UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

EDUARDO LIMA DA COSTA

SISTEMA SUPERVISÓRIO PARA DISPOSITIVOS MODBUS TCP/IP VIA APLICAÇÃO *WEB*

Recife
2022

EDUARDO LIMA DA COSTA

SISTEMA SUPERVISÓRIO PARA DISPOSITIVOS MODBUS TCP/IP VIA APLICAÇÃO WEB

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em engenharia de controle e automação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Bacharel em engenharia de controle e automação.

Orientador(a): Prof. Dr. Douglas Contente Pimentel Barbosa

Recife

2022

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Costa, Eduardo Lima da.
Sistema supervisório para dispositivos modbus tcp/ip via aplicação web / Eduardo Lima da Costa. - Recife, 2022.
62 p. : il.

Orientador(a): Douglas Contente Pimentel Barbosa
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia de Controle e Automação - Bacharelado, 2022.
Inclui referências, apêndices.

1. Protocolos de comunicação . 2. modbus. 3. tcp/ip. 4. sistemas supervisórios. 5. aplicação web. I. Barbosa, Douglas Contente Pimentel. (Orientação). II. Título.

620 CDD (22.ed.)

EDUARDO LIMA DA COSTA

SISTEMA SUPERVISÓRIO PARA DISPOSITIVOS MODBUS TCP/IP VIA APLICAÇÃO *WEB*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em engenharia de controle e automação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Bacharel em engenharia de controle e automação.

Aprovado em: 26/10/2022.

BANCA EXAMINADORA

_____

Prof. Dr. Douglas Contente Pimentel Barbosa (Orientador)

Universidade Federal de Pernambuco

_____

Prof. Dr. Rafael Cavalcanti Neto (Examinador Interno)

Universidade Federal de Pernambuco

Dedico esse trabalho aos meus pais, Idílio e Valquiria.

AGRADECIMENTOS

Agradeço a minha família e meus amigos por todo apoio concedido até hoje. Agradeço também ao prof. Dr. Douglas pela orientação e ajuda.

Todas as inovações eficazes são surpreendentemente simples. Na verdade, maior elogio que uma inovação pode receber é haver quem diga: “Isto é óbvio! Por que não pensei nisso antes?” (Peter Drucker).

RESUMO

Atualmente, vários equipamentos utilizados em processos produtivos como indústrias ainda utilizam protocolos com meios físicos do tipo serial. Apesar da sua confiabilidade ser assegurada dado sua predominância no mercado, existem dificuldades na versatilidade de parametrização e aquisição dos dados desses dispositivos. Com isso em mente, é possível observar que a grande maioria dos medidores de energia do mercado (equipamento essencial em aplicações industriais, comerciais e residenciais) ainda utilizam o protocolo MODBUS RTU via meio físico RS485 para comunicação. Logo, nesse trabalho, foi possível desenvolver uma estratégia para conversão dessas informações para o protocolo MODBUS TCP/IP, permitindo assim uma maior facilidade na parametrização e aquisição de dados.

Além disso, desenvolveu-se também scripts em *Python* que cominam em um supervisório, ferramenta para envio e recebimento de dados, através de uma aplicação web. Dessa forma, facilita-se a visualização dos dados de campo pelo usuário e evita a necessidade de instalar um sistema supervisório inteiro (ex. ScadaBR), para aquisitar e apresentar tais informações. Observa-se também que como se trata de uma aplicação web, existe a garantia que todos os dados apresentados são iguais para todos os usuários e o seu acesso pode ser feito a partir de qualquer dispositivo que possua um navegador web a disposição, inclusive com smartphones. Além disso, devido aos scripts serem feitos em *Python*, também é possível executá-lo nativamente de uma *Raspberry PI*, facilitando implementações em redes onde um dispositivo como um computador não está disponível para executar o programa de forma contínua.

Palavras-chave: Protocolo de comunicação; MODBUS; TCP/IP; Sistemas supervisórios; *Python*; Aplicação web.

ABSTRACT

Nowadays, many devices used in productive processes still use protocols whose physical layers rely on serial connections. Even though the reliability of these devices can be assured given their high position in the market share, their versability in parametrization and data acquisition still lacks ease of use. With that in mind, it's possible to observe that the majority of the power meters (devices whose application in industrial, commercial, and residential solutions are of the utmost importance) on the market still utilize the MODBUS RTU protocol through serial means, mostly RS485 as their communication solution. Knowing that in this Undergraduate thesis we could develop a protocol conversion strategy, allowing the use of MODBUS TCP/IP. Resulting in easier device parametrization and data acquisition.

Furthermore, scripts in *Python* were developed to create a SCADA (Supervisory Control and Data Acquisition) system through a web application. By doing that, we can improve data visualization without the necessity to install a whole SCADA system (e.g., ScadaBR) for data acquisition and display. Another advantage that can be explored from the web application is the guarantee of data fidelity through all users since only one host device handles all incoming data requests. Beyond that, any device with browser access can use the SCADA freely, including smartphones. One of the great advantages that *Python* provides, is its native use in embedded multipurpose devices like the *Rspberry PI*. This guarantees that the web application developed using Only *Python* is compatible with this device, allowing flexibility to which devices can run the web application continuously.

Keywords: Communication protocol; MODBUS; TCP/IP; SCADA; *Python*; Web application.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ADU	*Application data unit*
CLP	Controlador lógico programável
IHM	Interface homem máquina
IP	*Internet Protocol*
ISO	*International Organization for Standardization*
LSB	*Least significant bits*
MSB	*Most significant bits*
OSI	*Open System Interconnection*
PDU	*Protocol data unit*
PyPI	*Python Package Index*
SCADA	*Supervisory Control And Data Acquisition*
SQL	*Structured query language*
TCP	*Transmission Control Protocol*

SUMÁRIO

1 INTRODUÇÃO

Sabe-se que redes industriais robustas são necessárias para o bom funcionamento de processos automáticos. Devido a isso, sistemas de controle industriais são apenas uma parte da arquitetura geral de uma indústria, contendo dispositivos de controle e seus equipamentos de campo associados para uma dada aplicação da mesma [1]. Alguns exemplos de protocolos utilizados nessas arquiteturas são o OPC, MODBUS, DNP3, ICCP, CIP, PROFIBUS entre outros [1]. Nesse trabalho, foi desenvolvida comunicações com o protocolo MODBUS juntamente ao TCP/IP, que utilizam a tecnologia *ethernet* para a troca de informações.

Apesar da hegemonia da tecnologia *ethernet* para comunicação entre equipamentos industriais, contemplando aproximadamente 64% dos mesmos [2], muitos desses dispositivos ainda se comunicam exclusivamente por protocolos antigos, como o MODBUS. Apesar da sua idade, devido a sua alta difusão no meio industrial, o protocolo MODBUS-RTU por meio serial ainda é presente em diversos dispositivos e contemplam pelo menos 5% do mercado industrial [2]. Exemplo desses dispositivos são os medidores de energia, utilizados nas mais diversas aplicações, depender de um medidor que se comunica apenas de forma serial apresenta ser um desafio para ambientes cujo rede de comunicação é predominantemente baseada na tecnologia e meio físico *ethernet*.

Em vista disso, a escolha do medidor ideal pode ser baseada em dois critérios principais, complexidade de instalação e custo dos mesmos. Como foi citado anteriormente, a maioria dos ambientes industriais utilizam da tecnologia *ethernet* assim como empreendimentos comerciais e residenciais. Ao utilizar um medidor que possua uma porta *ethernet*, a complexidade da rede pode ser resumida a uma simples conexão entre esse dispositivo e um *switch* ou roteador. Para o medidor serial essa afirmação continua verdadeira, porém será necessário a adição de um *gateway* para que seja feita a interface entre os dados serial e *ethernet*. Ao analisar o custo, utilizando como referência apenas produtos da *Schneider Electric,* o menor preço entre medidores RS-485 e *ethernet* das linhas *PowerLogic e EasyLogic* são respectivamente R$1499,00 [3] e R$5019,00 [4]. Dessa forma, percebe-se a

vantagem em relação a custo dos medidores do tipo serial, fato esse que também explica a sua longevidade.

Sabendo disso, o trabalho desenvolvido explora a estratégia de conversão do protocolo MODBUS-RTU para MODBUS-TCP/IP assim como a criação de uma ferramenta que permita a supervisão do mesmo em uma aplicação *web*, através de um sistema supervisório rudimentar. Devido ao seu desenvolvimento ser exclusivamente em *Python*, existe a possibilidade de executar do supervisório através de uma *Raspberry PI*, permitindo assim aplicações fora do escopo industrial, como a supervisão da qualidade da energia elétrica de um ambiente residencial através de um *smartphone*.

1.1 Objetivos

1.1.1 Geral

Desenvolver uma aplicação em *Python* que aquisite dados de um medidor de energia, cujo protocolo MODBUS-RTU em meio físico RS-485 será convertido em MODBUS-TCP/IP através de um *gateway* e apresente-os em uma aplicação *web* similar a um sistema supervisório através de um navegador.

1.1.2 Específicos

Desenvolver uma fonte de dados que simula um medidor de energia cuja comunicação é feita exclusivamente por MODBUS-TCP/IP. Informações de tensão, corrente e potência monofásicas serão aquisitadas do mesmo, apenas com o intuito de validação do funcionamento do supervisório.

Em seguida, desenvolver dois programas na linguagem de programação *Python*. O primeiro fará a comunicação com o dispositivo simulado, salvando as informações aquisitadas em um banco de dados SQLite. O segundo programa realiza a organização do supervisório por meio de uma página web, onde as informações aquisitadas são apresentadas e os parâmetros definidos.

1.2 Organização do Trabalho

O trabalho foi dividido em fundamentação teórica e em desenvolvimento do mesmo. Na fundamentação, foram abordados temas sobre MODBUS, definição de um *gateway*, dispositivos de campo, noções gerais sobre sistemas supervisórios e aplicações da linguagem de programação *Python.* No desenvolvimento, a topologia de rede da solução será apresentada. Em seguida, a estratégia de conversão de protocolo será apresentada assim como a forma como o dispositivo de campo foi simulado localmente. Além disso, o funcionamento final do supervisório será apresentado junto com a explicação dos códigos desenvolvidos durante o trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 O PROTOCOLO MODBUS

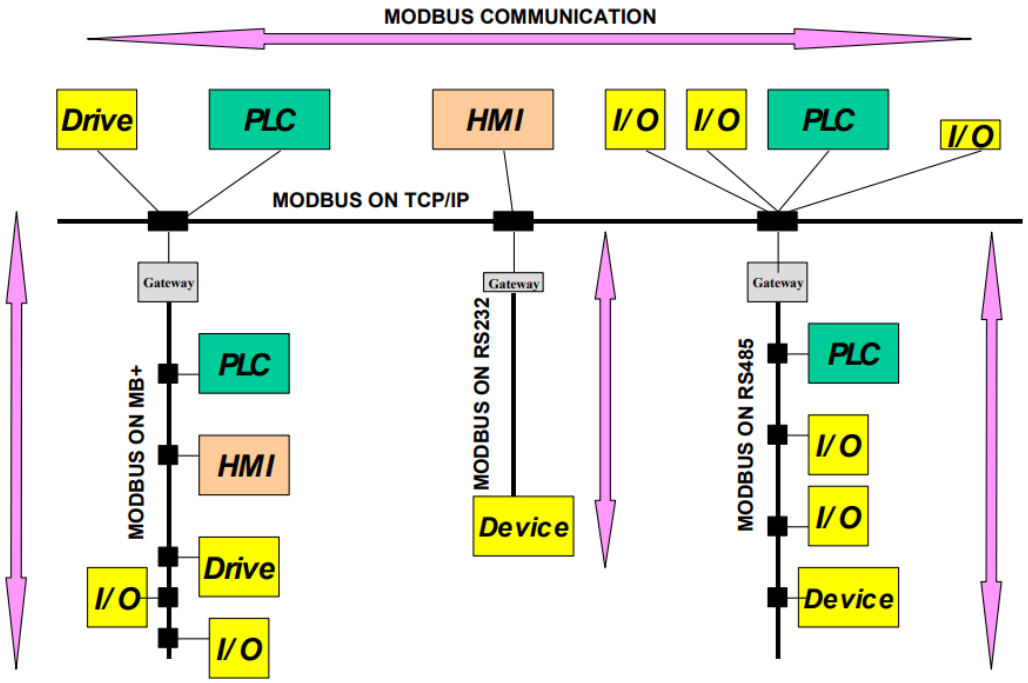
O MODBUS é um protocolo de mensagem estruturada criado no ano de 1979 pela Modicon. É considerado o protocolo mais difundido dentro de redes industriais de manufatura, mas seu escopo de utilização abrange diversas outras áreas industriais e não industriais. Devido a sua grande difusão, esse protocolo pode ser considerado uma “língua franca” entre dispositivos industriais, permitindo assim interoperabilidade entre equipamentos de diferentes marcas [5]. No modelo OSI (*Open System Interconnection*) da ISO (*International Organization for Standardization*) o MODBUS é definido com um protocolo de comunicação na camada de aplicação para dispositivos que comunicam utilizando o padrão mestre-escravo/cliente-servidor [6]. A implementação do protocolo é feita, majoritariamente, de modo serial ou por TCP/IP. Para o modo serial, a comunicação é assíncrona e pode utilizar como meio físico os padrões RS-232, RS-485, Fibra, radio, etc. Felizmente, a interoperabilidade entre redes *serial* e *ethernet* TCP/IP pode ser resolvida com o uso de *gateways*, que fazem a interface entre os protocolos [6]. A Figura 1 apresenta uma arquitetura de rede apresentando essa flexibilidade do protocolo.

O pacote de dados utilizado no MODBUS pode ser dividido em duas partes, PDU (*Protocol Data Unit*) e ADU (*Application Data Unit*) como mostra a Figura 2. O PDU consiste de informações necessárias para a interpretação do protocolo, como qual tipo de função será utilizada e a informação propriamente dita. Enquanto isso, o ADU introduz informações que são variáveis a camada de aplicação utilizada, contendo endereçamento e checagem de erros [6]. Por exemplo, um pacote *serial* e um pacote TCP/IP que realizam a mesma funcionalidade possuem PDUs idênticos, porém utilizarão ADUs diferentes para encapsular a informação no seu meio.

A comunicação é sempre iniciada pelo mestre/cliente, onde o mesmo utiliza de uma *Function Code* (informação de 1 *byte*) para indicar qual operação deve ser feita pelo escravo/servidor. Juntamente, inclui-se uma informação *Data,* que pode incluir o endereço de registradores do servidor e informações a serem escritas [6]. Feito isso, o escravo/servidor processa o pacote recebido e o envia de volta utilizando a mesma

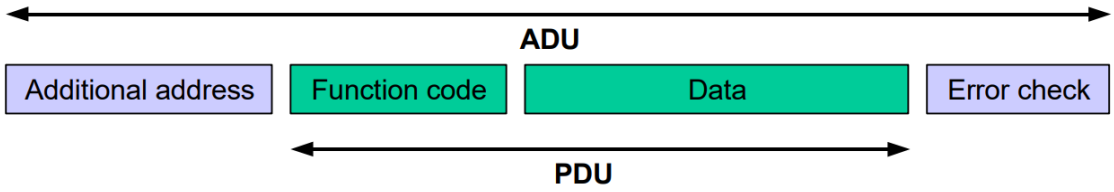
Function Code, e com *Data* contendo os dados requisitados ou a mesma mensagem enviada em casos de escrita. Isso pode ser visto na Figura 3.

Figura 1 – Exemplo de arquitetura de rede MODBUS.



Fonte: MODBUS *application protocol specification* (2012, p. 03) [7].

Figura 2 – Exemplo da organização de um pacote MODBUS.

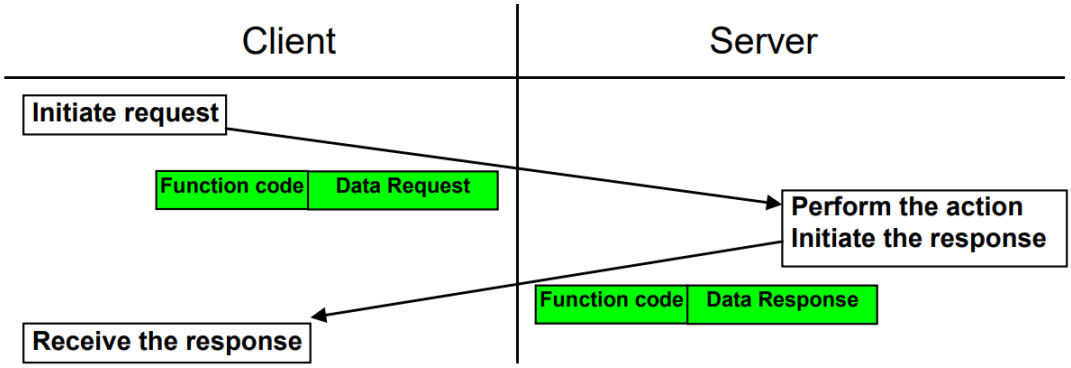


Fonte: MODBUS *application protocol specification* (2012, p. 03) [7].

O modo como os dados são formatados e nomeados no MODBUS podem ser vistos na Figura 4. Nela, observa-se dados que contém apenas 1 bit de informação e outros com 16 bits. Usualmente, informações digitais são armazenadas nos *discretes input* e *coils* enquanto dados analógicos e configurações do dispositivo são guardados em *input registers* e *holding registers*. Além disso, as *Function Codes* disponíveis para

uso encontram-se apresentadas na Figura 5. Vale ressaltar que, em muitos casos, a informação não será contida em apenas um registrador. Por exemplo, alguns medidores utilizam da formatação FLOAT 32 (também conhecida pelo nome *binary32* pela IEEE-754) para armazenar valores reais. Como indicado no nome, o número 32 refere-se à quantidade de *bits* necessário para codificar essa informação. Sabendo disso, durante a aquisição de informações analógicas de um medidor será necessário aquisitar dois registradores e tratar a informação recebida.

Figura 3 – Exemplo de comunicação entre dispositivos MODBUS.



Fonte: MODBUS *application protocol specification* (2012, p. 04) [7].

Figura 4 – Formatação de dados MODBUS.

Primary tables	Object type	Type of	Comments
Discretes Input	Single bit	Read-Only	This type of data can be provided by an I/O system.
Coils	Single bit	Read-Write	This type of data can be alterable by an application program.
Input Registers	16-bit word	Read-Only	This type of data can be provided by an I/O system
Holding Registers	16-bit word	Read-Write	This type of data can be alterable by an application program.

Fonte: MODBUS *application protocol specification*. (2012, p. 06) [7].

Um exemplo de PDU para leitura de *holding registers,* aplicação essa que será necessária para a aquisição dos dados do medidor estudado, pode ser visto na Figura 6. Nesse PDU, observa-se a utilização da *Function Code* 03, referente a leitura assim como o endereço do registrador inicial e a quantidade de registradores subsequentes. Na resposta, observa-se que existe a utilização do mesmo *Function Code* e a

quantidade de *bytes* da resposta. Como foram requisitados três registradores, a resposta de cada inclui um par de *bytes* contendo o MSB e o LSB da informação. Logo, para o registrador 108 temos em hexadecimal 0x022B ou em decimal 555.

Figura 5 – *Function Codes* do MODBUS.

				Function Codes		
				code	*Sub code*	*(hex)*
Data Access	**Bit access**	Physical Discrete Inputs	Read Discrete Inputs	**02**		**02**
		Internal Bits Or Physical coils	Read Coils	**01**		**01**
			Write Single Coil	**05**		**05**
			Write Multiple Coils	**15**		**0F**
	16 bits access	Physical Input Registers	Read Input Register	**04**		**04**
		Internal Registers Or Physical Output Registers	Read Holding Registers	**03**		**03**
			Write Single Register	**06**		**06**
			Write Multiple Registers	**16**		**10**
			Read/Write Multiple Registers	**23**		**17**
			Mask Write Register	**22**		**16**
			Read FIFO queue	**24**		**18**
	File record access		Read File record	**20**		**14**
			Write File record	**21**		**15**
Diagnostics			Read Exception status	**07**		**07**
			Diagnostic	**08**	**00-18,20**	**08**
			Get Com event counter	**11**		**OB**
			Get Com Event Log	**12**		**0C**
			Report Server ID	**17**		**11**
			Read device Identification	**43**	**14**	**2B**
Other			Encapsulated Interface Transport	**43**	**13,14**	**2B**
			CANopen General Reference	**43**	**13**	**2B**

Fonte: MODBUS *application protocol specification* (2012, p. 11) [7].

Figura 6 – Leitura de registradores no MODBUS.

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**03**	Function	**03**
Starting Address Hi	**00**	Byte Count	**06**
Starting Address Lo	**6B**	Register value Hi (108)	**02**
No. of Registers Hi	**00**	Register value Lo (108)	**2B**
No. of Registers Lo	**03**	Register value Hi (109)	**00**
		Register value Lo (109)	**00**
		Register value Hi (110)	**00**
		Register value Lo (110)	**64**

Fonte: MODBUS *application protocol specification* (2012, p. 15) [7].

2.2 *GATEWAY* MODBUS

Um *gateway* pode ser definido como um dispositivo físico que faz a interface entre redes que possuem protocolos diferentes. Ou seja, o *gateway* realiza tradução de protocolos diferentes para permitir a interoperabilidade de diferentes redes. Sabendo que o protocolo MODBUS permite a sua comunicação de forma serial ou por TCP/IP, pode ser desenvolvido um *gateway* que permita essa conversão. Utilizando microcontroladores, a conversão de MODBUS RTU / MODBUS TCP pode ser feita para reduzir a complexidade de topologias de redes de comunicação [8].

Sabendo da viabilidade dessa conversão, pode-se encontrar facilmente conversores de protocolo no mercado industrial. Um exemplo deles é o dispositivo TSXETG100, mostrado na Figura 7 e desenvolvido pela Schneider *Electric*, antiga Modicon e criadora do protocolo MODBUS. Esse dispositivo permite a conversão do protocolo MODBUS RTU/ASCII em MODBUS TCP/IP utilizando os princípios da documentação oficial do MODBUS.

Figura 7 – *Gateway* TSXETG100.

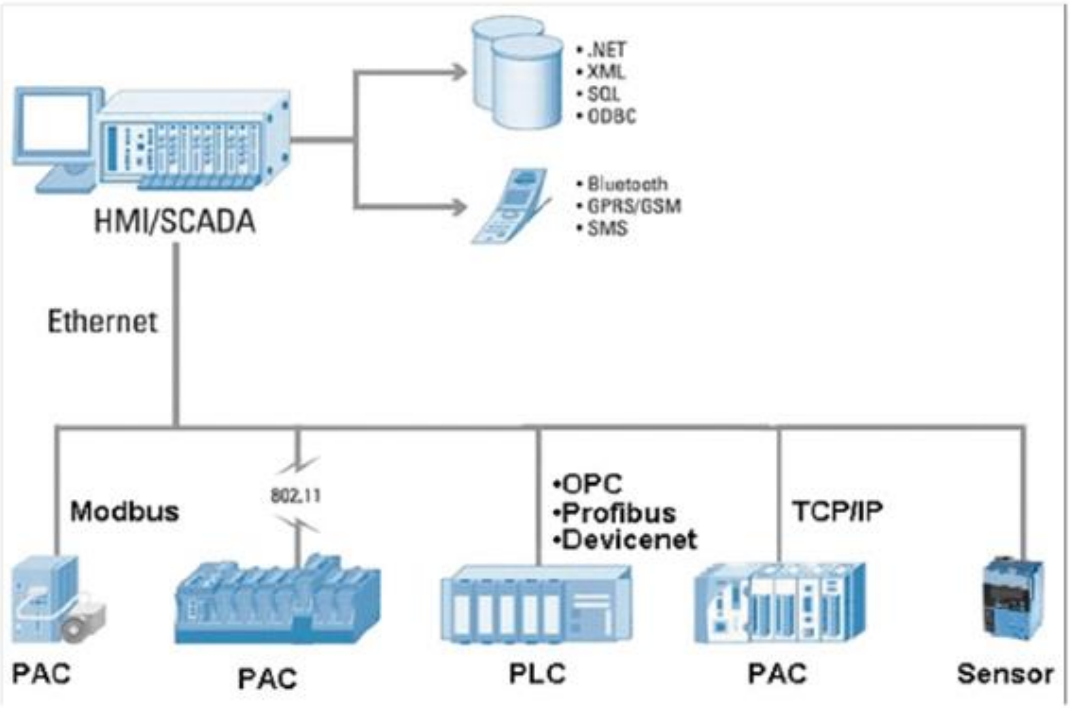
Fonte: *Schneider electric Gateway* TSXETG100 [9].

2.3 SISTEMAS SUPERVISÓRIOS

Um sistema supervisório, ou SCADA (Supervisory Control And Data Acquisition), pode ser definido como um sistema de supervisão e aquisição de dados. Como o próprio nome indica, o sistema foca em supervisionar algum tipo de processo a partir de dispositivos operantes dentro desse meio como CLP's [10]. Pode-se distinguir o sistema supervisório em duas partes, existe o "*cliente layer*" que opera as interações homem máquina e a "*data server layer*" que responde pela aquisição e tratamento das informações vindas de campo [10]. A comunicação pode ser feita com uma grande variedade de protocolos, como TCP/IP, MODBUS, PROFIBUS, etc. A limitação está atrelada a implementação do *driver* por *software* ou pela disponibilidade cartões de interface [10]. A Figura 8 apresenta uma topologia de rede para um SCADA genérico.

Dentro das funcionalidades atribuídas a um SCADA, é possível observar a necessidade de utilização gráficos de tendência e do armazenamento de informações aquisitadas utilizando banco de dados [10].

Figura 8 – Topologia de rede para um SCADA.



Fonte: Profibus org. "5 Perguntas para fazer ao seu fornecedor de ferramentas SCADA" [11].

2.4 A LINGUAGEM DE PROGRAMAÇÃO *PYTHON*

O *Python* é uma linguagem de programação interpretada, interativa e orientada a objetos [12]. É considerada altamente flexível e pode ser utilizada nas mais diversas aplicações, de desenvolvimento *web* a aplicações em computação científica e numérica [13]. A Uma das grandes vantagens da utilização do *Python* é a integração com o PyPI, que é um repositório contendo bibliotecas criadas pela comunidade. Com ele, pode-se baixar e utilizar uma grande diversidade de bibliotecas para inúmeras aplicações. Exemplo disso é o que será feito nesse trabalho. Utilizando o PyPI, é possível utilizar uma biblioteca chamada uModbus, que facilita a comunicação entre dispositivos MODBUS-TCP/IP. Com ela, basta inserir as informações de endereço e porta do dispositivo procurado e com algumas funções já implementadas pode-se fazer leitura e escritas em registradores de dispositivos MODBUS.

Além disso, uma das grandes vantagens de utilizar *Python* para uma aplicação como a de um sistema supervisório é a possibilidade executar os códigos desenvolvidos em uma *Raspberry PI*. Isso deve-se ao suporte nativo desse dispositivo a essa linguagem de programação, dado que o sistema operacional oficial desse dispositivo, o *Raspberry PI OS,* oferece essa e outras inúmeras facilidades [14].

Apesar de tudo isso, o *Python* também possui suas desvantagens. Por exemplo, o seu tempo de execução é maior do que linguagens como Java e C. Essa lentidão pode ocasionar um tempo de execução 56 vezes maior para um programa em *Python* em relação a um com a mesma funcionalidade em C [15].

3 DESENVOLVIMENTO DO TRABALHO

No desenvolvimento do trabalho foi realizado a simulação de um medidor de energia. Esse dispositivo se comunica utilizando o protocolo MODBUS-TCP/IP e será simulado pelo *software MODBUS SLAVE*. A motivação pela abordagem do desenvolvimento dessa maneira deve-se ao alto custo dos equipamentos envolvidos e a procura por diminuir a complexidade do trabalho.

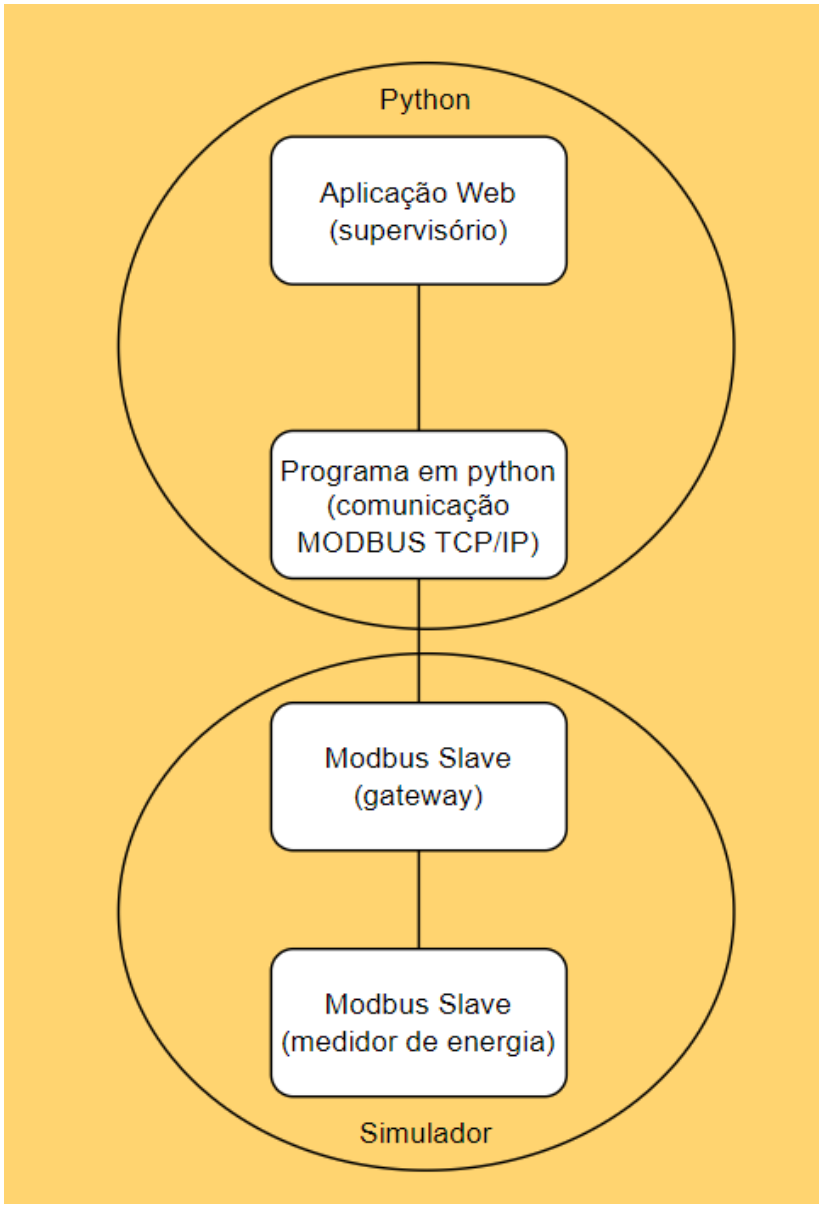
Além disso, utilizou-se a biblioteca *Flask* do *Python* para criar uma aplicação *web* que foi configurada para funcionar como um sistema supervisório, aquisitando as informações do medidor de energia simulado e apresentando as mesmas graficamente.

3.1 TOPOLOGIA DE REDE UTILIZADA

A topologia de rede utilizando os dispositivos físicos consiste de um medidor de energia denominado escravo/servidor que fornece dados em MODBUS-RTU de forma serial via RS-485 (3 condutores). Para realizar a interface, pode se utilizar um *gateway* (conversor de protocolo) para permitir a disponibilização dos pacotes MODBUS-TCP/IP na rede local com o auxílio de um *switch*. No entanto, como o desenvolvimento do trabalho foi feito utilizando apenas simuladores para representar o medidor de energia e o *gateway*, o *software MODBUS SLAVE* simula ambos dispositivos e comunica-se em MODBUS-TCP/IP com o supervisório desenvolvido. Essa topologia pode ser vista na Figura 9.

Com essas informações disponibilizadas na rede, o mestre/cliente pode operar de qualquer local físico desde que possua conexão à rede. Na aplicação desenvolvida, uma *Raspberry PI* pode servir como mestre/cliente sendo conectada através da sua porta RJ45 juntamente ao *switch* ou utilizando *wi-fi* caso exista essa possibilidade.

Figura 9 – Topologia de rede utilizada



Fonte: Autor (2022).

3.2 SIMULAÇÃO DO DISPOSITIVO DE CAMPO

3.2.1 Medidor de energia PM5100

Os medidores de energia são dispositivos que aferem grandezas elétricas como tensão e corrente de forma monofásica ou trifásica. A partir da sua programação interna e construção, informações sobre potência, ativa ou reativa, distorção

harmônica, fator de potência entre outras são grandezas calculadas dadas as aferições básicas já citadas. Usualmente, os dispositivos dessa categoria da marca *Schneider Electric* se comunicam MODBUS-RTU por RS-485 ou MODBUS-TCP/IP por *ethernet*. Nota-se que os dispositivos que não se comunicam são mais baratos que os que possuem essa funcionalidade.

Sabendo disso, utilizou-se parâmetros próprios do medidor PM5100 da *Schneider Electric*, apresentado na Figura 10, para fidelizar o máximo possível a simulação. Nele, os registradores que contém dados analógicos de tensão, corrente, potência e etc. estão localizados em *holding registers* no endereço 3000 em diante, isso é mostrado na Figura 11.

Figura 10 – Medidor PM5100 da *Schneider Electric.*

Fonte: PM5100 series User Manual [15].

Além disso, os dados estão codificados em FLOAT32(conhecido também como *Single-precision floating-point format*), ou seja, ocupam dois registradores de 16 bits para armazenar o dado em questão. Como não foi especificado no manual, assume-se que a informação é do tipo *Big-endian,* que nada mais é que uma forma de envio dos dados como o primeiro registrador representando os MSB e o segundo registrador os LSB.

Figura 11 – Tabela de registradores do medidor PM5100 da *Schneider Electric*.

Sub Cat 1	Sub Cat 2	Description	PM5110_11	PM5310	PM5330_31	PM5320	PM5340_41		Register	Units	Size (INT16)	Data Type
1s Metering (50/60 Cycles)			Y	Y	Y	Y	Y		3000	---		---
	Current		Y	Y	Y	Y	Y		3000	---		---
		Current A	Y	Y	Y	Y	Y		3000	A	2	FLOAT32
		Current B	Y	Y	Y	Y	Y		3002	A	2	FLOAT32
		Current C	Y	Y	Y	Y	Y		3004	A	2	FLOAT32
		Current N	Y	Y	Y	Y	Y		3006	A	2	FLOAT32
		Current G	Y	Y	Y	Y	Y		3008	A	2	FLOAT32
		Current Avg	Y	Y	Y	Y	Y		3010	A	2	FLOAT32
	Current Unbalance		Y	Y	Y	Y	Y		3012	---		---
		Current Unbalance A	Y	Y	Y	Y	Y		3012	%	2	FLOAT32
		Current Unbalance B	Y	Y	Y	Y	Y		3014	%	2	FLOAT32
		Current Unbalance C	Y	Y	Y	Y	Y		3016	%	2	FLOAT32
		Current Unbalance Worst	Y	Y	Y	Y	Y		3018	%	2	FLOAT32
	Voltage		Y	Y	Y	Y	Y		3020	---		---
		Voltage A-B	Y	Y	Y	Y	Y		3020	V	2	FLOAT32
		Voltage B-C	Y	Y	Y	Y	Y		3022	V	2	FLOAT32
		Voltage C-A	Y	Y	Y	Y	Y		3024	V	2	FLOAT32
		Voltage L-L Avg	Y	Y	Y	Y	Y		3026	V	2	FLOAT32
		Voltage A-N	Y	Y	Y	Y	Y		3028	V	2	FLOAT32
		Voltage B-N	Y	Y	Y	Y	Y		3030	V	2	FLOAT32
		Voltage C-N	Y	Y	Y	Y	Y		3032	V	2	FLOAT32
		Voltage N-G	R	R	R	R	R		3034	V	2	FLOAT32
		Voltage L-N Avg	Y	Y	Y	Y	Y		3036	V	2	FLOAT32
	Voltage Unbalance		Y	Y	Y	Y	Y		3038	---		---
		Voltage Unbalance A-B	Y	Y	Y	Y	Y		3038	%	2	FLOAT32
		Voltage Unbalance B-C	Y	Y	Y	Y	Y		3040	%	2	FLOAT32
		Voltage Unbalance C-A	Y	Y	Y	Y	Y		3042	%	2	FLOAT32
		Voltage Unbalance L-L Worst	Y	Y	Y	Y	Y		3044	%	2	FLOAT32
		Voltage Unbalance A-N	Y	Y	Y	Y	Y		3046	%	2	FLOAT32
		Voltage Unbalance B-N	Y	Y	Y	Y	Y		3048	%	2	FLOAT32
		Voltage Unbalance C-N	Y	Y	Y	Y	Y		3050	%	2	FLOAT32
		Voltage Unbalance L-N Worst	Y	Y	Y	Y	Y		3052	%	2	FLOAT32
	Power		Y	Y	Y	Y	Y		3054	---		---
		Active Power A	Y	Y	Y	Y	Y		3054	kW	2	FLOAT32
		Active Power B	Y	Y	Y	Y	Y		3056	kW	2	FLOAT32
		Active Power C	Y	Y	Y	Y	Y		3058	kW	2	FLOAT32

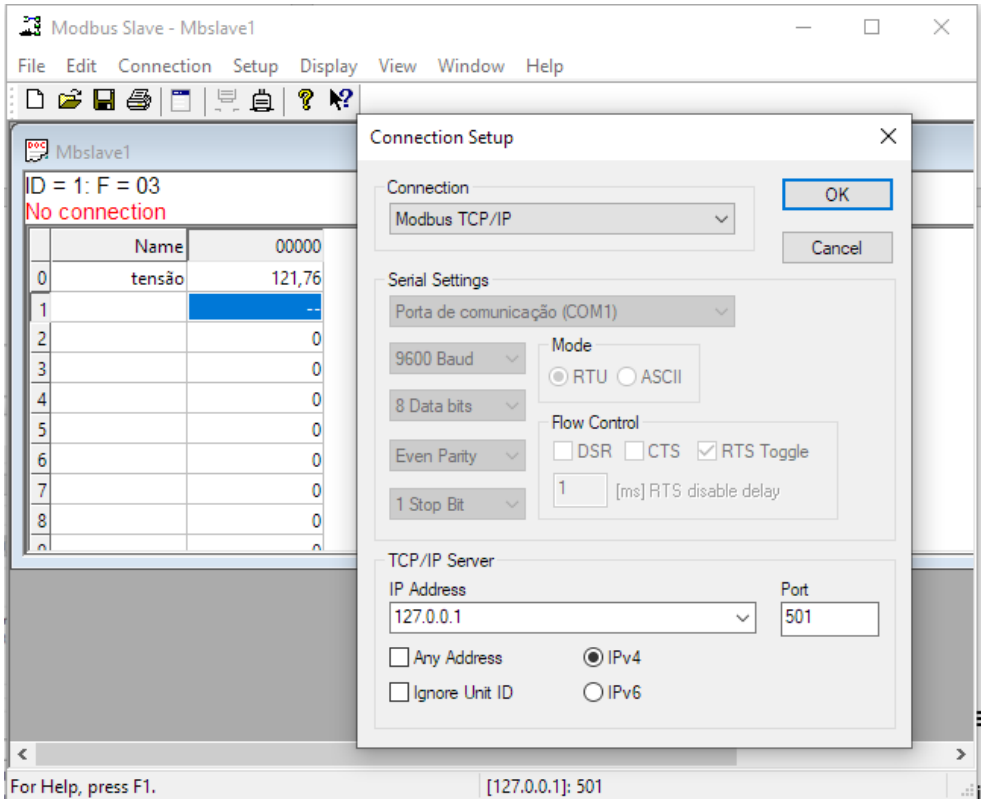
Fonte: Schneider electric PM51xx_PM53xx_PMC Register List_v2011_v2021_R01 (2021) [16].

3.2.2 Simulação do medidor de energia PM5100

Como já visto, um pacote de informação do protocolo MODBUS pode ser facilmente criado desde que exista o conhecimento do endereço do dispositivo a ser acessado, função a ser utilizada e registradores a serem lidos ou modificados. Logo, a utilização de simuladores de cliente/mestre ou servidor/escravo é algo comum para quem lida com o protocolo MODBUS no dia a dia.

Dessa forma, a visão dos códigos desenvolvidos será apenas de pacotes MODBUS-TCP na rede local originários de *softwares* de simulação como o *MODBUS SLAVE* da empresa *Modbustools,* apresentado na Figura 12. Ambos *softwares* tem como objetivo simular dispositivos MODBUS como escravos/servidores, permitindo diversas configurações como, endereço de escravo/servidor, modo de transmissão RTU ou ASCII, comunicação serial ou por TCP/IP, modificação de diferentes registros, endereço IP e porta do dispositivo e etc.

Figura 12 – Simulador *Modbus Slave.*



Fonte: Autor (2022).

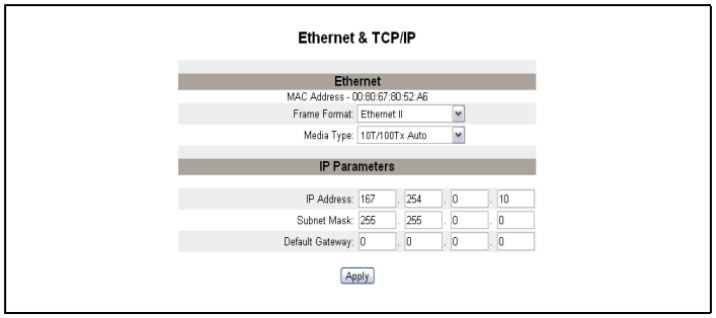
A configuração de simulação utilizada foi a de um dispositivo MODBUS-TCP utilizando um endereço fictício “*localhost*” que pode ser denominado equivalentemente como “127.0.0.1” utilizando a porta 502. Nota-se que essas configurações podem ser modificadas dentro do supervisório mais adiante, a critério do usuário.

3.3 CONFIGURAÇÃO DO *GATEWAY* MODBUS

Apesar do dispositivo físico utilizado (medidor de energia) ser simulado utilizando ferramentas como o *MODBUS SLAVE* da empresa Modbustools ou o ModRSim2 para gerar pacotes TCP/IP diretamente na rede de testes, julgou-se necessário apresentar o funcionamento de um *gateway* MODBUS. O conversor utilizado para essa apresentação foi o TSXETG100 da *Schneider Electric*. Com o dispositivo conectado fisicamente a rede o manual direciona o usuário a acessar as configurações do dispositivo utilizando o IP 169.254.0.10 através de um navegador [17]. Feito isso, é possível configurar a interface TCP/IP do dispositivo que comunicará todas as informações seriais recebidas em RS-485 para a rede inserida. Essa configuração está apresentada na Figura 13.

Por padrão, a porta utilizada é a 502, sem possibilidade de alteração. Feito isso, deve-se configurar os parâmetros seriais para que sejam lidos da forma correta. A tela de configuração serial está apresentada na Figura 14. O medidor estudado (PM5100) pode ser configurado internamente para possuir as seguintes características: Interface física RS482 com 2 fios; MODBUS RTU; *Baud Rate de 9600*; Paridade *Even*. Observa-se que o *mode* da porta serial se refere a como o *gateway* ficará visível na conexão RS-485. Logo, como queremos receber as informações de um medidor do tipo escravo/servidor nessa conexão, o modo de utilização da porta serial do *gateway* será de mestre/cliente. Quando finalizada essas configurações os pacotes em MODBUS-RTU enviados pelo medidor de energia na conexão serial RS-485 agora serão vistos na rede configurada anteriormente como pacotes MODBUS-TCP.

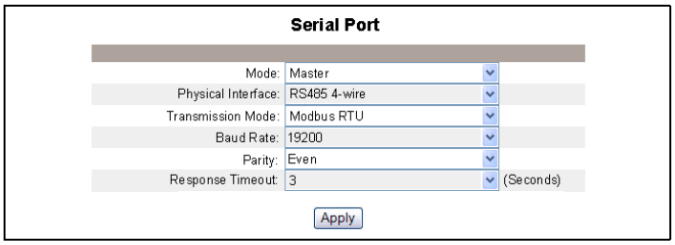
Figura 13 – Configurações TCP/IP do TSXETG100 da *Schneider Electric*.



Option	Description	Setting
Frame Format	Used to select the format for data sent over an Ethernet connection.	Ethernet II, 802.3 SNAP **Default:** Ethernet II
Media Type	Used to define the physical Ethernet connection or media type.	• 10T/100Tx Auto • 10BaseT-HD • 10BaseT-FD • 100BaseTX-HD • 100BaseTX-FD **Default:** 10T/100Tx Auto
IP Address	Used to enter the static IP address of the ETG.	**Default:** 169.254.0.10
Subnet Mask	Used to enter the Ethernet IP subnet mask address of your network.	**Default:** 255.255.0.0
Default Gateway	Used to enter the gateway (router) IP address used for wide area network (WAN) communications.	**Default:** 0.0.0.0

Fonte: Schneider *electric* TSXETG100 *user guide* [17].

Figura 14 – Configuração *Serial* do TSXETG100 da *Schneider Electric*.



Option	Description	Setting
Mode	Used to select how the COM port on the ETG is utilized (master or slave). *NOTE: When the Mode is changed, the ETG reboots.*	Master, Slave **Default:** Master
Physical Interface	Used to select how the ETG serial port is physically wired.	RS485 4-wire, RS485 2-wire, or RS232 **Default:** RS485 2-wire
Transmission Mode	Used to select how data is transmitted over a serial connection.	Modbus RTU, Modbus ASCII **Default:** Modbus RTU
Baud Rate	Used to select the data transmission speed over a serial connection.	2400, 4800, 9600, 19200, 38400, 56000*, 57600* **Default:** 19200
Parity	Used to select if data is checked for accuracy using a parity bit.	Even, Odd, None **Default:** Even
Response Timeout	Used to select how long the ETG will wait to receive a response from a serial device.	0.1 to 10 seconds **Default:** 3 seconds
Remote Modbus TCP/IP Connections (Slave mode only)	Used to define a list of Modbus TCP/IP addresses for the ETG to use during slave mode communications.	—
* Available only if the physical interface and transmission mode is RS232/Modbus ASCII.		

Fonte: Schneider *electric* TSXETG100 *user guide* [17].

3.4 CÓDIGOS DESENVOLVIDOS

Para o desenvolvimento do supervisório utilizou-se da linguagem de programação *Python* juntamente com algumas bibliotecas para auxiliar na comunicação MODBUS-TCP, comunicação com o banco de dados SQLite e criação da aplicação *web.*

3.4.1 COMUNICAÇÃO MODBUS-TCP/IP

A biblioteca uModbus (ou µModbus) criada por Auke Willem Oosterhoff, consiste de uma implementação feita puramente em *Python,* seguindo a documentação oficial MODBUS *Application Protocol Specification* V1.1b3 [17]. Utilizando essa biblioteca pode-se criar rotinas de mestres/clientes ou escravo/servidores em MODBUS-TCP/IP ou MODBUS-RTU. Nela é possível abstrair toda a montagem dos pacotes MODBUS e concentrar-se apenas em qual função deseja-se usar, para qual dispositivo e quais registradores serão lidos ou escritos. O código desenvolvido para a comunicação MODBUS-TCP/IP pode ser encontrado no Apêndice A.

3.4.2 BANCO DE DADOS SQLITE

A decisão por usar o banco de dados relacional SQLite, onde relacional significa a organização das informações em tabelas, parte da existência de uma biblioteca chamada sqlite3 que é nativa do *Python* e permite criar ou manipular informações dentro de um banco de dados desse tipo. As informações salvas dentro do banco já citado consistem de duas tabelas. Uma tabela tem título “PARAMETROS”, está apresentada na Figura 15, e contém colunas referentes a parametrização do dispositivo escravo/servidor que possui informações a serem aquisitadas.

Figura 15 – Tabela PARAMETROS do banco de dados.

	ID	IP	PORTA	ENDERECO	REGISTRADOR_TENSAO	REGISTRADOR_CORRENTE	REGISTRADOR_POTENCIA
	Search columr	Search column	Search column...	Search column...	Search column...	Search column...	Search column...
1	1	127.0.0.1	502	1	40001	40002	40003

Fonte: Autor (2022).

A outra tabela criada foi intitulada “MEDIDOR_1”, ela armazena as informações aquisitadas do medidor juntamente a uma estampa de data e hora. A mesma está sendo apresentada na Figura 16. Os códigos relacionados ao armazenamento de informação do medidor de energia foram inseridos no Apêndice B.

Figura 16 – Tabela MEDIDOR_1 do banco de dados.

	ID	TENSAO	CORRENTE	POTENCIA	DATA_HORARIO
	Search column...	Search column...	Search column...	Search column...	Search column...
7809	7809	471	255	1248	2022-10-13 20:06:36
7810	7810	472	256	1249	2022-10-13 20:06:37
7811	7811	473	257	1250	2022-10-13 20:06:38
7812	7812	474	258	1251	2022-10-13 20:06:39
7813	7813	475	259	1252	2022-10-13 20:06:41
7814	7814	476	260	1253	2022-10-13 20:06:42
7815	7815	477	261	1254	2022-10-13 20:06:43
7816	7816	478	262	1255	2022-10-13 20:06:44
7817	7817	479	263	1256	2022-10-13 20:06:45
7818	7818	480	264	1257	2022-10-13 20:06:47

Fonte: Autor (2022).

3.4.3 APLICAÇÃO WEB

A decisão da escolha de uma página web como representação do supervisório traz algumas vantagens na interoperabilidade do mesmo. Visto que um navegador de internet pode ser encontrado na maioria dos dispositivos eletrônicos de uso pessoal como *smartphones*, computadores e *notebooks*, é possível criar uma plataforma onde a informação é a mesma independente do dispositivo.

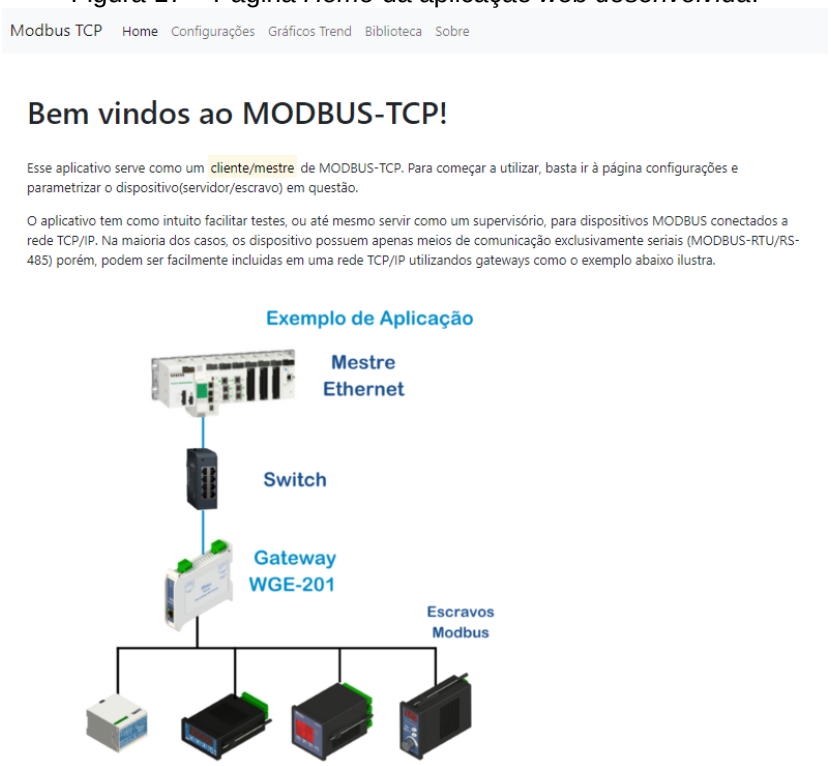
Além disso, o desenvolvimento *web* traz consigo a possibilidade de utilização de *frameworks,* que nada mais são que códigos auxiliares que fazem a estrutura do seu site se adaptar na maior variedade possível de dispositivos, independentemente do tamanho do *display*. Além disso, garante-se uma unificação das informações apresentadas a todos usuários assim como necessita-se de apenas um dispositivo executando o supervisório para que os demais tenham acesso ao mesmo. Logo, esse dispositivo pode ser uma simples *Raspberry PI*.

O código necessário para a execução da aplicação *web* está apresentado no Apêndice C e foi feito em *flask,* que nada mais é que uma biblioteca nativa do *Python* criada para auxiliar no desenvolvimento *web*. Não somente, a estruturação de cada página do supervisório foi feita em HTML e estão sendo apresentadas nos Apêndices de D à H.

3.5 UTILIZAÇÃO DO SUPERVISÓRIO

Para conectar-se a aplicação *web*, é necessário utilizar o IP da máquina que está executando o código em *Python* e utilizar uma porta pré-estabelecida. Durante os testes, esse endereço foi “192.168.15.7:5000”. Observa-se a utilização de “:” para indicar qual porta está sendo utilizada, nesse caso foi a de número 5000. A página inicial da aplicação desenvolvida, denominada “*Home”,* encontra-se na Figura 17 e seu código está apresentado no Apêndice E.

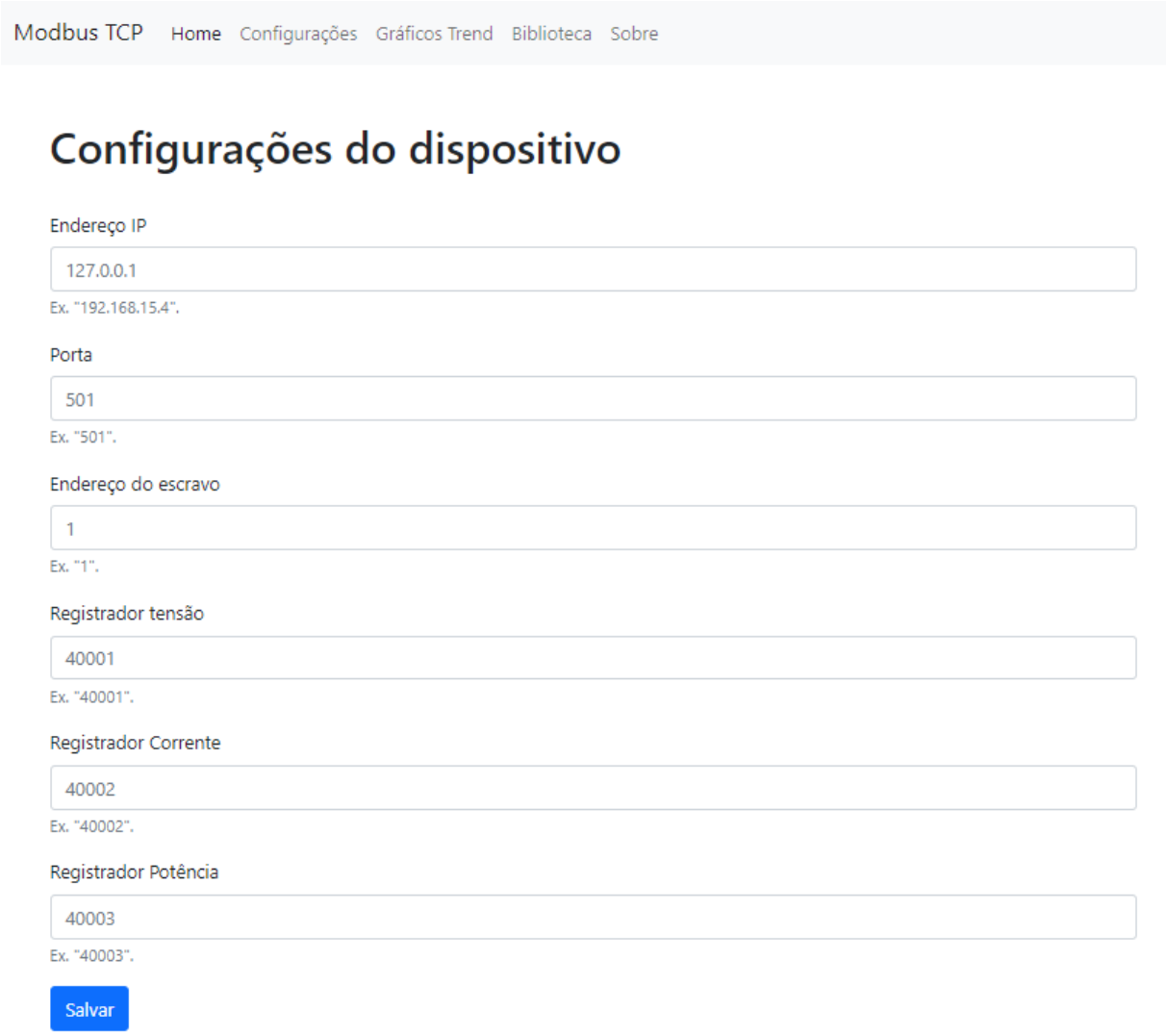
Figura 17 – Página *Home* da aplicação *web desenvolvida*.



Fonte: Autor (2022).

Para realizar configurações pertinentes ao escravo/servidor que está comunicando-se em MODBUS-TCP/IP com a aplicação *web*, utiliza-se a página “Configurações”. Na Figura 18, é possível observar que a página permite checar e modificar todos os parâmetros referentes ao medidor de energia. Essa informação está sincronizada com a tabela “PARAMETROS” contida no banco de dados.

Figura 18 – Página configurações da aplicação *web* desenvolvida.



Fonte: Autor (2022).

Na página “Gráficos Trend”, Figura 19, observa-se a utilização de um gráfico de tendência para auxiliar na visualização das informações aquisitadas dos medidores de energia.

Figura 19 – Página gráficos *Trend* da aplicação *web* desenvolvida.

Fonte: Autor (2022).

Com o intuito de auxiliar os usuários, foi incluído uma página denominada “Biblioteca” que contém explicações gerais do MODBUS, assim como sua documentação oficial. Essa página está apresentada na Figura 20.

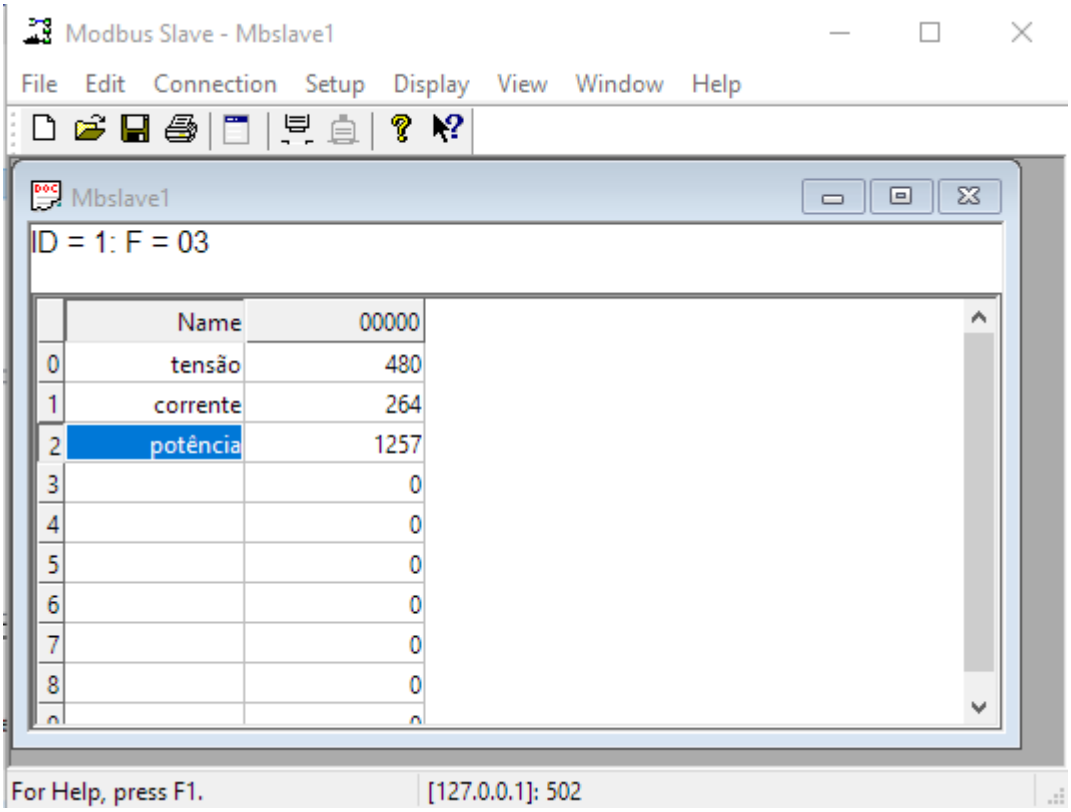
Figura 20 – Página Biblioteca da aplicação *web* desenvolvida.

Fonte: Autor (2022).

3.6 TESTES DO SUPERVISÓRIO

Com o intuito de validar o funcionamento do supervisório, utilizou-se de um simulador de escravo/servidor MODBUS-TCP/IP para gerar dados que serão aquisitados. Na Figura 21, observa-se que foram incluídas três informações que incrementam automaticamente nos *holdings registers* de endereços 40001, 40002 e 40003. O endereço IP utilizado para esse escravo/servidor simulado foi o 127.0.0.1 na porta 502.

Figura 21 – Simulação do medidor de energia.



Fonte: Autor (2022).

Ao executar o código do Apêndice A, foram recebidas três informações como apresenta a Figura 22. Essas informações estão de acordo com os valores inseridos anteriormente. Para validar o banco de dados, inspecionou-se os últimos dados salvos como mostra a Figura 23. Da mesma forma, os valores estão compatíveis com os

simulados. Logo, pode-se afirmar que o supervisório está recebendo os dados esperados.

Figura 22 – Informações recebidas durante a simulação.

```
tensao '479'salvo com sucesso
Corrente '263'salvo com sucesso
Potencia '1256'salvo com sucesso
resposta leitura_tensao:  [480, 264, 1257]
```

Fonte: Autor (2022).

Figura 23 – Banco de dados durante a simulação.

	ID	TENSAO	CORRENTE	POTENCIA	DATA_HORARIO
	Search column...	Search column...	Search column...	Search column...	Search column...
7809	7809	471	255	1248	2022-10-13 20:06:36
7810	7810	472	256	1249	2022-10-13 20:06:37
7811	7811	473	257	1250	2022-10-13 20:06:38
7812	7812	474	258	1251	2022-10-13 20:06:39
7813	7813	475	259	1252	2022-10-13 20:06:41
7814	7814	476	260	1253	2022-10-13 20:06:42
7815	7815	477	261	1254	2022-10-13 20:06:43
7816	7816	478	262	1255	2022-10-13 20:06:44
7817	7817	479	263	1256	2022-10-13 20:06:45
7818	7818	480	264	1257	2022-10-13 20:06:47

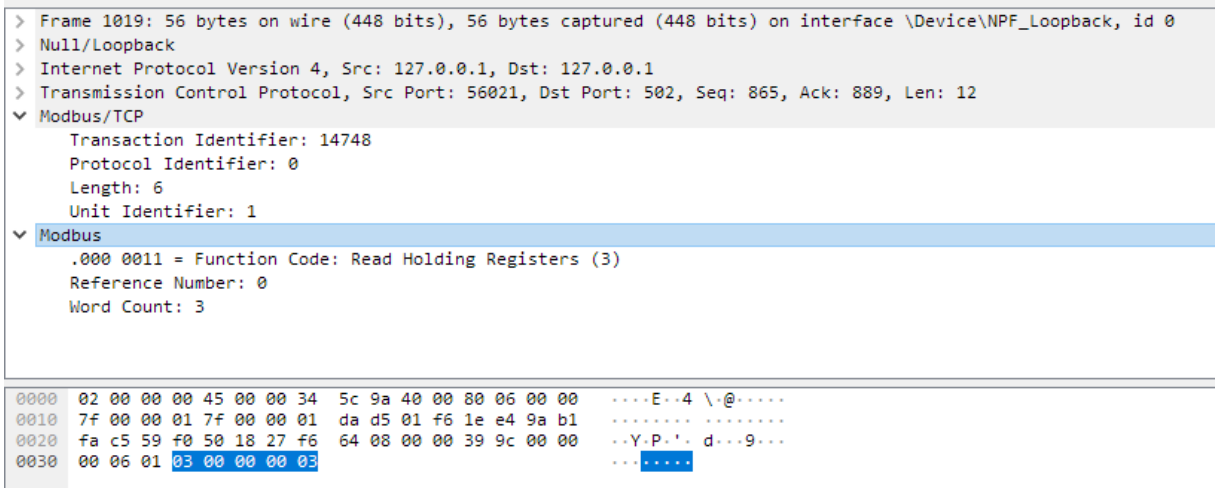
Fonte: Autor (2022).

Apesar disso, julgou-se necessário analisar os pacotes de informações utilizando o *software Wireshark*. Dessa maneira, é possível ter certeza que a comunicação realizada utiliza o protocolo MODBUS-TCP/IP e que o PDU da requisição e da resposta estão de acordo com o padrão apresentado na fundamentação teórica.

Na Figura 24, o pacote de requisição está apresentado e é possível observar que a montagem do PDU é a esperada. Nela utilizou-se a *Function Code* 03 em base hexadecimal e foram requisitados três registradores a partir do registrador 0. Para os *holding registers* sabe-se que o endereço 0 refere-se ao 40001. Logo, foi requisitado ao medidor de energia simulado as informações dos registradores 40001, 40002 e 4003. Da mesma forma, para o pacote de resposta apresentado na Figura 25, o PDU

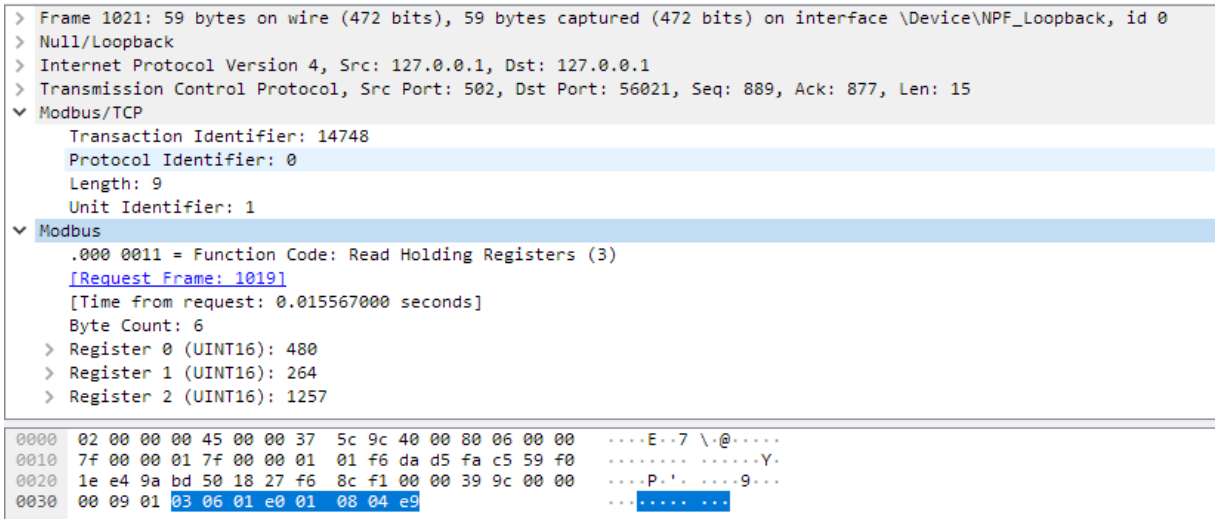
contém um quantitativo de *bytes* de resposta e os valores MSB e LSB para cada registrador aquisitado. Pela imagem, é possível aferir que os dados simulados foram salvos da forma correta no banco de dados utilizando o protocolo MODBUS-TCP/IP.

Figura 24 – Pacote de requisição.



Fonte: Autor (2022).

Figura 25 – Pacote de resposta.



Fonte: Autor (2022).

4 CONCLUSÕES E PROPOSTAS DE CONTINUIDADE

4.1 CONCLUSÃO

Como a aplicação desenvolvida procura apenas aquisitar dados de *holding registers*, que correspondem a grandezas analógicas como Tensão, corrente etc. pode-se dizer que o supervisório cumpre o seu papel de aquisitar e salvar informações em um banco de dados para geração de relatórios futuros. Não somente isso, foi possível também gerar uma tela de gráficos de tendência, onde o usuário recebe em tempo real valiosas informações do dispositivo de campo.

Apesar disso, o supervisório carece de configurações mais flexíveis dado os diversos padrões utilizados para o modbus. Exemplo disso seria a possibilidade de utilizar um dispositivo ASCII no lugar de RTU, ou permitir a leituras de diferentes registradores que não sejam do tipo *holding registers*. Dessa forma, seria interessante desenvolver melhor a tela de configurações afim de permitir a comunicação utilizando qualquer *Function Code* disponível pelo protocolo MODBUS. Não somente, a realização de testes estatísticos para validar a integridade da comunicação poderia confirmar o bom funcionamento do supervisório em longo períodos de execução.

Existe também uma demanda de envio de informações para os registradores dos medidores, permitindo configurar o dispositivo físico remotamente pelo supervisório. Outra implementação futura seria permitir comunicação com diferentes protocolos, atualizar os visualizadores de dados para que fossem mais elaborados e criar uma página específica para geração de relatórios.

4.2 TRABALHOS FUTUROS

- Melhorar as configurações do supervisório para que seja possível parametrizar a comunicação MODBUS-TCP/IP na sua totalidade.
- Implementar a geração de relatórios a partir dos dados aquisitados.
- Implementar a criação de alarmes.

- Utilizar de um simulador MODBUS-RTU para simular o medidor de energia e o *Nodered* para simular o *gateway*, permitindo assim que todos os dispositivos estejam representados e a topologia fique representada de forma fidedigna.
- Implementar a comunicação em outros protocolos.
- Validar a integridade da comunicação por meios estatísticos.

REFERÊNCIAS

1. D. KNAPP, E.; THOMAS LANGILL, J. Industrial Network Security. 2ª. ed. Waltham: Elsevier, 2015. Cap. 2, p. 14-16.

2. HMS NETWORKS. Industrial network market shares 2020 according to HMS Networks. **Site da HMS networks**, 2020. Disponivel em: <https://www.hms-networks.com/news-and-insights/news-from-hms/2020/05/29/industrial-network-market-shares-2020-according-to-hms-networks>. Acesso em: 31 Outubro 2022.

3. SCHNEIDER ELECTRIC. Medidor METSEDM6200HCL10RS. **Site da Schneider Electric**, 2022. Disponivel em: <https://www.se.com/br/pt/product/METSEDM6200HCL10RS/multimedidor-de-tensao-corrente-frequencia-com-comunicacao-modbus-rtu-metsedm6200hcl10rs/?parent-subcategory-id=4115&range=65255-medidor-easylogic-dm6x00h&selected-node-id=12144303478>. Acesso em: 31 Outubro 2022.

4. SCHNEIDER ELECTRIC. MULTIMEDIDOR PM5340 COM PORTA ETHERNET. **Site da Schneider Electric**, 2022. Disponivel em: <https://www.se.com/br/pt/product/METSEPM5340/multimedidor-pm5340-com-porta-ethernet/?parent-subcategory-id=4115&range=61281-pm5000&selected-node-id=12146169702>. Acesso em: 31 Outubro 2022.

5. ORGANIZATION, M. Modbus FAQ, 2022. Disponivel em: <https://modbus.org/faq.php>. Acesso em: 26 Setembro 2022.

6. MODBUS ORGANIZATION. **MODBUS APPLICATION PROTOCOL SPECIFICATION**. [S.l.]: [s.n.], v. V1.1b3, 2012.

7. ORGANIZATION, M. MODBUS specification. **Site do MODBUS organization**, 2012. Disponivel em: <https://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf>. Acesso em: 27 Outubro 2022.

8. AKBATI, O.; KARACA, S. MODBUS GATEWAY DESIGN USING ARM MICROPROCESSOR. **International Journal of Innovation Scientific Research and Review**, 30 Junho 2021. 1-4.

9. SCHNEIDER ELECTRIC. CONVERSOR ETHERNET PARA MODBUS SERIAL TSXETG100, 2022. Disponivel em: <https://www.se.com/br/pt/product/TSXETG100/conversor-ethernet-para-modbus-serial/>. Acesso em: 27 Outubro 2022.

10. DANEELS, A.; SALTER, W. WHAT IS SCADA? **International Conference on Accelerator and Large Experimental Physics Control Systems**, Trieste, Itália, 1999.

11. PROFIBUS ORGANIZATION. 5 Perguntas para fazer ao seu fornecedor de ferramentas SCADA. Disponivel em: <https://www.profibus.org.br/news/fevereiro2012/tutorial>. Acesso em: 27 Outubro 2022.

12. PYTHON SOFTWARE FOUNDATION. General Python FAQ, 2022. Disponivel em: <https://docs.python.org/3/faq/general.html>. Acesso em: 14 Outubro 2022.

13. PYTHON ORGANIZATION. Applications for Python. Disponivel em: <https://www.python.org/about/apps/>. Acesso em: 14 Outubro 2022.

14. SCHNEIDER ELECTRIC. PowerLogic™ PM5100 series – User manual. **Site da Schneider electric**, 2022. Disponivel em: <https://www.productinfo.schneider-electric.com/pm5100/pm5100-user-manual/PM5100%20User%20Manual/English/BM_PM5100UserManual_EAV15105_0000425845.ditamap>. Acesso em: 27 Outubro 2022.

15. SCHNEIDER ELECTRIC. PM51xx_PM53xx_PMC Register List_v2011_v2021_R01.xls, 2022. Disponivel em: <https://www.se.com/ww/en/faqs/FA234017/>. Acesso em: 31 Outubro 2022.

16. SCHNEIDER ELECTRIC. TSXETG100 CONVERSOR ETHERNET PARA MODBUS SERIAL. Disponivel em: <https://download.schneider-

electric.com/files?p_enDocType=User+guide&p_File_Name=31007166_K11_000_03.pdf&p_Doc_Ref=31007166K11000>. Acesso em: 09 Outubro 2022.

17. WIREBUS. Gateway Ethernet / Modbus RTU (1 Canal) WGE-201. **Site da Wirebus**, 2019. Disponivel em: <https://www.wirebus.com.br/wp-content/uploads/2020/09/MAN-PT-DE-WGE-201-01.01_19.pdf>. Acesso em: 27 Outubro 2022.

APÊNDICE A – CÓDIGO COMUNICAÇÃO MODBUS-TCP/IP

```
# Esse código refere-se ao "supervisório", ou seja, o mestre cuja a função é
acessar as informações do dispositivo (medidor)
# TCP-CLIENTE
# MODBUS MESTRE

#!/usr/bin/env python
# scripts/examples/simple_tcp_client.py
import socket
from time import sleep
from umodbus import conf
from umodbus.client import tcp
from salva_dados_modbus import salva_dados_bd
import sqlite3

conn = sqlite3.connect(r"site_flask\historico_modbus.sqlite")
cursor = conn.cursor()
cursor.execute("SELECT * FROM PARAMETROS ORDER BY ROWID ASC LIMIT 1")
linha_parametros = cursor.fetchone()
conn.close()

ip = linha_parametros[1]
porta = int(linha_parametros[2])
endereco_escravo = int(linha_parametros[3])
registrador_tensao = int(linha_parametros[4])
registrador_corrente = int(linha_parametros[5])
registrador_potencia = int(linha_parametros[6])

# Enable values to be signed (default is False).
conf.SIGNED_VALUES = True

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((ip, porta))

# Returns a message or Application Data Unit (ADU) specific for doing
# Modbus TCP/IP.

leitura_tensao = tcp.read_holding_registers(
    slave_id=endereco_escravo, starting_address=registrador_tensao - 40001,
quantity=3
)

leitura_corrente = tcp.read_holding_registers(
    slave_id=endereco_escravo, starting_address=registrador_corrente - 40001,
quantity=1
```

```
)

leitura_potencia = tcp.read_holding_registers(
    slave_id=endereco_escravo, starting_address=registrador_potencia - 40001,
quantity=1
)

respostas_leitura_medidor = []  # [tensao, corrente, potência]

# Response depends on Modbus function code. This particular returns the
# amount of coils written, in this case it is.

while True:
    resposta_leitura_tensao = tcp.send_message(leitura_tensao, sock)
    resposta_leitura_corrente = tcp.send_message(leitura_corrente, sock)
    resposta_leitura_potencia = tcp.send_message(leitura_potencia, sock)

    print("resposta leitura_tensao: ", resposta_leitura_tensao)

    # respostas_leitura_medidor =
[str(resposta_leitura_tensao[0]),str(resposta_leitura_corrente[0])
,str(resposta_leitura_potencia[0]) ]

    salva_dados_bd(
        str(resposta_leitura_tensao[0]),
        str(resposta_leitura_corrente[0]),
        str(resposta_leitura_potencia[0]),
    )
    sleep(1)
```

APÊNDICE B – CÓDIGO BANCO DE DADOS SQLITE

```python
import sqlite3

def salva_dados_bd(tensao, corrente, potencia):

    connection = sqlite3.connect(r"site_flask\historico_modbus.sqlite")
    cursor = connection.cursor()

    cursor.execute(
        "INSERT INTO MEDIDOR_1 (TENSAO,CORRENTE,POTENCIA) VALUES (?,?,?)",
        (
            tensao,
            corrente,
            potencia,
        ),
    )

    connection.commit()

    cursor.close()

    connection.close()

    print(
        f"\ntensao '{tensao}'salvo com sucesso",
        f"\nCorrente '{corrente}'salvo com sucesso",
        f"\nPotencia '{potencia}'salvo com sucesso",
    )
```

APÊNDICE C – CÓDIGO SERVIDOR WEB

```
import sqlite3

from flask import (
    Flask,
    Response,
    request,
    render_template,
    stream_with_context,
)
import json
import time

app = Flask(__name__)

def get_db_ultima_linha():
    conn = sqlite3.connect("historico_modbus.sqlite")
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM MEDIDOR_1 ORDER BY id DESC LIMIT 1")
    ultima_linha = cursor.fetchone()
    # print(ultima_linha)
    # id, tensao, timestamp = ultima_linha
    # print(id, tensao, timestamp)
    conn.close()
    return ultima_linha


def get_db_parametros():
    conn = sqlite3.connect("historico_modbus.sqlite")
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM PARAMETROS ORDER BY ROWID ASC LIMIT 1")
    linha_parametros = cursor.fetchone()
    conn.close()
    return linha_parametros


def set_db_parametros(parametros):
    conn = sqlite3.connect("historico_modbus.sqlite")
    cursor = conn.cursor()
    cursor.execute(
        "INSERT OR REPLACE INTO PARAMETROS (ID, IP, PORTA, ENDERECO,
REGISTRADOR_TENSAO, REGISTRADOR_CORRENTE, REGISTRADOR_POTENCIA) VALUES
(?,?,?,?,?,?,?)",
        (
            1,
            parametros[0],
```

```
            parametros[1],
            parametros[2],
            parametros[3],
            parametros[4],
            parametros[5],
        ),
    )
    conn.commit()
    cursor.close()
    conn.close()


@app.route("/")
def index():
    return render_template("index.html")

@app.route("/configuracoes", methods=["GET", "POST"])
def configuracoes():
    if request.method == "POST":
        parametros_novos = [
            request.form.get("ip"),
            request.form.get("porta"),
            request.form.get("endereco"),
            request.form.get("registrador_tensao"),
            request.form.get("registrador_corrente"),
            request.form.get("registrador_potencia"),
        ]

        set_db_parametros(parametros_novos)

    (
        id,
        ip,
        porta,
        endereco,
        registrador_tensao,
        registrador_corrente,
        registrador_potencia,
    ) = get_db_parametros()

    return render_template(
        "configuracoes.html",
        ip=ip,
        porta=porta,
        endereco=endereco,
        registradores=[registrador_tensao, registrador_corrente,
registrador_potencia],
```

```
    )


@app.route("/graph")
def graph():
    return render_template("graph.html")


@app.route("/relatorio")
def relatorio():
    return render_template("relatorio.html")


@app.route("/biblioteca")
def biblioteca():
    return render_template("biblioteca.html")


@app.route("/sobre")
def sobre():
    return render_template("sobre.html")


@app.route("/chart-data")
def chart_data():
    def generate_random_data():
        while True:
            id, tensao, corrente, potencia, timestamp = get_db_ultima_linha()
            json_data = json.dumps(
                {
                    "time": timestamp,
                    "value": tensao,
                }
            )
            yield f"data:{json_data}\n\n"
            time.sleep(1)

    response = Response(
        stream_with_context(generate_random_data()), mimetype="text/event-
stream"
    )
    response.headers["Cache-Control"] = "no-cache"
    response.headers["X-Accel-Buffering"] = "no"
    return response


@app.route("/chart-data2")
def chart_data2():
```

```
    def generate_random_data2():
        while True:
            id, tensao, corrente, potencia, timestamp = get_db_ultima_linha()
            json_data = json.dumps(
                {
                    "time": timestamp,
                    "value": corrente,
                }
            )
            yield f"data:{json_data}\n\n"
            time.sleep(1)

    response = Response(
        stream_with_context(generate_random_data2()), mimetype="text/event-
stream"
    )
    response.headers["Cache-Control"] = "no-cache"
    response.headers["X-Accel-Buffering"] = "no"
    return response


@app.route("/chart-data3")
def chart_data3():
    def generate_random_data3():
        while True:
            id, tensao, corrente, potencia, timestamp = get_db_ultima_linha()
            json_data = json.dumps(
                {
                    "time": timestamp,
                    "value": potencia,
                }
            )
            yield f"data:{json_data}\n\n"
            time.sleep(1)

    response = Response(
        stream_with_context(generate_random_data3()), mimetype="text/event-
stream"
    )
    response.headers["Cache-Control"] = "no-cache"
    response.headers["X-Accel-Buffering"] = "no"
    return response


if __name__ == "__main__":
    app.run(debug=True, threaded=True)
```

APÊNDICE D – CÓDIGO DA *NAVBAR*

```
<!doctype html>
<html lang="en">

<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
        integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">

    <title>{% block title %}{% endblock %}</title>
</head>

<body>

    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="{{url_for('index')}}">Modbus TCP</a>
            <button class="navbar-toggler" type="button" data-bs-
toggle="collapse"
                data-bs-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false"
                aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarSupportedContent">
                <ul class="navbar-nav me-auto mb-2 mb-lg-0">
                    <li class="nav-item">
                        <a class="nav-link active" aria-current="page"
href="{{url_for('index')}}">Home</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link"
href="{{url_for('configuracoes')}}">Configurações</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link"
href="{{url_for('graph')}}">Gráficos Trend</a>
                    </li>
                    <!-- <li class="nav-item">
```

```
                        <a class="nav-link"
href="{{url_for('relatorio')}}">Relatórios</a>
                    </li> -->
                    <li class="nav-item">
                        <a class="nav-link"
href="{{url_for('biblioteca')}}">Biblioteca</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link"
href="{{url_for('sobre')}}">Sobre</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>

    <br />

    {% block content %}

    {% endblock %}

    <!-- Optional JavaScript; choose one of the two! -->

    <!-- Option 1: Bootstrap Bundle with Popper -->
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min
.js"
        integrity="sha384-
MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
        crossorigin="anonymous"></script>

</body>

</html>
```

APÊNDICE E – CÓDIGO PÁGINA “HOME”

```
{% extends 'base.html'%}

{% block title %}Home{% endblock %}


{% block content%}

<div class="container">
    <br>
    <h1>Bem vindos ao MODBUS-TCP!</h1>
    <br>
    <p>Esse aplicativo serve como um <mark>cliente/mestre</mark> de MODBUS-
TCP. Para começar a utilizar, basta ir à
        página
        configurações e parametrizar o dispositivo(servidor/escravo) em
questão. </p>

    <p>O aplicativo tem como intuito facilitar testes, ou até mesmo servir
como um supervisório, para dispositivos
        MODBUS conectados a rede TCP/IP. Na maioria dos casos, os dispositivo
possuem apenas meios de comunicação
        exclusivamente seriais
        (MODBUS-RTU/RS-485) porém, podem ser facilmente incluidas em uma rede
TCP/IP utilizandos gateways como o exemplo
        abaixo ilustra.
    </p>

    <div class="ratio ratio-4x3">
        <iframe src="https://www.wirebus.com.br/wp-
content/uploads/2020/09/Mestre-Ethernt-1-600x600.png"
            title="Topologia modbus-gateway"></iframe>
    </div>
</div>

{% endblock %}
```

APÊNDICE F – CÓDIGO PÁGINA “CONFIGURAÇÕES”

```
{% extends 'base.html'%}

{% block title %}Configuração do dispositivo{% endblock %}

{% block content%}

<div class="container">
    <br>
    <h1>Configurações do dispositivo</h1>
    <br>
    <form action="/configuracoes" method="POST">
        <div class="mb-3">
            <label for="exampleInputEmail1" class="form-label">Endereço
IP</label>
            <input class="form-control" id="exampleInputEmail1" name="ip"
placeholder={{ip}}>
            <div id="emailHelp" class="form-text">Ex. "192.168.15.4".</div>
        </div>

        <div class="mb-3">
            <label for="exampleInputEmail1" class="form-label">Porta</label>
            <input class="form-control" id="exampleInputEmail2" name="porta"
placeholder={{porta}}>
            <div id="emailHelp" class="form-text">Ex. "501".</div>
        </div>

        <div class="mb-3">
            <label for="exampleInputEmail1" class="form-label">Endereço do
escravo</label>
            <input class="form-control" id="exampleInputEmail2"
name="endereco" placeholder={{endereco}}>
            <div id="emailHelp" class="form-text">Ex. "1".</div>
        </div>

        <div class="mb-3">
            <label for="exampleInputEmail1" class="form-label">Registrador
tensão</label>
            <input class="form-control" id="exampleInputEmail3"
name="registrador_tensao"
                placeholder={{registradores.0}}>
            <div id="emailHelp" class="form-text">Ex. "40001".</div>
        </div>

        <div class="mb-3">
            <label for="exampleInputEmail1" class="form-label">Registrador
Corrente</label>
```

```
            <input class="form-control" id="exampleInputEmail4"
name="registrador_corrente"
                placeholder={{registradores.1}}>
            <div id="emailHelp" class="form-text">Ex. "40002".</div>
        </div>

        <div class="mb-3">
            <label for="exampleInputEmail1" class="form-label">Registrador
Potência</label>
            <input class="form-control" id="exampleInputEmail5"
name="registrador_potencia"
                placeholder={{registradores.2}}>
            <div id="emailHelp" class="form-text">Ex. "40003".</div>
        </div>

        <button type="submit" class="btn btn-primary">Salvar</button>
    </form>
</div>


{% endblock %}
```

APÊNDICE G – CÓDIGO PÁGINA “GRÁFICOS TREND”

```
{% extends 'base.html'%}
{% block title %}Gráficos Trend{% endblock %}
{% block content%}

<div class="container">
    <div class="row">
        <div class="col-12">
            <div class="card">
                <div class="card-body">
                    <canvas id="canvas"></canvas>
                    <br />
                    <canvas id="canvas2"></canvas>
                    <br />
                    <canvas id="canvas3"></canvas>
                </div>
            </div>
        </div>
    </div>
</div>
<!--suppress JSUnresolvedLibraryURL -->
<script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.4.0/jquery.min.js"></scri
pt>
<!--suppress JSUnresolvedLibraryURL -->
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css">
</script>
<!--suppress JSUnresolvedLibraryURL -->
<script
src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.8.0/Chart.min.js"></scr
ipt>

<!-- FUNÇÃO TENSÃO  -->
<script>
    $(document).ready(function () {
        const config = {
            type: 'line',
            data: {
                labels: [],
                datasets: [{
                    label: "Tensão (V)",
                    backgroundColor: 'rgb(240, 10, 40)',
                    borderColor: 'rgb(240, 10, 40)',
                    data: [],
```

```
                fill: false,
            }],
        },
        options: {
            responsive: true,
            title: {
                display: true,
                text: 'Medidor de energia 1 - Tensão'
            },
            tooltips: {
                mode: 'index',
                intersect: false,
            },
            hover: {
                mode: 'nearest',
                intersect: true
            },
            scales: {
                xAxes: [{
                    display: true,
                    scaleLabel: {
                        display: true,
                        labelString: 'Estampa de tempo'
                    }
                }],
                yAxes: [{
                    display: true,
                    scaleLabel: {
                        display: true,
                        labelString: 'Tensão (V)'
                    }
                }]
            }
        }
    };

    const context = document.getElementById('canvas').getContext('2d');

    const lineChart = new Chart(context, config);

    const source = new EventSource("/chart-data");

    source.onmessage = function (event) {
        const data = JSON.parse(event.data);
        if (config.data.labels.length === 20) {
            config.data.labels.shift();
            config.data.datasets[0].data.shift();
```

```
            }
            config.data.labels.push(data.time);
            config.data.datasets[0].data.push(data.value);
            lineChart.update();
        }
    });
</script>
<!-- FUNÇÃO CORRENTE  -->
<script>
    $(document).ready(function () {
        const config = {
            type: 'line',
            data: {
                labels: [],
                datasets: [{
                    label: "Corrente (A)",
                    backgroundColor: 'rgb(5, 75, 161)',
                    borderColor: 'rgb(5, 75, 161)',
                    data: [],
                    fill: false,
                }],
            },
            options: {
                responsive: true,
                title: {
                    display: true,
                    text: 'Medidor de energia 1 - Corrente'
                },
                tooltips: {
                    mode: 'index',
                    intersect: false,
                },
                hover: {
                    mode: 'nearest',
                    intersect: true
                },
                scales: {
                    xAxes: [{
                        display: true,
                        scaleLabel: {
                            display: true,
                            labelString: 'Estampa de tempo'
                        }
                    }],
                    yAxes: [{
                        display: true,
                        scaleLabel: {
```

```
                            display: true,
                            labelString: 'Corrente (A)'
                        }
                    }]
                }
            }
        };

        const context = document.getElementById('canvas2').getContext('2d');

        const lineChart = new Chart(context, config);

        const source = new EventSource("/chart-data2");

        source.onmessage = function (event) {
            const data = JSON.parse(event.data);
            if (config.data.labels.length === 20) {
                config.data.labels.shift();
                config.data.datasets[0].data.shift();
            }
            config.data.labels.push(data.time);
            config.data.datasets[0].data.push(data.value);
            lineChart.update();
        }
    });
</script>
<!-- FUNÇÃO POTÊNCIA INSTÂNTANEA  -->
<script>
    $(document).ready(function () {
        const config = {
            type: 'line',
            data: {
                labels: [],
                datasets: [{
                    label: "Potência instantânea (kW)",
                    backgroundColor: 'rgb(10, 145, 53)',
                    borderColor: 'rgb(10, 145, 53)',
                    data: [],
                    fill: false,
                }],
            },
            options: {
                responsive: true,
                title: {
                    display: true,
                    text: 'Medidor de energia 1 - Potência'
                },
```

```
                tooltips: {
                    mode: 'index',
                    intersect: false,
                },
                hover: {
                    mode: 'nearest',
                    intersect: true
                },
                scales: {
                    xAxes: [{
                        display: true,
                        scaleLabel: {
                            display: true,
                            labelString: 'Estampa de tempo'
                        }
                    }],
                    yAxes: [{
                        display: true,
                        scaleLabel: {
                            display: true,
                            labelString: 'Potência instantânea (kW)'
                        }
                    }]
                }
            }
        };

        const context = document.getElementById('canvas3').getContext('2d');

        const lineChart = new Chart(context, config);

        const source = new EventSource("/chart-data3");

        source.onmessage = function (event) {
            const data = JSON.parse(event.data);
            if (config.data.labels.length === 20) {
                config.data.labels.shift();
                config.data.datasets[0].data.shift();
            }
            config.data.labels.push(data.time);
            config.data.datasets[0].data.push(data.value);
            lineChart.update();
        }
    });
</script>

{% endblock %}
```

APÊNDICE H – CÓDIGO PÁGINA “BIBLIOTECA”

```
{% extends 'base.html'%}
{% block title %}Biblioteca{% endblock %}
{% block content%}

<div class="container">
    <h1>MODBUS segundo o wikipedia</h1>
    <br>
    <div class="ratio ratio-4x3">
        <iframe src="https://pt.wikipedia.org/wiki/Modbus" title="Modbus
Wikipedia"></iframe>
    </div>
    <br>
    <br>
    <h1>Documentação oficial</h1>
    <br>
    <div class="ratio ratio-4x3">
        <iframe
src="https://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf"
            title="Documentacao MODBUS"></iframe>
    </div>
</div>

{% endblock %}
```