REVISTA BRASILEIRA DE MECATRÔNICA
FACULDADE SENAI DE TECNOLOGIA MECATRÔNICA

SISTEMA IoT PARA MONITORAMENTO DE BOMBEAMENTO DE EFLUENTES

IoT SYSTEM FOR MONITORING EFFLUENT PUMPING

Rodrigo Sousa Botelho[1], [i]
Daniel Barbuto Rossato[2], [ii]
André Luis dos Santos[3], [iii]
Caio Vinícius Ribeiro da Silva[4], [iii]

Data de submissão: (22/11/2022) Data de aprovação: (27/04/2023)

RESUMO

Para aumentar a disponibilidade em uma estação de tratamento de efluentes, foi desenvolvido um projeto de sistema IoT para um sistema de bombeamento. Utilizando uma placa concentradora, os dados são transmitidos com a tecnologia MQTT com protocolo de segurança TLS para a nuvem AWS. Estas informações são armazenadas em um banco de dados SQL de servidor local usando Node-RED. Os dados são visualizados em tempo real em um dashboard leve, de baixo custo de implantação e alta flexibilidade de aplicação. Como resultado, o acompanhamento do processo de forma remota, permitiu economia de recursos por meio de uma análise de erros e falhas.

Palavras-chave: Internet das Coisas, Banco de Dados; Modbus, MQTT.

ABSTRACT

In order to increase availability in an effluent treatment plant, an IoT system project was developed for a pumping system. Using a concentrator card, data is transmitted using MQTT technology with TLS security protocol to the AWS cloud. This information is stored in a local server SQL database using Node-RED. Data is visualized in real-time in a lightweight dashboard, with low deployment cost and high application flexibility. As a result, monitoring the process remotely allowed resources to be saved through an analysis of errors and failures.

Keywords: Internet of Things, Database, Modbus, MQTT.

[1] Pós-graduado em Internet das Coisas na Faculdade SENAI São Paulo. E-mail: rodrigo@catuiengenharia.com.br
[2] Docente na Faculdade SENAI São Paulo – Campus Mariano Ferraz. E-mail: daniel.rossato@sp.senai.br
[3] Docente na Faculdade SENAI São Paulo – Campus Mariano Ferraz. E-mail: andre.lsantos@sp.senai.br
[4] Docente na Faculdade SENAI São Paulo – Campus Mariano Ferraz. E-mail: caio.vinicius@sp.senai.br

1 INTRODUÇÃO

Os sistemas de bombeamento em estações de tratamento de efluentes (ETE) possuem como requisito altos índices de disponibilidade. Estes sistemas de bombeamento possuem painéis elétricos para acionamento das bombas que, devido a questões econômicas e falta de mão de obra especializada, não possuem um operador local para monitoramento e detecção de falhas. Sendo assim, quando ocorre algum erro, só é detectado horas ou até um dia após o ocorrido, gerando então um grande prejuízo tanto financeiro, por possíveis multas; como ambiental, devido ao extravasamento desses efluentes ao meio ambiente, sem o devido tratamento.

De modo a manter a alta disponibilidade do sistema de bombeamento e evitar os prejuízos financeiros e ambientais mencionados, foi elaborado um sistema de monitoramento com tecnologias de Internet das Coisas (IoT – *Internet of Things*) de modo a agilizar a detecção de falhas.

A proposta envolve desenvolver uma placa eletrônica para coletar os sinais digitais e analógicos de um painel elétrico, enviá-los em formato JSON (*JavaScript Object Notation*) via protocolo MQTT (*Message Queuing Telemetry Transport*) por rede sem fio (Wi-fi) para um banco de dados e fornecer a visualização dos dados por meio de painel de indicadores (*dashboard*).

Dentre os sinais a serem coletados podemos destacar:

a) Sinais via rede Modbus RTU do inversor de frequência do motor acoplado à bomba: tensão, corrente, frequência e status do motor e bomba ligados (em operação) ou desligados;
b) Sinal digital: seletor de modo de controle de nível do reservatório (manual ou automático);
c) Sinal analógico: medidor hidrostático do nível do reservatório (4 a 20 mA).

Desta forma, podem ser detectados problemas devido a falhas no motor, bomba, inversor de frequência e manobras irregulares. O armazenamento destes dados, além de possibilitar o monitoramento em um painel de indicadores, servirá como base, a médio prazo, para a implementação de um sistema de manutenção preditiva que poderá acompanhar o desgaste dos equipamentos ao longo do tempo.

Para mitigar problemas de segurança, foi adotado o protocolo MQTTS, ou seja, MQTT com camada TLS (*Transport Layer Security*), fornecendo um canal de comunicação seguro entre um cliente e um servidor por meio da porta reservada TCP 8883. O TLS é um protocolo criptográfico que usa um mecanismo de *handshake* para negociar vários parâmetros na criação de uma conexão segura entre o cliente e o servidor. Após a conclusão do *handshake*, uma comunicação criptografada entre o cliente e o servidor é estabelecida, criando dificuldades para acesso aos dados por meio de espionagem (*sniffing*). Além disso, para garantir a autenticidade das mensagens, os dispositivos cliente e servidor podem enviar certificados digitais *X.509* (normalmente emitido por uma autoridade confiável) usando este protocolo (HIVEMQ TEAM,2015).

2 SISTEMA IOT

Nesta seção é apresentada a arquitetura do sistema IoT utillizado nesta solução incluindo seus componentes e sinais, e em especial, o protocolo de rede Modbus.

2.1 Arquitetetura do sistema IOT

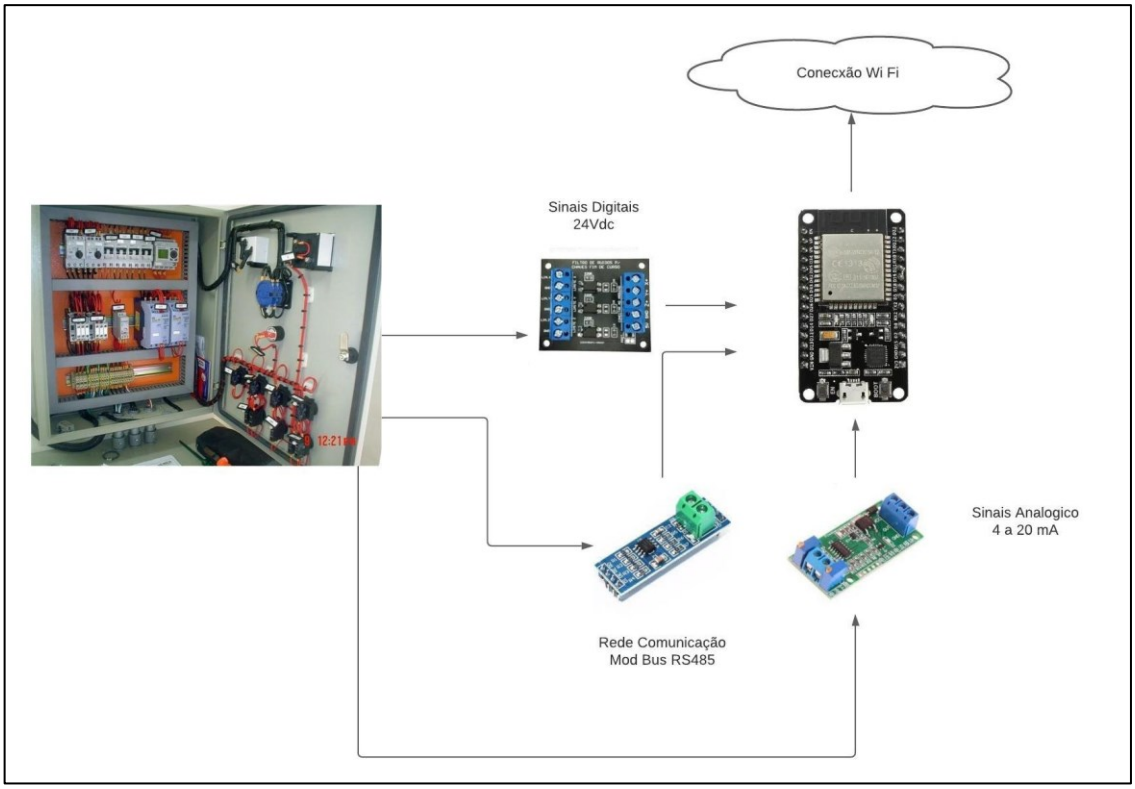
A Internet das coisas industrial (IIoT) consiste na aplicação de sistemas inteligentes para aprimorar os processos de fabricação industriais. Os sistemas inteligentes incorporam funções de detecção, atuação e controle para descrever e analisar uma situação e tomar decisões com base nos dados disponíveis de maneira preditiva ou adaptativa, realizando assim ações inteligentes. A filosofia de condução por trás da IIoT é que as máquinas inteligentes não são apenas melhores do que os humanos na captura e análise de dados, mas também na comunicação de informações importantes que podem ser usadas para conduzir decisões de negócios com maior rapidez e precisão (POSEY; ROSENCRANCE; SHEA, 2021).

Nesta aplicação, o sistema IoT proposto possibilita a integração do equipamento atual com a Internet por meio de uma placa eletrônica para coleta e armazenamento dos dados gerados possibilitando análise do funcionamento do sistema.

Na figura 1, podemos observar a arquitetura da coleta de dados IoT proposto. Foi desenvolvido um protótipo de placa eletrônica tendo como base um microcontrolador (MCU – *MicroController Unit*) modelo ESP32 e circuitos conversores de sinais. Desta forma, os sinais digitais de 24 V do painel elétrico foram convertidos para 3,3 V. O sinal analógico de 4 a 20 mA foi convertido para a faixa de 0 a 3,3 V. E o sinal Modbus RTU com padrão RS485 do inversor de frequência foi convertido para utilização na USART do MCU.

O ESP32 possui conexão Wi-Fi permitindo a integração com a Internet e envio dos dados para a nuvem.

Figura 1 – Arquitetura da coleta de dados IoT



Fonte: Elaborado pelo autor.

2.2 Coleta dos dados Modbus

Dentre os protocolos de comunicação entre dispositivos, pode-se afirmar que o Modbus é um dos mais utilizados na indústria por ter sido desenvolvido ainda na década de 70 pela empresa Modicon, primeiro fabricante de Controladores Lógicos Programáveis (CLPs). Além disso, após ser adquirida pela Schneider, seus direitos de uso foram liberados pela organização Modbus (TRACEY, 2021).

No modbus existe três padrões no meio de transmissão que são:

a) RS-232;
b) RS-485;
c) Ethernet TCP/IP (Modbus TCP).

Nos três meio de transmissão a diferença e velocidade de comunicação, quantidade máxima de dispositivos e comprimento máximo que cada dispositivo pode estar um do outro.

No padrão RS232 e utilizado comunicação ponto a ponto, ou seja podendo admitir apenas 2 dispositivos na rede, nesse modelo modbus só pode ter um mestre e um escravo na rede, a velocidade de comunicação desse padrão pode chegar ao máximo de 115Kbps, a distancia máxima dessa rede pode chegar a 30 metros para baixas taxas de transmissão.

No padrão RS485 e o padrão mais utilizado muito comum na indústria, nesse modelo modbus diferente do RS232 o RS485 pode ter até 32 dispositivos em cada barramento, podendo ter 1 mestre e ate 31 escravos na rede, a velocidade de comunicação desse padrão pode chegar ao máximo de 12 Mbps, a distancia máxima dessa rede pode chegar a 1200 metros.

No padrão Ethernet TCP e utilizado o cabo Ethernet, seguindo as mesmas descrição da de dispositivos do RS485 mudando sua velocidade de comunicação que varia entre 100 Mbps ate 10 Gbps. E sua distancia máxima vai ate 100 metros.

No modbus temos dois tipos de transmissão que são:

a) ASCII;
b) RTU.

A diferença entre eles são o modo de transmissão dos bytes da mensagem, e como a mensagem e empacotada e descompactada não sendo possível utilizar os dois modelos ao mesmo tempo.

Quando configurado em modbus usando o ASCII (*American Standard Code for Information Interchange*), cada byte em uma mensagem envia dois caracteres.

Quando configurado em modbus RTU (*Remote Terminal Unit*), cada mensagem de 8 bits contém dois caracteres hexadecimais de 4 bits.

3 DESENVOLVIMENTO

Nesta seção, a configuração e programação dos dispositivos do sistema IoT são descritas em detalhes.

3.1 Coleta de dados digitais e analógicos

Para a leitura dos sinais digitais, o hardware foi montado de forma que o ESP32 fique protegido de ligações erradas e convertidos de 24 VDC para 3,3 VDC por meio de circuitos com opto-acopladores. A figura 2 mostra o código-fonte para leitura do sinal digital.

Figura 2 – Leitura das entradas digitais

```
//Leitura das entradas digitais
//Entrada 1
  if(!digitalRead(14)){
   dig1="1";
  }
else{
   dig1="0";
}
```

Fonte: Elaborado pelo autor.

O medidor de nível (sonda hidrostática) possui um range de 0 a 100 mca e um sinal de saída analógico de 4 a 20 mA. Sendo assim, foi necessário um conversor para transformar de 4 a 20 mA para 0 a 3,3 VDC. O conversor de sinal analógico-digital (ADC) do ESP32 possui resolução de 12 bits. Portanto teremos uma mudança de escala de 0 a 100 mca de medição do nível do reservatório para um valor de 0 a 4095 no ESP32. Na figura 3 podemos observar o código-fonte de leitura do sinal analógico e conversão da escala para 0 a 100 mca.

Figura 3 – Leitura da entrada analógica

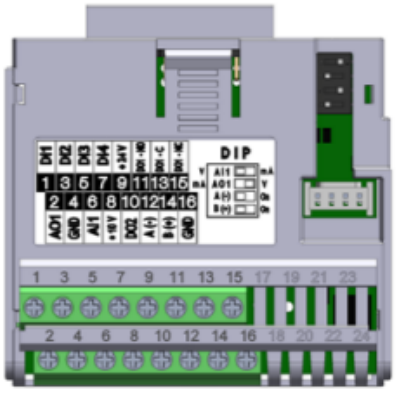
```
//Lendo Analogica
int sensorValue = analogRead(34);

//convertendo analogica de 4 a 20ma P 3V
float voltage = sensorValue * (20 / 4095.0);
```

Fonte: Elaborado pelo autor.

3.2 Configuração da Comunicação Modbus e Teste Leitura

Para realizar a comunicação Modbus, foi utilizado uma expansão de rede RS-485 para o inversor de frequência WEG, cujo modelo é CFW500-CRS485. Na figura 4 podemos observar os bornes de ligação.

Figura 4 – Módulo de expansão RS-485 do inversor WEG CFW500



Pino	Nome	Função
12	A-Line (-)	RxD/TxD negativo
14	B-Line (+)	RxD/TxD positivo
16	Ref.	0V do circuito RS485

Fonte: WEG (2016).

Foram ajustados os parâmetros do inversor de frequência WEG CFW500 conforme tabela 1.

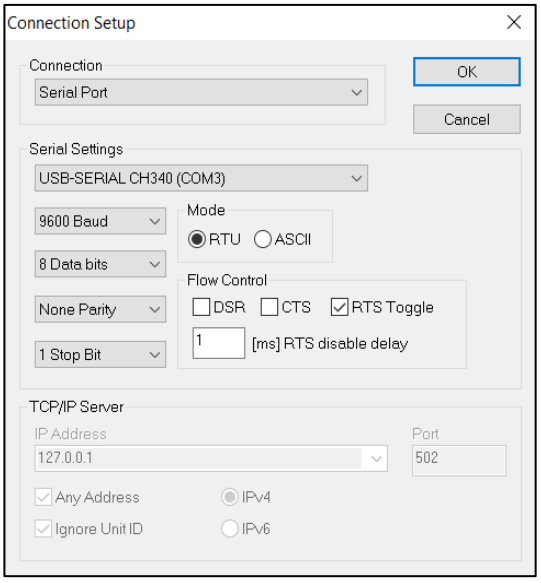
Tabela 1 - Parâmetros ajustados no inversor de frequência

Parâmetros	Descrição parâmetros	Configuração	Descrição
P0000	Acesso aos Parâmetros	5	Acesso a configuração
P0220	Seleção LOC/REM	0	Sempre local
P0221	Seleção Giro LOC	0	Horário
P0224	Seleção Gira/Para LOC	0	TECLA HMI
P0308	Endereço Serial	1	ENDEREÇO MODBUS
P0310	Taxa Comunic. Serial	0	9600
P0311	1 Config. Bytes Serial	0	8 bits, sem, 1
P0312	Protocolo Serial (1) (2)	2	MODBUS RTU (1)

Fonte: Elaborado pelo autor.

Para os testes de comunicação do inversor com o ESP32 foi utilizado o simulador de Modbus "Modbus Poll" (https://www.modbustools.com/modbus_poll.html) com um conversor USB – Serial (RS485). Para realizar a leitura do inversor de frequência, foi utilizado o Modbus Poll no modo Mestre com as seguintes características: leitura via porta Serial, taxa de comunicação 9600 Baud, 8 data bits, sem paridade e 1 stop bit, conforme configuração mostrada na figura 5.

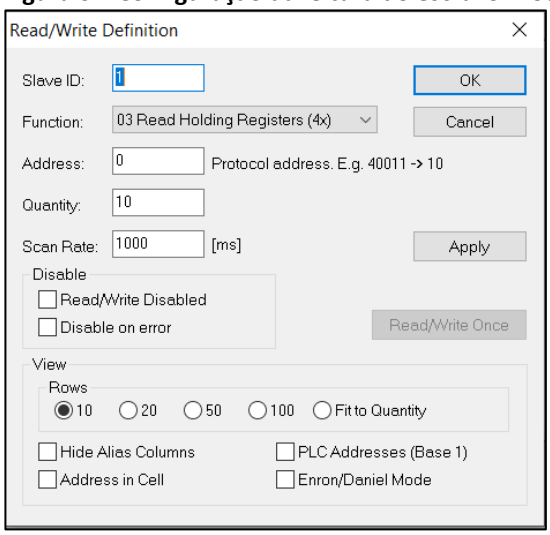
Figura 5 – Configuração do simulador Modbus Poll em modo Mestre



Fonte: Elaborado pelo autor.

Para realizar a leitura dos dados do inversor, foi configurado o escravo ID 1 na função 03 Read Holding Registers (4x), nos Endereços 0 a 10 com uma taxa de varredura 1000 ms.

Figura 6 – Configuração da leitura do escravo Modbus



Fonte: Elaborado pelo autor.

Utilizando o simulador, conseguimos ler os endereços P0002 - velocidade do motor; P0003 - corrente do motor; e P0005 - frequências do motor de acordo com a tabela 2.

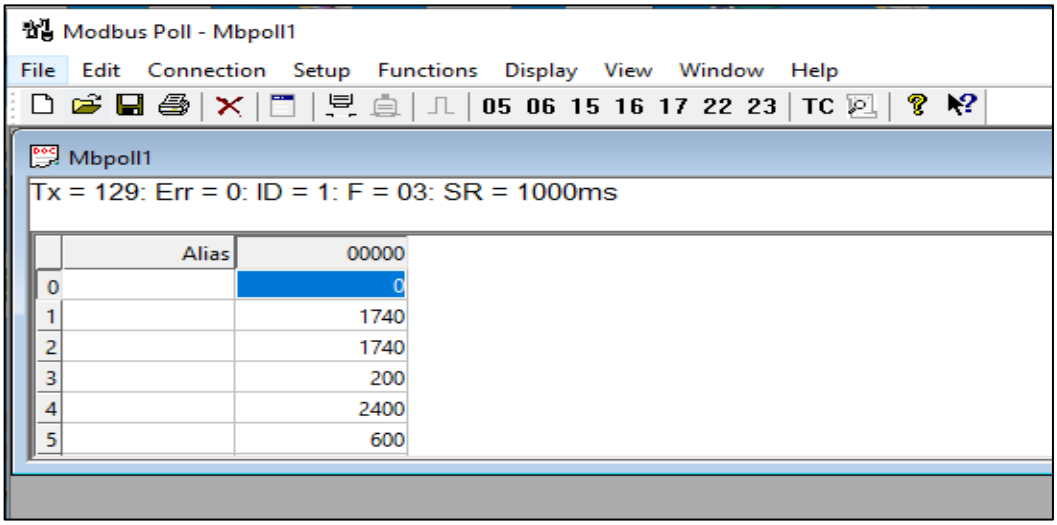
Tabela 2 – Parâmetros de leitura de variáveis do inversor

P0001	Referência Velocidade	0 a 65535			ro	READ	17-1
P0002	Velocidade de Saída (Motor)	0 a 65535			ro	READ	17-1
P0003	Corrente do Motor	0,0 a 200,0 A			ro	READ	17-1
P0004	Tensão Barram. CC (Ud)	0 a 2000 V			ro	READ	17-1
P0005	Frequência de Saída (Motor)	0,0 a 500,0 Hz			ro	READ	17-2

Fonte: WEG (2016).

No simulador é possível verificar a transmissão dos dados que são lidos através do inversor de frequência CFW500 com a placa MODBUS RS485 conforme figura 7 abaixo. Neste caso, os endereços 0 ao 5 do inversor de frequência.

Figura 7 – Leitura dos parâmetros do inversor de frequência no simulador Modbus Poll



Fonte: Elaborado pelo autor.

Para leitura das informações do inversor no ESP32, foi utilizado a biblioteca Modbus Master disponível no repositório público do Github (MELIS, 2019).

Porém, durante sua utilização, não foi obtido êxito na leitura dos dados. Com isso, houve a necessidade de adequar o programa, para manter as variáveis na memória. Foram criados então, quatro registradores para salvar as informações que estavam sendo lidas pelo mestre, e assim gerar um padrão de leitura correta. Houve também a necessidade de inversão dos bytes devido à inversão que é feito na leitura das variáveis. Na figura 8, podemos verificar o código-fonte da leitura Modbus.

Figura 8 – Código-fonte da leitura Modbus

```
  Serial1.begin(9600, SERIAL_8N1, 17, 4, true);  // Modbus connection

  modbus.onData([](uint8_t serverAddress, esp32Modbus::FunctionCode fc, uint16_t address, uint8_t* data, size_t length) {
    Serial.printf("\n id 0x%02x fc 0x%02x len %u: 0x", serverAddress, fc, length);

    for (size_t i = 0; i < length; ++i) {
      Serial.printf("%02x", data[i]);
    }
    for (size_t j=0; j< (int)length/2; ++j) {
    reg[j] = data[j*2]*256+data[j*2+1];
    }
  });
  modbus.onError([](esp32Modbus::Error error) {
    Serial.printf("error: 0x%02x\n\n", static_cast<uint8_t>(error));
  });
  modbus.begin();
}
```

Fonte: Elaborado pelo autor.

3.3 Envio dos dados via MQTT

A conexão com a Internet é realizada pelo ESP32 por meio da rede Wi-fi. O dispositivo se cadastra na rede por meio do Service Set Identifier (SSID) e senha. Foi programado para que ele realize tentativas de conexão até obter sucesso ou esgotar o tempo determinado. Após a conexão, os parâmetros de conexão TCP/IP são configuradas por meio do servidor DHCP.

O segundo passo será a conexão com o serviço broker MQTT. O protocolo MQTT fornece um método leve de execução de mensagens usando um modelo de publicação/assinatura (publisher/subscriber). Isso o torna adequado para mensagens de dispositivos IoT, como sensores de baixa potência e dispositivos móveis - telefones, computadores integrados (SBC – Single Board Computers) e microcontroladores (MCU) (ECLIPSE FOUNDATION, 2021).

Figura 9 – Código-fonte da conexão MQTT com broker

```
void conectaMQTT() {
    while (!MQTT.connected()) {
        Serial.print("Conectando ao Broker MQTT: ");
        Serial.println(BROKER_MQTT);
        if (MQTT.connect(ID_MQTT)) {
            Serial.println("Conectado ao Broker com sucesso!");
        }
        else {
            Serial.println("Noo foi possivel se conectar ao broker.");
            Serial.println("Nova tentatica de conexao em 10s");
            delay(10000);
        }
    }
}
```

Fonte: Elaborado pelo autor.

Assim que a conexão é executada, o ESP32 envia uma publicação com um tópico no formato JSON que é um formato baseado em texto padrão para representar dados estruturados com base na sintaxe do objeto JavaScript**.** Na figura 10, podemos observar o código-fonte para o envio da mensagem.

Figura 10 – Código-fonte da publicação da mensagem MQTT

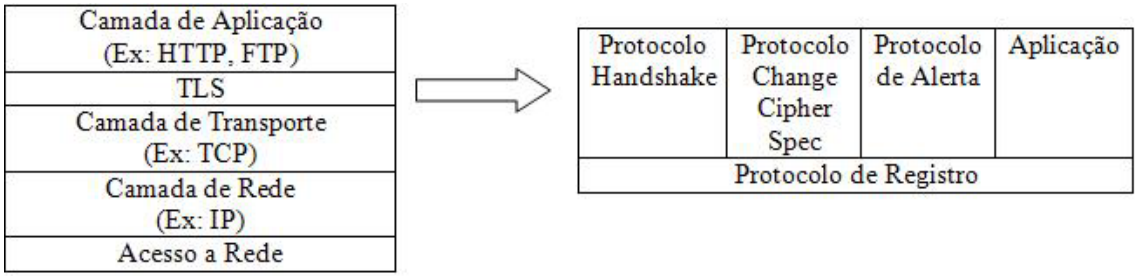
```
MQTT.publish(TOPIC,jsonsaida);
Serial.println("\n publicado valor, Payload enviado." ); //envio das informaçoes MQTT via JSon
```

Fonte: Elaborado pelo autor.

Como alternativa foi utilizado o broker MQTT de plataformas de serviço de nuvem, como a AWS. O broker AWS exige o uso do protocolo de aplicação MQTT com TLS, ou seja, o MQTTS, definido na porta TCP 8883. Desta forma, a solução IoT pode ganhar em segurança, estabilidade, escalabilidade, além da disponibilidade de uso de diversos serviços na plataforma de nuvem.

O protocolo TLS é utilizado para garantir a privacidade e a integridade dos dados em uma comunicação entre as aplicações. O protocolo é composto de duas camadas: o protocolo de Registro, usado para o encapsulamento dos protocolos do nível acima, e os protocolos de *Handshaking*, usados para prver a segurança da conexão, por meio de autenticação entre cliente e servidor e negociação de algoritmos de encriptação antes da transmissão, conforme figura 11.

Figura 11 –Camada de aplicação TLS



Fonte: Coutinho e Silva (2021).

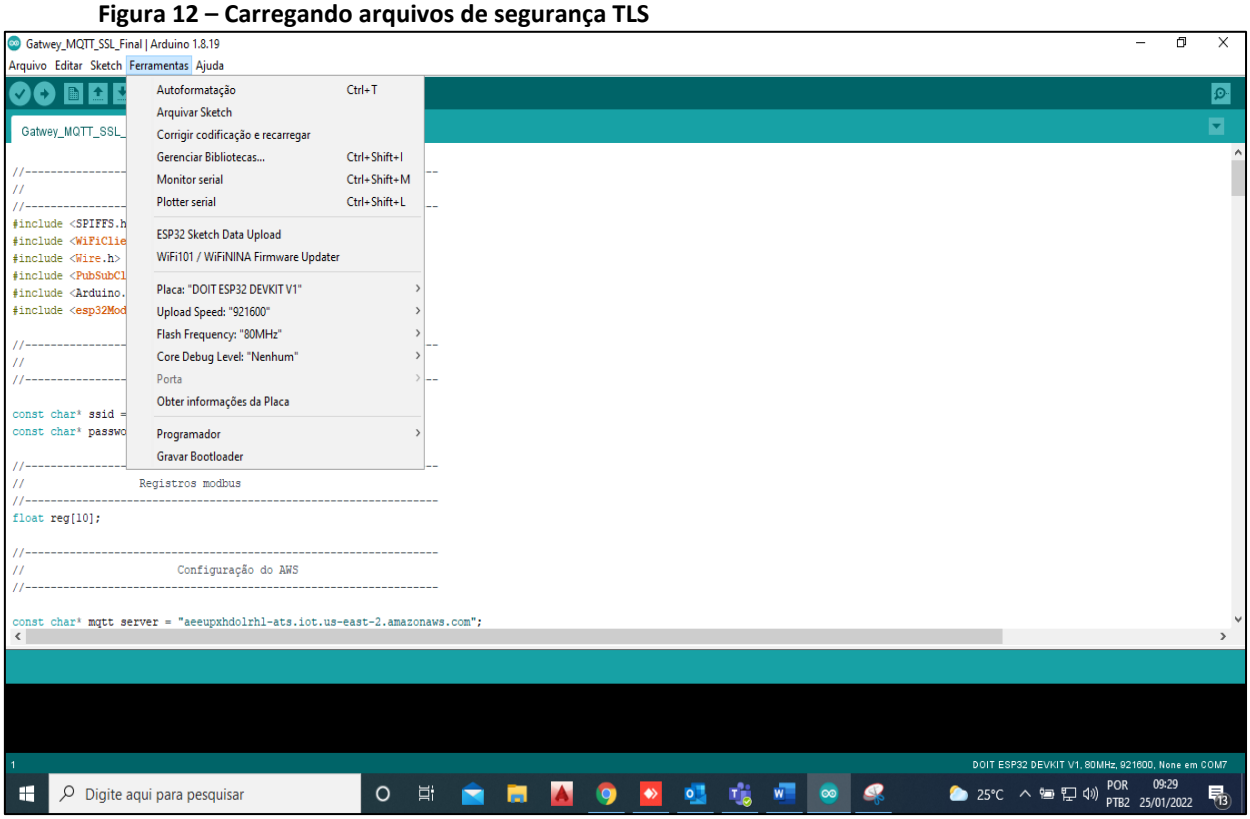
O protocolo TLS fornece duas funções de segurança importantes para uma conexão entre aplicações que usam um protocolo de transporte da suite TCP/IP, como o TCP ou UDP: privacidade dos dados, por meio de criptografia simétrica, sendo que, para cada sessão é gerada uma nova chave negociada por meio do protocolo *Handshake*, e confiabilidade dos dados, por meio de verificação da integridade da mensagem usando a função HASH no HMAC (*Hashing Message Authentication Code*) (COUTINHO; SILVA, 2021).

Importante destacar, que o TLS é independente dos protocolos de aplicação. Desta forma, protocolos de nível acima podem comunicar com o TLS de forma transparente (COUTINHO; SILVA, 2021).

Ao cadastrar o dispositivo no serviço de nuvem AWS IoT Core, é possível gerar e baixar o certificado para o dispositivo para comunicação via MQTTS.

Após a geração do certificado, é necessário carregar para o ESP32 através da aba “Ferramentas” no item “ESP32 Sketch Data Upload”, conforme figura 12.

Figura 12 – Carregando arquivos de segurança TLS



Fonte: Elaborado pelo autor.

Nesse momento, o arquivo do certificado foi carregado para a memória ESP32. Na programação, será necessário a leitura desses arquivos. O primeiro a ser lido será o arquivo de Root CA com a descrição de segurança, conforme figura 13.

Figura 13 – Leitura do arquivo Root CA

```
//---------------------------------------------------------------
//                    Leitura do arquivo Root CA
//---------------------------------------------------------------

File file2 = SPIFFS.open("/AmazonRootCA1.pem","r");
if (!file2) {
Serial.println("Falha ao abrir o arquivo Root CA para leitura");
return;
}
Serial.println("Conteúdo do arquivo Root CA:");
while (file2.available()) {
Read_rootca = file2.readString();
Serial.println(Read_rootca);
}
```

Fonte: Elaborado pelo autor.

O próximo arquivo a ser lido a leitura é o certificado, conforme figura 14.

Figura 14 – Leitura do arquivo de certificado

```
//---------------------------------------------------------------
//                    Leitura do arquivo Cert
//---------------------------------------------------------------

File file4 = SPIFFS.open("/0e135-certificate.pem.crt","r");
if (!file4) {
Serial.println("Falha ao abrir o arquivo Cert para leitura");
return;
}
Serial.println("Conteúdo do arquivo Cert:");
while (file4.available()) {
Read_cert = file4.readString();
Serial.println(Read_cert);
}
```

Fonte: Elaborado pelo autor.

Por fim, o certificado de chave privada deve ser lido, conforme figura 15.

Figura 15 – Leitura do arquivo de chave privada

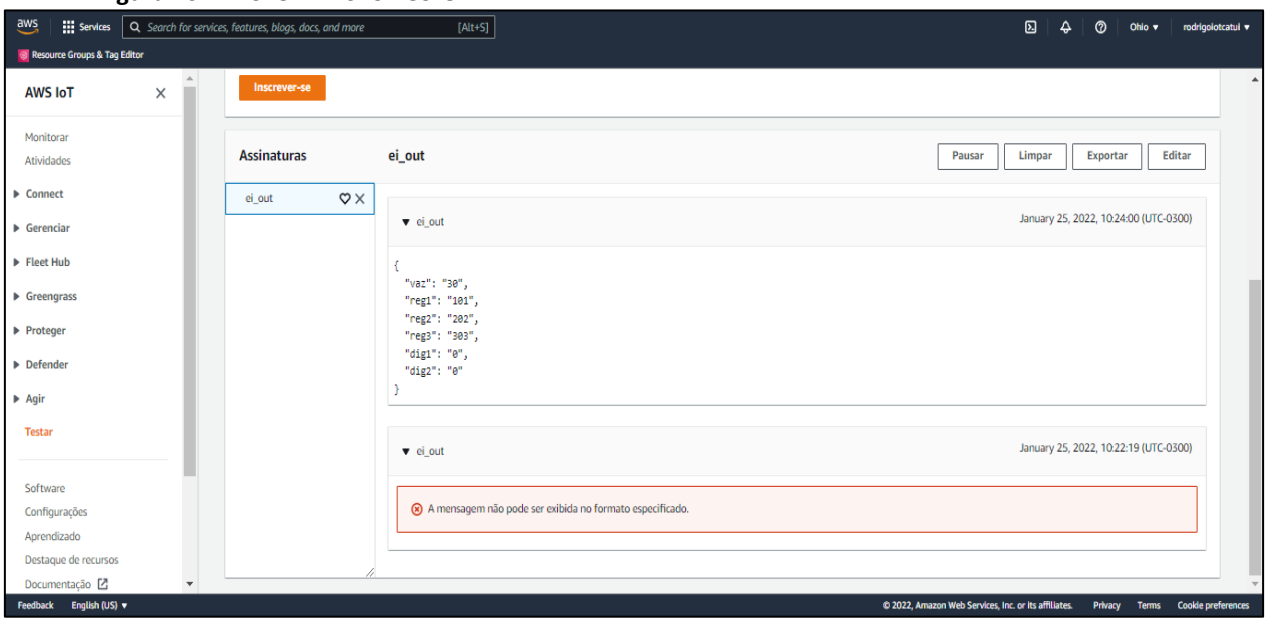
```
//---------------------------------------------------------------
//                    Leitura do arquivo da chave privada
//---------------------------------------------------------------

File file6 = SPIFFS.open("/0e135-private.pem.key","r");
if (!file6) {
Serial.println("Falha ao abrir o arquivo Chave Privada para leitura");
return;
}
Serial.println("Conteúdo do arquivo Chave Privada:");
while (file6.available()) {
Read_privatekey = file6.readString();
Serial.println(Read_privatekey);
}
char* pRead_rootca;
pRead_rootca = (char *)malloc(sizeof(char) * (Read_rootca.length()+1));
strcpy(pRead_rootca, Read_rootca.c_str());
char* pRead_cert;
pRead_cert = (char *)malloc(sizeof(char) * (Read_cert.length()+1));
strcpy(pRead_cert, Read_cert.c_str());
char* pRead_privatekey;
pRead_privatekey = (char *)malloc(sizeof(char) * (Read_privatekey.length()+1));
strcpy(pRead_privatekey, Read_privatekey.c_str());
Serial.println("=========================================================");
Serial.println("Certificados passando para o método espClient");
Serial.println();
Serial.println("Root CA:");
Serial.write(pRead_rootca);
Serial.println("=========================================================");
Serial.println();
Serial.println("Cert:");
Serial.write(pRead_cert);
Serial.println("=========================================================");
Serial.println();
Serial.println("Private Key:");
Serial.write(pRead_privatekey);
Serial.println("=========================================================");

espClient.setCACert(pRead_rootca);
espClient.setCertificate(pRead_cert);
espClient.setPrivateKey(pRead_privatekey);
client.setServer(mqtt_server, mqtt_port);
client.setCallback(callback);
WiFi.macAddress(mac);
snprintf(mac_Id, sizeof(mac_Id), "%02x:%02x:%02x:%02x:%02x:%02x",
mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);
Serial.print(mac_Id);
}
```

Fonte: Elaborado pelo autor.

Desta maneira, está concluído a leitura dos arquivos digitais válidos para estabelecer a publicação dos dados com sistema de segurança TLS via MQTT (MQTTS) para o broker IoT Core da AWS.

Após o envio da mensagem MQTT, podemos utilizar da ferramenta disponível na plataforma AWS IoT Core para verificar o tráfego dessas informações se inscrevendo no tópico de publicação, conforme figura 16.

Figura 16 – Broker AWS IoT Core

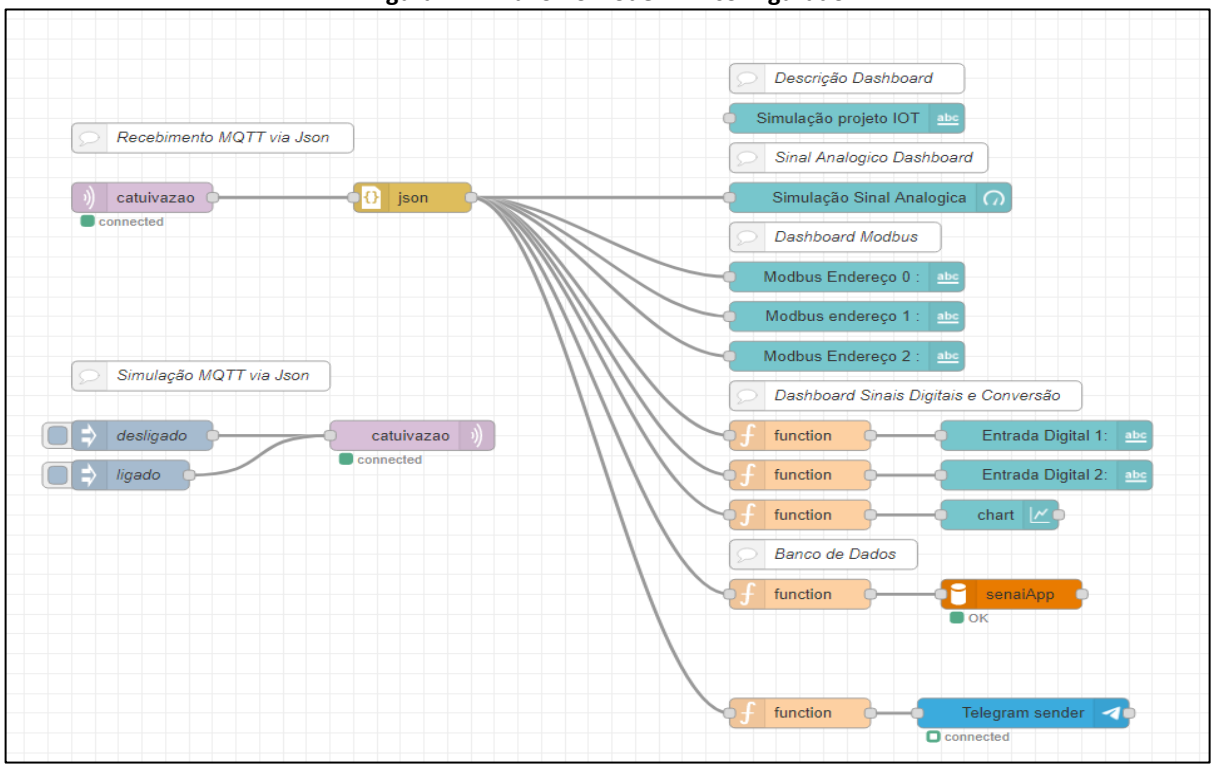


Fonte: Elaborado pelo autor.

Após a coleta dos dados e o envio das informações para o broker MQTT, é utilizado o *Node-RED* para a conexão com banco de dados e sistemas de nuvem, e criação de uma interface com o usuário.

Recebendo os dados via MQTT, o fluxo configurado em Node-RED realiza um tratamento dos dados recebidos e faz a conexão com o banco de dados. Além disso, cria uma interface com o usuário no qual podemos ver em tempo real o status do painel elétrico da estação elevatória de efluentes sanitários. O fluxo no Node-RED pode ser visto na figura 17.

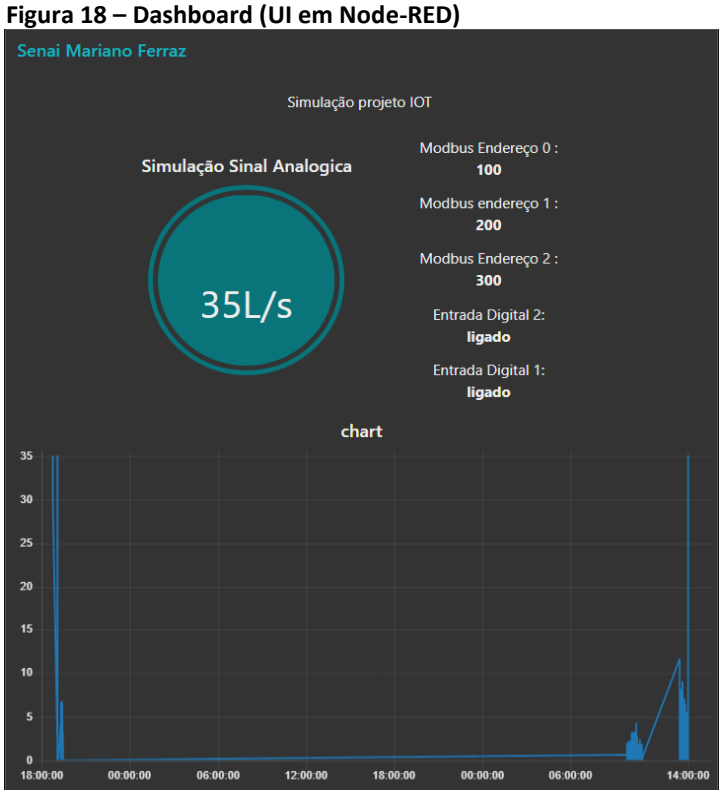
Figura 17 – Fluxo no Node-RED configurado



Fonte: Elaborado pelo autor.

Com os dados direcionados para o banco de dados e para o *dashboard,* o usuário pode visualizar em tempo real as informações coletadas, conforme se observa na figura 18.

Figura 18 – Dashboard (UI em Node-RED)

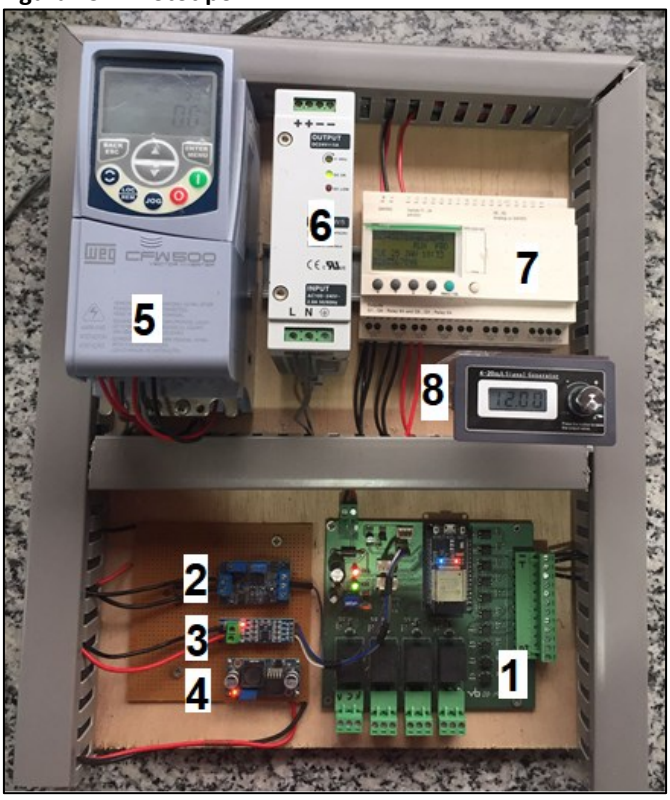


Fonte: Elaborado pelo autor.

4 RESULTADOS

Para a realização dos testes foi montado um protótipo (figura 19). A placa concentradora coleta as informações de um CLP, por meio de sinais digitais. O sinal analógico foi testado utilizando um simulador de sinal 4 a 20 mA que possui um potenciômetro para ajuste do valor desejado. São lidas as informações de tensão, frequência e corrente de um inversor de frequência WEG modelo CFW500, por meio da rede de comunicação Modbus usando a placa de comunicação CFW500-RS485.

Figura 19 – Protótipo



Fonte: Elaborado pelo autor.

Os componentes utilizados no protótipo e numerados na Figura 19 são descritos a seguir:

1. Placa concentradora;
2. Circuito de conversão 4 a 20 mA para 0 a 3,3 VDC;
3. Circuito MAX485 Modbus RTU;
4. Regulador de tensão da placa concentradora;
5. Inversor de Frequência de simulação da rede Modbus;
6. Fonte de alimentação geral;
7. CLP de simulação dos sinais digitais ;
8. Simulador de sinal analógico 4 a 20 mA.

Por meio dos testes realizados, foram atingidos os objetivos propostos em questão, adquirindo assim as informações do CLP através de sinais digitais, sinal analógico e do inversor de frequência através de rede Modbus RTU 485. Os dados foram apresentados em um *dashboard* conforme mostrado anteriormente. Foi utilizado um servidor com um banco de dados local tornando o projeto de baixo custo.

Visando aprimoramentos no sistema IoT, o banco de dados, que atualmente está em uma máquina local, pode ser implantado em um serviço de nuvem, melhorando ainda mais as questões de segurança e estabilidade.

Tendo em vista uma melhor funcionalidade no hardware, pode ser aprimorado a placa concentradora, com a elaboração de uma placa única contemplando todos os componentes, e confeccionando uma placa de circuito impresso comercial, para evitar problemas de ruídos e mau contato.

5 CONSIDERAÇÕES FINAIS

Este trabalho apresentou a construção e aplicação de um sistema IoT para monitoramento de uma estação de tratamento de efluentes sanitários.

O objetivo foi atingido em um sistema IoT com uma placa concentradora capaz de capturar dados de CLP e inversor de frequência por meio de sinais digitais, analógico e rede de comunicação Modbus RTU.

O seu funcionamento é eficaz em redes de Internet com banda limitada devido ao seu funcionamento utilizando protocolo MQTT com JSON, mensagem de formato compacto tornando o sistema mais ágil e com baixa latência.

O protocolo de comunicação MQTT com Node-RED trouxe benefícios para a IoT. O MQTT é um protocolo leve, permitindo que os equipamentos IoT se comuniquem de forma rápida e simplificada. Ainda permite a utilização do protocolo TLS para mitigar os riscos de segurança dos dados em trânsito.

O Node-RED é uma ferramenta com conceito de programação gráfica (*No-Code*), de fácil uso e aprendizado, para conectar dispositivos e serviços locais e em nuvem utilizando APIs.

Sendo assim, o sistema proposto oferece compartilhamento de informações, padronização de comunicação entre diversos equipamentos concentrando em um equipamento local, e fácil visualização dos dados em um *dashboard*. A implantação se mostrou de baixo custo, altamente flexível para novas funcionalidades e escalável, em especial ao se utilizar os serviços em nuvem.

Nos próximos passos, pretende-se inserir essa aplicação em um cenário industrial, para estudo em uma aplicação real, fazendo a instalação e acompanhamento em períodos maiores para verificação do comportamento da aplicação, com a aplicação sendo confiável E, possivelmente, utilizar não somente em uma aplicação de monitoramento, mas também para para o controle do processo dentro do sistema como um todo, substituindo o CLP tornando ainda mais flexível a aplicação tornando em um produto com alto valor agregado.

REFERÊNCIAS

BECKER, Ansgar. **HEIDISQL.** 2021. Disponível em: https://www:heidisql:com. Acesso em: 31 ago. 2021.

COUTINHO, Gustavo Lacerda; SILVA, Renan Galvão Machado e. **Funcionamento do TLS.** UFRJ. 2021. Disponível em: https://www.gta.ufrj.br/grad/06_1/ssl/func_tls.htm. Acesso em: 22 dez. 2021.

ECLIPSE FOUNDATION. **Mosquitto**. 2021. Disponível em: https://test.mosquitto.org/. Acesso em: 13 out. 2021.

HIVEMQ TEAM. **MQTT Security Fundamentals**. 2015. Disponível em: https://www.hivemq.com/blog/mqtt-security-fundamentals-tls-ssl/. Acesso em: 04 set. 2021.

MELIS, Bert. **Esp32ModbusRTU**: Modbus RTU client for ESP32. 2019. Disponível em: https://github.com/bertmelis/esp32ModbusRTU. Acesso em: 20 set. 2021.

OPEN JS FOUNDATION. **Node-RED:** low-code programming for event-driven applications. 2021. Disponível em: https://nodered.org/. Acesso em: 13 out. 2021.

POSEY, Brien; ROSENCRANCE, Linda; SHEA, Sharon. **Definition industrial internet of things (IIoT)**. 2021. Disponível em: https://internetofthingsagenda.techtarget.com/definition/Industrial-Internet-of-Things-IIoT. Acesso em: 31 ago. 2021.

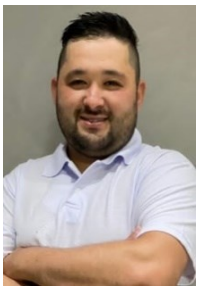
TRACEY, Lenore. **Modbus Organization replaces Master-Slave with Client-Server**. 2021. Disponível em: https://www.modbus.org/docs/Client-ServerPR-07-2020-final.docx.pdf. Acesso em: 09 nov. 2021.

WEG. **Inversor de frequência CFW500**: manual de programação. 2016. Disponível em: https://static.weg.net/medias/downloadcenter/hc4/hd3/WEG-cfw500-manual-de-programacao-10001469555-1.1x-manual-portugues-br.pdf. Acesso em: 10 nov. 2021.

SOBRE O(S) AUTOR(ES)

_____

[i] **RODRIGO DE SOUSA BOTELHO**



Possui pós-graduação em Internet das Coisas pela Faculdade SENAI São Paulo – Campus Mariano Ferraz (2022) e graduação em Automação industrial pela Faculdade Flamingo (2015). Tem experiência na área de Automação Industrial, com ênfase em projetos, manutenção e programação de controladores industriais. Atualmente, é supervisor na empresa Catui Engenharia, sendo responsável pelos setores da elétrica, automação e programação de softwares. https://orcid.org/0009-0005-8503-4385

[ii] **DANIEL BARBUTO ROSSATO**

Possui doutorado em Engenharia Eletrônica e Computação pelo ITA (2022), mestrado em Engenharia Elétrica – Sistemas pela EPUSP (2009) e bacharelado em Engenharia Elétrica – Automação e Controle pela EPUSP (2002). Atualmente é docente na Faculdade SENAI São Paulo – Campus Mariano Ferraz, atuando nas áreas de Sistemas e Controle, Internet das Coisas, Redes Neurais Artificiais, Robótica e Segurança. Tem experiência em manutenção em sistemas de automação industrial. https://orcid.org/0000-0003-1654-3424

[iii] **ANDRÉ LUIS DOS SANTOS**

Possui mestrado em Engenharia Mecânica pela Universidade de São Paulo (2016) e graduação em Engenharia Mecatrônica pela Universidade Paulista (2001). Atualmente é docente na Faculdade de Tecnologia SENAI São Paulo – Campus Mariano Ferraz. Tem experiência em automação industrial e desenvolvimento de software, atuando nos temas: processamento de sinais, tomografia por impedância elétrica, redes de comunicação e microcontroladores. https://orcid.org/0000-0001-6627-3886

[iv] **CAIO VINÍCIUS RIBEIRO DA SILVA**

Possui pós-graduação em Automação Industrial pela Faculdade SENAI de Mecatrônica (2017), pós-graduação em Inovação e Competitividade Industrial pela Faculdade SENAI "Theobaldo de Nigris" (2021) e graduação em Tecnologia em Eletrônica Industrial pela Faculdade de Tecnologia SENAI Anchieta (2012). Atualmente, é docente e coordenador de estágios na Faculdade de Tecnologia SENAI São Paulo – Campus Mariano Ferraz, atuando em Internet das Coisas e Tecnologia da Informação. https://orcid.org/0000-0002-9421-2471