visuais, facilitando o entendimento da operação da microrrede. Além disso, esta interface também permitiu que o usuário realizasse algumas alterações no funcionamento dos equipamentos, seja por meio de acesso em dispositivos móveis ou não, desde que estes estejam conectados a rede do sistema. Assim, este capítulo apresenta uma visão geral relacionado aos procedimentos utilizados para estabelecer a interface web para supervisão e controle dos inversores, utilizando o Node-RED como meio para este fim.

5.1 NODE-RED

O Node-RED, segundo [45], se trata uma ferramenta de programação baseada em fluxos, criada inicialmente pela IBM e atualmente mantida pela *OpenJS Foundation*. Esta ferramenta permite o desenvolvimento de aplicações através da conexão de "nós" que executam determinadas funções específicas, de forma a processar e transmitir dados de maneira visual e intuitiva.

Assim, este software oferece um editor de códigos acessível via navegador, em que os fluxos são criados, configurados e compartilhados com facilidade, tendo em vista que eles são integralmente baseados em JSON (*JavaScript Object Notation*). Dessa forma, desde o lançamento do software como projeto de código aberto em 2013, o Node-RED ganhou destaque por sua flexibilidade e integração com dispositivos IoT, sistemas e APIs, tornando-se uma ferramenta essencial para desenvolvedores que buscam rapidez e eficiência na criação de soluções [45].

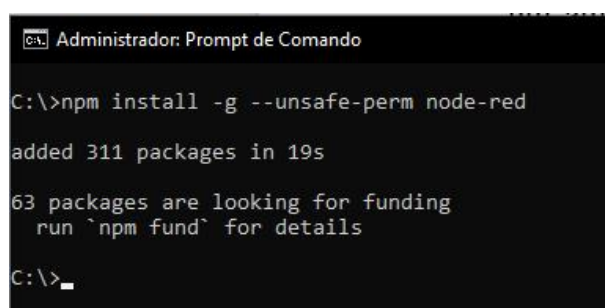
Com base nessa alta gama de recursos disponíveis na ferramenta de programação do Node-RED, que incluem a compatibilidade com a comunicação MQTT, operação com banco de dados SQL e a criação de *dashboards*, optou-se pela utilização deste software para o

desenvolvimento da interface de supervisão e controle deste trabalho. Além disso, a facilidade para o desenvolvimento dos fluxos também contribuiu de maneira positiva para a utilização da ferramenta, uma vez que o desenvolvimento completo de uma interface com todas essas funcionalidades exigiria um nível elevado de conhecimentos técnicos e recursos.

5.1.1 Instalação do servidor Node-RED

Para que fosse possível realizar a instalação do Node-RED no Windows, fez-se necessário, primeiramente, garantir que uma versão compatível do Node.js estivesse instalada no computador, conforme destaca o manual de instalação do software em [46]. O Node.js, por sua vez, consiste em um ambiente de código aberto que permite a execução de algoritmos de JavaScript fora do navegador, e seu arquivo para a instalação pôde ser encontrada em [47], que no caso deste trabalho, consistiu na versão v22.13.0.

Com a instalação bem sucedida do Node.js, a próxima etapa ocorreu com a instalação do Node-RED, que neste caso, foi realizada via *prompt* de comando, uma vez que todos os comandos necessários para a realização desse procedimento se tornaram acessíveis com a instalação do Node.js. Para isso, executou-se o *prompt* de comando do Windows como administrador, de forma que o comando “npm install -g --unsafe-perm node-red” pudesse ser inserido. Após a finalização do período de instalação do Node-RED, o terminal adquiriu uma característica semelhante a apresentada na Figura 55, de forma a indicar que o processo foi realizado com êxito .



```
Administrador: Prompt de Comando

C:\>npm install -g --unsafe-perm node-red

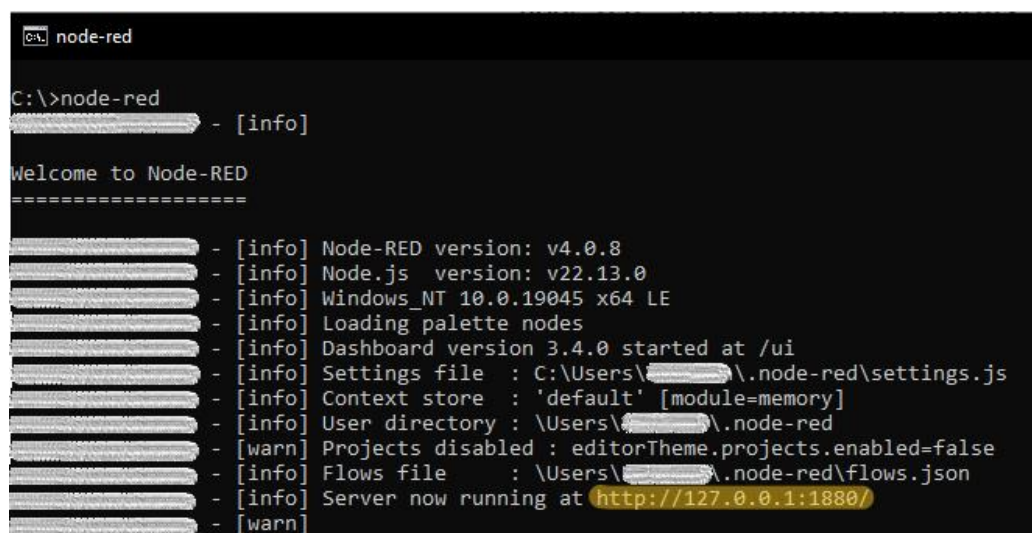
added 311 packages in 19s

63 packages are looking for funding
  run `npm fund` for details

C:\>_
```

Figura 55 - Comando de instalação do Node-RED
Fonte: autoria própria

Por fim, para que o servidor Node-RED se tornasse funcional, bastou utilizar do comando “node-red” no mesmo terminal, de maneira que a situação final do *prompt* fosse próxima da presente na Figura 56.

A screenshot of a Windows command prompt window titled "node-red". The user has entered the command "C:\>node-red". The terminal displays a series of status messages: "[info] Welcome to Node-RED", "[info] Node-RED version: v4.0.8", "[info] Node.js version: v22.13.0", "[info] Windows_NT 10.0.19045 x64 LE", "[info] Loading palette nodes", "[info] Dashboard version 3.4.0 started at /ui", "[info] Settings file : C:\Users\...\node-red\settings.js", "[info] Context store : 'default' [module=memory]", "[info] User directory : \Users\...\node-red", "[warn] Projects disabled : editorTheme.projects.enabled=false", "[info] Flows file : \Users\...\node-red\flows.json", and "[info] Server now running at http://127.0.0.1:1880/". The IP address "127.0.0.1:1880/" is highlighted in yellow.

```
C:\>node-red
- [info]
Welcome to Node-RED
=====
- [info] Node-RED version: v4.0.8
- [info] Node.js version: v22.13.0
- [info] Windows_NT 10.0.19045 x64 LE
- [info] Loading palette nodes
- [info] Dashboard version 3.4.0 started at /ui
- [info] Settings file : C:\Users\...\node-red\settings.js
- [info] Context store : 'default' [module=memory]
- [info] User directory : \Users\...\node-red
- [warn] Projects disabled : editorTheme.projects.enabled=false
- [info] Flows file : \Users\...\node-red\flows.json
- [info] Server now running at http://127.0.0.1:1880/
- [warn]
```

Figura 56 - Comando de inicialização do servidor Node-Red
Fonte: autoria própria

Na Figura 56, destacado na cor amarela, é possível observar endereço de IP em que o servidor foi configurado para operar, que neste caso, corresponde ao próprio endereço de IP da máquina em que o Node-RED foi instalado, de forma que ele pudesse ser acessado por qualquer dispositivo que tivesse acesso a esta mesma rede. Assim, bastou inserir o endereço destacado na Figura 56 na barra de pesquisa do navegador da máquina, para que a interface de edição do Node-RED fosse aberta, conforme apresentado na Figura 57.

Com o ambiente de programação operando adequadamente, conforme destacado na Figura 57, a próxima etapa do trabalho consistiu na implementação de um sistema de autenticação no editor, de forma a garantir que os fluxos ali criados não fossem alterados facilmente por qualquer pessoa com acesso a rede.

Dessa forma, utilizou-se do procedimento apresentado em [48], que consistiu, primeiramente, em converter a senha de interesse em um conjunto de caracteres criptografados. Para isso, foi necessário realizar um novo acesso ao terminal de comandos do Windows, e assim, utilizar o comando “node-red admin hash-pw”, de forma que este procedimento ocasionasse na execução de um algoritmo que convertesse a senha inserida no campo “*Password*” para um conjunto de caracteres criptografados, tendo em vista as informações apresentadas na Figura 58.

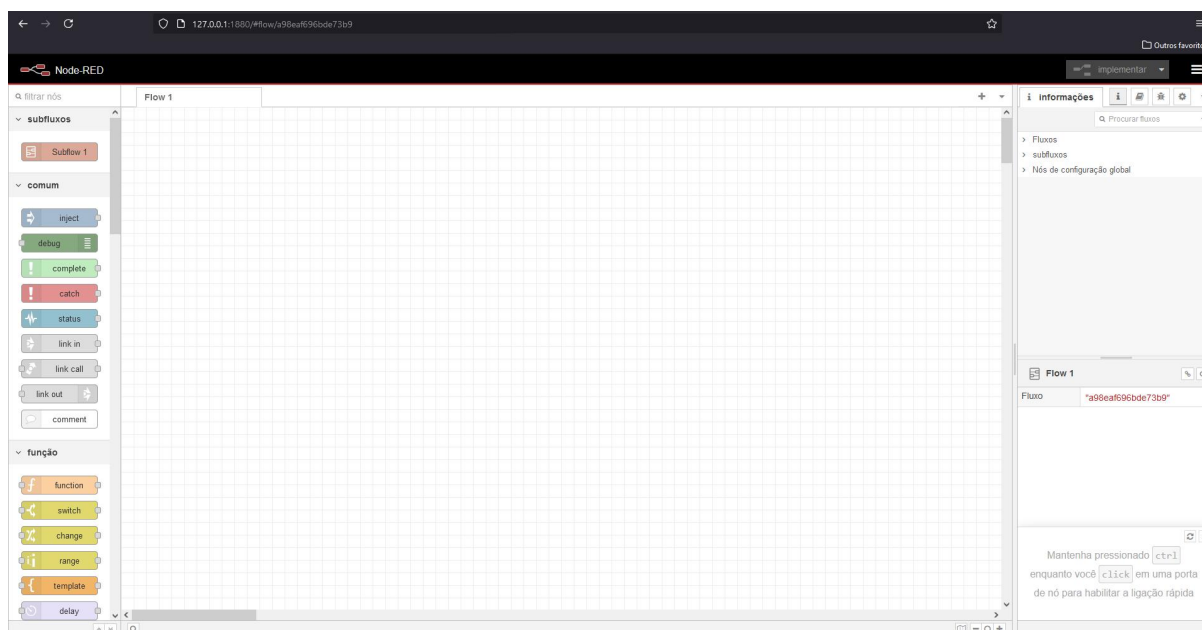


Figura 57 - Ambiente de programação do Node-RED

Fonte: autoria própria

```
C:\WINDOWS\system32\cmd.exe

C:\>node-red admin hash-pw
Password: $2y$08$nuauAbmF8G2hSYmmy5Aht0wxnSB4kVCUiAEzZdT83F1P93.HbCJIK
C:\>
```

Figura 58 - Definição da senha de acesso ao editor do Node-RED

Fonte: autoria própria

Dessa forma, o conjunto de caracteres gerado no procedimento demonstrado na Figura 58 foi copiado, tendo em vista que é essa informação que deve ser utilizada na configuração da criação do usuário para a autenticação de acesso ao editor. Assim, a próxima etapa consistiu em acessar a pasta em que o servidor Node-RED foi instalado, de forma a encontrar e editar o arquivo “setting.js”. O caminho para acesso dessa pasta pode ser encontrado na Figura 56, na linha “*Settings Files*”, e da mesma forma que ocorreu a configuração de autenticação do *broker* no capítulo anterior, este arquivo também foi editado pelo software “Notepad++”.

Assim, ao realizar o acesso ao arquivo “setting.js”, navegou-se até a linha 76 do arquivo, e, removeu-se os comentários de uma parcela do código, que neste caso compreendeu o intervalo da linha 76 a linha 83. Após isso, o campo “*username*” foi alterado para o nome desejado do usuário, enquanto que o campo “*password*” foi preenchido com os caracteres gerados na Figura 58. O campo “*permissions*”, por sua vez, permaneceu marcado

com “*”, tendo em vista que essa configuração permitiu que este usuário conseguisse editar os fluxos integrados ao ambiente de programação do Node-RED. Ademais, caso fosse necessário que ele apenas visualizasse o editor, a alteração deste campo para “read” deveria ser realizada. Assim, na Figura 59 é apresentada a configuração final desta etapa de configuração da autenticação de acesso ao editor do Node-RED.



Figura 59 - Definição de usuário e senha de acesso ao editor do Node-RED
 Fonte: autoria própria

Por fim, para que este procedimento fosse validado, bastou realizar o salvamento do arquivo “setting.js” editado, e assim, reiniciar o servidor Node-RED, procedimento este que consistiu, basicamente, em fechar o terminal apresentado na Figura 56, e executar o comando “node-red” novamente no *prompt* de comandos. Dessa forma, ao acessar o endereço de operação do servidor Node-RED no navegador da máquina, informação esta que foi destacada em amarelo na Figura 56, a interface ilustrada na Figura 60 foi aberta. Com isso, teve-se a indicação de que o processo de autenticação de acesso já foi estabelecido no ambiente de programação de fluxos.

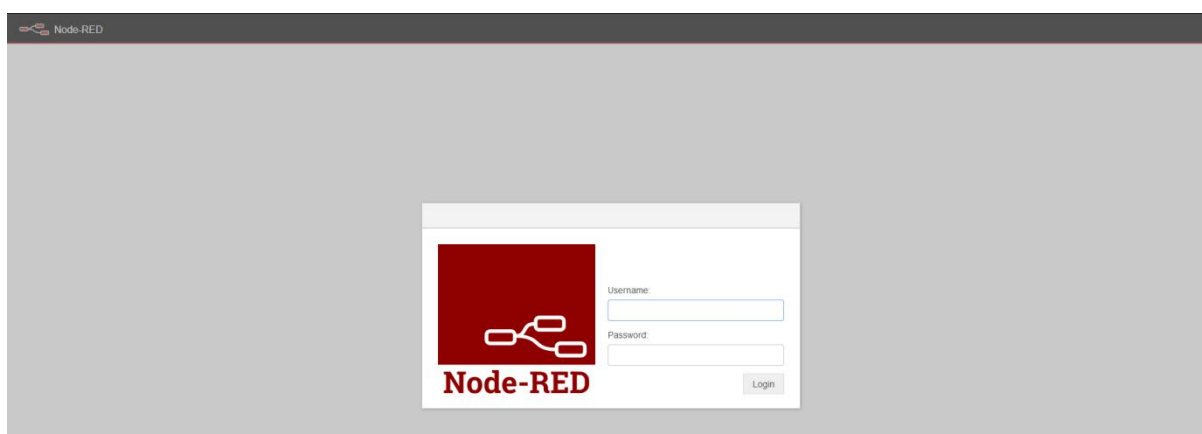


Figura 60 - Interface de autenticação do editor do Node-RED
 Fonte: autoria própria

Com base nisso, tem-se que o processo de instalação e configuração inicial do Node-RED foi finalizado, bastando, a partir de agora, realizar a programação do sistema para o recebimento de dados via rede MQTT, a apresentação dessas informações em um *dashboard* e a integração deste sistema com um banco de dados local.

5.1.2 Vinculação com o *broker* MQTT

Com o ambiente para a programação dos fluxos operando adequadamente, a próxima etapa do trabalho consistiu em realizar a conexão do servidor Node-RED com a rede MQTT configurada na seção 4.1.1. Para isso, utilizou-se do nó nomeado de “*mqtt in*”, disponibilizado na borda esquerda da tela inicial do editor de fluxos, no grupo “*network*”, conforme apresentado na Figura 61.

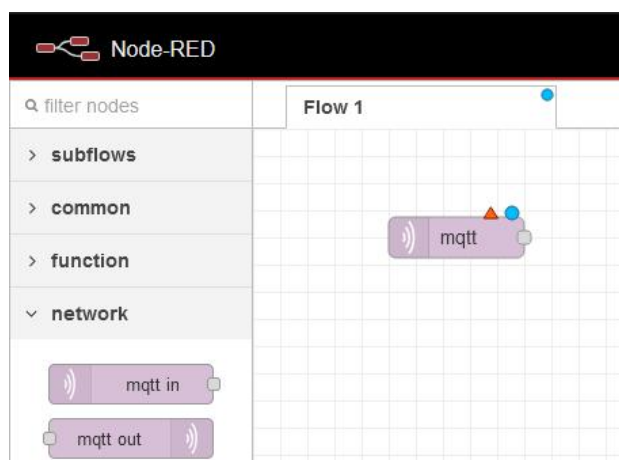


Figura 61 - Inserção do nó "mqtt in" no Node-RED

Fonte: autoria própria

0

Ao clicar sobre o nó inserido, uma tela de configuração do nó foi aberta, conforme apresentado na Figura 62. Nesta figura, pode-se perceber alguns pontos de interesse, como os campos específicos para a configuração do *broker* e do tópico da mensagem, assim como um botão de “ajuda” no canto inferior esquerdo da figura, destacado em cor amarela. Basicamente, a maioria dos nós possuem este botão, que oferecem uma visão geral acerca das funcionalidades deste nó, funcionando como uma documentação em menor escala de uma biblioteca.

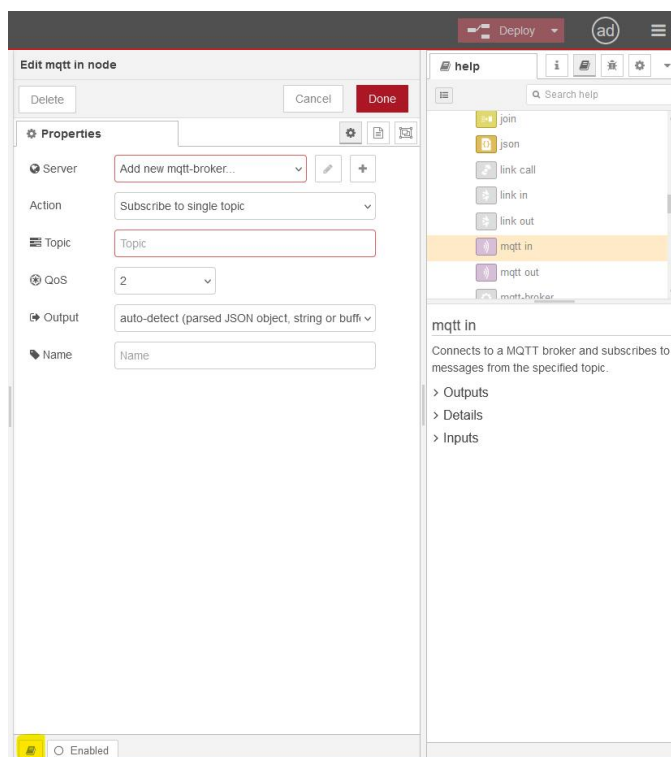


Figura 62 - Edição do nó "mqtt in" no Node-RED
Fonte: autoria própria

Em relação aos campos de configuração do *broker*, apresentados na Figura 62, destaca-se o campo “*Server*”, em que se deve realizar a parametrização do servidor MQTT cuja a conexão é pretendida. Para realizar esta configuração, clicou-se sobre o ícone reservado a adição de uma configuração, de forma que uma janela semelhante a visível na Figura 63 fosse sobreposta. Assim, foi fornecido um espaço para o preenchimento das configurações apresentadas na seção 4.1.1, de forma que o endereço de IP do *broker* fosse configurado no campo “*Server*”. O protocolo utilizado deve ser compatível com o a versão do *broker* instalado, que neste caso correspondeu a versão V3.1.1. As demais configurações foram mantidas de acordo com o padrão do estabelecido na Figura 63. Por fim, na aba “*Security*”, inseriu-se os dados de autenticação do broker, no que diz respeito ao usuário e senha de acesso que foram configurados na Figura 44.

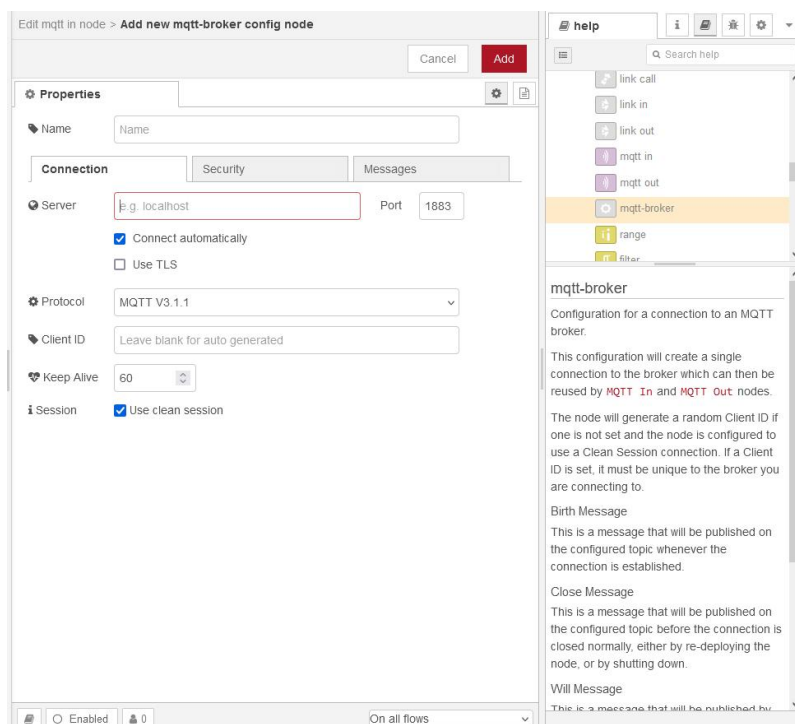


Figura 63 - Configuração do *broker* de conexão do nó "*mqtt in*"

Fonte: autoria própria

Dessa maneira, com a configuração do *broker* no nó "*mqtt in*" finalizada, a próxima etapa consistiu na definição do tópico em que este nó foi vinculado para operação como *subscriber*, aguardando a publicação de dados por parte do microcontrolador. Para isso, bastou digitar o tópico de interesse no campo "*topic*" presente na Figura 62, obedecendo a formatação de nomenclatura dos tópicos que foi definida na Figura 49.

Com o tópico definido, o próximo consistiu na validação da configuração realizada no nó através de um simples clique sobre o botão "*Deploy*", que por sua vez, pôde ser localizado no canto superior direito da interface de edição dos fluxos. Como procedimento padrão, essa validação das alterações deve ser executada em todo momento que uma nova configuração for realizada no Node-RED. Assim, após garantir que todas as configurações foram realizadas adequadamente, uma notificação de "conectado" passou a ser apresentada na parte inferior do nó, conforme destacado na Figura 64, indicando êxito na conexão do nó com o *broker*.



Figura 64 - Configuração do nó "*mqtt in*" validado

Fonte: autoria própria

5.1.2.1 Formatação e Direcionamento das Mensagens Recebidas

Com a configuração de conexão do *broker* realizada, e o tópico definido no nó “*mqtt in*”, a próxima etapa do procedimento para recebimento dos dados do inversor consistiu na formatação da mensagem recebida. Essa formatação deve ser implementada em concordância com o padrão estabelecido na Figura 53, de forma que o nó utilizado seja capaz de separar a informação real do parâmetro, do seu respectivo fator de escala, e a partir disso, executar a correta correção desta informação.

Para isso, utilizou-se do nó “*function*”, que pode ser encontrado no grupo de mesmo nome, no canto esquerdo do editor do Node-RED, de forma que a entrada do componente esteja conectada a saída do nó “*mqtt in*”, conforme destacado na Figura 65. No contexto do Node-RED, este nó é utilizado como uma alternativa para qualquer ação que não esteja predefinida em algum dos nós disponíveis no ambiente de edição, tendo em vista que a partir dele é possível realizar a escrita de qualquer algoritmo desejado.

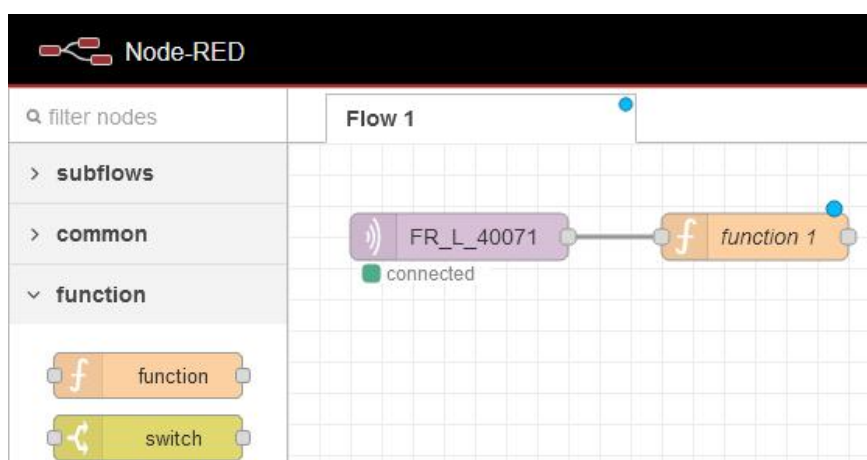


Figura 65 - Utilização do nó "function" no Node-RED
Fonte: autoria própria

Ao abrir a tela de configuração do nó implementado na Figura 65, uma interface semelhante a ilustrada na Figura 66 foi apresentada, de forma que fosse possível notar um espaço reservado para implementação do código de interesse, que por sua vez, deve ser baseado na linguagem JavaScript. Neste sentido, a aba “*On Message*” na Figura 66, se refere ao código que deverá ser executado sempre que uma mensagem for recebida pelo nó de função, que no caso deste trabalho, esteve relacionado com o recebimento de informações do *broker*. Ademais, ainda se tem a possibilidade de escrever um outro código a ser executado

uma única vez, na primeira execução de toda programação realizada no Node-RED, sendo que, para isso, basta alternar para a aba “On Start”. De forma complementar, o mesmo pode acontecer de maneira inversa, na aba “On Stop”, em que o código implementado ali será executado uma única vez ao fim da execução do programa. Por fim, a aba “Setup” se refere as configurações possíveis de serem realizadas no nó “function”, como o aumento de saídas disponíveis, por exemplo.

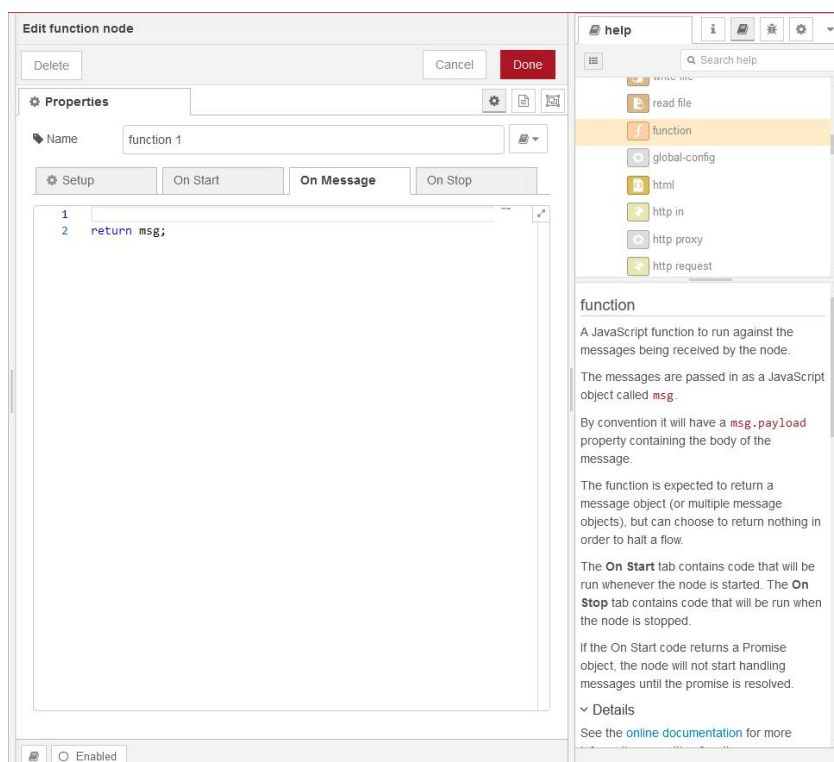


Figura 66 - Configuração do nó "function" do Node-RED
Fonte: autoria própria

Assim, para realizar a correção do fator de escala da mensagem enviada pelo microcontrolador ao *broker*, utilizou-se apenas da aba “On Message” para a implementação do código responsável por adequar o valor lido nos inversores. O algoritmo utilizado pode ser visualizado no APÊNDICE B, assim como no fluxograma apresentado na Figura 67. Neste algoritmo, e nos demais que foram utilizados dentro do Node-RED, fez-se necessário a manipulação da propriedade “msg.payload”, que por sua vez, corresponde a forma padrão de como a mensagem é transmitida entre os nós de um determinado fluxo. Em outras palavras, este parâmetro corresponde a propriedade do objeto “msg”, dentro do Node-RED, em que está armazenado o conteúdo efetivo da mensagem. De maneira complementar, outras informações ainda podem ser acessadas ou criadas por meio de outras propriedades vinculadas ao objeto

“msg”, como no caso da “msg.topic”, que, por padrão, corresponde ao tópico da mensagem recebida ou enviada por um nó.

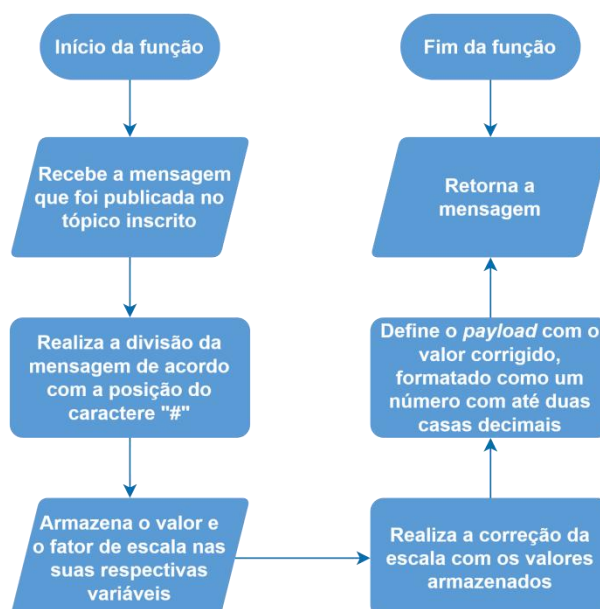


Figura 67 - Fluxograma do algoritmo para correção de escala no Node-RED

Fonte: autoria própria

Com a implementação da função de formatação da mensagem realizada, o próximo passo consistiu na utilização do nó “link out”, que por sua vez, foi utilizado para fornecer uma conexão virtual entre o nó inserido na Figura 65, e os futuros nós a serem utilizados pela interface. Para isso, este nó pode ser encontrado no grupo “common”, e seu funcionamento está baseado apenas na transmissão da mensagem para outros nós do tipo “link in”, sem que seja necessário o acúmulo de conexões em um único nó de função, por exemplo. A conexão desta tipologia de nó pode ser visualizada na Figura 68.

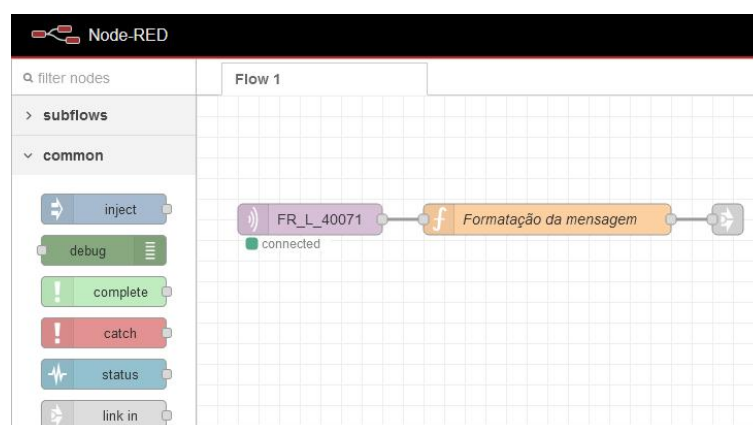


Figura 68 - Implementação do nó "link out" no Node-RED

Fonte: autoria própria

Já a configuração do nó “*link out*”, pode ser encontrada na Figura 69, e se tratou de uma configuração simples, baseada apenas na definição de um nome passível de ser identificado posteriormente, e na configuração do modo “*Send to all connected link nodes*”. Como padrão, estabeleceu-se o nome dos nós de “*links*” de acordo com o respectivo tópico em que eles estão vinculados, que por sua vez, foram submetidos ao padrão destacado na Figura 49.

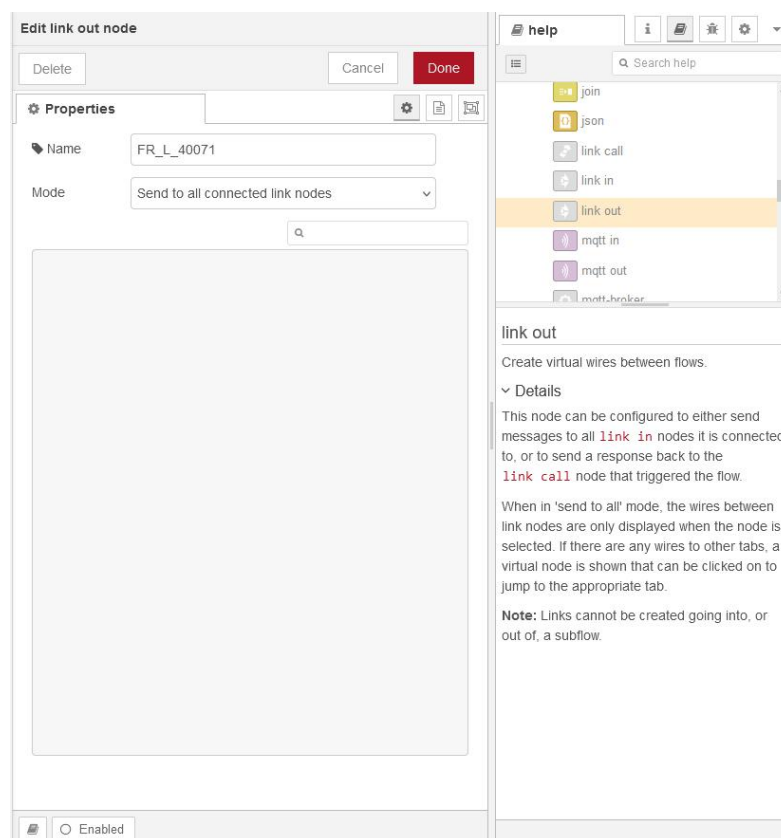


Figura 69 - Configuração do nó "link out" no Node-RED
 Fonte: autoria própria

Por fim, como uma questão de organização e padronização, nomeou-se o fluxo criado tendo em vista a sua respectiva funcionalidade. Para isso, bastou realizar um clique sobre a região do nome “*flow 1*”, que pode ser identificado na Figura 68 logo acima dos nós implementados, de forma a que uma interface de configuração semelhante a apresentada na Figura 70 fosse ativada. Assim, buscou-se manter este tipo de padronização quanto ao nome dos fluxos em todo trabalho, de forma a facilitar no processo de identificação de onde se encontram os fluxos específicos para cada funcionalidade, como aqueles para o tratamento dos nós de *dashboard*, ou então com as requisições de banco de dados.

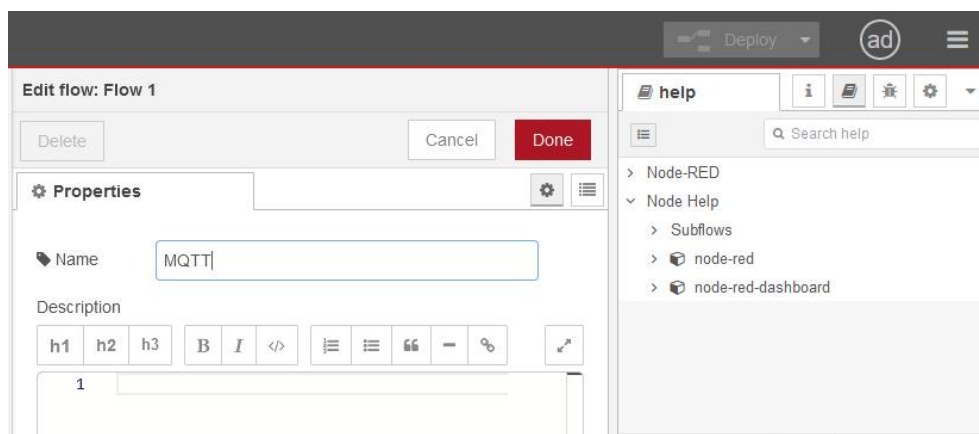


Figura 70 - Identificação do fluxo criado

Fonte: autoria própria

Com base nisso, essas foram as metodologias padrão utilizadas para a vinculação do Node-RED com as informações publicadas no servidor MQTT, sendo que, para cada tópico em que houve o interesse de extrair as informações, fez-se necessário repetir todo o procedimento apresentado nesta seção. No caso deste trabalho, realizou-se este procedimento em todos os tópicos que estiveram relacionados com algum parâmetro de operação ou configuração dos inversores, de forma a tornar necessário a utilização de diversas estruturas de nós como apresentado na Figura 71. Assim, este procedimento foi realizado como uma forma de deixar disponível, e com acesso facilitado, todas as informações importantes da comunicação com os inversores.

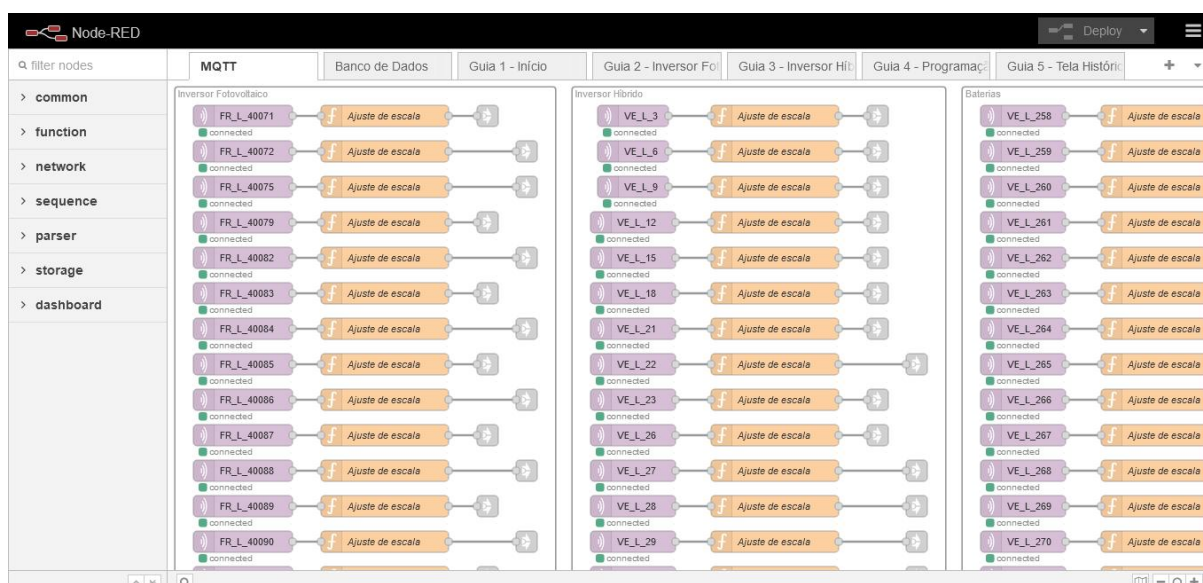


Figura 71 - Amostra do fluxo final da comunicação MQTT no Node-RED

Fonte: autoria própria

Com isso, realizou-se todas as configurações necessárias para a integração do servidor Node-RED com o *broker* da rede MQTT, de forma que, a partir de então, fosse necessário realizar apenas a configuração do *dashboard* para o sistema, assim como a vinculação deste sistema com um banco de dados local.

5.1.3 Desenvolvimento da Interface de Supervisão e Controle

Com a comunicação com os inversores estabelecida via comunicação MQTT, a próxima etapa do trabalho consistiu na elaboração efetiva da interface para supervisão e controle dos inversores instalados na microrrede. Para viabilizar esta parte do trabalho, fez-se necessário a instalação de um novo conjunto de nós, que puderam ser encontrados ao acessar a configuração “*Manage palette*”, que se tornou apresentável ao clicar sobre o ícone presente no canto superior direito da tela, conforme ilustrado na Figura 72.

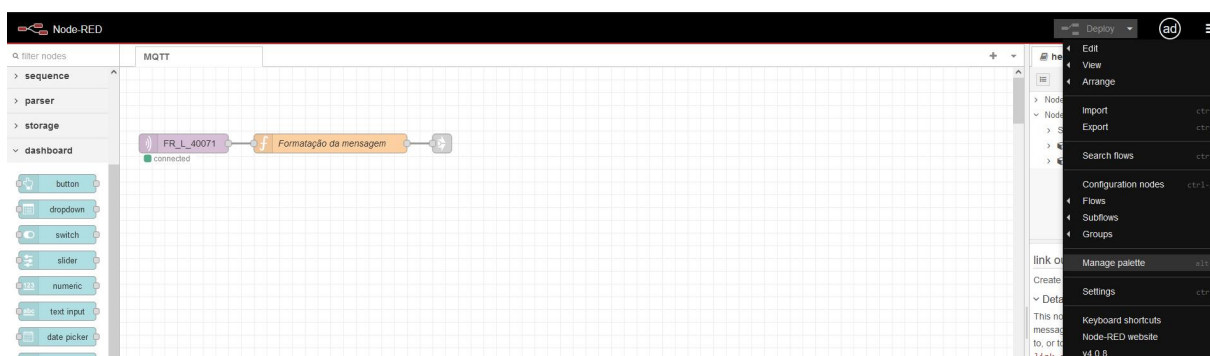


Figura 72 - Acesso a configuração para inserção de nós
Fonte: autoria própria

Ao realizar a seleção da configuração “*Manage palette*”, seguiu-se até a aba “*Install*” da interface aberta, de forma que fosse possível pesquisar pelo nome “node-red dashboard”. Assim, na Figura 73 é destacada a interface resultante desta busca, de maneira que o primeiro resultado apresentado fosse exatamente aquele que foi utilizado para o desenvolvimento do trabalho.

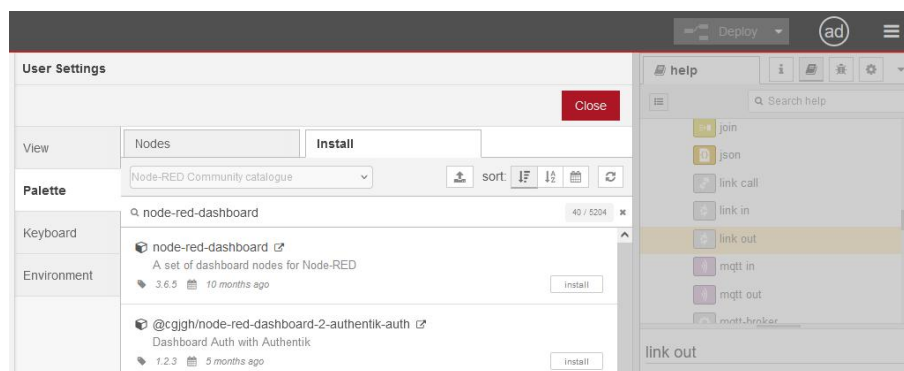


Figura 73 - Instalação dos nós do Node-RED Dashboard

Fonte: autoria própria

Dessa forma, prosseguiu-se com a instalação desse conjunto ao clicar sobre o botão “*install*” que foi apresentado na Figura 73, sendo que, após a finalização do processo instalação, um novo grupo de nós ficou disponível na tela inicial do editor, identificado pelo nome “*dashboard*”. Assim, na Figura 74 é destacado os nós disponibilizados com a instalação do novo módulo, ademais, antes de efetivamente iniciar a modelagem do fluxo da interface, fez-se necessário criar uma nova área para o tratamento específico desta tipologia de fluxo.

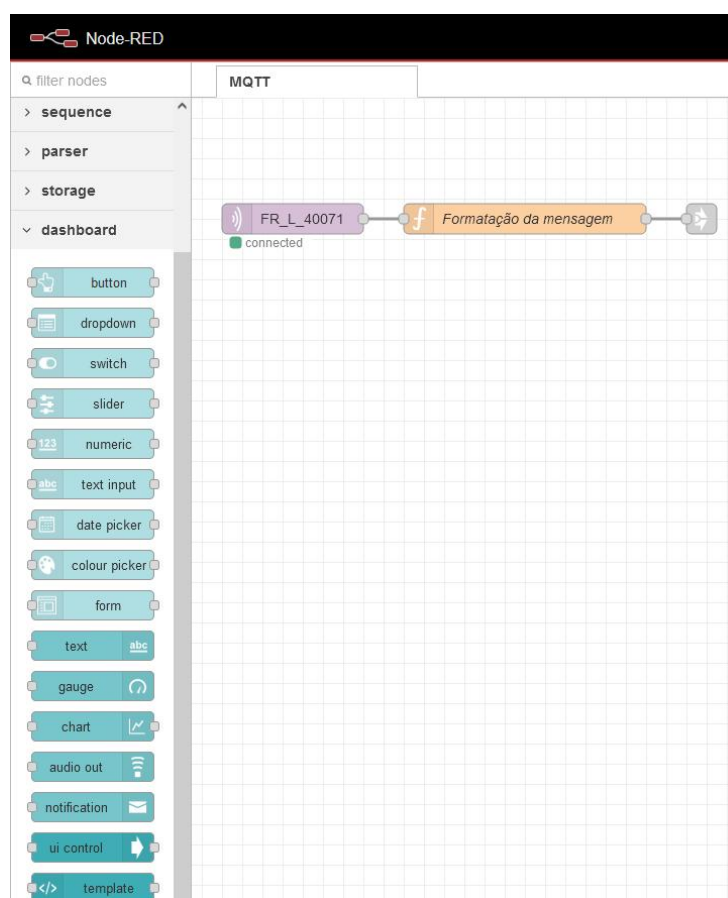


Figura 74 - Nós para Dashboard instalados

Fonte: autoria própria

Dessa forma, para criar um novo fluxo exclusivo para o tratamento dos nós de *dashboard*, utilizou-se o botão de “adicionar”, que pôde ser encontrado no canto inferior esquerdo do botão de “*deploy*”, tendo em vista a ambientação da tela apresentada na Figura 72. Ao clicar sobre este botão, um novo fluxo foi apresentado de maneira imediata, e o mesmo procedimento apresentado na Figura 70 pode ser utilizado para nomeá-lo como “Dashboard”. Assim, obteve-se uma área reservada no editor para tratar especificamente dos fluxos referentes a utilização de recursos gráficos no Node-RED Dashboard.

5.1.3.1 Implementação dos Recursos de Supervisão

Com a interface específica para gerenciamento dos nós de *dashboard* aberta, realizou-se, primeiramente, a busca da informação que se deseja apresentar na tela de supervisão. Para isso, utilizou-se do nó “*link in*”, de forma que ele fosse configurado para receber a informação que se tem o interesse de apresentar. Assim, na Figura 75 é apresentado a configuração utilizada nesta configuração no editor do Node-RED, que neste caso, representava a busca pela informação do registrador 40071 do inversor fotovoltaico. Este registrador, por sua vez, armazena a informação da corrente de saída do equipamento.



Figura 75 - Inserção do nó "link in"
Fonte: autoria própria

Dessa forma, para que fosse possível apresentar a informação que foi buscada de maneira direta, em formato de texto, utilizou-se do nó “*text*”, disponibilizado no grupo de nós “*dashboard*”. Assim, a configuração dele foi executada tendo em vista suas propriedades de configuração, apresentadas na Figura 76, em que o campo “*group*” se refere ao local em que um conjunto de informações e recursos podem ser agrupados na guia da interface web criada. Nota-se que esta se trata da única configuração obrigatória para a implementação deste nó. As

demais parametrizações, como o tamanho, a descrição, e a formatação do conteúdo e do estilo também fazem parte das configurações possíveis de serem realizadas nesse nó, porém elas devem ser configuradas de acordo com a necessidade do usuário.

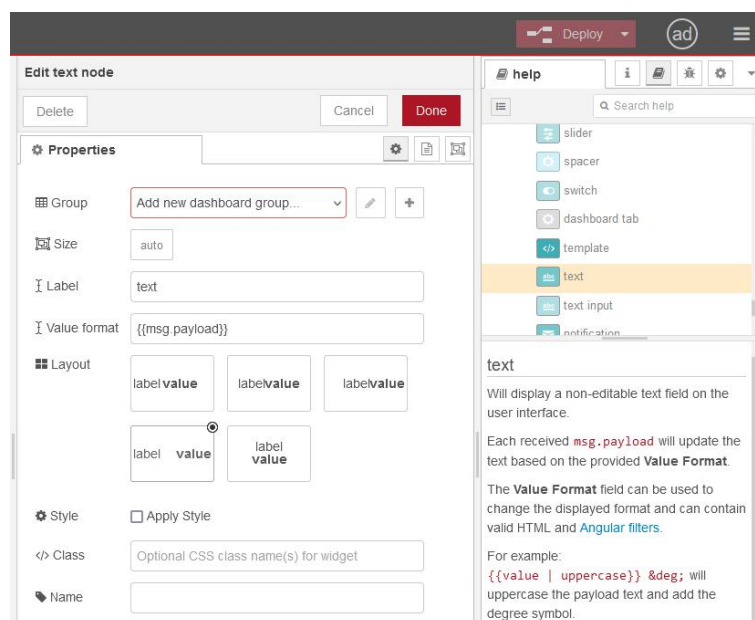


Figura 76 - Configuração do nó "text" no Node-RED
Fonte: autoria própria

Com base nisso, foi adicionado um novo grupo para o nó de texto, de forma que uma nova interface de configuração fosse aberta, como demonstrado na Figura 77. Nesta figura, verificou-se a necessidade de criação de uma nova guia no campo “tab”, que na prática, esteve relacionado com a possibilidade de navegar entre diferentes guias na tela de supervisão. Ao clicar sobre o botão de adição desta guia, uma nova interface foi sobreposta no editor, de forma que se tornasse possível identificar os parâmetros utilizados para configuração desta guia na Figura 78.

Com isso, realizou-se a configuração básica para apresentação de dados via campo de texto, sendo que as demais configurações levam em consideração aspectos visuais, que podem ser alterados de acordo com a necessidade do usuário. Para isso, ainda há a possibilidade de implementação do nó “template”, que permite a estilização das páginas por meio da linguagem CSS.

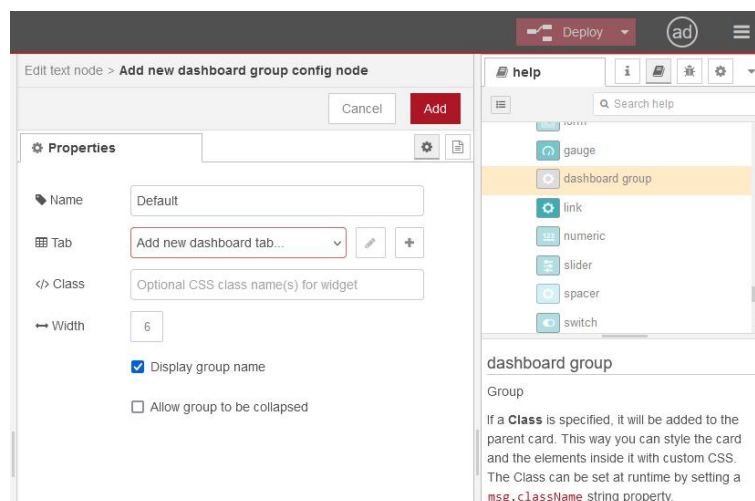


Figura 77 - Configuração de grupos para a interface web do Node-RED Dashboard
Fonte: autoria própria

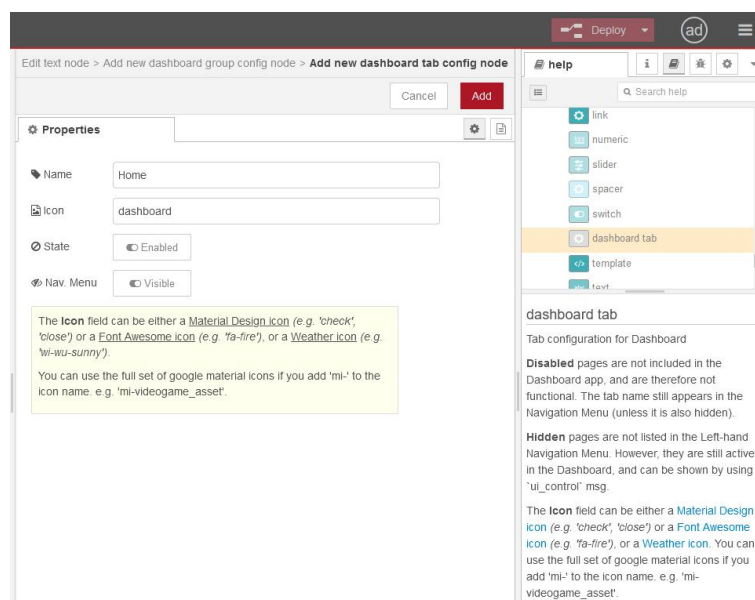


Figura 78 - Configuração de guias para a interface web do Node-RED Dashboard
Fonte: autoria própria

Um outro recurso que também pode ser utilizado para a apresentação dos dados se trata do nó “*chart*”, que, por sua vez, permite a visualização da dinâmica de uma determinada informação através de um gráfico 2D, e que ainda pode ser atualizado em tempo real. A configuração deste nó pode ser observada na Figura 79, de forma que a definição para que ele operasse no mesmo grupo que o nó “*text*” inserido anteriormente, ocasionasse no agrupamento dessas informações em uma única guia, com os dados próximos um dos outros. As demais configurações podem ser alteradas de acordo com a necessidade do usuário, tendo em vista que também se tratam de parâmetros voltados a estilização do gráfico na interface final.

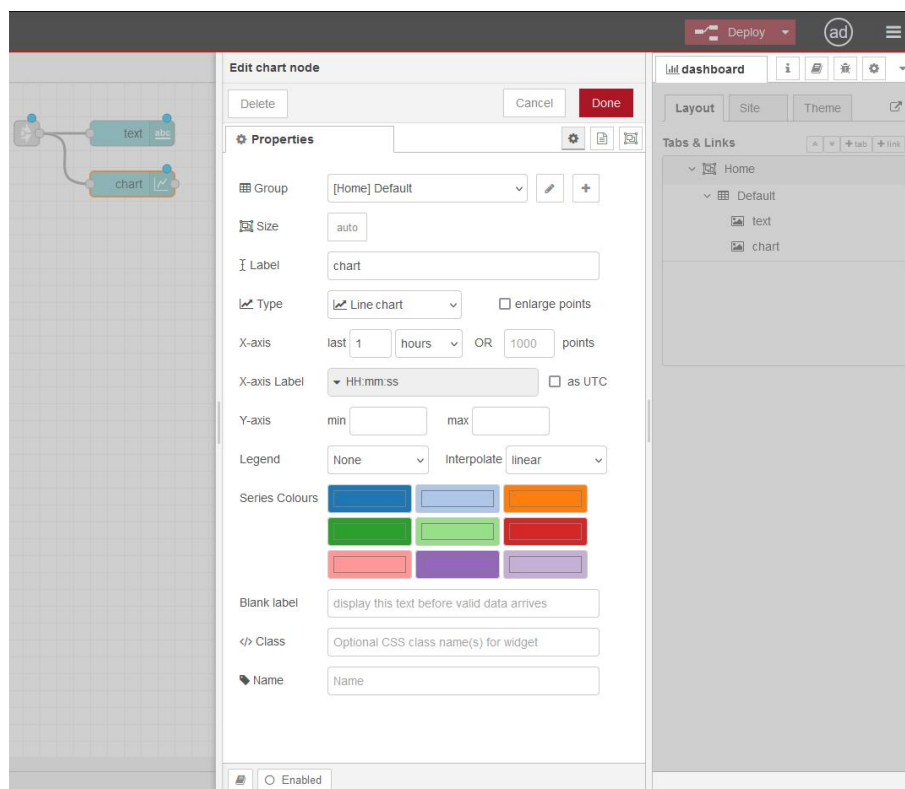


Figura 79 - Configuração do nó "chart" no Node-RED
Fonte: autoria própria

De maneira complementar, desenvolveu-se um padrão para todos os recursos visuais do Node-RED Dashboard, de forma que nenhum dos grupos configurados na Figura 77 possuam largura superior a 6 unidades. Essa padronização decorreu como uma alternativa para manter a compatibilidade da interface web para a visualização tanto em *smartphones*, quanto em notebooks e desktops, de forma que não se tenha perda de informações em ambas as telas.

Assim, para que fosse possível visualizar a interface web resultante dessas configurações, foi realizado o acesso ao ícone situado imediatamente ao lado direito do botão “*Configuration nodes*”, localizado no canto superior direito na Figura 80, e destacado na cor amarela. Com a abertura das opções disponíveis, selecionou-se o campo “*Dashboard*”, de forma que a interface final se assemelhasse a presente na Figura 80. Após isso, foi realizado um clique sobre o ícone destacado na cor verde na Figura 80, o que ocasionou na abertura de uma nova página no navegador, sendo esta a página específica para a visualização do resultado dos nós configurados do Node-RED Dashboard. Uma outra forma de acessar a interface web gerada pelo Node-RED *Dashboard* seria através da utilização do endereço

“IP_NODE-RED:1880/ui” a ser buscado no navegador, de forma que o campo “IP_NODE-RED” fosse substituído pelo endereço de IP equivalente ao apresentado na Figura 56.

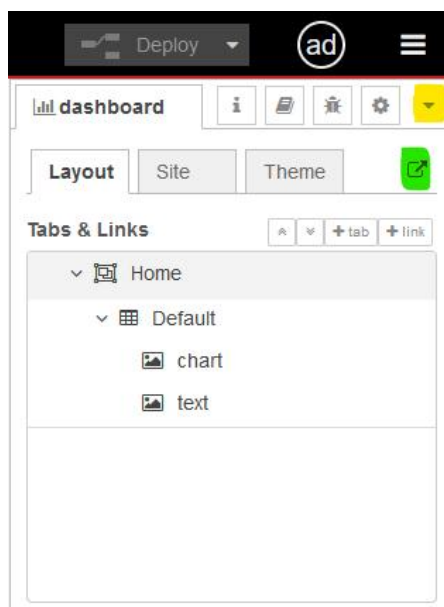


Figura 80 - Processo de abertura da interface de supervisão configurada
Fonte: autoria própria

Dessa forma, a interface que foi aberta se encontra apresentada na Figura 81. Nesta figura, pode-se notar que tanto o gráfico quanto o campo de texto configurados, se encontraram inseridos no mesmo grupo, e por isso foram alocados com o gráfico na parte superior, e o texto na parte inferior. Para realizar a alteração desse posicionamento, basta alterar a posição destes recursos no campo destacado como “Tabs & Links” na Figura 80.

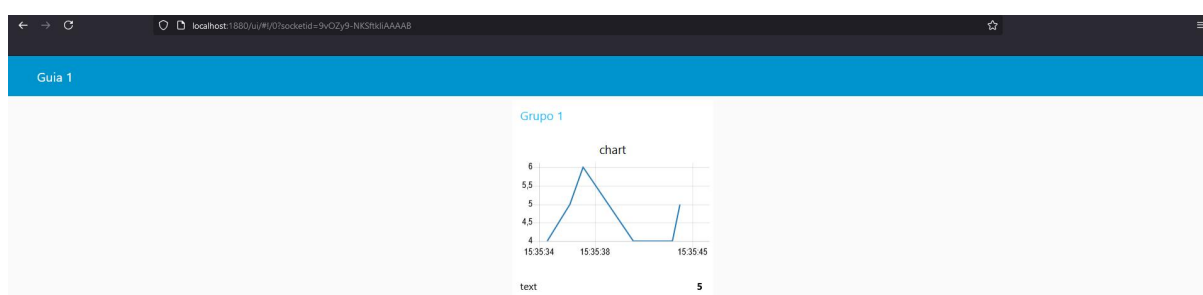


Figura 81 - Interface de supervisão inicial configurada
Fonte: autoria própria

Com base nessas configurações, a guia de supervisão geral deste trabalho foi elaborada seguindo a mesma metodologia apresentada nessa seção, assim como as demais telas que também envolvem os aspectos de leitura de dados dos inversores.

5.1.3.2 Implementação dos recursos de controle

Em relação ao desenvolvimento das funcionalidades que envolvem o controle dos inversores instalados na microrrede, esta ocorreu baseada principalmente na utilização do nó “*form*”, alocada dentro do grupo “*dashboard*” no editor do Node-RED. Isso se deu devido a possibilidade de o usuário fornecer múltiplas configurações em um só ambiente, de forma que fosse possível a realização de configurações imediatas de alguns parâmetros de operação dos inversores.

Para isso, este nó utiliza da configuração de múltiplos elementos dentro dele, de forma que estes elementos possam ser manipulados pelo usuário no *dashboard* de maneira individual, e tratados como um JSON dentro da programação do Node-RED. Neste sentido, na Figura 82 é apresentada as configurações passíveis de serem realizadas nesse nó, de forma que e o resultado esperado na interface que foi gerada por esta configuração, possa ser encontrado na Figura 83.

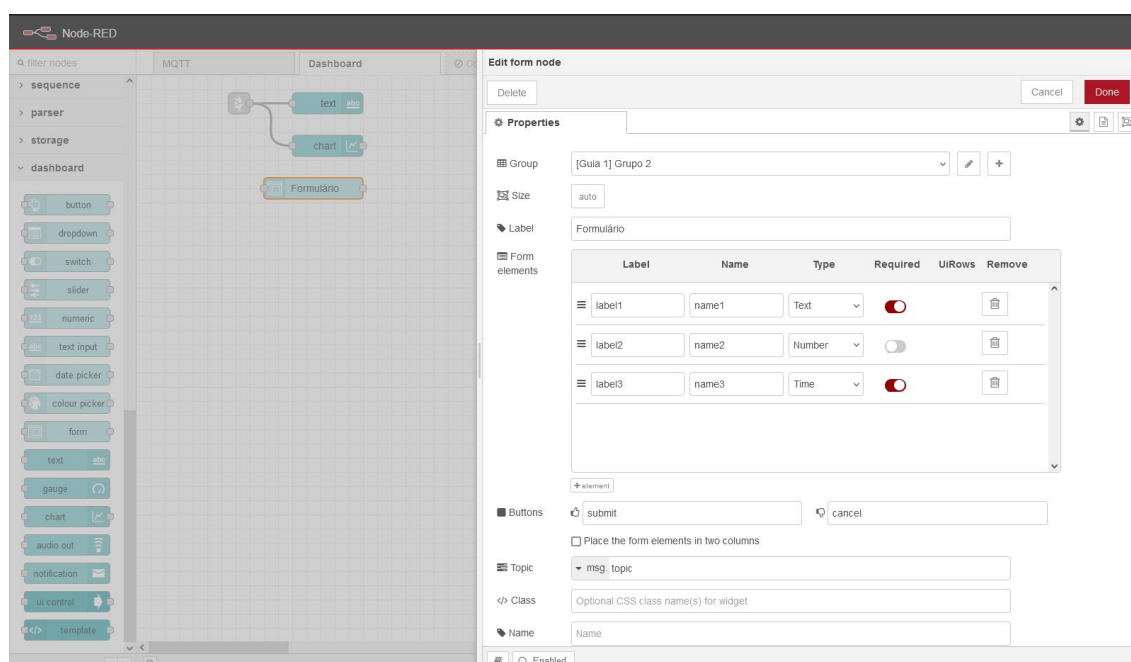


Figura 82 - Configuração do nó "*form*" no Node-RED
Fonte: autoria própria

Grupo 2

Formulário

label1 *

label2

label3 *

-- : --

SUBMIT CANCEL

Figura 83 - Interface gerada pelo nó "form" no *dashboard*
Fonte: autoria própria

Assim, observou-se que o campo “*label*” apresentado na Figura 82, correspondia a identificação desse respectivo campo na interface presente na Figura 83, enquanto que o campo “*name*” não podia ser identificado no *dashboard*. Isso se ocorreu por conta do fato desta última configuração estar relacionada com o nome a ser utilizado como identificador no JSON da mensagem gerada pelo nó “*form*”, após o usuário submeter alguma configuração, como destacado na Figura 84. Já a configuração “*Type*” estava relacionado com a tipologia de dado a ser inserida em cada campo no *dashboard*, de forma que ela pudesse ser configurada para aceitar apenas texto, números, e-mail, senhas, definições de data e hora, caixas de confirmação ou *switchs*. Por fim, o campo “*Required*” era utilizado para definir a obrigatoriedade ou não do preenchimento de seu respectivo elemento configurado no formulário.

```
msg.payload : Object
▶ { name1: "Teste 1", name2: 10, name3: "1970-01-01T21:00:00.000Z" }
```

Figura 84 - Mensagem de resposta do nó "form" no Node-RED
Fonte: autoria própria

Dessa forma, como exemplo da aplicação deste nó no desenvolvimento deste trabalho, na Figura 85 é apresentado os registradores que poderiam ser alterados durante o comando de limitação de potência ativa do inversor fotovoltaico, fazendo com seja necessário a utilização de múltiplos “*inputs*” em uma única operação de escrita. O primeiro corresponde ao efetivo valor a ser configurado para a limitação dessa potência, e neste caso, ela deve ser configurada como uma porcentagem entre 0 e 100% da capacidade máxima do inversor. Os próximos três parâmetros estão vinculados, respectivamente, ao intervalo de atraso para que configuração

seja implementada, o intervalo de tempo em que a configuração deve ser mantida para depois retornar a configuração anterior, e o intervalo para formar a rampa de alteração desta configuração. Além disso, ao realizar a parametrização de todos esses registradores, o último registrador apresentado na Figura 85 também deve ser alterado para 1, pois somente assim será enviado a confirmação para liberação do comando de limitação de potência.

Register	Name	Description	Units	Range of values
40233	WMaxLimPct	Set power output to specified level.	% WMax	0% - 100%
40234	WMaxLimPct_WinTms	Time window for power limit change.	Secs	0 - 300
40235	WMaxLimPct_RvrtTms	Timeout period for power limit.	Secs	0 - 28800
40236	WMaxLimPct_RmpTms	Ramp time for moving from current setpoint to new setpoint.	Secs	0 - 65534
40237	WMaxLim_Ena	Enumerated valued. Throttle enable/disable control.		0: Disabled 1: Enabled

Figura 85 - Amostra dos registradores para limitação da potência de saída do inversor fotovoltaico
Fonte: adaptado de [31]

Com base nisso, para que fosse possível encaminhar o envio de todos esses comandos ao inversor fotovoltaico, o fluxo apresentado na Figura 86 foi criado, em que o funcionamento dele se baseia na operação do nó de formulário descrito como “Limite de Potência Ativa”. Além deste nó, foram utilizados outros dois nós vinculados ao módulo instalado do Node-RED *Dashboard*, sendo eles o nó de “switch” e o nó “notification”, que na Figura 86 foram identificados por “Ativa Limitação” e “Aviso”, respectivamente. O nó “inject”, nomeado como “Início” na Figura 86, foi utilizado apenas uma vez, no início da execução de todo programa, como um gatilho para que esta interface fosse iniciada com o switch e o formulário bloqueado para alteração.

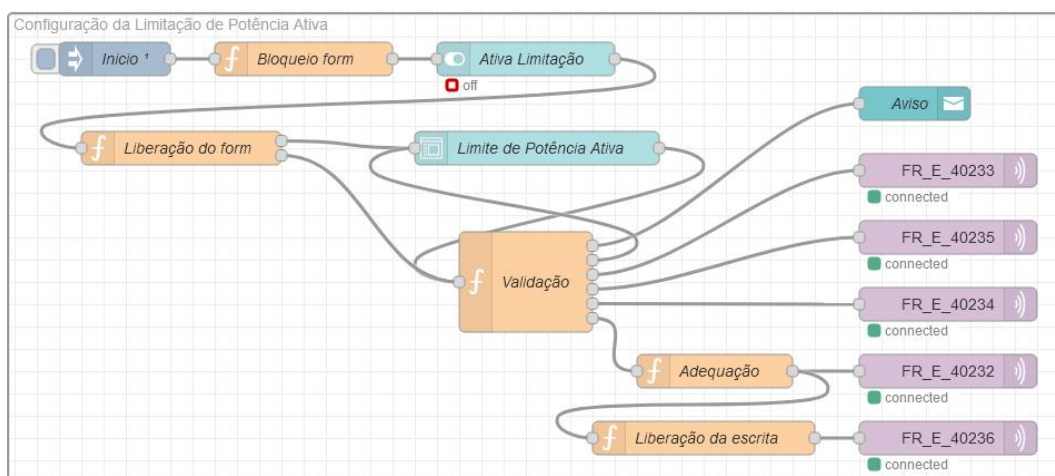


Figura 86 - Fluxo para a interface de limitação de potência do inversor fotovoltaico
Fonte: autoria própria

Ainda em relação ao fluxo apresentado na Figura 86, os nós de função, nomeados de “Bloqueio form” e “Liberação do form”, foram os responsáveis por permitir que o usuário inserisse as configurações de interesse de acordo com a posição do *switch*, além de garantir que os valores padrões sejam preenchidos em seus respectivos campos. Já o nó de função “Validação”, foi o responsável por checar se as configurações inseridas estavam em conformidade para o envio ao *broker*, sendo que em caso negativo, uma mensagem de aviso fosse enviada para a interface. Através desta função, também foi realizado a separação de cada informação inserida, de forma que elas fossem individualmente encaminhadas para o seu respectivo tópico no nó “*mqtt out*”. Por fim, o nó “Adequação” teve como função a conversão do valor de potência inserido em Watts pelo usuário, para porcentagem, tendo em vista os valores aceitos pelo registrador 40233, apresentado na Figura 85. Assim que este valor fosse convertido, um comando seria enviado para o nó “Liberação de escrita”, que, por sua vez, definiria a mensagem com valor de 1 para ser encaminhada ao registrador 40237 na Figura 85, e assim, permitir que todas as configurações fossem implementadas pelo inversor. De maneira complementar, no APÊNDICE C é apresentado o código em JavaScript utilizado em cada um desses nós de função, de forma que a interface final criada fosse utilizada para realizar a configuração imediata da limitação de potência na saída do inversor fotovoltaico.

Com base nessa metodologia apresentada, mais duas interfaces de controle foram implementadas considerando suas respectivas adequações do fluxo apresentado na Figura 86, totalizando três opções de controle para o inversor fotovoltaico. Dentre essas opções, estão incluídas a limitação de potência ativa do inversor, a alteração do fator de potência de operação, e a configuração da desconexão da saída do inversor fotovoltaico.

Em relação ao inversor híbrido, a mesma metodologia também foi implementada, porém de maneira mais simples do que no inversor fotovoltaico, tendo em vista que aquele possui parâmetros mais pontuais a serem configurados na comunicação Modbus. No caso deste trabalho, o fluxo apresentado na Figura 86 foi adaptado para fornecer a configuração de despacho das baterias do inversor híbrido, de forma a manter uma interface padronizada para este tipo de configuração.

Além disso, a definição de configurações como a liberação para carregamento das baterias, ou injeção na rede, puderam ser implementadas apenas com a utilização de nós do tipo “switch”, tendo em vista que apenas a informação de “Ativado” ou “Desativado” se faziam importantes para essa configuração.

Por fim, a configuração do modo de operação do inversor híbrido foi implementada com a utilização do nó “dropdown” do Node-RED *Dashboard*, que na prática, permite ao usuário escolher elementos de uma lista pré definida de dados. Neste contexto, foi desenvolvido uma interface em que o usuário poderia definir se o inversor híbrido deveria operar sem restrições, apenas como carregador das baterias, totalmente ilhado ou se ele deveria se desligar. Dessa forma, na Figura 87 é apresentado o fluxo utilizado para a implementação dessa interface. Nesta figura, pode-se notar a utilização auxiliar de um nó do tipo “link in”, cuja função é atualizar o nó “dropdown” para o parâmetro mais atual que se encontra configurado internamente no inversor híbrido.

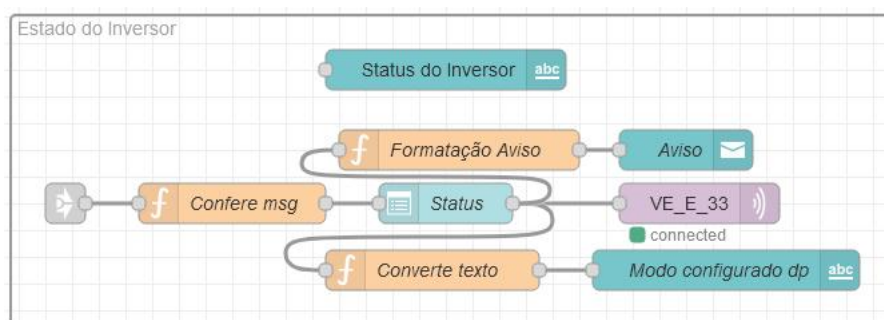


Figura 87 - Fluxo para interface de configuração do modo de operação do inversor híbrido
Fonte: autoria própria

Já a função “Confere msg”, presente na Figura 87, foi utilizada para verificar se o valor recebido no nó “link in” sofreu alguma alteração recente, de forma que a mensagem só prossiga no fluxo caso ela seja distinta da ocorrência anterior. Assim, essa função foi implementada como uma forma de evitar a redundância no recebimento dessa informação, tendo em vista que o nó “dropdown” só necessita de ser atualizado quando houver alguma

alteração concreta no registrador vinculado a essa configuração. Esta alteração, por sua vez, foi realizada no nó “*mqtt out*”, identificado com “VE_E_33” no fluxo presente na Figura 87, que depende da configuração realizada pelo usuário no “*dropdown*” presente na interface do *dashboard*, identificado como o nó “Status” no fluxo do editor do Node-RED. Assim, foi previsto que, assim que o usuário inserir uma determinada configuração no nó “*dropdown*”, esta informação seria devidamente encaminhada ao *broker*, que consecutivamente seria recebida pelo microcontrolador como um comando de escrita. Ao receber este comando de escrita, a transação Modbus com o inversor híbrido seria iniciada, de forma a alterar os dados do registrador de interesse, que neste caso estaria relacionado ao endereço 33. Com o sucesso dessa escrita, e um novo ciclo de leitura do sistema de supervisão realizado, essa informação seria repassada por todos os nós dos recursos de supervisão da interface, até chegar ao nó de “*link in*” destacado na Figura 87, e assim atualizar o valor apresentado na interface final.

As demais funções criadas na Figura 87, por sua vez, estiveram relacionadas com a formatação de uma mensagem de aviso quando as configurações forem enviadas ao inversor, e a formatação de um campo de texto para apresentar uma breve descrição do modo que se encontra configurado o inversor híbrido. Assim, todos os códigos utilizados para essas funções podem ser encontrados no APÊNDICE D.

Com base nisso, tem-se apresentado todas as informações necessárias para a implementação dos grupos de supervisão e controle que foram utilizados na elaboração da interface criada no Node-RED deste trabalho.

5.1.3.3 Implementação dos recursos de programação horária

Como parte final dos recursos que envolvem a supervisão e o controle dos inversores instalados na microrrede, implementou-se uma guia específica para permitir que o usuário consiga programar um intervalo do dia para que uma determinada configuração seja implementada. Assim, esta funcionalidade busca oferecer uma ferramenta mais útil ao usuário, no que diz respeito a necessidade de se realizar certas configurações em períodos determinados de tempo, como o despacho da energia armazenada nas baterias durante o horário de ponta, por exemplo.

Com base nisso, na Figura 88 é destacado o fluxo utilizado para permitir a programação horária da limitação de potência do inversor fotovoltaico. Sendo assim, pode-se

notar que este fluxo foi totalmente baseado naquele apresentado na Figura 86, com o único acréscimo de dois nós, identificados como “2s” e “Verificação do horário”. O primeiro nó, neste contexto, se trata de um nó do tipo “inject”, e que foi utilizado para enviar um sinal de verificação a cada intervalo de dois segundos ao nó de função conectado a ele. Este nó de função, por sua vez, foi inserido com o objetivo de verificar se o horário atual corresponde ao intervalo configurado pelo usuário. Em caso positivo, a configuração definida pelo usuário no formulário “Limite de Potência Ativa” seria enviada ao inversor, de forma que esta configuração fosse mantida durante todo o intervalo configurado. Com o fim do período programado, as configurações inseridas deixavam de possuir validade, e as configurações padrões de fábrica seriam novamente inseridas como parâmetros de operação para o dispositivo.

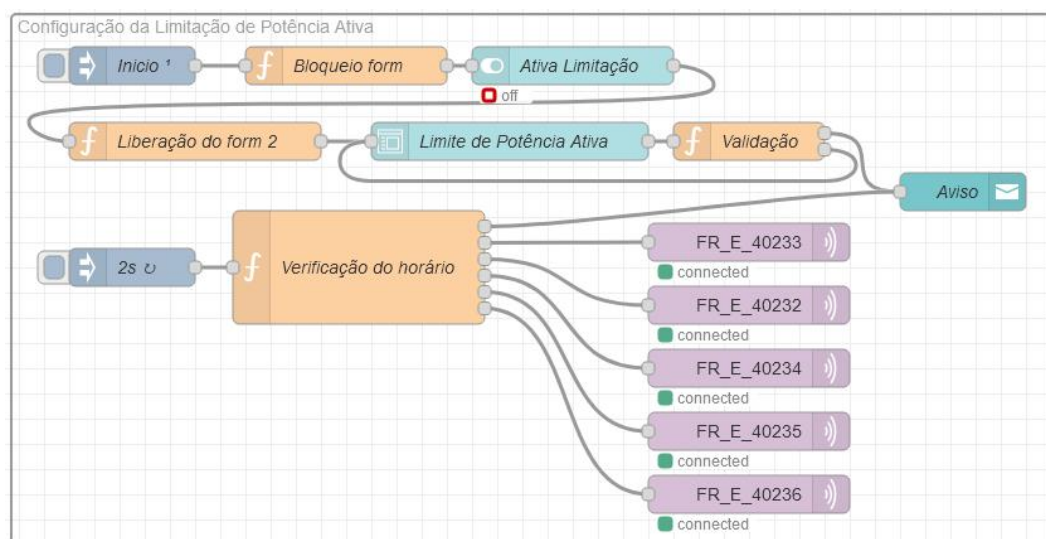


Figura 88 - Fluxo para interface de programação horária de limitação de potência do inversor fotovoltaico
Fonte: autoria própria

Dessa forma, no APÊNDICE E é apresentado o código das funções utilizadas para a programação do fluxo ilustrado na Figura 88, de forma que o fluxograma destacado na Figura 89, represente o algoritmo utilizado no nó de função denominado “Verificação do horário”. Portanto, assim como nas situações anteriores, este fluxo foi utilizado como a estrutura básica, havendo as suas respectivas alterações, para a implementação das outras interfaces de programação horária, que neste caso envolvem a alteração do fator de potência do inversor fotovoltaico, e o despacho de energia armazenada nas baterias.

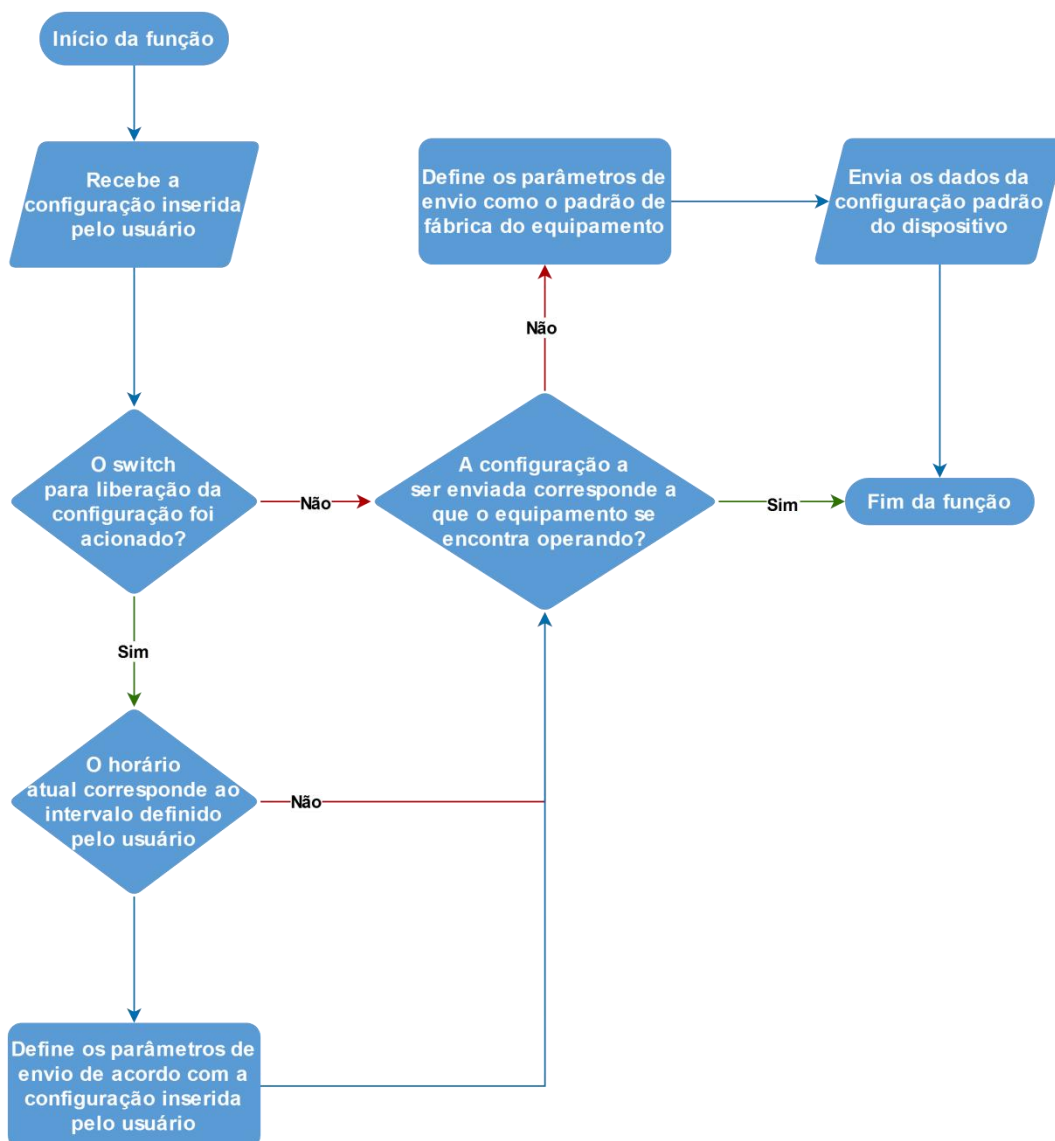


Figura 89 - Fluxograma do algoritmo de verificação do nó "Verificação do horário"
 Fonte: autoria própria

Com base nisso, foi apresentado todas as informações necessárias para a implementação dos grupos de programação horária que foram utilizados na elaboração da interface criada no Node-RED.

5.1.4 Desenvolvimento da interface de consulta histórica

Tendo em vista todas informações apresentadas, a última etapa do desenvolvimento da interface web de supervisão e controle deste trabalho consistiu na implementação de uma guia específica para consultar dados históricos dos inversores que compõem a microrrede.

Para isso, deve-se garantir o acesso a um banco de dados confiável, que permita que o sistema seja capaz de realizar o armazenamento e a requisição de maneira segura dos dados de interesse. Pensando nisso, optou-se por utilizar de um banco de dados instalado localmente, na mesma máquina em que o *broker* e o servidor Node-RED se encontram operando, de forma a facilitar todo o processo de estudo, modelagem e integração do banco de dados com o sistema desenvolvido.

Neste caso, para o desenvolvimento desta parte do trabalho, utilizou-se do Software XAMPP, que se trata de uma ferramenta livre de licenças de aquisição, que permite a instalação de um servidor web Apache em sistemas Windows, Linux ou MAC. Com a instalação deste software, um dos recursos disponibilizados se trata de um servidor MySQL, que permite a criação e gerenciamento de um banco de dados baseado em código aberto, ideal para aplicação deste trabalho [49][50]. Assim, com a implementação deste software no computador instalado no laboratório de Fontes Renováveis, tornou-se possível prosseguir com a implementação da interface de consultas históricas, uma vez que todos os serviços necessários se encontravam conectados na mesma rede.

5.1.4.1 Instalação e Modelagem do Banco de Dados

Para a utilização do banco de dados MySQL disponível no XAMPP, faz-se necessário a instalação do software disponibilizado em [51], que no caso deste trabalho, correspondeu a versão 8.2.12-0-VS16 para Windows. Durante o processo de instalação do software, iniciou-se o arquivo executável como administrador, de forma a garantir que a opção “MySQL” estivesse marcada, conforme destacado na Figura 90.

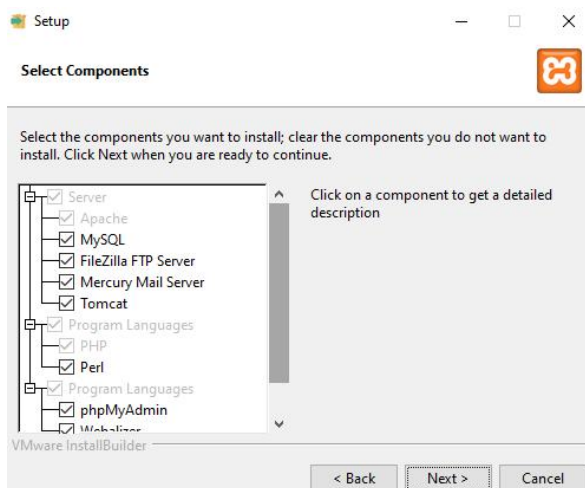


Figura 90 - Instalação do banco de dados MySQL no XAMPP
Fonte: autoria própria

Com a instalação concluída, realizou-se a abertura do painel de controle do XAMPP, de forma que a interface se assemelhasse a visível na Figura 91.

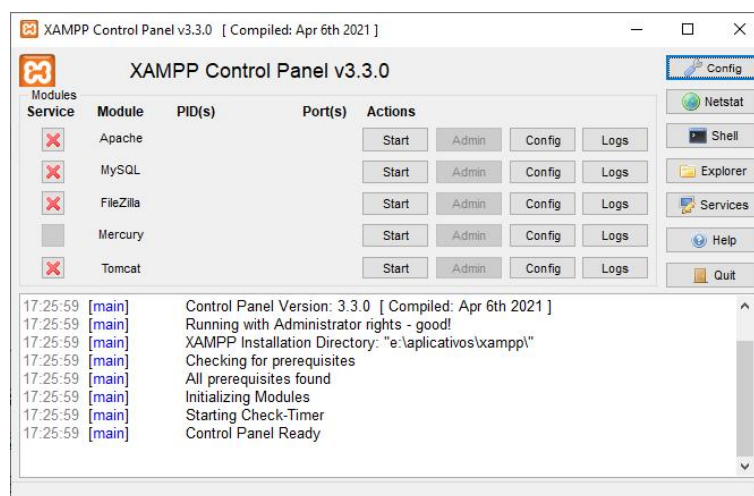


Figura 91 - Painel de controle do XAMPP
Fonte: autoria própria

No contexto do painel de controle na Figura 91, para que fosse possível liberar o acesso ao banco de dados, iniciou-se os serviços “Apache” e “MySQL”, através dos botões de ação “Start”. Com os dois serviços iniciados, o botão “Admin” foi liberado, de forma que, ao clicar sobre aquele vinculado ao serviço do MySQL, a interface para gerenciamento do banco de dados fosse aberta no navegador de maneira automática. Dessa forma, na Figura 92 é apresentada a interface para o gerenciamento do banco de dados MySQL que foi disponibilizada pelo XAMPP neste processo, e que também foi utilizada para o desenvolvimento deste trabalho.

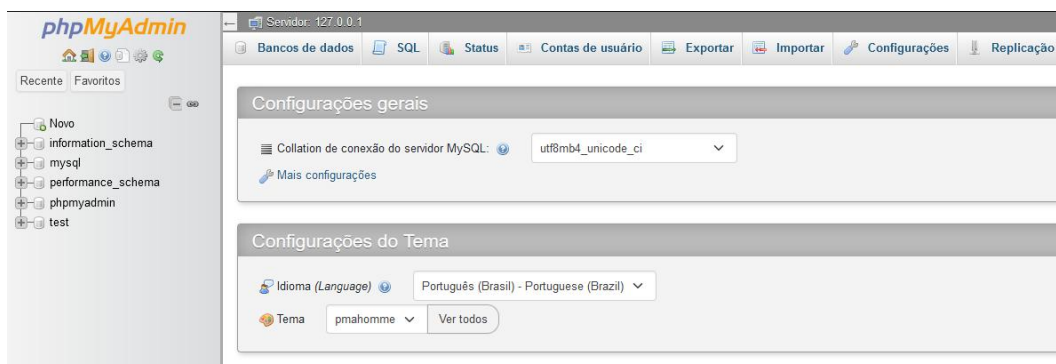


Figura 92 - Interface de gerenciamento do banco de dados MySQL
Fonte: autoria própria

Com a interface de gerenciamento do banco de dados aberta, a próxima etapa de configuração do sistema consistiu na definição da autenticação de acesso ao banco de dados, por meio da configuração “Contas de Usuário”, destacada na região superior central da Figura 92. Ao selecionar esta configuração, a interface ilustrada na Figura 93 foi aberta, de forma que a configuração de interesse se encontrasse disponível no campo “Adicionar conta de usuário”, localizada na região inferior da Figura 93.

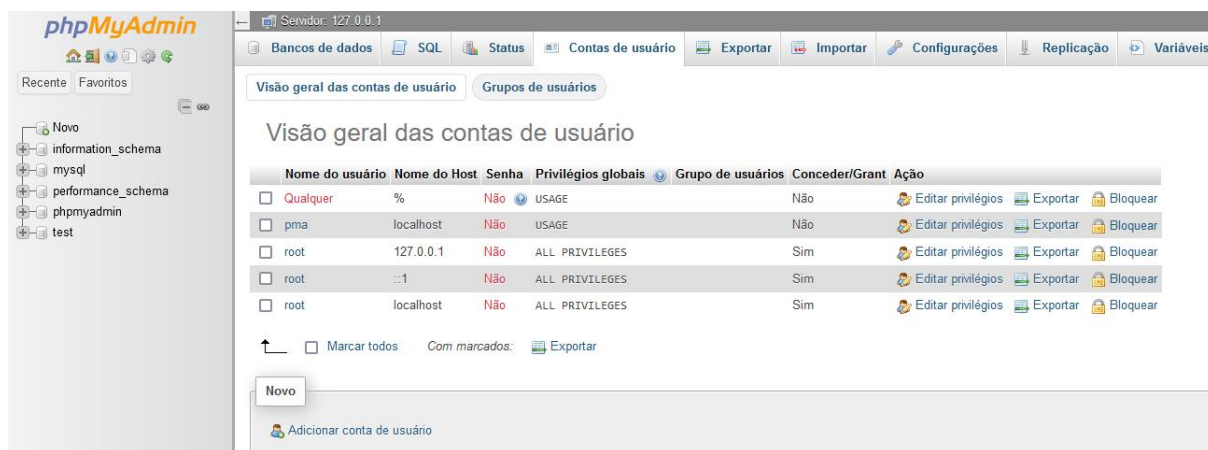


Figura 93 - Adição de conta de usuário para acesso ao banco de dados MySQL
Fonte: autoria própria

Ao selecionar a opção para adicionar uma nova conta de usuário, uma nova interface foi apresentada, em que se tornou possível a definição das informações de *login* dele, como o nome, senha e privilégios que poderiam ser acessados.

Com base nisso, na Figura 94 é destacada todas as configurações que foram realizadas para o usuário que foi implementado para ser utilizado pelo Node-RED para acessar o banco de dados, de forma que, ao término da realização dessas configurações, o botão de “Executar”

fosse pressionado, validando as alterações inseridas. Destaca-se, em tempo, que a configuração “Nome do host” foi configurada como “Qualquer host” pois esta definição garante que outros dispositivos consigam acessar o banco de dados através da rede implementada no laboratório. Além disso, no campo “Privilégios globais”, foi definido que o usuário criado apenas poderia utilizar dos comandos relacionados com o manejo de dados no sistema, tendo em vista que a aplicação do banco de dados no Node-RED envolve apenas o registro e a requisição de informações. Dessa forma, os outros comandos que envolviam a estrutura e o gerenciamento do banco, não poderiam ser utilizados por este usuário, até mesmo por uma questão de segurança.

Assim, com o usuário e a senha definidos de acordo com o procedimento na Figura 94, a próxima etapa do trabalho consistiu na criação de um novo banco de dados para armazenar as informações de interesse, relacionadas a operação dos inversores na microrrede. Para isso, bastou selecionar a opção “Novo”, presente no canto superior esquerdo da Figura 92, de forma que a página fosse redirecionada para a interface apresentada na Figura 95.

Com base nisso, para prosseguir com a criação do banco de dados, inseriu-se um nome para ele no campo reservado para esta configuração na Figura 95, e depois foi realizada a validação desta configuração através do botão “Criar”. Logo após a confirmação desta ação, uma nova interface foi apresentada, solicitando um nome para tabela, e a quantidade de colunas que ela poderia possuir.

No caso deste trabalho, foi previsto a utilização de duas tabelas, nomeadas de “inversor_fr” e “inversor_ve”, que deveriam armazenar os dados do inversor fotovoltaico e híbrido, respectivamente. Para cada uma dessas tabelas, foi previsto um conjunto de quatro colunas, identificadas como “id”, “topico”, “valor” e “timestamp”, que, por sua vez, armazenariam os respectivos dados de interesse de uma determinada informação. Assim, é esperado uma estrutura final para este banco de dados como a apresentada na Figura 96, em que o banco de dados criado deveria armazenar duas tabelas distintas, que por sua vez, devem guardar o mesmo tipo de dado em cada uma das suas 4 colunas.

Dessa forma, o procedimento utilizado para a criação da primeira tabela, referente ao armazenamento dos dados do inversor fotovoltaico, prosseguiu de acordo com o apresentado na Figura 97, através do botão “Criar”, levando em consideração o nome da tabela, e a quantidade de colunas que ela deveria possuir.

phpMyAdmin

Recente Favoritos

Novo

- information_schema
- mysql
- performance_schema
- phpmyadmin
- test

Servidor: 127.0.0.1

Bancos de dados SQL Status Contas de usuário Exportar

Adicionar conta de usuário

Informação de login

Nome de usuário: Usar campo texto user-teste

Nome do Host: Qualquer host %

Senha: Usar campo texto Força:

Re-digite:

Plugin de autenticação: Autenticação nativa do MySQL

Gerar senha: Gerar

Banco de dados por conta de usuário

☐ Criar banco de dados com o mesmo nome e conceder todos os privilégios.

☐ Conceder todos os privilégios no nome coringa (nome_do_usuario_%).

Privilegios globais ☐ Marcar todos

Nota: os nomes de privilégios do MySQL são expressos em inglês.

Dados	Estrutura	Administração
☒ SELECT	☐ CREATE	☐ GRANT
☒ INSERT	☐ ALTER	☐ SUPER
☒ UPDATE	☐ INDEX	☐ PROCESS
☒ DELETE	☐ DROP	☐ RELOAD
☒ FILE	☐ CREATE TEMPORARY TABLES	☐ SHUTDOWN
	☐ SHOW VIEW	☐ SHOW DATABASES
	☐ CREATE ROUTINE	☐ LOCK TABLES
	☐ ALTER ROUTINE	☐ REFERENCES
	☐ EXECUTE	☐ REPLICATION CLIENT
	☐ CREATE VIEW	☐ REPLICATION SLAVE
	☐ EVENT	☐ CREATE USER
	☐ TRIGGER	

Console

Executar

Figura 94 - Configuração do usuário de acesso ao banco de dados via Node-RED
Fonte: autoria própria

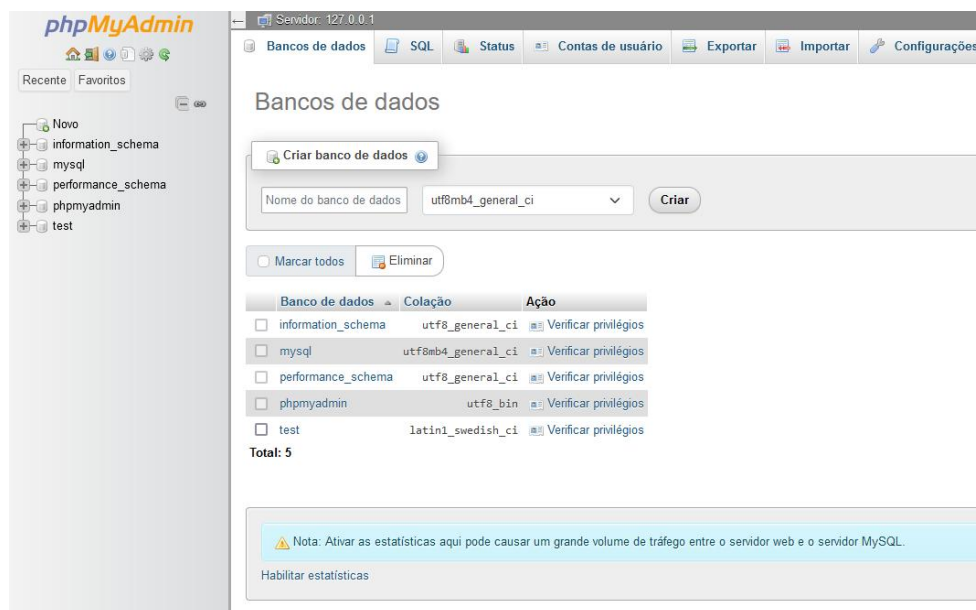


Figura 95 - Interface para criação de um novo banco de dados
Fonte: autoria própria

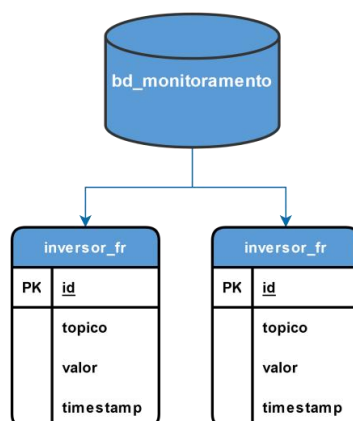


Figura 96 - Esquema do banco de dados implementado
Fonte: autoria própria

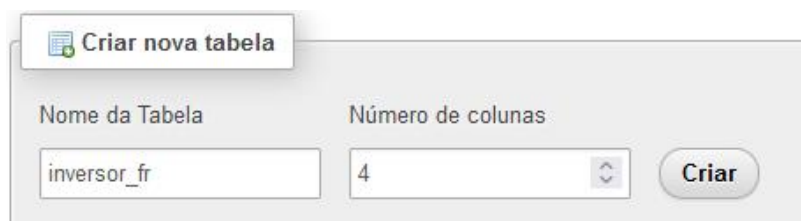


Figura 97 - Criação de uma nova tabela para o banco de dados
Fonte: autoria própria

Com isso, a próxima etapa consistiu na especificação das colunas da tabela criada na Figura 97, e para isso, fez-se necessário conhecer o tipo de dado que seria armazenado em cada coluna. No caso deste trabalho, para a primeira coluna (id), houve a identificação de chave primária de cada registro realizado nesta tabela, e por isso ela foi configurada como um

número do tipo inteiro (INT), com a configuração de “Índice” definida como “Primary”, e o campo A_I preenchido (*Auto_Increment*). Já a segunda coluna (topico), armazenaria a informação a respeito do tópico na comunicação MQTT que aquele dado armazenado na terceira coluna (valor) está vinculado. Assim, a segunda coluna foi definida como do tipo “VARCHAR”, com tamanho de 15 caracteres, enquanto a terceira coluna foi configurada como do tipo FLOAT. Por fim, a última coluna da tabela (timestamp) teve de ser configurada como do tipo TIMESTAMP, tendo em vista que ela armazenaria o momento exato em que estas dados fossem registrados. Para isso, bastou configurar o campo “Padrão” para CURRENT_TIMESTAMP, de forma que a cada registro realizado na tabela, esta coluna fosse automaticamente preenchida com a data e hora do momento de armazenamento dos dados.

Com base nisso, na Figura 98 é apresentada a definição final dessas configurações das colunas da tabela na interface de gerenciamento do banco de dados, fazendo com que fosse necessário apenas validar essas configurações no sistema ao apertar o botão “Salvar”, localizado no canto inferior esquerdo da figura.

Nome	Tipo	Tamanho/Valores	Padrão	Colação	Atributos	Nulo	Índice	A_I	Comentários
id	INT		Nenhum			☐	PRIMARY	☒	
topico	VARCHAR	15	Nenhum			☐	---	☐	
valor	FLOAT		Nenhum			☐	---	☐	
timestamp	TIMESTAMP		CURRENT_TIMESTAMP			☐	---	☐	

Comentários de tabela: Colação: Mecanismo de armazenamento: InnoDB

Definição da PARTIÇÃO: (Expressão ou lista de col)

Partição por: Partições:

Ver SQL Salvar

Figura 98 - Especificação das colunas das tabelas do banco de dados
Fonte: autoria própria

Com a confirmação das configurações realizadas, a última interface apresentada foi a de visão geral da estrutura da tabela criada, conforme ilustrado na Figura 99. Para prosseguir com a criação da tabela “inversor_ve”, referente aos dados do inversor híbrido, bastou clicar sobre o campo “Nova”, destacado com a cor amarela na estrutura da árvore situada no canto

esquerdo da Figura 99, e assim prosseguir com as mesmas etapas que foram utilizadas para a tabela “inversor_fr”.

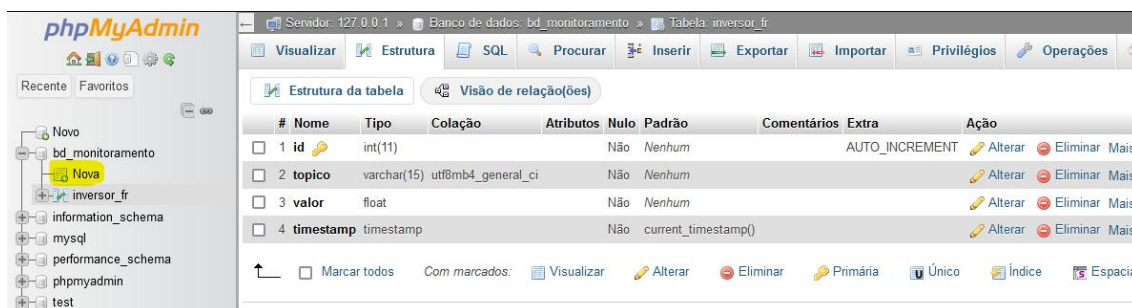


Figura 99 - Criação de uma nova tabela no banco de dados "bd_monitoramento"
Fonte: autoria própria

Com isso, realizou-se os procedimentos para a modelagem do banco de dados utilizado neste trabalho, de forma que, para prosseguir com o desenvolvimento da interface de consultas históricas, houve a necessidade de realizar a instalação de um novo conjunto de nós no editor do Node-RED.

5.1.4.2 Inserção de registros no banco de dados a partir do Node-RED

Com o banco de dados criado, a próxima etapa do trabalho consistiu na vinculação do Node-RED com este sistema, de forma que fosse possível armazenar as informações de interesse em suas respectivas tabelas. Para isso, fez-se necessário a instalação de um novo nó no editor do Node-RED, denominado de “node-red-node-mysql”, seguindo a mesma metodologia apresentada na seção 5.1.3.

Após a instalação, o nó pôde ser encontrado no grupo “storage”, no canto esquerdo da interface do editor do Node-RED, conforme destacado na Figura 100. Nesta figura, nota-se que o nó possui uma conexão de entrada, e uma de saída, de forma que, ao realizar uma requisição ao banco de dados, a resposta para ela seja encaminhada pela saída do nó MySQL.

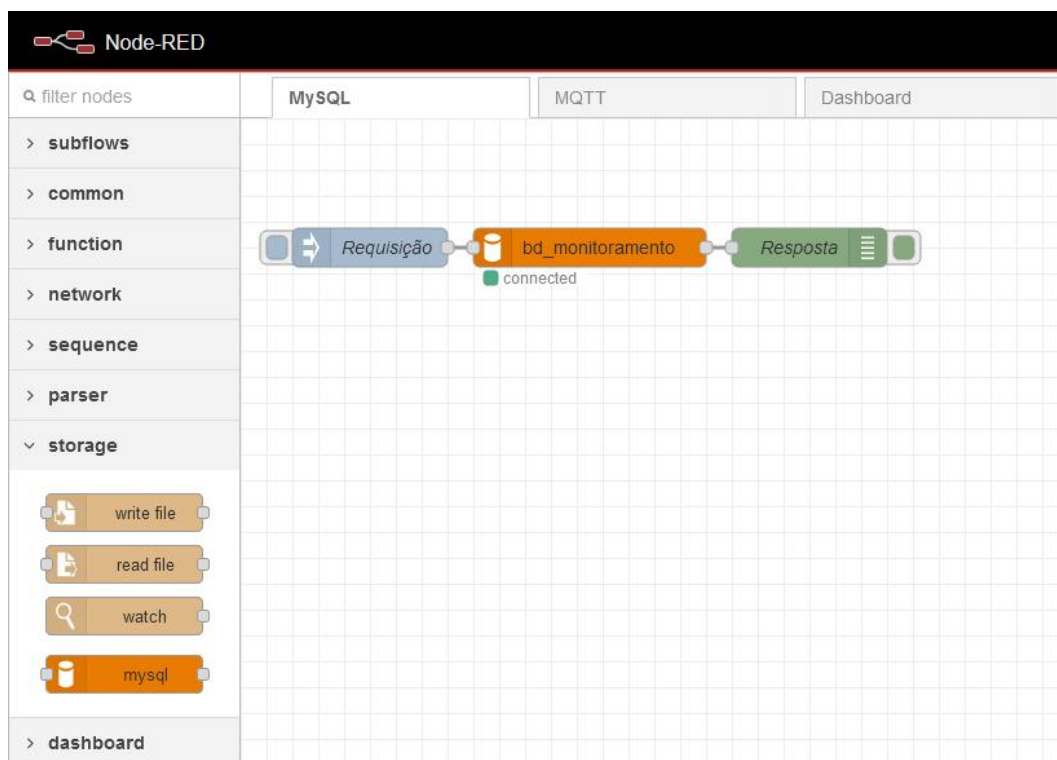


Figura 100 - Estrutura básica de utilização do nó "mysql"
Fonte: autoria própria

Em relação a configuração do nó “mysql”, esta foi realizada ao selecionar o nó quando ele já se encontrava implementado no campo de edição de fluxos do Node-RED. Assim, tornou-se possível adicionar um novo banco de dados no campo específico, de forma que as configurações a serem realizadas, fossem apresentadas de acordo com as informações presentes na Figura 101.

Assim, tendo em vista as configurações que podem ser realizadas no nó “mysql” o campo “name” foi preenchido com um nome utilizado apenas como meio para identificação do nó no fluxo do editor do Node-RED. Já os campos “Host” e “Port” foram preenchidos com o IP do dispositivo na rede em que o banco de dados se encontra instalado, assim como a porta utilizada por ele para o recebimento e envio de dados. Esta última configuração foi definida de acordo com a informação presente na Figura 101, tendo em vista que ela se trata da porta padrão para a comunicação com o banco de dados. Em relação aos campos “User” e “Password”, estes foram alterados de acordo com o usuário e a senha definidos no banco de dados, procedimento este que foi ilustrado na Figura 94. De maneira complementar, o campo “database” foi configurado levando em consideração o nome estabelecido para o banco de dados, conforme a configuração realizada na Figura 95. Por fim, os últimos dois campos foram mantidos de acordo com o padrão do nó apresentado na Figura 101, sem a necessidade, portanto, de alteração.

Figura 101 - Configuração do nó "mysql" no Node-RED
Fonte: autoria própria

Com o nó MYSQL configurado, restou-se a partir de agora, a realização da modelagem dos fluxos que foram utilizados para as operações de registro e requisição de dados. Para a primeira dessas operações, a estrutura básica utilizada no desenvolvimento deste trabalho pode ser encontrada na Figura 102. Com base nesta figura, tem-se que a configuração do nó “*link in*” foi realizada para extrair a informação desejada do fluxo MQTT. Este fluxo, por sua vez, disponibilizou todos os dados da comunicação do microcontrolador com os inversores, conforme a metodologia apresentada na Figura 71.

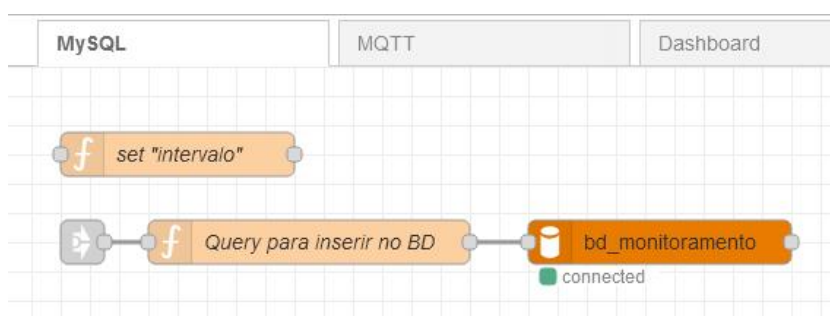


Figura 102 - Fluxo para o registro de dados no banco de dados via Node-RED
Fonte: autoria própria

Assim, no momento em que uma informação fosse publicada no tópico configurado no nó “*link in*” presente na Figura 102, ela seria direcionada para o bloco de função denominado “Query para inserir no BD”, que por sua vez, foi utilizado para realizar a formatação da mensagem para a operação de registro desta informação no banco de dados. Esta operação de registro no banco foi viabilizada de acordo com a utilização do nó “bd_monitoramento”.

Em relação ao nó de função “set ‘intervalo’”, este foi utilizado como uma ferramenta para definir um intervalo, em milissegundos, que, por sua vez, seria utilizado como parâmetro na função implementada no nó “Query para inserir no BD”. Dessa forma, com a junção dos códigos inseridos nos dois nós de função na Figura 102, obteve-se um algoritmo completo que realiza a média de toda informação recebida do nó “link in” a cada quinze minutos, sendo que, somente após esse intervalo, o registro no banco de dados seria efetivamente realizado, levando em consideração a média calculada para este mesmo intervalo. Assim, as funções em JavaScript utilizadas nestes nós, podem ser encontradas no APÊNDICE F, enquanto o fluxograma apresentado na Figura 103 realiza uma descrição visual do algoritmo criado para essa operação de registro.

Em tempo, destaca-se que, para qualquer operação envolvendo a comunicação com o banco de dados via nó “mysql”, fez-se necessário a utilização da propriedade “msg.topic” do Node-RED, tendo em vista que este se trata de um padrão para a utilização deste nó. Assim, diferentemente dos demais nós utilizados neste trabalho, a propriedade “msg.payload” não possuiu utilidade para operações que envolvem o tratamento direto com o banco de dados.

Dessa forma, foi apresentado toda metodologia necessária para implementar os fluxos de registro de dados do Node-RED para o banco de dados modelado. No caso deste trabalho, elaborou-se o fluxo para a ocorrência de 22 operações de registros, sendo 12 dessas operações para o inversor fotovoltaico, e 10 para o inversor híbrido, que em ambos os casos, envolveram parâmetros de tensão, corrente e potência de operação dos componentes conectados a esses dispositivos. Com isso, espera-se fornecer uma ferramenta capaz de auxiliar no entendimento da operação da microrrede como um todo, fazendo com que a próxima etapa do trabalho consistisse na viabilização da requisição dessas informações armazenadas para consultas históricas.

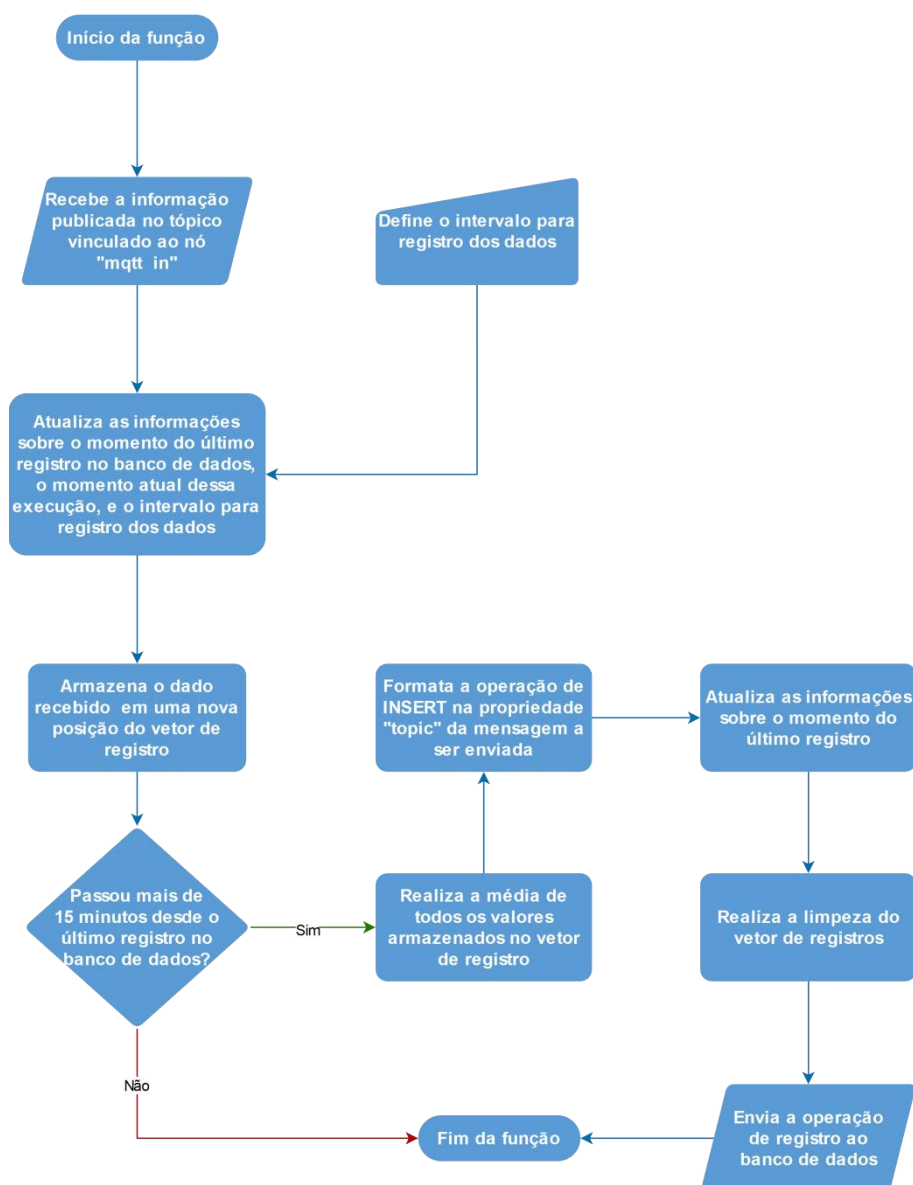


Figura 103 - Fluxograma para operação de registro no banco de dados via Node-RED
Fonte: autoria própria

5.1.4.3 Requisição de registros do banco de dados a partir do Node-RED

Com os dados de operação dos inversores armazenados em suas respectivas tabelas, surgiu, a partir de então, a necessidade de viabilizar um fluxo no Node-RED que fosse capaz de buscar esses dados através de uma interface específica. Além disso, pensando em uma melhor experiência com o usuário, fez-se necessário que essas informações sejam apresentadas de uma maneira visual, de forma que o usuário seja capaz de filtrar as informações que interessam a ele.

Pensando nessas situações, foi desenvolvido uma interface que oferece ao usuário a possibilidade realizar múltiplas consultas dentre os parâmetros armazenados no banco de dados. Além disso, foi previsto a possibilidade de ele ainda selecionar um intervalo composto por data e hora para visualizar os registros de interesse, que por sua vez, foram tratados para serem apresentados em uma interface gráfica semelhante a utilizada na Figura 81.

Com base nisso, o fluxo desenvolvido para viabilizar essa interface de consultas históricas pôde ser dividido em duas partes, sendo uma para receber as informações de filtro do usuário, e outra para realizar a formatação da operação de requisição ao banco de dados, juntamente com a formatação dos dados recebidos para a apresentação em um gráfico de linha.

Assim, a primeira parte deste fluxo pode ser observada na Figura 104, havendo destaque para os três tipos de nós que foram implementados, sendo eles o nó “*dropdown*”, já utilizado anteriormente, e o nó do tipo “*text input*”, ambos do grupo “*dashboard*” instalado na seção 5.1.3 deste trabalho. Por fim, também foi utilizado um nó de função para cada um dos outros nós, de forma que eles sejam implementados com o objetivo de armazenar o valor configurado pelo usuário nessas outras interfaces.

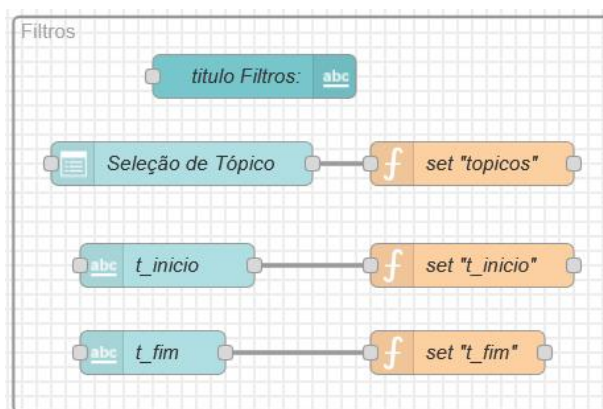


Figura 104 - Primeira parte do fluxo para consulta histórica via Node-RED
Fonte: autoria própria

Tendo como base parte do fluxo apresentado na Figura 104, na Figura 105 é demonstrado a configuração realizada para o nó de “*dropdown*”, que corresponde a interface em que o usuário passa a ter a capacidade de filtrar as informações que ele deseja realizar a consulta. Nesta figura, nota-se que no campo “*Options*” duas colunas foram preenchidas, de forma que a primeira corresponda ao tópico a ser buscado dentro da sua respectiva tabela no banco de dados, enquanto a segunda coluna apresenta as informações a serem visualizadas pelo usuário na interface. Assim, a configuração realizada neste nó já foi estabelecida de forma a gerar uma relação direta entre esses dois parâmetros, de forma a tornar o processo de

formatação da requisição mais facilitado. Em tempo, destaca-se que validação da configuração “*Allow multiples selections from list*” foi selecionada, possibilitando que múltiplos tópicos sejam encaminhados em uma única consulta ao banco de dados.

Edit dropdown node

Delete Cancel Done

Properties

Group: [Histórico] Seleção do intervalo

Size: auto

Label: Tópico

Tooltip: optional tooltip

Placeholder: Select option

Options:

- FR_L_40071 FV - Corrente CA
- FR_L_40087 FV - Potência CA (VA)
- FR_L_40089 FV - Potência CA (VAr)
- FR_L_40083 FV - Potência CA (W)
- FR_L_40100 FV - Potência CC

+ option

Allow multiple selections from list: ☒

If msg arrives on input, pass through to output: ☒

Figura 105 - Configuração do nó "dropdown" da interface de consulta histórica no Node-RED
Fonte: autoria própria

Em relação aos nós do tipo “*text input*”, estes apenas foram configurados para receber a informação se eles estão relacionados ao período inicial ou final do intervalo de requisição. Além disso, foi necessário defini-los para operar sobre o modo “*datetime picker*”, de forma a formatar a interface automaticamente para o recebimento de data e hora pelo usuário. Essas configurações podem ser visualizadas na Figura 106, e possuem validade para os dois nós do tipo do tipo “*text input*”, utilizados na Figura 104.

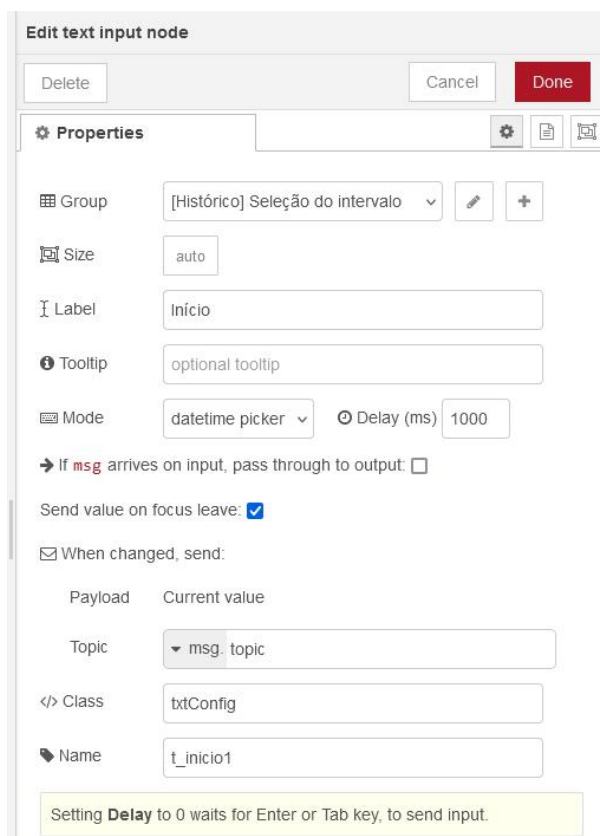


Figura 106 - Configuração do nó "text input" da interface de consulta histórica utilizada no Node-RED
Fonte: autoria própria

Dessa forma, o restante do fluxo pode ser visualizado na Figura 107, em que os nós localizados a direita da figura representam as funcionalidades de formatação da operação de requisição de dados com o banco, em conjunto a apresentação dessa informação requisitada em um gráfico.

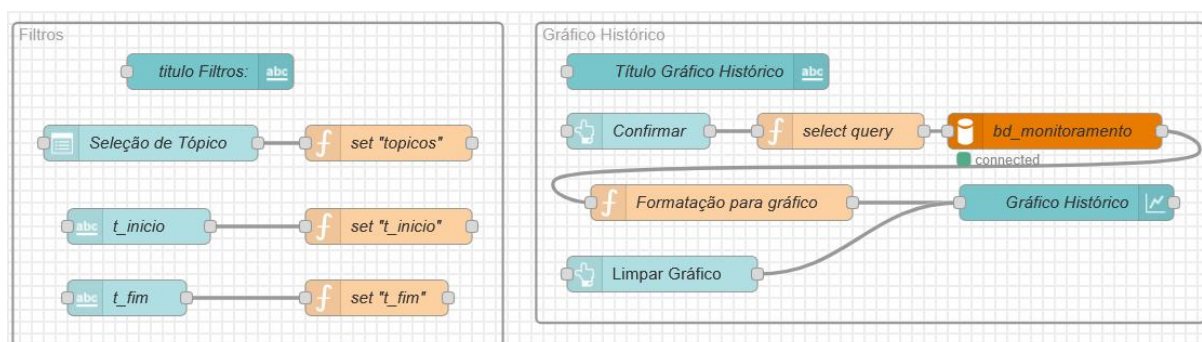


Figura 107 - Fluxo completo da interface de consulta histórica via Node-RED
Fonte: autoria própria

Na Figura 107, nota-se a utilização de um nó do tipo “button”, denominado de “Confirmar”, sendo que este foi configurado como o responsável por iniciar todo o processo

de requisição com o banco de dados, logo após o usuário realizar o preenchimento dos filtros apresentados no fluxo presentes na Figura 104. Após isso, o nó “*select query*” foi implementado para realizar toda a formatação da mensagem de acordo com os *inputs* do usuário. Dessa maneira, com a resposta do servidor, o nó “Formatação para gráfico” se tornou o responsável por formatar todos os dados recebidos do banco de dados, para o padrão adequado para ser utilizado na interface do nó “Gráfico Histórico”. Essa formatação, por sua vez, foi feita de acordo com a documentação do nó “*chart*” do Node-RED *Dashboard*, disponibilizada em [52]. Com base nisso, a configuração do botão de confirmação utilizado neste trabalho pode ser encontrada na Figura 108, enquanto que o código utilizado nos nós de função, estão disponíveis no APÊNDICE G.

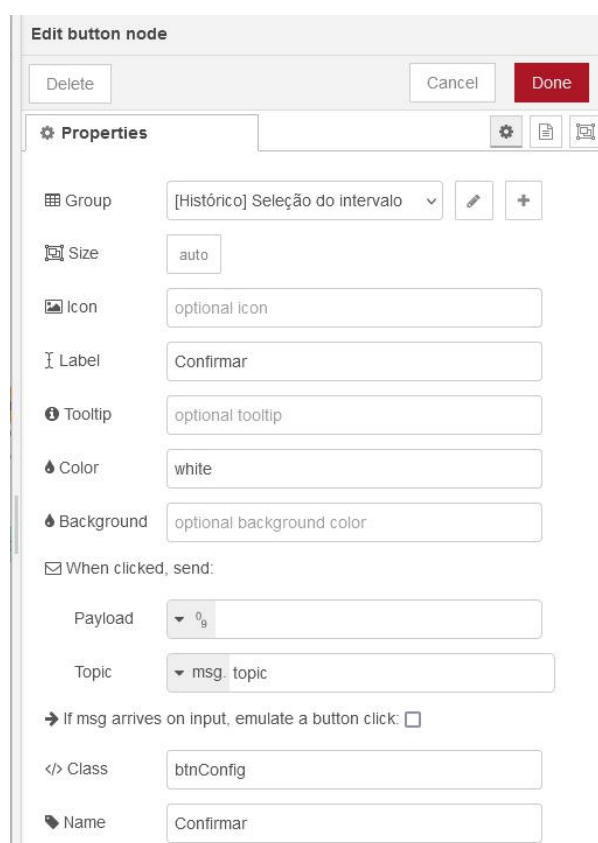


Figura 108 - Configuração do botão de confirmação da interface de consulta histórica do Node-RED
Fonte: autoria própria

Além disso, o fluxograma que representa o algoritmo utilizado pelo fluxo na Figura 107 pode ser encontrado na Figura 109, finalizando, assim, todas as configurações necessárias para realizar a implementação da interface de consulta histórica no sistema de supervisão e controle dos inversores.

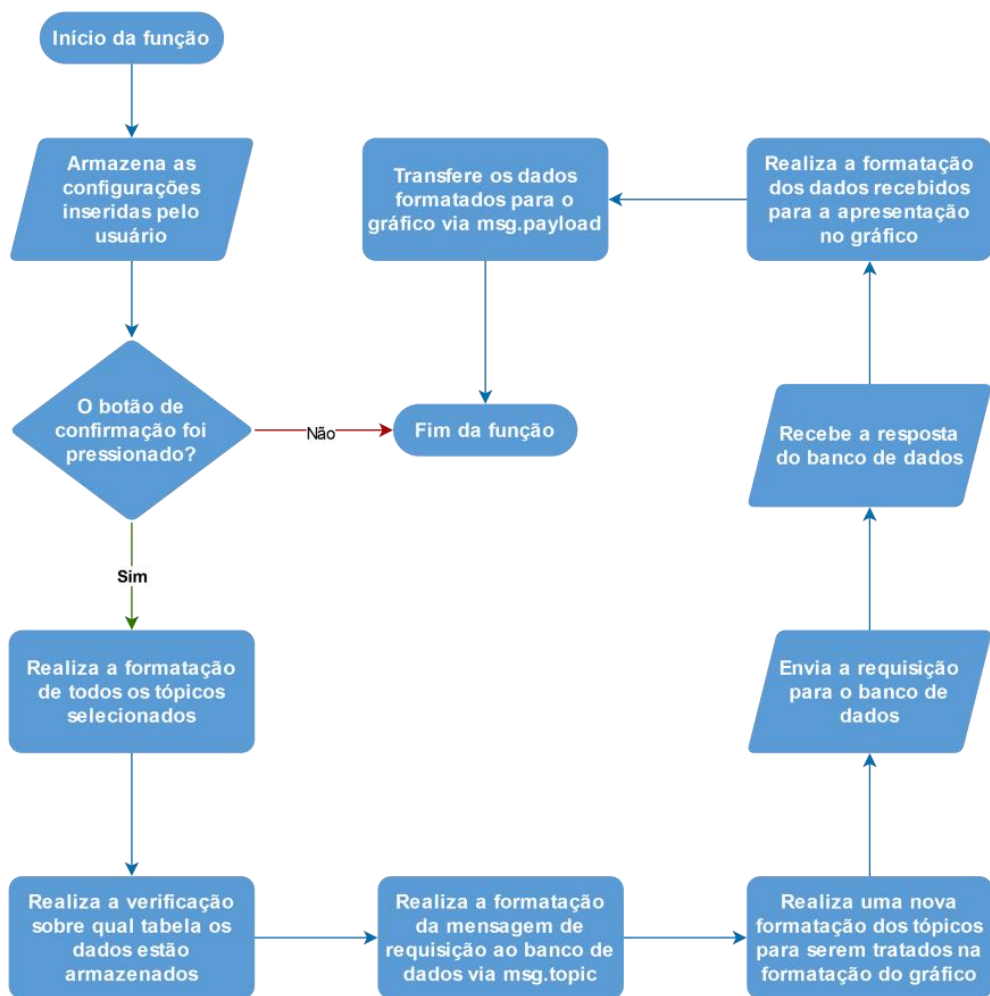


Figura 109 - Fluxograma do fluxo da interface de consulta histórica via Node-RED
Fonte: autoria própria

6 CAPÍTULO VI - INTERFACE DE MONITORAMENTO LOCAL

Para complementar a apresentação das informações da comunicação estabelecida entre o microcontrolador e os inversores da microrrede em uma interface local, utilizou-se de uma Interface Homem-Máquina (IHM) de baixo custo. O modelo adotado foi o EA-043A, fabricado pela Samkoon, sendo este um dispositivo industrial de alta performance com compatibilidade *touchscreen*, de forma que a aplicação dessa tecnologia permitisse que o usuário interaja de forma simples e direta com as informações da microrrede.

Dessa forma, a IHM foi configurada para atender às necessidades de monitoramento local dos inversores da microrrede, tendo em vista que ela possibilita a exibição de dados em tempo real relacionados a operação desses equipamentos. Esses dados, por sua vez, estiveram relacionados com o monitoramento de parâmetros de tensão, corrente e potência dos dispositivos. Na Figura 110, pode-se observar a IHM utilizada no desenvolvimento deste trabalho.



Figura 110 - IHM Samkoon EA-043

Fonte: [53]

Como referência para a configuração da IHM, utilizou-se do trabalho desenvolvido por [54], em que é realizado a configuração de uma interface para leitura e escrita de dados de

um microcontrolador modelo ESP32. Para isso, é utilizado de uma rede Modbus Serial, sobre o padrão RS232, que permite com que a IHM opere como cliente da rede, enquanto o microcontrolador é caracterizado como servidor. Assim, a IHM pode solicitar ao ESP32 as informações de interesse para serem apresentadas nas interfaces criadas, desde que esta solicitação esteja submetida as regras de comunicação do protocolo Modbus no padrão serial.

6.1 CONFIGURAÇÃO DA IHM

No caso deste trabalho, tendo como referência a metodologia apresentada em [54], foram desenvolvidas duas interfaces específicas para o monitoramento dos parâmetros da microrrede. Dentre esses parâmetros, pode-se destacar a apresentação dos dados de potência, tensão e corrente vinculados a entrada e a saída do inversor híbrido, a saída do inversor fotovoltaico, e a operação das baterias.

A primeira dessas interfaces foi desenvolvida com o intuito de fornecer uma visão geral instantânea da operação da microrrede, apresentando os dados em tempo real no que diz respeito a potência, tensão e corrente do equipamento. Os resultados dessa configuração podem ser visualizados na Figura 111.

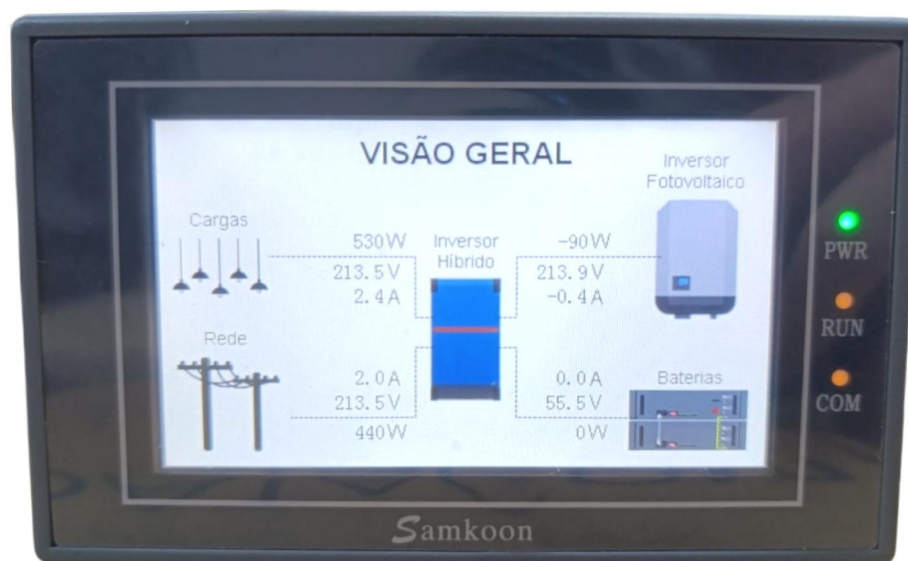


Figura 111 - Tela de supervisão geral da IHM
Fonte: autoria própria

Enquanto isso, a segunda interface foi criada com o objetivo de apresentar o comportamento das potências de cada um dos componentes do sistema através de um gráfico dinâmico, conforme demonstrado na Figura 112.



Figura 112 - Tela dos gráficos de potência da IHM
Fonte: autoria própria

Dessa forma, para que fosse possível iniciar o procedimento de montagem da interface de monitoramento local, fez-se necessário instalar, primeiramente, o software para edição dessa interface. Neste caso, a fabricante da IHM disponibiliza o arquivo de instalação no site oficial da empresa, disponível em [55], destacado como SATOOL. Na situação do desenvolvimento deste trabalho, a versão V6.2.0.30 foi utilizada na instalação do software.

6.1.1 Instalação do software de configuração da IHM

Para prosseguir com a instalação do software, após o download do arquivo disponibilizado em [55], executou-se a aplicação intitulada de “setup SATOOL.exe”, de forma que fosse possível iniciar os procedimentos de instalação do software para a edição da interface da IHM. Assim, a primeira etapa do procedimento de instalação consistiu em definir a linguagem da interface para inglês, conforme destacado na Figura 113.

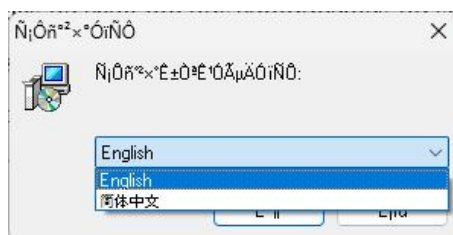


Figura 113 - Definição da linguagem do software SATOOL

Fonte: autoria própria

Após isso, prosseguiu-se normalmente com a instalação da aplicação, de forma que, ao fim do procedimento, uma nova interface fosse aberta, referente a instalação dos *drivers* a serem utilizados pela IHM. Ao prosseguir com a instalação desses *drivers*, que são essenciais para o funcionamento do componente, uma interface semelhante a presente na Figura 114 foi apresentada, confirmando a instalação adequada do software de edição da IHM. Caso o assistente para instalação de drivers não fosse executado, a fabricante disponibiliza em [55] os arquivos para que esse procedimento seja executado manualmente.

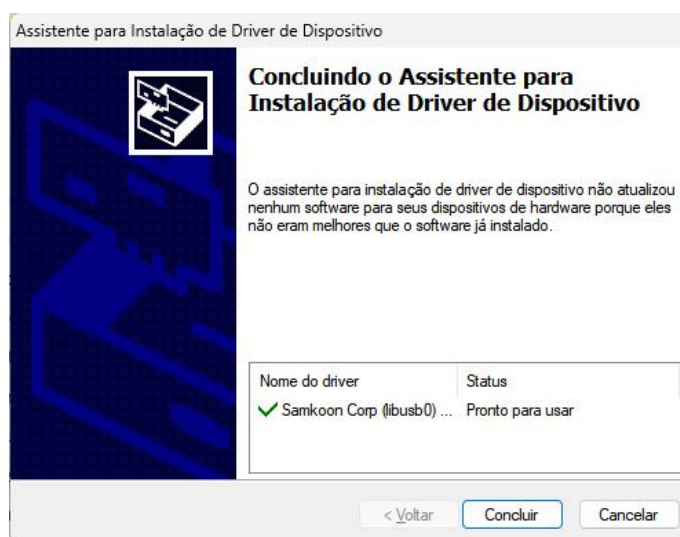


Figura 114 - Confirmação da instalação dos drivers da IHM

Fonte: autoria própria

6.1.2 Configurações básicas da IHM

Após a instalação e execução do software SATOOL, o ambiente para a edição das interfaces da IHM foi aberto conforme demonstrado na Figura 115.

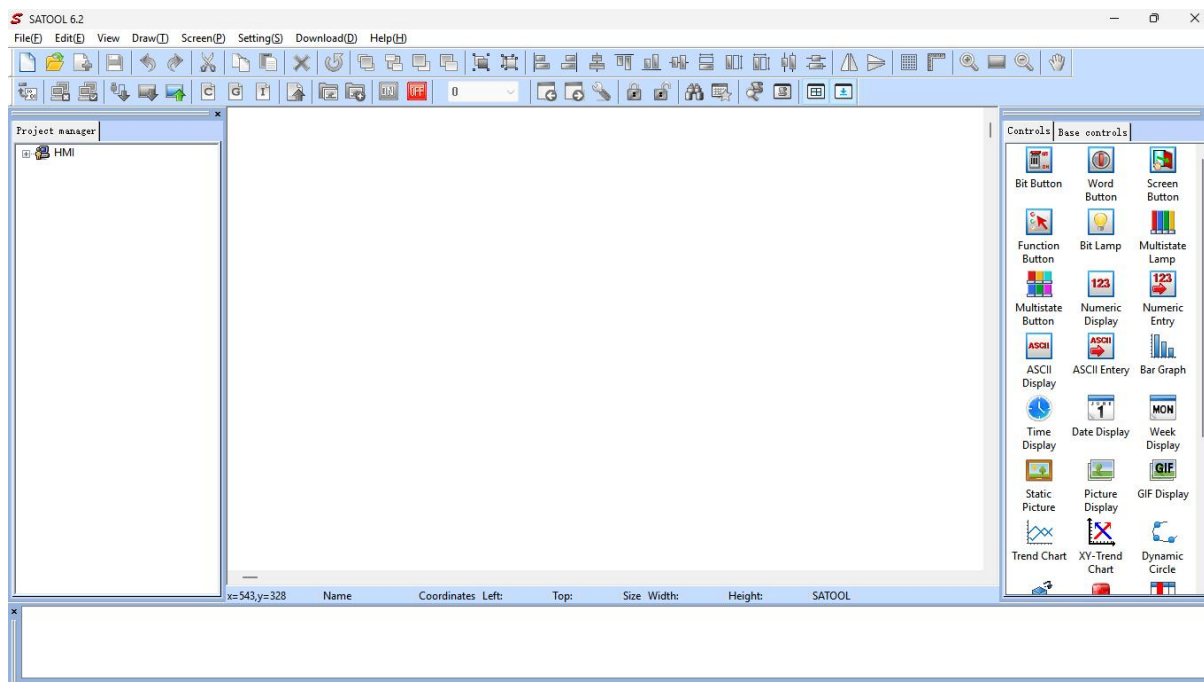
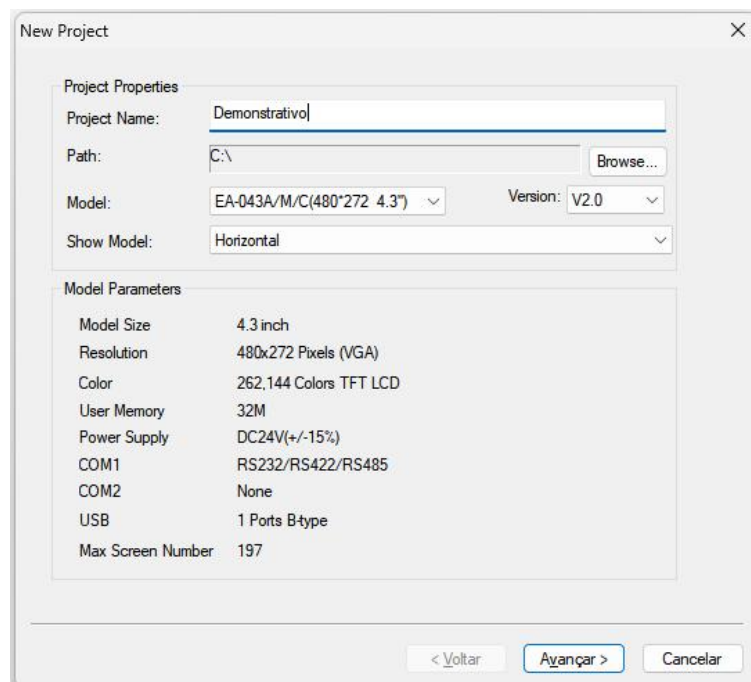


Figura 115 - Ambiente do software SATOOL
Fonte: autoria própria

Dessa forma, para prosseguir com a modelagem dessas interfaces, criou-se, primeiramente, um novo projeto através da seleção do campo “*File*”, localizado no canto superior esquerdo da Figura 115. Com isso, uma nova interface de configuração foi apresentada, de forma que os parâmetros a serem configurados se relacionassem com o caminho do diretório em que o projeto deve ser salvo, e o modelo da IHM utilizada como referência. Neste caso, na Figura 116 é destacada as configurações realizadas no desenvolvimento deste trabalho, tendo em vista o modelo apresentado na Figura 110.

Com a definição básica do arquivo de projeto realizada, avançou-se para a próxima interface de configuração, que neste caso estava relacionada com a conexão entre a IHM e o dispositivo fonte dos dados de interesse a serem apresentados no equipamento. No caso deste trabalho, o dispositivo fonte se trata do ESP32 configurado no capítulo 3, que possui os dados de operação dos inversores instalados na microrrede. Assim, para que fosse possível extrair esses dados, a IHM teve de ser configurada como mestre de uma rede Modbus RTU, conforme destacado na Figura 117.

Por fim, foi aberta uma interface para a configuração da primeira tela a ser utilizada na IHM, de forma que as únicas configurações possíveis de serem realizadas, estivessem relacionadas ao nome de identificação da tela, e a cor de fundo a ser utilizada pela interface. No caso deste trabalho, todas as telas foram configuradas tendo como base o fundo branco, conforme os parâmetros apresentados na Figura 118.



New Project

Project Properties

Project Name: Demonstrativo

Path: C:\ Browse...

Model: EA-043A/M/C(480*272 4.3") Version: V2.0

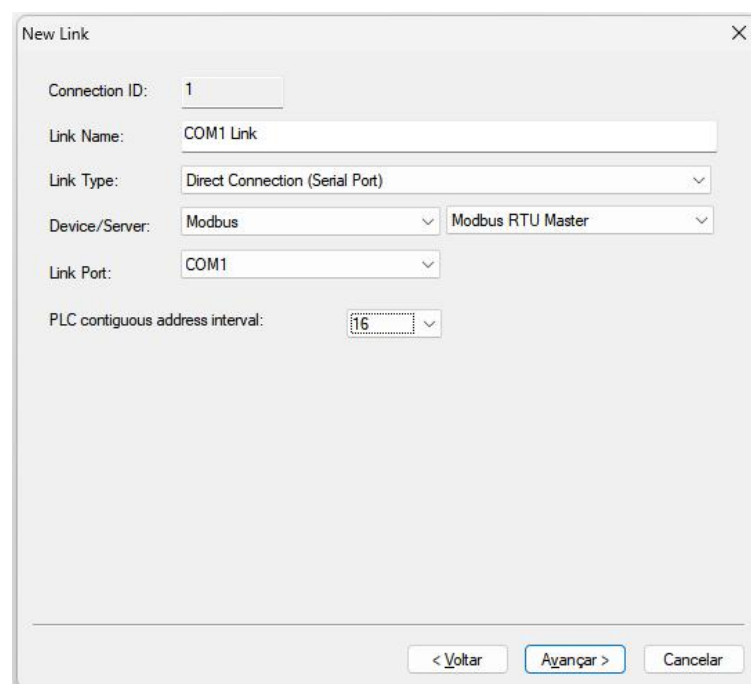
Show Model: Horizontal

Model Parameters

Model Size	4.3 inch
Resolution	480x272 Pixels (VGA)
Color	262,144 Colors TFT LCD
User Memory	32M
Power Supply	DC24V(+/-15%)
COM1	RS232/RS422/RS485
COM2	None
USB	1 Ports B-type
Max Screen Number	197

< Voltar Avançar > Cancelar

Figura 116 - Configuração do projeto básico para interface da IHM
Fonte: autoria própria



New Link

Connection ID: 1

Link Name: COM1 Link

Link Type: Direct Connection (Serial Port)

Device/Server: Modbus Modbus RTU Master

Link Port: COM1

PLC contiguous address interval: 16

< Voltar Avançar > Cancelar

Figura 117 - Configuração do protocolo de comunicação da IHM
Fonte: autoria própria

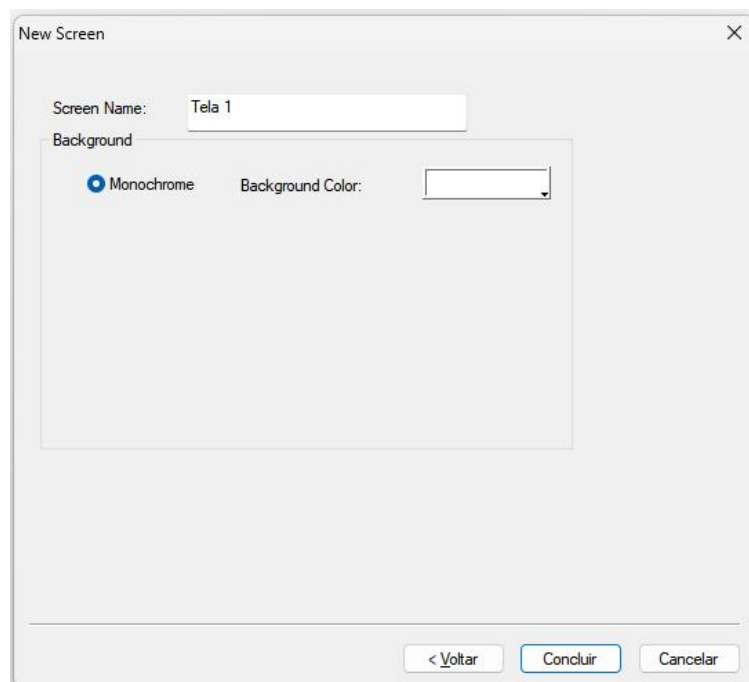


Figura 118 - Configuração da Tela 1 da IHM
Fonte: autoria própria

Com isso, o projeto criado foi aberto no ambiente apresentado na Figura 115, de forma que fosse possível iniciar o processo de modelagem da tela de supervisão da IHM. Ademais, antes disso, fez-se necessário realizar uma última configuração referente a comunicação da IHM com o ESP32, que pôde ser realizada ao selecionar o campo “*COM1 Link*” no canto esquerdo da Figura 119.

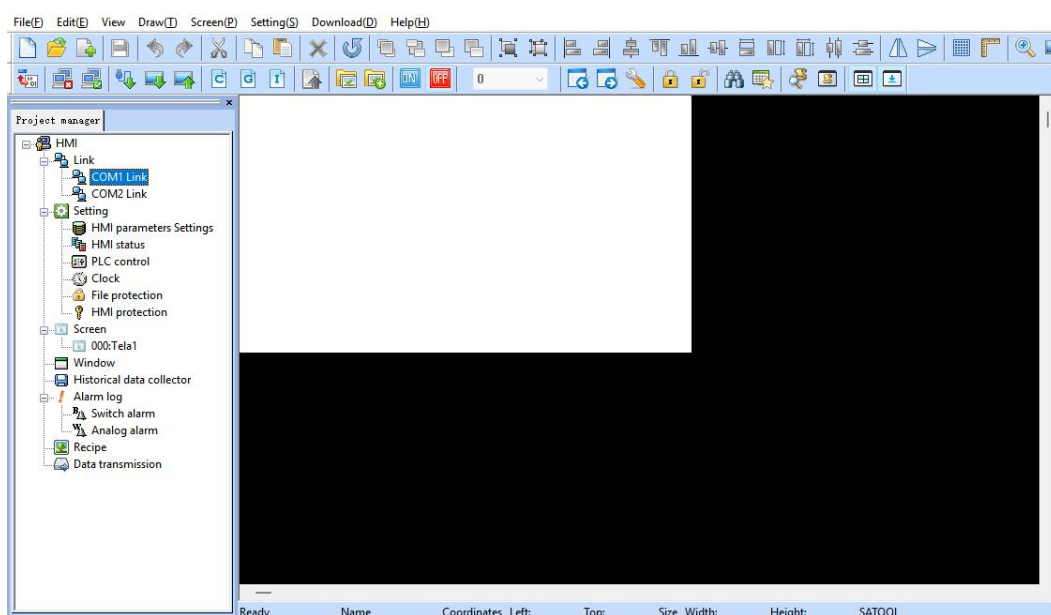


Figura 119 - Configuração da comunicação da IHM com o ESP32
Fonte: autoria própria

Ao selecionar a configuração destacada na Figura 119, a interface visível na Figura 120 foi apresentada, de forma que o campo “*Communication port*” pudesse ser alterado para “RS232”. Os demais parâmetros dessa interface não foram ser alterados, pois correspondem aquelas informações inseridas na Figura 117.

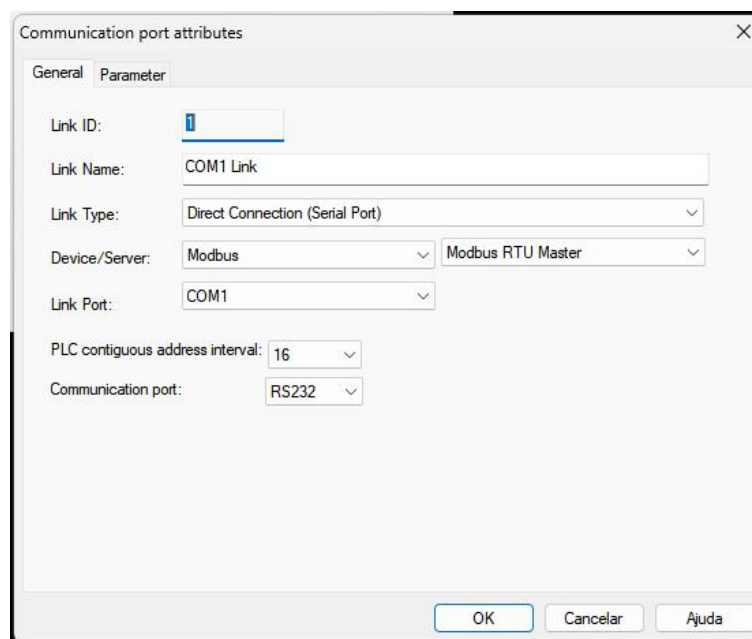


Figura 120 - Configuração do padrão RS232 na IHM
Fonte: autoria própria

Por fim, ao acessar a guia “*Parameter*”, presente na Figura 120, foi possível visualizar os parâmetros de comunicação da IHM, conforme destacado na Figura 121. Nessa figura, nota-se a definição padrão do “*Baud rate*” como 9600 e o “*PLC Address*” como 1, que representam, respectivamente, a taxa de transferência de dados comunicação serial, e o endereço do servidor ao qual a requisição deve ser realizada na transação Modbus. Com base nisso, estabeleceu-se essas configurações como padrão para a comunicação Modbus serial a ser implementada com o microcontrolador, uma vez que o ele também necessita de ser configurado para operar na mesma taxa de transferência (9600 baud), e com o endereço igual a 1.

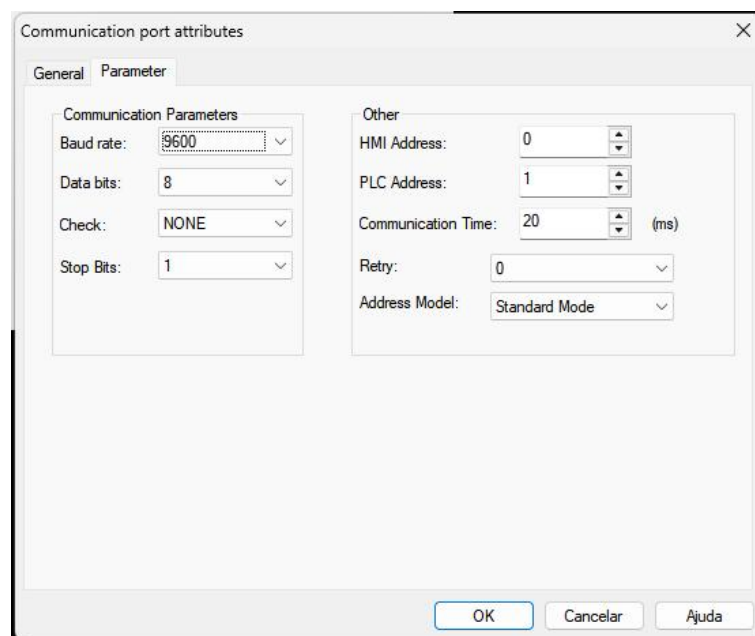


Figura 121 - Configurações adicionais da comunicação Modbus da IHM
Fonte: autoria própria

6.1.3 Configuração da tela de supervisão geral da IHM

Com as configurações básicas da IHM realizadas no software SATOOL, a próxima etapa para a modelagem da interface a ser utilizada no dispositivo, consistiu na inserção e configuração dos elementos gráficos do software. No caso da primeira tela desenvolvida neste trabalho, utilizou-se de quatro recursos principais para a apresentação dos dados, sendo eles o “*Numeric Display*”, “*Static Picture*”, “*Text*” e “*Fold Line*”. O primeiro destes componentes se trata daquele que foi utilizado para apresentar, de maneira efetiva, os dados da comunicação com o microcontrolador, enquanto que os demais se trataram apenas de complementações gráficas para a interface, de forma que fosse possível criar uma melhor contextualização para a interação com o usuário.

Dessa forma, quanto a inserção e configuração do “*Numeric Display*”, esta ocorreu através do campo “*Control*”, situado no canto direito do ambiente de configuração da interface da IHM, conforme ilustrado na Figura 122. Ao selecionar este componente, uma interface de configuração foi apresentada, conforme destacado na Figura 122, de forma que na guia “*General*” fosse possível de realizar todas a configurações necessárias para a utilização do componente.

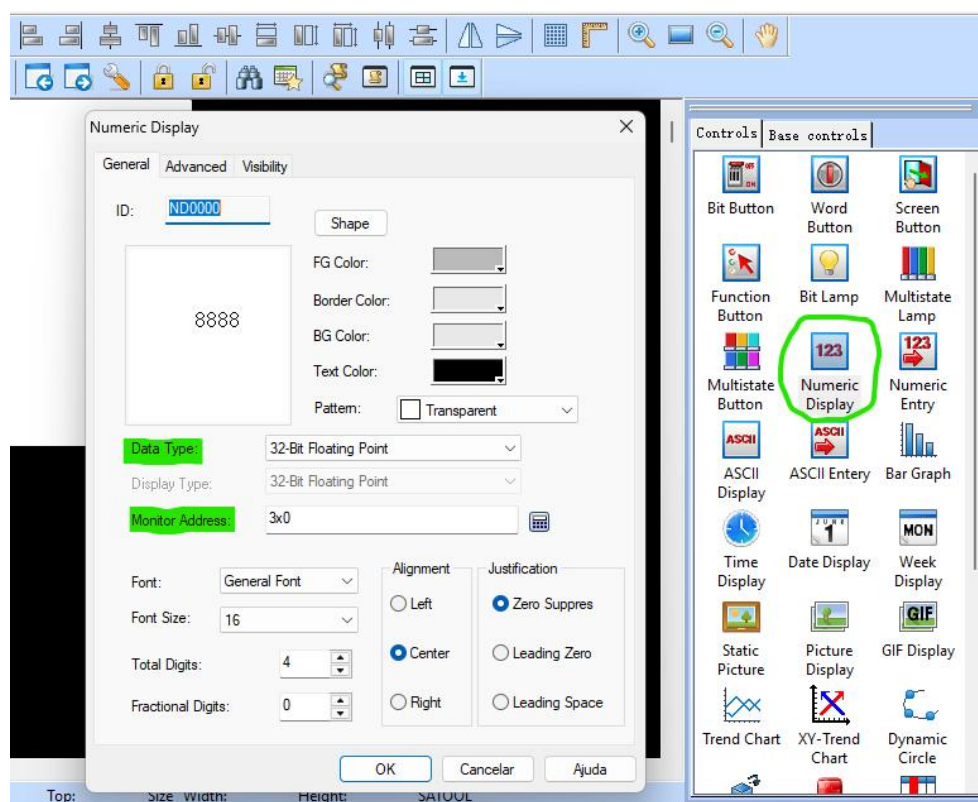


Figura 122 - Configuração do *Numeric Display* da IHM
Fonte: autoria própria

Assim, com base na interface de configuração apresentada na Figura 122, os campos destacados como “*Data Type*” e “*Monitor Address*” representam as configurações que envolvem a leitura e apresentação dos dados de interesse. Em relação aos demais parâmetros que podem ser configurados nesta interface, eles estão vinculados a estilização do *Numeric Display*, de forma que a sua configuração não impacte de maneira direta na apresentação da informação.

Com base nisso, tendo em vista a situação registrada na Figura 122, o campo “*Data Type*” foi definido para a leitura de 32 bits de dados, que correspondem a um número do tipo ponto flutuante (float). Esse conjunto de bits deve estar armazenado em um endereço de registrador conhecido, que no caso deste trabalho, foi configurado no microcontrolador na seção 6.2. De maneira complementar, cada um desses registradores é capaz de armazenar apenas um conjunto máximo de 16 bits em seu respectivo endereço, sendo este um padrão da comunicação Modbus. Assim, fez-se necessário utilizar de dois registradores consecutivos para armazenar uma determinada informação de 32 bits, de forma que, ao definir o endereço do primeiro registrador, o software SATOOL fosse capaz de localizar o próximo endereço de

maneira automática, e assim realizar toda a conversão dos dados para um conjunto de 32 bits como ponto flutuante.

Dessa forma, para que a informação fosse corretamente apresentada na IHM, o campo “*Monitor Address*” teve de ser configurado obedecendo a formatação “3xENDEREÇO”, em que o caractere 3 representa a função de leitura de um registrador de entrada analógica. De maneira complementar, o campo “ENDEREÇO” foi substituído pelo endereço de início onde a informação de interesse se encontrava armazenada no microcontrolador. No caso deste trabalho, na Tabela 3 é destacado o endereço de todos os registradores utilizados na modelagem desta interface, de forma que eles foram configurados obedecendo a formatação em cadeia de dois registradores de 16 bits, para armazenar uma única informação.

Tabela 3 - Endereçamento utilizado no *Monitor Address*

Monitor Address	Descrição
3x0	Potência do inversor fotovoltaico
3x2	Potência da rede
3x4	Potência das baterias
3x6	Potência das cargas
3x10	Tensão do inversor fotovoltaico
3x12	Tensão da rede
3x14	Tensão das baterias
3x16	Tensão das cargas
3x20	Corrente do inversor fotovoltaico
3x22	Corrente da rede
3x24	Corrente das baterias
3x26	Corrente das cargas

Fonte: autoria própria

Com as configurações do *Numeric Display* realizadas, o ambiente de edição foi liberado para a inserção do componente em qualquer localização da tela da IHM, conforme apresentado na Figura 123. Com isso, tornou-se possível editar a posição e tamanho deste componente, desde que ele estivesse enquadrado no campo reservado a ocupação real da tela de apresentação. Se necessário, a interface presente na Figura 122 poderia ser acessada novamente, bastando apenas clicar sobre o *Numeric Display* posicionado.

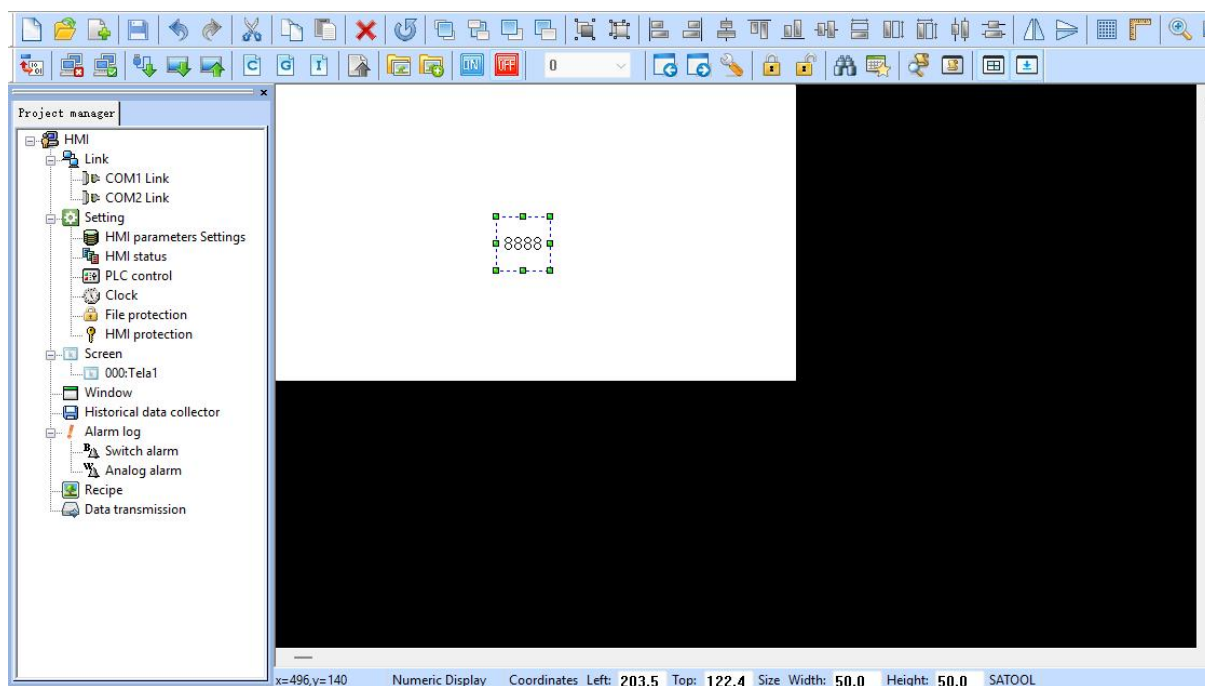


Figura 123 - Posicionamento do *Numeric Display*
Fonte: autoria própria

Portanto, tendo como base os endereços apresentados na Tabela 3, e a configuração do componente na Figura 122, foi desenvolvido a interface de supervisão geral apresentada na Figura 124. Nesta figura, é possível notar que os campos de *Numeric Display* foram destacados com o caractere “8”, como na Figura 123, enquanto que os demais componentes que envolvem texto foram formatados ao inserir o componente “*Text*”, situado na guia “*Base Controls*” no canto direito da Figura 122. De maneira complementar, as linhas pontilhadas foram inseridas utilizando o componente “*Fold line*”, localizado no mesmo grupo que o componente “*Text*”. Por fim, as representações gráficas das cargas, rede e inversores foram implementadas através do componente “*Static Picture*”, situada no mesmo grupo de funções que o *Numeric Display*. Assim, a configuração básica para a utilização desse último recurso pode ser encontrada na Figura 125, de forma que o campo “*Picture path*” fosse configurado com o local em que o arquivo da imagem se encontra armazenado.

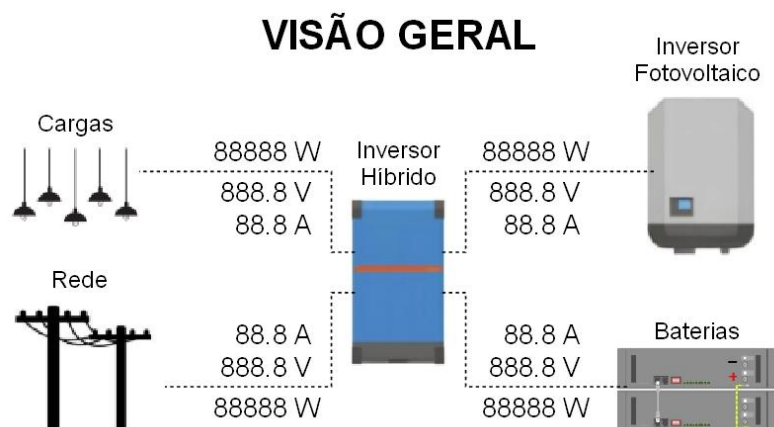


Figura 124 - Configuração da tela de visão geral da IHM
Fonte: autoria própria

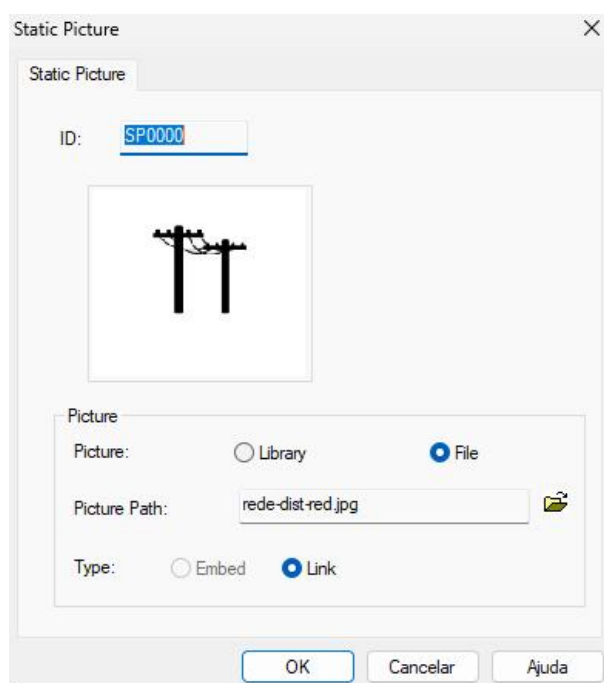


Figura 125 - Configuração básica do *Static Picture*
Fonte: autoria própria

6.1.4 Configuração do gráfico dinâmico

Em relação a interface de supervisão local da microrrede, a última tela desenvolvida para apresentação dos dados de operação dos equipamentos consistiu na apresentação dos dados de potência de cada um dos recursos apresentados na Figura 124, por meio de um

gráfico dinâmico. Para isso, utilizou-se do componente “*Trend chart*”, localizado no grupo “*Control*” do ambiente de configuração da IHM.

Ao selecionar este recurso gráfico, uma interface de configuração semelhante a visível na Figura 126 foi apresentada, com destaque para quatro guias possíveis de serem parametrizadas. A primeira dessas guias, destacada na Figura 126, refere-se aos aspectos gerais de configuração para a apresentação dos dados no gráfico, de forma que os campos “*Data type*” e “*Read Address*” fossem configurados, respectivamente, com o tipo de dado a ser mostrado no gráfico, e o endereço de início do registrador que armazena essa informação. No caso de se fazer necessário apresentar mais de um conjunto de dados em um mesmo gráfico, o campo “*Number of Data*” deve ser preenchido com a quantidade de linhas totais do gráfico. Destaca-se, em tempo, que o software realiza a vinculação dessa informação de acordo com o endereço configurado no campo “*Read Address*”.

Figura 126 - Configuração geral do Trend chart
Fonte: autoria própria

Assim, os dados que foram selecionados para serem apresentados na configuração realizada na Figura 126 se tratavam das potências definidas anteriormente na Tabela 3, tendo em vista que elas iriam ocupar um total de 4 conjuntos de registradores do tipo ponto

flutuante de 32 bits. Neste sentido, para que mais de um conjunto de dados fossem apresentados no *Trend Chart*, fez-se necessário garantir que os dados de interesse estivessem localizados em registradores subsequentes, obedecendo a formatação utilizada na Tabela 3.

Ainda em relação as configurações presentes na Figura 126, o campo “*Read trigger*” e “*Clear trigger*” foram os responsáveis por atribuir um endereço de memória como gatilho para iniciar o registro de informações no gráfico, no caso do primeiro campo, ou limpar todos os dados apresentados neste recurso, no caso do segundo campo. Dessa forma, tendo em vista a configuração na Figura 126, somente foi possível apresentar qualquer informação no gráfico caso o bit vinculado ao endereço “LB1” estivesse com valor lógico alto (1). Caso contrário, nenhuma nova informação seria atualizada no ambiente, de forma que este recurso não possuísse nenhuma utilidade para a interface de supervisão.

Com base nisso, desenvolveu-se um botão virtual e invisível para ser utilizado na interface da IHM como gatilho para o funcionamento do gráfico de potências. Para isso, foi inserido um “*Bit Button*” na tela antecessora àquela em que o gráfico se encontrava, com as configurações realizadas conforme as informações apresentadas na Figura 127. Destaca-se, nesta figura, que os aspectos visuais do botão, como cor de fundo e cor da letra, foram definidos para transparentes, e a função do componente (*Funtion*) para “*Set ON*”. Além disso, o mesmo endereço do campo “*Read Trigger*” na Figura 126, foi inserido no campo “*Write Address*” da Figura 127, de forma que, ao pressionar este botão, o gráfico de potências inicie a atualização constante dos dados em sua respectiva interface.

Ainda em relação a configuração do botão de gatilho na Figura 127, dessa vez levando em consideração a posição do componente no ambiente da IHM, nota-se que ele foi alocado à direita da tela da interface, de forma a ocupar metade do espaço disponível para a implementação de recursos da IHM. De maneira complementar, para que a IHM tivesse a capacidade de operar adequadamente, fez-se necessário enviar este botão para a camada mais inferior do ambiente de configuração do dispositivo de apresentação de dados. Para isso, bastou selecionar o botão, e posteriormente clicar sobre o ícone destacado no canto superior esquerdo da Figura 127. Um procedimento alternativo pode ser visualizado na Figura 128 em que foi possível enviar o item selecionado à camada mais inferior da tela, após clicar com o botão direito sobre o botão, e navegar pelo caminho *Layer > Send to Bottom*.

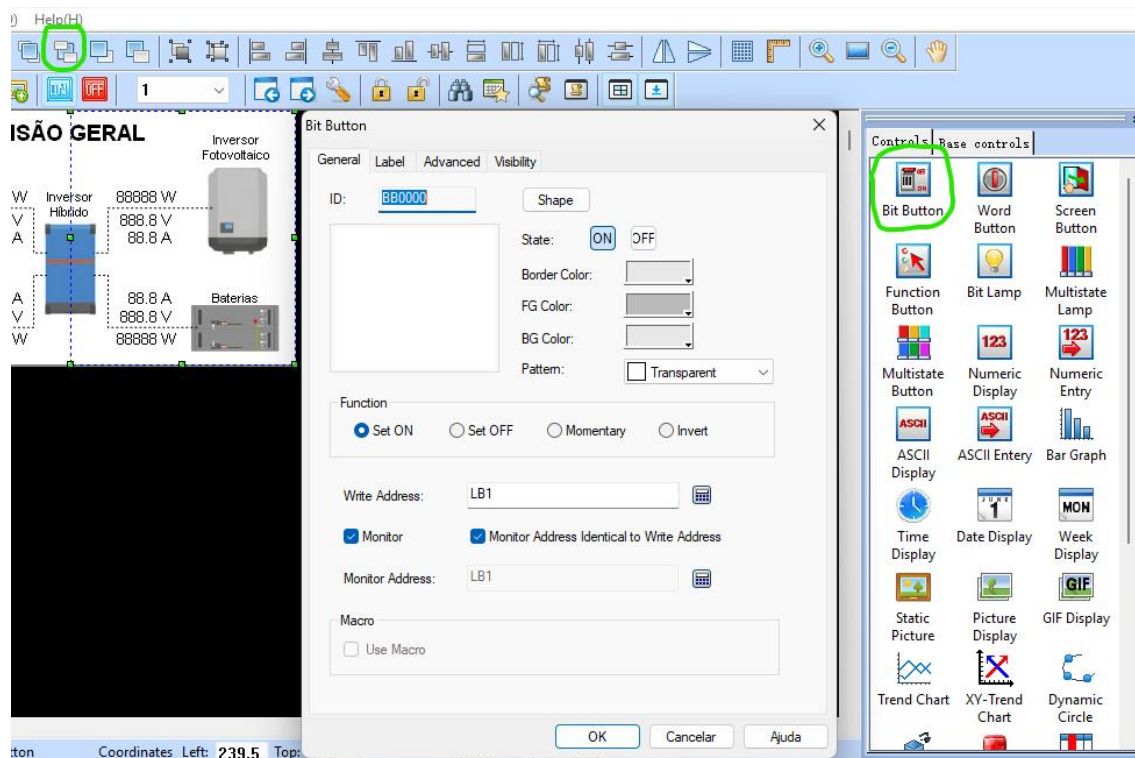


Figura 127 - Botão de gatilho para funcionamento do gráfico
Fonte: autoria própria

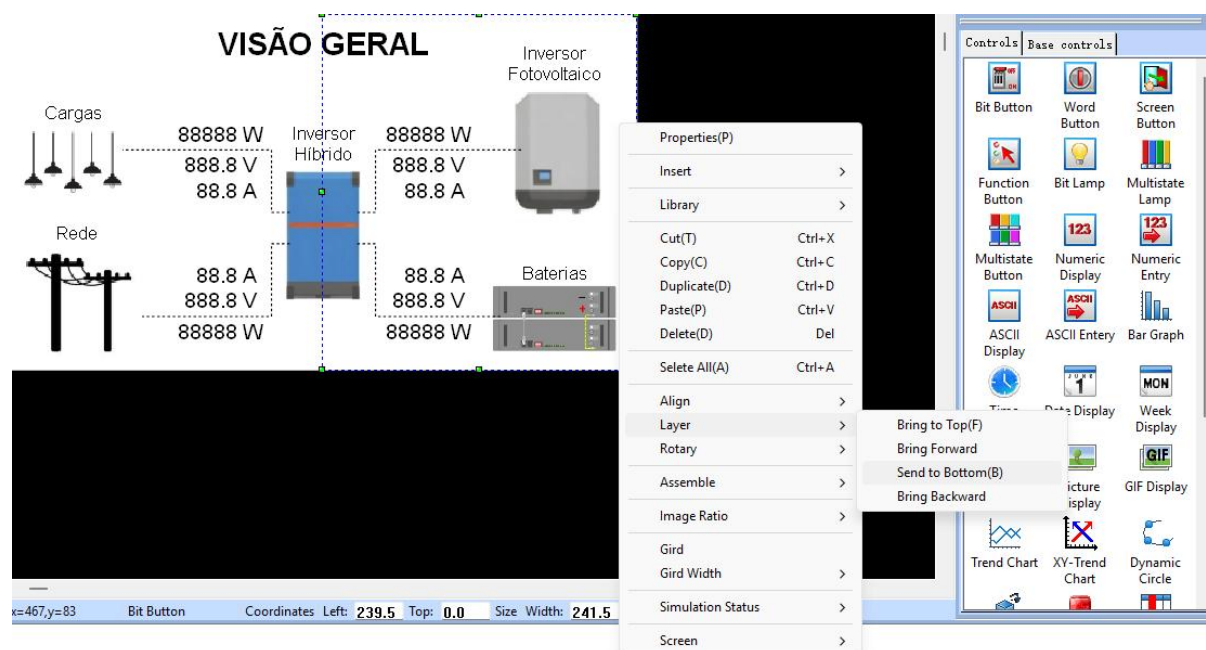


Figura 128 - Alocação do botão de gatilho para a camada inferior
Fonte: autoria própria

Com isso, tem-se configurados os aspectos funcionais para a implementação do *Trend Chart* neste trabalho, de forma que as demais configurações realizadas nas guias “Pen” e “XY Axis” possam ser visualizadas na Figura 129 e na Figura 130. Estas configurações, por sua vez,

são vinculadas, respectivamente, a estética de cada linha a ser apresentada, e ao padrão adotado para a divisão das grades no gráfico.

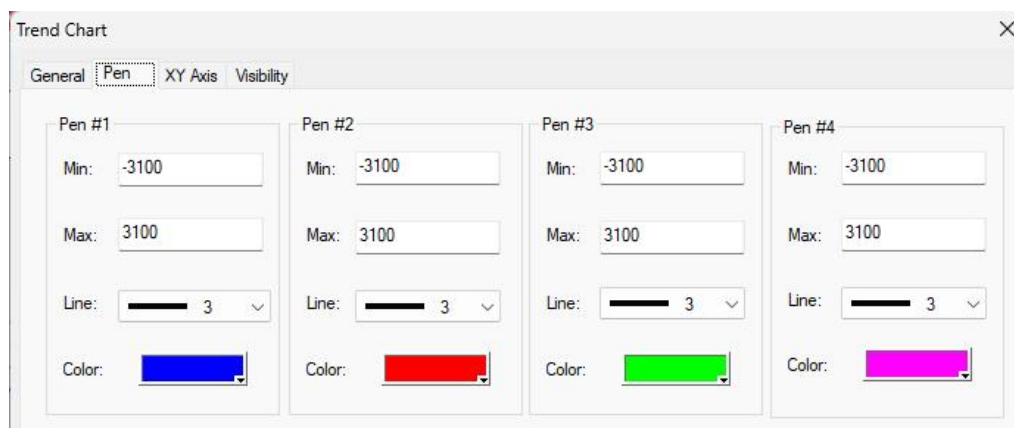


Figura 129 - Configuração das linhas do gráfico

Fonte: autoria própria

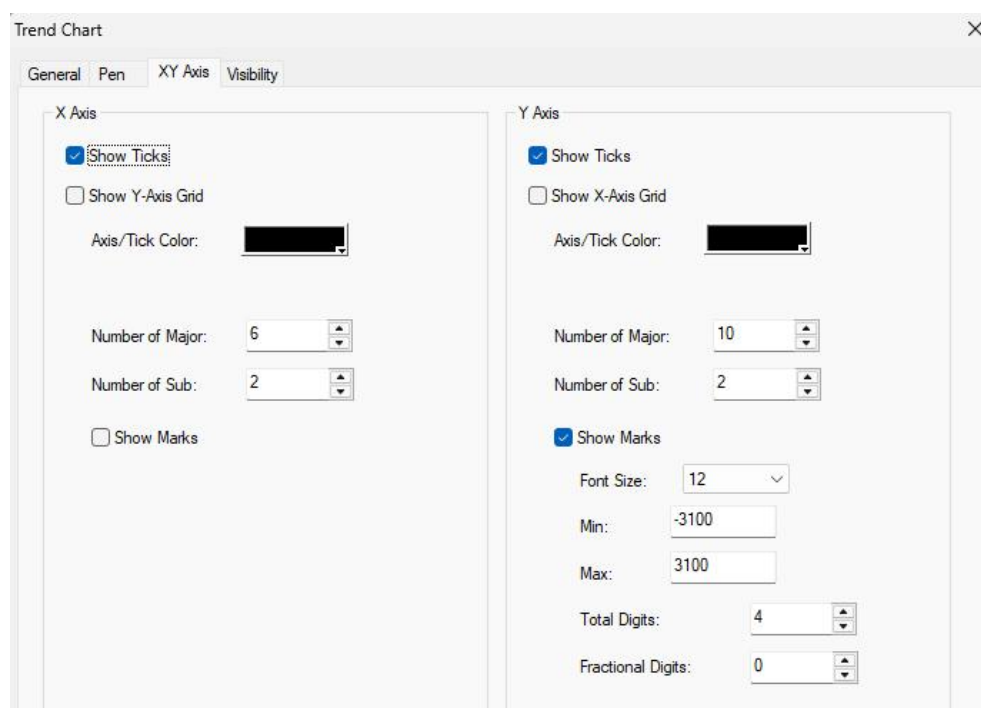


Figura 130 - Configuração das grades do gráfico

Fonte: autoria própria

Dessa forma, sempre que o usuário clicasse sobre o lado direito da interface da tela 1 da IHM, com o objetivo de transitar para a próxima tela, o gatilho para início dos registros dos dados no gráfico seria acionado. Ao mesmo tempo, um outro botão para a troca de tela teve de ser inserido no mesmo espaço que o botão de gatilho do gráfico, desde que aquele esteja alocado em uma camada imediatamente anterior a este, em conformidade com a metodologia

apresentada na seção 6.1.5. Dessa forma, tanto a ação de inicialização do gráfico, quanto a ação da troca de tela, seriam executadas ao mesmo tempo.

Portanto, na Figura 131 é destacada a visão final da tela para visualização dos gráficos de potência da IHM. Nesta figura, nota-se a presença de um título e de uma legenda que foram inseridos manualmente, através da utilização dos recursos “Text” e “Line” do ambiente de configuração da IHM.

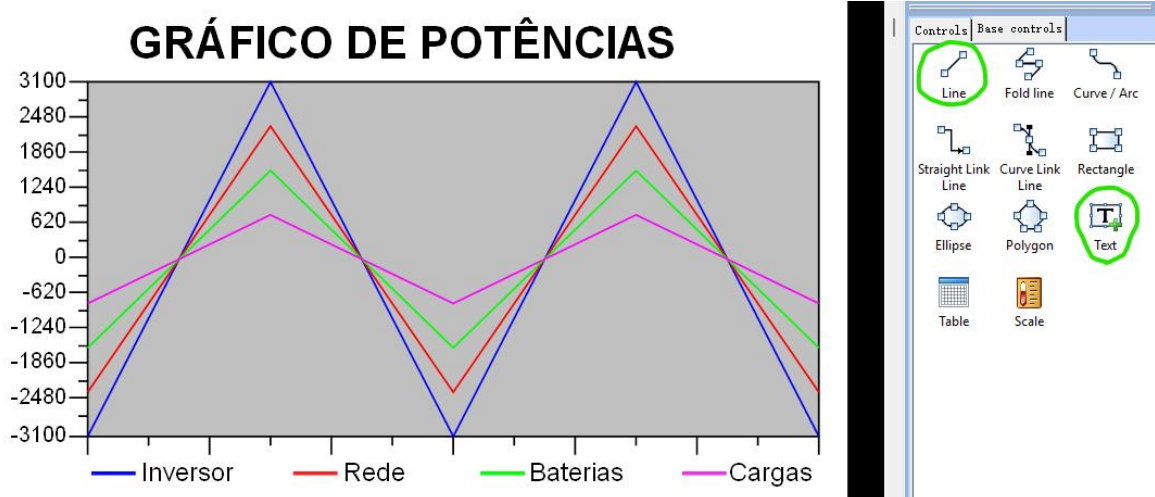


Figura 131 - Vista final da Tela de gráficos da IHM
Fonte: autoria própria

6.1.5 Configuração da navegação de telas

Levando em consideração as funcionalidades básicas da IHM, a última etapa de configuração das interfaces consistiu em inserir os botões de navegação entre as telas desenvolvidas nas seções anteriores. Dessa forma, teve-se o objetivo de permitir que o usuário conseguisse navegar entre as telas disponíveis na IHM de maneira intuitiva, de forma a garantir o acesso a todas as funcionalidades do sistema de supervisão local.

Para isso, a mesma metodologia para a inserção do *Bit Button* utilizado na Figura 127 foi usada para a alocação dos botões responsáveis pela troca de telas na IHM. Neste sentido, o componente empregado no ambiente de configuração do SATOOL, se tratou do “*Screen Button*”, que foi configurado conforme apresentado na Figura 132.

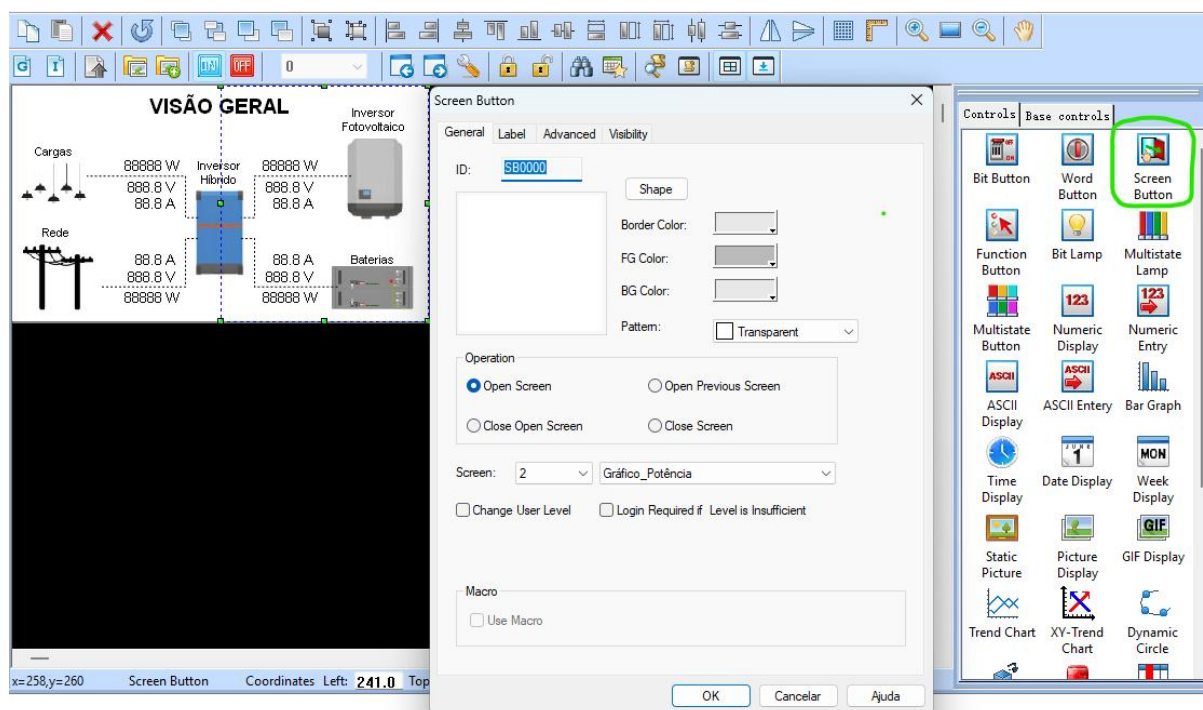


Figura 132 - Configuração do botão de avanço de tela
Fonte: autoria própria

Nas configurações presentes na Figura 132, nota-se que o padrão do botão foi definido como transparente no campo “*Pattern*”, de forma que os aspectos visuais do botão não tivessem influência sobre a tela da IHM. Além disso, o campo “*Operation*” foi definido como “*Open Screen*”, de forma que este componente estivesse configurado para abrir a tela referenciada em “*Screen*”, localizada logo abaixo do campo “*Operation*” na Figura 132. Neste caso, a tela configurada para a abertura se tratou daquela nomeada como “*Gráfico_Potência*”, que por sua vez foi implementada na Figura 131.

Por fim, o tamanho do *Screen Button* foi configurado para ocupar a metade da tela disponibilizada pela IHM, de forma que ele fosse alocado no lado direito da interface, conforme destacado na Figura 132. Dessa forma, padronizou-se o avanço de tela com a inserção desses botões na metade da tela à direita da IHM, enquanto que a metade à esquerda passou a ser referido a opção de retornar para a tela anterior. No caso dessa última opção, ela foi parametrizada na segunda tela criada na IHM, que no contexto deste trabalho, se referia a interface de apresentação dos gráficos, conforme destacado na Figura 133. Nota-se, nesta figura, que a tela para retorno foi configurada como sendo a tela de “*Visão_Geral*”, de forma a fornecer uma maneira de transitar entre as duas telas disponíveis na IHM.

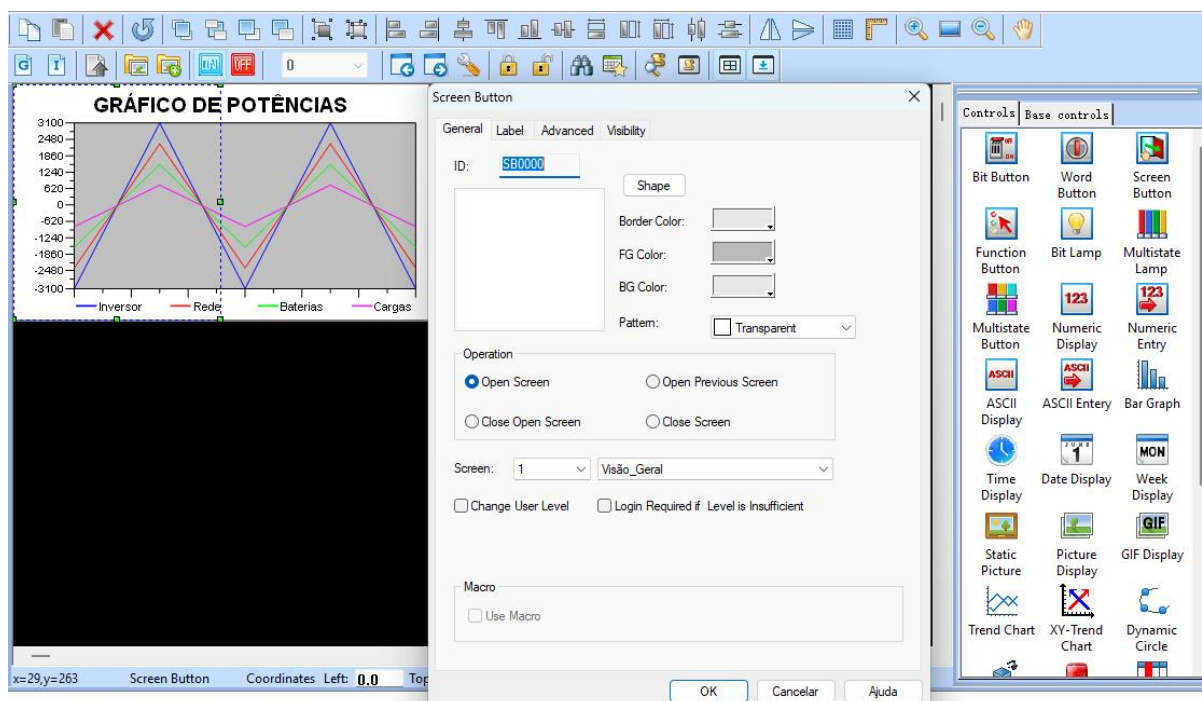


Figura 133 - Configuração do botão de retorno de tela
Fonte: autoria própria

Assim, a última etapa para a configuração dos botões de transição de telas da IHM consistiu em alocar estes botões na camada mais inferior da interface, seguindo a metodologia implementada no procedimento ilustrado na Figura 128. Destaca-se, em tempo, que estes *Screen Button* devem ser garantidamente alocados na última camada da tela, de forma a assegurar o correto funcionamento de todos os componentes da interface.

Com base nisso, o download dessas configurações na IHM foi realizado ao selecionar a opção “Compile+Download”, no canto superior esquerdo do ambiente do software de parametrização da interface, conforme apresentado na Figura 134. Após a abertura desta interface, bastou clicar sobre o botão “Download” situado no canto inferior esquerdo da tela, de forma que o procedimento fosse realizado automaticamente. Destaca-se, em tempo, a necessidade de garantir a correta conexão da porta USB do dispositivo em que o software SATOOL se encontra instalado, com a porta USB tipo B da IHM, visível na face inferior do dispositivo, conforme ilustrado na Figura 110.

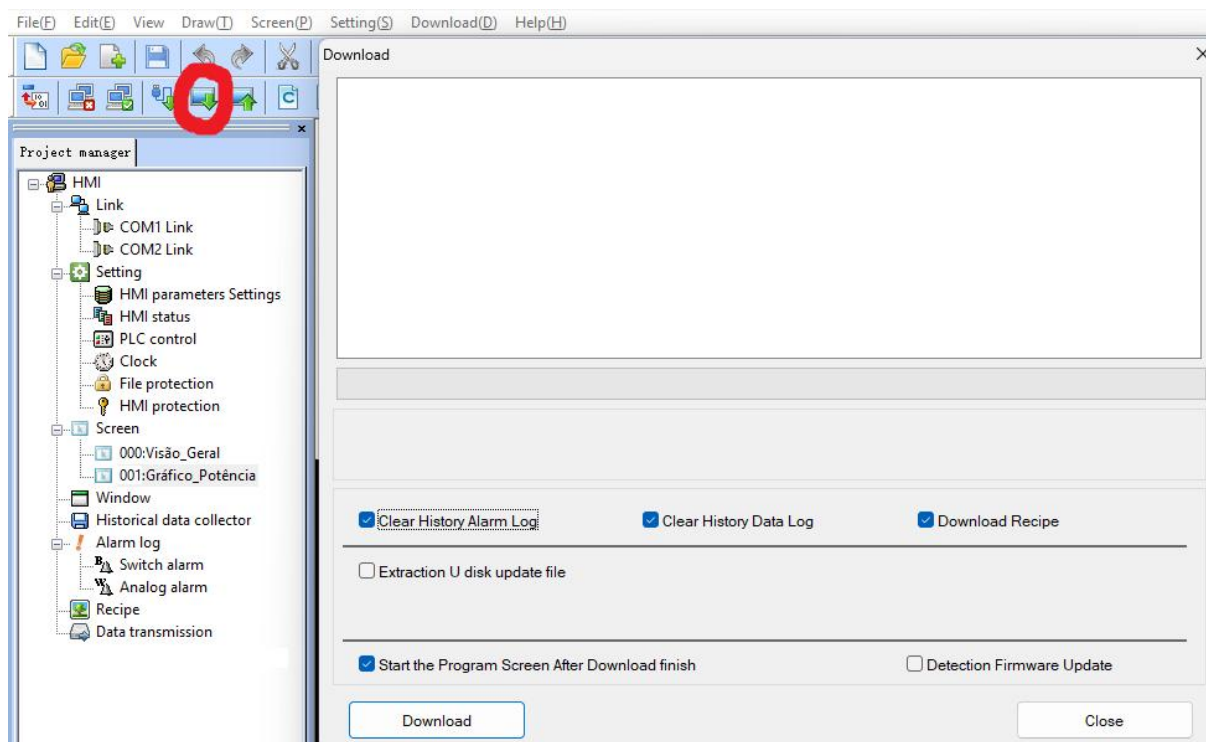


Figura 134 - Download do projeto para a IHM
Fonte: autoria própria

6.2 PROGRAMAÇÃO DA PLATAFORMA MICROCONTROLADA

Com a configuração da IHM realizada de acordo com a seção 6.1, fez-se necessário garantir que o microcontrolador estivesse corretamente configurado como o dispositivo servidor da rede Modbus Serial estabelecida entre o ESP32 e a IHM. Assim, seguindo a configuração realizada no ambiente do SATOOL, esperou-se garantir que o microcontrolador tivesse as informações apresentadas na Tabela 3 corretamente implementadas no código do programa, de forma a permitir que a IHM conseguisse realizar a leitura dos dados de interesse para cada interface modelada.

Para a viabilização desta etapa do trabalho, foi utilizado da biblioteca “ModbusRTU.h”, que pode ser encontrada em [39], e que permitiu configurar o ESP32 para operar como servidor em uma rede Modbus sobre o padrão RS232.

De maneira geral, a programação da plataforma microcontrolada consistiu em definir e inicializar os pinos do ESP a serem utilizados exclusivamente para a comunicação com a IHM. No caso deste trabalho, esses pinos se referem as GPIOs (do inglês *General Purpose Input*

Output) 16 e 17 do microcontrolador, que na prática, correspondem ao segundo conjunto de portas UART do dispositivo. Além disso, essas portas foram configuradas para operarem no mesmo baud de comunicação que aquele definido na Figura 121 (9600), de forma a garantir a correta transmissão de informação entre o ESP32 e a IHM.

Com essas configurações realizadas, realizou-se, também, a implementação da operação do microcontrolador como servidor da rede Modbus RTU, tendo em vista que nesta situação, o cliente configurado para a comunicação se trata da IHM. Assim, definiu-se, logo em seguida, o endereço do servidor a ser utilizado pelo ESP32, e os endereços dos registradores que devem ser utilizados para armazenar as informações na plataforma microcontrolada. Para este último procedimento, fez-se necessário obedecer o endereçamento utilizado na configuração do *Monitor Address* do ambiente de programação da IHM, que neste caso foi estabelecido em conformidade com os dados na Tabela 3.

Por fim, também foi implementado a atualização dos dados armazenados nesses registradores que foram inseridos no microcontrolador, tendo em vista que, para isso, fez-se necessário extrair as informações obtidas na comunicação do ESP32 com os inversores, via Modbus TCP/IP. No caso deste trabalho, essas informações foram armazenadas em seus respectivos componentes, que estão vinculados a estrutura apresentada na seção 3.2.3.1. Assim, enquanto o microcontrolador mantivesse a comunicação com inversores na rede Modbus TCP/IP, os dados obtidos em relação a tensão, corrente e potência de operação desses equipamentos também seriam constantemente atualizados nos registradores destacados na Tabela 3, de forma que a IHM conseguisse ter acesso a essas informações, via rede Modbus RS232.

Com base nessas funcionalidades, o fluxograma na Figura 135 destaca o algoritmo utilizado na implementação da comunicação Modbus RTU entre o microcontrolador e a IHM. Neste fluxograma, destaca-se a utilização da função “mbRTU.task”, relacionada a biblioteca ModbusRTU.h, que serviu de base para processar os dados de interesse na comunicação entre os dispositivos. Reitera-se que, através de experimentos empíricos, percebeu-se a necessidade de implementar a execução desta função em duas situações, sendo elas no início do *loop* que envolve o algoritmo do microcontrolador, e em toda situação que se faz necessário atualizar os registradores implementados no ESP32. Assim, além de ser chamada dentro da função “void loop” do microcontrolador, ela também foi inserida dentro da função “att_Reg_ESP”, de maneira a permitir que a comunicação entre a IHM e o microcontrolador fosse mantida de maneira mais estável. Esta funcionalidade, por sua vez, pode ser melhor observada no código completo do microcontrolador, disponibilizado no APÊNDICE A.

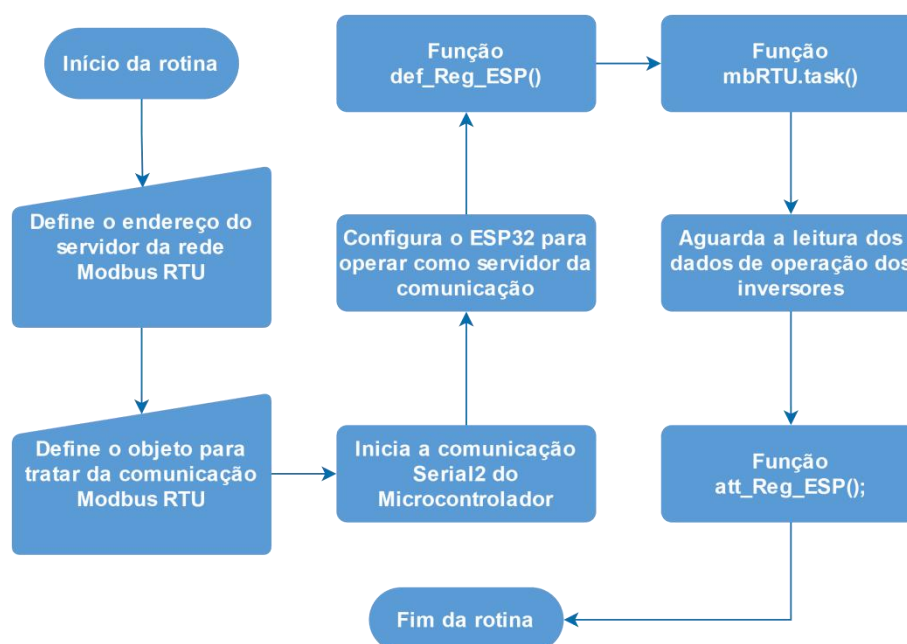


Figura 135 - Fluxograma da rotina de comunicação Modbus RTU do ESP32
Fonte: autoria própria

Em relação as funções “def_Reg_ESP” e “att_Reg_ESP”, essas foram implementadas para definir e atualizar, respectivamente, os registradores destacados na Tabela 3. No caso da primeira, o algoritmo utilizado no código desta função consistiu, basicamente, em utilizar da função “addlreg” da biblioteca “ModbusRTU.h” para adicionar os registradores com seus respectivos endereços no microcontrolador, de forma que o primeiro valor atribuído a cada um desses registradores fossem zero.

Destaca-se, em tempo, que a informação a ser armazenada em cada um dos endereços desses registradores, possui uma capacidade máxima de 16 bits, de forma que, para armazenar uma informação com resolução superior a esta, fosse necessário utilizar um conjunto de 2 ou mais registradores adjacentes. No caso deste trabalho, para a utilização de um dado do tipo ponto flutuante, houve a necessidade de utilizar de um conjunto de 2 registradores para cada informação armazenada, fazendo com que existisse a necessidade de definir esses dois registradores de maneira separada na função “def_Reg_ESP”. Apesar disso, para a correta interpretação da informação armazenada nos registradores do ESP32, deve-se realizar a leitura desses dois endereços de maneira conjunta, conforme ilustrado na Figura 136.

Códificação Binária de um número de 32 bits:

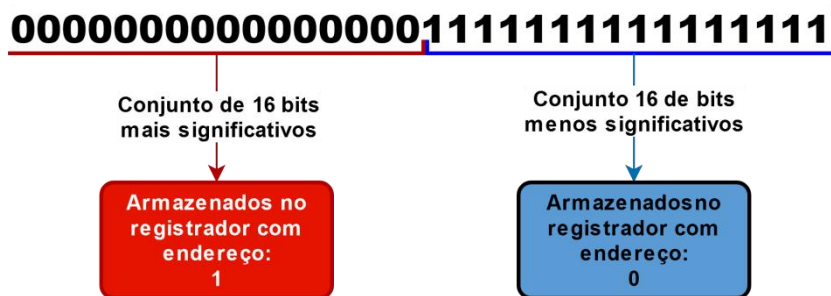


Figura 136 - Representação da alocação de dados nos registradores Modbus
 Fonte: autoria própria

Tendo em vista essas peculiaridades deste protocolo, na Tabela 4 é destacado todos os registradores que foram adicionados no microcontrolador, através da execução da função “def_Reg_ESP”, de forma que ela pudesse ser compreendida como uma complementação dos dados apresentados na Tabela 3, no que diz respeito ao endereço dos registradores da comunicação Modbus RTU.

No caso da função “att_Reg_ESP”, esta foi utilizada para atualizar o valor armazenado nos registradores na Tabela 4, tendo em vista os dados recebidos da comunicação do microcontrolador com os inversores. Neste sentido, na Figura 137 é apresentado o fluxograma relacionado a este procedimento de atualização dos dados armazenados nesses registradores.

No fluxograma presente na Figura 137, nota-se a utilização da função “mbRTU.task” novamente, como uma alternativa para a garantia da estabilidade na comunicação entre o microcontrolador e a IHM.

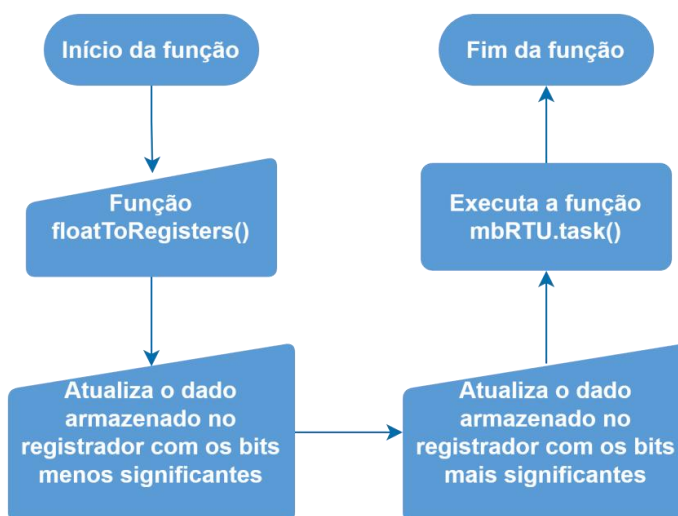


Figura 137 - Fluxograma da função att_Reg_ESP
 Fonte: autoria própria

Tabela 4 - Endereço dos registradores inseridos no microcontrolador

Endereço do Registrador	Descrição
0	Potência do inversor fotovoltaico - Bits menos significantes
1	Potência do inversor fotovoltaico - Bits mais significantes
2	Potência da rede - Bits menos significantes
3	Potência da rede - Bits mais significantes
4	Potência das baterias - Bits menos significantes
5	Potência das baterias - Bits mais significantes
6	Potência das cargas - Bits menos significantes
7	Potência das cargas - Bits mais significantes
10	Tensão do inversor fotovoltaico - Bits menos significantes
11	Tensão do inversor fotovoltaico - Bits mais significantes
12	Tensão da rede - Bits menos significantes
13	Tensão da rede - Bits mais significantes
14	Tensão das baterias - Bits menos significantes
15	Tensão das baterias - Bits mais significantes
16	Tensão das cargas - Bits menos significantes
20	Corrente do inversor fotovoltaico - Bits menos significantes
21	Corrente do inversor fotovoltaico - Bits mais significantes
22	Corrente da rede - Bits menos significantes
23	Corrente da rede - Bits mais significantes
24	Corrente das baterias - Bits menos significantes
25	Corrente das baterias - Bits mais significantes
26	Corrente das cargas - Bits menos significantes
27	Corrente das cargas - Bits mais significantes

Fonte: autoria própria

Além disso, a função “floatToRegisters” foi implementada com o objetivo de realizar a divisão dos dados dos registradores ilustrados na Figura 136, através da utilização de uma classe do tipo União (*union*). No caso da linguagem C++, utilizada como referência para a programação do microcontrolador, uma *union* se trata de um tipo semelhante a uma estrutura (*struct*), em que os membros dela compartilham o mesmo espaço de memória. Ademais,

apesar desse compartilhamento da mesma área de memória, cada membro pode ser interpretado como um tipo de dado distinto [56].

Como exemplo, uma *union* pode ser configurada para possuir 2 elementos, de forma que um deles seja do tipo ponto flutuante (*float*) e outro do tipo inteiro sem sinal (*unsigned int*). Assim, apesar de ambos ocuparem o mesmo espaço de memória, e possuírem o mesmo conjunto de bits, eles podem ser interpretados de maneira distinta, devido a conversão da codificação binária para a decimal específica de cada um desses tipos de dados.

Com base nisso, tendo em vista o fluxograma apresentado na Figura 138, uma *union* foi utilizada na função “floatToRegisters”, como um meio para armazenar um dado do tipo ponto flutuante de 32 bits, de maneira que ele também pudesse ser interpretado como um inteiro sem sinal de 32 bits. Neste caso, a informação armazenada nessa *union* se referia ao dado que foi recebido da comunicação do ESP32 com os inversores via comunicação Modbus TCP/IP, com a escala já corrigida, e formatado como *float*.

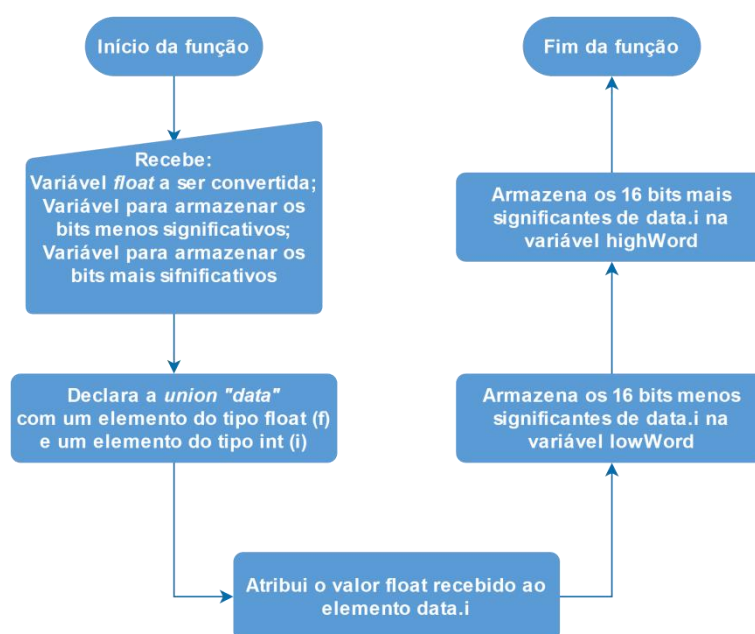


Figura 138 - Função "floatToRegisters"
Fonte: autoria própria

Assim, o algoritmo implementado na função ilustrada na Figura 138 consistia em receber uma informação que corresponde a um *float* de 32 bits, e separá-los em duas outras variáveis inteiras de 16 bits e sem sinal. Dessa forma, a primeira dessas variáveis do tipo inteira ficaria relacionada ao conjunto dos bits menos significantes (lowWord), enquanto a segunda corresponderia aos bits mais significantes (highWord) da informação, de forma que

fosse possível alocar elas separadamente nos seus respectivos registradores na Tabela 4, através da manipulação das variáveis “lowWord” e “highWord”.

Portanto, a partir disso, tem-se implementado todo o procedimento de programação da plataforma microcontrolada, de forma que após garantir a correta configuração da IHM em relação aos registradores definidos para o ESP32, houvesse a possibilidade de realizar a leitura dos dados definidos nesses endereços de memória. Assim, espera-se que a IHM consiga apresentar, em tempo real, os dados de operação tanto do inversor híbrido, quanto do inversor fotovoltaico, no que diz respeito ao parâmetros de tensão, corrente e potência.

6.3 CONEXÃO DO MICROCONTROLADOR COM A IHM

Com a configuração da IHM realizada, e a programação do microcontrolador concluída, a última etapa do procedimento para a implementação da interface de supervisão local consistiu em realizar a conexão física entre o ESP32 e a IHM. Para isso, utilizou-se das portas seriais de ambos dispositivos, que podem ser visualizadas na Figura 139.

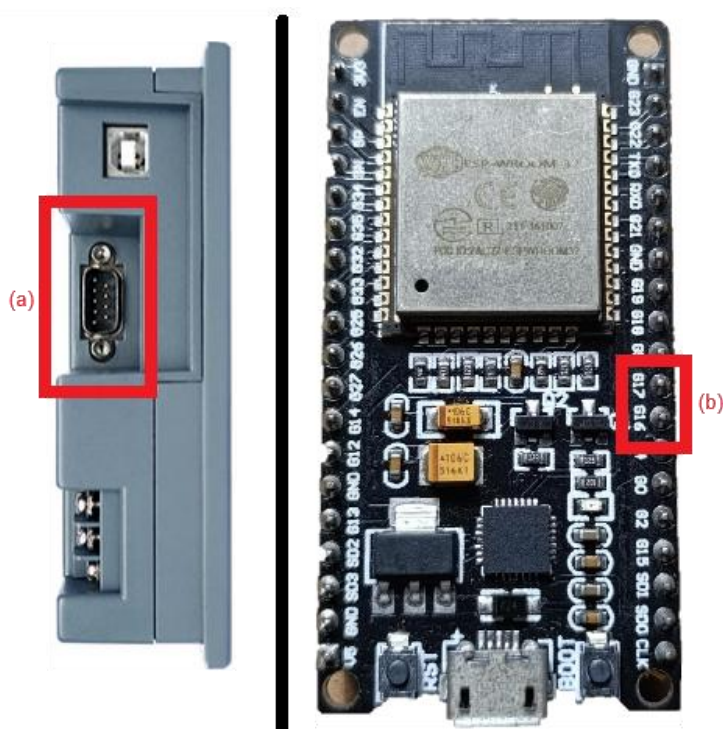


Figura 139 - Portas de comunicação serial da IHM (a) e do ESP32 (b)
Fonte: autoria própria

No caso da porta de comunicação serial da IHM, esta é disponibilizada através de um conector DB9 macho, conforme destacado na Figura 139, de forma que a conexão dos cabos de transmissão de dados nos terminais desse conector devam ser realizados de acordo com o padrão estabelecido para a comunicação Modbus, que por sua vez é ilustrado na Figura 140. Nesta figura, percebe-se que esta padronização para a conexão entre cliente e servidor Modbus no padrão RS232, deve ser feito seguindo a lógica de conexão cruzada RX (2) com TX (3) [27].

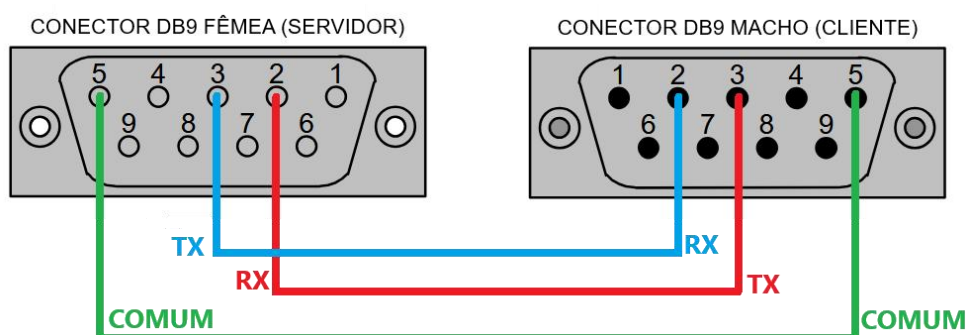


Figura 140 - Padrão de conexão do conector DB9 para Modbus RS232
 Fonte: adaptado de [27]

Ademais, levando em consideração a conexão com as portas seriais utilizadas no microcontrolador, que no caso deste trabalho se referiam aos terminais 16 (RX) e 17 (TX) do ESP32, fez-se necessário a implementação de um circuito adicional para a conversão do sinal enviado e recebido por este dispositivo. Isso decorreu do fato de o sinal enviado pelas portas seriais do microcontrolador não ser compatível com os sinais esperados da comunicação Modbus pela IHM. Assim, no caso da comunicação Modbus no padrão RS232, é esperado um nível de tensão entre -3 V e -25 V para indicar nível lógico alto, enquanto que os níveis de tensão entre 3 e 25 V fossem utilizados para indicar nível lógico baixo na transação de dados entre os dispositivos. Destaca-se, ainda, que o intervalo entre -3 V e 3 V representa uma zona de transição do sinal, em que qualquer informação compreendida entre estes dois valores passa a ser ignorada [25].

Com base nisso, observa-se que a tensão máxima fornecida pelos terminais do ESP32 é de 3,3 V, e que o protocolo de comunicação serial padrão desse dispositivo se trata do UART, através do padrão TTL. Dessa forma, o microcontrolador apenas tem a capacidade de enviar dados seguindo as regras deste protocolo de comunicação serial, em que, basicamente, os sinais de nível lógico alto correspondem a tensões de 3,3V, enquanto que tensões de 0 V destacam um bit de nível lógico baixo [57].

Portanto, tendo em vista essas características de comunicação que tanto o ESP32, quanto a IHM estão submetidos, surgiu a necessidade de utilizar de um dispositivo intermediário para a conversão dos sinais entre a comunicação desses equipamentos. Neste sentido, utilizou-se de um conversor do padrão TTL para RS232, ilustrado na Figura 141, de forma que o sinal enviado pelo microcontrolador no protocolo UART em 3,3 V, fosse convertido para o padrão RS232. De maneira complementar, utilizou-se desse mesmo conversor para realizar a normalização do sinal enviado pela IHM no padrão RS232, para o padrão TTL de 3,3 V, estabelecido no protocolo UART, ocasionando na garantia da correta interpretação dos dados transacionados com o microcontrolador.

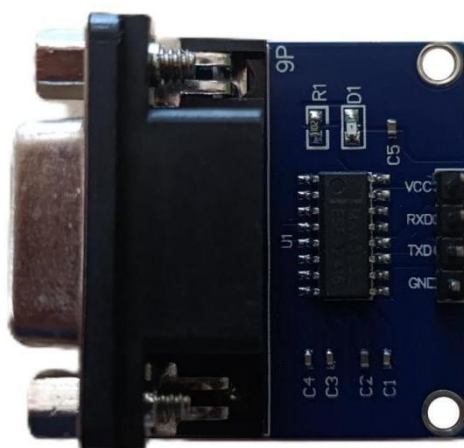


Figura 141 - Conversor TTL para RS232
Fonte: autoria própria

Assim, tendo como base os terminais VCC, RXD, TXD E GND do conversor TTL/RS232, visíveis na Figura 141, o circuito utilizado para integrar a IHM, o conversor de nível lógico, e o ESP32 pode ser encontrado na Figura 142. Neste esquema, percebe-se que os pinos RX e TX do microcontrolador foram conectados diretamente nos pinos RX e TX do conversor, tendo em vista que este se trata do padrão implementado neste último equipamento. Já os pinos VCC e GND do conversor foram conectados nas portas de 3,3V e GND do microcontrolador, já que este se trata do pino responsável por fornecer o sinal de referência ao terra, enquanto aquele deve fornecer a tensão de alimentação do conversor, que deve estar entre -0,3 V e 6 V [58]. Por fim, tanto a alimentação geral do microcontrolador, realizada nos pinos V5 e GND, quanto da IHM, através dos seus respectivos terminais de alimentação, foram realizadas através de uma fonte externa, de forma que ela fosse capaz de fornecer níveis de tensão estáveis de 5 V para o ESP32, e de 24 V para a IHM.

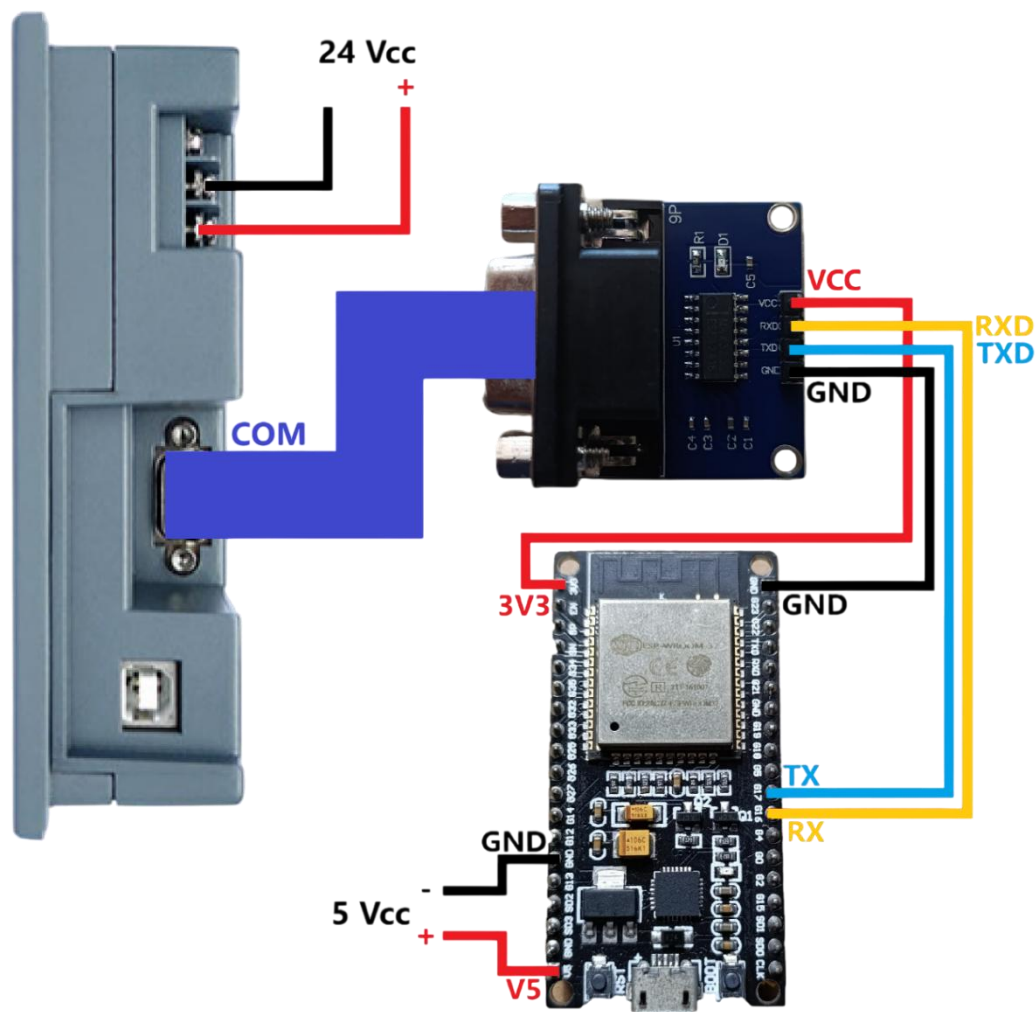


Figura 142 - Esquema de ligação do microcontrolador com a IHM
 Fonte: autoria própria

Com isso, foram realizadas todas as configurações necessárias para que a IHM conseguisse apresentar os dados de operação da microrrede, por meio da comunicação Modbus RTU estabelecida com o microcontrolador, no padrão RS232. Dessa forma, esperou-se ter um sistema capaz de monitorar os parâmetros de tensão, corrente e potência em tempo real, no que diz respeito a operação do inversor híbrido e do inversor fotovoltaico da microrrede.

7 CAPÍTULO VII - RESULTADOS E DISCUSSÃO

Com todas as configurações realizadas conforme as seções 3.2, 4.2, 5.1 e 6.1 foi estabelecido o sistema para a supervisão e controle de inversores para a aplicação em microrredes. Assim, com base no esquemático apresentado na Figura 1, obteve-se um sistema capaz de realizar o monitoramento e o controle em tempo real dos parâmetros de operação desses equipamentos, seja por meio de *smarthphones* ou computadores. Além disso, também é possível utilizar da interface web criada para realizar algumas funções de controle, como o desligamento dos inversores, controle de reativos, controle de potência injetada e despacho energia de energia armazenada. Neste sentido, na Tabela 5 é apresentado um resumo de todas as funcionalidades obtidas com a implementação deste sistema para supervisão e controle dos inversores.

Com base nisso, a operação do sistema foi baseada primeiramente pela requisição dos dados de operação dos dois inversores instalados no laboratório pelo microcontrolador. Esse processo ocorreu através da comunicação Modbus TCP/IP, e conforme os testes realizados, podem acontecer a uma frequência de atualização em torno de 2,5 Hz, de forma que 99 registradores sejam lidos na comunicação com o inversor fotovoltaico, assim como 169 registradores do inversor híbrido.

Com os dados dessa operação de leitura armazenados, o microcontrolador foi programado para realizar a publicação no servidor MQTT das últimas informações recebidas ao fim de um intervalo de três segundos, caso não tenha ocorrido nenhum problema na comunicação com os inversores. Assim, foi estabelecido que a atualização dos dados de monitoramento dos inversores ocorressem a cada três segundos no *broker*, de forma que esta também seja a taxa de atualização da interface de monitoramento desenvolvida no Node-RED.

Dessa forma, esses dados publicados foram tratados de forma a serem apresentados em cinco guias distintas na interface web, fazendo com que o usuário consiga navegar dentro do *dashboard*, e realizar a leitura e alteração dos parâmetros de interesse.

Ao mesmo tempo, esses dados foram registrados na memória do microcontrolador de maneira que a IHM pudesse realizar a leitura dos parâmetros de potência, tensão e corrente dos equipamentos da microrrede, através da comunicação Modbus no padrão RS232. Com isso, esses dados possuem uma taxa de atualização superior na interface de monitoramento local, tendo em vista que essas informações passam a ser renovadas assim que novos

procedimento de leitura são realizados no microcontrolador, em relação a comunicação com os inversores.

Tabela 5 - Resumo das funcionalidades do sistema de supervisão e controle

Funcionalidade	Descrição
Supervisão do inversor fotovoltaico	Monitoramento em tempo real dos parâmetros de tensão, corrente e potência na entrada e saída do inversor fotovoltaico
Supervisão do inversor híbrido	Monitoramento em tempo real dos parâmetros de tensão, corrente e potência na conexão com a rede, cargas e baterias
Controle do inversor fotovoltaico	Ajuste da limitação de injeção de potência do sistema fotovoltaico. Capacidade de ajuste de injeção ou consumo de potência reativa. Controle de conexão do inversor fotovoltaico à rede.
Controle do inversor híbrido	Controle do despacho de energia armazenada nas baterias. Capacidade de alterar o modo de operação da microrrede. Possibilidade de permitir ou negar o carregamento das baterias e/ou injeção de energia na rede.
Programação horária	Possibilidade de realizar a programação em um determinado intervalo do dia, para realização do despacho das baterias, limitação da injeção do inversor fotovoltaico e controle de reativos.
Consulta histórica	Consulta de dados históricos referente a tensão, corrente e potência da entrada e saída do inversor fotovoltaico, assim como da rede, cargas e baterias conectadas ao inversor híbrido.
Monitoramento local	Visualização do comportamento em tempo real dos parâmetros de supervisão em uma IHM local.

Fonte: autoria própria

7.1 GUIA WEB DE SUPERVISÃO GERAL

A primeira das guias de supervisão web, denominada de “Início”, apresenta as características gerais de operação da microrrede, fornecendo dados de tensão, corrente e potência dos inversores em tempo real. Essas informações de potência ainda são apresentadas

em gráficos, que demonstram a dinâmica desses parâmetros nos últimos 30 minutos, de forma a fornecer ao usuário uma visão geral do sistema para este intervalo. Assim, na Figura 143 é apresentado a interface gerada para a guia de supervisão geral deste trabalho.



Figura 143 - Guia de supervisão geral dos inversores
Fonte: autoria própria

7.2 GUIA WEB DE SUPERVISÃO E CONTROLE DO INVERSOR FOTOVOLTAICO

Em relação as funcionalidades de supervisão e controle específicas do inversor fotovoltaico, esta pode ser encontrada em uma guia própria, contendo uma gama de informações a respeito da operação do equipamento. Neste sentido, na Figura 144 é ilustrada a interface desenvolvida para tratar especificamente dos parâmetros de supervisão e controle do inversor fotovoltaico.

De acordo com a interface apresentada na Figura 144, nota-se a possibilidade de monitorar os parâmetros referentes a entrada e a saída do inversor fotovoltaico, com destaque para a supervisão a nível de MPPT do equipamento. Nos parâmetros de saída, percebe-se o monitoramento de tensão, corrente e frequência, além de todas as componentes de potência do equipamento. A dinâmica desses parâmetros ainda podem ser visualizadas por meio de recursos gráficos, que fornecem dados a respeito dos últimos 10 minutos de operação na entrada e saída do inversor.

Além disso, em relação as funcionalidades de controle do sistema, destaca-se um grupo específico para a configuração da limitação da potência ativa de injeção do equipamento, assim como do fator de potência de operação, e a desconexão do equipamento.

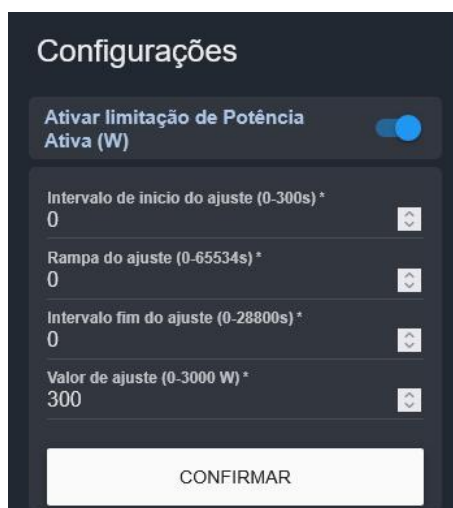


Figura 144 - Guia de supervisão e controle do inversor fotovoltaico
Fonte: autoria própria

7.2.1 Ajuste da limitação de potência

Com base na interface específica para a supervisão e controle do inversor fotovoltaico apresentada na Figura 144, tornou-se possível a configuração da limitação da potência de injeção do dispositivo, através da configuração na interface específica. Para isso, basta ativar

a configuração para esta operação, conforme destacado na Figura 145, e assim parametrizar os respectivos campos de acordo com indicado na figura. Dessa forma, ao confirmar a configuração desejada, será encaminhado um comando via MQTT ao microcontrolador, que por sua vez, realizará a validação desta parametrização diretamente no inversor, via comunicação Modbus, seguindo o esquema apresentado na Figura 1.



Configurações

Ativar limitação de Potência Ativa (W) ☒

Intervalo de início do ajuste (0-300s) *
0

Rampa do ajuste (0-65534s) *
0

Intervalo fim do ajuste (0-28800s) *
0

Valor de ajuste (0-3000 W) *
300

CONFIRMAR

Figura 145 - Configuração da limitação de potência ativa do inversor fotovoltaico
Fonte: autoria própria

Ainda é possível destacar que a parametrização dos campos que envolvem os intervalos de ajuste, assim como a rampa característica, podem ser configurados com valor de 0, de forma a definir a implementação da configuração como imediata.

De maneira complementar, o microcontrolador realizará uma nova atualização de leitura dos registradores do dispositivo, de forma que o resultado da configuração implementada seja possível de visualizar nos recursos de monitoramento do sistema. Assim, na Figura 146 é destacado o resultado da configuração realizada na Figura 145.



Figura 146 - Operação do inversor fotovoltaico com a potência de injeção limitada
Fonte: autoria própria

Por fim, para que o inversor volte a operar nas suas condições padrões de funcionamento, basta desativar o campo “Ativar limitação de Potência Ativa (W)” na Figura 145, de forma que assim o inversor volte a operar sem limitação de potência.

7.2.2 Ajuste do fator de potência de operação

Da mesma maneira que se realiza a limitação da potência do inversor fotovoltaico, também se tornou possível a alteração do fator de potência de operação, conforme destaca a configuração na Figura 147. Destaca-se, em tempo, a possibilidade de gerenciar o fator de potência do equipamento tanto para o consumo, quanto para a injeção de reativos, através do ajuste do sinal utilizado no campo “Valor de ajuste”. Assim, ao configurar o fator de potência com o sinal positivo, o sistema adquire característica indutiva, aumentando o consumo de reativos da instalação. Enquanto isso, se este campo for alterado para um valor negativo, a caracterização de um sistema capacitivo é implementada, de forma que o sistema passe a fornecer energia reativa para os equipamentos locais.

Ativar alteração de FP ☒

Intervalo de início do ajuste (0-300s) *
0

Rampa do ajuste (0-65534s) *
0

Intervalo fim do ajuste (0-28800s) *
0

Valor de ajuste (1-±0,85) *
0,92

CONFIRMAR

(a)

Ativar alteração de FP ☒

Intervalo de início do ajuste (0-300s) *
0

Rampa do ajuste (0-65534s) *
0

Intervalo fim do ajuste (0-28800s) *
0

Valor de ajuste (1-±0,85) *
-0,92

CONFIRMAR

(b)

Figura 147 - Configuração do fator de potência (a) indutivo e (b) capacitivo
 Fonte: autoria própria

Com base na configuração da Figura 147, foi possível obter o comportamento do sistema para as duas configurações realizadas, conforme destacado na Figura 148. Nesta figura, nota-se o acréscimo de potência reativa no sistema, durante a operação com fator de potência indutivo. Por outro lado, ao definir o fator de potência como capacitivo, nota-se que os índices de potência reativa se tornaram negativos, consolidando a característica capacitiva do sistema.



Figura 148 - Operação do inversor com fator de potência (a) indutivo e (b) capacitivo
 Fonte: autoria própria

7.2.3 Ajuste da conexão do inversor fotovoltaico

Por fim, a última configuração possível na interface de controle e supervisão do inversor fotovoltaico está relacionado com conexão e desconexão do equipamento da rede. Na

prática, esta configuração consiste no fechamento e abertura do relé na saída do equipamento, de forma que, ao ativar esta configuração, um comando para a desconexão do dispositivo será encaminhado ao microcontrolador. Diferentemente das demais configurações, esta possui uma parametrização mais simples, conforme destacado na Figura 149.

Figura 149 - Configuração de desconexão do inversor fotovoltaico
Fonte: autoria própria

Com base nisso, na Figura 150 é apresentado o resultado da configuração de desconexão do inversor fotovoltaico, sendo esta a última configuração possível de ser realizada na interface específica de supervisão e controle do dispositivo.



Figura 150 - Resultado da configuração de desconexão do inversor fotovoltaico
Fonte: autoria própria

7.3 GUIA DE SUPERVISÃO E CONTROLE DO INVERSOR HÍBRIDO

Da mesma forma que o inversor fotovoltaico possui uma interface específica para agrupar suas funcionalidades de monitoramento e controle, foi desenvolvido uma interface

específica para esses recursos, tendo em vista os dados da comunicação do microcontrolador com o inversor híbrido. Esta guia, por sua vez, pode ser visualizada na Figura 151.

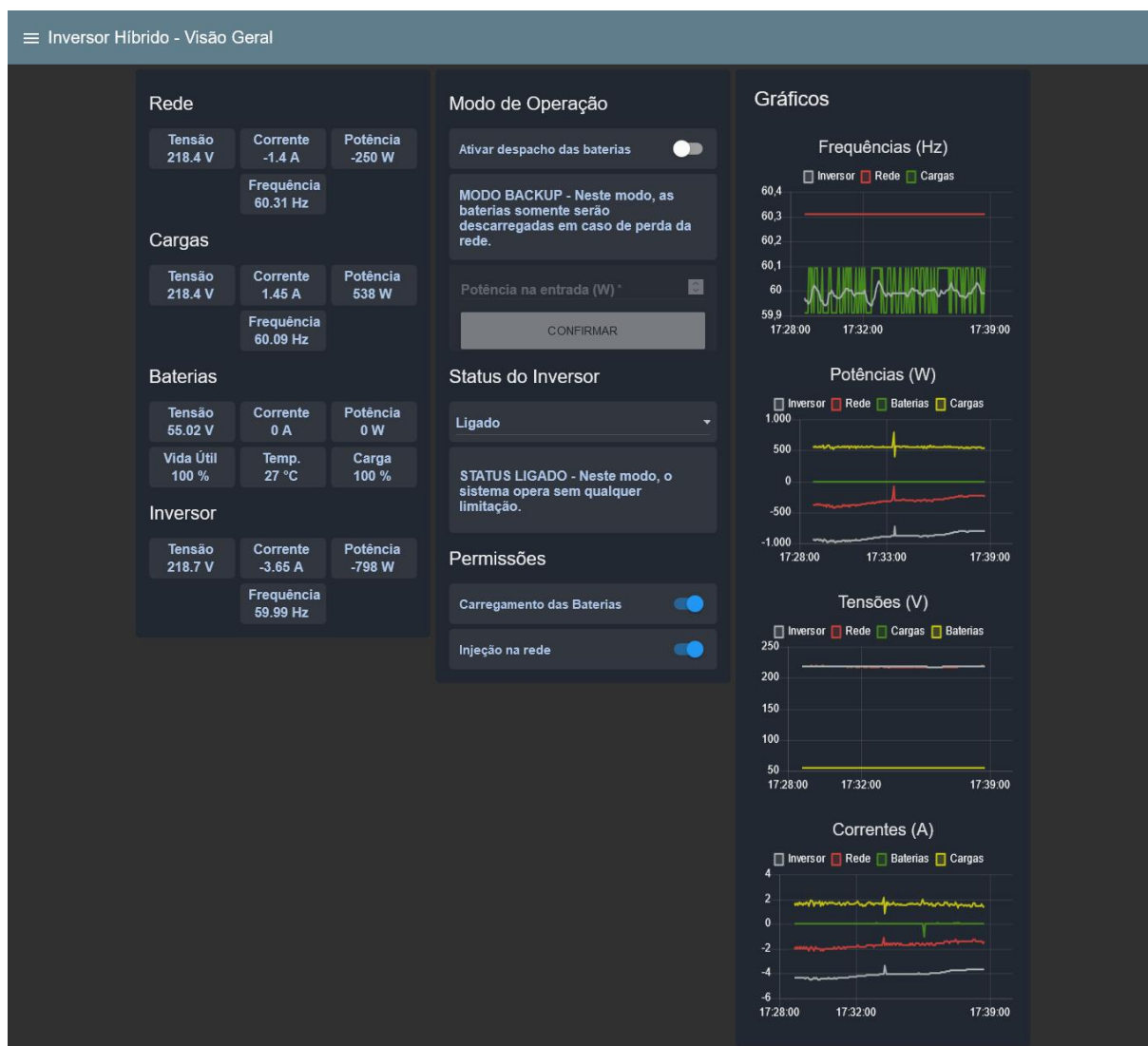


Figura 151 - Guia específica para a supervisão e controle do inversor híbrido

Fonte: autoria própria

Dentre as funcionalidades desta interface, pode-se destacar a supervisão dos parâmetros que envolvem tensão, corrente, potência e frequência da rede de entrada do dispositivo, das cargas conectadas ao inversor híbrido, e do inversor fotovoltaico. Além disso, ainda é possível monitorar estes mesmos dados de operação em relação as baterias, assim como a vida útil, a temperatura e o estado de carga que o conjunto de armazenamento se encontra.

De maneira complementar, a interface ainda fornece os meios necessários para o controle de energia despachada das baterias, assim como a alteração do status do inversor, e o controle do carregamento e da injeção na rede local.

7.3.1 Ajuste do despacho de energia armazenada

Tendo em vista a interface de supervisão e controle do inversor híbrido, na Figura 152 é destacado as configurações que podem ser realizadas para o gerenciamento da energia armazenada no banco de baterias. Ao ativar esta função, um novo modo de operação do inversor híbrido será definido, de forma que se tenha um maior controle da energia que se encontra armazenada no banco de baterias.

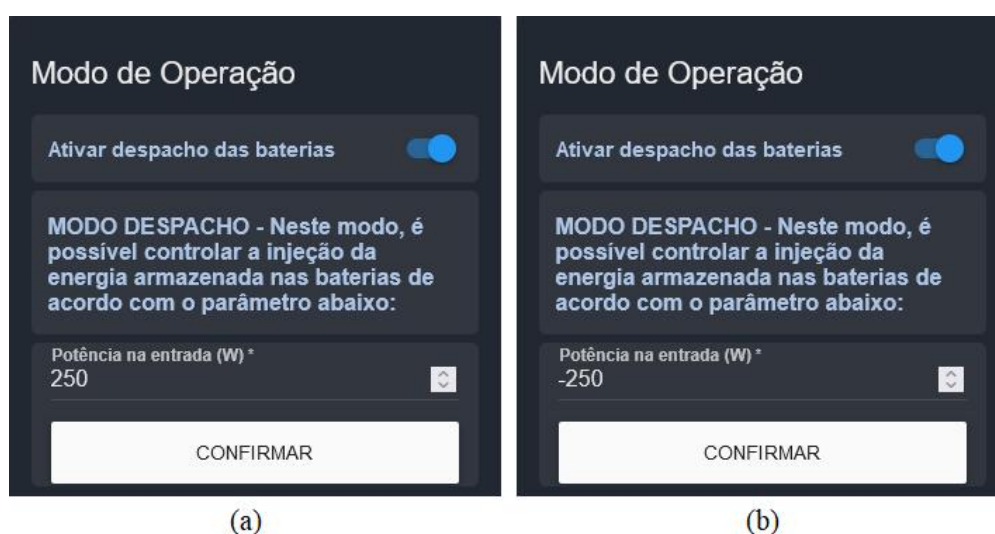


Figura 152 - Configuração do despacho de energia armazenada considerando (a) consumo máximo da microrrede e (b) injeção mínima na rede
Fonte: autoria própria

Como exemplo, na Figura 152 (a), tem-se configurado o valor de 250 W como referência para o inversor, sendo que, na prática, este valor corresponde a referência que o equipamento utilizará para gerenciar a energia interna da microrrede. Assim, caso o inversor fotovoltaico não tenha geração o suficiente para suprir a demanda das cargas da microrrede, o inversor híbrido utilizará apenas 250 W da rede elétrica conectada na entrada deste equipamento, e o restante será suprido pela energia das baterias. Caso o inversor fotovoltaico possua capacidade de fornecer essa energia para as cargas, ou até mesmo injetar energia na

rede de distribuição, essa configuração será mantida como a referência mínima, de forma que, na ocorrência de qualquer intermitência, o nível máximo de consumo da microrrede ainda seja de 250 W.

Dessa forma, o comportamento dessa configuração no sistema pode ser observado na Figura 153, em que, inicialmente, o sistema opera normalmente, sem o efeito de qualquer ajuste, e com o inversor fotovoltaico desligado. Com o recebimento do comando na Figura 152 (a), é configurado para que 250 W de potência das cargas sejam supridas pelas baterias, enquanto o restante passe a ser fornecido pela rede de distribuição. Nota-se que, com a posterior inicialização do inversor fotovoltaico, as baterias passaram a consumir o excedente de energia da geração fotovoltaica como uma forma de recarregar a sua carga e manter o ajuste realizado. Dessa forma, durante todo esse processo, percebe-se que o valor de potência em relação a rede não ultrapassou o valor de 250 W configurado na Figura 152 (a), e que, após o carregamento das baterias, quando a potência utilizada por elas atinge 0 W, o valor de potência injetada na rede se estabilizou próximo de -200 W. Isso acontece, devido a existência de excedente de geração fotovoltaica na rede interna da instalação, fazendo com que o inversor injete essa potência na rede de maneira automática, respeitando a configuração máxima de 250 W realizada na Figura 152 (a).



Figura 153 - Reposta do sistema a configuração positiva no despacho das baterias
Fonte: autoria própria

Por outro lado, na configuração realizada na Figura 152 (b), o valor negativo indica que o inversor híbrido deve injetar pelo menos 250 W na rede elétrica de entrada. Assim, nos momentos em que o inversor fotovoltaico possuir geração superior a demanda da microrrede, todo excedente será injetado, independente da configuração estabelecida nesta interface. Ademais, quando ele não possuir potência o suficiente para isso, o inversor híbrido irá utilizar a energia armazenada nas baterias para atingir esse valor, de forma que parte dela seja utilizada para suprir a demanda interna da microrrede, e o restante seja utilizado na injeção configurada.

Com base nisso, o comportamento dessa configuração no sistema pode ser observado na Figura 154, em que, inicialmente, o inversor fotovoltaico se encontra desconectado. Com isso, a energia armazenada nas baterias é utilizada para garantir o nível de injeção configurado na Figura 152 (b), e também a energia utilizada pelas cargas. Com a reconexão do inversor fotovoltaico, o excedente de energia da geração do sistema é utilizado para carregar as baterias, de forma a manter o valor próximo de -250 W na entrada do inversor híbrido.



Figura 154 - Resposta do sistema a configuração negativa no despacho das baterias
Fonte: autoria própria

Por fim, ao desativar essa configuração na interface apresentada na Figura 152, o inversor retorna para o modo de backup, em que a energia armazenada nas baterias somente será utilizada quando ocorrer perda da rede de entrada.

7.3.2 Ajuste do status do inversor híbrido

A outra configuração passível de ser realizada, apresentada na Figura 155, está relacionada com a característica de operação em regime permanente do equipamento.

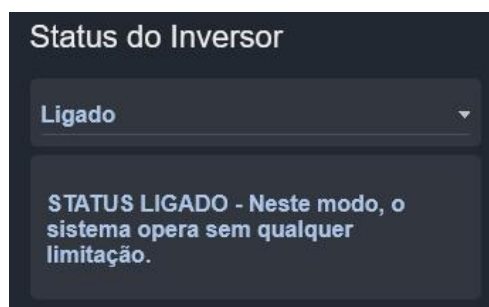


Figura 155 - Configuração do status do inversor híbrido
Fonte: autoria própria

Nesta configuração, é possível configurar o inversor para alterar o seu estado de acordo com quatro opções. A primeira dessas configurações, destacada na Figura 155, permite que o inversor funcione sem restrições, com todas as funcionalidades operando. Já a segunda, define que o inversor deve operar apenas como carregador das baterias, de forma que a energia armazenada no banco nunca seja utilizada em qualquer hipótese. Há ainda a configuração para operação ilhada, em que o inversor irá realizar a desconexão completa com a rede de entrada, de forma a ser mantido apenas com os REDs internos da microrrede. Por fim, o último modo é utilizado para enviar um sinal de desligamento imediato ao inversor, que por sua vez será mantido em *stand-by* enquanto aguarda por uma nova alteração do seu estado.

7.3.3 Ajuste das permissões de operação do inversor híbrido

Por fim, a última configuração possível de ser utilizada nessa interface, está relacionada com a permissão de carregamento das baterias, e a injeção de energia na rede de entrada. Conforme apresentado na Figura 156, essas alterações podem ser realizadas por meio dos *switches* da interface de controle das permissões, de forma que, mantê-los ativados garante que o inversor híbrido consiga carregar as baterias, e também injetar o excedente de

geração do inversor fotovoltaico e do despacho das baterias, como destaca a configuração da seção 7.3.1.



Figura 156 - Configuração da permissão de carregamento e injeção do inversor híbrido
Fonte: autoria própria

7.4 GUIA DE CONFIGURAÇÃO PROGRAMÁVEL DOS INVERSORES

Em relação a guia de configuração programável dos inversores, essa pode ser visualizada na Figura 157, de forma que seja possível notar que essas interfaces se tratam de adaptações daquelas usadas nas seções 7.2.1, 7.2.2 e 7.3.1, de forma a aceitarem a entrada de um horário de início para a implementação da configuração, e um horário para a finalização dela.

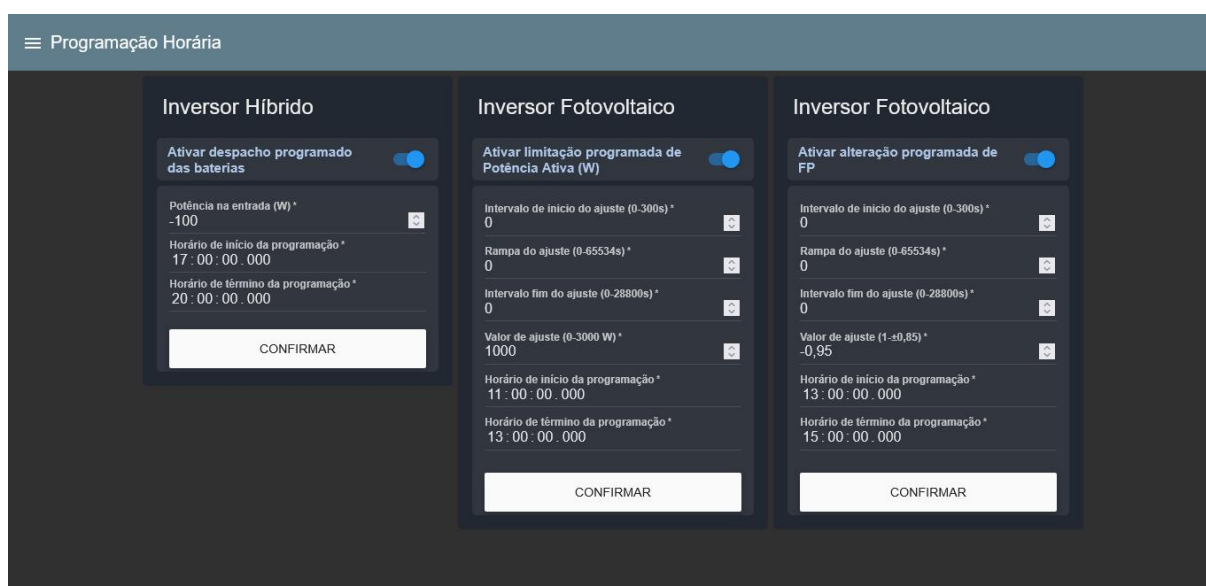


Figura 157 - Interface de programação horária dos inversores
Fonte: autoria própria

Assim as configurações desenvolvidas nessas interfaces somente serão implementadas nos horários estabelecidos em cada uma, de forma que, nos momentos fora desse intervalo, a configuração padrão de cada equipamento seja estabelecida. Ao desativar os “switches” de cada uma das interfaces, a configuração padrão também será reestabelecida para os dispositivos.

Com base nisso, na Figura 158, Figura 159 e Figura 160, é apresentada a resposta do sistema para cada uma das configurações demonstradas na Figura 157, sendo elas a de despacho das baterias, limitação da potência do inversor fotovoltaico, e alteração do fator de potência, respectivamente. No caso da primeira dessas figuras, pode-se notar que a energia das baterias somente passou a ser utilizada após o horário predefinido na Figura 157, de forma que potência total fornecida por elas seja igual a potência das cargas somadas a configuração de injeção na rede. A partir do momento em que o horário pré estabelecido é finalizado, nota-se uma elevação da potência consumida pela rede, ocasionado pela demanda de recarga das baterias, já que o consumo das cargas sempre se estabeleceu constante.



Figura 158 - Resposta da programação de despacho da energia armazenada nas baterias
Fonte: autoria própria

No caso do comportamento destacado na Figura 159, percebe-se o efeito da limitação da potência de injeção do inversor fotovoltaico, considerando tanto a entrada quanto a saída do equipamento. Neste sentido, percebe-se que este comportamento perdurou apenas durante o horário programado na Figura 157, e que tanto a potência de entrada, quanto a de saída foram afetadas.



Figura 159 - Resposta da programação de limitação de injeção do inversor fotovoltaico
 Fonte: autoria própria

Por fim, na Figura 160, pode-se visualizar o comportamento do sistema para a injeção de reativos pelo inversor fotovoltaico, de forma que a potência reativa do sistema seja caracterizada como capacitiva, uma vez que o valor configurado na Figura 157 e visível na Figura 160, são negativos.



Figura 160 - Resposta da programação de alteração do fator de potência do inversor fotovoltaico
 Fonte: autoria própria

7.5 GUIA DE CONSULTAS HISTÓRICAS

Por fim, a última interface disponível no sistema de supervisão e controle web dos inversores se trata da guia de consultas históricas, que, conforme destacado na Figura 1, depende inteiramente da comunicação do Node-RED com um banco de dados MySQL local. Essa comunicação, por sua vez, permite que o Node-RED realize o registro de dados no banco ao fim de cada intervalo de quinze minutos, tendo em vista que, para registrar esses parâmetros, é calculado uma média de todos os dados de interesse que foram publicados no *broker* durante esse mesmo período.

Assim, os dados que foram configurados para serem registrados, e por este motivo estão disponíveis para a consulta histórica, são as tensões, correntes e potências na entrada e na saída do inversor fotovoltaico. Em relação ao inversor híbrido, os mesmos parâmetros também estão disponíveis para consulta na interface, conforme apresentado na Figura 161, com o complemento das informações relacionadas ao conjunto de baterias conectada ao equipamento.

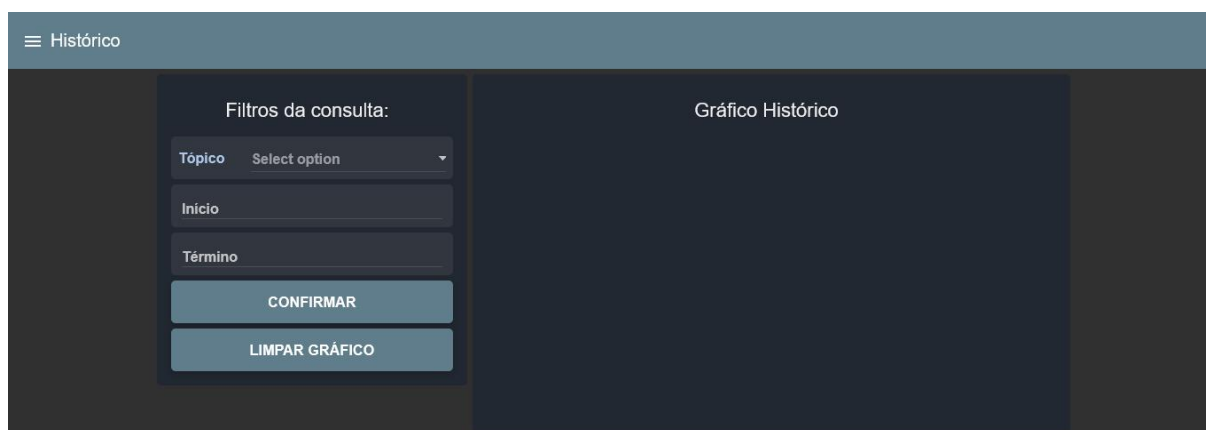


Figura 161 - Interface para a configuração da consulta histórica dos dados dos inversores
Fonte: autoria própria

Assim, para que seja possível utilizar essa interface, basta selecionar todos os tópicos de interesse para a consulta, assim como o momento de início e fim do intervalo de interesse para a análise. Destaca-se, em tempo, que os campos identificados com “BATERIA”, “CARGAS” e “REDE” são referentes aos dados do inversor híbrido, enquanto os demais destacam as características do inversor fotovoltaico.

Com base nisso, na Figura 158, Figura 159 e Figura 160, é destacado os resultados das consultas históricas realizadas com a utilização desta funcionalidade, de forma que todas as informações apresentadas possam ser condensadas com a configuração apresentada na Figura 162, considerando o período completo de um dia. Com isso, todos os tópicos referentes ao conjunto de potências na microrrede foram selecionados, de forma que o gráfico apresentado nessas figuras apresentassem uma visão geral a respeito do comportamento do sistema durante essas configurações.



Figura 162 - Configuração de consulta na interface de dados históricos
Fonte: autoria própria

Assim, o resultado esperado para a consulta configurada na interface visível na Figura 162 pode ser melhor visualizado na Figura 163, de forma que, para o mesmo intervalo, a Figura 164 e a Figura 165 também tenham sido capturadas para demonstrar o comportamento das correntes e tensões do sistema, respectivamente.



Figura 163 - Consulta histórica para os dados de potência da microrrede
 Fonte: autoria própria



Figura 164 - Consulta histórica para os dados de corrente da microrrede
 Fonte: autoria própria



Figura 165 - Consulta histórica para os dados de tensão da microrrede
Fonte: autoria própria

7.6 INTERFACE DE SUPERVISÃO LOCAL

Em relação a IHM utilizada como meio para a supervisão local dos parâmetros de operação dos inversores da microrrede, desenvolveu-se um invólucro para alojar todos os componentes necessários para o funcionamento deste componente do sistema. Neste sentido, foi utilizado de uma caixa em material de MDF, que comportasse todos os dispositivos apresentados na Figura 142, resultando no protótipo apresentado na Figura 166.

Com isso, a primeira interface de monitoramento pode ser acessada ao tocar sobre a tela da IHM, uma vez que ela responde a este tipo de estímulo, de forma que os dados apresentados sejam atualizados em tempo real, na tela destacada na Figura 111.

Conforme apresentado anteriormente, ainda se torna possível navegar pelas telas desenvolvidas através dos toques pelo lado direito (avanço de tela) ou esquerdo (retorno de tela) da IHM. Neste sentido, é possível navegar até a interface dos gráficos de potência, de forma que seja possível visualizar a dinâmica desses parâmetros nos últimos 3 minutos. Assim, na Figura 112 são ilustrados os dados do recurso gráfico implementado para a supervisão local da microrrede objeto de estudo deste trabalho.



Figura 166 - Protótipo do dispositivo de supervisão local
Fonte: autoria própria

8 CAPÍTULO VII - CONSIDERAÇÕES FINAIS

Com base nas informações apresentadas nas seções anteriores, observa-se que o sistema desenvolvido neste trabalho apresenta uma interface para supervisão e controle da microrrede de estudo em um nível primário, tendo em vista a sua capacidade de comunicação direta entre os dispositivos da instalação. Reitera-se que esse gerenciamento dos inversores instalados na microrrede é feito diretamente pelo microcontrolador e a lógica implementada neste equipamento, uma vez que somente este dispositivo realiza a comunicação direta com esses equipamentos. Assim, as interfaces criadas funcionam apenas como uma ferramenta entre o controlador e o usuário, de forma que todo processamento seja realizada pelo ESP32, no que diz respeito a operação desses dispositivos.

Dessa forma, entende-se que este trabalho cumpriu com os objetivos estabelecidos, uma vez que o resultado final envolve duas interfaces para supervisão, e uma para o controle, dos inversores que compõem a microrrede instalada no IFG Campus Itumbiara. Além disso, considerando a interface web desenvolvida, foi disponibilizado uma ferramenta capaz de realizar a consulta de informações históricas do sistema, através da integração com um banco de dados local.

Em relação ao desenvolvimento das interfaces de controle web do sistema, estas oferecem recursos para o controle de potência ativa e reativa de operação do inversor fotovoltaico, que por sua vez, de destacam como funcionalidades com alta aplicabilidade no contexto das microrredes. Isso se deve ao fato de este primeiro controle ser essencial para a manutenção da relação de geração e demanda da microrrede, principalmente durante a operação isolada da rede de distribuição, mas também para operação conectada, quando se tem a necessidade de limitar a demanda de geração. Da mesma maneira, o controle auxiliar de energia reativa através da utilização dessa interface, também se trata de uma funcionalidade com alta aplicabilidade para consumidores cuja a utilização dessa energia é considerada nos períodos de faturamento. Com isso, obteve-se a possibilidade de utilizar do inversor fotovoltaico como uma ferramenta para o controle de reativos de uma determinada instalação.

Em relação a possibilidade de realizar o controle do despacho da energia armazenada nas baterias, esta também se trata de uma funcionalidade extremamente vantajosa para a operação conectada a rede de distribuição. Essa afirmação se deve a possibilidade de utilizar desta energia para suprir a demanda interna de uma microrrede em sua totalidade durante o período de ponta, por exemplo, em que o preço da tarifa de energia é mais elevado.

Ainda se torna possível o estabelecimento de um cronograma para o carregamento e descarregamento de energia da armazenada, com o objetivo de maximizar o retorno financeiro ou a eficiência na compensação energética. Para isso, o sistema pode ser configurado para que o inversor híbrido carregue as baterias apenas durante os períodos de maior produção do sistema fotovoltaico. Assim, com as baterias completamente carregadas, torna-se viável a programação da injeção dessa energia na rede de distribuição durante os horários de pico, garantindo que haja uma compensação mais vantajosa para a energia fornecida para a rede de distribuição. Assim, a utilização das interfaces de programação horária se fazem úteis nesses momentos em que se tem um período pré estabelecido do dia para a utilização dessas ferramentas de controle.

Por fim, também se tem a possibilidade de realizar a injeção da energia armazenada a qualquer momento, o que em conjunto com sistemas de geração fotovoltaica ou eólica, pode ser uma alternativa para o suprimento de geração durante os momentos de intermitência dessas fontes.

Portanto, com a disponibilidade de todas essas informações via *broker*, no servidor Node-RED, no banco de dados, e também na IHM, tem-se um ambiente propício para realização de novos trabalhos que podem envolver modelos avançados para análises de dados, assim como novas ferramentas de controle integrados a IHM.

No caso da primeira alternativa, pode-se utilizar de modelos de Inteligência Artificial que busquem por possíveis padrões que possam auxiliar na tomada de decisões no gerenciamento dos REDs disponíveis na microrrede. Com isso, torna-se possível obter indicadores quantitativos quanto a implementação de melhorias na instalação, visando um aumento na eficiência do sistema, por exemplo.

Em relação a segunda opção, ainda há a possibilidade de adaptar as mesmas ferramentas de controle utilizadas na interface web, de forma a fornecer uma maior gama de recursos para controle local dos inversores.

Dessa forma, este trabalho comprovou que a integração de diversos sistemas cria um ambiente robusto e flexível, capaz de viabilizar o aprimoramento contínuo do gerenciamento desta microrrede. Assim, as propostas apresentadas abrem novas perspectivas para a implementação de soluções ao setor elétrico, que podem ocasionar no aumento da eficiência e da confiabilidade de seus componentes, contribuindo significativamente para o avanço tecnológico e a sustentabilidade de todo o sistema de geração e distribuição de energia.

REFERÊNCIAS

- [1] ANEEL divulga os resultados do desempenho das distribuidoras na continuidade do fornecimento de energia elétrica em 2023. **Agência Nacional de Energia Elétrica**, 2024. Disponível em: <<https://www.gov.br/aneel/pt-br/assuntos/noticias/2024/aneel-divulga-os-resultados-do-desempenho-das-distribuidoras-na-continuidade-do-fornecimento-de-energia-eletrica-em-2023>>. Acesso em 10 de dez. de 2024.
- [2] 2022: Confira o ranking da continuidade do serviço de distribuição de energia elétrica de 2022. **Agência Nacional de Energia Elétrica**, 2023. Disponível em: <<https://www.gov.br/aneel/pt-br/centrais-de-conteudos/relatorios-e-indicadores/distribuicao/ranking-de-continuidade/2022>>. Acesso em 10 de dez. de 2024.
- [3] 2021: Confira o ranking da continuidade do serviço de distribuição de energia elétrica de 2021. **Agência Nacional de Energia Elétrica**, 2022. Disponível em: <<https://www.gov.br/aneel/pt-br/centrais-de-conteudos/relatorios-e-indicadores/distribuicao/ranking-de-continuidade/2021>>. Acesso em 10 de dez. de 2024.
- [4] PLANO Decenal de Expansão de Energia 2034. **EPE**. Disponível em: <<https://www.epe.gov.br/pt/publicacoes-dados-abertos/publicacoes/plano-decenal-de-expansao-de-energia-2034>>. Acesso em 11 de dez. de 2024.
- [5] NASCIMENTO, A. C. V. **Projeto, estudo e construção de uma microrrede experimental para testes de equipamentos e controle**. Dissertação (Mestrado em Engenharia Elétrica) - Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal do Maranhão. São Luís, 187 p. 2022.
- [6] ALCÂNTARA, M. V. P. **Microrredes inteligentes: um novo modelo de negócio para a distribuição de energia elétrica**. O setor elétrico, v. 71, p. 36-45, 2011.
- [7] SOUZA, M. E. T; FREITAS, L. C. G. **Microrredes: Estado da arte, desafios e tendências para geração, distribuição e uso sustentável de energia elétrica / Microgrids: State-of-the-art, challenges and trends for the sustainable generation, distribution and use of electricity**. Brazilian Applied Science Review, 2020.
- [8] MARQUES, D. da C. **Controle de frequência em microrredes utilizando baterias baseado no aprendizado emocional do cérebro**. 2023. 208 f. Tese (Doutorado em Engenharia Elétrica). Centro de Tecnologia de Geociências, Universidade Federal de Pernambuco, Recife, 2023.
- [9] HOSSAIN, M. A. et al. **Evolution of microgrids with converter-interfaced generations: Challenges and opportunities**. International Journal of Electrical Power and Energy Systems, v. 109, p. 160–186, 2019.
- [10] BRASIL. Lei Nº 14.300, de 6 de janeiro de 2022. Institui o marco legal da microgeração e minigeração distribuída, o Sistema de Compensação de Energia Elétrica (SCEE) e o Programa de Energia Renovável Social (PERS). Brasília, DF: Diário Oficial da União, 2022.

- [11] WANG, X. et al. **A review of power electronics based microgrids**. Journal of power electronics, v. 12, p. 181–192, 2012.
- [12] EID, B. M. et al. **Control methods and objectives for electronically coupled distributed energy resources in microgrids: A review**. IEEE Systems Journal, v. 10, n. 2, p. 446–458, 2016.
- [13] DRIESEN, J; KATIRAEI, F. **Design for Distributed Energy Resources**. IEE power & energy magazine, v. 6, p. 30–40, 2008.
- [14] 2030.7-2017 - IEEE Standard for the Specification of Microgrid Controllers. **IEE Xplore**, 2018. Disponível em: <<https://ieeexplore.ieee.org/document/8340204>>. Acesso em 15 de dez. de 2024.
- [15] Painel de Dados de Micro e Minigeração Distribuída. **Empresa de Pesquisa Energética, 2023**. Disponível em: <<https://dashboard.epe.gov.br/apps/pdgd/>>. Acesso em 16 de dez. de 2024.
- [16] VILLALVA, M. G; GAZOLI, J. R. **Energia solar fotovoltaica: conceitos e aplicações**. São Paulo: Érica, 2012.
- [17] KASSATLY, S. **The Lithium-Ion Battery Industry for Electric Vehicles**. 2010. 124 f. Thesis (Master of Science in Mechanical Engineering). Department of Mechanical Engineering, Massachusetts Institute of Technology, 2010.
- [18] Victron & BYD. **Victron Energy: Blue Power**. Disponível em: <https://www.victronenergy.com/live/battery_compatibility:byd_b-box>. Acesso em 17 de dez. de 2024.
- [19] B-Box user Manual. **BYD**, 2016. Disponível em: <<https://portal.segensolar.co.za/reseller/docs/B-BOX2.5-10.0%20User%20manual%20V2.0.pdf>>. Acesso em 19 de dez. de 2024.
- [20] VE.Can to CAN-bus BMS cables manual. **Victron Energy: Blue Power**. Disponível em: <https://www.victronenergy.com/live/battery_compatibility:can-bus_bms-cable>. Acesso em 19 de dez. de 2024.
- [21] TANENBAUM, A; FEAMSTER N; WETHERALL, D. **Redes de computadores**. 6ª Edição. Distrito Federal: Pearson Educação, 2003.
- [22] ROSÁRIO, J. M. **Princípios de Mecatrônica**. 1ª Edição. São Paulo: Prentice Hall, 2005.
- [23] **MODBUS FAQ: About the protocol**. Modbus Organization. Disponível em: <<https://modbus.org/faq.php>>. Acesso em: 27 de dez. de 2024.
- [24] **MODBUS Application Protocol Specification V1.1b3**. Modbus Organization, 2012. Disponível em: <https://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf>. Acesso em: 28, de dez. de 2024
- [25] **THE RS232 Standard**. Cami Research Inc, 2024. Disponível em: <https://www.camiresearch.com/Data_Com_Basics/RS232_standard.html#anchor1155404>. Acesso em: 06 de fev. de 2025.

- [26] KUGELSTADT, T. **The RS-485 Design Guide**. Texas Instruments, 2021.
- [27] **MODBUS over Serial Line Specification and Implementation Guide V1.02**. Modbus Organization, 2006. Disponível em: <https://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf>. Acesso em: 28, de dez. de 2024.
- [28] SANTANA, B. O que é o Protocolo TCP/IP e Como Ele Funciona?. Hostinger Tutoriais, 2024. Disponível em: <<https://www.hostinger.com.br/tutoriais/tcp-ip>>. Acesso em: 28, de dez. de 2024.
- [29] **MODBUS Messaging On TCP/IP Implementation Guide V1.0b**. Modbus Organization, 2006. Disponível em: <https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf>. Acesso em: 28, de dez. de 2024.
- [30] **FRONIUS Primo 3.0-1 - 8.2-1 Manual de Instruções**. Fronius. Disponível em: <<https://manuals.fronius.com/html/4204102165/pt-BR.html>>. Acesso em: 29 de dez. de 2024.
- [31] **GEN24 Primo Symo Inverter Register Map Int&SF**. Fronius Company. Disponível em: <<https://www.fronius.com/~/downloads/Solar%20Energy/Operating%20Instructions/gen24-modbus-api-external-docs.zip>>. Acesso em: 29 de dez. de 2024.
- [32] **OPERATING Instructions: Fronius Datamanager Modbus TCP & RTU**. Fronius Company. Disponível em: <<https://www.fronius.com/~/downloads/Solar%20Energy/Operating%20Instructions/42,0410,2049.pdf>>. Acesso em: 29 de dez. de 2024.
- [33] **INFORMAÇÃO Técnica**. Victron Energy. Disponível em: <<https://www.victronenergy.pt/support-and-downloads/technical-information>>. Acesso em: 02 de jan. de 2025.
- [34] KERSCHBAUMER, R. **Microcontroladores**. Luzerna, 2018.
- [35] LOPES, G. V. Q. **Desenvolvimento de uma Estação Solarimétrica de Baixo Custo Utilizando Internet das Coisas – IOT**. TCC (Bacharelado em Engenharia Elétrica) – Instituto Federal de Educação, Ciência e Tecnologia de Goiás - Campus Itumbiara. Itumbiara, p. 169. 2022.
- [36] ARDUINO. **Docs: Overview of the Arduino IDE 1**. Arduino Documentation, 2024. Disponível em: <<https://docs.arduino.cc/software/ide-v1/tutorials/Environment/#boards>>. Acesso em: 03 de jan. de 2025.
- [37] ARDUINO. **Docs: WiFi**. Arduino Documentation, 2018. Disponível em: <<https://docs.arduino.cc/libraries/wifi/>>. Acesso em: 03 de jan. de 2025.
- [38] **ESTRUTURAS**. Universidade Federal de Minas Gerais Instituto de Ciências Exatas Departamento de Ciência da Computação: Algoritmos e Estruturas de Dados I. Disponível em: <<https://homepages.dcc.ufmg.br/~rodolfo/aeds-i-2-10/Estruturas/estruturas.html>>. Acesso em: 04 de jan. de 2025.

- [39] EMELIANOV, A.; BARBOSA, A. S. **library-arduino-modbus-esp8266**. GitHub, 2017. Disponível em: <<https://github.com/m2m-solutions/library-arduino-modbus-esp8266>>. Acesso em: 03 de jan. de 2025.
- [40] ERI, R.; LOMBA, M.; BULHÕES, G. **MQTT**. MQTT, 2019. Disponível em: <<https://www.gta.ufrrj.br/ensino/eel878/redes1-2019-1/vf/mqtt/>>. Acesso em: 06 de jan. de 2025.
- [41] **ECLIPSE Mosquitto: An Open Source MQTT Broker**. mosquitto. Disponível em: <<https://mosquitto.org/>>. Acesso em: 06 de jan. de 2025.
- [42] **DOWNLOAD**. mosquitto. Disponível em: <<https://mosquitto.org/download/>>. Acesso em: 06 de jan. de 2025.
- [43] **DOWNLOADS**. Notepad++. Disponível em: <<https://notepad-plus-plus.org/downloads/>>. Acesso em: 07 de jan. de 2025.
- [44] O'LEARY, Nicholas. **knolleary/pubsubclient**. GitHub, 2020. Disponível em: <<https://github.com/knolleary/pubsubclient>>. Acesso em: 09 de jan. de 2025.
- [45] **ABOUT**. Node-RED. Disponível em: <<https://nodered.org/about/>>. Acesso em: 11 de jan. de 2025.
- [46] **RUNNING on Windows**. Node-RED. Disponível em: <<https://nodered.org/docs/getting-started/windows>>. Acesso em: 11 de jan. de 2025.
- [47] **RUN JavaScript Everywhere**. Node.js. Disponível em: <<https://nodejs.org/en/>>. Acesso em: 11 de jan. de 2025.
- [48] **SECURITY Node-RED**. Node-RED. Disponível em: <<https://nodered.org/docs/user-guide/runtime/securing-node-red>>. Acesso em: 11 de jan. de 2025.
- [49] **SOBRE**. Apache Friends, 2024. Disponível em: <https://www.apachefriends.org/pt_br/about.html>. Acesso em: 14 de jan. de 2025.
- [50] **O que é o MYSQL?**. Oracle, 2025. Disponível em: <<https://www.oracle.com/br/mysql/what-is-mysql/>>. Acesso em: 14 de jan. de 2025.
- [51] **XAMPP**. SOURCEFORGE, 2023. Disponível em: <<https://sourceforge.net/projects/xampp/>>. Acesso em: 14 de jan. de 2025.
- [52] **NODE-RED Dashboards Charts**. GitHub, 2020. Disponível em: <<https://github.com/node-red/node-red-dashboard/blob/master/Charts.md>>. Acesso em: 15 de jan. de 2025.
- [53] EA-043A. Samkoon. Disponível em: <<https://www.samkoon.store/ea-043a/>>. Acesso em: 30 de jan. de 2025.
- [54] FERREIRA, T. G. **Como utilizar IHM com ESP32**. CRESCER Indústria de Automação, 2024. Disponível em: <<https://www.crescerengenharia.com/post/utilizar-ihm-esp32>>. Acesso em: 30 de jan. de 2025.

- [55] **DOWNLOAD HMI Software**. Samkoon. Disponível em: <<https://www.samkoon.store/download/>>. Acesso em: 30 de jan. de 2025.
- [56] **UNION**. Microsoft Learn, 2024. Disponível em: <<https://learn.microsoft.com/pt-br/cpp/cpp/unions?view=msvc-170>>. Acesso em: 06 de fev. de 2025.
- [57] UART Protocol. Labcenter Eletronics, 2025. Disponível em: <<https://www.labcenter.com/blog/sim-uart/#header-10>>. Acesso em: 06 de fev. de 2025.
- [58] **3.0V to 5.5V, Low-Power, up to 1Mbps, True RS-232 Transceivers Using Four 0.1μF External Capacitors**. Rev. 5. Maxim, Sunnyvale, 1999.
- [59] MQTT: The Standard for IoT Messaging. **MQTT**, 2024. Disponível em: <<https://mqtt.org/>>. Acesso em: 19 de fev. de 2025.

APÊNDICE A - CÓDIGO DO MICROCONTROLADOR ESP32

```
#include <WiFi.h> // Biblioteca para comunicação Wi-Fi do ESP32
#include <ModbusIP_ESP8266.h> // Biblioteca para comunicação Modbus TCP com o
ESP como servidor/server
#include <PubSubClient.h> // Biblioteca para comunicação MQTT
#include <ModbusRTU.h> // Biblioteca para a comunicação Modbus RTU com
a IHM Local

// Estrutura para armazenar os dados dos registros da comunicação Modbus
struct dados {
    uint16_t registro; // Armazena os registros das variáveis na
    comunicação Modbus
    uint16_t qtd_Reg; // Quantidade de registros que os dados de
    leitura/escrita ocupam
    bool FS_Dinamico; // Define se o fator de escala (FS) da
    variável é dinâmico (true) ou estático (false).
    int16_t FS; // Armazena o fator de escala (FS), que
    representa o expoente na base 10 para o ajuste da escala. Se a escala for
    dinâmica, ele armazena o valor do registro que contém o valor do FS para o
    momento da leitura
    int8_t UNITY_ID; // Define o UNITY_ID dos dados na comunicação
    Modbus
    bool escrita; // Demonstra se o registro é passível ou não
    de escrita
    uint16_t valor[2] = { 0, 0 }; // Armazena o valor lido nos respectivos
    registro
    bool sinal; // Define se o valor lido pode ou não possuir
    sinal

    // Código da função "addReg()"
    // Função que permite adicionar na struct "dados", todas as informações das
    variáveis que envolvem a comunicação Modbus
    // Recebe como parâmetro o registro a ser armazenado (stc_registro), assim
    como a quantidade de registros que compõe o dado de leitura ou escrita
    (stc_qtd_Reg), se o fator de escala é dinâmico ou não (stc_FS_Dinamico), o
    valor efetivo do valor da escala (stc_FS), o Unity ID da comunicação
    (stc_UNITY_ID), e, por fim, se a variável é passível de escrita (stc_escrita);
    void addReg(uint16_t stc_registro, uint16_t stc_qtd_Reg, bool
    stc_FS_Dinamico, int16_t stc_FS, int8_t stc_UNITY_ID, bool stc_escrita, bool
    stc_sinal) {
        registro = stc_registro;
        qtd_Reg = stc_qtd_Reg;
        FS_Dinamico = stc_FS_Dinamico;
        FS = stc_FS;
        UNITY_ID = stc_UNITY_ID;
```

```

    escrita = stc_escrita;
    sinal = stc_sinal;
}

float correcFS() {
    int16_t escala = FS;
    float val = int16_t(valor[0]);

    if (qtd_Reg == 2) { // verifica se o dado completo está armazenado em
dois registradores
        val = int16_t(val) << 16;
        val += valor[1];
    }

    while (escala != 0) { // Realiza a correção de escala
        if (escala > 0) { // Se o FS for positivo, ocasiona na multiplicação
do valor
            val *= 10;
            escala--;
        } else { // Se ele for negativo, ocasiona na divisão do valor
            val /= 10;
            escala++;
        }
    }
    return val;
}
};

// Declaração de variáveis globais da comunicação Modbus TCP
const uint8_t TOT_REG_FR = 99; // Quantidade total de registros a serem
apresentados no supervisão, referentes ao inversor fotovoltaico FRONIUS
const uint8_t TOT_REG_VE = 169; // Quantidade total de registros a serem
apresentados no supervisão, referentes ao inversor híbrido VICTRON
bool ERRO_COMUNIC_FR = false; // Valor alterado quando há a ocorrência de
erro na comunicação Modbus entre o ESP32 e o inversor FRONIUS
bool ERRO_COMUNIC_VE = false; // Valor alterado quando há a ocorrência de
erro na comunicação Modbus entre o ESP32 e o inversor VICTRON
bool noite = false; // Variável que verifica se é dia ou noite.
Se noite == true, ignora a comunicação com o inversor fotovoltaico até que
noite == false
uint8_t cont_ciclo_FR = 0; // Variável que contabilizará quantos ciclos
serão realizados sem a comunicação com o inversor fotovoltaico. Após a
contagem desses ciclos, a comunicação com ele é realizada novamente

// Declaração de variáveis globais da comunicação Modbus RTU

```

```

const uint8_t SERVER_RTU = 1; // Endereço do ESP32 como servidor da
comunicação Modbus RTU
uint16_t lowWord, highWord; // Variáveis utilizadas na função
floatToRegisters, que converte o um float de 32 bits em dois registradores de
16 bits

// Definição do objeto de manipulação dos dados referentes ao Inversor
Fotovoltaico FRONIUS
dados Fronius[TOT_REG_FR]; // Definição da Struct que armazenará os dados de
interesse do registros do Inversor Fronius
void def_Reg_FR(); // Protótipo da função que realiza a implementação
da função da Struct "dados.addReg()

// Definição do objeto de manipulação dos dados referentes ao Inversor
Fotovoltaico VICTRON
dados Victron[TOT_REG_VE]; // Definição da Struct que armazenará os dados
referente aos registros do Inversor Victron
void def_Reg_VE(); // Protótipo da função que realiza a implementação
da função da Struct "dados.addReg()

// Protótipo da função que realiza a leitura em blocos dos dados dos
dispositivos na comunicação Modbus
// Recebe como parâmetros o endereço de IP do dispositivo de interesse (IP), o
registro inicial do bloco de leitura (reg_inicial), o registro final do bloco
de leitura (reg_final), o Unity ID do dispositivo cujo IP foi inserido (UNITY),
a struct "dados" ao qual ele se refere (s[]), e, por fim, a quantidade de
registros que serão utilizados no sistema de supervisão e controle
(qtd_reg_arm), definidos anteriormente nas variáveis TOT_REG_FR e TOT_REG_VE
void ler_mb(IPAddress IP, uint16_t reg_inicial, uint16_t reg_final, uint16_t
UNITY, struct dados s[], uint8_t qtd_reg_arm);

// Protótipo da função que realiza a escrita individual dos dados dos
dispositivos configurados na comunicação Modbus
// Recebe como parâmetros o valor a ser escrito (valor), o endereço de IP do
dispositivo de interesse (IP), o registro que contém a informação a ser
escrita (reg), o UNITY ID do dispositivo (UNITY), e a struct em que ele está
vinculado (dados s[])
void esc_mb(float valor, IPAddress IP, uint16_t reg, struct dados s[]);

// Protótipo das funções que envolvem a comunicação Modbus TCP
void def_Reg_ESP(); // Função utilizada para definir os endereços dos
registros do ESP32 a serem utilizados na comunicação com a IHM. A IHM
realizará a leitura dos registradores definidos nessa função
void att_Reg_ESP(); // Função que atualiza os registros definidos na
função def_Reg_ESP()

```

```

void floatToRegisters(); // Função que converte o um float de 32 bits em dois
registradores de 16 bits

// Definição do IP dos clientes Modbus (Inversor fotovoltaico e híbrido)
IPAddress IP_FR(192, 168, 0, 000); // IP do Inversor Fronius na rede Wi-Fi
IPAddress IP_VE(192, 168, 0, 000); // IP do Inversor Victron na rede Wi-Fi

// Definição do objeto Modbus para ser utilizado nas funções que estão
vinculadas as bibliotecas de comunicação Modbus
ModbusIP mbTCP; // Objeto para tratar da comunicação Modbus TCP
ModbusRTU mbRTU; // Objeto para tratar da comunicação Modbus RTU

// Declaração das variáveis globais da comunicação MQTT
const char* mqtt_server = "192.168.0.000"; // IP do broker para a comunicação
MQTT - BROKER LOCAL: USUÁRIO: / SENHA:
const char* usuario = "admin"; // Usuário utilizado para o login
com o Broker
const char* senhaBroker = "****"; // Senha utilizada para o login do
usuário definido anteriormente
const int MQTT_PORT = 1883; // Porta utilizada para a
comunicação MQTT
long int t1, t2 = 0; // Variáveis para comparação do
intervalo de envio dos dados ao broker
uint16_t intervalo = 3000; // Variável para definição do
intervalo em milissegundos em que os dados serão enviados ao broker

// Protótipos das funções referentes a comunicação MQTT
void reconnect_MQTT(); //
Função responsável por realizar a conexão com o broker em caso de qualquer
perda
void pub_MQTT(struct victron s[]); //
Função que realiza a publicação dos dados no broker, de acordo com a struct
que é passada na função
void callback_MQTT(char* topic, byte* payload, unsigned int length); //
Função responsável pelo callback da comunicação MQTT. Nela o ESP lê os dados
encaminhados nos tópicos em que o microcontrolador estiver inscrito

// Definição do objeto client para a comunicação MQTT. Define o
microcontrolador como cliente na rede MQTT
WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
    // Configurações para estabelecer a comunicação Modbus RTU com a IHM

```

```

Serial2.begin(9600, SERIAL_8N1, 16, 17); // Inicialização da comunicação
Serial para envio de dados a IHM. Utiliza as portas RX2 e TX2, que
correspondem aos pinos 16 e 17 respectivamente.
mbRTU.begin(&Serial2); // Define a Serial2 como as portas
a serem utilizadas na comunicação Modbus RTU
mbRTU.server(SERVER_RTU); // Configura o endereço do ESP
como servidor da rede Modbus RTU
def_Reg_ESP(); // Função utilizada para definir o
efetivo endereço de cada registrador do ESP. A IHM realizará a leitura dos
registradores definidos nessa função

// Chama a função responsável pela definição dos parâmetros e registros a
serem lidos e escritos pelo microcontrolador. Essas informações serão
armazenadas nas respectivas structs "dados"
def_Reg_FR(); // Define a base de todos os dados da comunicação Modbus com
o Inversor Fronius
def_Reg_VE(); // Define a base de todos os dados da comunicação Modbus com
o Inversor Victron

// Configuração da conexão WiFi do microcontrolador
WiFi.begin("Labfontes", "labfontes"); // Conecta na rede "Labfontes", com
a senha "labfontes"
while (WiFi.status() != WL_CONNECTED) { // Se falhar, tenta conectar na
rede a cada 0,5s
    delay(500);
}

// Faz com que o ESP32 opere como Cliente na rede Modbus TCP
mbTCP.client();

// Configuração da comunicação MQTT
client.setServer(mqtt_server, MQTT_PORT); // Realiza a conexão com o broker
que foi configurado inicialmente setServer(IP do broker, porta da comunicação
MQTT - Padrão 1883)
client.setCallback(callback_MQTT); // Configura a função callback
para realizar o monitoramento de dados nos tópicos em que o ESP for inscrito
}

int a = 1;
void loop() {
    t1 = millis();

    // Função de rotina que devem ser utilizadas periodicamente no loop para
    manter a comunicação Modbus. Devem ser chamadas sempre que se tem o interesse
    em atualizar os dados da comunicação
    mbRTU.task(); // Comunicação Modbus RTU

```

```

mbTCP.task(); // Comunicação Modbus TCP

// Loop para manter a comunicação via protocolo MQTT operando
if (!client.connected()) {
    reconnectMQTT(); // Caso o ESP se desconecte da rede MQTT, ele será
reconectado
}
client.loop(); // Função obrigatória para manter o ESP na rede MQTT
delay(10);

// Leitura dos dados dos inversores via comunicação Modbus
if (noite == false) { // A comunicação com o inversor fotovoltaico só é
estabelecida caso seja dia
    def_Reg_FR(); // Define a base de todos os dados da comunicação
Modbus com o Inversor Fronius
    // Algoritmo que verifica se houve erro na comunicação com o inversor
fotovoltaico.
    // Caso positivo, ele somente tenta comunicar novamente após 100 ciclos do
programa
    if (cont_ciclo_FR > 0) {
        cont_ciclo_FR--;
    } else {
        cont_ciclo_FR = 0;
        ler_mb(IP_FR, 40071, 40299, 1, Fronius, TOT_REG_FR); // Leitura de
todos os parâmetros do inversor Fronius
    }
}

att_Reg_ESP(); // Função utilizada para atualizar os dados a serem lidos
pela IHM

ler_mb(IP_VE, 3, 104, 228, Victron, TOT_REG_VE); // Leitura dos
parâmetros referente a rede no inversor Victron
ler_mb(IP_VE, 258, 326, 225, Victron, TOT_REG_VE); // Leitura dos
parâmetros referente a bateria no inversor Victron
ler_mb(IP_VE, 1282, 1305, 225, Victron, TOT_REG_VE); // Leitura dos
parâmetros referente a bateria no inversor Victron
ler_mb(IP_VE, 2700, 2712, 100, Victron, TOT_REG_VE); // Leitura dos
parâmetros referente a configuração de operação no inversor Victron
ler_mb(IP_VE, 2900, 2903, 100, Victron, TOT_REG_VE); // Leitura dos
parâmetros referente a configuração de operação no inversor Victron

att_Reg_ESP(); // Função utilizada para atualizar os dados a serem lidos
pela IHM

```

```

    // Publicação dos dados lidos dos inversores, no broker configurado, via
    protocolo MQTT.
    if (t1 - t2 > intervalo) {
        if (ERRO_COMUNIC_FR == false && Fronius[13].valor[0] != 32768) { // Esses
            dados somente serão enviados caso nenhum erro tenha ocorrido na comunicação
            Modbus
            pub_MQTT(Fronius); //
            Publica todos os valores contidos em cada registro configurado na struct
            "Fronius"

        } else {
            if (cont_ciclo_FR == 0) {
                cont_ciclo_FR = 100; // Incrementa a quantidade de loops
                que o ESP ficará sem se comunicar com o inversor fotovoltaico em caso de erro
                def_Reg_FR();
                pub_MQTT(Fronius);
                client.publish("FR_Erro", "1"); // Publica no tópico "FR_Erro", com o
                objetivo de forçambTCP uma mensagem do Node-RED retornando se é dia ou não.
                Também notifica do erro acontecido na interface
            }
        }

        if (ERRO_COMUNIC_VE == false) { // Esses dados somente serão enviados
            caso nenhum erro tenha ocorrido na comunicação Modbus
            pub_MQTT(Victron); // Publica todos os valores contidos em
            cada registro configurado na struct "Victron"
        } else {
            def_Reg_VE();
            client.publish("VE_Erro", "1"); // Publica no tópico "VE_Erro" para
            notificar o erro na comunicação com o inversor híbrido
        }

        t2 = t1;
    }
}

// Código da função "def_Reg_FR()"
// Definição dos registros de interesse que envolvem a comunicação Modbus com
o inversor Fronius
// Recebe como parâmetro o registro a ser armazenado (stc_registro), assim
como a quantidade de registros que compõe o dado de leitura ou escrita
(stc_qtd_Reg), se o fator de escala é dinâmico ou não (stc_FS_Dinamico), o
valor efetivo do valor da escala (stc_FS), o Unity ID da comunicação
(stc_UNITY_ID), e, por fim, se a variável é passível de escrita (stc_escrita);
void def_Reg_FR() {
    Fronius[0].addReg(40071, 1, true, 75, 1, false, false);

```

```

Fronius[1].addReg(40072, 1, true, 75, 1, false, false);
Fronius[2].addReg(40075, 1, true, 0, 1, false, true);
Fronius[3].addReg(40079, 1, false, -2, 1, false, false);
Fronius[4].addReg(40082, 1, false, -2, 1, false, true);
Fronius[5].addReg(40083, 1, true, 84, 1, false, true);
Fronius[6].addReg(40084, 1, true, 0, 1, false, true);
Fronius[7].addReg(40085, 1, false, -2, 1, false, false);
Fronius[8].addReg(40086, 1, false, -2, 1, false, true);
Fronius[9].addReg(40087, 1, true, 88, 1, false, true);
Fronius[10].addReg(40088, 1, true, 0, 1, false, true);
Fronius[11].addReg(40089, 1, true, 90, 1, false, true);
Fronius[12].addReg(40090, 1, true, 0, 1, false, true);
Fronius[13].addReg(40091, 1, false, -4, 1, false, true);
Fronius[14].addReg(40092, 1, false, -4, 1, false, true);
Fronius[15].addReg(40093, 2, true, 95, 1, false, false);
Fronius[16].addReg(40095, 1, true, 0, 1, false, true);
Fronius[17].addReg(40100, 1, true, 101, 1, false, true);
Fronius[18].addReg(40101, 1, true, 0, 1, false, true);
Fronius[19].addReg(40102, 1, false, -1, 1, false, true);
Fronius[20].addReg(40106, 1, false, -1, 1, false, true);
Fronius[21].addReg(40124, 1, false, 1, 1, false, false);
Fronius[22].addReg(40125, 1, false, 1, 1, false, true);
Fronius[23].addReg(40126, 1, false, 1, 1, false, false);
Fronius[24].addReg(40127, 1, false, 1, 1, false, true);
Fronius[25].addReg(40128, 1, false, 1, 1, false, true);
Fronius[26].addReg(40129, 1, false, 1, 1, false, true);
Fronius[27].addReg(40130, 1, false, 1, 1, false, true);
Fronius[28].addReg(40131, 1, false, 1, 1, false, true);
Fronius[29].addReg(40132, 1, false, 1, 1, false, true);
Fronius[30].addReg(40133, 1, false, 2, 1, false, false);
Fronius[31].addReg(40134, 1, false, 2, 1, false, true);
Fronius[32].addReg(40135, 1, false, -3, 1, false, true);
Fronius[33].addReg(40136, 1, false, -3, 1, false, true);
Fronius[34].addReg(40137, 1, false, -3, 1, false, true);
Fronius[35].addReg(40138, 1, false, -3, 1, false, true);
Fronius[36].addReg(40139, 1, false, -3, 1, false, true);
Fronius[37].addReg(40151, 1, false, 1, 1, false, false);
Fronius[38].addReg(40152, 1, false, 0, 1, false, false);
Fronius[39].addReg(40153, 1, false, 0, 1, false, true);
Fronius[40].addReg(40156, 1, false, 1, 1, false, false);
Fronius[41].addReg(40157, 1, false, 1, 1, false, true);
Fronius[42].addReg(40158, 1, false, 1, 1, false, true);
Fronius[43].addReg(40159, 1, false, 1, 1, false, true);
Fronius[44].addReg(40160, 1, false, 1, 1, false, true);
Fronius[45].addReg(40162, 1, false, -3, 1, false, true);
Fronius[46].addReg(40163, 1, false, -3, 1, false, true);
Fronius[47].addReg(40164, 1, false, -3, 1, false, true);
Fronius[48].addReg(40165, 1, false, -3, 1, false, true);
Fronius[49].addReg(40171, 1, false, 1, 1, false, true);

```

```

Fronius[50].addReg(40172, 1, false, 0, 1, false, true);
Fronius[51].addReg(40173, 1, false, 0, 1, false, true);
Fronius[52].addReg(40174, 1, false, 0, 1, false, true);
Fronius[53].addReg(40175, 1, false, 1, 1, false, true);
Fronius[54].addReg(40176, 1, false, 1, 1, false, true);
Fronius[55].addReg(40178, 1, false, -3, 1, false, true);
Fronius[56].addReg(40183, 1, false, 0, 1, false, false);
Fronius[57].addReg(40184, 1, false, 0, 1, false, false);
Fronius[58].addReg(40185, 1, false, 0, 1, false, false);
Fronius[59].addReg(40216, 2, false, 0, 1, false, false);
Fronius[60].addReg(40222, 2, false, 0, 1, false, false);
Fronius[61].addReg(40225, 1, true, 226, 1, false, false);
Fronius[62].addReg(40226, 1, true, 0, 1, false, true);
Fronius[63].addReg(40229, 1, false, 0, 1, true, false);
Fronius[64].addReg(40230, 1, false, 0, 1, true, false);
Fronius[65].addReg(40231, 1, false, 0, 1, true, false);
Fronius[66].addReg(40232, 1, false, -2, 1, true, false);
Fronius[67].addReg(40233, 1, false, 0, 1, true, false);
Fronius[68].addReg(40234, 1, false, 0, 1, true, false);
Fronius[69].addReg(40235, 1, false, 0, 1, true, false);
Fronius[70].addReg(40236, 1, false, 0, 1, true, false);
Fronius[71].addReg(40237, 1, false, -3, 1, true, true);
Fronius[72].addReg(40238, 1, false, 0, 1, true, false);
Fronius[73].addReg(40239, 1, false, 0, 1, true, false);
Fronius[74].addReg(40240, 1, false, 0, 1, true, false);
Fronius[75].addReg(40241, 1, false, 0, 1, true, false);
Fronius[76].addReg(40243, 1, false, 0, 1, true, true);
Fronius[77].addReg(40245, 1, false, 0, 1, true, false);
Fronius[78].addReg(40246, 1, false, 0, 1, true, false);
Fronius[79].addReg(40247, 1, false, 0, 1, true, false);
Fronius[80].addReg(40248, 1, false, 0, 1, false, false);
Fronius[81].addReg(40249, 1, false, 0, 1, true, false);
Fronius[82].addReg(40250, 1, false, -2, 1, false, true);
Fronius[83].addReg(40251, 1, false, -3, 1, false, true);
Fronius[84].addReg(40252, 1, false, 0, 1, false, true);
Fronius[85].addReg(40255, 1, true, 0, 1, false, true);
Fronius[86].addReg(40256, 1, true, 0, 1, false, true);
Fronius[87].addReg(40257, 1, true, 0, 1, false, true);
Fronius[88].addReg(40258, 1, true, 0, 1, false, true);
Fronius[89].addReg(40272, 1, true, 255, 1, false, false);
Fronius[90].addReg(40273, 1, true, 256, 1, false, false);
Fronius[91].addReg(40274, 1, true, 257, 1, false, false);
Fronius[92].addReg(40275, 2, true, 258, 1, false, false);
Fronius[93].addReg(40277, 2, false, 0, 1, false, false);
Fronius[94].addReg(40292, 1, true, 255, 1, false, false);
Fronius[95].addReg(40293, 1, true, 256, 1, false, false);
Fronius[96].addReg(40294, 1, true, 257, 1, false, false);
Fronius[97].addReg(40295, 2, true, 258, 1, false, false);
Fronius[98].addReg(40297, 2, false, 0, 1, false, false);

```

```
}
```

```
// Código da função "def_Reg_VE()"
// Definição dos registros de interesse que envolvem a comunicação Modbus com
o inversor Victron
// Recebe como parâmetro o registro a ser armazenado (stc_registro), assim
como a quantidade de registros que compõe o dado de leitura ou escrita
(stc_qtd_Reg), se o fator de escala é dinâmico ou não (stc_FS_Dinamico), o
valor efetivo do valor da escala (stc_FS), o Unity ID da comunicação
(stc_UNITY_ID), e, por fim, se a variável é passível de escrita (stc_escrita);
void def_Reg_VE() {
    Victron[0].addReg(3, 1, false, -1, 228, false, false);
    Victron[1].addReg(6, 1, false, -1, 228, false, true);
    Victron[2].addReg(9, 1, false, -2, 228, false, true);
    Victron[3].addReg(12, 1, false, 1, 228, false, true);
    Victron[4].addReg(15, 1, false, -1, 228, false, false);
    Victron[5].addReg(18, 1, false, -1, 228, false, true);
    Victron[6].addReg(21, 1, false, -2, 228, false, true);
    Victron[7].addReg(22, 1, false, -1, 228, true, true);
    Victron[8].addReg(23, 1, false, 1, 228, false, true);
    Victron[9].addReg(26, 1, false, -2, 228, false, false);
    Victron[10].addReg(27, 1, false, -1, 228, false, true);
    Victron[11].addReg(28, 1, false, 0, 228, false, false);
    Victron[12].addReg(29, 1, false, 0, 228, false, false);
    Victron[13].addReg(30, 1, false, -1, 228, true, false);
    Victron[14].addReg(31, 1, false, 0, 228, false, false);
    Victron[15].addReg(32, 1, false, 0, 228, false, false);
    Victron[16].addReg(33, 1, false, 0, 228, true, false);
    Victron[17].addReg(34, 1, false, 0, 228, false, false);
    Victron[18].addReg(35, 1, false, 0, 228, false, false);
    Victron[19].addReg(36, 1, false, 0, 228, false, false);
    Victron[20].addReg(37, 1, false, 0, 228, true, true);
    Victron[21].addReg(38, 1, false, 0, 228, true, false);
    Victron[22].addReg(39, 1, false, 0, 228, true, false);
    Victron[23].addReg(40, 1, false, 0, 228, true, true);
    Victron[24].addReg(41, 1, false, 0, 228, true, true);
    Victron[25].addReg(42, 1, false, 0, 228, false, false);
    Victron[26].addReg(43, 1, false, 0, 228, false, false);
    Victron[27].addReg(44, 1, false, 0, 228, false, false);
    Victron[28].addReg(45, 1, false, 0, 228, false, false);
    Victron[29].addReg(46, 1, false, 0, 228, false, false);
    Victron[30].addReg(47, 1, false, 0, 228, false, false);
    Victron[31].addReg(56, 1, false, 0, 228, true, false);
    Victron[32].addReg(57, 1, false, 0, 228, false, false);
    Victron[33].addReg(58, 1, false, 0, 228, false, false);
    Victron[34].addReg(59, 1, false, 0, 228, false, false);
    Victron[35].addReg(60, 1, false, 0, 228, false, false);
}
```

```

Victron[36].addReg(61, 1, false, -1, 228, false, true);
Victron[37].addReg(62, 1, false, 0, 228, true, false);
Victron[38].addReg(63, 1, false, 0, 228, false, false);
Victron[39].addReg(64, 1, false, 0, 228, false, false);
Victron[40].addReg(65, 1, false, 0, 228, true, false);
Victron[41].addReg(66, 1, false, 2, 228, true, false);
Victron[42].addReg(69, 1, false, 0, 228, false, false);
Victron[43].addReg(70, 1, false, 0, 228, false, false);
Victron[44].addReg(71, 1, false, 0, 228, true, false);
Victron[45].addReg(72, 1, false, 0, 228, true, false);
Victron[46].addReg(73, 1, false, 0, 228, false, false);
Victron[47].addReg(74, 2, false, -2, 228, false, false);
Victron[48].addReg(76, 2, false, -2, 228, false, false);
Victron[49].addReg(78, 2, false, -2, 228, false, false);
Victron[50].addReg(80, 2, false, -2, 228, false, false);
Victron[51].addReg(82, 2, false, -2, 228, false, false);
Victron[52].addReg(84, 2, false, -2, 228, false, false);
Victron[53].addReg(86, 2, false, -2, 228, false, false);
Victron[54].addReg(88, 2, false, -2, 228, false, false);
Victron[55].addReg(90, 2, false, -2, 228, false, false);
Victron[56].addReg(92, 2, false, -2, 228, false, false);
Victron[57].addReg(94, 1, false, 0, 228, false, false);
Victron[58].addReg(95, 1, false, 0, 228, false, false);
Victron[59].addReg(96, 2, false, 0, 228, true, true);
Victron[60].addReg(102, 1, false, 0, 228, true, false);
Victron[61].addReg(103, 1, false, 0, 228, true, false);
Victron[62].addReg(104, 1, false, 0, 228, false, false);
Victron[63].addReg(258, 1, false, 0, 225, false, true);
Victron[64].addReg(259, 1, false, -2, 225, false, false);
Victron[65].addReg(260, 1, false, -2, 225, false, false);
Victron[66].addReg(261, 1, false, -1, 225, false, true);
Victron[67].addReg(262, 1, false, -1, 225, false, true);
Victron[68].addReg(263, 1, false, -2, 225, false, false);
Victron[69].addReg(264, 1, false, -2, 225, false, false);
Victron[70].addReg(265, 1, false, -1, 225, false, false);
Victron[71].addReg(266, 1, false, -1, 225, false, false);
Victron[72].addReg(267, 1, false, 0, 225, false, false);
Victron[73].addReg(268, 1, false, 0, 225, false, false);
Victron[74].addReg(269, 1, false, 0, 225, false, false);
Victron[75].addReg(270, 1, false, 0, 225, false, false);
Victron[76].addReg(271, 1, false, 0, 225, false, false);
Victron[77].addReg(272, 1, false, 0, 225, false, false);
Victron[78].addReg(273, 1, false, 0, 225, false, false);
Victron[79].addReg(274, 1, false, 0, 225, false, false);
Victron[80].addReg(275, 1, false, 0, 225, false, false);
Victron[81].addReg(276, 1, false, 0, 225, false, false);
Victron[82].addReg(277, 1, false, 0, 225, false, false);
Victron[83].addReg(278, 1, false, 0, 225, false, false);
Victron[84].addReg(279, 1, false, 0, 225, false, false);

```

```

Victron[85].addReg(280, 1, false, 0, 225, true, false);
Victron[86].addReg(281, 1, false, -1, 225, false, false);
Victron[87].addReg(282, 1, false, -1, 225, false, false);
Victron[88].addReg(283, 1, false, -1, 225, false, false);
Victron[89].addReg(284, 1, false, 0, 225, false, false);
Victron[90].addReg(285, 1, false, 0, 225, false, false);
Victron[91].addReg(286, 1, false, -1, 225, false, false);
Victron[92].addReg(287, 1, false, -2, 225, false, false);
Victron[93].addReg(288, 1, false, -2, 225, false, false);
Victron[94].addReg(289, 1, false, 2, 225, false, false);
Victron[95].addReg(290, 1, false, 0, 225, false, false);
Victron[96].addReg(291, 1, false, 0, 225, false, false);
Victron[97].addReg(292, 1, false, 0, 225, false, false);
Victron[98].addReg(293, 1, false, 0, 225, false, false);
Victron[99].addReg(294, 1, false, 0, 225, false, false);
Victron[100].addReg(295, 1, false, -2, 225, false, false);
Victron[101].addReg(296, 1, false, -2, 225, false, false);
Victron[102].addReg(297, 1, false, 0, 225, false, false);
Victron[103].addReg(298, 1, false, 0, 225, false, false);
Victron[104].addReg(299, 1, false, -2, 225, false, false);
Victron[105].addReg(300, 1, false, -2, 225, false, false);
Victron[106].addReg(301, 1, false, -1, 225, false, false);
Victron[107].addReg(302, 1, false, -1, 225, false, false);
Victron[108].addReg(303, 1, false, 2, 225, false, false);
Victron[109].addReg(304, 1, false, -1, 225, false, false);
Victron[110].addReg(305, 1, false, -1, 225, false, false);
Victron[111].addReg(306, 1, false, -1, 225, false, false);
Victron[112].addReg(307, 1, false, -1, 225, false, false);
Victron[113].addReg(308, 1, false, -1, 225, false, false);
Victron[114].addReg(309, 1, false, -1, 225, false, false);
Victron[115].addReg(310, 2, false, 0, 225, false, true);
Victron[116].addReg(312, 2, false, 0, 225, false, true);
Victron[117].addReg(314, 2, false, 0, 225, false, true);
Victron[118].addReg(316, 2, false, 0, 225, false, true);
Victron[119].addReg(318, 1, false, -1, 225, false, true);
Victron[120].addReg(319, 1, false, -1, 225, false, true);
Victron[121].addReg(320, 1, false, 0, 225, false, false);
Victron[122].addReg(321, 1, false, 0, 225, false, false);
Victron[123].addReg(322, 1, false, 0, 225, false, false);
Victron[124].addReg(323, 1, false, 0, 225, false, false);
Victron[125].addReg(324, 1, false, 0, 225, false, false);
Victron[126].addReg(325, 1, false, 0, 225, false, false);
Victron[127].addReg(326, 1, false, 0, 225, false, false);
Victron[128].addReg(1282, 1, false, 0, 225, false, false);
Victron[129].addReg(1283, 1, false, 0, 225, false, false);
Victron[130].addReg(1284, 1, false, 0, 225, false, false);
Victron[131].addReg(1285, 1, false, 0, 225, false, false);
Victron[132].addReg(1286, 1, false, 0, 225, false, false);
Victron[133].addReg(1287, 1, false, 0, 225, false, false);

```

```

Victron[134].addReg(1288, 1, false, 0, 225, false, false);
Victron[135].addReg(1289, 1, false, 0, 225, false, false);
Victron[136].addReg(1290, 1, false, -2, 225, false, false);
Victron[137].addReg(1291, 1, false, -2, 225, false, false);
Victron[138].addReg(1292, 1, false, 0, 225, false, false);
Victron[139].addReg(1293, 1, false, 0, 225, false, false);
Victron[140].addReg(1294, 1, false, 0, 225, false, false);
Victron[141].addReg(1295, 1, false, 0, 225, false, false);
Victron[142].addReg(1296, 1, false, 0, 225, false, false);
Victron[143].addReg(1297, 1, false, 0, 225, false, false);
Victron[144].addReg(1298, 1, false, 0, 225, false, false);
Victron[145].addReg(1299, 1, false, 0, 225, false, false);
Victron[146].addReg(1300, 1, false, -2, 225, false, false);
Victron[147].addReg(1301, 1, false, -2, 225, false, false);
Victron[148].addReg(1302, 1, false, 0, 225, false, false);
Victron[149].addReg(1303, 1, false, 0, 225, false, false);
Victron[150].addReg(1304, 1, false, 0, 225, false, false);
Victron[151].addReg(1305, 1, false, 0, 225, false, false);
Victron[152].addReg(2700, 1, false, 0, 100, true, true);
Victron[153].addReg(2701, 1, false, 0, 100, true, false);
Victron[154].addReg(2702, 1, false, 0, 100, true, false);
Victron[155].addReg(2703, 1, false, 2, 100, true, true);
Victron[156].addReg(2704, 1, false, 1, 100, true, true);
Victron[157].addReg(2705, 1, false, 0, 100, true, true);
Victron[158].addReg(2706, 1, false, 2, 100, true, true);
Victron[159].addReg(2707, 1, false, 0, 100, true, true);
Victron[160].addReg(2708, 1, false, 0, 100, true, true);
Victron[161].addReg(2709, 1, false, 0, 100, false, true);
Victron[162].addReg(2710, 1, false, -1, 100, true, false);
Victron[163].addReg(2711, 1, false, 0, 100, true, false);
Victron[164].addReg(2712, 1, false, 0, 100, true, false);
Victron[165].addReg(2900, 1, false, 0, 100, true, false);
Victron[166].addReg(2901, 1, false, -1, 100, true, false);
Victron[167].addReg(2902, 1, false, 0, 100, true, false);
Victron[168].addReg(2903, 1, false, -1, 100, false, false);
}

// Código da função "ler_mb"
// Função que efetivamente realiza a comunicação de leitura entre o
microcontrolador e os dispositivos configurados na função "ler_mb()"
// Além disso, ela também realiza determinados ajustes, e armazena os dados de
leitura na variável "valor[2]" contida dentro da struct "dados"
void ler_mb(IPAddress IP, uint16_t reg_inicial, uint16_t reg_final, uint16_t
UNITY, struct dados s[], uint8_t qtd_reg_arm) {
    uint16_t reg_atual = reg_inicial; // Variável de auxilio que conta quantos
registros já foram lidos no intervalo compreendido entre reg_inicial e
reg_final

```

```

while (reg_atual <= reg_final) { // Mantém o loop enquanto o último
registro lido (reg_atual) ainda é inferior ao registro final (reg_final)
    uint8_t erro = 0;           // Variável que retorna a ocorrência de
erro em uma transação da comunicação Modbus
    uint8_t tent_reconex = 2;   // Variável que compara a quantidade de
tentativas a serem realizadas, buscando a reconexão com o servidor Modbus
    const uint8_t TAM = 99;     // Variável que armazena o tamanho máximo
do bloco possível de leitura na comunicação Modbus
    uint8_t qtd = 0;           // Variável de auxilio que armazena a
quantidade de registros a serem lidas no intervalo compreendido entre
reg_atual e reg_inical
    uint16_t resp[TAM];        // Vetor que armazena os dados lidos por
bloco, de maneira temporária, durante a resposta recebida pelo
microcontrolador na comunicação Modbus TCP. Estes valores são, posteriormente,
repassados a sua respectiva variável na struct

    for (int i = 0; i < TAM; i++) { // Loop para zerar todas as posições do
vetor anterior
        resp[i] = 0;
    }

    // Checagem da quantidade total de registros a serem lidos. Este valor não
pode ser maior que 99, e o intervalo não pode ocasionar em um registro
superior ao reg_final. Caso este número de registros de leitura seja maior que
a quantidade disponibilizada pelo dispositivo, há a ocorrência de erro na
comunicação.
    if (reg_atual + TAM > reg_final) { // Se a quantidade de registros atuais,
somados do maior bloco de registros possíveis da comunicação, resultar em um
valor superior ao último registro do bloco...
        qtd = reg_final - reg_atual; // ... há a delimitação de "qtd" para
realizar a leitura apenas dos dispositivos restantes
    } else {
        qtd = TAM;
    }

    // Algoritmo de leitura na comunicação MQTT
    while (erro < tent_reconex) {
        if (mbTCP.isConnected(IP))
        {
            // Checagem se a conexão
com o servidor foi estabelecida. Em caso de erro, a operação de leitura é
repetida por até 5 vezes
            uint16_t trans = mbTCP.readHreg(IP, reg_atual, &resp[0], qtd, NULL,
UNITY); // Inicia a leitura dos registradores - mbTCP.readHreg(IP do server,
Registro de leitura, Variavel que irá processar os resp de leitura quando
chamado a função .task, número de registros para ser lido, NULL, Unity ID do
server);

```

```

        while (mbTCP.isTransaction(trans))
        {
            // Checa se a transação da mensagem
            está ativa
            mbTCP.task();
            // Faz com que a resposta do server seja armazenada na variável
            definida anteriormente
            delay(10);
        }
        break;
    } else {
        mbTCP.connect(IP); // Tenta reconectar se a comunicação falhar
        erro++;
        delay(50);
    }
}

// Após a leitura dos parâmetros, é realizado a alocação dos parâmetros
recebidos na resposta do inversor (vetor resp[]), em sua respectiva struct
for (int c = 0; c < qtd_reg_arm; c++) { // Checagem dos parâmetros na
struct
    for (int i = 0; i < qtd; i++) { // Checagem da resposta
armazenada no vetor resp[]
        if (i + reg_atual > reg_final) { // Checagem se todos os registros
da structs já foram lidos
            break;
        } else if (i + reg_atual == s[c].registro) { // Checa se o valor
guardado na posição i do vetor resp[] corresponde ao seu respectivo registro,
armazenado na posição c da struct s[]
            if (s[c].qtd_Reg == 2) { // Verifica se a
resposta recebida é relativa a uma variável composta por 1 ou 2 registros
consecutivos
                s[c].valor[0] = resp[i]; // Caso possua 2
registros, eles são armazenados nas posições 0 e 1, respectivamente
                s[c].valor[1] = resp[i + 1];
            } else {
                s[c].valor[0] = resp[i]; // Se houver apenas 1 registro, o valor
é armazenado na posição 0
                s[c].valor[1] = 0;
            }
        }
    }
}
reg_atual = reg_atual + qtd + 1; // Contagem da quantidade de registros
lidos até esta parte do loop

// Algoritmo da verificação de erros.
if (IP == IP_FR) { // Erro envolvendo a comunicação com o inversor
Fronius

```

```

        if (erro >= tent_reconex) {
            ERRO_COMUNIC_FR = true; // Se ocorrer 5 erros ou mais na comunicação
            com o inversor Fronius, esses dados não serão enviados ao broker da interface
            de supervisão
        } else {
            ERRO_COMUNIC_FR = false; // Se não ocorrer erros até a 5° tentativa,
            os dados serão publicados normalmente
        }
    } else if (IP == IP_VE) { // Erro envolvendo a comunicação com o inversor
        Victron
        if (erro >= tent_reconex) {
            ERRO_COMUNIC_VE = true; // Se ocorrer 5 erros ou mais na comunicação
            com o inversor Victron, esses dados não serão enviados ao broker da interface
            de supervisão
        } else {
            ERRO_COMUNIC_VE = false; // Se não ocorrer erros até a 5° tentativa,
            os dados serão publicados normalmente
        }
    }
}

// Leitura e definição dos fatores de escala dinâmicos do Inversor Fronius
if (IP == IP_FR) { // Verifica se o IP é o
    mesmo do inversor Fronius, uma vez que a escala dinâmica só ocorre neste
    equipamento
    for (int c = 0; c < qtd_reg_arm; c++) { // Loop que varre todos
        os registros [c] armazenados na struct referente ao inversor Fronius
        if (s[c].FS_Dinamico == true && s[c].FS != 0) { // Verifica se o
            registro atual possui escala dinâmica ou não
            for (int i = 0; i < qtd_reg_arm; i++) { // Loop que varre todos
                os registros novamente [i], para buscar o registro que contém o valor dinâmico
                do fator de escala para o registro anterior [c]
                if ((s[c].FS + 40000) == s[i].registro) { // Verifica se o
                    registro atual [i], corresponde ao registro [c] que contem o valor de escala
                    dinâmica. A soma s[c].FS + 40000 corresponde a um ajuste, tendo em vista que a
                    variável FS é do tipo int de 16 bits, sendo limitada a um valor decimal entre
                    +/-32767. Assim o valor armazenado nessa variável possui um offset de -40000,
                    quando se trata de um fator de escala dinâmico.
                    s[c].FS = s[i].valor[0]; // Transfere o valor
                    contido no registro [i], para a escala do registro [c]
                }
            }
        }
    }
}
}
}
}
}

```

```

// Código da função "esc_mb()"
// Função que efetivamente realiza a comunicação de escrita entre o
microcontrolador e os dispositivos configurados na função "esc_mb()"
// Recebe como parâmetro o valor a ser inscrito (int16_t valor), o IP do
dispositivo ao qual se quer alterar o valor do registro (IPAddress IP), o
registro de interesse (uint16_t reg) e a struct ao qual estes estão
configurados (struct dados s[])
void esc_mb(float valor, IPAddress IP, uint16_t reg, struct dados s[]) {
    uint8_t tot_reg = 0; // Variável para armazenar a quantidade total de
registros configurados na struct "dados"
    uint8_t erro = 0;    // Variável que retorna a ocorrência de erro em uma
transação da comunicação Modbus

    // Definição da quantidade de registros a serem verificados, de acordo com o
IP do dispositivo
    if (IP == IP_FR) { // Adequação caso se trate do Inversor Fronius
        tot_reg = TOT_REG_FR;
    } else if (IP == IP_VE) { // Adequação caso se trate do Inversor Victron
        tot_reg = TOT_REG_VE;
    }

    // Varredura de todos os registros da struct passada em "dados", de forma a
buscar aquele referente ao registro inserido no parâmetro "reg"
    for (int i = 0; i < tot_reg; i++) {
        if (reg == s[i].registro) { // Prossegue com a escrita somente se o
registro de interesse de escrita, corresponder ao registro da struct na
posição i do vetor s[i].

            // Ajuste do valor a ser escrito, de acordo com o seu respectivo Fator
de Escala
            // O valor da escala está relacionado com o expoente em base 10. Exemplo
- FS = -2 --> 10^(-2) | FS = 3 --> 10^(3)
            int16_t escala = s[i].FS;
            while (escala != 0) {
                if (escala > 0) { // Por se tratar de um procedimento de escrita, o
processo é inverso.
                    valor /= 10; // Caso o valor da escala seja > 0, faz se
necessário dividir o valor a ser escrito por 10
                    escala--; // Após a decremenção, o processo é repetido até
que a escala atinja o valor de 0
                } else {
                    valor *= 10; // Processo inverso. escala < 0 implica na
multiplicação do valor a ser escrito por 10
                    escala++;
                }
            }
        }
    }
}

```

```

        // Escrita do parâmetro configurado
        while (erro < 5)
        {
            // Tenta
            realizar a escrita até 5 vezes em caso de falha na comunicação com o
            dispositivo Modbus
            if (mbTCP.isConnected(IP))
            {
                // Checagem se a
                conexão com o servidor foi estabelecida
                uint16_t valor2 = uint16_t(valor);
                uint16_t trans = mbTCP.writeHreg(IP, s[i].registro, valor, NULL,
s[i].UNITY_ID); // Escrita do parâmetro, já corrigido, via comunicação Modbus
                break;
            } else {
                mbTCP.connect(IP); // Tenta reconectar se a comunicação falhar
                erro++;
                delay(50);
            }
        }
    }
}
}

// Código da função "def_Reg_ESP()"
// Função utilizada para definir o efetivo endereço de cada registrador do ESP.
A IHM realizará a leitura dos registradores definidos nessa função
void def_Reg_ESP() {
    mbRTU.addIreg(0, 0); // Endereço do registrador 0 a ser utilizado na IHM
para apresentar a Potência do Inversor
    mbRTU.addIreg(1, 0); // Endereço do registrador 1 a ser utilizado na IHM
para apresentar a Potência do Inversor
    mbRTU.addIreg(2, 0); // Endereço do registrador 2 a ser utilizado na IHM
para apresentar a Potência na Rede
    mbRTU.addIreg(3, 0); // Endereço do registrador 3 a ser utilizado na IHM
para apresentar a Potência na Rede
    mbRTU.addIreg(4, 0); // Endereço do registrador 4 a ser utilizado na IHM
para apresentar a Potência nas Baterias
    mbRTU.addIreg(5, 0); // Endereço do registrador 5 a ser utilizado na IHM
para apresentar a Potência nas Baterias
    mbRTU.addIreg(6, 0); // Endereço do registrador 6 a ser utilizado na IHM
para apresentar a Potência das cargas
    mbRTU.addIreg(7, 0); // Endereço do registrador 7 a ser utilizado na IHM
para apresentar a Potência das cargas

    mbRTU.addIreg(10, 0); // Endereço do registrador 10 a ser utilizado na IHM
para apresentar a Tensão do Inversor
    mbRTU.addIreg(11, 0); // Endereço do registrador 11 a ser utilizado na IHM
para apresentar a Tensão do Inversor

```

```

    mbRTU.addIreg(12, 0); // Endereço do registrador 12 a ser utilizado na IHM
    para apresentar a Tensão na Rede
    mbRTU.addIreg(13, 0); // Endereço do registrador 13 a ser utilizado na IHM
    para apresentar a Tensão na Rede
    mbRTU.addIreg(14, 0); // Endereço do registrador 14 a ser utilizado na IHM
    para apresentar a Tensão nas Baterias
    mbRTU.addIreg(15, 0); // Endereço do registrador 15 a ser utilizado na IHM
    para apresentar a Tensão nas Baterias
    mbRTU.addIreg(16, 0); // Endereço do registrador 16 a ser utilizado na IHM
    para apresentar a Tensão das cargas
    mbRTU.addIreg(17, 0); // Endereço do registrador 17 a ser utilizado na IHM
    para apresentar a Tensão das cargas

```

```

    mbRTU.addIreg(20, 0); // Endereço do registrador 20 a ser utilizado na IHM
    para apresentar a Corrente do Inversor
    mbRTU.addIreg(21, 0); // Endereço do registrador 21 a ser utilizado na IHM
    para apresentar a Corrente do Inversor
    mbRTU.addIreg(22, 0); // Endereço do registrador 22 a ser utilizado na IHM
    para apresentar a Corrente na Rede
    mbRTU.addIreg(23, 0); // Endereço do registrador 23 a ser utilizado na IHM
    para apresentar a Corrente na Rede
    mbRTU.addIreg(24, 0); // Endereço do registrador 24 a ser utilizado na IHM
    para apresentar a Corrente nas Baterias
    mbRTU.addIreg(25, 0); // Endereço do registrador 25 a ser utilizado na IHM
    para apresentar a Corrente nas Baterias
    mbRTU.addIreg(26, 0); // Endereço do registrador 26 a ser utilizado na IHM
    para apresentar a Corrente das cargas
    mbRTU.addIreg(27, 0); // Endereço do registrador 27 a ser utilizado na IHM
    para apresentar a Corrente das cargas
}

```

```

// Código da função "att_Reg_ESP()"
// Função que atualiza os registros definidos na função def_Reg_ESP()
void att_Reg_ESP() {
    // Definição da potência de saída do inversor fotovoltaico
    floatToRegisters(Fronius[5].correcFS() * (-1), lowWord, highWord);
    mbRTU.Ireg(0, lowWord);
    mbRTU.Ireg(1, highWord);

    // Definição da potência de entrada do inversor híbrido
    floatToRegisters(Victron[3].correcFS(), lowWord, highWord);
    mbRTU.Ireg(2, lowWord);
    mbRTU.Ireg(3, highWord);

    // Definição da potência das baterias
    floatToRegisters(Victron[63].correcFS(), lowWord, highWord);

```

```

mbRTU.Ireg(4, lowWord);
mbRTU.Ireg(5, highWord);

// Definição da potência da carga
floatToRegisters(Victron[3].correcFS() - (Fronius[5].correcFS() * (-1)),
lowWord, highWord);
mbRTU.Ireg(6, lowWord);
mbRTU.Ireg(7, highWord);

// Definição da tensão de saída do inversor fotovoltaico
floatToRegisters(Fronius[3].correcFS(), lowWord, highWord);
mbRTU.Ireg(10, lowWord);
mbRTU.Ireg(11, highWord);

// Definição da tensão de entrada do inversor híbrido
floatToRegisters(Victron[0].correcFS(), lowWord, highWord);
mbRTU.Ireg(12, lowWord);
mbRTU.Ireg(13, highWord);

// Definição da tensão das baterias
floatToRegisters(Victron[64].correcFS(), lowWord, highWord);
mbRTU.Ireg(14, lowWord);
mbRTU.Ireg(15, highWord);

// Definição da tensão da carga
floatToRegisters(Victron[4].correcFS(), lowWord, highWord);
mbRTU.Ireg(16, lowWord);
mbRTU.Ireg(17, highWord);

// Definição da corrente de saída do inversor fotovoltaico
floatToRegisters(Fronius[0].correcFS() * (-1), lowWord, highWord);
mbRTU.Ireg(20, lowWord);
mbRTU.Ireg(21, highWord);

// Definição da corrente de entrada do inversor híbrido
floatToRegisters(Victron[1].correcFS(), lowWord, highWord);
mbRTU.Ireg(22, lowWord);
mbRTU.Ireg(23, highWord);

// Definição da corrente das baterias
floatToRegisters(Victron[66].correcFS(), lowWord, highWord);
mbRTU.Ireg(24, lowWord);
mbRTU.Ireg(25, highWord);

```

```

    // Definição da corrente da carga
    floatToRegisters(Victron[1].correcFS()-(Fronius[0].correcFS() * (-1)),
lowWord, highWord);
    mbRTU.Ireg(26, lowWord);
    mbRTU.Ireg(27, highWord);

    mbRTU.task();
    delay(15);
}

// Código da função "floatToRegisters()"
// Função que converte o um float de 32 bits em dois registradores de 16 bits
void floatToRegisters(float value, uint16_t &lowWord, uint16_t &highWord) {
    union {
        float f;
        uint32_t i;
    } data;
    data.f = value;
    lowWord = (uint16_t)(data.i & 0xFFFF);
    highWord = (uint16_t)(data.i >> 16);
}

// Código da função "reconnect_MQTT()"
// Função utilizada para manter a conexão do ESP32 com o broker
void reconnectMQTT() {
    while (!client.connected()) { // Tenta realizar a conexão com o broker caso
o ESP não esteja conectado
        String clientID = "ESP32_LabFontes"; // ID de cliente do ESP32
        delay(10);

        if (client.connect(clientID.c_str(), usuario, senhaBroker)) { // Realiza
a conexão no broker, com os dados configurados
            // Inscrição no tópico para verificar se é noite
            client.subscribe("noite");

            // Inscrição nos tópicos relacionados com a escrita dos parâmetros de
operação dos dispositivos da comunicação Modbus, através de comandos recebidos
do Node-Red Dashboard
            for (int i = 0; i < TOT_REG_VE; i++) { // Inscrição dos
tópicos referentes ao inversor Victron
                if (Victron[i].escrita == true) { // Primeiro checa
se o registro a ser inscrito é passível de escrita (apenas este tipo é
interessante)

```

```

        char topico_esc[10] = "VE_E_"; // Identificador
de que se trata de um tópico relacionado com a escrita de parâmetros do
Inversor VICTRON
        char reg_p_topico[5]; // Variável de
auxilio para armazenar o valor do registro convertido para char
        dtostrf(Victron[i].registro, 1, 0, reg_p_topico); // Realiza a
conversão de um dado numérico para uma string de caracteres dtostrf(valor
float a ser convertido, tamanho minimo da string de saída, N° de casas após a
vírgula, String de saída). Neste caso, converte o registro do tipo uint para
char
        strcat(topico_esc, reg_p_topico); // Define o
tópico referente, exclusivamente, a escrita dos registros do inversor VICTRON
        client.subscribe(topico_esc); // Depois realiza
a inscrição nos tópicos referentes a escrita do VICTRON
    }
}

// Inscrição nos tópicos relacionados com a alteração dos parâmetros de
operação do Fronius, através de comandos no Node-Red Dashboard
for (int i = 0; i < TOT_REG_FR; i++) { // Realiza a mesma operação que
a função anterior, porém em relação ao inversor Fronius
    if (Fronius[i].escrita == true) {
        char topico_esc[11] = "FR_E_";
        char reg_p_topico[6];
        dtostrf(Fronius[i].registro, 1, 0, reg_p_topico);
        strcat(topico_esc, reg_p_topico);
        client.subscribe(topico_esc);
    }
}

} else { // Em caso de erro, tenta reconectar novamente
    delay(2000);
}
}
}

// Código da função "pub_MQTT()"
// Função responsável por publicar os valores lidos na comunicação Modbus, no
broker configurado na rede MQTT
// Recebe como parâmetro apenas a struct do dispositivo que se deseja publicar
os parâmetros lidos
void pub_MQTT(struct dados s[]) {
    // Verificação da quantidade total de registros armazenados na struct que
foi passada
    uint8_t tot_reg = 0;
    if (s == Fronius) {

```

```

    tot_reg = TOT_REG_FR; // Total de registros relacionados ao inversor
    Fronius
} else {
    tot_reg = TOT_REG_VE; // Total de registros relacionados ao inversor
    Victron
}

// Algoritmo para escrita via Modbus
for (int i = 0; i < tot_reg; i++) {
    char reg_p_topico[6] = ""; // Variável que irá armazenar o valor
    convertido dos registros do tipo uint para char, de forma a ser usado como
    tópico na comunicação MQTT
    char topico[11] = ""; // Variável que armazenará o valor do topico
    de escrita. Exemplo - topicoE[11] = "FR_E_40071" (FR - Inversor Fronius; E -
    Escrita na comunicação Modbus; 40071 - Registro de interesse)

    // Verificação através do nome da struct para verificar de qual
    dispositivo se trata
    if (s == Fronius) {
        strcat(topico, "FR_L_"); // Armazena o valor FR_L nas primeiras
        posições do array topico[11]. Estes caracteres indicam que se tratam de uma
        mensagem vinculada ao inversor FRonius, sendo uma operação de Leitura dos
        dados deste dispositivo. O restante da mensagem a ser publicada será
        concatenada abaixo.
    } else {
        strcat(topico, "VE_L_"); // Armazena o valor VE_L nas primeiras
        posições do array topico[11]. Estes caracteres indicam que se tratam de uma
        mensagem vinculada ao inversor Victron Energy, sendo uma operação de Leitura
        dos dados deste dispositivo. O restante da mensagem a ser publicada será
        concatenada abaixo.
    }

    // Definição do tópico da mensagem a ser encaminhada, de acordo com o
    registro atual.
    dtostrf(s[i].registro, 1, 0, reg_p_topico); // Realiza a conversão de um
    dado numérico para uma string de caracteres dtostrf(valor float a ser
    convertido, tamanho minimo da string de saída, N° de casas após a virgula,
    String de saída). Neste caso, converte o valor do registro para uma string
    strcat(topico, reg_p_topico); // Concatena a segunda string
    com a primeira, e armazena o valor final nesta. Assim o topico da mensagem
    fica completo conforme o exemplo anterior ("FR_E_40071")

    char mensagem[15] = ""; // Variável utilizada para armazenar o conteúdo
    da mensagem a ser enviada para o broker. Esta mensagem será do tipo 000000#-0
    (valor_lido#fator_de_escalade_valor)

```

```

    char fs[2] = "";          // Variável utilizada para armazenar o fator de
                              escala da mensagem a ser enviada para o broker. Será concatenada com a
                              variável anterior

    // Conversão do valor a ser escrito de acordo com a quantidade de
    registros que este parâmetro possui
    if (s[i].qtd_Reg == 2) {                                     // Verifica se o
        valor a ser enviado possui mais de um registro armazenado. Se houver, realiza
        a junção de s[i].valor[0]+s[i].valor[1]
        if (s[i].sinal == true) {                               // Verifica se há
            a necessidade de converter o valor lido para um valor com sinal
            int32_t valor_sinal = int32_t(s[i].valor[0]) << 16; // Desloca o
            s[i].valor[0] 16 bits para a esquerda, e o armazena em uma variável de 32 bits
            com sinal, resultando em algo parecido: 11111111111111110000000000000000
            valor_sinal += (s[i].valor[1]);                     // Concatena o
            restante dos bits nas primeiras posições da variável de 32 bits com sinal.
            Resultado final: 11111111111111111111111111111111
            dtostrf(valor_sinal, 1, 0, mensagem);               // Converte este
            valor do tipo int32 para char, de forma a permitir encaminhá-lo ao broker. Em
            seguida, o armazena na variável "mensagem"
            dtostrf(s[i].FS, 1, 0, fs);                         // Realiza a
            conversão para texto do fator de escala da variável, e o armazena na variável
            "fs"
            strcat(mensagem, "#");                               // Concatena o
            identificador de separador do valor real e do fator de escala, com o valor a
            ser escrito, que já se encontra armazenado em "mensagem". Resulta em: 000000#
            strcat(mensagem, fs);                                // Concatena o
            valor convertido do FS com o restante da mensagem. Resulta em: 000000#-0
        } else {                                                 // Caso não tenha
            sinal, realiza as mesmas configurações anteriores, porém para uma variável
            uint32
            uint32_t valor_sinal = uint32_t(s[i].valor[0]) << 16;
            valor_sinal += (s[i].valor[1]);
            dtostrf(valor_sinal, 1, 0, mensagem);
            dtostrf(s[i].FS, 1, 0, fs);
            strcat(mensagem, "#");
            strcat(mensagem, fs);
        }
    } else { // Realiza as mesmas configurações anteriores, porém para uma
        variável com apenas 1 registro
        if (s[i].sinal == true) {
            int16_t valor_sinal = int(s[i].valor[0]);
            dtostrf(valor_sinal, 1, 0, mensagem);
            dtostrf(s[i].FS, 1, 0, fs);
            strcat(mensagem, "#");
            strcat(mensagem, fs);
        } else {
            dtostrf(s[i].valor[0], 1, 0, mensagem);

```

```

        dtostrf(s[i].FS, 1, 0, fs);
        strcat(mensagem, "#");
        strcat(mensagem, fs);
    }
}

// Publicação no broker
bool erro_pub = false;
erro_pub = client.publish(topico, mensagem); // Publica o tópico
especificado e o valor convertido da mensagem
while (erro_pub == false) { // Se houver erro na publicação, tenta
novamente
    client.disconnect();
    reconnectMQTT();
    erro_pub = client.publish(topico, mensagem); // Publica o tópico
especificado e o valor convertido da mensagem
}
}
}

// Código para a Função "callback_MQTT()"
// Função responsável pelo callback na comunicação MQTT. Utilizada sempre
quando há a publicação em um tópico em que o microcontrolador se encontra
inscrito
void callback_MQTT(char* topic, byte* payload, unsigned int length) {
    String topico = topic; // Armazena o tópico da mensagem recebida na
variável especificada

    // Faz a varredura da mensagem, e armazena seu conteúdo completo na variável
msg
    String msg = ""; // Variável que irá armazenar o conteúdo efetivo da
mensagem recebido no tópico acima
    for (int i = 0; i < length; i++) {
        msg += (char)payload[i];
    }

    if (topico == "noite") {
        if (msg == "true") {
            noite = true;
        } else {
            noite = false;
        }
    } else {
        // Verificação se a mensagem recebida está relacionada com o inversor
Fronius ou Victron. Se tratam de parâmetros necessários para configurar a
escrita na comunicação Modbus

```

```

uint8_t tot_reg = 0; // Variável de auxilio que
armazenará a quantidade de registros que está configurada na struct do seu
respectivo inversor. Utilizado para auxiliar na varredura dos registros
struct dados* s; // Ponteiro utilizado para
apontar para a struct que está relacionada com a mensagem recebida do broker
IPAddress IP; // Variável de auxilio para
armazenar o IP referente ao dispositivo ao qual se quer realizar a escrita da
mensagem recebida.
const char* topico_char = topico.c_str(); // Conversão da mensagem
recebida no tipo String, para o tipo const char
if (strncmp(topico_char, "FR_E", 4) == 0) { // Verifica se as 4 primeiras
letras do tópico referente a mensagem recebida do broker, são idênticas a
"FR_E". Caso positivo, indica que se trata de uma operação de Escrita no
inversor Fronius
    tot_reg = TOT_REG_FR; // Caso a condição anterior
tenha sido satisfeita, armazena os valores referente a configuração deste
inversor
    s = Fronius;
    IP = IP_FR;
} else if (strncmp(topico_char, "VE_E", 4) == 0) { // Verifica se as 4
primeiras letras do tópico referente a mensagem recebida do broker, são
idênticas a "VE_E". Caso positivo, indica que se trata de uma operação de
Escrita no inversor Victron Energy
    tot_reg = TOT_REG_VE; // Caso a condição
anterior tenha sido satisfeita, armazena os valores referente a configuração
deste inversor
    s = Victron;
    IP = IP_VE;
}

// FUNÇÃO RELACIONADA COM A COMUNICAÇÃO MODBUS
// Algoritmo para escrita via Modbus, no inversor relacionado ao parâmetro
definido no tópico da mensagem recebida na comunicação MQTT
for (int i = 0; i < tot_reg; i++) {
    if (s[i].escrita == true) { // Primeiro checa se o parâmetro é
passível de escrita
        char reg_p_topico[6] = ""; // Variável de auxílio que irá armazenar o
valor convertido dos registros da comunicação Modbus, para ser utilizado como
tópico na comunicação MQTT
        char topicoE[11] = ""; // Variável que armazenará o valor do
topico de escrita. Exemplo - topicoE[11] = "FR_E_40071" (FR - Inversor Fronius;
E - Escrita na comunicação Modbus; 40071 - Registro de interesse)

        // Verificação através do nome da struct para verificar de qual
dispositivo se trata
        if (s == Fronius) {
            strcat(topicoE, "FR_E_"); // Armazena o valor FR_E nas primeiras
posições do array topicoE[11]. O restante da mensagem será concatenada abaixo

```

```

    } else {
        strcat(topicoE, "VE_E_"); // Armazena o valor VE_E nas primeiras
posições do array topicoE[11]. O restante da mensagem será concatenada abaixo
    }

    // Conversão do valor do registro do tipo uint para char
    dtostrf(s[i].registro, 1, 0, reg_p_topico); // Realiza a conversão de
um dado do tipo numérico para uma string de caracteres - dtostrf(valor float a
ser convertido, tamanho minimo da string de saída, N° de casas após a vírgula,
String de saída)
    strcat(topicoE, reg_p_topico); // Concatena a segunda
string com a primeira, e armazena o valor final nesta. Neste caso, concatena o
valor do registro com o parâmetro "FR_E" ou "VE_E"
    if (String(topicoE) == topico) { // Depois checa se o
tópico da mensagem recebida é compatível com o nome do registro da comunicação
Modbus, que foi convertido para uma String
        float valor = float(atoi(msg.c_str())); // Se o tópico da
mensagem for referente ao registro atual, realiza-se a conversão do valor
recebido na variável "msg" como uma string de caracteres, para um valor do
tipo numerico através da função atoi()
        esc_mb(valor, IP, s[i].registro, s);
    }
}
}
}
}
}

```

APÊNDICE B - FUNÇÃO PARA CORREÇÃO DE FATOR DE ESCALA NO NODE-RED

```
// Correção da mensagem considerando o fator de escala
let array = msg.payload.split("#");
let valor = parseFloat(array[0]);
let FS = array[1];
if (FS > 0) {
    while (FS != 0) {
        valor *= 10;
        FS--;
    }
} else if (FS < 0) {
    while (FS != 0) {
        valor /= 10;
        FS++;
    }
}
msg.payload = Number(valor.toFixed(2));
return msg;
```

APÊNDICE C - CÓDIGOS PARA OS NÓS DE FUNÇÃO DA INTERFACE DE CONTROLE DA LIMITAÇÃO DE POTÊNCIA ATIVA DO INVERSOR FOTOVOLTAICO

```
// Código do nó de função "Bloqueio form"
```

```
msg.payload = false;
```

```
return msg;
```

```
// Código do nó de função "Liberação do form"
```

```
if(msg.payload == true){
```

```
    msg.enabled = true;
```

```
    msg.payload = {
```

```
        "Inicio_P_Limit": 0,
```

```
        "Fim_P_Limit": 0,
```

```
        "Rampa_P_Limit": 0,
```

```
        "P_Limit": 3000
```

```
    };
```

```
    return [msg, null];
```

```
}else{
```

```
    msg.payload = {
```

```
        "Inicio_P_Limit": 0,
```

```
        "Fim_P_Limit": 0,
```

```
        "Rampa_P_Limit": 0,
```

```
        "P_Limit": 3000
```

```
    };
```

```
    msg.enabled = false;
```

```
    return [msg, msg];
```

```
}
```

```
// Código do nó de função "Validação"
```

```
// Objetos JSON para o envio separado de cada valor contido no msg.payload
```

```
let Notificacao = { "payload": "" };
```

```
let Inicio_P_Limit = { "payload": msg.payload["Inicio_P_Limit"] };
```

```
let P_Limit = { "payload": msg.payload["P_Limit"] };
```

```
let Fim_P_Limit = { "payload": msg.payload["Fim_P_Limit"] };
```

```
let Rampa_P_Limit = { "payload": msg.payload["Rampa_P_Limit"] };
```

```
// Verifica se os valores inseridos estão de acordo com o esperado
```

```
if ((Inicio_P_Limit.payload <= 300 && Inicio_P_Limit.payload >= 0) &&
```

```
(Rampa_P_Limit.payload >= 0 && Rampa_P_Limit.payload <= 65534) &&
```

```
(Fim_P_Limit.payload <= 28800 && Fim_P_Limit.payload >= 0) && (P_Limit.payload
```

```
<= 3000 && P_Limit.payload >= 0)) {
```

```
    Notificacao.payload = "Configuração enviada";
```

```
    return [Notificacao, msg, Inicio_P_Limit, Rampa_P_Limit, Fim_P_Limit,
```

```
P_Limit];
```

```
} else {
```

```
    Notificacao.payload = "Erro nos valores inseridos"
    Notificacao.ok = "OK"
    return [Notificacao, msg, null, null, null, null];
}

// Código do nó de função "Adequação"
// Conversão de W para %
msg.payload = Number(((msg.payload/3000)*100).toFixed(2));
return msg;

// Código do nó de função "Liberação de escrita"
msg.payload = 1;
return msg;
```

APÊNDICE D - CÓDIGOS PARA OS NÓS DE FUNÇÃO DA INTERFACE DE CONTROLE DO MODO DE OPERAÇÃO DO INVERSOR HÍBRIDO

```
// Código da função "Confere msg"
// Algoritmo que verifica se a msg atual é igual a msg anterior
// Caso seja, a msg não é repassada
let v1 = context.get("valor_anterior");
let v2 = msg.payload;
if(v1 != v2){
    context.set("valor_anterior", v2);
    return msg;
}

// Código da função "Formatação Aviso"
msg.payload = "Configuração enviada";
msg.topic = null;
return msg;

// Código da função "Converte texto"
// Apresenta o texto explicativo de acordo com o modo de operação configurado
if(msg.payload == 1){
    msg.payload = "STATUS APENAS CARREGADOR - Neste modo, a energia das baterias nunca será utilizada.";
}
else if(msg.payload == 2){
    msg.payload = "STATUS ILHADO - Neste modo, o sistema opera em modo ilhado.";
}
else if(msg.payload == 3){
    msg.payload = "STATUS LIGADO - Neste modo, o sistema opera sem qualquer limitação.";
}
else if(msg.payload == 4){
    msg.payload = "STATUS DESLIGADO - Neste modo, é enviado o sinal de desligamento para o inversor.";
}
return msg;
```

APÊNDICE E - CÓDIGOS PARA OS NÓS DE FUNÇÃO DA INTERFACE DE PROGRAMAÇÃO DE INJEÇÃO DO INVERSOR FOTOVOLTAICO

```
// Código para a função "Bloqueio form"
msg.payload = false;
return msg;
```

```
// Código para a função "Liberação do form 2"
//Liberação do formulário de acordo com o switch
if (msg.payload == true) {
    msg.enabled = true;
} else {
    flow.set("Lib_P_Limit_Prog", false);
    msg.enabled = false;
}
msg.payload = {
    "Inicio_P_Limit": 0,
    "Fim_P_Limit": 0,
    "Rampa_P_Limit": 0,
    "P_Limit": 3000,
    "T_Inicio": 3 * 60 * 60 * 1000, // Inicia os campos reservados ao
    intervalo com a definição de 3h, que corresponde a 0h por conta do UTC (é
    retirado -3 ao enviar para o form)
    "T_Fim": 3 * 60 * 60 * 1000
};
return msg;
```

```
// Código para a função "Validação"
// Verifica se os valores inseridos estão de acordo com o esperado
let Notificacao = { "payload": "" };
let Inicio_P_Limit = { "payload": msg.payload["Inicio_P_Limit"] };
let P_Limit = { "payload": msg.payload["P_Limit"] };
let Fim_P_Limit = { "payload": msg.payload["Fim_P_Limit"] };
let Rampa_P_Limit = { "payload": msg.payload["Rampa_P_Limit"] };
msg.payload["Config"] = "Configuração de Limitação de Potência do Inversor";
flow.set("Prog_P_Limit", msg.payload);
if ((Inicio_P_Limit.payload <= 300 && Inicio_P_Limit.payload >= 0) &&
(Rampa_P_Limit.payload >= 0 && Rampa_P_Limit.payload <= 65534) &&
(Fim_P_Limit.payload <= 28800 && Fim_P_Limit.payload >= 0) && (P_Limit.payload
<= 3000 && P_Limit.payload >= 0)) {
    Notificacao.payload = "Configuração salva";
    flow.set("Lib_P_Limit_Prog", true);
} else {
    Notificacao.payload = "O valor inserido deve estar entre -3000 e +3000
W"
```

```

}
return [Notificacao,msg];

// Código para a função "Verificação do horário"
// Realiza a configuração dos parâmetros a serem configurados no inversor
durante o intervalo programado
let Inicio_P_Limit_Anterior = global.get("FR_L_40233");
let P_Limit_Anterior = global.get("FR_L_40232");
let Fim_P_Limit_Anterior = global.get("FR_L_40234");
let Rampa_P_Limit_Anterior = global.get("FR_L_40235");
let Lib_P_Limit_Prog = flow.get("Lib_P_Limit_Prog"); // Variável que verifica
se a programação de despacho foi ativada, e se o valor já foi confirmado no
form
msg.payload = flow.get("Prog_P_Limit"); // Recebe todos os parâmetros
configurados na programação de despacho. Eles estão formatados como JSON no
mesmo modo que são utilizados no Form do Node-RED
if (Lib_P_Limit_Prog == true) { // Se o despacho programado foi ativado,
realiza a leitura dos parâmetros que foram configurados no seu respectivo form
    let Inicio_P_Limit = msg.payload["Inicio_P_Limit"];
    let P_Limit = Number(((msg.payload["P_Limit"])/3000) * 100).toFixed(2));
    let Fim_P_Limit = msg.payload["Fim_P_Limit"];
    let Rampa_P_Limit = msg.payload["Rampa_P_Limit"];
    let T_Inicio = new Date(msg.payload["T_Inicio"]);
    let T_Fim = new Date(msg.payload["T_Fim"]);
    let agr = new Date();
    T_Inicio = new Date(agr.getFullYear(), agr.getMonth(), agr.getDate(),
T_Inicio.getHours(), T_Inicio.getMinutes()); // Realiza a normalização do
intervalo definido, tendo como base a data atual
    T_Fim = new Date(agr.getFullYear(), agr.getMonth(), agr.getDate(),
T_Fim.getHours(), T_Fim.getMinutes());
    if (T_Fim.getTime() <= T_Inicio.getTime()) { // Adição de um dia caso o
tempo de término seja menor que o tempo de início
        T_Fim = new Date(T_Fim.getTime() + 24 * 60 * 60 * 1000);
    }
    // Verificação se este momento atual está dentro do intervalo configurado
    if (agr.getTime() >= T_Inicio.getTime() && agr.getTime() < T_Fim.getTime())
{
        if (Inicio_P_Limit_Anterior != Inicio_P_Limit ||
Number(P_Limit_Anterior) != P_Limit || Fim_P_Limit_Anterior != Fim_P_Limit ||
Rampa_P_Limit_Anterior != Rampa_P_Limit) { // Permite a escrita caso a mesma
configuração já não esteja ativada, e já esteja no intervalo esperado para
realização da configuração
            global.set("FR_L_40233", Inicio_P_Limit);
            global.set("FR_L_40232", P_Limit);
            global.set("FR_L_40234", Fim_P_Limit);
            global.set("FR_L_40235", Rampa_P_Limit);

```

```

        // Formatação das mensagens em 3 objetos JSON distintos, cada um
        contendo o "payload" como propriedade, devido a necessidade dessa formatação
        no bloco MQTT
        let msg1 = { "payload": "Configuração Iniciada" };
        let msg2 = { "payload": Inicio_P_Limit };
        let msg3 = { "payload": P_Limit };
        let msg4 = { "payload": Fim_P_Limit };
        let msg5 = { "payload": Rampa_P_Limit };
        let msg6 = { "payload": 1 };
        return [msg1, msg2, msg3, msg4, msg5, msg6];
    }
}
else { // caso o intervalo programado acabe, retorna para as
configurações padrões de fábrica (modo back-up, com P_Entrada de 30W)
    if (Inicio_P_Limit_Anterior != 0 || Number(P_Limit_Anterior) != 100 ||
Fim_P_Limit_Anterior != 0 || Rampa_P_Limit_Anterior != 0) {
        Inicio_P_Limit = 0;
        P_Limit = 100;
        Fim_P_Limit = 0;
        Rampa_P_Limit = 0;
        global.set("FR_L_40233", Inicio_P_Limit);
        global.set("FR_L_40232", P_Limit);
        global.set("FR_L_40234", Fim_P_Limit);
        global.set("FR_L_40235", Rampa_P_Limit);
        // Formatação das mensagens em 3 objetos JSON distintos, cada um
        contendo o "payload" como propriedade, devido a necessidade dessa formatação
        no bloco MQTT
        let msg1 = { "payload": "Configuração Finalizada" };
        let msg2 = { "payload": Inicio_P_Limit };
        let msg3 = { "payload": P_Limit };
        let msg4 = { "payload": Fim_P_Limit };
        let msg5 = { "payload": Rampa_P_Limit };
        let msg6 = { "payload": 0 };
        return [msg1, msg2, msg3, msg4, msg5, msg6];
    }
}
}
else { // Caso o botão de programação seja desconectado, retorna para as
configurações padrões de fábrica (modo back-up, com P_Entrada de 30W)
    if (Inicio_P_Limit_Anterior != 0 || Number(P_Limit_Anterior) != 100 ||
Fim_P_Limit_Anterior != 0 || Rampa_P_Limit_Anterior != 0) {
        let Inicio_P_Limit = 0;
        let P_Limit = 100;
        let Fim_P_Limit = 0;
        let Rampa_P_Limit = 0;
        global.set("FR_L_40233", Inicio_P_Limit);
        global.set("FR_L_40232", P_Limit);
        global.set("FR_L_40234", Fim_P_Limit);
        global.set("FR_L_40235", Rampa_P_Limit);
    }
}

```

// Formatação das mensagens em 3 objetos JSON distintos, cada um contendo o "payload" como propriedade, devido a necessidade dessa formatação no bloco MQTT

```
let msg1 = { "payload": "Configuração Desativada" };
let msg2 = { "payload": Inicio_P_Limit };
let msg3 = { "payload": P_Limit};
let msg4 = { "payload": Fim_P_Limit };
let msg5 = { "payload": Rampa_P_Limit };
let msg6 = { "payload": 0 };
return [msg1, msg2, msg3, msg4, msg5, msg6];
}
}
```

APÊNDICE F - CÓDIGOS PARA OS NÓS DE FUNÇÃO DA INTERFACE DE REGISTRO DE INFORMAÇÕES NO BANCO DE DADOS MYSQL

```
// Código da função "set 'intervalo'"
flow.set("intervalo",15*60*1000)// Configura o intervalo de registro dos dados
no BD - em milissegundos
// Código da função "Query para inserir no BD"
// Envia a query para o DB de acordo com o valor definido em "t1"
let t1 = context.get("t1");
let t2 = new Date().getTime();
let intervalo = flow.get("intervalo");
let valArray = context.get("valArray"); // Array que irá armazenar os
valores até o limite definido para o intervalo. Posteriormente será realizado
a média desses valores
let qtdItens = valArray.push(msg.payload); // Armazena a mensagem atual na
última posição do array
if((t2-t1) >= intervalo){ // Query é enviada a cada intervalo definido
em "t1"
    //Código para realizar a média dos valores enviados no intervalo de tempo
configurado
    let soma = 0;
    for(let i=0;i<qtdItens;i++){
        soma = soma + valArray[i];
    }
    let valor = Number(soma/qtdItens).toFixed(2); // Valor lido que será
copiado para o BD
    let topico = msg.topic; // Nome a ser inserido no BD
    msg.topic = 'INSERT INTO `inversor_fr` (`topico`,`valor`) VALUES
(''+topico+'', ''+valor+'')';
    context.set("t1",t2); // Atualiza o tempo do último envio
    context.set("valArray",[]) // Limpa os valores armazenados anteriormente,
e que já foram encaminhados ao DB
    return msg;
}
```

APÊNDICE G - CÓDIGOS PARA OS NÓS DE FUNÇÃO DA INTERFACE DE CONSULTA HISTÓRICA NO BANCO DE DADOS MYSQL

```
// Código da função "set topicos"
let topicos = msg.payload;
flow.set("topicos1",topicos);
return msg;
```

```
// Código da função "set t_inicio"
// Algoritmo para armazenar o inicio do intervalo a ser apresentado no gráfico
let t_inicio = msg.payload/1000.0; // Converte para o padrão em milissegundos,
e adiciona a compensação do fuso-horário, devido a incompatibilidade na
apresentação dos gráficos
t_inicio = t_inicio + 3*60*60;

flow.set("t_inicio1",t_inicio);

return msg;
```

```
// Código da função "t_fim"
// Algoritmo para armazenar o inicio do intervalo a ser apresentado no gráfico
let t_fim = msg.payload/1000.0 + 3*60*60; // Converte para o padrão em
milissegundos, e adiciona a compensação do fuso-horário, devido a
incompatibilidade na apresentação dos gráficos
flow.set("t_fim1",t_fim);
return msg;
```

```
// Código da função "select query"
// Ajuste da consulta ao banco de dados
let payload = flow.get("topicos1");
let topicos = [];

for(let i = 0; i<payload.length; i++){
    if(i == payload.length - 1){
        topicos += payload[i]; // Ajuste da última posição dos tópicos
    }
    else{
        topicos += payload[i] + ","; // Concatenação dos tópicos para
realizar apenas uma consulta
    }
}
```

```
msg.topic = 'SELECT * FROM `inversor_fr` WHERE `topico` IN ('' + topicos + '')
AND `timestamp` BETWEEN from_unixtime(' + flow.get("t_inicio1") + ') AND
from_unixtime(' + flow.get("t_fim1") + ') UNION ALL SELECT * FROM
`inversor_ve` WHERE `topico` IN ('' + topicos + '') AND `timestamp` BETWEEN
from_unixtime(' + flow.get("t_inicio1") + ') AND from_unixtime(' +
flow.get("t_fim1") + ')';
```

```
// Formatação dos tópicos de acordo com o esperado para a apresentação no
gráfico
```

```
topicos = topicos.split(',');
```

```
flow.set("topicos1",topicos);
```

```
return msg;
```

```
// Código da função "formatação para gráfico"
```

```
// Código para formatar os dados lidos do BD, de forma que eles possam ser
lidos pelo gráfico do dashboard
```

```
let payload = msg.payload;
```

```
let dados1 = [];
```

```
let dados2 = [];
```

```
let topicos = flow.get("topicos1");
```

```
if (payload.length == 0) { // Verifica se há dados a serem apresentados no
intervalo definido anteriormente
```

```
    msg.payload = []; // Se não houver dados, é acionado o comando para
limpeza dos dados anteriores
```

```
}
```

```
else { // Se houver qualquer dado a ser apresentado, o código abaixo é
utilizado para formatação e apresentação dos dados
```

```
    for (let c = 0; c < topicos.length; c++) { // Laço que verifica os
dados filtrados pelo usuário
```

```
        for (let i = 0; i < payload.length; i++) { // Laço que verifica os
dados recebidos do Banco de Dados
```

```
            if (payload[i] == "") {
```

```
                break;
```

```
            }
```

```
            if (topicos[c] == payload[i].topico) { // Comparação entre os
topicos para permitir a divisão adequada dos dados
```

```
                let d = {};
```

```
                let timestamp = new Date(msg.payload[i].timestamp); //
```

```
Leitura/adequação do parâmetro de timestamp para cada valor lido do banco de
dados
```

```

        d = { "x": (timestamp[Symbol.toPrimitive]("number")), "y":
msg.payload[i].valor }; // Formatação padrão para apresentação do gráfico
(Coordenada x e y)
        dados1.push(d); // Array que armazena temporariamente os dados
correspondentes aos ao mesmo tópico
    }
}
dados2.push(dados1); // Separação dos dados de acordo com os
tópicos
dados1 = []; // Limpeza do array para a próxima interação

// Switch para ajuste do tópico, de forma que fique mais apresentável
na legenda
switch (topicos[c]) {
    case "FR_L_40071":
        topicos[c] = "FV - Corrente CA";
        break;
    case "FR_L_40087":
        topicos[c] = "FV - Potência CA (VA)";
        break;
    case "FR_L_40089":
        topicos[c] = "FV - Potência CA (VAr)";
        break;
    case "FR_L_40083":
        topicos[c] = "FV - Potência CA (W)";
        break;
    case "FR_L_40100":
        topicos[c] = "FV - Potência CC";
        break;
    case "FR_L_40079":
        topicos[c] = "FV - Tensão CA";
        break;
    case "FR_L_40272":
        topicos[c] = "MPPT1 - Corrente";
        break;
    case "FR_L_40274":
        topicos[c] = "MPPT1 - Potência";
        break;
    case "FR_L_40273":
        topicos[c] = "MPPT1 - Tensão";
        break;
    case "FR_L_40292":
        topicos[c] = "MPPT2 - Corrente";
        break;
    case "FR_L_40294":
        topicos[c] = "MPPT2 - Potência";
        break;
    case "FR_L_40293":

```

```

        topicos[c] = "MPPT2 - Tensão ";
        break;
        case "VE_L_261":
            topicos[c] = "BATERIA - Corrente";
            break;
        case "VE_L_258":
            topicos[c] = "BATERIA - Potência";
            break;
        case "VE_L_266":
            topicos[c] = "BATERIA - SOC";
            break;
        case "VE_L_259":
            topicos[c] = "BATERIA - Tensão";
            break;
        case "Carga_I":
            topicos[c] = "CARGAS - Corrente";
            break;
        case "Carga_W":
            topicos[c] = "CARGAS - Potência";
            break;
        case "VE_L_15":
            topicos[c] = "CARGAS - Tensão";
            break;
        case "VE_L_6":
            topicos[c] = "REDE - Corrente";
            break;
        case "VE_L_12":
            topicos[c] = "REDE - Potência";
            break;
        case "VE_L_3":
            topicos[c] = "REDE - Tensão";
            break;
    }
}

// Formatação dos dados a serem apresentados no gráfico
msg.payload = [{
    "series": topicos, // Tópicos/legendas
    "data": dados2,    // Dados
    "labels": [""],    // Categorias
}]
}

return msg;

```