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

JOSÉ ANACLETO DA SILVA BELO JÚNIOR

# MOBILE VISION: Monitoramento de controladores lógico programáveis e de suas variáveis em plataforma mobile

Recife
2023

JOSÉ ANACLETO DA SILVA BELO JÚNIOR

# MOBILE VISION: Monitoramento de controladores lógico programáveis e de suas variáveis em plataforma mobile

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
Coorientador: Prof. Dr. Davidson da Costa Marques

Recife
2023

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

Belo Júnior, José Anacleto da Silva.
Mobile Vision: Monitoramento de controladores lógico programáveis e de suas variáveis em plataforma mobile / José Anacleto da Silva Belo Júnior. - Recife, 2023.
109p. : il., tab.

Orientador(a): Douglas Contente Pimentel Barbosa
Cooorientador(a): Davidson da Costa Marques
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia de Controle e Automação - Bacharelado, 2023.

1. Aplicativo. 2. CLP. 3. IIoT. 4. Modbus. 5. MQTT. I. Barbosa, Douglas Contente Pimentel. (Orientação). II. Marques, Davidson da Costa. (Coorientação). IV. Título.

620 CDD (22.ed.)

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: 02/10/2023.

**BANCA EXAMINADORA**

_____
Prof. Dr. Douglas Contente Pimentel Barbosa (Orientador)
Universidade Federal de Pernambuco

_____
Prof. Dr. Davidson da Costa Marques (Coorientador)
Universidade Federal de Pernambuco

_____
Prof. Dr. Herbert Albérico de Sá Leitão (Examinador Interno)
Universidade Federal de Pernambuco

_____
Prof. M.Sc. Geraldo Leite Maia Júnior (Examinador Interno)
Universidade Federal de Pernambuco

“Porque D’Ele por Ele e para Ele são todas as coisas.” (Rm 11.36a).

# AGRADECIMENTOS

Primeiramente utilizo este espaço para agradecer a Deus, que mesmo nos momentos mais difíceis foi meu lugar de refúgio e abrigo. Ele é e sempre será minha fonte de esperança. Agradeço a Ele por ter me redimido e me ajudar em todo o tempo. Tenho certeza de que todos os sonhos que alcançar em minha vida terão a mão e ajuda d'Ele.

Gostaria de agradecer a meu pai José Anacleto e minha mãe Lúcia pelo apoio incondicional que dedicaram a mim durante toda a minha trajetória. Junto de vocês obtive grandes ensinamentos e grandes histórias. Agradeço todos os dias a Deus por ter coloca-los em minha vida. Com a bênção de Deus serão muitos dias pela frente.

Agradeço a meu irmão Paulo que me ajudou e tem me ajudado muito nessa caminhada. Muito obrigado pelo companheirismo, conselhos e momentos de alegria que compartilhamos. Com certeza não é coincidência estarmos crescendo juntos e aprendendo um com o outro. Que possamos evoluir cada dia mais um com o outro.

Agradeço a meu orientador, o professor Dr. Douglas Contente. O compartilhar conhecimento é uma missão especial. Pude ver a sua inteligência e preparo para tal missão. Muito obrigado pela oportunidade de ser seu orientado.

Agradeço a meu coorientador, o professor Dr. Davidson Marques. Muito obrigado por aceitar esse desafio. Muito obrigado por compartilhar o seu conhecimento e por estar sempre disponível a ajudar. Muito obrigado por ser seu orientado e por ter acreditado desde o início nesse projeto e persistido nele.

Gostaria de agradecer a todos os professores do DEE com quem tive oportunidade de obter conhecimento. Com certeza todos vocês têm contribuição no fato de ter chegado até aqui. Muito obrigado por tudo!

Gostaria de agradecer aos técnicos e demais funcionários. A missão de vocês é grandiosa dentro do departamento e da universidade. Muito obrigado pela disponibilidade em todo esse ciclo.

Por fim, gostaria de agradecer aos alunos do DEE. Cada interação com cada pessoa foi importantíssima para gerar conhecimento e evolução. Desde já meu muito obrigado!

Grandes coisas fez o Senhor por nós, por isso estamos alegres. (Salmos 126.3 - ARC)

## RESUMO

A utilização do conceito de IoT (Internet of Things) dentro do ambiente industrial forneceu novas oportunidades à indústria. A conectividade entre dispositivos, associada ao alto processamento de dados permitiu a otimização da produção e de custos mediante a prevenção de falhas, identificação de defeitos e customização da produção; tornando a operação industrial mais rentável e afetando positivamente na qualidade final dos produtos gerados. O desenvolvimento de aplicações envolvendo IoT na indústria também alcançou a área da segurança do trabalho, permitindo a formação de um ambiente industrial mais seguro, reduzindo a possibilidade de acidentes a seus colaboradores. O desenvolvimento de novas tecnologias utilizando o conceito alteraram o relacionamento entre o processo e sua gestão, tornando possíveis uma gestão remota e distribuída do processo de maneira mais acessível. Os crescentes investimentos na área demandam por aplicações que possam transformar números em informações para promover ações e tomadas de decisão mais eficazes, permitindo uma gestão de processo mais simples, mitigando possíveis problemas. Dentre as possibilidades que a IoT aplicada à indústria fornece, este trabalho apresenta o desenvolvimento de uma aplicação para smartphone com o objetivo de realizar o monitoramento de variáveis de processo armazenadas em um controlador lógico programável, utilizando linguagens de programação *open source e* protocolos de comunicação abertos de forma a propiciar uma solução de baixo custo envolvendo IIoT (Industrial Internet of Things).



**Palavras-chave:** Aplicativo; CLP; IIoT; Modbus; MQTT;

## ABSTRACT

The use of the Internet of Things (IoT) concepts within the industrial environment has provided new opportunities to the industry. The connectivity between devices coupled with high data processing capabilities has enabled the optimization of production and cost efficiency through fault prevention, defect identification, and production customization, rendering industrial operations more profitable and positively impacting the final quality of generated products. The development of IoT applications in the industry has also extended to the realm of occupational safety, allowing for the creation of a safer industrial environment and reducing the likelihood of accidents for its employees. The development of new technologies using this concept has transformed the relationship between processes and their management, enabling a more accessible remote and distributed process management. The increasing investments in this field demand applications that can convert data into actionable insights to facilitate more effective actions and decision-making, enabling simpler process management and mitigating potential issues. Among the possibilities that IoT applied to the industry offers, this work presents the development of a smartphone application aimed at monitoring process variables stored in a programmable logic controller. This is achieved using open-source programming languages and open communication protocols to provide a low-cost solution involving Industrial Internet of Things (IIoT).



**Keywords:** App; IIoT; Modbus; MQTT; PLC.

# LISTA DE ILUSTRAÇÕES

# LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| ADU | *Application Data Unity* |
| API | *Application Programming Interface* |
| APK | *Android Application Package* |
| CLP | Controlador Lógico Programável |
| CRC | *Cyclic Redundancy Check* |
| HTML | *Hypertext Markup Language* |
| HTTP | *Hypertext Transfer Protocol* |
| HTTPS | *Secure Hypertext Transfer Protocol* |
| ICANN | *Inernet Corporation for Assigned Names and Numbers* |
| IEEE | *Institute of Electrical and Electronic Engineers* |
| IHM | Interface Homem Máquina |
| IIoT | *Industrial Internet of Things* |
| IoT | *Internet of Things* |
| IP | *Internet Protocol* |
| JSON | *JavaScript Object Notation* |
| LAN | *Local Area Network* |
| MAC | *Media Access Control* |
| MODICON | *Modular Digital Controller* |
| MQTT | *Message Queuing Telemetry Transport* |
| M2M | *Machine to Machine* |
| OSI | *Open Systems Interconnection* |
| PDU | *Protocol Data Unity* |
| PHP | *Hypertext Preprocessor* |
| PyPi | *Python Package Index* |
| QoS | *Quality of Service* |
| REST | *Representational State Transfer* |
| SMTP | *Simple Mail Transfer Protocol* |
| SQL | *Structured Query Language* |
| SSL | *Secure Sockets Layer* |
| TCP | *Transmission Control Protocol* |
| TLS | *Transport Layer Security* |

## LISTA DE SÍMBOLOS

| | |
|---|---|
| $B$ | Byte |
| $I$ | Registrador de Entrada |
| $M$ | Registrador de Memória |
| $Q$ | Registrador de Saída |
| $W$ | Word |
| $\rightarrow$ | Transformação de endereçamentos |

## SUMÁRIO

# 1 INTRODUÇÃO

A Internet das Coisas (IoT) consiste em uma rede de dispositivos físicos, aparelhos e outros objetos físicos que são incorporados com sensores, software, serviços de rede e outras funcionalidades para realização de coleta e compartilhamento de dados [1,2]. IIoT (Industrial Internet of Things) refere-se à aplicação da tecnologia IoT em ambientes industriais, principalmente mediante utilização de sensores e dispositivos compatíveis com tecnologias *cloud*. Este conceito possibilita a coleta e transmissão de dados, a execução de análises e a otimização da produção por meio do aumento da eficiência e redução dos custos de produção [2,3].

De acordo com [3], em 2019 foram investidos mais de 300 bilhões de Dólares em IIoT e a projeção é de que até 2025 esse investimento dobre. Segundo [3], é estimado que o número de dispositivos que utilizam o conceito IoT chega a 30 bilhões. Dentre as várias possibilidades que a utilização da IIoT disponibiliza está o monitoramento de processos de produção dentro das indústrias.

As fábricas podem aumentar sua competitividade ao utilizar o monitoramento da linha de produção para permitir a realização de manutenções preditivas em equipamentos. Mediante a utilização de alertas de sensor, pode-se identificar falhas iminentes em equipamentos utilizados nas linhas de produção, possibilitando assim uma manutenção ou troca do aparelho com menor impacto à produção, reduzindo custos operacionais e elevando o tempo de atividade, o nível de gerenciamento e produtividade na linha [2].

A utilização de conceitos IIoT dentro de uma linha de produção industrial pode permitir um aumento significativo na praticidade de acesso aos dados por parte de gestores, permitindo um acompanhamento de processos em tempo real, à longa distância e sem investimentos elevados. Com um investimento mínimo, é possível configurar um smartphone conectado à nuvem para monitorar os processos de fabricação de praticamente qualquer lugar [4].

Mediante ao elevado crescimento da utilização de ferramentas que utilizam o conceito IoT dentro da indústria e projeções positivas que o envolve, se faz necessário o desenvolvimento de aplicações que o utilizem. Dentro das vastas possibilidades da

utilização do conceito de IoT dentro da indústria, este trabalho concentra seus esforços na elaboração de um aplicativo para monitoramento de variáveis em um processo de produção através de um smartphone.

## 1.1 Objetivos

A execução deste trabalho foi norteada pelo cumprimento de objetivos previamente traçados. As seções 1.1.1 e 1.1.2 apresentam os objetivos geral e específicos referentes ao mesmo, respectivamente.

### *1.1.1 Geral*

Apresentar uma solução para o monitoramento de dados armazenados em um controlador lógico programável (CLP) através de uma plataforma móvel.

### *1.1.2 Específicos*

- Produzir um sistema de monitoramento de dados presentes no CLP S7-1200 sem a necessidade de adicionar hardwares ao chão de fábrica;
- Configurar um controlador lógico programável (CLP) S7-1200 para comunicar-se com meio externo utilizando protocolo MODBUS;
- Desenvolver um software para computador que possibilite a captura de dados presentes em um controlador lógico programável Siemens S7-1200;
- Desenvolver um aplicativo que possa ser utilizado na plataforma Android capaz de receber atualizações de dados em tempo real, de forma a se apresentar como uma opção de baixo custo para monitoramento de processos industriais;
- Promover uma aplicação que interaja com o usuário de maneira amigável, fornecendo a possibilidade de customização e fácil compreensão de conteúdo a ele;

- Criar um sistema de cadastro, login e recuperação de senha para aplicações tanto em dispositivos móveis quanto em computadores;

## 1.2 Organização do Trabalho

No início do capítulo 1 é realizada uma apresentação sobre o tema pertencente ao trabalho desenvolvido. No referido capítulo também são apresentados os objetivos que norteiam a execução dele.

No capítulo 2 são apresentados os fundamentos que permitiram o desenvolvimento e conclusão deste trabalho. Neste capítulo são fornecidas informações sobre as ferramentas utilizadas na execução do mesmo.

No capítulo 3 é apresentada a construção do trabalho mediante apresentação metodológica de como cada ação foi realizada e pensada nele. No capítulo 4 são apresentados os resultados obtidos após a utilização da metodologia exposta em uma situação idealizada. Os resultados são apresentados no capítulo separados em seções para melhor entendimento do processo como um todo.

Por fim, no capítulo 5, é apresentada uma conclusão, tomando como base os resultados obtidos e os objetivos propostos no capítulo 1. No referido capítulo também são apresentadas propostas de continuidade para o trabalho.

# 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentadas conceitualmente as ferramentas utilizadas na realização deste trabalho. Na seção 2.1 são apresentados os controladores lógico programáveis. Na seção 2.2 são apresentados os servidores locais. Na seção 2.3 são apresentados os protocolos de comunicação e conceitos relacionados. Na seção 2.4 é apresentada a linguagem de programação Python e, por fim, na seção 2.5 são apresentados conceitualmente os bancos de dados e o *Realtime Database* do *Google Firebase*.

## 2.1 Controladores Lógico Programáveis

O controlador lógico programável (CLP) é um tipo de computador industrial no qual podem ser implementadas funções de controle, mediante uso de programação, com o objetivo de atuar em processos e dispositivos. Eles são a tecnologia dedicada a controle de processos mais empregada em ambientes industriais [5,6].

Os controladores lógico programáveis foram idealizados no final da década de 1960 com o objetivo de promover uma solução mais prática e versátil para a automação e controle de plantas industriais se comparadas com as implementações difundidas na época (onde se utilizava principalmente grandes painéis de relés e contatores para introduzir lógicas de controle). O desenvolvimento de uma tecnologia que possibilitasse uma mudança nos processos das linhas de produção mais rapidamente, que fosse mais robusta e que fosse menor que os quadros de relés e contatores utilizados impulsionou a criação deste tipo de controlador [6].

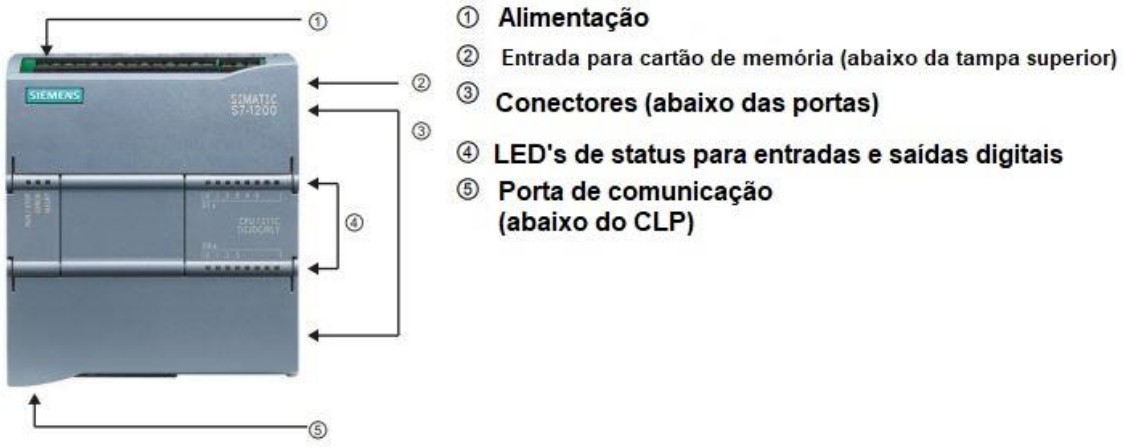
De acordo com [6], no final da década de 1960 foi produzido e comercializado um dispositivo com esta finalidade, chamado de MODICON (*Modular Digital Controller*), considerado o precursor do CLP. Desde então, foram produzidos ao longo do tempo diferentes modelos de CLP por diversos fabricantes para suprir as mais distintas necessidades. Dentre esta elevada variedade de modelos produzidos e comercializados, estão os pertencentes à série S7-1200 produzida pela empresa alemã Siemens.

### *2.1.1 A Série S7-1200*

A série S7-1200 é uma linha de CLPs com vasta aplicabilidade na área de automação [7]. Entre os controladores presentes na série S7-1200 da Siemens está o CLP S7-1200 DC/DC/DC (6ES7 214-1AG31-0XB0), utilizado neste trabalho.

Segundo [7], o referido CLP deve ser alimentado a uma tensão contínua entre 20.4 V e 28.8 V para operações acima de 0 ºC ou entre 22 V e 28.8 V para operação entre -20 ºC e 0 ºC. O controlador citado disponibiliza 14 entradas digitais, 10 saídas digitais (sendo elas transistorizadas) e 2 entradas analógicas. Ainda há a possibilidade de inclusão de módulos adicionais. Módulos adicionais são componentes de hardware que, ao serem adicionados, fornecem uma funcionalidade não existente nativamente ou estendem o uso de uma funcionalidade existente. O CLP utilizado no trabalho conta com uma placa de sinais SB 1232 AQ 1 x 12 bits (6ES7 232-4HA30-0XB0), responsável por disponibilizar uma saída analógica de resolução de 12 bits que trabalha entre -10 V e 10 V, quando opera utilizando níveis de tensão, ou entre 0 e 20 mA, quando a saída envolve corrente elétrica (utilizando neste caso uma resolução de 11 bits), com a possibilidade de configuração para operação entre 4 e 20 mA [7]. A Figura 1 apresenta a imagem do CLP utilizado no trabalho.

Figura 1 - CLP S7-1200




Fonte: Adaptado de [7].

O CLP S7-1200 escolhido possui uma interface PROFINET integrada que pode ser utilizada para programação do mesmo e para comunicação com Interfaces Homem Máquina (IHMs). Dentre os possíveis dispositivos que podem se comunicar com o CLP estão o servidor local.

## 2.2 Servidor local

De acordo com [8], um servidor é um computador que possui capacidade de processamento mais elevada que a de computadores de uso pessoais e que possui como função dentro de uma rede disponibilizar serviços a mesma. De acordo com [8], uma rede de computadores (ou rede, no contexto desse trabalho) diz respeito a um conjunto de computadores independentes conectados entre si por uma única tecnologia.

Segundo [8], os servidores podem ser classificados em dois tipos diferentes de acordo com a função que desempenham em uma rede. Podem ser dedicados ou não dedicados. Os servidores dedicados são aqueles que são utilizados apenas em tarefas de rede (como receber solicitações de computadores e respondê-las) enquanto os servidores não dedicados podem atuar tanto estações (computadores de uso pessoais) quanto como servidores dentro da rede.

De acordo com [9, 10], uma rede local (*Local Area Network* – LAN) é uma rede de caráter privado que atua no interior e nas proximidades de uma determinada dependência (como uma casa, uma indústria entre outros). Desta forma, um servidor local é um computador que possui alta capacidade de processamento, responsável por fornecer serviços aos dispositivos que estão conectados a ele em uma rede particular.

Uma vez que existem dispositivos que prestam serviços a outros e dispositivos que se utilizam de serviços de outros em uma rede de computadores, se faz necessário que haja uma interação entre eles de forma que o objetivo final de ambos seja alcançado. Para que essa interação seja bem sucedida, uma série de processos devem ser realizados desde a criação do conteúdo da comunicação, transmissão dele e posterior leitura deste conteúdo por parte do destinatário. Esses processos são

divididos em camadas dentro de uma rede e utilizam regras bem definidas de padronização, os chamados protocolos de comunicação, apresentados na seção 2.3.

## 2.3 Protocolos de comunicação

Um protocolo de comunicação é um conjunto de tratativas utilizado entre duas entidades pares de mesma camada para definir formatação e significado de mensagens e pacotes entre elas. Os protocolos de comunicação estão organizados dentro de uma rede de acordo com a função que executam, ou seja, de acordo com a camada de rede a qual fazem parte. O modelo de referência OSI (*Open Systems Interconnection*), estrutura em 7 diferentes camadas, sendo elas da mais baixa à mais alta: camada física, camada de enlace, camada de rede, camada de transporte, camada de sessão, camada de apresentação e camada de aplicação (apresentadas em pilha na Figura 2). Em cada camada são utilizadas um protocolo diferente a depender da arquitetura de rede adotada (ou seja, do conjunto de camadas e protocolos que serão utilizados na aplicação) [9].

Figura 2 - Estruturação em camadas do modelo de referência OSI


Fonte: Retirado de [8].

De acordo com [9], a camada física trata da transmissão de bits através de um canal de comunicação pelo qual os dados são transmitidos entre transmissor e

destinatário. A camada de enlace é responsável por transformar um canal de transmissão suscetível a problemas em uma linha livre de erros (com isso detectar e corrigir possíveis erros de transmissão e verificar a integração dos dados enviados e recebidos). A camada de rede é responsável por operações ligadas à sub-rede (tais como o caminho que a mensagem percorrerá até chegar ao destino, processo chamado roteamento; a identificação de um dispositivo dentro de uma rede, processo chamado de endereçamento, controle de tráfego de mensagens, entre outras questões) [9].

A função da camada de transporte é receber dados da camada imediatamente superior a ela, segmentá-los em pacotes menores e repassar esses pacotes para a camada de rede (em uma situação de envio). Para o caso de recebimento de mensagens, a camada de transporte é responsável por remontar os dados segmentados no envio e repassar a mensagem remontada para a camada superior a ela [9,10].

A camada de sessão é responsável por permitir a abertura de sessões de comunicação e realizar o gerenciamento delas [9,10]. A camada de apresentação é responsável por traduzir as informações que ela recebe de forma que as mensagens trocadas possam ser utilizadas corretamente pela camada de aplicação de cada participante da comunicação [9,10].

Por fim, a camada de aplicação, mais elevada do modelo OSI e também a mais próxima do usuário neste modelo, tem a função de disponibilizar acesso a serviços de rede para aplicações utilizadas pelo usuário. A referida camada não presta serviços a nenhuma camada OSI, mas se utiliza dos serviços das camadas inferiores a ela para prestar serviços a aplicações que atuam externamente a este modelo [8,10].

A depender da aplicação envolvida, podem ser realizadas ações envolvendo todas as camadas de rede ou não. Tudo depende de como são estruturadas a utilização dos protocolos dentro da rede na aplicação em questão. As seções de 2.3.1 a 2.3.7 apresentam os protocolos de comunicação utilizados neste trabalho e sua atuação.

### *2.3.1 Protocolo Modbus*

Modbus é um protocolo de comunicação pertencente à camada de aplicação (camada 7 do modelo de referência OSI). Ele fornece comunicação entre dispositivos mediante utilização da arquitetura cliente/servidor, na qual a comunicação é realizada através de ações de solicitação e resposta [11]. Os serviços disponibilizados pelo Modbus consistem em ações que compreendem desde leituras e escritas de registradores no servidor a testes de erro. Essas funcionalidades são disponibilizadas pelo protocolo mediante códigos de funções [11]. É utilizada comumente a expressão função Modbus para descrever um serviço disponibilizado pelo Modbus.

A estrutura de comunicação no modelo cliente/servidor (utilizada no protocolo Modbus e citada anteriormente) se caracteriza pela hierarquia entre um dispositivo que realiza requisições (chamado de cliente) e outro dispositivo que as executa e responde (chamado de servidor), havendo assim um fluxo de requisições de maneira unidirecional dentro de uma determinada comunicação. Portanto, a posição de cada participante da comunicação deve ser previamente definida [11].

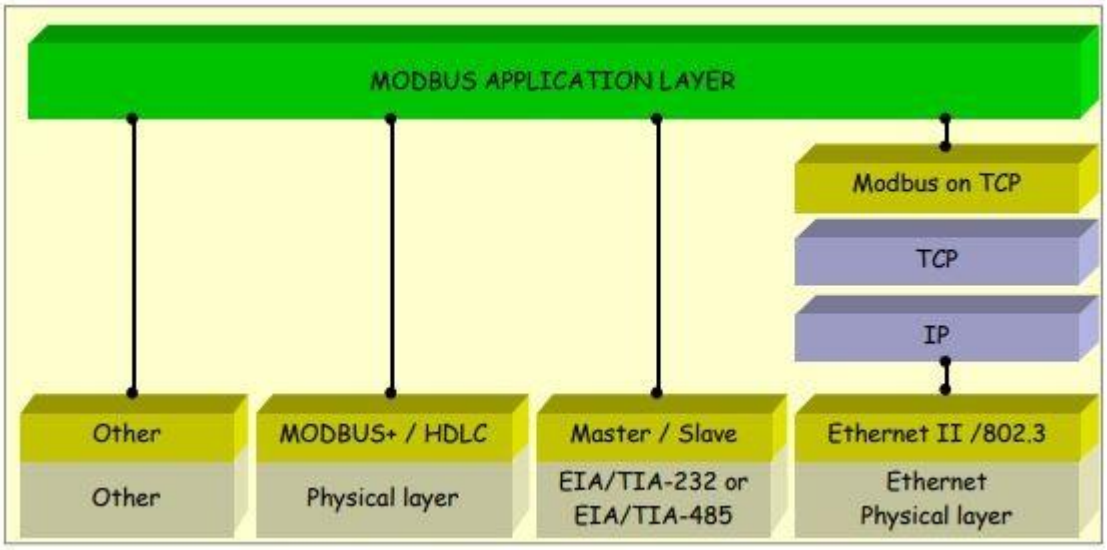
O protocolo Modbus realiza uma classificação de dados em quatro tipos básicos, sendo eles: *Discrete inputs*, *Discrete Coils*, *Input Registers* e *Holding Registers*. A associação desses tipos de dados a registradores dentro dos dispositivos é realizada de forma independente por cada fabricante e, consequentemente, pode variar a depender do hardware envolvido na utilização (inclusive com um determinado grupo de registradores podendo ser associado a mais de um tipo básico de dados), sendo a forma mais comum de associação: *Discrete Inputs* com entradas digitais; *Discrete Coils* com saídas digitais; *Input Registers* com entradas analógicas, e, por fim, *Holding Registers* com saídas analógicas ou posições de memória [11].

É possível dentro do Modbus endereçar até 65536 registradores em cada tipo básico de dados, cabendo ao desenvolvedor do dispositivo a tarefa de associar os registradores físicos do dispositivo desenvolvido a um endereço Modbus dentro de um determinado grupo. Em um determinado dispositivo, podem ou não existir todos os 65536 registradores de cada tipo básico de dados, a depender da quantidade de registradores presentes em sua memória e da implementação realizada pelo fabricante nesta associação. O endereçamento Modbus de cada um dos registradores é realizado dentro da trama de comunicação iniciando do 0 e indo até 65535 (0 a

FFFF, em hexadecimal). Esta classificação associa o endereço de número 0 ao primeiro registrador do grupo, o de número 1 ao segundo registrador, e assim sucessivamente, até o último registrador [11].

Como apresentado na Figura 3, o protocolo Modbus pode ser utilizado dentro de diversas arquiteturas de rede. Uma delas consiste na utilização do protocolo Modbus utilizando um padrão de rede Ethernet, usando o protocolo TCP no transporte de dados. Esta implementação do protocolo Modbus é chamada de Modbus TCP [12].

Figura 3 - Utilizações do protocolo Modbus em variadas arquiteturas de rede.



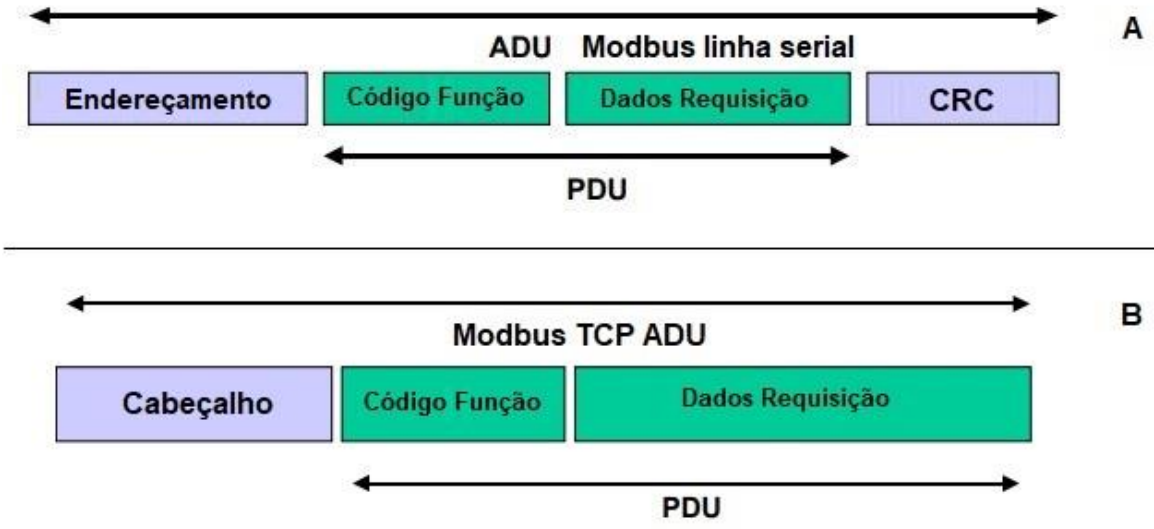

Fonte: Retirado de [11].

Para a realização da comunicação, o Modbus trabalha com uma unidade comum de informações a todas as implementações envolvendo o protocolo, chamada de Protocol Data Unit (PDU). O PDU é utilizado para que o servidor identifique a ação desejada pelo cliente e para que sejam informados parâmetros necessários para que o servidor desempenhe esta ação [11,12].

Associado ao PDU, acompanha uma estrutura que varia de acordo com a arquitetura de rede utilizada na aplicação. Na implementação pertencente a aplicações com linhas seriais, se utiliza uma estrutura que possui o endereço do destinatário e um CRC (*Cyclic Redudancy Check*) para checagem de erro relacionada a mensagem enviada (Figura 4, parte A) [11].

Por sua vez, na implementação envolvendo Modbus TCP, associado ao PDU está associado um cabeçalho contendo informações de identificador da transação, identificador de protocolo, tamanho da mensagem e identificador do cliente (Figura 4, parte B) [12]. O frame completo utilizado na comunicação envolvendo o protocolo Modbus é chamado de *Application Data Unity* (ADU).

Figura 4 - Formatação das tramas de comunicação no protocolo Modbus em linha serial (parte A) e Modbus TCP (parte B).




Fonte: Adaptado de [12].

### *2.3.2 O padrão Ethernet*

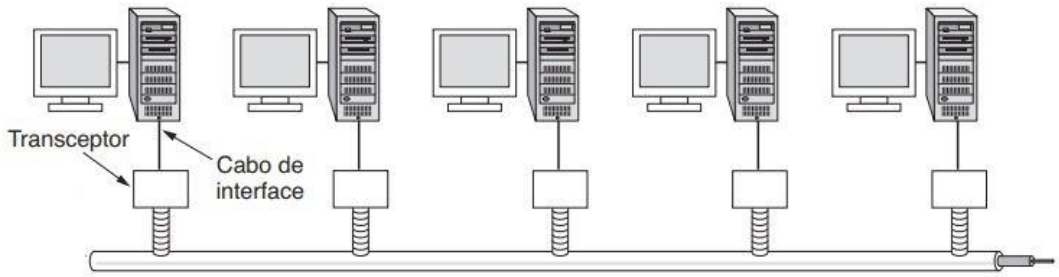
Ethernet é a tecnologia de redes locais mais utilizada ao redor do mundo. Esta tecnologia foi desenvolvida na década de 1970, tendo Robert Metcalfe como seu principal idealizador. Nesta concepção inicial, conhecida como Ethernet clássica, os dispositivos eram conectados a um cabo longo ao qual todos os dispositivos da localidade estavam conectados, sendo dessa forma realizada a conexão física entre os participantes da rede [9,10].

Desde 1985 foram criados pelo *Institute of Electrical and Electronic Engineers* (IEEE) padrões baseados na tecnologia de rede Ethernet levando em consideração o desenvolvimento de novas tecnologias com o passar do tempo. O padrão IEEE 802.3,

popularmente chamado de padrão Ethernet, é o padrão de rede mais conhecido e mais disseminado relacionado à LAN com fios. Este padrão de rede define implementações das camadas física e de enlace referentes ao modelo OSI [9,10].

O padrão de rede Ethernet se caracteriza pela comunicação de dispositivos a um meio comum onde são realizadas a transferência de dados no meio físico. Os dispositivos são conectados a este meio comum por um cabo de interface que pode ser de vários tipos diferentes (como fibras ópticas ou cabos de pares trançados). A arquitetura de rede Ethernet é apresentada na Figura 5 [9,10].

Figura 5 - Arquitetura da Ethernet.




Fonte: Retirado de [9].

Como apresentado na Figura 5, na rede Ethernet os dispositivos são conectados a um meio comum por um cabo de interface. Estes cabos de interface podem ser de vários tipos diferentes (como fibras ópticas ou cabos de pares trançados). Este último, ao substituir os cabos coaxiais, propiciou redução de custo e um aumento na facilidade de instalação com relação aos substituídos. As redes que utilizam padrão Ethernet utilizam este tipo de conceito em sua camada física [8, 9, 10].

Como a rede Ethernet se utiliza de um meio comum onde todos os participantes da rede estão conectados, se fez necessário identificar cada um dentro da rede para que as mensagens sejam recebidas corretamente pelos destinatários desejados. Para solucionar este problema, foi incorporada à rede Ethernet um sistema de endereçamento físico para cada dispositivo. A utilização desse sistema de endereçamento permite que cada dispositivo seja identificado dentro da rede de maneira única e, consequentemente, possa ser encontrado, permitindo que as mensagens recebidas possam chegar ao destinatário.

O nome dado a este sistema de endereçamento físico é *Media Access Control* (MAC). Ele consiste em um identificador de 48 bits apresentados como 12 termos hexadecimais. Este sistema de identificação de dispositivos dentro da rede faz parte da camada de enlace do padrão Ethernet. Dessa forma, o padrão Ethernet estabelece identificação física aos participantes da rede [10].

### *2.3.3 Internet Protocol (IP)*

De acordo com [10], IP é um protocolo de camada de rede (correspondente à camada 3 do modelo OSI). Este protocolo foi desenvolvido com o objetivo principal de promover a interconexão das redes. O IP fornece de maneira não orientada a conexão a melhor maneira de encaminhar os pacotes de dados até o seu destinatário final, não se importando com o conteúdo dos dados repassados a ele. Também são atribuições do protocolo IP a definição de um sistema de endereçamento (no intuito de identificação e localização de interfaces de rede) e a definição de um pacote de dados para envio e o direcionamento deles pela rede [9,10].

De acordo com [10] um pacote de dados corresponde a uma estrutura de informações constituída por um cabeçalho, onde contém informações de controle utilizadas por um protocolo, e, na maioria das vezes, também estão contidos dados do usuário ou dados vindos de camadas superiores. No que diz respeito ao trabalho do protocolo IP, as informações recebidas de camadas superiores são empacotadas para envio (mediante acréscimo de informações referente à camada de rede, tais como endereço do destinatário do pacote) e repassadas às camadas inferiores. A identificação de cada interface de rede existente é realizada no protocolo IP por meio de um sistema de endereçamento chamado de endereço IP [10].

### *2.3.4 Transmission Control Protocol*

De acordo com [10], o *Transmission Control Protocol* (TCP) é um protocolo pertencente à camada de transporte na rede (camada 4 do modelo OSI). O TCP é responsável por dividir as mensagens em pedaços menores no remetente e remontá-las em seu destino. Em caso de necessidade, ele providencia o reenvio dos pedaços

não recebidos pelo destinatário para que as mensagens sejam remontadas por completo e corretamente [10].

Por ser um protocolo orientado à conexão, o TCP precisa que uma conexão seja estabelecida entre remetente e destinatário antes que a transferência de dados se inicie. Este processo fornece mais confiabilidade à transmissão de dados. O TCP também se utiliza de portas de comunicação para passar as mensagens recebidas às camadas superiores. Estas portas são identificadas por números, chamados de números de porta. Esta metodologia adotada pelo protocolo TCP tem o intuito de permitir que haja várias comunicações ocorrendo simultaneamente. A associação entre um endereço de IP e um número de porta TCP é chamado de socket [9,10].

Segundo [9] existem algumas portas que são reservadas no protocolo TCP para a utilização por parte de alguns protocolos de camadas superiores. Os números de porta reservados são chamados de *well-known port numbers* e compreendem as portas de número inferior a 1024. O registro e utilização de números de porta no TCP são administrados pela *Internet Corporation for Assigned Names and Numbers* (ICANN) [10].

Comunicações que não utilizam uma *well-known port* number utilizam números de porta dentro de um intervalo acima de 1023. Apesar de existirem portas reservadas a protocolos de camadas superiores, não há um impeditivo para não utilizar uma porta que não seja a reservada no protocolo [10]. A porta de número 502 do TCP é reservada para comunicações que envolvem o protocolo Modbus [12].

### *2.3.5 Hypertext Transfer Protocol*

O *Hypertext Transfer Protocol* (HTTP) é um protocolo de comunicação da camada de aplicação estruturado na arquitetura cliente/servidor. Estruturado dessa forma, ele se utiliza do modelo de solicitação/resposta onde clientes se conectam a um servidor e realizam as requisições desejadas (chamadas no caso do protocolo em questão de requisições HTTP) enquanto o servidor se empenha em responder as requisições de clientes mediante o envio de respostas (chamadas na utilização deste protocolo como respostas HTTP). Além de ser um protocolo baseado em texto, sua utilização envolve o uso do protocolo TCP na camada de transporte [9, 13].

As possíveis mensagens de solicitações que um cliente pode enviar a um servidor são especificadas como operações chamadas métodos. Estas solicitações vão desde requisições de conteúdos a deletá-los. As respostas a estes pedidos contêm uma linha de status, informando o resultado da requisição (se foi bem-sucedida, se houve algum erro no processo, entre outros) e, a depender do que foi pedido ao servidor e do resultado da requisição, esta linha de status é acompanhada por informações adicionais [9, 13]. A Figura 6 apresenta os possíveis grupos de status que uma resposta HTTP pode conter.

Figura 6 - Possíveis grupos de status que uma resposta a uma solicitação HTTP pode conter.

| Código | Significado | Exemplos |
|---|---|---|
| 1xx | Informação | 100 = servidor concorda em tratar da solicitação do cliente |
| 2xx | Sucesso | 200 = solicitação com sucesso; 204 = nenhum conteúdo presente |
| 3xx | Redirecionamento | 301 = página movida; 304 = página em cache ainda válida |
| 4xx | Erro do cliente | 403 = página proibida; 404 = página não localizada |
| 5xx | Erro do servidor | 500 = erro interno do servidor; 503 = tente novamente mais tarde |

Fonte: Retirado de [9].

Com o intuito de especificar a utilização do protocolo HTTP, é atribuído o nome de um cliente que realiza solicitações mediante este protocolo de “cliente HTTP” e o servidor que recebe e responde solicitações envolvendo este protocolo é chamado de “servidor HTTP”. A porta de comunicação reservada ao HTTP dentro do TCP é a porta de número 80 [9,13].

### *2.3.5.1 Secure Hypertext Transfer Protocol*

O *Secure* HTTP também denominado de HTTPS trata-se da utilização do protocolo HTTP em conjunto com um protocolo chamado *Secure Sockets Layer* (SSL). O protocolo SSL tem a função de promover uma conexão segura para a comunicação e promover a integridade dos dados comunicados entre as partes realizando a proteção dos dados por meio de criptografia e autenticação de participantes da

comunicação [9, 14]. O posicionamento do SSL dentro da pilha de protocolos é apresentado na Figura 7.

Figura 7 – Posicionamento do protocolo SSL dentro de uma pilha de protocolos.


Fonte: Retirado de [9].

Como apresentado na Figura 7, a atuação do protocolo SSL está posicionada entre a camada de aplicação (da qual faz parte o HTTP neste caso) e a camada de transporte. De acordo com [14], a porta reservada para o HTTPS dentro do TCP é a porta de número 443.

### *2.3.6 Simple Mail Transfer Protocol*

O *Simple Mail Transfer Protocol* (SMTP) é um protocolo de comunicação responsável por gerenciar a transmissão de e-mails através da rede. Este protocolo é utilizado pelos servidores de e-mail para o processo de envio e recepção de e-mails. O transporte de mensagens no protocolo SMTP é realizado no formato ASCII utilizando o protocolo TCP. A utilização do SMTP fornece um status informando se a entrega da mensagem foi realizada de maneira bem-sucedida ou se houve alguma falha no processo [9,10].

A porta TCP associada ao protocolo SMTP padrão é a porta de número 25. Existem outras implementações do protocolo SMTP que envolvem a inserção de uma camada de segurança com o intuito de fornecer privacidade aos dados enviados. Essas camadas de segurança envolvem protocolos como o SSL ou TLS (*Transport*

*Layer Security,* apresentado por [15] como uma evolução do SSL dotada de mais segurança) [9,15]. A utilização do SMTP envolvendo o SSL e o TLS utilizam respectivamente as portas TCP de número 465 e 587.

### *2.3.7 Message Queuing Telemetry Transport*

O *Message Queuing Telemetry Transport* (MQTT) é um protocolo de mensagens que utiliza a arquitetura de publicação-assinatura [16]. Esta arquitetura de comunicação utilizada no MQTT se caracteriza pela comunicação entre remetente (chamado de publicador) e destinatários (chamados de assinantes) ser realizada de maneira indireta, mediante a utilização de um intermediário (chamado de *broker*), responsável por receber as mensagens do publicador e encaminhar as mensagens àqueles que estão interessados no conteúdo da mensagem em questão [17].

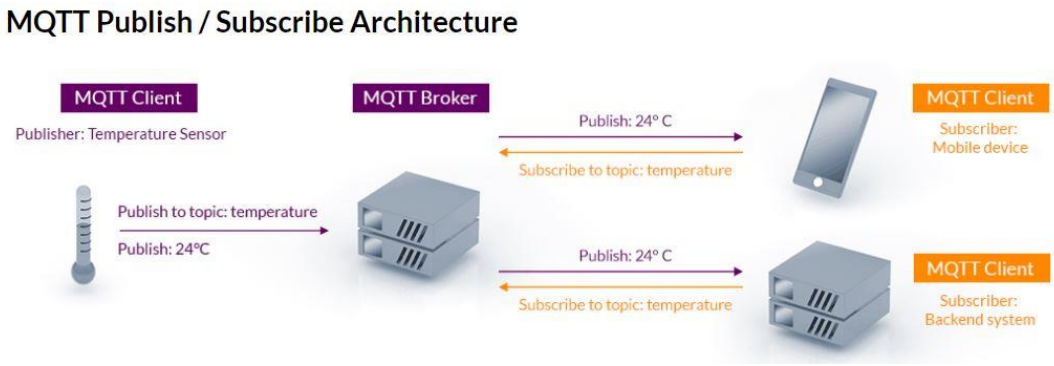
No protocolo MQTT os publicadores se conectam ao *broker* e publicam as mensagens em tópicos (que são strings às quais as mensagens estão associadas). Os assinantes, por sua vez, se conectam ao *broker*, se inscrevem em tópicos desejados e recebem as mensagens que são publicadas nos tópicos em que realizaram assinatura. Nesta estrutura de comunicação, o *broker* é responsável por repassar as mensagens de um determinado tópico apenas para os assinantes do tópico ao qual a mensagem pertence, processo este chamado de filtragem [17].

A estruturação da comunicação no protocolo MQTT utilizando o modelo de publicação-assinatura faz com que publicadores estejam desassociados de assinantes, diferentemente de estruturas de comunicação onde se faz necessária uma conexão direta entre o remetente da mensagem e o destinatário da mensagem. O publicador publica as mensagens em um determinado tópico e o *broker* se comunica com o assinante do tópico pertencente à mensagem (que está conectado a ele) para repassar a mensagem publicada [17].

De acordo com [18] um programa ou dispositivo que utiliza o MQTT para ações de publicação ou assinatura são chamados de clientes. Segundo [17] um mesmo cliente pode publicar mensagens em tópicos e se inscrever em tópicos para receber mensagens. Um *broker,* chamado por [18] de servidor, é caracterizado pelo citado como um programa ou dispositivo que realiza a intermediação de uma comunicação

entre publicador e assinantes. O modelo de comunicação MQTT é apresentado na Figura 8.

Figura 8 - Estrutura de comunicação MQTT.




Fonte: Retirado de [19].

O MQTT disponibiliza a possibilidade de entrega de mensagens em três diferentes níveis de qualidade de serviço (*Quality of Service*, QoS). O nível de Qualidade de Serviço (QoS) é um padrão adotado entre o remetente e o destinatário de uma mensagem que define a garantia de entrega dela. Os níveis de qualidade de serviço disponibilizados pelo MQTT são o QoS 0, QoS 1 e QoS 2 [17].

O QoS 0 disponibiliza uma entrega mais rápida, porém menos confiável. Ao utilizá-lo, as mensagens são enviadas apenas uma vez e não há garantias de que a mensagem será entregue. Caso esta seja perdida, ela não é enviada novamente [17].

Nas transmissões envolvendo Qos 1, as mensagens são enviadas pelo menos uma vez. Caso elas sejam enviadas e não haja uma confirmação de recebimento pelo receptor, elas são reenviadas e assim sucessivamente até que haja a confirmação de recebimento. Este nível de qualidade de serviço não trata a questão de múltiplo recebimento de uma mesma mensagem, fazendo assim com que as que chegam mais de uma vez sejam processadas mais de uma vez nos receptores da mensagem [17].

Por fim, na transmissão envolvendo QoS 2, a mensagem é enviada, caso o receptor não confirme o recebimento dela, esta é reenviada e assim quantas vezes necessário (assim como no QoS 1). Mas, diferentemente de transmissões envolvendo QoS 1, há um tratamento realizado no recebimento de mensagens de forma que

mensagens duplicadas que chegam ao receptor não sejam processadas mais de uma vez [17].

Desta forma, serviços com níveis de qualidade de serviço superiores são mais lentos, mas possuem graus de confiança maior. Segundo [17], o serviço mais lento de todos é o de Qos 2 enquanto o serviço de entrega mais rápida e menos confiável (como citado anteriormente) é o nível de qualidade de serviço 0.

## 2.4 Python

Python consiste em uma linguagem de programação de uso geral desenvolvida pelo matemático e professor Guido van Rossum em 1991. A linguagem de programação Python trabalha mediante a utilização de uma licença de código aberto, o que viabiliza a sua livre utilização por parte de usuários e permite que as aplicações desenvolvidas nela possam ter uma livre distribuição mesmo em utilizações comerciais [20, 21, 22].

De acordo com [22], o Python possui um repositório que abriga extensões pertencentes ao próprio Python ou criadas por terceiros para a referida linguagem. O nome deste repositório é chamado de PyPi (*Python Package Index – Índice de Pacotes Python*, em tradução direta). As extensões que podem ser utilizadas são denominadas módulos, quando realizadas por um só arquivo ou bibliotecas, quando compreendem um conjunto de arquivos. De acordo com [23], o repositório PyPi possui várias extensões que fornecem as mais diferentes funcionalidades tais como utilização de protocolos de comunicação, desenvolvimento de interface gráfica e análise de dados.

## 2.5 Bancos de dados e Google Firebase

Os bancos de dados são caracterizados por [24] como um conjunto lógico e ordenado de dados que apresentam significado, que representa uma parte do mundo real, construído com dados que possuem um objetivo definido. Segundo o referido autor, dados são uma representação de algum fato ou conhecimento do mundo real. O mesmo define os fatos ou conhecimentos do mundo real como informações, estas

ao serem arquivadas, tornam-se uma representação de um fato. Dentre as diversas aplicações de bancos de dados disponíveis está o *Realtime Database do Google Firebase*.

De acordo com [25], o *Google Firebase* consiste em uma plataforma de desenvolvimento de aplicações que possui o suporte da empresa Google. Dentre as ferramentas disponibilizadas pelo *Google Firebase* estão o *Realtime Database*. Segundo [26], o *Realtime Database* é um banco de dados hospedado na nuvem, não relacional (não organizado em tabelas) e multiplataforma, podendo ser utilizado em aplicações para dispositivos móveis e em aplicações ligadas a internet, chamada de aplicações web.

Segundo [26], o *Realtime Database* do *Google Firebase* fornece a possibilidade de sincronização de dados com clientes (entenda-se clientes como dispositivos conectados ao banco de dados) em tempo real permitindo que as informações que o cliente tem do banco sejam atualizadas pouco tempo após as alterações no banco de dados terem sido realizadas.

Segundo [27], o conteúdo de um *Realtime Database* criado pode ser acessado mediante requisições utilizando protocolo HTTPS (*Secure Hypertext Transport Protocol)* através da REST API do *Google Firebase*.

# 3 METODOLOGIA

Esta seção apresenta passo a passo a utilização de conceitos e as execuções realizadas para concretização do trabalho. Na seção 3.1, é apresentado o processo avaliação e escolha da melhor estrutura de comunicação para ser utilizada no desenvolvimento do trabalho. Na seção 3.2 é apresentado como se deu o processo de programação do CLP levando em consideração a abordagem escolhida na seção 3.1. Na seção 3.3 é apresentada a metodologia utilizada para o desenvolvimento de um software que atuará no servidor local utilizado para obter dados do CLP e repassá-los aos dispositivos móveis. Por fim, na seção 3.4 é apresentada a metodologia desenvolvida para produção do aplicativo que será utilizado nos aparelhos móveis, informando dados de processo ao usuário.

## 3.1 Estruturação das comunicações

Na necessidade de comunicar dispositivos, se faz necessário definir como serão realizadas as comunicações entre eles. Cada dispositivo participante da comunicação possui características próprias que devem ser levadas em consideração na hora de decidir qual é a melhor opção para os envolvidos, tais como protocolos incorporados por cada um, padrão de conexão física que aceitam, capacidade de processamento, entre outros. Fatores como possíveis custos adicionais, velocidade na transmissão de dados e finalidade de uso também devem ser analisados.

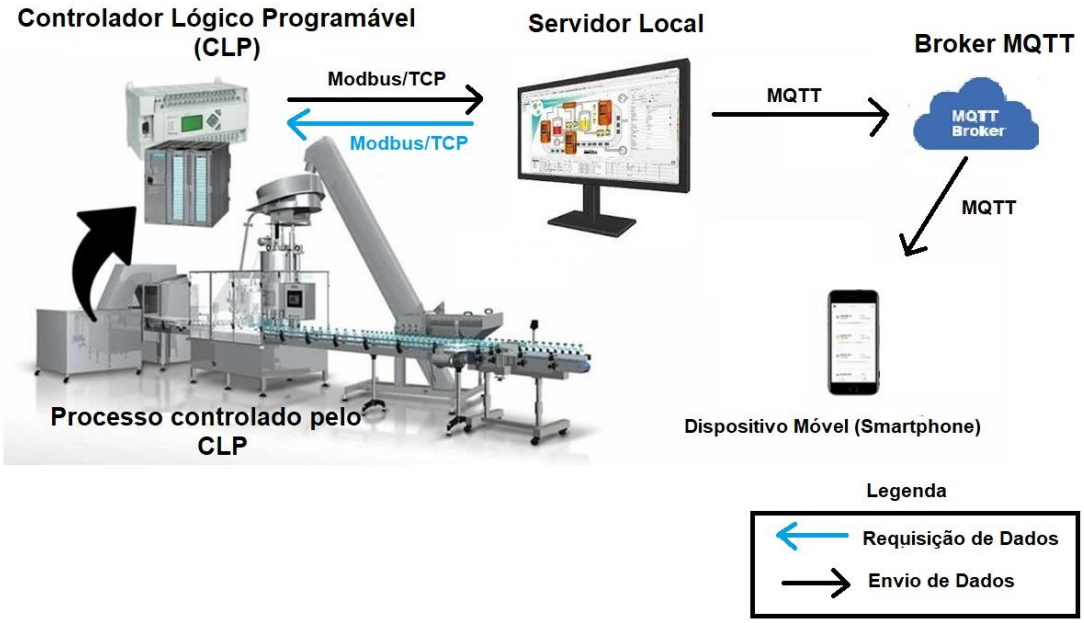
Após verificação e análise dos fatores listados, foi escolhida a utilização do protocolo Modbus TCP na conexão entre CLP e servidor local. A utilização do protocolo Modbus TCP, por ser um protocolo aberto, permite que seja desenvolvido um software próprio que o utilize. O protocolo também já está implementado no CLP S7-1200 DC/DC/DC versão (6ES7 214-1AG31-0XB0), da Siemens, fato que faz com que a utilização do protocolo não possua nenhum impeditivo no CLP utilizado.

A conexão utilizando padrão Ethernet por parte do protocolo Modbus TCP também é facilitadora para a utilização do protocolo. Neste padrão, a conexão física pode ser realizada por cabos de pares trançados (como o CAT 5, mais utilizado em aplicações envolvendo padrão Ethernet e o CAT 7, que possui blindagem nos pares trançados para atenuar efeitos de interferência externa, segundo [9]) com a utilização de conectores RJ45 acoplados as suas extremidades, com o qual os dois hardwares já possuem compatibilidade.

Para a conexão entre o servidor local e os dispositivos móveis, foi escolhido o protocolo de comunicação MQTT. A comunicação entre o servidor local e estes dispositivos deve se utilizar de um meio de transporte não guiado para a transmissão de dados, uma vez que deve haver um desacoplamento físico entre eles. A viabilidade desta conexão por cabos, mesmo que fosse possível, comprometeria a mobilidade dos aparelhos. A possibilidade de existência de múltiplos destinatários para uma mesma mensagem; se adequar às limitações de hardware de dispositivos móveis, como alguns modelos de smartphones; trabalhar em velocidades mais lentas de conexão e poder assegurar a entrega da informação devido à presença de níveis de qualidade de serviço também favorecem a utilização do protocolo MQTT neste caso.

A utilização do protocolo MQTT acrescenta à comunicação mais um elemento: o *broker* MQTT. Este atua como um intermediário na transmissão da informação entre o servidor local e dispositivo móvel. O servidor local publicará os dados no *broker* MQTT em tópicos especificados e este repassará os dados aos dispositivos móveis nos tópicos especificados pelo servidor local. Para a efetuação desta comunicação, o *broker* MQTT utilizado foi um *broker* MQTT externo. A representação da estrutura de comunicação utilizada no trabalho é apresentada na Figura 9.

Figura 9 - Representação da estrutura de comunicação utilizada.



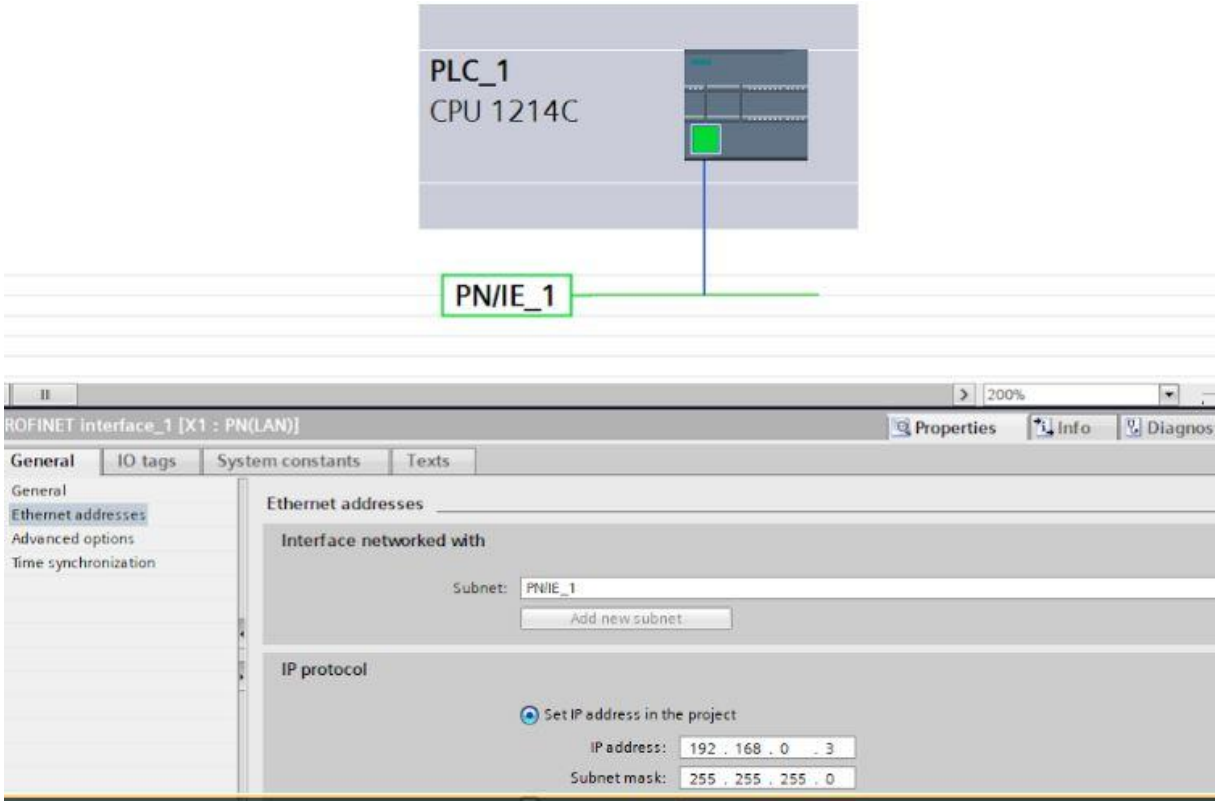

Fonte: Próprio autor.

## 3.2 Programação do CLP

Para que o CLP possa se comunicar com dispositivos externos via protocolo Modbus TCP e enviar dados sempre que solicitado, é necessário que ele seja configurado para desempenhar esta função. É necessário configurá-lo como servidor Modbus TCP, tornando-o um dispositivo participante da rede que receberá requisições dos clientes Modbus TCP e irá respondê-las. Nesta aplicação o servidor local foi o cliente Modbus TCP e realizou solicitações sobre dados ao controlador lógico programável em questão.

As configurações do CLP foram realizadas no software Siemens Tia Portal versão 15. Siemens Tia Portal é o ambiente de desenvolvimento de programação para CLPs Siemens e outros dispositivos utilizados na área de automação industrial da referida fabricante. Inicialmente, ajustou-se o endereço de IP dele para um endereço de IP da mesma rede à qual pertence o servidor local. Posteriormente, criou-se uma conexão de rede para o CLP dentro do Tia Portal. Após estas configurações iniciais, pode-se configurar o CLP como um servidor Modbus TCP. A Figura 10 apresenta a configuração da interface de rede do CLP finalizada.

Figura 10 - Configuração da interface de rede do CLP.




Fonte: Próprio Autor.

### *3.2.1 Configuração do CLP como servidor Modbus*

Para realizar ações de leitura de dados do CLP se faz necessário configurar a utilização do protocolo da camada de aplicação, o protocolo Modbus, na programação do CLP. O CLP deve receber requisições do servidor local para leitura de variáveis

contidas nele. Por precisar receber requisições do cliente Modbus TCP (o servidor local) e respondê-las o CLP foi configurado como um servidor Modbus TCP.

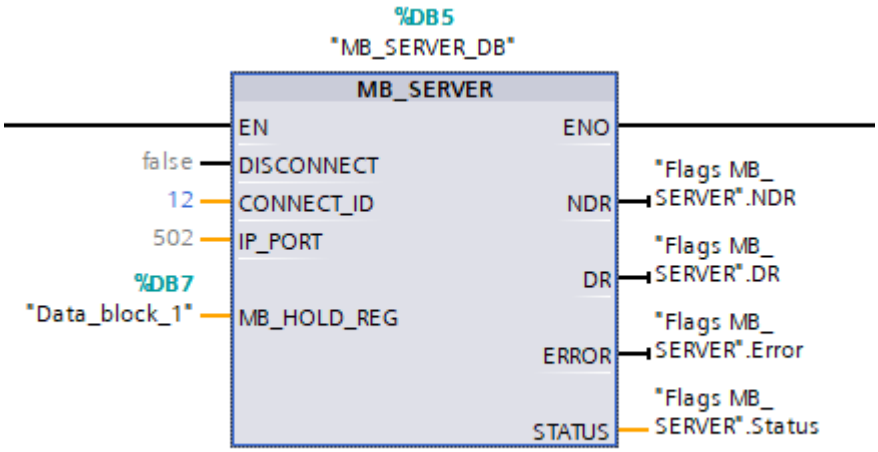
A configuração do CLP como um servidor Modbus TCP foi executada através da utilização do bloco MB_SERVER do Tia Portal. O bloco MB_SERVER para o CLP-S7 1200 DC/DC/DC na versão 6ES7 214-1AG31-0XB0, por ter um firmware de versão abaixo da versão 4, requereu informações de porta de comunicação TCP a ser utilizada (configurada nesta aplicação como a porta 502, padrão para aplicações envolvendo protocolo Modbus utilizando TCP) e um registrador para ser utilizado como *holding register* da comunicação. Para o uso nesta aplicação foi utilizado um *data block* não otimizado como *holding register*. Um *data block* é um espaço reservado através da programação do CLP utilizado para armazenar variáveis e informações utilizadas durante funcionamento do controlador lógico programável [7].

O MB_SERVER também solicita que seja associado a ele um número identificador da conexão (parâmetro ID do bloco). Este identificador deve ser um número inteiro de uso único dentre as instruções de comunicação utilizados no CLP. Ou seja, caso sejam utilizados outros blocos de comunicação pelo mesmo CLP simultaneamente, eles devem ter identificadores diferentes. Nesta utilização foi atribuído o número 12 como identificador do bloco.

Por fim, o parâmetro *Disconnect* possibilita a implementação de um grau de controle na resposta de requisições. Quando este parâmetro recebe valor *true* (ou seu correspondente inteiro 1) o CLP não aceita requisições enviadas pelos clientes Modbus. No caso receber o valor *false* (ou seu correspondente inteiro 0) o CLP aceita e responde as requisições que forem realizadas por seus clientes (desde que não esteja ocupado com outras requisições). Neste trabalho, foi adotada a utilização contínua do valor false possibilitando que o CLP responda as requisições realizadas a ele em qualquer momento.

A configuração final do bloco MB_SERVER utilizada é apresentada na Figura 11. A verificação da situação de conexão pode ser visualizada em tempo de execução através das variáveis de saída da instrução (NDR, DR, ERROR e STATUS) mediante valores e códigos que aparecem no próprio Tia Portal. Em caso de possível erro, a variável ERROR recebe valor *true* e STATUS apresenta o código do erro ocorrido em MB_SERVER para verificação e análise do problema.

Figura 11 - Configuração Bloco MB_SERVER.




Fonte: Próprio autor.

## 3.3 Software para servidor local

A comunicação entre CLP e servidor local via protocolo Modbus TCP impõe a utilização de um software no servidor local para o estabelecimento da comunicação entre os dispositivos nas diferentes camadas de comunicação visando a obtenção de dados do controlador lógico programável. A criação de um software próprio, se utilizando do fato do protocolo Modbus ser um protocolo aberto, propicia uma opção de custo reduzido, com alto nível de customização para a situação enfrentada nesta aplicação.

Para o desenvolvimento do software que foi executado no servidor local foi utilizada a linguagem de programação Python. Esta foi escolhida levando em consideração a necessidade de utilização de uma linguagem de uso geral, visto a multiplicidade de requisitos que seriam, e foram, solicitados à linguagem ao longo do trabalho, e, por ser uma linguagem de programação de código aberto, o que facilita o acesso a extensões para prover funcionalidades que seriam necessárias para a execução satisfatória do projeto. A versão do Python utilizada neste trabalho foi a 3.7.0.

### *3.3.1 Interface gráfica e interação com usuário*

Para o recolhimento de informações que devem ser inseridas pelo usuário e para apresentar informações de processo a ele, se faz necessária a utilização de uma interface gráfica para tornar o processo de interação com o usuário mais amigável e preciso. Dentro das linguagens de programação, existem alguns meios por onde a interação com o usuário é provida, seja pelo terminal de comunicação, frameworks ou bibliotecas dedicadas. Cabe ao desenvolvedor do software escolher qual é a melhor maneira para interagir com o usuário na aplicação em questão.

No atual trabalho, foi utilizado o framework Python *Kivy* para desenvolvimento da interface gráfica. Além de apresentar a versatilidade como uma de suas vantagens, pelo fato de um software desenvolvido utilizando *Kivy* poder ser utilizado nos mais diversos sistemas operacionais, o *Kivy* é um framework aberto, sem custos para utilização em fins comerciais (um facilitador em projetos futuros envolvendo uma evolução deste trabalho), algo que outros frameworks que trabalham com Python não são (como o PyQt5 que é pago em utilizações comerciais).

Outra vantagem presente na utilização do *Kivy* é que este, como forma de aumentar a praticidade do desenvolvimento, possui uma linguagem de desenvolvimento própria que pode ser utilizada para auxiliar o desenvolvedor a criar um software: a linguagem KV. A linguagem KV não exclui o Python do desenvolvimento (pois não se pode criar um software apenas com a linguagem KV), mas se integra ao Python para permitir a execução de um desenvolvimento mais intuitivo.

O *Kivy* também apresenta uma extensão, o *Kivymd* onde são encontrados widgets com acabamentos mais sutis e onde também estão presentes widgets dedicados para utilização em plataformas móveis, o que impacta positivamente na experiência do usuário com o software desenvolvido. A utilização do *Kivy* não impede a utilização de bibliotecas Python para suprir outras funcionalidades solicitadas.

### *3.3.2 Sistema de Registro e Autenticação usuários*

O software desenvolvido conta com um sistema de registro de usuários, o que permite a utilização do software em mais de uma planta, ao mesmo tempo, sem que haja sobreposição de dados, com as informações de uma planta interferindo nas informações de outra planta. A implementação de um sistema de registro de usuários também permite o uso do mesmo software, no mesmo servidor em plantas diferentes sem que uma utilização atrapalhe a outra. Além disso, a utilização de um sistema de registro associado a um sistema de autenticação de usuários, provê mais segurança ao sistema, evitando que qualquer pessoa possa livremente entrar no sistema e capturar dados sem autorização.

A utilização de um sistema de registro demanda armazenamento de informações. Por conta disso, se fez necessário a utilização de um banco de dados. O banco de dados utilizado para armazenamento de dados no sistema de registro trata-se do *Realtime Database* do *Google Firebase*.

Para criar um banco dados para ser utilizado no projeto no *Realtime Database* do *Google Firebase*, foi criada uma conta de serviços do Google específica para o uso no trabalho em questão. Através desta conta, pode-se obter acesso ao *Google Firebase*. Após o acesso ao *Firebase*, foi criada uma aplicação no *Google Firebase*. Dentro desta aplicação, foi criado um banco de dados no *Realtime Database* do *Google Firebase*.

Durante a criação de um registro de uma planta no sistema, o usuário precisa fornecer algumas informações. Estas são: nome da planta, e-mail do responsável pela planta, senha para entrada no sistema e e-mail de pessoas que estarão autorizadas para acessar informações da planta via smartphone.

Após a confirmação das informações e solicitação de cadastro, este é realizado mediante armazenamento das informações passadas através do sistema de registro no *Realtime Database* do *Google Firebase* pertencente ao projeto criado. As informações obtidas no registro foram armazenadas no banco de dados citado utilizando a biblioteca Python *Requests* que se utiliza do protocolo HTTP para se comunicar através da REST API do *Google Firebase* e armazenar os dados no banco

de dados utilizado. O método da biblioteca *Requests* utilizado para escrita no *Google Firebase* foi o método *post.*

As informações passadas através da biblioteca *Requests* ao *Realtime Database* do *Google Firebase* foram armazenadas no banco de dados em um ramo chamado plantas, criado previamente para armazenar as informações necessárias ao cadastro de plantas. Ao se utilizar a escrita de dados no *Google Firebase* através da REST API, as informações de cadastro são salvas dentro do ramo plantas em um ramo criado pelo *Firebase* com um identificador único gerado pela própria aplicação do Google. Essa ação do *Firebase* faz com que não haja mistura entre os dados escritos no banco.

A organização de como os dados vindos do cadastro de plantas são armazenados no *Realtime Database* do *Google Firebase* é apresentada na Figura 12. A Figura mostra como foram armazenadas as informações de registro de uma planta chamada Empacotadora. As senhas foram armazenadas no banco de dados na sua forma encriptada, utilizando a encriptação “sha256*”* para proteção de dados do usuário. A encriptação é realizada no software desenvolvido utilizando a biblioteca Python *Hashlib*. O nome da planta, e-mail do responsável pela leitura de dados da planta, senha encriptada e usuários permitidos para monitoramento mobile são armazenados nos identificadores “*nome_planta”*, “e-mail”, “senha” e “usuários” do novo ramo criado dentro do ramo plantas, respectivamente.

Figura 12 - Organização dos dados advindos de cadastros de plantas no *Realtime Database* do *Google Firebase*.


Fonte: Próprio autor.

Após o cadastro de plantas, o usuário pode se logar no sistema com a planta criada, podendo realizar a entrada de informações e dar início à operação de requisição de dados e envio destes por meio do servidor local. Enquanto isso, os usuários que receberam permissão para monitorar as informações da planta cadastrada através de seus dispositivos móveis, recebem um e-mail, nos endereços fornecidos no momento do cadastro, requisitando o cadastro deles no sistema, sem o qual eles não podem monitorar informações, mesmo que tenham permissão.

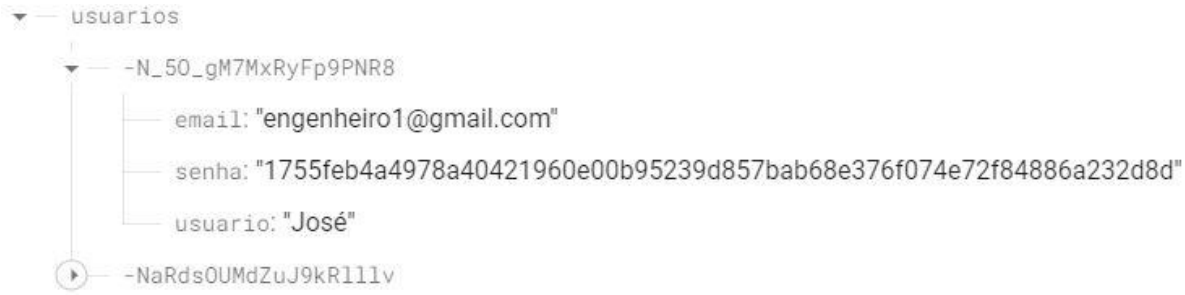
Através do software desenvolvido, os e-mails são enviados aos usuários que receberam permissão. O envio de e-mails utiliza as bibliotecas Python *Smtplib* e *email.message.* O envio de mensagens aos usuários utilizando essas referidas bibliotecas é realizado vinculado a uma conta de e-mail existente. Foi utilizada uma conta de e-mail do Google (a mesma que está associada a utilização do *Google Firebase*) criada antes da execução do projeto.

O e-mail enviado contém um link que os envia para uma página web, de própria criação, utilizando as linguagens de programação PHP e Javascript e linguagem de organização textual HTML (*Hypertext Markup Language*), responsável por receber as informações do usuário e armazená-las no banco de dados utilizado (*Realtime Database* do *Google Firebase*). Estas informações são nome de usuário e senha.

Após confirmação de cadastro por parte de um usuário permitido, as informações dele são salvas no ramo usuários do *Realtime Database* do *Google Firebase*, criado previamente. As informações salvas no *Google Firebase* no ramo “usuários” contêm as informações de e-mail cadastrado (que já era conhecido antes da finalização do cadastro, e-mail por onde foi enviado o link), nome, senha da pessoa em questão.

A organização de informações de usuários cadastrados no sistema é apresentada na Figura 13. A Figura apresenta os dados de um usuário armazenados no banco. Após confirmação de informações para cadastro, as informações de um usuário são armazenadas dentro do ramo “usuários”, mas dentro de um ramo criado pelo *Google Firebase* de identificador único assim como acontece no cadastro de plantas. A senha escolhida pelo usuário em questão é armazenada na forma encriptada. Para encriptação foi utilizada a encriptação “sha256”. A encriptação é realizada no *script* PHP desenvolvido para cadastro de usuários utilizando a função chamada *hash*.

Figura 13- Organização de informação de usuários cadastradas para uso de monitoramento em plataforma mobile.




Fonte: Próprio autor.

Assim como o sistema de registro de plantas e de usuários do software desenvolvido, o sistema de login se utiliza da biblioteca Python *Requests* para confirmar as informações de login e permitir a entrada dele no login da planta em questão. A biblioteca *Requests*, assim como nas situações de registro de plantas e usuários, se utiliza do protocolo de comunicação HTTPS para se comunicar o *Realtime Database* do *Google Firebase*.

A diferença que se deu durante a execução do projeto é que como na situação de login se faz necessário ler informações do banco de dados e não escrever informações no banco de dados, as funcionalidades utilizadas da biblioteca *Requests* foram diferentes. Para ler as informações do ramo plantas (ramo contido no banco que armazena todas as plantas cadastradas) foi utilizado o método *get* da biblioteca *Requests*.

A resposta da requisição em questão é um dicionário Python com as plantas cadastradas e suas informações contidas no *Realtime Database* do *Google Firebase*. O software desenvolvido foi programado para realizar a procura pelo nome da planta que foi digitado pelo usuário e se a senha digitada por ele corresponde à senha que foi armazenada no banco de dados após a senha digitada também ser encriptada utilizando a biblioteca Python *Hashlib* utilizando encriptação “sha256”.

Caso as informações coincidam, o usuário terá acesso à tela principal do software com o login da planta que foi utilizado. Caso contrário, mensagens de erro são apresentadas, tais como “Senha inválida, tente novamente.”, ou “ID da planta não existe nos registros”, em caso de tentativa de login sem prévio cadastro da planta, entre outros. Tudo a depender do erro causado pela entrada do usuário em questão.

Em casos necessários também pode ser realizada uma recuperação de senha. A recuperação de senha requere que sejam dados como entrada o nome da planta e o e-mail que está vinculado à criação da planta. Assim como no sistema de login, na recuperação de senha, o software, inicialmente, se comunica com o *Realtime Database* do *Google Firebase* através da REST API do *Firebase* utilizando o método *get* da biblioteca *Requests*.

Após a resposta das requisições, o software analisa se as informações do usuário de nome da planta e e-mail vinculado a planta estão corretas. Em caso serem corretas, é enviado um e-mail para o endereço de e-mail vinculado à planta criada. O e-mail criado contém um link que direciona o usuário para uma página web, de criação própria, que é responsável por obter do usuário a nova senha que ele deseja utilizar e realizar essas alterações no *Realtime Database* do *Google Firebase*.

A página web de recuperação de senha é apresentada na Figura 14. A página web de recuperação de senha se utilizou das linguagens de programação PHP e Javascript e de organização textual HTML para realizar a ação de captura de nova senha do usuário e alterações dentro do banco de dados do *Firebase*.

Figura 14 - Página *web* de recuperação de senha de plantas cadastradas.


Fonte: Próprio autor.

Em caso de incompatibilidade das informações passadas pelo usuário em recuperação de senha, são apresentadas mensagens de erro ao usuário tais como “E-mail incompatível com e-mail vinculado ao ID da planta”. Para o caso de haver tentativa de recuperação de senha da planta mesmo sem criar uma planta antes é apresentada a mensagem “ID da planta não está contido nos registros”.

### *3.3.3 Inserção de dados por parte do usuário*

Após o desenvolvimento do sistema de cadastro, login e recuperação de senha de plantas no software, foi desenvolvida a interface onde foram realizadas as inserções de dados por parte do usuário. O fornecimento de informações por parte do usuário se fez necessário para que fossem obtidas as informações mínimas para abertura de conexão com o servidor Modbus TCP, papel cumprido pelo CLP, e, assim fossem realizadas as leituras de registradores através da utilização do protocolo Modbus TCP.

Os dados que foram solicitados ao usuário através da tela se tratam de endereço IP do CLP na rede e a porta de comunicação por onde foram desenvolvidas as comunicações, parâmetros previamente configurados no CLP através do uso do software Siemens Tia Portal. Além desses dois parâmetros, é requisitado ao usuário a tabela de *tags* do CLP com o intuito de conhecer quais variáveis devem ser requisitadas leituras ao CLP e apresentar seus respectivos comportamentos ao usuário.

Uma *tag* é um nome que está associado a um endereço físico do CLP (entrada, saída ou memória) [7]. Uma tabela de *tags* é o conjunto de *tags* configuradas pelo usuário durante o desenvolvimento do programa do CLP. Em uma tabela de *tags* estão contidas informações como o nome dado pelo desenvolvedor para representar cada endereço físico que ele utilizou, bem como os endereços que ele utilizou, o tipo de dado que será armazenado dentro desse *endereço* (se será 1 *bit*, *byte*, *word*, inteiro, entre outros) entre outras informações.

A tabela de *tags* utilizada na programação de um CLP no Tia Portal pode ser exportada em um formato compatível com apresentação em planilhas (formato xlsx). A Figura 15 apresenta a imagem de uma tabela de *tags* utilizada na programação de

um CLP S7-1200 exportada do Tia Portal. Para cada *tag* configurada pelo programador do CLP, é descrita na tabela de *tags* o nome da *tag* que foi utilizado pelo programador, o endereço ao qual o nome está associado e o tipo de dado que será armazenado no endereço em questão (*bit*, *byte*, *word* entre outros), primeira, terceira e quarta colunas, respectivamente.

Figura 15 - Tabela de *tags* de uma programação de CLP S7-1200.

| Name | Path | Data Type | Logical Address | Comment | Hmi Visible | Hmi Accessible | Hmi Writeable | Typeobject ID | Version ID |
|---|---|---|---|---|---|---|---|---|---|
| Liga Contator | Default tag table | Bool | %Q0.5 | | True | True | True | | |
| Vel Motor | Default tag table | Word | %QW80 | | True | True | True | | |
| Processo Ligado | Default tag table | Bool | %I0.3 | | True | True | True | | |
| Acionamento Rele | Default tag table | Bool | %Q1.1 | | True | True | True | | |
| Potenciometro | Default tag table | Word | %IW64 | | True | True | True | | |
| Aciona misturador | Default tag table | Bool | %Q0.0 | | True | True | True | | |
| Liga Esteira | Default tag table | Bool | %Q0.7 | | True | True | True | | |
| Defeito Chave | Default tag table | Bool | %I0.4 | | True | True | True | | |
| Num processos | Default tag table | Word | %MW5 | | True | True | True | | |

Fonte: Próprio autor.

O carregamento de uma tabela de *tags* escolhida pelo usuário foi desenvolvido no software utilizando a biblioteca Python *Plyer*. *Plyer* é uma biblioteca Python que auxilia o acesso à recursos de hardware e de plataformas [28]. Se utilizou o objeto *filechooser* do *Plyer* para prover a função de seleção da tabela de *tags* desejada pelo usuário.

Após o carregamento da tabela de *tags*, foi utilizada a biblioteca Python *Pandas* para extrair as informações da tabela de *tags* dentro do software. *Pandas* é uma biblioteca Python que disponibiliza ferramentas para permitir o trabalho com dados relacionais ou rotulados de maneira mais simplificada [29]. Os dados extraídos das tabelas de *tags* inseridas são as informações da primeira, terceira e quarta colunas de cada linha, que correspondem ao nome da *tag*, o endereço da *tag* e o tipo de dado que foi armazenado na *tag*.

Após a extração de dados contidos na tabela de *tags*, os dados obtidos precisaram ser tratados e processados dentro do software para que pudessem ser realizadas requisições de leituras de dados ao CLP e, posteriormente, conseguir transmiti-los para o *broker* MQTT. Mediante a necessidade de tratar os dados que foram obtidos através do carregamento da tabela de *tags* do CLP, foi desenvolvido na

codificação do software uma rotina para tratamento dos dados obtidos da tabela de *tags*. Esta rotina é apresentada na seção 3.3.4

### *3.3.4 Rotina de tratamento de dados inseridos da tabela de tags*

A rotina de tratamento de dados inseridos via tabela de *tags* foi criada com o objetivo de processar os dados recolhidos da tabela de *tags* e fornecer ao sistema as informações necessárias para se efetuar requisições de leitura utilizando o protocolo Modbus. Informações estas que são o endereço de onde serão iniciadas as leituras e a quantidade de dados que serão lidos dentro de um determinado grupo de registradores. Faz parte também da rotina de tratamento de dados separar os nomes das *tags* de acordo com o grupo de registradores ao qual cada *tag* faz parte, separando os nomes das *tags* nos grupos entradas digitais, saídas digitais, entradas analógicas, saídas analógicas e memórias para posterior envio ao *broker* MQTT.

Esta rotina realiza uma conversão de endereçamentos onde os endereços fornecidos pela tabela de *tags*, apresentados na formatação Siemens, são convertidos para o padrão de endereçamento Modbus. Após a conversão de endereçamentos são obtidos os endereços onde serão iniciadas as leituras e a quantidade de dados que devem ser lidos nas entradas digitais, saídas digitais, entradas analógicas, saídas analógicas e *holding registers* no caso de existirem registradores desses respectivos grupos na tabela de *tags* carregada.

#### *3.3.4.1 Conversão de endereçamento Siemens em endereçamento Modbus*

O protocolo Modbus possui um endereçamento de registradores diferente do endereçamento utilizado pela Siemens. Enquanto a Siemens, para endereçar os registradores de seus CLPs, usa as formatações (1), (2) e (3), sendo (1):

$$X\ n_1.n_2 \tag{1}$$

Para bits, onde $(X)$ representa o grupo de registradores ao qual pertence o bit: $(I)$ para o caso de uma entrada, $(Q)$ para o caso de uma saída e $(M)$ para o caso de

uma memória; $(n_1)$ representa o número do byte do qual o respectivo bit faz parte e $(n_2)$ representa a posição do bit em questão dentro do byte que ele faz parte.

Sendo (2):

$$XWn \tag{2}$$

Para words, onde $(X)$ representa o grupo de registradores ao qual pertence a word: $(I)$ para o caso de uma entrada, $(Q)$ para o caso de uma saída e $(M)$ para o caso de uma memória; $(W)$ indica que o registrador em questão é do tipo word e $(n)$ indica a posição da word dentro do grupo de registradores do qual ela faz parte.

E, por fim, a expressão (3):

$$XBn \tag{3}$$

Para bytes, onde $(X)$ representa o grupo de registradores ao qual pertence o byte: $(I)$ para o caso de uma entrada, $(Q)$ para o caso de uma saída e $(M)$ para o caso de uma memória; $(B)$ indica que o tipo do registrador em questão é do tipo byte; $(n)$ indica a posição do byte dentro do grupo de registradores do qual ele faz parte.

Diferentemente do endereçamento utilizado pela Siemens, o protocolo Modbus representa os registradores de 0 a 65535 em cada grupo de registradores. Para transformar os endereços que são providos pela tabela de tags em um valor compatível com o padrão de endereçamento Modbus, se fez necessário realizar uma transformação de endereços. Essa transformação leva o endereço Siemens de um determinado registrador no seu equivalente endereço dentro de seu grupo de acordo com o modelo de endereçamento Modbus (valor entre 0 e 65535). A transformação não é a mesma para todos os tipos de dados, pois as sintaxes do endereçamento Siemens para cada tipo de dado (bit, byte e word) são diferentes e cada tipo de registrador tem suas particularidades dentro do hardware Siemens. O tipo de registrador envolvido é identificado no protocolo Modbus através da função Modbus relacionada à requisição realizada.

Semelhantemente, a biblioteca utilizada no software para comunicação com o CLP utilizando protocolo Modbus TCP, biblioteca PyModbusTCP, utiliza funções específicas para leitura de cada grupo de registradores, ficando assim subentendido qual tipo de dado está envolvido na requisição, sendo necessário apenas fornecer a

posição dentro do grupo de registradores na qual serão iniciadas as requisições e a quantidade de dados que serão lidos a partir do endereço de início. A biblioteca realiza procedimentos internos que se preocupam em prover esta informação na forma de requisição Modbus ao servidor Modbus TCP. Por conta disso, as transformações apresentadas nas seções de 3.3.4.1.1 a 3.3.4.1.5 levam padrões de endereçamento Siemens em endereçamento Modbus.

*3.3.4.1.1 Transformação de endereçamento tipo bit para entradas e saídas digitais*

O endereçamento Siemens para variáveis tipo bit apresentado em (1) informa a posição do bit em função do byte no qual ele está inserido. Para transformar endereços nesta formatação em um endereçamento compatível com o utilizado pelo Modbus, que informa a posição do bit dentro de seu grupo de registradores se fez necessário utilizar a expressão (4):

$$Xn_1.n_2 \rightarrow (8 * n_1) + n_2 \tag{4}$$

Onde $(X)$ representa o grupo de registradores do qual o bit faz parte e pode ser $(I)$ para o caso de uma entrada ou $(Q)$ para o caso de uma saída; $(n_1)$ representa o número do byte do qual o bit faz parte; $(n_2)$ representa a posição do bit dentro do byte ao qual ele pertence e $(\rightarrow)$ significa “leva a”, indicando uma transformação.

Exemplificando, o bit de entrada com endereço I1.2, é o bit de entrada de número 10. Há 8 bits de entrada do byte 0 em sua frente (bits 0 a 7) e mais dois bits do byte 1 em sua frente (bits 0 e 1, posicionamento 8 e 9), fazendo assim com que ele seja o bit de número 10 dentro do grupo de registradores de entradas digitais. O avanço de um byte a mais nos registradores representa o avanço de 8 bits. Por isso multiplica-se o número do byte no qual o bit faz parte por oito. Da mesma forma, o recuo de um byte nos registradores representa o recuo de 8 bits no endereçamento Modbus.

Semelhantemente, o bit de saída de endereço Q0.7 é o bit de posição 7 no grupo de saídas digitais. Há 7 bits em sua frente no byte 0, os bits de Q0.0 a Q0.6, compreendendo os endereços de 0 a 6 e fazendo com que o bit Q0.7 seja o de endereço 7 dentro do grupo de saídas digitais.

*3.3.4.1.2 Obtenção de endereçamento tipo byte para entradas e saídas digitais*

O endereçamento Siemens para variáveis do tipo byte segue o apresentado em (3). A requisição de um byte é identificada no PyModbusTCP como a requisição de oito bits. Sendo desta forma, se faz necessário obter através do endereçamento Siemens a posição inicial do byte dentro do grupo de bits. Para obter a posição inicial do byte foi utilizada a transformação (5):

$$XBn \rightarrow 8 * n \tag{5}$$

Onde $(X)$ representa o grupo de registradores do qual o byte faz parte, podendo ser $(I)$ representando uma entrada ou $(Q)$ representando uma saída; $(B)$ indica que o registrador em questão é do tipo byte; $(n)$ representa a posição do byte e $(\rightarrow)$ significa "leva a" indicando uma transformação que leva à posição do bit menos significativo do byte.

O posicionamento inicial do byte de entrada IB1, é a posição oito. Estão na frente dele todos os bits do byte de entrada número 0, ou seja, os bits de posição 0 a 7. O posicionamento inicial do byte de entrada QB2, é a posição dezesseis. Estão na frente dele todos os bits dos bytes de saída 0 e 1, ou seja, os bits de posição 0 a 7 (byte de saída 0) e os bits de posição 8 a 15 (byte de saída 1).

*3.3.4.1.3 Transformação de Endereçamento de Entradas Analógicas*

O endereçamento de entradas analógicas no CLP Siemens S7-1200 possui uma peculiaridade com relação aos demais registradores. São reservados para um registrador de entrada analógica duas words. Essa peculiaridade faz com que a relação entre o endereçamento Siemens de entradas analógicas para o endereçamento utilizado pelo PyModbusTCP não seja direto, ou seja, que o índice utilizado no endereçamento Siemens leve ao mesmo valor no endereçamento Modbus.

Podem ser utilizados como endereçamentos analógicos valores como IW0, IW2, IW4 e etc. Sempre utilizando índices pares. A word seguinte, de índice ímpar, se torna reservada junto com a word anterior.

Este fato faz com que a word IW0 seja endereçada como registrador de entrada analógica 0; a IW2, endereçada como registrador de entrada analógica 1; IW4 como registrador de entrada analógica 2 e assim sucessivamente. Notou-se, a partir deste fato, que a relação de endereçamento Siemens e endereçamento no PyModbusTCP se dá por uma divisão por fator 2 apresentada em (6):

$$IWn \rightarrow floor\left(\frac{n}{2}\right) \tag{6}$$

Onde $(IW)$ indica que o registrador em questão é uma entrada do tipo word; $(n)$ representa o índice do registrador dentro das entradas analógicas; $(\rightarrow)$ significa “leva a” indicando uma transformação e $(floor)$ representa o arredondamento para baixo do quociente da divisão e se dá como uma garantia em caso de possíveis valores de entradas inesperados envolvendo índices ímpares de word. Como $(n)$ para entradas analógicas sempre é par, não há risco de problemas em situações normais de uso.

Esta relação foi comprovada através de mapeamento de variáveis. Foi alterado o registrador da entrada analógica utilizada pelo CLP S7-1200 seguidas vezes e observado qual o endereçamento Modbus onde estava armazenado o conteúdo da variável através da leitura de registradores de entradas analógicas um por um. Foi utilizado um *script* em Python para a captura dos valores dos registradores de entrada analógica e seu posicionamento.

#### *3.3.4.1.4 Transformação de endereçamento para saídas analógicas*

Não existe dentro das funções Modbus uma função dedicada a ler saídas analógicas ou outros registradores de saída que não sejam bits. Apesar deste fato, a leitura de registradores de saída analógica é possível.

Um registrador de saída analógica é compreendido por uma word. Uma word corresponde a um conjunto de 16 bits. Estes 16 bits representam um determinado valor que deve ser aplicado na saída analógica correspondente do CLP em forma de tensão.

Não se pode ler o registrador de maneira direta, mas, pode-se ler os 16 bits que compreendem o registrador de saída analógica e, posteriormente, transformá-los para o formato de apresentação desejada, seja em hexadecimal, decimal, percentual ou

outras. Para obter a leitura desejada é necessário apenas obter o endereço de início do registrador de saída. A quantidade de bits que deve ser obtida, por ser uma word, é definida como 16 bits.

Após mapeamento de variáveis, realizando alterações no registrador da saída analógica e observando onde os bits pertencentes ao registrador de saída analógica eram armazenados, notou-se que a relação entre endereçamento Siemens de saída analógica e o bit inicial do correspondente registrador de saída analógica é dada por (7):

$$QWn \rightarrow n * 8 \tag{7}$$

Onde $(QW)$ indica que o registrador em questão é uma saída do tipo word; $(\rightarrow)$ significa “leva a” indicando uma transformação e $(n)$ representa o índice atribuído ao registrador de saída analógica.

*3.3.4.1.5 Endereçamento de holding registers*

Por serem configurados nesta aplicação como variáveis dentro de um *data block* não otimizado, *holding registers* não são apresentados na tabela de *tags* principal, o que dificulta seu monitoramento uma vez que as variáveis monitoradas estão na tabela de *tags* padrão (*Default tag table)*. Mas, para que fossem utilizados na transmissão de dados e monitoramento de variáveis, receberam valores de registradores de memória que são utilizados ao longo da programação. A operação pode ser realizada através de um bloco chamado MOVE presente no Tia Portal e a opção de uso em futuras utilizações foi liberada como uma escolha do usuário.

Ao decidir utilizar o monitoramento de parâmetros de memória através de *holding registers,* é necessário usuário criar uma *tag* de memória do tipo word e associá-la ao respectivo *holding register*. A relação utilizada entre *holding register* e memória no software foi direta. Ou seja, a word de memória MW0 foi levada ao *holding register* 0, a MW1 ao *holding register* 1 e assim sucessivamente. Desta forma, no presente trabalho, uma *tag* de memória do tipo word é associada a um *holding register*.

Após os endereços recebidos via tabela de *tags* serem convertidos em um padrão de endereçamento Modbus, padrão aceito pela biblioteca PyModbusTCP, os

valores dos endereços convertidos foram armazenadas em listas, separados de acordo com o grupo de registrador ao qual pertencem e de acordo com o tipo de variável que armazenam, ou seja, separados em: endereços de entradas analógicas, endereços de saídas analógicas, endereços de saídas digitais tipo bit, endereços de saídas digitais tipo byte, endereços de entradas digitais tipo bit, endereços de entradas digitais tipo byte e memórias.

A divisão no entre bit e byte para variáveis digitais se dá por conta do tratamento que cada tipo de variável precisará receber na montagem das requisições e também após a chegada dos dados. A requisição realizada por bytes possui um tamanho mínimo de 8 bits (no caso de um byte apenas) enquanto as requisições realizadas de tipo bit possuem tamanho livre (desde que seja número inteiro maior que zero).

Após os endereços recebidos via tabela de *tags* serem convertidos em um padrão de endereçamento Modbus, aceito pela biblioteca PyModbusTCP, e agrupados de acordo com o grupo de registradores aos quais fazem parte, foi possível obter os parâmetros necessários para a requisição de leitura de dados utilizando o protocolo Modbus TCP. A forma de obtenção dos parâmetros para realização da requisição é apresentada na seção 3.3.4.2.

#### *3.3.4.2 Obtenção dos parâmetros de requisição de leitura*

Como apresentado no início de 3.3.4, as requisições de leitura de dados utilizando o protocolo Modbus requerem o endereço onde serão iniciadas as leituras e a quantidade de dados que serão lidos a partir do endereço onde será iniciada a leitura. Após converter os endereços fornecidos através da tabela de *tags* em endereços compatíveis com o endereçamento utilizado no PyModbusTCP, pode-se obter os parâmetros necessários para criar as requisições de leitura de dados.

Realizar as requisições dado por dado seria muito oneroso computacionalmente em caso de grande número de dados a ser lidos. Por conta disso, a metodologia apresentada nesta seção visa reduzir a quantidade de requisições realizadas de modo a reduzir o tempo gasto na obtenção dos dados no processo de leitura de dados do CLP.

A programação neste trabalho foi montada para que dados de mesmo tipo e grupo de registradores fossem requisitados de uma única vez. Por exemplo, que saídas digitais do tipo bit sejam requisitadas todas ao mesmo tempo, que todas as entradas digitais do tipo byte sejam requisitadas ao mesmo tempo, que entradas analógicas sejam requisitadas ao mesmo tempo, reduzindo assim o tempo gasto com a obtenção de dados do CLP como desejado.

Com os endereços de todas as *tags* separadas em seus respectivos grupos, armazenados em listas, os valores do endereço onde começarão a ser lidos os dados e a quantidade de registradores foram obtidos, respectivamente, através das expressões (8) e (9):

$$end_{início} = \min(lista_{end}) \tag{8}$$

$$tam_{dados} = max(lista_{end}) - \min(lista_{end}) + 1 \tag{9}$$

Onde $(end_{início})$ representa o endereço onde serão iniciadas as leituras no grupo de registradores em questão; $(lista_{end})$ representa a lista que contém os endereços das variáveis utilizadas de um determinado grupo de registradores; $(tam_{dados})$ representa a quantidade de dados que serão requisitados; $(min)$ é o método Pyhton pelo qual se obtém o menor valor contido na em uma lista e $(max)$ é o método Python pelo qual se obtém o maior valor contido em uma lista.

Ou seja, o endereço de onde será iniciada a leitura em um determinado tipo de dado é o endereço que possui o valor mais baixo no endereçamento Modbus dentre os endereços que estão na lista. Para que todos os dados sejam lidos de uma única vez, a lista de dados recebidos da requisição deve ter o tamanho da diferença entre o maior endereço que deve ser lido e o menor endereço que deve ser lido somado de 1 unidade, pelo fato do primeiro dado também estar contido na lista de dados que será recebida. No software desenvolvido, o trabalho de cálculo do endereço de início de leitura e tamanho do frame de dados que deve ser solicitado é realizado por uma função apresentada na Figura 16. Caso não haja dados para ser colhidos, ela retorna uma lista com valores *false*, valor que informa que não há dados para ser solicitados naquele tipo de dado.

Figura 16 - Função utilizada para obtenção de tamanho de frame de dados para leitura e endereço de início de leitura.

```
def c_max(self, lista):
    '''
        Caso não haja dados para ser monitorados em um registrador
        a lista passada como parâmetro está vazia
    '''
    if not lista:
        return [False, False, False]

    # Caso existam dados para ser monitorados,
    # são obtidos o maior endereço e o menor endereço Modbus
    # entre os dados requisitados e calculado o tamanho do frame
    # de dados que deverá ser lido
    else:
        mx_lst = max(lista)
        mn_lst = min(lista)
        df_lst = mx_lst - mn_lst + 1
        return [mx_lst, mn_lst, df_lst]
```

Fonte: Próprio autor.

Uma vez definidas a metodologia para se obter os parâmetros para realizar as requisições de dados ao servidor Modbus TCP, o CLP S7-1200, e a implementação desta metodologia em software, foram estabelecidas as comunicações entre CLP e servidor local, utilizando o protocolo Modbus TCP e a comunicação entre servidor local e *broker* MQTT externo (situações apresentadas nas seções 3.3.5 e 3.3.6, respectivamente). Posteriormente, foram desenvolvidas rotinas para que sejam requisitados os dados ao CLP e, posteriormente, enviá-los ao *broker* MQTT. O desenvolvimento e implementação destas rotinas são apresentadas na seção 3.3.7.

### *3.3.5 Conexão entre servidor local e CLP*

Após a obtenção dos parâmetros necessários para se realizar as requisições de dados, foi estabelecida em software a comunicação entre o servidor local e o CLP S7-1200 utilizando o protocolo Modbus TCP. Para estabelecer a comunicação entre esses dois dispositivos foi utilizada a biblioteca Python PyModbusTCP. A biblioteca PyModbusTCP permite acesso a um servidor Modbus TCP através de um objeto *ModbusClient* [30].

A biblioteca PyModbusTCP foi responsável por realizar a abertura de comunicação com CLP S7-1200 através de um objeto *ModbusClient*, que representa um cliente Modbus. A utilização do método *open* deste objeto fez do servidor local um cliente Modbus TCP do CLP S7-1200, previamente configurado como servidor Modbus TCP (seção 3.2.1), ao qual ele já estava conectado fisicamente por meio de cabos de pares trançados. Como parâmetros necessários para a abertura de conexão foram requeridos pelo objeto *ModbusClient* o IP do CLP e a porta por onde se dará a comunicação via protocolo Modbus TCP, parâmetros que foram recebidos como inserção do usuário (seção 3.3.3).

No PyModbusTCP, a requisição de dados é realizada através de funções presentes na biblioteca que são baseadas nas funções Modbus. As funções Modbus são ações que podem ser realizadas no servidor Modbus através de solicitação realizada pelo cliente Modbus utilizando o protocolo. Cada função Modbus tem um código que a identifica dentro do protocolo.

No software desenvolvido foram utilizadas apenas funções de leitura. A possibilidade de escrita em variáveis sendo realizada de maneira remota pode constituir um risco à segurança de um sistema caso não sejam tomadas as devidas precauções.

As funções de leitura presentes no PyModbusTCP incorporadas ao software utilizado no servidor local foram leitura de entradas digitais (*read_discrete_inputs*), leitura de entradas analógicas (*read_input_registers)*, leitura de saídas digitais (*read_coils)* e leitura de *holding registers* (*read_holding_registers)*. A leitura de saídas analógicas, como apresentado na seção 3.3.4.1.4, é obtida realizando a leitura dos 16 bits que constituem um registrador de saída analógica no CLP S7-1200 da Siemens. A Figura 17 apresenta a correspondência na biblioteca PyModbusTCP para algumas funções Modbus suportadas na referida biblioteca.

Figura 17 - Funções Modbus que possuem correspondência no PyModbusTCP.

| Dado | Nome da função Modbus | Código da função | Função no objeto ModbusClient |
|---|---|---|---|
| Bit | Read Discrete Inputs | 2 | `read_discrete_inputs()` |
| | Read Coils | 1 | `read_coils()` |
| | Write Single Coil | 5 | `write_single_coil()` |
| | Write Multiple Coils | 15 | `write_multiple_coils()` |
| Register | Read Input Registers | 4 | `read_input_registers()` |
| | Read Holding Registers | 3 | `read_holding_registers()` |
| | Write Single Register | 6 | `write_single_register()` |
| | Write Multiple Registers | 16 | `write_multiple_registers()` |

Fonte: Adaptado de [30].

Após abertura de conexão com servidor Modbus TCP, o CLP, é realizada a conexão do servidor local a um *broker* MQTT externo. A metodologia para conexão do servidor local com o *broker* MQTT externo pelo software desenvolvido é apresentado na seção 3.3.6.

### *3.3.6 Conexão entre servidor local e broker MQTT externo*

No desenvolvimento deste projeto foi utilizado um *broker* MQTT externo fornecido pela empresa EMQ, o EMQX. Segundo [31], a EMQ é uma empresa que fornece softwares de estrutura de dados IoT de código aberto. O *broker* MQTT utilizado é fornecido pela empresa para desenvolvimento de softwares. Em caso de utilização em usos comerciais, a utilização do *broker* EMQX torna-se paga.

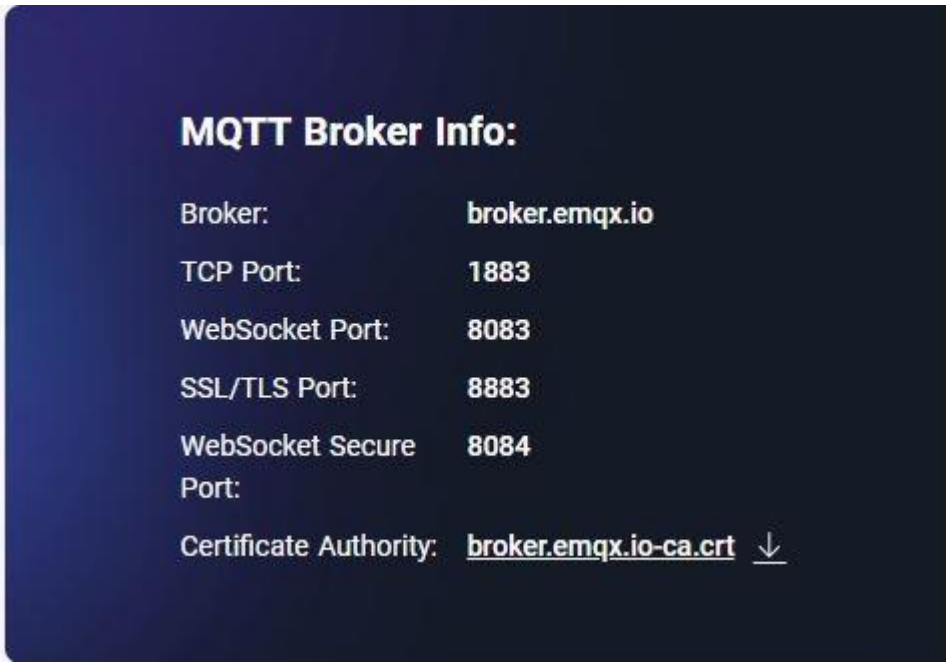
A conexão do servidor local com o *broker* MQTT EMQX é realizada através do software desenvolvido utilizando a biblioteca Python Paho MQTT. A biblioteca Paho MQTT foi desenvolvida pela Eclipse Foundation e é a implementação do projeto Paho da Eclipse Foundation dentro do Python. O projeto Eclipse Paho disponibiliza implementações de código aberto de protocolos de mensagens utilizados em aplicações que utilizam as filosofias de *Machine-to-Machine* (M2M) e Internet das Coisas (IoT) [32].

Foi desenvolvida no software uma rotina responsável pelo estabelecimento da comunicação entre *broker* MQTT e servidor local. Nesta rotina também são implementadas *callbacks* para eventos de conexão estabelecida, conexão perdida e publicação de mensagens no *broker* bem-sucedidas por parte do software

desenvolvido. *Callbacks* são rotinas que são realizadas a partir do acontecimento de determinados eventos.

Dentro da rotina de estabelecimento de conexão entre servidor local e *broker* MQTT, a comunicação entre os dois dispositivos é realizada mediante a criação de um objeto *Client*, e, posteriormente a utilização do método *connect* deste objeto, responsável por abrir uma conexão com o *broker* MQTT desejado. Para que a conexão pudesse ser estabelecida, foi necessário utilizar as informações do *broker* EMQX, fornecidos pela EMQ, apresentados na Figura 18. A rotina utilizada para estabelecer a conexão entre o servidor local e o *broker* EMQX é acionada ao mesmo tempo em que o usuário insere os dados necessários para comunicação com o CLP, reduzindo o tempo de processamento do acódigo visando um estabelecimento de comunicação mais rápido. O processamento de rotinas paralelamente é realizado no software utilizando a biblioteca Python *Threading*.

Figura 18 - Parâmetros de conexão ao *broker* público EMQX.




Fonte: Retirado de [33].

Após o estabelecimento da conexão entre servidor local e *broker* MQTT, se fez necessário habilitar o envio de dados ao *broker* MQTT e a captura de eventos que ocorrem na conexão entre os dois dispositivos, responsáveis por acionar os *callbacks.*

Este trabalho é realizado através da utilização do método “*loop_start”* do objeto *Client*. Com a utilização deste método, o *Client* envolvido na conexão fica à espera de novas mensagens para publicação e fica à espera de novas mensagens em caso de inscrição em algum tópico. No caso do software utilizado no servidor, como não é feita inscrição em nenhum tópico, apenas envio de dados por meio de publicações, o cliente ficou à espera apenas de novas mensagens para publicação e capturando eventos ocorridos durante à conexão. Uma vez utilizado o método *loop_start*, este também se torna responsável pelos processos de reconexão em caso queda da conexão entre servidor local e o *broker* EMQX.

### *3.3.7 Rotinas de Requisição de Dados e envio de informações*

Após definidos os parâmetros para requisição de leituras de registradores do CLP e realizadas as devidas conexões ao CLP e ao *broker* MQTT, são acionadas rotinas que tem o objetivo de ler os dados do CLP, tratar os dados de chegada de forma a retirar os dados que interessam ao usuário e, posteriormente, enviá-los ao *broker* MQTT para ser disponibilizados aos dispositivos móveis de usuários permitidos.

No total foram desenvolvidas cinco diferentes rotinas responsáveis por ler os dados dos registradores no CLP, tratar os dados enviados pelo CLP e publicar os dados no *broker* MQTT EMQX, uma por realizar o procedimento descrito em entradas digitais, outra por realizá-lo em saídas digitais, outra em entradas analógicas, uma em saídas analógicas e, por fim realizá-lo em *holding registers*, registradores que foram associados às memórias do tipo word (associação apresentada na seção 3.3.4.1.5). A separação das rotinas dessa forma, permite que o processo executado pelas rotinas seja realizado mais rapidamente como um todo pela possibilidade de se utilizar de programação paralela.

A programação paralela, neste caso, fornece a vantagem de permitir que o tratamento de dados recebidos não seja um gargalo maior na transmissão de dados ao *broker* MQTT. Caso as rotinas não fossem executadas paralelamente, uma rotina só poderia ser executada após a finalização da outra, fazendo com que uma

requisição de dados ao CLP só pudesse ser realizada após a finalização da rotina anterior (o que inclui obtenção dos dados do CLP, tratamento de dados e publicação no *broker* MQTT), fazendo com que o CLP e o *broker* MQTT ficassem ociosos durante a realização do tratamento de dados obtidos. A utilização de programação paralela faz que, com a execução das rotinas ao mesmo tempo, o tratamento de dados tenha um impacto menor no tempo total do processo.

As rotinas operam de maneira cíclica em intervalos de tempo de 1 segundo, sendo acionadas pelo objeto *Clock* da biblioteca *Kivy*, objeto responsável por acionar rotinas que foram previamente agendadas. A operação das rotinas é agendada ou não de acordo com a presença de *tags* do respectivo tipo de registrador na tabela de *tags* carregada pelo usuário. Caso não haja *tags* criadas de entradas analógicas na tabela de *tags* carregada pelo usuário não será realizada agendada a execução cíclica da rotina que executa leitura de entradas analógicas, por exemplo.

As execuções contidas nas rotinas são dissecadas nas seções 3.3.7.1 e 3.3.7.2. A forma como foram realizadas as requisições de leituras e como foram tratados os dados recebidos do CLP é apresentado na seção 3.3.7.1 e a publicação no *broker* MQTT dos valores dos registradores após tratamento de dados é apresentado na seção 3.3.7.2.

### *3.3.7.1 Requisição de leituras ao CLP e tratamento de dados recebidos*

A requisição de leituras de registradores ao CLP é realizada através da biblioteca PyModbusTCP mais precisamente pelo objeto *ModbusClient*. Para cada tipo de registradores envolvidos na requisição é utilizada uma função específica do objeto *ModbusClient*. As funções utilizadas dentro das rotinas criadas para cada leitura de registradores de acordo com cada grupo de registrador envolvido na requisição são apresentadas na tabela 1.

Tabela 1 - Associação realizada entre funções de *ModbusClient* e registradores do CLP S7-1200.

| Funções de leitura de registradores de *ModbusClient* |
|---|

| | |
|---|---|
| Entradas digitais | *read_discrete_inputs* |
| Saídas digitais | *read_coils* |
| Entradas analógicas | *read_input_registers* |
| Saídas Analógicas | *read_coils* |
| Memórias do tipo word | *read_holding_registers* |

Fonte: Próprio autor.

Como resposta à requisição de leitura é recebido uma lista. Caso a lista recebida esteja vazia, representa que a conexão com o servidor Modbus TCP, o CLP no caso em questão, não foi estabelecida de maneira bem-sucedida. Caso a requisição tenha sido bem-sucedida, são devolvidos em uma lista os respectivos valores requisitados, sendo o valor de menor endereço Modbus o primeiro da lista e seguindo assim de forma crescente, de endereço a endereço, até o último requisitado.

No caso do grupo de registradores do qual foi requisitada leitura armazenar variáveis do tipo booleano (ou seja, 1 bit), os valores recebidos pela lista estarão nas formas *True* em caso de 1 lógico, ou *False* em caso de 0 lógico. No caso do grupo de registradores do qual foi requisitada leitura armazenar variáveis do tipo word, a lista recebida conterá valores em hexadecimal correspondente ao valor armazenado no CLP nos registradores em questão.

#### *3.3.7.1.1 Tratamento de dados recebidos*

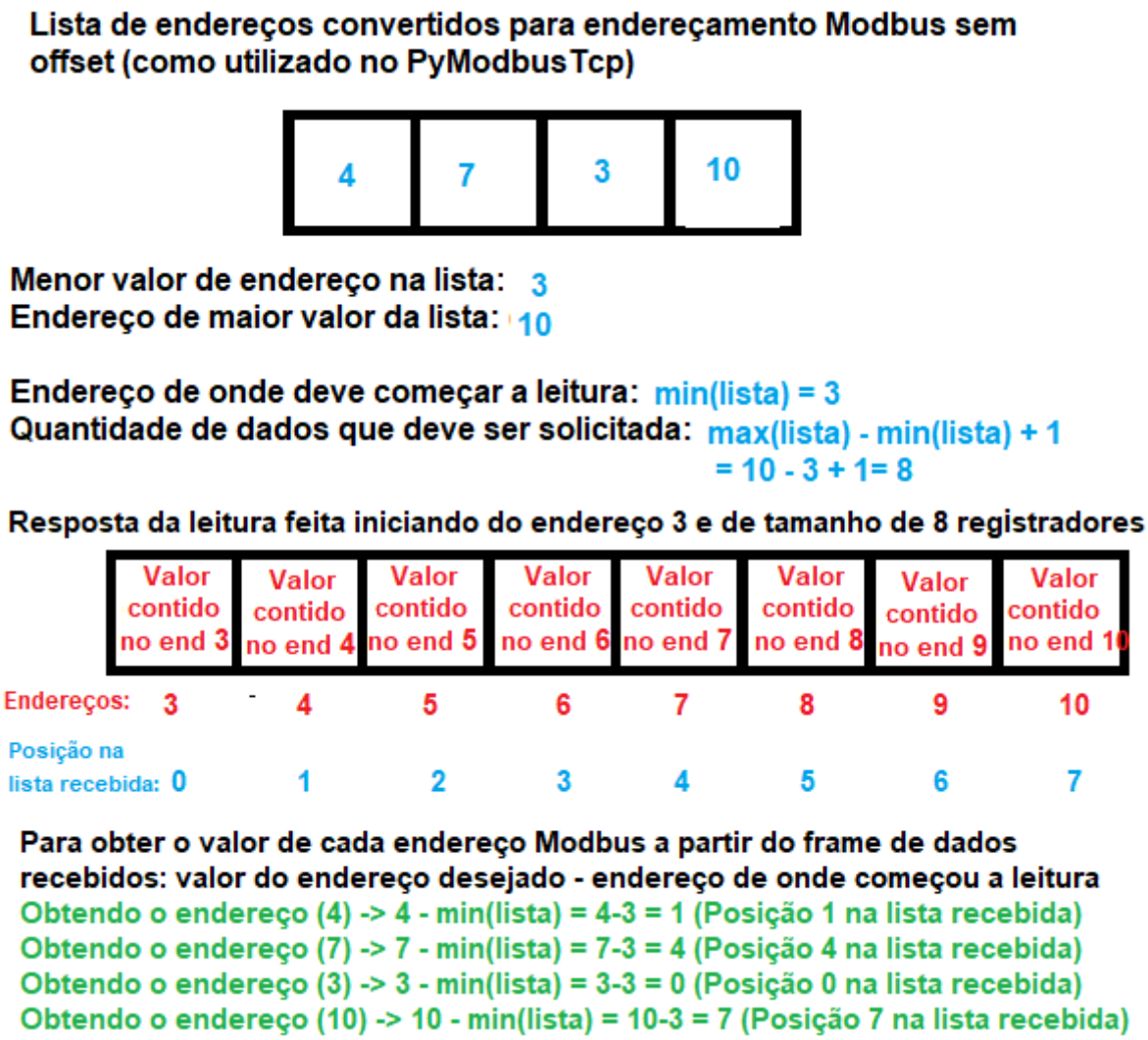
Como citado na seção 3.3.4.2, a programação em software foi desenvolvida para que os dados pertencentes ao mesmo grupo de registradores sejam requisitados de uma única vez. Essa metodologia requere que sejam tratados os dados de chegada por conta da possibilidade de presença de dados não desejados presentes na lista de chegada.

Esse fenômeno ocorre por conta da natureza das requisições no protocolo Modbus, onde não se pode requisitar leitura ou escrita em registradores de endereços não vizinhos de uma única vez sem envolver os registradores que estão entre eles.

Portanto, para economizar tempo no processamento de leitura de registradores, foram requisitados ao CLP a leitura dos valores desde o menor endereço desejado até o de maior endereço desejado, e, posteriormente, esses dados foram tratados.

Desta forma, para colher da lista recebida apenas os dados contidos nos registradores presentes na tabela de *tags*, são utilizados os endereços Modbus de cada *tag* (convertidos do endereçamento Siemens para o endereçamento Modbus conforme a metodologia apresentada na seção 3.3.4.1). A lista recebida após cada requisição tem como primeiro valor, o elemento de endereço menor endereço Modbus dentro daqueles que estão presentes na tabela de *tags*. Portanto, ele é o elemento de posição 0 na lista. Ele é o elemento 0 porque os elementos que vem antes dele não foram requisitados, portanto, não fazem parte da lista de resposta. Desse modo, a lista fica deslocada no valor do endereço Modbus de menor índice e foi necessário, para obter a posição de cada elemento desejado dentro da lista, subtrair o endereçamento Modbus real (obtido pelos métodos apresentados na seção 3.3.4.1) pelo termo de menor endereço na lista. A Figura 19 apresenta uma exemplificação de todo o processo de leitura e tratamento de dados envolvendo uma requisição de leitura de dados de um tipo de registrador qualquer utilizando o PyModbusTCP.

Figura 19 - Exemplificação do processo de leitura e tratamento de dados realizado.




Fonte: Próprio autor.

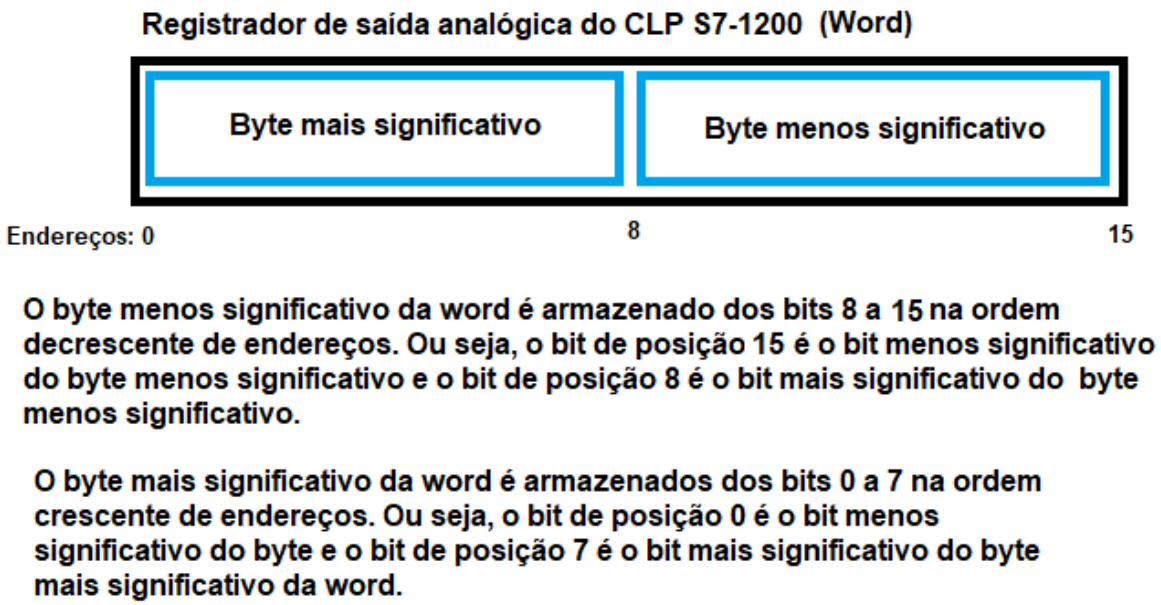
Os dados recolhidos mediante o processo apresentado na Figura 19, são armazenados em listas. Esse processo de tratamento de dados é utilizado em entradas digitais do tipo bit, saídas digitais do tipo bit, entradas analógicas e *holding registers.* Essas funções possuem funções Modbus específicas dentro do protocolo Modbus para leitura e escrita. Saídas analógicas e entradas e saídas digitais do tipo byte outros métodos de tratamentos de dados.

O tratamento de entradas e saídas digitais do tipo byte envolve a obtenção dos 8 bits que correspondem ao byte. Para obter o valor correspondente ao byte, se faz necessário capturar os 8 bits iniciando pelo bit de menor endereço do byte. Havendo mais de 1 byte para ser requisitado, a requisição de leitura é feita do bit menos

significativo do byte de menor endereço ao bit mais significativo do byte de maior endereço. Posteriormente, os 8 bits de cada byte são recolhidos. Os bytes captados são transformados posteriormente em um número de 0 a 255 para prover um melhor entendimento ao usuário

A leitura de saídas analógicas foi realizada através da leitura dos 16 bits que compreendem o registrador de saída analógica. Posteriormente, ao obter-se os 16 bits correspondentes ao registrador de saída analógica, foi necessário convertê-los considerando a forma na qual são armazenados os registradores de saídas analógicas apresentado na Figura 20. A conversão foi realizada de modo a transformar os 16 bits em um número inteiro. Posteriormente à conversão, o valor obtido é armazenado em uma lista assim como foram todos os outros tipos de dados.

Figura 20 - Representação de armazenamento de saídas analógicas no CLP S7-1200.




Fonte: Próprio autor.

Após o armazenamento em listas, os dados de mesmo grupo de registradores são todos unidos em uma única *string* e rotulados com uma estampa de tempo indicando o momento de recebimento. Uma estampa de tempo é o valor temporal no qual foi obtida a informação. A estampa de tempo permitirá que o usuário de dispositivos móveis possa monitorar o comportamento da *tag* em função do tempo.

Posteriormente a essa preparação, os valores dos registradores são encaminhados para publicação no *broker* MQTT EMQX.

#### *3.3.7.2 Publicação de dados obtidos no Broker MQTT*

Após o tratamento de dados ser completado, os valores de cada *tag* estão prontos para ser publicados no *broker* MQTT, podendo ser assim enviados aos dispositivos móveis dos usuários que obtém a permissão para monitoramento de variáveis da planta.

A publicação utilizando o protocolo de comunicação MQTT requere do publicador um tópico ao qual a mensagem enviada será associada. Ao ser associada a um determinado tópico, o conteúdo da mensagem será repassado a todos os dispositivos que são assinantes deste tópico.

Como um modo de facilitar a recepção de dados por parte do aplicativo desenvolvido para uso nos dispositivos móveis, cada tipo de dado possui seu tópico específico para envio de mensagens. Separar os tópicos de acordo com os tipos de registradores também reduz problemas de gargalos com tratamento de dados, uma vez que não é necessário esperar que os dados de todos os tipos de registradores estejam prontos para que sejam realizados uma única publicação. Dessa forma, a publicação de dados de um determinado tipo de registradores pode ser realizada independentemente do que acontece nos outros tipos de dados.

Como o software foi idealizado para um uso em mais de um processo industrial simultaneamente, os tópicos utilizados no software não podem ser os mesmos em cada planta distinta. Caso isso ocorresse haveria uma mistura de dados, onde os dados lidos dos registradores de uma planta iriam se sobrescrever aos da outra, causando assim uma interferência e, consequentemente, impossibilitando a viabilidade do monitoramento.

Desta forma, para que fossem evitados estes problemas de interferência foram utilizados nos nomes dos tópicos informações específicas da planta. A informação utilizada foi o identificador criado pelo *Realtime Database* do *Google Firebase* no momento do cadastro das informações da planta no referido banco de dados. Sendo assim, os tópicos utilizados em cada tipo de uma determinada planta apresentados

na tabela 2. A Figura 21 apresenta o identificador dos dados de uma planta registrada no *Realtime Database* do *Google Firebase* utilizado no referido projeto.

Figura 21 - Exemplo de identificador gerado após cadastro de planta no *Realtime Database* do *Google Firebase*.


Fonte: Próprio autor.

Tabela 2- Tópicos utilizado para cada publicação no *broker* MQTT.

| Tópicos utilizados para cada publicação no broker | |
|---|---|
| Grupo de Registrador | Nome do tópico utilizado |
| Entradas Digitais | $entdig(id\_planta)$ |
| Saídas Digitais | $saidig(id\_planta)$ |
| Entradas Analógicas | $entana(id\_planta)$ |
| Saídas Analógicas | $saiana(id\_planta)$ |
| *Holding Registers* | $memoria(id\_planta)$ |

Fonte: Próprio autor.

Onde $(id\_planta)$ representa o identificador dos dados da planta gerado pelo *Realtime Database* do *Google Firebase* sendo concatenado ao nome anterior.

Uma vez definidos os tópicos utilizados na publicação de mensagens para cada tipo de registrador, pode-se finalmente publicar as mensagens. A mensagem é publicada utilizando-se o método *publish* presente no objeto *Client* da biblioteca Paho MQTT.

O método *publish* do objeto *Client* exigiu como parâmetros de entrada o tópico da mensagem, já definido na tabela 2 e o conteúdo da mensagem (já definido em

3.3.7.1.1). Também foi passado como parâmetro o *QoS* que estará associado à mensagem, definido como 2 nesta utilização.

Também foi criada uma rotina durante o período de testes para informar ao usuário o resultado da tentativa de publicação da mensagem. Caso a mensagem seja publicada com sucesso, é informado ao usuário que a mensagem foi enviada com sucesso bem como o identificador da mensagem no *broker* MQTT (um número dado a mensagem com o objetivo de identificá-la entre as demais). Caso contrário, informa-se a ocorrência de um erro na publicação da mensagem em questão.

As rotinas de requisição de leitura de dados ao CLP e envio de dados ao CLP tem sua execução iniciada mediante comando do usuário sendo executadas ciclicamente. Durante sua execução são apresentados na interface gráfica do software que atua no servidor local a situação momentânea da conexão Modbus TCP e a situação momentânea da conexão com o *broker* MQTT. A execução destas rotinas é finalizada mediante comando do usuário.

### *3.3.8 Produção de Arquivo Executável*

Para que as máquinas que vão utilizar o software desenvolvido não tenham que ter um ambiente Python dentro de si e para que não seja disponibilizado diretamente o *script* produzido para que a aplicação seja utilizada, se fez necessário gerar um arquivo executável do *script* desenvolvido.

Para resolução desta questão foi utilizada uma biblioteca Python chamada Pyinstaller. O Pyinstaller une um *script* Python e todas os arquivos utilizados por ele em um único executável. O usuário pode utilizar o executável produzido sem instalar um interpretador Python ou qualquer módulo [34].

O Pyinstaller recebeu como entrada o *script* Python produzido, o arquivo *kv* desenvolvido associado ao *script* Python e as imagens e ícones utilizado no projeto. Como saída, o desenvolvedor recebe um arquivo executável que pode ser utilizado em qualquer dispositivo de mesmo sistema operacional no qual ele foi gerado, sem a necessidade de instalação de nenhum pacote adicional ou ambiente de desenvolvimento.

## 3.4 Desenvolvimento de Aplicativo para Smartphone

O aplicativo para o smartphone foi desenvolvido utilizando a linguagem de programação Python. Após a escolha da linguagem de programação Python foi definido o uso do *framework Kivy* para desenvolvimento do aplicativo, mais precisamente em sua extensão *Kivymd*. Os motivos para a utilização do *framework Kivy* no desenvolvimento das aplicações deste trabalho foram apresentados na seção 3.3.1

### *3.4.1 Sistema de login e recuperação de senha*

Primeiramente, no desenvolvimento do aplicativo foi criado o sistema de autenticação de usuários e de recuperação de senha. Assim como realizado no sistema de login e recuperação de senha no software desenvolvido para atuar no servidor local (apresentado na seção 3.3.2), foi utilizada a biblioteca Python *Requests* para se comunicar com o *Realtime Database* do *Google Firebase* pertencente ao projeto, através da REST API do *Google Firebase* com o intuito de obter as informações dos usuários cadastrados e verificar a veracidade das informações inseridas pelo usuário tanto na ação de login quanto na ação de recuperação de senha.

Para o sistema de recuperação de senha no aplicativo, foi desenvolvido um sistema que solicita do usuário em questão seu endereço de e-mail. Ao se verificar a existência desse endereço de e-mail dentre os usuários cadastrados no banco de dados citado, é enviado um e-mail com um link de uma página web, desenvolvida utilizando linguagens de programação web PHP e Javascript e formatação de texto HTML5. Esta página web foi desenvolvida com o objetivo de receber a nova senha digitada pelo usuário, encriptar esta senha na encriptação utilizada no desenvolvimento do trabalho, “sha256” (citada na seção 3.3.2), se comunicar com o *Realtime Database* do projeto criado no *Firebase* e realizar a alteração de senha.

### *3.4.2 Escolha de Planta para execução de monitoramento*

Após o desenvolvimento do sistema de login e recuperação de senha, foi desenvolvida uma interface que, após a realização do login por parte do usuário, permitisse a este uma escolha de qual planta seria monitorada no momento em questão. A tela interage com o usuário apresentando todas as plantas às quais ele tem acesso e aguarda por uma escolha do usuário sobre qual planta será monitorada no momento em questão.

Para que pudesse ser apresentado ao usuário todas as plantas às quais ele tem acesso a monitoramento na execução do aplicativo, foi implementada em código a realização de uma varredura entre as informações das plantas existentes do banco de dados *Realtime Database* do *Google Firebase* desenvolvido no projeto, procurando se há ou não a presença do usuário em questão dentre os usuários permitidos de cada planta. Para esta ação, foi utilizada a biblioteca Python *Requests* com o objetivo de requisitar as informações das plantas armazenadas no *Realtime Database* do *Google Firebase* desenvolvido. Após recebê-las, é realizada a procura pelo e-mail do usuário atual dentre os e-mails que possuem permissão para monitoramento de cada planta. Caso o e-mail do usuário em questão esteja presente, ele é um usuário permitido e deve aparecer a opção de acesso à monitoramento de dados da planta em questão, caso contrário não é.

### *3.4.3 Processo de abertura de comunicação MQTT*

Após a escolha da planta que será monitorada pelo usuário, é possível abrir comunicação com o *broker* MQTT conhecendo quais tópicos serão utilizados para a requisição de dados. Os tópicos são armazenados no *broker* MQTT pelo aplicativo que é executado no servidor utilizando o identificador gerado pelo *Google Firebase* no momento de cadastro da planta como elemento presente nos tópicos.

Após o desenvolvimento da interface de escolha da planta que será monitorada no momento, foi desenvolvida a abertura de comunicação com o *broker* MQTT da EMQ, o *broker* EMQX, onde são publicados os valores dos registradores do CLP e os nomes das *tags* criadas no desenvolvimento do código executado no CLP ambas

ações realizadas pelo software executado no servidor local. A abertura de comunicação com o *broker* EMQX é realizada através da utilização da biblioteca Python Paho MQTT mediante a execução de uma rotina desenvolvida especificamente com este objetivo. A conexão entre o *broker* MQTT e os dispositivos móveis foram executadas de modo semelhante à metodologia utilizada para estabelecer a conexão entre servidor local e *broker* MQTT, apresentada da seção 3.3.6.

Uma vez desenvolvida a conexão entre *broker* MQTT e o dispositivo móvel foi desenvolvida a assinatura de tópicos no aplicativo uma vez que se faz necessário que os dispositivos móveis recebam as informações armazenadas no *broker* MQTT sobre a planta monitorada. A escolha da planta por parte do usuário, realizada em passos anteriores, provê as informações necessárias para a assinatura dos tópicos que devem ser utilizados no momento, uma vez que cada planta possui tópicos exclusivos para si.

A assinatura de tópicos no aplicativo é realizada através da execução de uma rotina criada. Esta se utiliza do objeto *Client* da biblioteca Paho MQTT criado na rotina de abertura de conexão com o *broker* MQTT, para realizar a assinatura de tópicos por meio do método *subscribe* pertencente ao objeto citado, confirmando assim a assinatura dos tópicos desejados na comunicação aberta.

Juntamente com a criação de uma rotina responsável por realizar a inscrição em tópicos, se fez necessário criar uma sub-rotina dentro desta rotina. A criação desta teve por objetivo realizar a recepção de mensagens enviadas pelo *broker* MQTT. Dentro desta sub-rotina, foi desenvolvido um procedimento de tratamento de dados para organização das informações recebidas. A sub-rotina criada é utilizada como um *callback* para o evento de mensagens recebidas do *broker* MQTT.

#### *3.4.3.1 Tratamento de dados recebidos através da sub-rotina*

O tratamento de dados recebidos na sub-rotina executada no momento em que os dados são recebidos, tem como objetivo identificar os tópicos aos quais pertencem as mensagens recebidas e, posteriormente à identificação, tratar a mensagem recebida separando os dados corretamente (uma vez que eles vêm unidos em uma única *string*) e armazená-los de acordo com o tópico ao qual a mensagem recebida

pertence. A sub-rotina criada possui dois tipos diferentes de tratamentos: um para recebimento de nomes de *tags* e outros para recebimento de valores de registradores.

Caso a mensagem recebida contenha nomes de *tags*, a sub-rotina identifica a qual tipo de registradores pertence o nome das *tags* enviadas, e armazena os nomes das *tags* recebidas em uma lista, após realizar a separação dos nomes enviados em uma única *string*. Os nomes das *tags* enviados, obtidos da tabela de *tags* que é carregada na execução do programa do servidor, serão utilizados para informar ao usuário quais *tags* estão sendo monitoradas.

Caso a mensagem recebida contenha valores de registradores, a sub-rotina identifica a qual tipo de registradores pertencem os valores enviados, identificação realizada mediante o nome do tópico que está associado à mensagem, separa os valores enviados e os armazena em uma lista. Esses valores serão utilizados para informar o usuário o valor momentâneo da *tag* monitorada.

Após o desenvolvimento dos processos de abertura de comunicação entre o aplicativo desenvolvido e *broker* MQTT e, posteriormente, desenvolvimento de rotina de inscrição em tópicos e de sub-rotina de tratamento de mensagens recebidas, se fez necessário habilitar a recepção de dados e captura de eventos envolvendo a conexão entre o dispositivo móvel utilizado e o *broker* MQTT. A habilitação de recepção de dados e captura de eventos é realizada mediante a utilização do método *loop_start* pertencente ao objeto *Client* da biblioteca Paho MQTT assim como realizado no software desenvolvido que foi executado no servidor local, procedimento apresentado na seção 3.3.6.

Após a utilização do método *loop_start* no objeto *Client* criado na abertura de conexão, este fica à espera de mensagens enviadas e captura eventos ocorridos na conexão, tais como desconexão, reconexão entre outros. Uma vez desenvolvidas as funções de comunicação MQTT utilizadas no aplicativo desenvolvido, foi desenvolvida a interface de apresentação de informações ao usuário, processo apresentado na seção 3.4.4.

### *3.4.4 Interface de apresentação de informações ao usuário*

No aplicativo desenvolvido, a apresentação de informações ao usuário foi dividida de acordo com os grupos de registradores existentes (entradas digitais, saídas digitais, entradas analógicas, saídas analógicas e memórias). Esta divisão foi realizada para fornecer uma maior intuitividade na visualização de informações, uma vez que este terá conhecimento dos grupos de registradores que estão sendo observados no momento, algo que não aconteceria caso a apresentação de informações de *tags* de grupos de registradores diferentes fossem todas juntas, fazendo com que o usuário possuísse uma experiência menos agradável no uso do aplicativo.

A apresentação de informações ao usuário foi desenvolvida no aplicativo criado com o intuito de prover a este uma informação quantitativa e qualitativa a respeito registradores em questão. Para que esse objetivo pudesse ser alcançado, foram idealizados serviços que deveriam ser entregues ao usuário na apresentação de informações dos registradores.

#### *3.4.4.1 Serviços de apresentação de informações ao usuário*

Os serviços de apresentação de informações ao usuário fornecem de diferentes maneiras informações ao usuário com o intuito de prover a este um melhor entendimento do que acontece no processo monitorado como um todo. Os serviços idealizados e entregues ao usuário tratam-se de: apresentação de valor instantâneo de valor de variável, apresentação de comportamento da variável em gráficos em tempo real, possibilidade de customização de alarmes, apresentação de histórico entre outros. Os serviços são apresentados e descritos nas seções de 3.4.4.1.1 a 3.4.4.1.6.

##### *3.4.4.1.1 Apresentação de valor atualizado de registradores*

A apresentação de valores atualizados de registradores lidos do CLP é realizada mediante a apresentação do último valor recebido da *tag* correspondente, recebido

através de envio por parte do *broker* MQTT utilizado (o *broker* EMQX). O valor atualizado da *tag* foi apresentado ao lado do nome de sua variável.

Foi escolhida a representação numérica decimal para apresentar os valores contidos nos registradores. Ou seja, a representação em 1 em caso de verdadeiro lógico ou 0 em caso de falso lógico para os registradores que armazenam bits. Para a representação de variáveis tipo byte, os valores apresentados consistem em números apresentados de 0 a 255, que compreendem o *range* total da representação através de bytes. Por fim, a representação de variáveis do tipo word é apresentada em valores decimais, convertidos do formato hexadecimal para decimal no software desenvolvido executado no servidor local.

#### *3.4.4.1.2 Apresentação de gráficos de tempo real*

Foi idealizado e posteriormente disponibilizado ao usuário a apresentação do comportamento dos valores contidos nos registradores do CLP em tempo real mediante a apresentação de gráficos. Os gráficos gerados em tempo real no aplicativo utilizam os valores armazenados por cada *tag, lidos pelo CLP ciclicamente,* obtidos seguidamente durante toda a execução do aplicativo através das mensagens enviadas pelo *broker* MQTT aos dispositivos assinantes durante a comunicação.

Como apresentado na seção 3.3.7.1.1, as mensagens enviadas ao *broker* MQTT contém os valores armazenados nas *tags* utilizadas pertencentes a um determinado grupo de registradores (sejam eles entradas ou saídas, analógicas ou digitais, ou memórias) e junto a esses valores acompanha na mensagem uma estampa de tempo, criada após o recebimento da leitura realizada pelo CLP (requisitada pelo software executado no servidor via protocolo Modbus) dos valores que estão contidos nos registradores solicitados. Para criação da estampa de tempo utilizada foi utilizado o objeto *datetime* pertencente à biblioteca *Datetime.*

Da mesma forma que a mensagem é enviada ao *broker* MQTT esse envia para os dispositivos que assinaram o tópico pertencente à mensagem. Ou seja, os dispositivos recebem os valores que estão armazenados nos registradores desejados e a estampa de tempo criada no ato do recebimento dos valores lidos. Os valores lidos pelo CLP estão associados à estampa de tempo em questão, fazendo com que a

mensagem recebida contenha a informação de valor lido de cada *tag* e tempo de recebimento da leitura para cada dado pertencente ao grupo de registradores.

Para que possam ser apresentados gráficos de comportamento temporal das *tags*, os valores recebidos são armazenados separadamente, de acordo com a *tag* correspondente, em listas, de forma que cada lista contenha os seguidos valores recebidos correspondentes a uma determinada *tag*. Da mesma forma, a estampa de tempo criada é adicionada a uma lista responsável por armazenar as estampas de tempo do referido tipo de registrador ao qual a *tag* faz parte.

De posse dos valores das *tags* e das estampas de tempo foi utilizada a biblioteca Python *Matplotlib* para gerar os gráficos em tempo real. *Matplotlib* é uma biblioteca que fornece ferramentas para a criação de visualizações estáticas, dinâmicas e interativas em Python [35].

Para a criação de gráficos em tempo real de uma determinada *tag* são utilizadas listas que contém os valores recebidos pela *tag* e a lista contendo as estampas de tempo pertencentes ao tipo de registrador ao qual pertence a *tag* em questão. Os gráficos gerados são atualizados a cada 1 segundo. A plotagem dos gráficos em tempo real foi realizada pela função *plot* do módulo *pyplot* do *Matplotlib*.

Após o gráfico ter sido gerado na função *plot* do *Matplotlib*, o gráfico gerado precisa ser apresentado ao usuário (após o uso da função *plot* o gráfico gerado se encontra no *buffer* do *Matplotlib*). Para que o gráfico gerado possa ser visualizado pelo usuário dentro do *Kivymd* se faz necessário que ele esteja dentro de um widget suportado pelo *Kivy*.

Como solução para o problema encontrado foi utilizado um *widget* presente no *Kivy Garden* chamado *FigureCanvasKivyAgg. Kivy Garden* é um projeto que reúne extensões para o *Kivy* desenvolvidas e mantidas por usuários [36]. As extensões utilizadas no *Kivy* adicionadas ao *Kivy* Garden podem ser acessadas pelo Python através do uso da biblioteca *kivy.garden* em um código.

*FigureCanvasKivyAgg* é um *widget* no qual podem ser adicionados gráficos gerados pelo *Matplotlib* e estes serem tratados dentro do *Kivy* como um *widget* qualquer. Após a adição do *widget FigureCanvasKivyAgg* ao projeto e, inserir os gráficos gerados dentro do *FigureCanvasKivyAgg* em código, os gráficos gerados em tempo real puderam ser disponibilizados ao usuário dentro do aplicativo.

A apresentação de gráficos ao usuário é realizada mediante ação do usuário. Associada a ação de apresentação do comportamento de gráficos em tempo real de *tags* foi realizada a implementação do serviço de apresentação de histórico de *tags* apresentado na seção 3.4.4.1.5.

*3.4.4.1.3 Apresentação de alarmes ao usuário*

Foi idealizada no desenvolvimento do aplicativo a apresentação de alarmes ao usuário. A customização de um alarme foi implementada no aplicativo de forma que o usuário seja o responsável por associar ou não um alarme à *tag* em questão.

Caso o usuário desejasse associar um alarme à *tag* ele acrescentaria os valores máximos e mínimos aos quais ela não poderia ultrapassar. Caso a *tag* ficasse acima do valor máximo, acionaria um alarme que notifica o usuário que foi ultrapassado o valor limite superior da respectiva *tag*. Caso a *tag* ficasse abaixo do valor mínimo, seria acionado um alarme que notifica o usuário que a *tag* em questão recebeu um valor menor que o valor limite mínimo estipulado pelo usuário.

Os alarmes configurados pelo usuário são armazenados dentro de um arquivo de extensão JSON *(JavaScript Object Notation)* que armazena as configurações realizadas pelo usuário em cada *tag* (alarmes, nomes editados entre outras configurações), sendo o conteúdo deste arquivo acessado sempre durante a execução do aplicativo, impedindo que os alarmes configurados sejam perdidos após o aplicativo ser fechado. É criado para cada planta acessada um arquivo JSON próprio para que dados de uma planta não sobrescrevam informações de outra. *A* implementação de alarmes foi utilizada de forma que em uma situação de acionamento de alarme, a notificação de alarme fosse enviada por e-mail ao usuário no endereço de e-mail utilizado no cadastro do usuário. Para o envio de e-mails de alarme ao usuário são utilizadas as bibliotecas Python *smtplib* e *email.message* utilizadas também no envio de e-mails no aplicativo para recuperação de senha tanto de usuários do aplicativo quanto de plantas registradas no software atuando no servidor (seções 3.4.1 e 3.3.2, respectivamente).

*3.4.4.1.4 Customização de nomes de tags*

Os nomes das *tags* apresentadas no aplicativo são os mesmos nomes utilizados na tabela de *tags* desenvolvida no Tia Portal. Isso é conseguido por conta da leitura da tabela de *tags* no software que é executado no servidor local, que envia os nomes das *tags* ao *broker* MQTT (como citado em 3.3.4). Os aplicativos dos usuários recebem esses nomes através da assinatura do tópico de nomes das *tags* referentes à planta monitorada no momento.

Caso o nome utilizado na tabela de *tags* desenvolvida no Tia Portal seja pouco intuitivo ao usuário, este pode realizar uma edição no nome da *tag*. Este nome será salvo no arquivo JSON que contém as configurações realizadas pelo usuário para cada *tag.* Este arquivo conterá os nomes da *tag* editada pelo usuário e o nome original da *tag*. O nome original da *tag* é conferido à cada utilização da planta. Caso seja percebida uma alteração no nome original da *tag*, entende-se que a tabela de *tags* mudou e as edições são desabilitadas de forma a apresentar as alterações na tabela de *tags* que foi realizada.

*3.4.4.1.5 Apresentação do histórico de tags*

O serviço de apresentação de histórico de *tags* foi idealizado com o objetivo de permitir ao usuário visualizar comportamentos anteriores das *tags* em questão, viabilizando análises posteriores de fatos ocorridos durante o funcionamento da planta. O serviço de apresentação de histórico conta com informações como alarmes associados à *tag*, última notificação de alarme ocasionada pela *tag* (caso tenha sido habilitado o alarme à *tag*) e apresentação de gráficos de registros salvos pelo usuário.

Dentro do sistema de apresentação de gráficos em tempo real (apresentado na seção 3.4.4.1.2) foi desenvolvida uma interface para que o usuário do aplicativo possa salvar os dados que foram recebidos até o determinado momento para que posteriormente sejam vistos por ele na parte de históricos da *tag*. O sistema de armazenamento de registro foi desenvolvido de modo que todas os dados de todas as *tags* fossem salvas, sem necessitar que o usuário salve o registro *tag* por *tag*. Esse modo de armazenamento foi implementado para casos em que um acontecimento com uma determinada *tag* impacte em outras, sendo necessário o acesso à

informação de muitas para executar uma análise mais profunda. Em casos como o citado anteriormente, esta metodologia de armazenamento de informações simplifica as ações do usuário do aplicativo, permitindo a este uma melhor experiência.

Mediante a ação do usuário solicitando salvamento de dados, foi desenvolvida uma rotina que capta as listas que contém os valores de todas as *tags* e das estampas de tempo associadas a elas e armazena o conteúdo delas em um banco de dados local, que armazenará os valores do intervalo de tempo salvo no próprio aparelho.

O banco de dados utilizado foi o SQLite3, um banco de dados local presente na biblioteca padrão do Python. O SQLite3 permite a utilização de um banco de dados leve, que dispensa a utilização de um servidor separado e que possibilita a realização de operações no banco de dados mediante utilização de linguagem de consulta compatível com o SQL (*Structured Query Language*) [37].

Por ser um banco de dados relacional, o SQLite3 organiza os dados em tabelas sendo preenchidos linha por linha que são inseridas à tabela. Dentro do banco de dados local criado no aplicativo, cada *tag* de cada planta acessada pelo usuário possui uma tabela específica para armazenar os dados referentes a ela.

As linhas das tabelas criadas para cada *tag* monitorada contém 3 colunas: uma para armazenar as estampas de tempo, uma para armazenar os valores que as *tags* receberam e a última para armazenar o horário em que foi solicitado o registro. Para armazenar as listas que contém estampas de tempo e os seguidos valores lidos relacionados à *tag*, estes são transformados em *strings* para armazenamento no banco de dados de forma que cada linha da tabela contenha os valores de estampa de tempo, os valores que a respectiva *tag* recebeu e na última coluna o instante do salvamento.

Os valores salvos no registro compreendem os valores recebidos deste a última abertura do aplicativo até o momento em que usuário solicitou o registro. Posteriormente ao salvamento, este registro fica disponível ao usuário através de uma interface na área de histórico da *tag*.

### *3.4.5 Desenvolvimento de Arquivo apk*

Para que possa ser instalado dentro de um sistema operacional Android, um *script* Python deve ser transformado em um arquivo na extensão *apk*, extensão própria para dispositivos móveis que possuem o Android. A produção de um arquivo *apk* para o software desenvolvido permitiu a utilização da ferramenta desenvolvida sem necessidade de instalações adicionais no smartphone na qual ela foi instalada e também faz que a ferramenta seja disponibilizada e executada de maneira mais prática, sem a necessidade de o usuário ter um ambiente de desenvolvimento Python em seu smartphone. Esta pendência foi solucionada utilizando-se o Google Colaboratory, também chamado de Google Colab.

O *Google Colab* (também chamado de *Google Colaboratory*) é uma ferramenta desenvolvida pelo *Google Research* (nome dado à divisão de pesquisas científicas do Google) que permite o desenvolvimento e execução de *scripts* Python em um ambiente web através do navegador. A utilização do *Google Colab* é disponibilizada gratuitamente a qualquer usuário [38].

Dentro do *Google Colab* utilizou-se uma ferramenta utilizada no Python chamada Buildozer para gerar o arquivo na extensão *apk*. Buildozer é uma ferramenta que dedicada ao empacotamento (de *scripts* e de arquivos utilizados por ele) para a produção de executáveis direcionados à dispositivos móveis [39].

Como parâmetros de entrada no Buildozer estão o *script* Python desenvolvido para a aplicação, o arquivo Kivy de extensão *kv* (caso seja utilizado um) associado ao projeto e todas as imagens e ícones utilizados no aplicativo. Após o início do funcionamento do Buildozer, este fornece um arquivo *buildozer.spec* no qual devem ser descritas as configurações do aplicativo, configurações estas como nome do aplicativo, versão do aplicativo, bibliotecas Python envolvidas, se o aplicativo ao ser aberto deve estar na orientação retrato ou paisagem dentre outras configurações possíveis.

Após um certo tempo de processamento, que pode chegar até mais de 2 horas, a depender da internet local, o arquivo *apk* associado ao *script* é gerado e pode receber *download* do usuário. O arquivo gerado pode ser utilizado em sistemas operacionais Android.

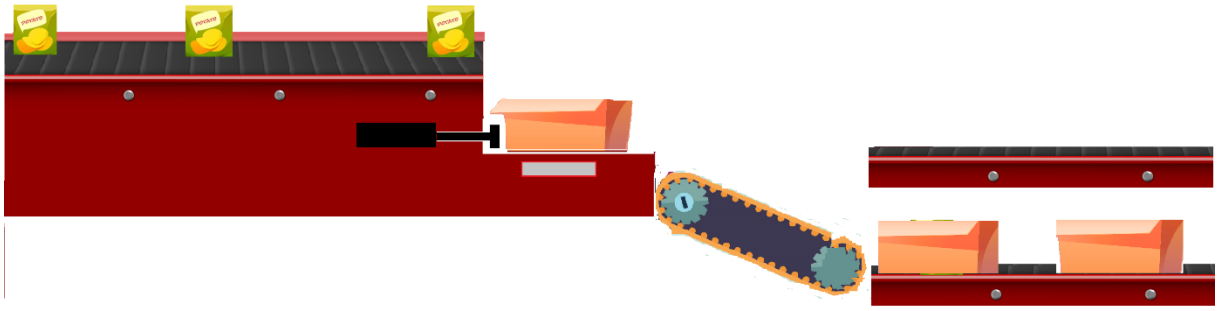
# 4 RESULTADOS OBTIDOS

Nesta seção são apresentados os resultados após o desenvolvimento do software executado no servidor local e desenvolvimento do aplicativo utilizado no smartphone para monitoramento, considerando a realização de um cadastro de um novo processo idealizado e, posterior monitoramento das variáveis lidas pelo CLP.

O processo idealizado consiste em uma simples planta de empacotamento, onde os produtos que chegam um a um por meio de uma esteira são embalados dentro de uma caixa, onde são depositadas após o final da esteira. Durante o processo de empacotamento, o conteúdo da caixa é pesado. Após alcançar o valor desejado, a caixa é empurrada por um atuador a uma plataforma, controlada por um motor de passo industrial, que direciona para qual das duas esteiras de saída a caixa irá a depender do tipo do lote selecionado.

Prevendo um possível problema no atuador que empurra as caixas, foi utilizada uma entrada digital para informar um possível problema com um atuador, indicando a necessidade de manutenção da plataforma. A imagem que descreve o processo é apresentada na Figura 22.

Figura 22 - Representação do processo utilizado para teste e obtenção de resultados.



Fonte: Próprio autor.

Após programação do CLP S7-1200 que controla o processo utilizando o Tia Portal, foi exportada a tabela de *tags* desenvolvida na programação, mediante

utilização do software citado. A tabela de *tags* do processo é apresentada na figura 23.

Figura 23 - Tabela de *tags* do processo utilizado para teste e obtenção de resultados.

| Name | Path | Data Type | Logical Address | Comment | Hmi Visible | Hmi Accessible | Hmi Writeable | Typeobject ID | Version ID |
|---|---|---|---|---|---|---|---|---|---|
| Processo Ligado | Default tag table | Bool | %I0.7 | | True | True | True | | |
| Esteira Início | Default tag table | Bool | %Q1.1 | | True | True | True | | |
| Peso Balança | Default tag table | Word | %IW64 | | True | True | True | | |
| Esteira Lote 1 | Default tag table | Bool | %Q1.0 | | True | True | True | | |
| Esteira Lote 2 | Default tag table | Bool | %Q1.2 | | True | True | True | | |
| Posição Atuador | Default tag table | Word | %QW80 | | True | True | True | | |
| Mtr Passo Saída | Default tag table | Byte | %QB0 | | True | True | True | | |
| Número Pacotes | Default tag table | Word | %MW3 | | True | True | True | | |
| Processo Ligado | Default tag table | Bool | %I1.1 | | True | True | True | | |
| Quant. Lotes | Default tag table | Word | %MW4 | | True | True | True | | |
| Lote | Default tag table | Bool | %I1.0 | | True | True | True | | |
| Defeito Atuador | Default tag table | Bool | %I0.1 | | True | True | True | | |

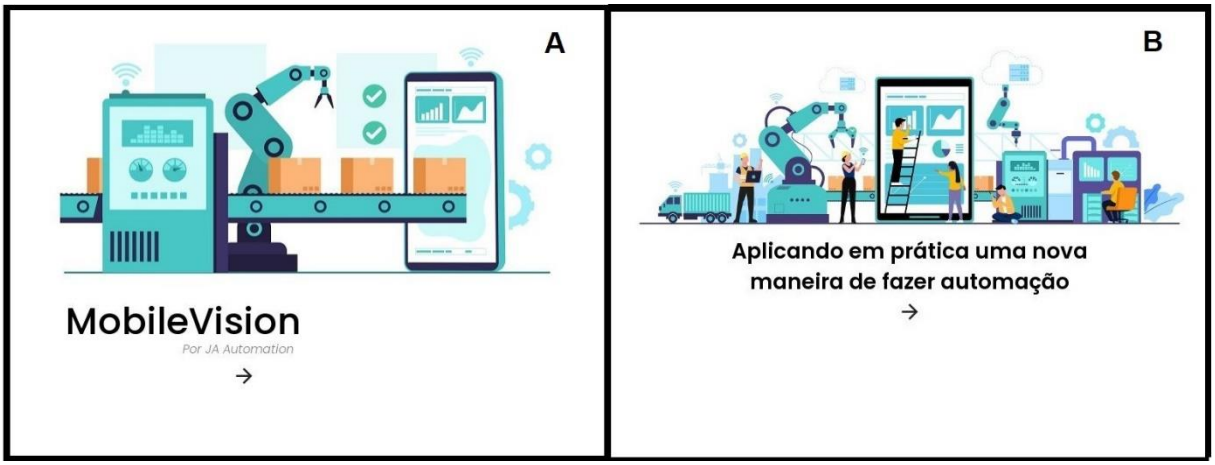
Fonte: Próprio autor.

## 4.1 Software desenvolvido para o servidor

As imagens contidas nesta seção apresentam os resultados obtidos mediante a experiência que um usuário tem durante a execução do software que é executado no servidor local. Entenda-se como usuário nesta seção, a pessoa que aplica este software no servidor local, que é diferente do usuário do aplicativo móvel que faz o monitoramento remoto da planta via smartphone.

Inicialmente, ao abrir o software o usuário é levado à tela inicial e, posteriormente, à sua tela introdutória, apresentadas na Figura 24 (partes A e B, respectivamente). Posteriormente, ao avançar, este é levado para a tela de login (Figura 25). Junto ao sistema de login foi implementado um sistema de cadastro para situações de primeiro acesso. Tal procedimento é iniciado ao clicar-se no botão de texto “Não tem planta cadastrada? Cadastre-a”. Onde é redirecionado para a tela de cadastro (Figura 26).

Figura 24 - Telas inicial (parte A) e introdutória (parte B) do software utilizado no servidor local.



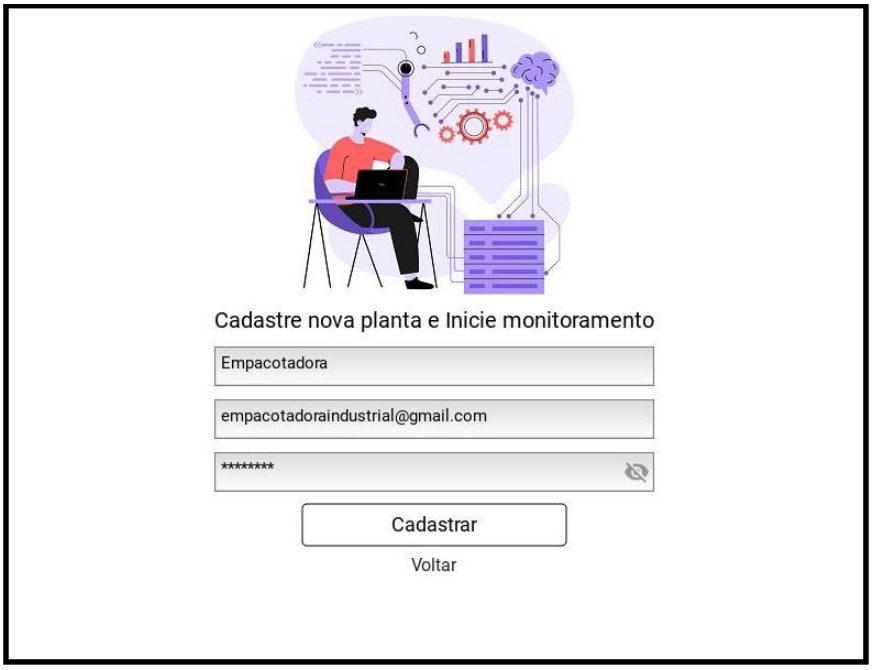

Fonte: Próprio autor.

Figura 25 - Tela de login do software utilizado no servidor local.


Fonte: Próprio autor.

Na tela de cadastro de planta, é solicitado um nome que será dado a ela, o e-mail que será associado a mesma e a senha para acesso. Caso o usuário forneça algum nome que já seja usado por alguém é solicitado que escolha outro nome. Caso o usuário acrescente um e-mail inválido também é apresentada uma mensagem de erro na tela.

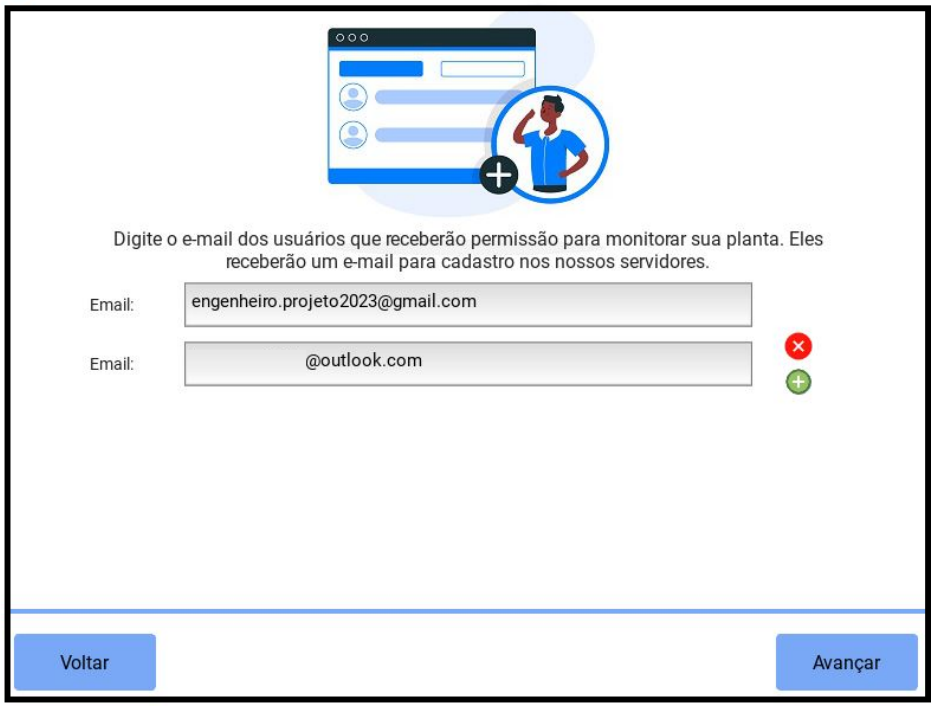
Figura 26 - Tela de cadastro do software utilizado no servidor local (representação de momento de cadastro).




Fonte: Próprio autor.

Após o usuário passar os parâmetros corretamente, é solicitado que ele adicione os endereços de e-mail das pessoas que terão acesso à planta (Figura 27). Em caso de possíveis erros na entrada de endereços de e-mail, este é informado que foi passado um endereço de e-mail inválido. Podem ser inseridos quantas pessoas for necessário, bastando adicionar uma a uma clicando no ícone “+” ou retirar clicando no ícone “x”. Após a inserção dos endereços de e-mail corretamente, o usuário é levado a confirmar suas escolhas e a planta é criada e este já pode fazer login no sistema.

Figura 27 – Adição de usuários com permissão para monitoramento.




Fonte: Próprio autor.

As pessoas que recebem permissão para monitoramento do respectivo processo recebem e-mails (Figura 28) contendo um link de uma página web desenvolvida no trabalho para cadastro no sistema e, consequentemente, para terem acesso a monitoramento das *tags* dela pelo aplicativo desenvolvido (Figura 29).

Figura 28 - Email enviado para pessoa que recebeu permissão de monitoramento da planta criada.


Fonte: Próprio autor.

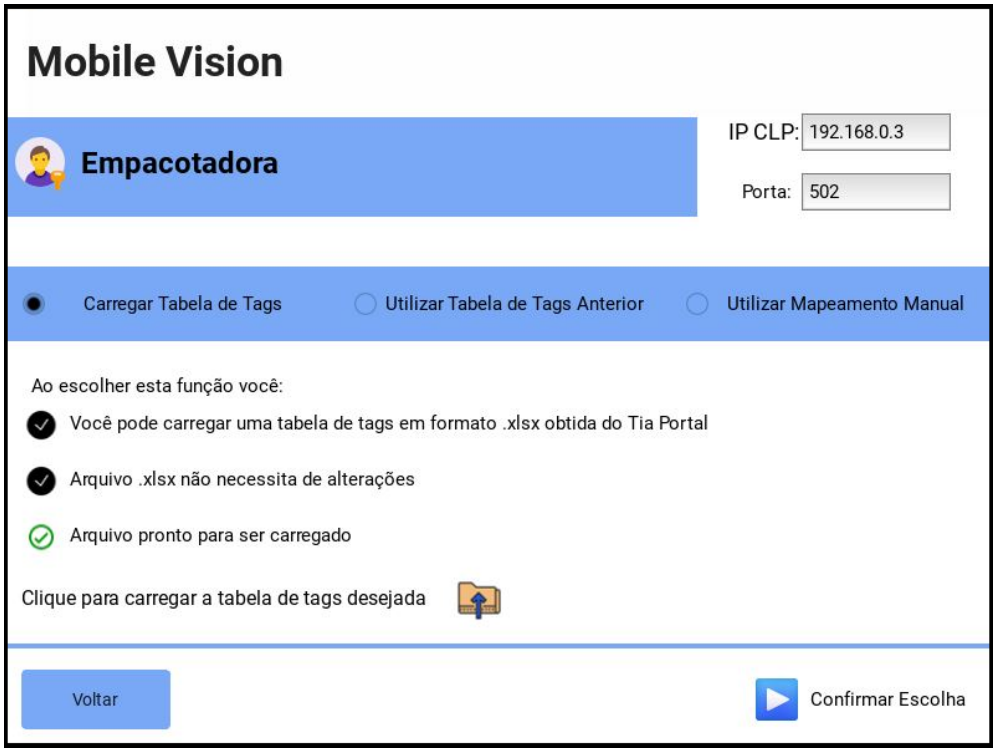
Figura 29 - Página web desenvolvida para cadastro de pessoas que receberam permissão para monitoramento via plataformas mobile.


Fonte: Próprio autor.

Como dito anteriormente, após a finalização do cadastro, o usuário pode realizar o login. Após o login, este é direcionado à página de configurações (Figura 30). Nesta tela, o usuário configura o IP do CLP e a porta por onde será realizada a comunicação nas caixas de entrada localizadas na parte superior direita da tela. Também nela, ele escolhe entre as possíveis opções de leitura de dados: carregamento de tabela de *tags* (para carregar uma nova tabela de *tags*), carregar última tabela de *tags* (no caso de já ter sido carregada uma tabela em algum momento) e *utilizar mapeamento manual*, para leitura de endereços sem o carregamento de uma tabela de tags, definindo manualmente os dados que serão requisitados. Como se trata de um primeiro uso e tendo uma tabela de tags do processo, foi escolhida a opção “carregar tabela de *tags*”.

Também na tela de configurações, o usuário pode alterar a lista de pessoas que possuem acesso à monitoramento de variáveis do processo. Ao clicar no ícone ao lado do nome da planta, o usuário é redirecionado para a tela de permissões, semelhante à tela apresentada na Figura 27. É permitido na utilização desta ação fornecer permissões ou retirá-las.

Figura 30 - Tela de configurações do software desenvolvido para o servidor local.

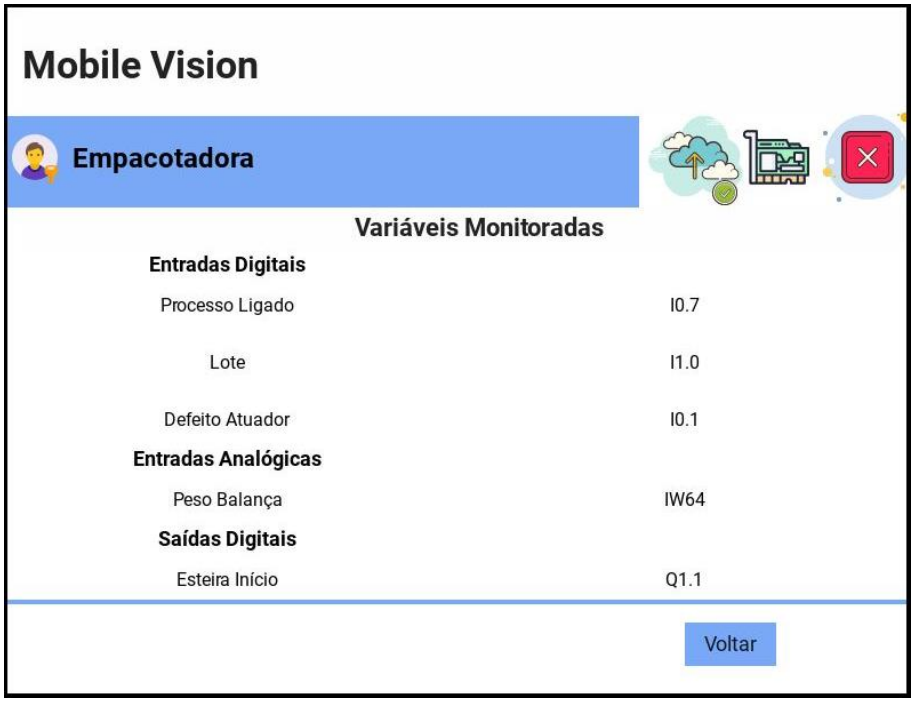



Fonte: Próprio autor.

Ao ser carregada uma tabela de *tags* dentro do formato especificado (formato .xlsx), os pontos que serão monitorados são extraídos dela e o usuário recebe a permissão para iniciar leitura mediante aparecimento do botão “confirmar escolha” na parte inferior da tela. Ao confirmar as suas escolhas, ele é redirecionado para a tela de operação.

Na tela de operação (Figura 31) são apresentadas ao usuário todas as *tags* (e seus respectivos endereços) que estão presentes na tabela de *tags* e que estão disponíveis para monitoramento via smartphone. Estas são apresentadas de acordo com o grupo de registradores ao qual fazem parte. Se a tabela carregada for muito grande, o usuário pode acessar as *tags* que não apareceram mediante o uso do *scroll* do *mouse*.

Figura 31 - Tela de Operação do software desenvolvido para o servidor local.




Fonte: Próprio autor.

Na parte superior direita da tela são informadas as situações atuais da comunicação MQTT e Modbus TCP (ícone mais à esquerda e ícone do centro, respectivamente). Quando a comunicação MQTT está aberta e conseguindo transmitir dados, o ícone correspondente a ela (uma nuvem com seta para cima) tem um sub-ícone de um *check* verde abaixo dele. Caso contrário o ícone da conexão fica com um sub-ícone de um *x* vermelho. Quando a comunicação Modbus TCP está aberta e o software está conseguindo receber dados é apresentado um CLP verde. Caso contrário é apresentado um CLP vermelho. Na Figura 32 é apresentada uma situação onde não há comunicações MQTT e Modbus TCP ao mesmo tempo.

Figura 32 – Tela de operação informa ao usuário situação onde não há conexão MQTT nem Modbus TCP.


Fonte: Próprio autor.

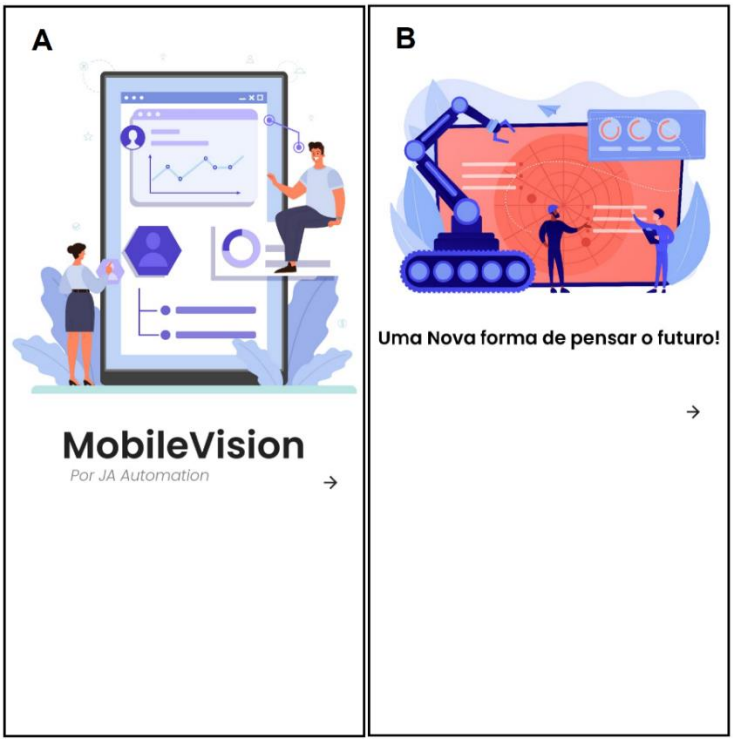
Também na tela de operação são disponibilizados botões de navegação. O botão de fechar aplicativo fica na parte superior direita ao lado do botão de *status* da comunicação Modbus TCP enquanto o botão de voltar para a tela de configuração fica na parte inferior direita da tela. Ao utilizar este botão, o abre-se uma pop-up pedindo a confirmação da ação pois voltar para a tela de configurações fecha a comunicação atual, fazendo o software aguardar novos ajustes por parte do usuário.

## 4.2 Aplicativo Desenvolvido Para smartphone

Os resultados obtidos nesta seção apresentam o procedimento que o usuário experiencia durante a realização do monitoramento do processo industrial descrito na seção 4 utilizando o aplicativo criado. Entenda-se como usuário nesta área, a pessoa que utiliza o aplicativo para monitoramento da planta utilizando seu smartphone.

Ao abrir o aplicativo o usuário é levado à sua tela inicial e depois a uma tela introdutória (partes A e B na Figura 33, respectivamente). Posteriormente, ao passar dessas telas, este é levado à tela de login do aplicativo.

Figura 33 - Tela inicial do aplicativo (parte A) e tela introdutória (parte B).



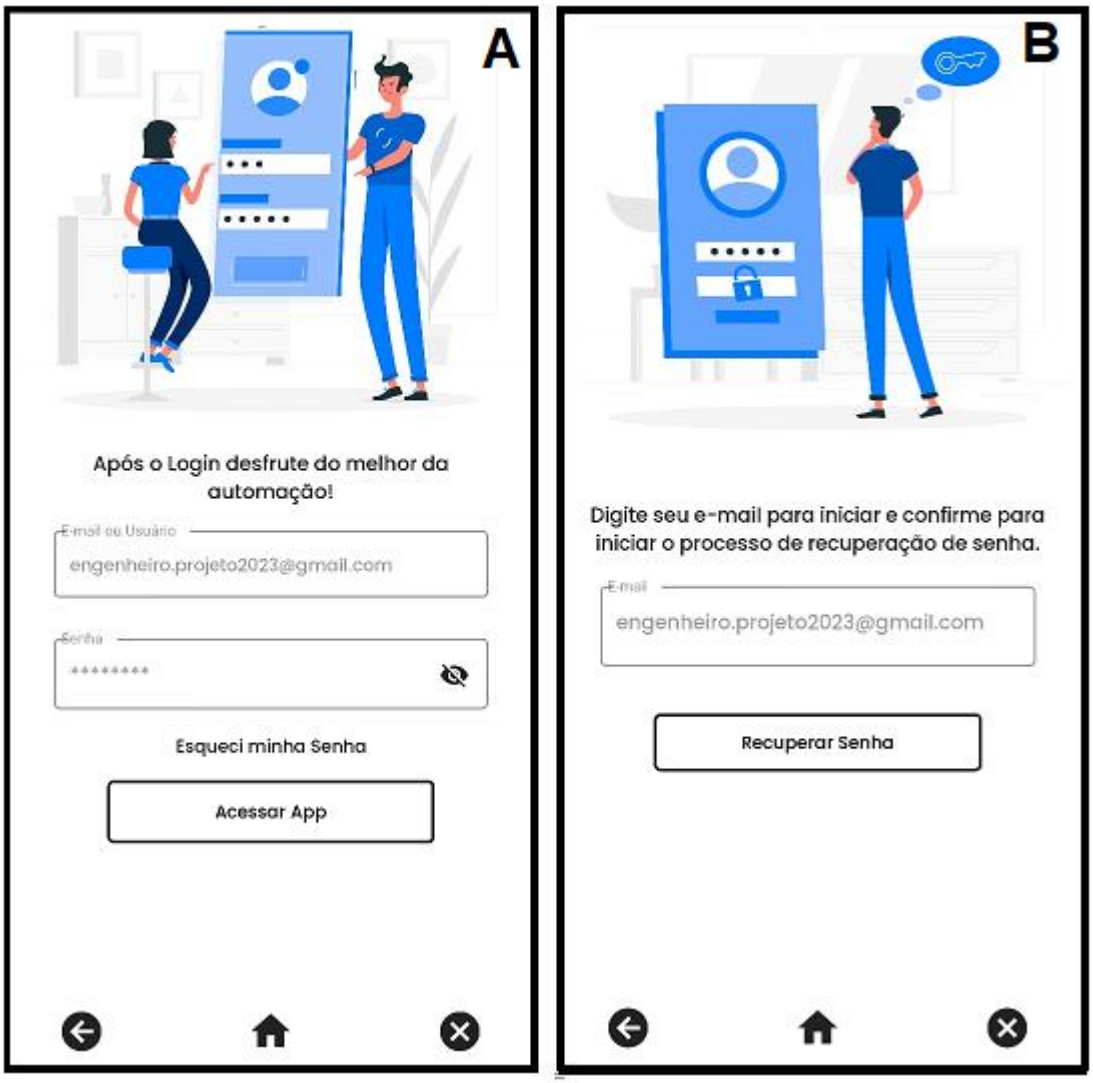

Fonte: Próprio autor.

O sistema de login solicita do usuário seu e-mail e senha cadastrados. Em caso de erro em algum dos dados fornecidos, o sistema apresenta uma mensagem de e-mail ou senha inválidos. A parte A da Figura 34 apresenta a tela de login no momento em que uma tentativa de acesso está sendo realizada. Com os parâmetros de login corretos, o usuário é redirecionado para a tela de escolha de plantas para monitoramento.

Para o caso de o usuário ter esquecido sua senha de login, foi desenvolvido no aplicativo um sistema de recuperação de senha que, solicita ao usuário seu endereço de e-mail cadastrado. Caso o endereço fornecido esteja correto, é enviado um e-mail para recuperação de senha de acesso ao aplicativo. Caso contrário, é informado que o endereço de e-mail informado não está contido entre os usuários cadastrados para uso do aplicativo.

A parte B da Figura 34 apresenta a tela de recuperação de senha do aplicativo no momento de início de um processo de recuperação de senha, onde usuário insere seu e-mail de cadastro. A página de recuperação de senha no aplicativo pode ser acessada mediante escolha da opção *Esqueci minha* senha na tela de login.

Figura 34 - Tela de login (parte A) e tela de recuperação de senha (parte B) do aplicativo desenvolvido.



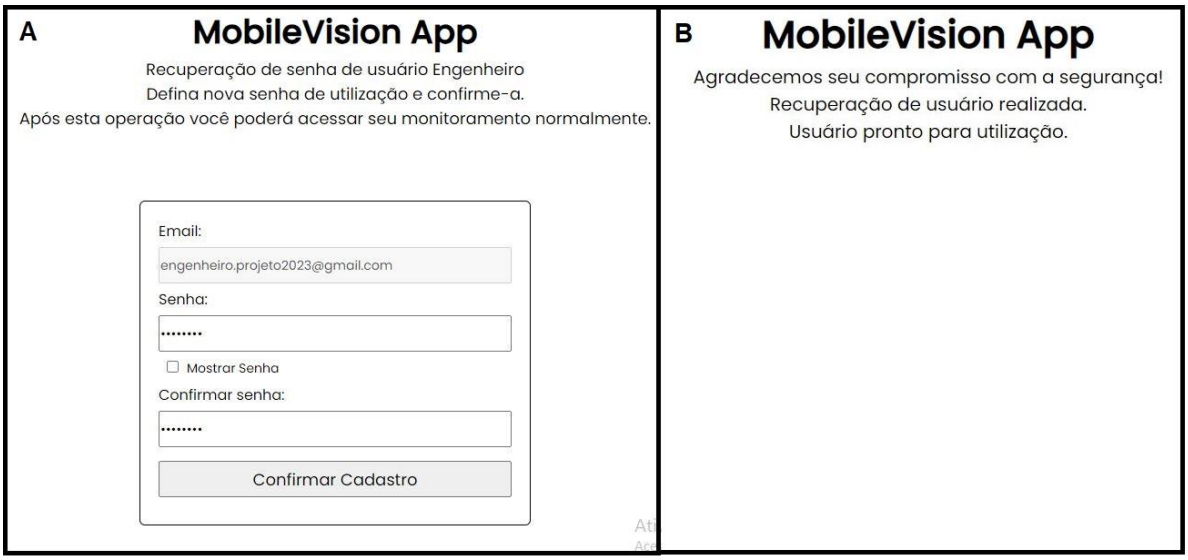

Fonte: Próprio autor.

A Figura 35 apresenta o e-mail enviado para o usuário que solicitou recuperação de senha. O e-mail contém um link que o redireciona para uma página web criada para receber a nova senha por parte do usuário. A Figura 36 apresenta a página web desenvolvida para a recuperação de senha (identificada por A na Figura) e a mesma página web informando a tentativa bem sucedida de alteração de senha após finalização do processo (identificada por B na Figura).

Figura 35 - E-mail de recuperação de senha do usuário para acesso ao aplicativo desenvolvido.


Fonte: Próprio autor.

Figura 36 - Página web para recuperação de senha nos momentos de inserção de nova senha (parte A), recuperação de senha concluída (parte B).

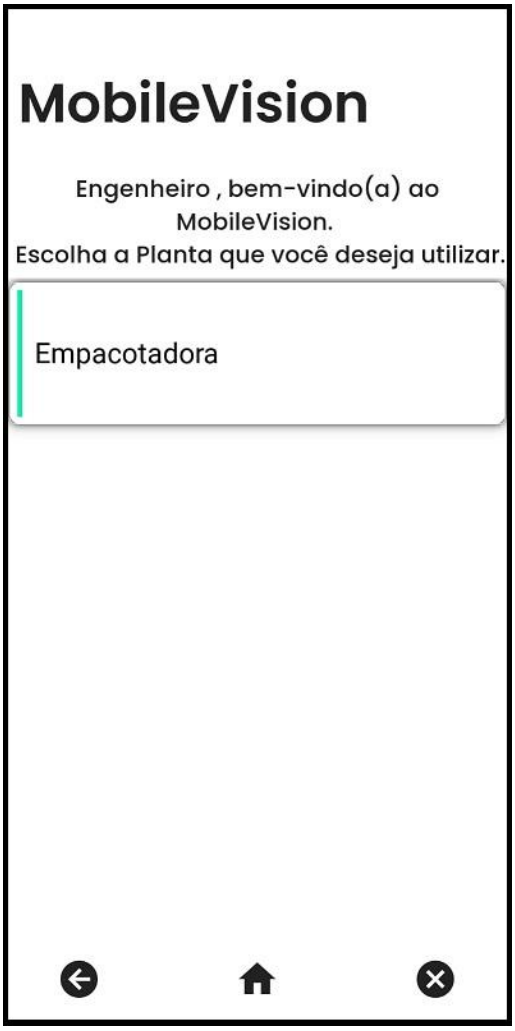



Fonte: Próprio autor.

Após o login ter sido realizado por parte do usuário de maneira bem-sucedida, este é conhecido e é redirecionado para a tela onde este vai realizar a escolha de qual planta ele deseja monitorar (Figura 37). As plantas que aparecem disponíveis são apenas as plantas nas quais ele recebeu permissão para monitoramento. Como o este

só tinha permissão no processo Empacotadora, ela foi o único processo que apareceu.

Figura 37 - Tela de escolha de planta para monitoramento no aplicativo desenvolvido.




Fonte: Próprio autor.

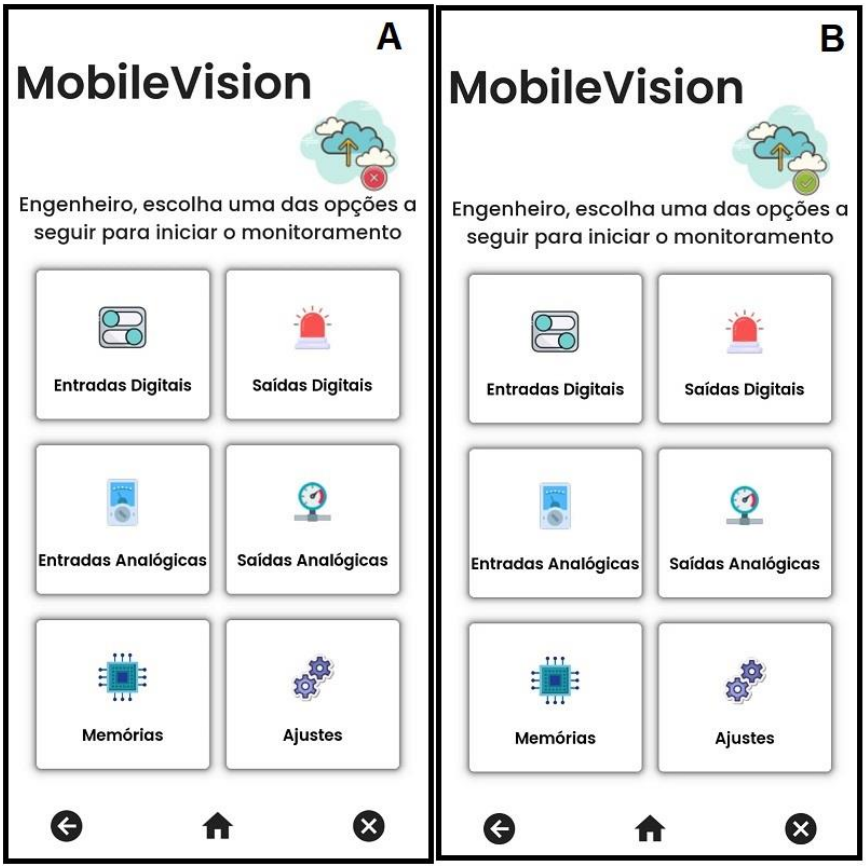
Após escolher a planta que deseja monitorar, o usuário é direcionado para uma tela de *loading*. O *loading* é utilizado para permitir que seja estabelecida uma conexão com o *broker* MQTT, sejam assinados os tópicos MQTT necessários para recebimento de informações do processo escolhido e para que sejam carregados no arquivo os nomes das *tags* monitoradas nele, recebidas através de envio do *broker* MQTT após assinatura de tópicos. Apesar da rápida passagem pela referida tela (em torno de 2 segundos em caso de boa conexão de internet), é disponibilizado ao usuário um

feedback do progresso de carregamento através do avanço de uma barra de progresso na tela.

O aplicativo desenvolvido possui uma tela de menu, onde o usuário escolhe o grupo de registradores que deseja monitorar no momento. Também nesta tela há um ícone na parte superior, indicando se há ou não uma conexão estabelecida com o *broker* MQTT da EMQ, o *broker* EMQX, podendo indicar ao usuário um possível problema de comunicação. O ícone é semelhante ao comentado na seção 4.1 onde há um sub-ícone de um *x* vermelho para o caso de conexão fechada e um sub-ícone verde com *check* para o caso de conexão aberta.

A Figura 38 apresenta a tela de menu nas duas situações, sem conexão aberta entre o smartphone e o *broker* MQTT (identificada por A na Figura) e com conexão aberta entre os dois (identificada por B na Figura). O usuário é livre para voltar à esta tela quantas vezes desejar durante a execução do aplicativo.

Figura 38 - Tela de menu do aplicativo desenvolvido sem conexão com o *broker* MQTT (parte A) e com conexão com o *broker* MQTT (parte B).




Fonte: Próprio autor.

A navegação entre as telas é provida pelos três botões contidos na parte inferior da tela: voltar à tela anterior (botão com ícone de seta circular à esquerda), voltar a tela inicial do aplicativo (botão com ícone *home*) e fechar o aplicativo (botão de ícone de placa com letra x), da esquerda para a direita. Estes três botões estão contidos em todas as telas a partir da tela de login para propiciar uma navegação com maior facilidade e intuitividade ao usuário.

As telas de apresentação de informações das *tags* contém as *tags* que são monitoradas no tipo de registrador escolhido no menu. Cada variável ocupa o espaço de uma linha na tela e é definida pelo seu nome (contido na tabela de *tags* e enviado ao aplicativo pelo *broker* MQTT, que recebeu essas informações do software contido no servidor local via protocolo MQTT). A Figura 39 contém a tela de apresentação de saídas digitais do processo em questão.

Figura 39 - Tela de apresentação de saídas digitais do processo Empacotadora.


Fonte: Próprio autor.

Ao lado dos nomes de cada *tag* estão os serviços disponíveis ao usuário envolvendo cada uma. Na parte de edições da *tag* (aberta mediante um toque no ícone de edições, representado por um lápis), é possível editar o nome da *tag*, definir um alarme para ela e acrescentar uma unidade de medida a mesma. A Figura 40 apresenta o processo de edição bem-sucedida do nome de *Mtr Passo Saída* onde na parte à esquerda da Figura (indicada por A) se apresenta o nome antigo da variável, na parte central da Figura (indicada por B) a edição é realizada e na parte mais à direita da Figura (indicada por C) apresenta-se processo de edição do nome concluído onde passou a se chamar *MP Plataforma*.

Figura 40 - Processo de mudança de nome da *tag* Mtr Passo Saída no aplicativo.


Fonte: Próprio autor.

Para se habilitar um alarme, deve-se clicar na *checkbox* e definir os limites superior e/ou inferior do alarme e, posteriormente, confirmar as alterações. Os alarmes são disparados quando os valores obtidos são maiores ou iguais (para os limites superiores) ou menores ou iguais (para limites inferiores) aos valores inseridos nos respectivos espaços. Os alarmes estão disponíveis para as *tags* de todos os tipos de registradores.

Para o caso da variável *Defeito Atuador* foi habilitado um alarme para o momento em que ela possuir o nível lógico alto, 1 em decimal (Figura 41). Enquanto o limite inferior foi deixado vazio por não ser necessário na utilização da variável em questão.

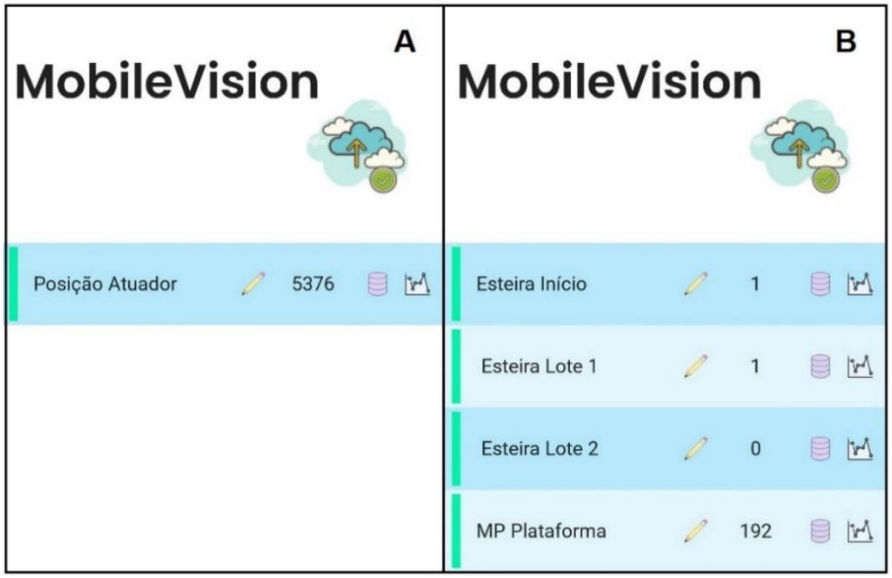
Figura 41 - Configuração de alarme utilizada para variável Defeito Atuador.


Fonte: Próprio Autor

Ao lado direito do ícone edições, é apresentado ao usuário o último valor recebido pela *tag*. Como apresentado na seção 3.4.4.1.1, os valores das *tags* são apresentados na sua forma decimal. Sendo assim, bits são apresentados em zeros e uns, bytes como números inteiros entre 0 e 255 e words em números inteiros entre 0 32767 (falando de maneira geral, pois entradas e saídas analógicas tem seu maior valor em 27648).

Na figura A são apresentadas a representação para os 3 tipos de dados (bit, byte e word). A parte A da Figura 42 apresenta o valor da saída analógica *Posição Atuador* do tipo word enquanto a parte B da Figura 42 apresenta as saídas digitais, onde o byte de saída MP Plataforma se faz presente além de outras *tags* de tipo bit, todas representadas em suas formas decimais.

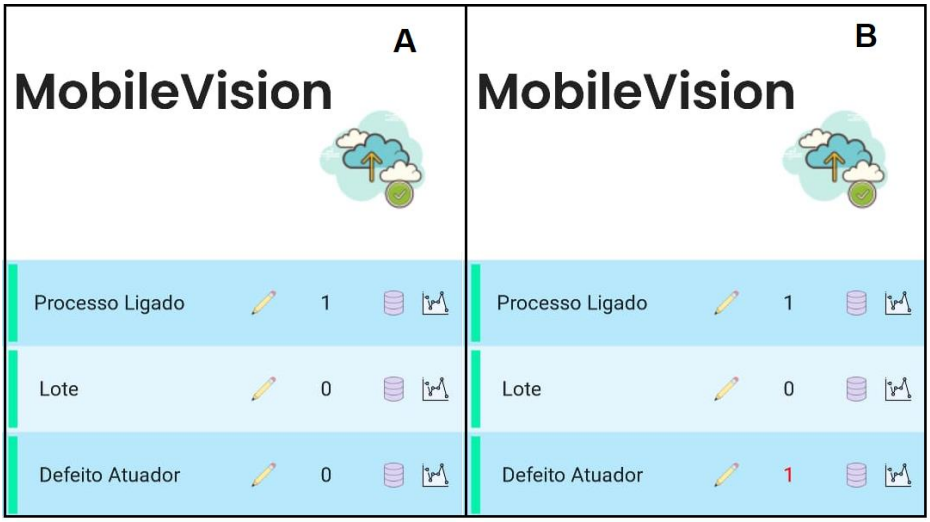
Figura 42 - Apresentação dos valores das tags *Saída Atuador* (parte A) e saídas digitais (parte B) em suas formas decimais de representação.




Fonte: Próprio autor.

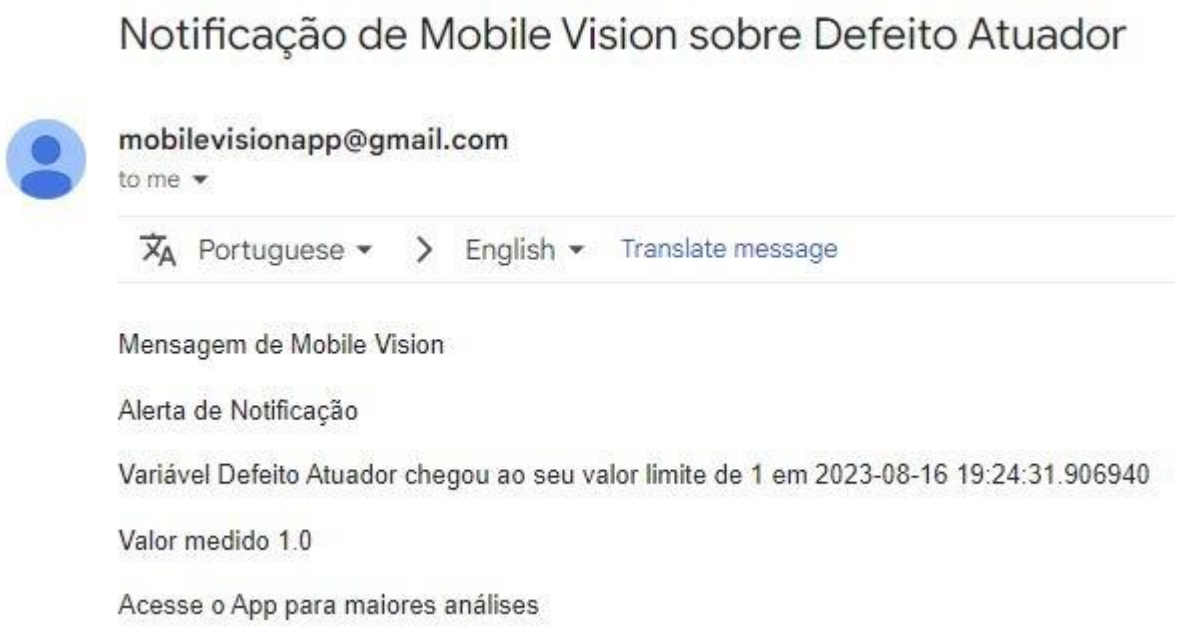
No caso de uma *tag* ter atingido um nível de alarme (acima do limite máximo ou abaixo do limite mínimo) a indicação do valor desta passa da cor preta para a cor vermelha. Na Figura 43 é apresentada a variável *Defeito Atuador* dentro da faixa desejada e, posteriormente a apresentada a mesma *tag* após ter alcançado um nível de alarme. Neste momento, também é enviado ao usuário um e-mail informando a *tag* que gerou o alarme, o valor que ela recebeu e o momento em que foi gerado o alarme. O e-mail é enviado ao endereço de e-mail em que o usuário está cadastrado para o uso do aplicativo. Este é apresentado na Figura 44.

Figura 43 - Apresentação do valor da *tag Defeito Atuador* fora do nível de alarme (parte A) e em nível de alarme (parte B).




Fonte: Próprio autor.

Figura 44 - E-mail enviado indicando alarme na variável *Defeito Atuador.*




Fonte: Próprio autor.

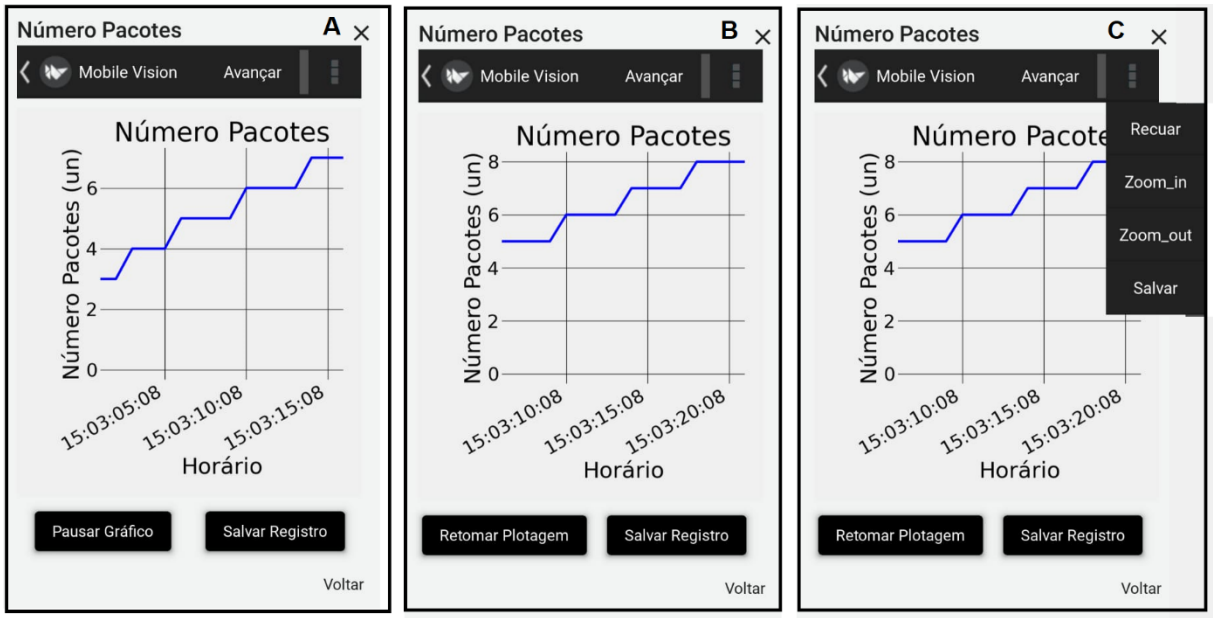
O último ícone à direita (representado por um gráfico) representa a apresentação variável na forma de um de um gráfico, atualizado em intervalos de tempo de 1 segundo, dos valores recebidos pela *tag*. O gráfico apresenta o valor recebido pela *tag* temporalmente. É disponibilizado ao usuário as funções de pausar o gráfico,

avançar e recuar o gráfico, dar zoom-in ou zoom-out no gráfico, funcionalidades estas providas através de botões.

Também é permitido ao usuário salvar registros (funcionalidade provida pelo botão salvar registro) para que sejam acessados posteriormente na aba de históricos. Ao apertar um botão *salvar registro* de uma *tag*, são salvos os registros de todas as *tags* como uma forma de salvar o momento do processo inteiro. É permitido o armazenamento simultâneo de 3 salvamentos de registros, sendo a partir do quarto salvamento de registros, apagado o mais antigo com o intuito de não sobrecarregar o armazenamento do smartphone do usuário.

A funcionalidade de apresentação de gráficos em tempo real é mostrada da Figura 45. Na referida figura são apresentados gráficos em diferentes momentos. Na parte A da Figura é apresentada o gráfico em tempo real com as opções de pausa e salvar registro. Na parte B da Figura é apresentada uma utilização da função pausa, sendo disponibilizada ao usuário a função *retomar plotagem* para a visualização novamente em tempo real. E, por fim, a parte C da Figura apresenta todas as outras funções citadas disponibilizadas ao usuário. (A situação em que foi obtida o gráfico A não foi a mesma em que foram obtidos os gráficos B e C).

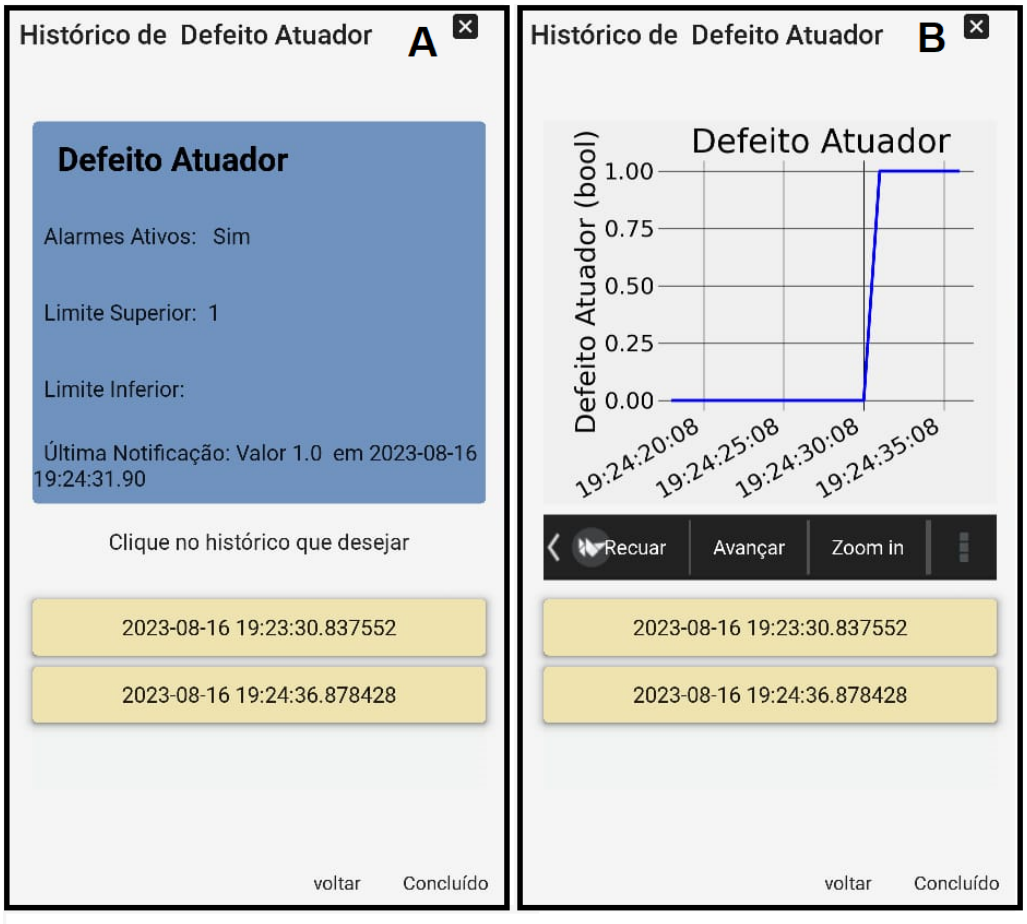
Figura 45 - Apresentação de gráficos em tempo real (parte A). Utilização da função Pausa (parte B). Outras funções disponíveis na apresentação de gráficos em tempo real (parte C).




Fonte: Próprio autor.

O ícone à esquerda do ícone do gráfico (representado por um símbolo de banco de dados) representa o serviço de histórico da *tag*, apresentado na Figura 46. Nesta aba de histórico estão contidas várias informações: se foi habilitado um alarme para a *tag* ou não, caso tenha sido habilitado quais foram os limites superior e inferior que foram aplicados, qual foi o último alarme gerado pela *tag* (se já foi gerado algum alarme por ela), apresentando valor e momento em que foi disparado o alarme e, por último, os registros salvos pelo usuário. A parte A da Figura 46 apresenta a aba de históricos referente à variável *Defeito Atuador* que teve um alarme habilitado para ela anteriormente.

Figura 46 – Informações de Histórico da *tag* no aplicativo (parte A). Gráfico de histórico da *tag* (parte B).




Fonte: Próprio autor.

Os registros salvos pelo usuário são apresentados em forma de botões. Os textos contidos nesses botões correspondem ao momento do salvamento do registro por parte do usuário. Este texto faz com que o usuário saiba qual registro deve escolher mediante sua necessidade. Ao pressionar um botão correspondente a um terminado registro, é gerado um gráfico correspondente ao intervalo de tempo salvo. O usuário pode interagir com este gráfico realizando zoom-in, zoom-out, avanço e recuo. Na parte B da Figura 46 é apresentado um gráfico de histórico gerado.

# 5 CONCLUSÕES E PROPOSTAS DE CONTINUIDADE

Nesta seção é realizada a finalização deste trabalho. Na seção 5.1 são apresentadas as conclusões obtidas deste trabalho enquanto na seção 5.2 são apresentadas as propostas futuras envolvendo a continuidade do trabalho aqui desenvolvido.

## 5.1 Conclusões

Mediante o desenvolvimento do presente trabalho, foi possível configurar o controlador lógico programável S7-1200 para enviar dados a um servidor local utilizando protocolo MODBUS. Esta configuração apresentou resultados satisfatórios permitindo o envio de dados de processo contidos no CLP a um servidor local conectado a ele quando lhe era solicitado.

Também foi produzido um software que atuou no respectivo servidor local para captura de dados do CLP e posterior envio a um dispositivo móvel que é utilizado de maneira remota. O software produzido agiu de maneira esperada, cumprindo o papel idealizado a ele no início do trabalho, sendo responsável por obter os dados do CLP e enviá-los ao dispositivo móvel.

Como explicitado em seções anteriores, o referido envio de dados do servidor local para dispositivos móveis foi realizado de maneira indireta mediante uso de protocolo MQTT, utilizando um *broker*. Este se comportou como esperado recebendo os respectivos dados do aplicativo desenvolvido no servidor local e repassando-os aos aparelhos móveis durante a realização de testes.

Foi produzido um aplicativo móvel que era responsável por receber os dados de processo, contidos no CLP S7-1200, encaminhados ao aplicativo pelo *broker* MQTT. A aplicação desenvolvida conteve nela a apresentação desses dados ao usuário de maneira quantitativa e qualitativa mediante apresentação de valores atualizados e apresentação por meio de gráficos, fornecendo ao usuário um alto grau de customização, com a possibilidade de configuração de alarmes, mudança de nomes e inserção de unidades de medida às variáveis de processo.

A referida aplicação também proveu ao usuário a funcionalidade de históricos de dados, mediante a possibilidade de salvar registros enquanto se monitora dados do processo em tempo real. Esta funcionalidade fornece a oportunidade de análise de dados em um momento qualquer desejado pelo usuário, fornecendo a este a possibilidade de uma posterior análise de momentos do processo.

Com a produção do software responsável para atuar no servidor local e do aplicativo para monitoramento remoto foram desenvolvidos sistemas de cadastro, login e recuperação de senha. No referido sistema foi produzido cadastramento de plantas onde em cada processo cadastrado fosse realizado gerenciamento de permissões, possibilitando controlar as pessoas que teriam acesso à dados da referida planta via celular, conferindo assim privacidade ao compartilhamento de dados de processo.

A utilização do sistema de cadastro e login também possibilitou a utilização desta metodologia de monitoramento em múltiplos processos ao mesmo tempo, abrindo espaço para utilização desta aplicação em âmbito comercial mediante apoio na utilização de linguagens de programação *open source* e protocolos de comunicação abertos utilizados neste trabalho.

Mediante todos os resultados obtidos ao final deste trabalho, citados anteriormente, conclui-se que a utilização de um aplicativo para monitoramento de processos industriais envolvendo o CLP S7-1200 utilizando dispositivos móveis é possível e viável. Também se conclui que a metodologia utilizada para a realização deste monitoramento apresenta resultados satisfatórios e apresenta a vantagem não solicitar de hardwares adicionais ao chão de fábrica para ser implementado.

## 5.2 Propostas de Continuidade

Mediante o desenvolvimento deste trabalho, foram percebidas novas possibilidades de continuidade envolvendo o conteúdo que foi desenvolvido no trabalho. Entre elas estão:

- A implementação de um sistema de *Firewall* entre o servidor local e a rede externa para elevar o grau de segurança no compartilhamento de dados do processo;

- A introdução de um administrador para o sistema responsável por aceitar ou recusar cadastros de usuários.
- Disponibilizar aos usuários a normalização e escalonamento de entradas e saídas analógicas.

# REFERÊNCIAS


1. **What is the Internet of Things?** Disponível em: <https://www.ibm.com/topics/internet-of-things>. Acesso em: 16 ago. 2023.
2. **What is Internet of Things (IoT)?** Disponível em: <https://www.oracle.com/internet-of-things/what-is-iot/>. Acesso em 16 ago. 2023.
3. CORPORATIVA, I. **What is IIoT? Discover the Industrial Internet Of Things.** 2023. Disponível em: <https://www.iberdrola.com/innovation/what-is-iiot>.Acesso em: 16 ago. 2023.
4. **What is Industry 4.0 and how does it work?** Disponível em: <https://www.ibm.com/topics/industry-4-0>. Acesso em: 16 ago. 2023.
5. Petruzella, F. D. **Controladores Lógico Programáveis.** Tradução: Romeu Abdo. Porto Alegre: AMGH, 2014.
6. Franchi, C. M; Camargo, V. L. A. **Controladores Lógico Programáveis:** Sistemas Discretos. São Paulo: Érica, 2008.
7. Siemens AG. **S7-1200 Programmable controller:** System Manual, abr. 2012, p.864.
8. Pansanato, L. T. E. **Redes locais de computadores.** Curitiba: Ed. UTFPR, 2016.
9. Tanenbaum, A. S.; Wetherall, D. **Redes de Computadores.** Tradução: Daniel Vieira. São Paulo: Pearson Prentice Hall, 2011.
10. Cisco Systems, I. **Cisco Networking Academy Program:** CCNA 1 and 2 Companion Guide. Third Edition. 20 maio 2003.
11. Modbus Organization. **MODBUS Application Protocol Specification:** V1.1 b3, 26 abr. 2012, p.50.
12. Modbus Organization. **MODBUS Messaging on TCP/IP:** Implementation Guide V1.0b, 24 out. 2006, p.46.
13. FIELDING, R. T.; NOTTINGHAM, M.; RESCHKE, J. **RFC 9110: HTTP semantics**. Disponível em: <https://www.rfc-editor.org/rfc/rfc9110.html>. Acesso em: 14 ago. 2023.
14. **IBM Documentation**. Disponível em: <https://www.ibm.com/docs/pt-br/ibm-http server/9.0.5?topic=communications-secure-sockets-layer-ssl-protocol>. Acesso em: 16 ago. 2023.
15. **Conexiones TLS y SSL.** Disponível em: <https://support.google.com/a/answer/100181?sjid=656354051100191372-SA>. Acesso em 27 ago. 2023.
16. Banks, A.; Gupta, R. **MQTT Version 3.1.1:** OASIS Standard. 29 out. 2014.
17. **MQTT & MQTT 5 Essentials:** A comprehensive overview of MQTT facts and features for beginners and experts alike. HiveMQ: Alemanha, 2020.

18. Banks A., et al. **MQTT Version 5.0: OASIS Standard.** 7 mar. 2019.
19. **MQTT - the standard for IoT messaging**. Disponível em: <https://mqtt.org/>. Acesso em: 14 ago. 2023.
20. Paiva, F. A. P, et al. **Introdução a Python com aplicações de sistemas operacionais.** Natal: IFRN, 2019.
21. Lutz, M.; Asher, D. **Aprendendo Python.** Tradução: João Tortello. Porto Alegre: Bookman, 2007.
22. **Welcome to**. Disponível em: <https://www.python.org/about/>. Acesso em: 15 de ago. 2023.
23. **Applications for Python**. Disponível em: <https://www.python.org/about/apps/>. Acesso em: 15 ago. 2023.
24. Alves, W. P. **Banco de Dados.** São Paulo: Érica, 2013.
25. **Make your app the it can be with Firebase.** Disponível em: <https://https://firebase.google.com/?hl=pt>. Acesso em: 15 ago. 2023.
26. **Firebase Realtime Database**. Disponível em: <https://firebase.google.com/docs/database?hl=pt-br>. Acesso em: 15 ago. 2023.
27. **API REST do banco de dados do**. Disponível em: <https://firebase.google.com/docs/reference/rest/database?hl=pt-br>. Acesso em: 15 ago. 2023.
28. Virbel, M.; et al. **Plyer Documentation:** Release 2.0.0. 23 jul. 2022, p.27.
29. **Package overview – pandas 2.0.3 documentation.** Disponível em: <https://pandas.pydata.org/docs/getting_started/overview.html>. Acesso em: 5 ago. 2023.
30. Lefebvre, L. **PyModbusTCP Documentation:** release 0.2.0. 5 jun. 2022, p.42.
31. EMQ TECHNOLOGIES INC. **About us.** Disponível em: <https://www.emqx.com/en/about>. Acesso em: 6 ago. 2023.
32. **WEB E. Eclipse Paho.** Disponível em: <https://projects.eclipse.org/projects/iot.paho> Acesso em: 6 ago. 2023.
33. EMQ TECHNOLOGIES INC. **The top 1 free public MQTT broker**. Disponível em: <https://www.emqx.com/en/mqtt/public-mqtt5-broker>. Acesso em: 6 ago. 2023.
34. **Pyinstaller Manual – Pyinstaller 5.13.0 documentation.** Disponível em: <https://pyinstaller.org/en/stable/>. Acesso em: 9 ago. 2023.
35. **Matplotlib – visualization with Python.** Disponível em: <https://matplotlib.org/> Acesso em: 8 ago. 2023.
36. **Garden – 2.2.1 documentation.** Disponível em: <https://kivy.org/doc/stable/api-kivy.garden.html> Acesso em: 8 ago. 2023.
37. **12.6. sqlite3 – DB-API 2.0 interface for SQLite databases – Python 3.7.0a2 documentation.** Disponível em: <https://python.readthedocs.io/en/latest/library/sqlite3.html> Acesso em: 8 ago. 2023.
38. **Google Colab.** Disponível em: <https://research.google.com/colaboratory/intl/pt-BR/faq.html> Acesso em: 9 ago. 2023.

39. **Welcome to Buildozer’s documentation! – Buildozer 0.11 documentation.** Disponível em: <https://buildozer.readthedocs.io/en/latest/> Acesso em: 9 ago. 2023.