UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE LORENA

KAUÃ LUCAS NUNES MACHADO

LETÍCIA ANTEGHINI MIGLIARI

RICARDO A. COPPOLA GERMANOS

VITOR YANG CHIBA

Projeto de Automação Industrial com Microcontrolador ESP32, IDE Python e *Power BI*

Lorena - SP

2020

RESUMO

O Mapeamento do fluxo do valor (*Value Stream Mapping*, do inglês, VSM), busca por meio dos preceitos de análise e identificação de fluxo de processo, identificar e isolar os locais nos quais os processos de uma linha de produção estejam realmente agregando valor ao produto final e onde estes processos de estocagem significam desperdício de tempo. Este trabalho propõe a identificação e mitigação dos fluxos originadores de desperdícios, implícitos dos inventários em meio aos processos de manufatura e o beneficiamento da linha de produção, tudo isso de forma remota. Para isto, foram utilizados microcontroladores ESP32, juntamente com leitor RFID, onde o monitoramento destes ocorreu através de um sistema de comunicação com a nuvem, assim, as informações coletadas pelo leitor RFID foram publicadas no *Broker* MQTT. Os dados enviados para a nuvem foram assinados por um computador coletor e tratados com linguagem de programação Python, gerando arquivos CSV possíveis de serem analisados e permitindo, dessa forma, a posterior visualização interativa dos dados através do *Power BI*.

Palavras-chave: VSM, microcontrolador, ESP32, monitoramento, nuvem.

1. INTRODUÇÃO

1.1 Justificativa

A quarta revolução industrial tem prometido um novo modelo de monitoramento, conhecido como indústria 4.0. O trabalho proposto neste relatório possui como contribuição para o desenvolvimento desta indústria, a priori, a redução do tempo gasto com processos de produção que não agregam valor para o produto final (os quais se identificam principalmente no período de estocagem do produto); utiliza-se para isso, da automatização e informatização com disponibilização do fluxo de valor em tempo real, para que o gerenciamento do processo e, consequente otimização deste, seja realizado remotamente.

1.1 *Startup Healthus*

Uma vez que os assuntos que esse trabalho abrange são voltados para a indústria, no segmento de automação e monitoramento, o qual possui alto potencial de crescimento e é de suma importância, foi decidido levar a frente a ideia, criando-se, portanto, a *Startup Healthus*.

O nome da *startup* vem da união de duas palavras, em inglês, *Health* e *Thus*.“Saúde do negócio, simples assim”. Logo, é uma startup que visa cuidar dos negócios de uma forma inovadora e totalmente remota.

A *Healthus* constitui-se de *HManufacturing*, *HFinancial*, *HLean*, entre outros. Portanto, abrange qualquer solução que a empresa necessitar, usando a inovação através de *big data* para trabalhar com grandes volumes de dados, computação paralela e *machine learning*. Além da geração de relatórios e a apresentação de dados e indicadores.

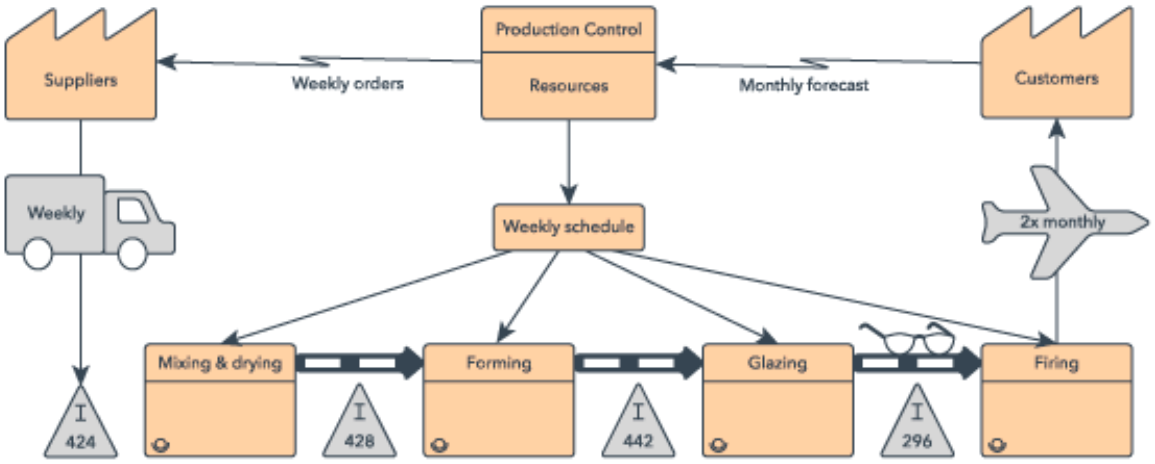
2. REVISÃO TEÓRICA

2.1 VSM

O Mapeamento de Fluxo de Valor, em inglês *Value Stream Mapping* (VSM), é um método de fluxograma para facilitar a visualização, ajudar na análise e melhorar as etapas essenciais para a entrega de um serviço ou produto (LUCIDCHART, 2020). O VSM permite a construção de um diagrama dinâmico, que apresenta o fluxo de materiais e dados ao longo de toda a cadeia de produção, com o objetivo de diminuir os desperdícios e estruturar ações

de melhorias (NOTEGUSBISIAN, 2018). Abaixo, na Figura 1, é apresentado um exemplo de VSM.

Figura 1 - Exemplo de VSM



Fonte: LUCIDCHART, 2020.

O VSM é uma ferramenta que segmenta os seus processos em atividades agregadoras de valor AV e atividades não agregadoras de valor, no primeiro grupo constam processos que envolvem a confecção e otimização do produto, enquanto no segundo estão os processos em que nenhum valor está sendo agregado a eles, por exemplo os processos de estocagem. Conhecendo o tempo de ambos os grupos, é possível determinar o grau de eficiência do processo (NOTEGUSBISIAN, 2018). Por definição, o PCE é a razão entre o tempo dos processos AV pelo tempo total dos processos (ISIXSIGMA, 2018), obedecendo então a seguinte equação:

$$PCE = \frac{AV}{(AV+NAV)}$$

O escopo desse projeto lida com o que foi chamado de vsmR, ou seja, Mapeamento de Fluxo de Valor Remoto. Sendo assim, não há necessidade de ninguém estar pessoalmente no chão de fábrica, a ideia é ter a segmentação de todos os processos e de todos os estoques remotamente.

2.2 ESP32

O ESP32 é um microcontrolador com um alto nível de integração; um consumo de energia muito baixo; um *design* robusto, capaz de operar em temperaturas que variam de

-40ºC a +125ºC e possuem a conectividade *WiFi* e *Bluetooth* embutidas na própria placa (ESPRESSIF, 2020). Esse microcontrolador pode ser programado por diferentes plataformas, inclusive o IDE do arduíno (ATHOS ELECTRONICS, 2019) e segue as especificações apresentadas na Tabela 1 abaixo.

Tabela 1 - Especificações do ESP32

CPU:	Xtensa® Dual-Core 32-bit LX6
Memória ROM:	448 KBytes
Clock máximo:	240MHz
Memória RAM:	520 Kbytes
Memória Flash:	4 MB
Wireless padrão 802.11 b/g/n	
Conexão Wifi de 2.4Ghz (máximo de 150 Mbps)	
Antena embutida na placa	
Conector micro USB para comunicação e alimentação	
Wi-Fi Direct (P2P), P2P Discovery, P2P Group Owner mode e P2P Power Management	
Modos de operação: STA/AP/STA+AP	
Bluetooth BLE 4.2	
Portas GPIO:	11
GPIO com funções de PWM, I2C, SPI, etc	
Tensão de operação:	4,5 ~ 9V
Conversor analógico digital (ADC)	

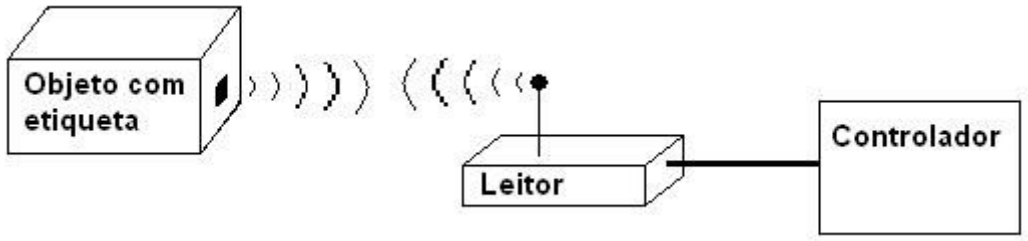
Fonte: ATHOS ELECTRONICS, 2019.

2.3 RFID

A identificação por frequência de rádio (RFID) é uma tecnologia que se utiliza de ondas de rádio para identificar objetos de forma automática. Esse sistema consegue se comunicar com a maioria dos objetos e por também ter um baixo custo envolvido, apresentam grandes vantagens para as linhas de produção modernas e cadeias de abastecimento (MONTEIRO et al., 2010).

Para se comunicar, o RFID utiliza principalmente três componentes: uma etiqueta, um leitor e um controlador (MONTEIRO et al., 2010). Esse esquema é ilustrado melhor na Figura 2 abaixo.

Figura 2 - Esquema simples do funcionamento de RFID



Fonte: MONTEIRO et al., 2010.

Sobre esses três componentes, temos primeiro a etiqueta, que pode ser um chip semicondutor, uma antena ou uma bateria; o leitor, que é formado por antena, módulo eletrônico de frequência de rádio e módulo de controle; e, por último, o controlador, que é um computador ou uma base que possua um banco de dados e seja capaz de rodar o software de controle. Ao entrar no campo do leitor, a etiqueta envia as informações que contém através das ondas de rádio e então o leitor as envia para o controlador (MONTEIRO et al., 2010).

Ainda sobre o RFID, existem três tipos de etiquetas, que são divididas de acordo com a sua memória: Read-only, que permite apenas a leitura de seus dados e normalmente possuem pouca informação; Read/Write (RW), que envia os dados de leitura, mas também permite que as informações sejam editadas ou apagadas; e a mista, que possui alguns dados que são fixos, como por exemplo o nome do produto, e outros que podem ser editados.

2.4 *NTP*Client

Os relógios de computadores e outros dispositivos não são muito precisos e podem ser descalibrados ao longo do tempo, acabando adiantando ou atrasando, o que faz com que alguns softwares e aplicativos possam apresentar problemas relacionados à sincronização do tempo. Nesse cenário o Protocolo de Tempo para Redes (NTP) se mostra importante, pois ele é capaz de sincronizar a data e hora do relógio dos dispositivos de uma rede com alta precisão (NTP, 2018).

2.5 *Broker* MQTT

Pensando nos dispositivos de Internet das Coisas IoT, existe um fator essencial para que consigam funcionar bem, que é a conexão com a internet, pois é exatamente essa conexão que permite que esses dispositivos consigam trabalhar entre si e com o serviço de backend, protocolo que se tornou padrão atualmente para a comunicação IoT é o MQTT

(*Message Queue Telemetry Transport*), para executar a sua função é utilizado um modelo de publicação e assinatura (YUAN, 2017).

Esse modelo pressupõe a definição de dois tipos de entidade: o *Broker* e os clientes. *Broker* é um servidor que recebe todas as mensagens dos clientes, então envia para os clientes de destino relevante. Os chamados "Clientes" são os dispositivos ou sistemas capazes de enviar ou receber informação desse servidor (YUAN, 2017).

A ordem do funcionamento desse protocolo é a seguinte: primeiro o cliente se conecta ao *Broker*, com a capacidade de assinar qualquer tópico de mensagem nesse servidor; então o cliente cliente publica uma mensagem em um tópico, enviando ambos para o *Broker*; por fim, o *Broker* encaminha a mensagem para todos os clientes de destino relevante, que são aqueles que assinaram o tópico que foi atualizado (YUAN, 2017).

2.6 Python

Python é uma linguagem poderosa e fácil de aprender, isso porque possui estruturas de dados de alto nível e apresenta uma abordagem simples, mas com eficácia para a programação orientada a objetos (PYTHON, 2020). Além disso, essa linguagem também é livre e multiplataforma, isso quer dizer que não é preciso pagar para a sua utilização e que seus programas podem rodar em diversas plataformas sem qualquer modificação, sendo que nas exceções é possível modificar o código da linguagem para tornar possível rodar (PYSCIENCE-BRASIL, 2020).

Atualmente, com um suporte para as bibliotecas melhorado, essa linguagem se tornou popular no quesito de análise de dados, levando em consideração a sua estrutura para engenharia de *software* de uma maneira geral, o Python é uma ótima opção para a construção de aplicações em dados (MCKINNEY, 2018).

2.7 *Power BI*

O *Power BI* é uma ferramenta da empresa *Microsoft*, sua principal função é a apresentação de informações por meio da criação de painéis de gestão. Outra funcionalidade dessa ferramenta é o ETL (*Extract, Transform and Load*) de dados, que permite a conexão com diferentes tipos de fontes de informação, o que possibilita criar um ambiente integrado com informações vindo de variados bancos de dados, que podem ser atualizadas em tempo real e podem ser acessadas por diferentes dispositivos (DEVMEDIA, 2017).

Um projeto com essa ferramenta pode ser resumida em três principais fases: criação das relações entre as fontes de dados (pode ser uma fonte ou mais); tratamento dos dados e desenvolvimento dos painéis a partir de componentes gráficos (DEVMEDIA, 2017).

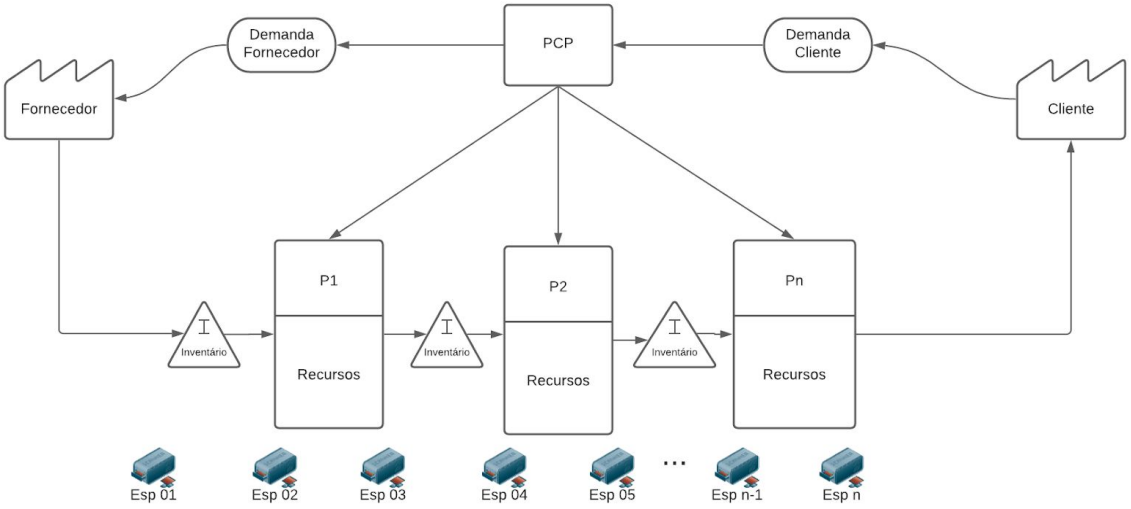
3. METODOLOGIA

Nessa seção do relatório é explicado como os ESP32 são dispostos no chão de fábrica, como é feita a conexão do ESP32 com o leitor RFID (englobando o circuito, a montagem e as bibliotecas utilizadas), é explicado também sobre a coleta de dados pelo computador coletor Python para posterior tratamento no *Power BI* e publicação na rede e, finalmente, é apresentado o esquema de todos os processos feitos no trabalho, desde o chão de fábrica até a publicação na rede.

3.1 ESP32 na linha de produção

Já foi explicado na Revisão Teórica sobre a importância do Mapeamento de Fluxo de Valor (VSM), e o fato de que esse trabalho visa um mapeamento remoto, denominado assim de vsmR. Com isso, para exemplificar como utilizamos os ESP32 para realizar esse mapeamento no chão de fábrica, é apresentada a Figura 3, ou seja, o esquema do vsmR.

Figura 3 - Esquema do vsmR



Fonte: Autoria própria.

Nesse esquema podem ser vistos os ESP32 posicionados na linha de produção de forma sequencial, portanto, o ESP01 se encontra no início do estoque 1 ao mesmo tempo que

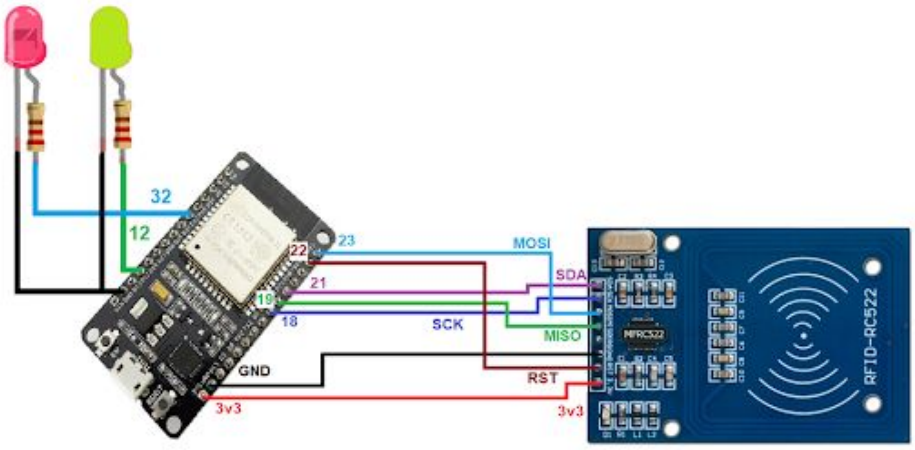
o ESP02 se encontra no final do estoque 1 e início do processo 1, depois o ESP03 se encontra no final do processo 1 e início do estoque 2, e assim sucessivamente até *n* estoques e *n* processos. Dessa forma, conseguimos obter o tempo de passagem dos produtos em cada etapa da linha de produção, e fazendo a diferença entre esse tempo conseguimos obter o tempo que o produto levou em cada etapa.

Logo, com essa disposição, é possível obter os dados dos tempos, que é o necessário para saber a eficiência do processo; a partir disso foi preciso trabalhar com a programação e o tratamento desses dados, a fim de conseguir transformar os dados em informações úteis e visuais.

3.2 ESP32 com RFID

Com os ESP32 posicionados no chão de fábrica os produtos contendo as tags RFID passam por eles, portanto, os leitores RFID recebem as informações enviadas pelas tags, codificam e enviam os dados para a aplicação no servidor. Assim, foi preciso construir um circuito e um código que fizesse a leitura dessa passagem das tags RFID, gravasse os dados e publicasse no *Broker*. Abaixo, na Figura 4 é apresentado o esquema do circuito. Nesse circuito o ESP32 é alimentado pela USB e ligado no serial do IDE Arduino.

Figura 4 - Esquema do circuito com ESP32, 2 LEDs e leitor RFID (RC522)



Fonte: KOYANAGI.

Montado o circuito foi preciso construir o código no Arduino, o código deste projeto utilizou 6 bibliotecas:

- WiFi.h: auxilia o leitor a iniciar a programação referente a comunicação *WiFi*;

- PubSubClient: para fazer mensagens simples de publicação/assinatura com um servidor que suporta MQTT;
- MFRC522: responsável pela comunicação com o módulo RFID-RC522;
- SPI.h: responsável pela comunicação com barramento SPI;
- *NTP*Client: responsável pela conexão ao servidor *NTP* para obtenção de data e hora;
- WiFiUdp.h: permite o envio de informações através da rede.

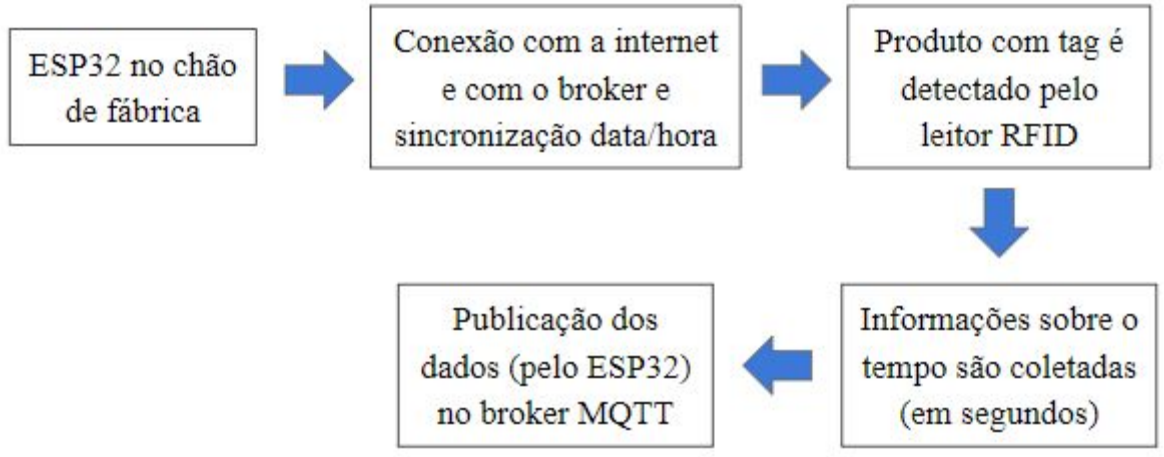
O código foi escrito com partes referentes à configuração do *WiFi*, determinação e configuração do *Broker*, equipamento de publicação do *Broker* (o RFID), conexão do ESP32 com o RFID, informações e configurações de data e hora (*NTP*Client), conversão do tempo para segundos, leitura da tag, acendimento dos LEDs para informações relacionadas à conexão do *WiFi* e do *Broker* e acendimento também para leitura da tag.

Logo, com o código escrito, as etapas que ocorrem são as seguintes: conexão com a internet, depois sincronização com o *NTP*Client coletando assim informações de data e hora, e conexão com o *Broker*. Quando a tag é lido ele é identificado pelo código único do cartão (ID) e os dados de leitura são coletados (dados de tempo convertidos em segundos), por último, essas informações são publicadas no *Broker* MQTT com a seguinte configuração:

Código do cartão	Código ESP na linha de produção	Tempo em segundos

Para resumir o que foi explicado acima é apresentada a Figura 5 a seguir.

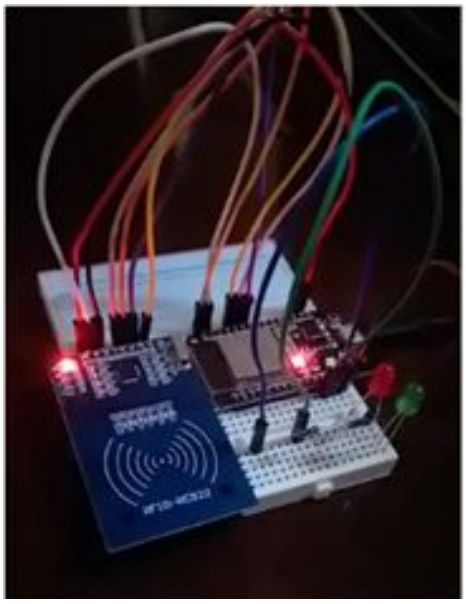
Figura 5 - Fluxograma das etapas do chão de fábrica até a publicação no *Broker* MQTT



Fonte: Autoria própria.

Por fim, na Figura 6 é mostrado o circuito finalizado, estando pronto para ser colocado em prática e iniciar a simulação do projeto.

Figura 6 - Circuito pronto para a simulação, com ESP32, leitor RFID, LED vermelho e LED verde



Fonte: Autoria própria.

3.3 Python para coleta de dados

Com os dados publicados no *Broker* MQTT, o computador coletor em Python pode assinar o *Broker*, coletando as informações e fazendo o tratamento necessário dos dados, conseguindo assim, gerar os arquivos CSV da variação de tempo que o produto levou tanto em cada processo, quanto em cada estoque da linha de produção.

Para isso foi preciso criar o código em Python. O código utilizou as seguintes bibliotecas:

- Paho.mqtt.client: responsável por conectar o Python ao *Broker* MQTT para subscrever tópicos e receber as mensagens publicadas;
- Sys: fornece funções e variáveis usadas para manipular diferentes partes do ambiente de tempo de execução do Python;
- Pandas: para manipulação e análise de dados;

- Numpy: usada principalmente para realizar cálculos em Arrays Multidimensionais, possuindo uma larga coleção de funções matemáticas para trabalhar com estas estruturas.

Foram criadas duas subrotinas no código, uma para subscrição e outra para coletar a mensagem da assinatura. Essa mensagem precisou ser tratada para ocorrer a transformação em arquivo CSV. O tratamento de dados foi feito através de listas e matrizes. Foi preciso primeiro verificar na leitura do tempo o ESP32, ou seja, verificando o ESP32 da entrada do primeiro estoque (ESP01) e o ESP32 da saída desse primeiro estoque (ESP02) foi obtido o tempo que produto demorou no primeiro estoque; o mesmo para o primeiro processo (ESP02 e ESP03) e assim sucessivamente, até o final da linha de produção, pegando cada tempo e calculando sua variação (tempo de saída menos tempo de entrada). Com isso, foram gerados os dois arquivos CSV contendo essas informações, um arquivo relacionado ao processo e o outro arquivo relacionado ao estoque.

Os arquivos CSV seguem a seguinte configuração:

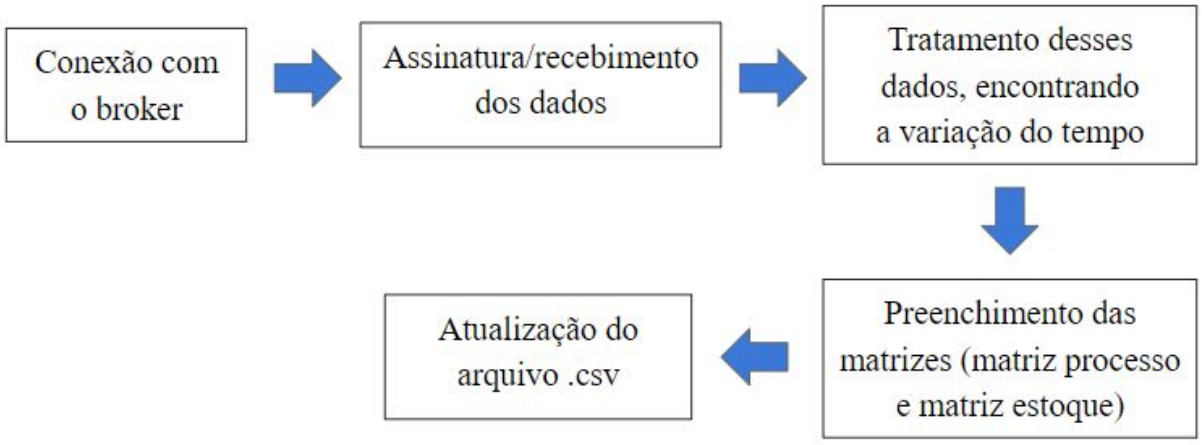
Código do cartão	Código ESP na linha de produção	Variação Δt

Vale ressaltar, que o código foi escrito de maneira que houvesse uma sequência correta das etapas que o produto precisava percorrer, para que assim fossem relatados possíveis erros na linha de produção, como por exemplo, a omissão de alguma etapa. Além disso, foi considerada a questão do reaproveitamento dos códigos do cartão, com isso, considerou-se que após a passagem na última etapa da linha de produção, o produto poderia retornar ao início, realizando novamente toda a sequência.

É importante citar que para que a comunicação entre os programas ocorresse corretamente, foi preciso utilizar os mesmos servidores, tanto no Arduino, quanto no Python, que no caso foi o *Broker* test.mosquitto.org usando o tópico MicroEEL/VSM.

Para resumir o ciclo descrito acima é apresentada a Figura 7, a qual refere-se ao fluxograma das etapas no computador coletor Python.

Figura 7 - Fluxograma das etapas relacionadas ao computador coletor Python, desde a conexão com o *Broker* até a atualização do arquivo CSV



Fonte: Autoria própria.

3.4 *Power BI* e publicação na rede

Os arquivos CSV gerados pelo Python foram importados pelo *Power BI* para observação dos dados de uma maneira facilmente visual e interativa. Vale ressaltar que antes de gerar todas as informações na *dashboard*, as tabelas dos arquivos CSV precisaram passar por um pequeno tratamento. Após esse tratamento e após todo o processo de organização das informações, foi possível verificar na *dashboard* diretamente a eficiência do ciclo (%), o tempo de valor agregado (processo), o tempo de valor não agregado (estoque), assim como a variabilidade desses tempos, ou seja, o desvio padrão.

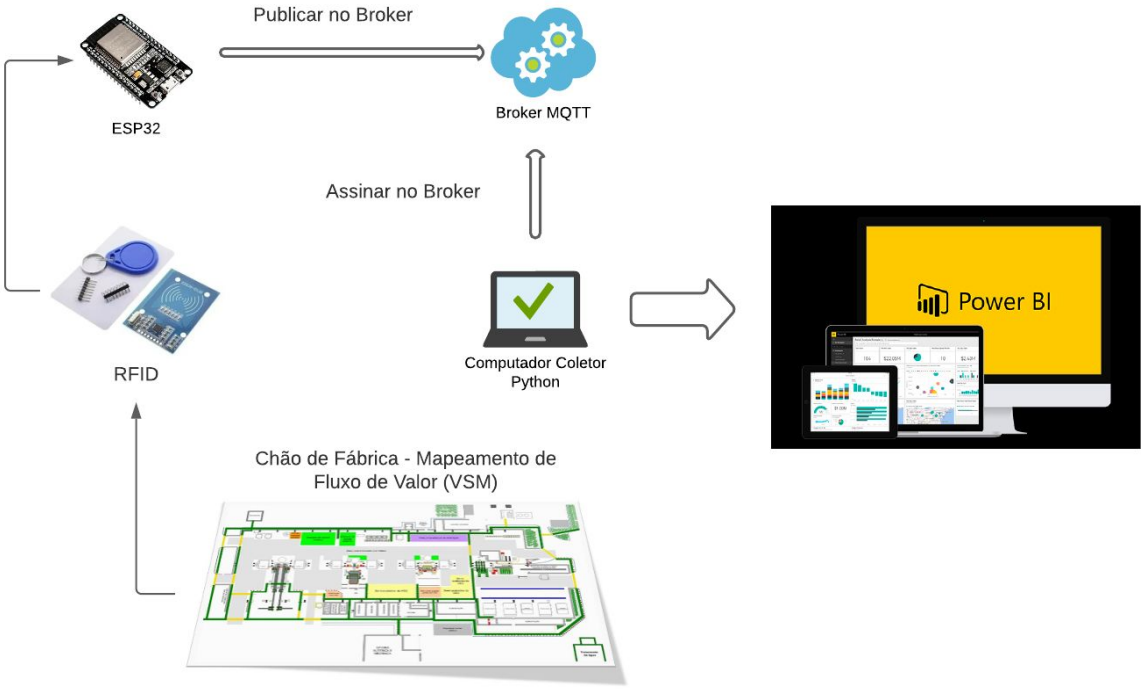
Além disso, também tem-se informações referentes a cada um dos processos e estoque separadamente, com valores mínimos, médios e máximos, e a linha de *Tack-Time*. A segmentação dos dados foi feita pelo número da *tag*, pelo número do processo e pelo número do estoque.

Por fim, a *dashboard* foi publicada na internet, para poder ser acessada através de celular, tablet ou qualquer computador na rede, sendo possível fazer o acompanhamento remoto das informações. Para isso, foi utilizada a função de publicação na rede *Microsoft*, onde nela, é permitida a configuração das atualizações, assim, pode-se escolher o tempo a qual se deseja que as informações sejam atualizadas. Portanto, qualquer pessoa, em qualquer hora e em qualquer lugar do mundo pode acompanhar as informações da linha de produção de forma interativa e atualizadas em tempo real.

3.5 Esquema completo das etapas do projeto

Para finalizar, é apresentado na Figura 8 um fluxograma completo com todas as etapas realizadas no projeto, desde o chão de fábrica até a visualização dos dados no *Power BI*. Essa figura exemplifica tudo que foi descrito anteriormente.

Figura 8 - Fluxograma de todas as etapas do projeto



Fonte: Autoria própria.

4 RESULTADOS E DISCUSSÃO

4.1 Execução do Protótipo

O protótipo desenvolvido em sistema embarcado no ESP32 foi programado para receber e tratar os dados fornecidos ao sensor por cada ID das *tags* embutidos nos produtos em linha de produção industrial. Conforme pode ser visto na Figura 9a, o código é inicializado através das bibliotecas que foram descritas na seção de metodologia, subseção 3.2. Como de praxe toda IDE Arduino passa pela subrotina `SETUP()` que será responsável pelas configurações iniciais da conexão *WiFi*, comunicação *Broker* MQTT, comunicação com o RFID e inicializações padrões como os LED internos (azul) e externos (verde e

vermelho). Além dessa subrotina de configuração tem-se a subrotina `LOOP()` que apresenta a estrutura de repetição principal do programa que inicia com a conexão da rede local através da biblioteca `WiFi.h`, que em seguida se conecta ao *Broker* que será descrito à frente. Ocorrendo alguma falha na conexão tanto com o rede *WiFi* quanto com o *Broker* o microcontrolador acende o LED azul interno ao ESP32. Em caso das duas conexões obtiverem êxito o LED azul pisca intermitentemente.

Ainda na Figura 9a, a subrotina `LOOP()` fica à espera da passagem da tag de identificação do RFID. Com o êxito dessa leitura o sistema prossegue para sincronização da data e hora para registrar o tempo exato que o produto na linha de produção passa por este ponto determinado da esteira para posteriormente ser computado com uma variação de tempo de estoque ou de processo.

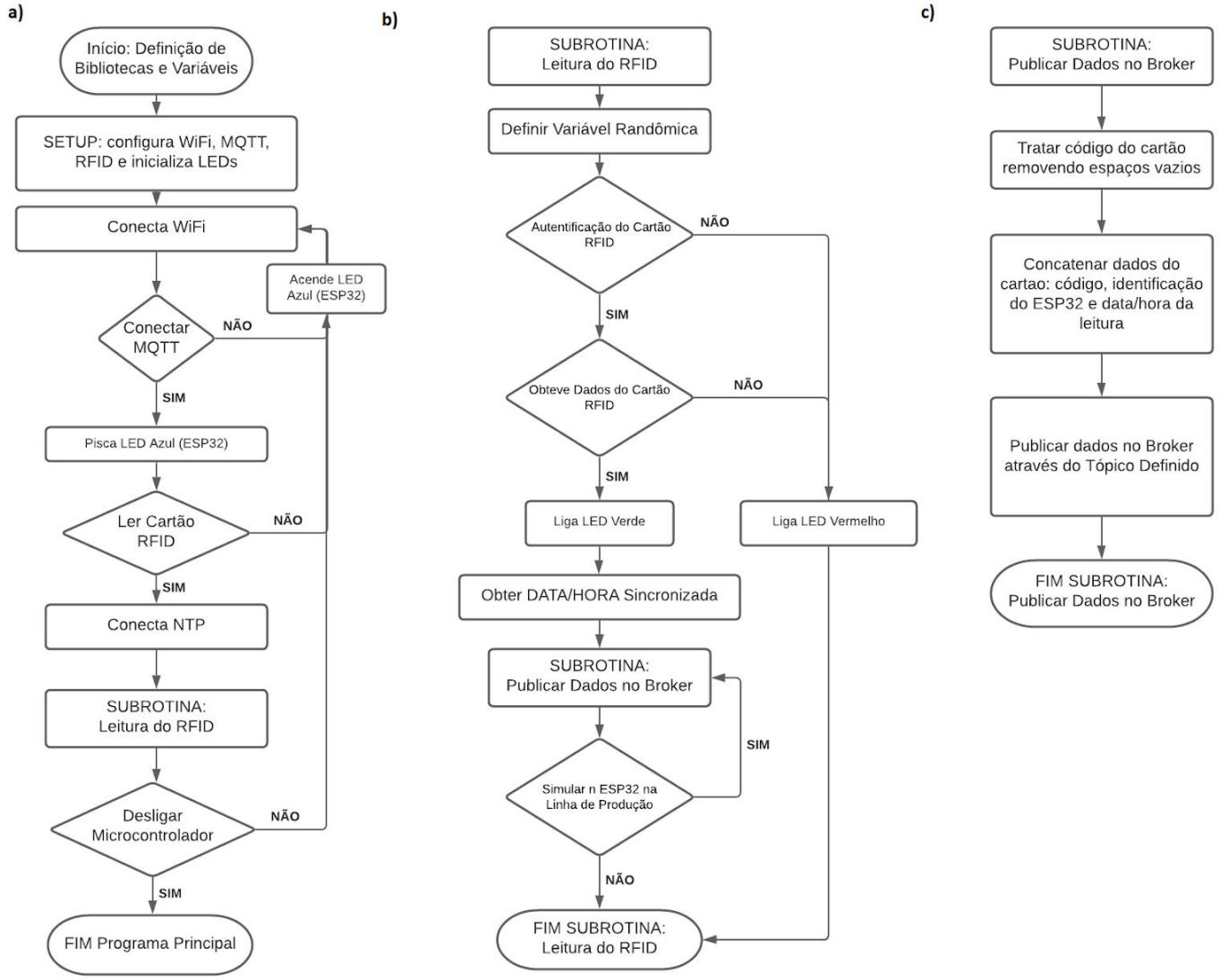
Em seguida, a subrotina Leitura do RFID é chamada e os detalhes desse algoritmo serão apresentados através da Figura 9b. Ainda na Figura 9a, para finalizar o processo principal, tem-se uma estrutura condicional que verifica se o programa principal finalizou através do desligamento do microcontrolador e, em caso contrário, retorna ao início da subrotina `LOOP()`, constituinte da IDE Arduino. Note que o LED interno volta a acender ininterruptamente para novamente passar pelos processos de conexão com a rede *WiFi*, *Broker* e sincronização com o protocolo de data/hora. Vale ressaltar que o microcontrolador pode ser operado como um servidor, contudo, tais configurações irão depender do ambiente requisitado pela empresa para a implementação dos sensores ao longo da linha de produção.

Como pode ser visto no fluxograma, primeiramente, é definido um código para gerar um valor randômico. Esta funcionalidade foi usada para criar uma simulação de microcontroladores que deveriam estar distribuídos pelo chão de fábrica para coletar o tempo que o produto em manufatura passa pelos vários estágios da linha de produção.

Após isso, é executada uma autenticação da leitura da *tag* do RFID e, consequentemente, a obtenção dos dados da *tag* que foi lido pelo leitor RFID e, este, em caso de erro, acende o LED vermelho indo para o final da subrotina. Em caso positivo, acende-se o LED verde dando continuidade ao processo de publicação no *Broker*. Em consequência, a data e hora atual são requisitadas para consolidar a informação que será passada ao *Broker*. Atualmente, a data não foi utilizada, porém para futuras implementações esta informação será incorporada ao programa para contemplar linhas de produção que trabalham de forma ininterrupta. A subrotina Publicar Dados no *Broker* será mostrada através da figura 9c. Dando continuidade ao fluxograma mostrado na Figura 9b, a etapa seguinte mostra a simulação de

vários microcontroladores espalhados pela linha(s) de produção. Essa estratégia foi usada para gerar tanto vários tempos de processo e estoque como criar uma variabilidade de tempo destes através da função randômica explicitada anteriormente. Umas das funcionalidades do vsmR ou mapeamento de fluxo de valor remoto é apontar tais variabilidade na linha de produção para resolver os problemas identificados no conceito *Lean Six Sigma*.

Figura 9 - Fluxograma do código do microcontrolador



Fonte: Autoria própria.

Por último, a figura 9c descreve as etapas da subrotina Publicar Dados no *Broker* que inicialmente realiza um tratamento em relação ao código da *tag* que o leitor RFID identificou. A ideia aqui é facilitar o tratamento de dados que será feito pelo servidor que realizará a assinatura no *Broker* e obterá os dados gerados pelos microcontroladores distribuídos pela linha de produção. De posse dos dados de identificação da *tag*, identificação do EPS32 e da

hora atual - lembrando que ainda não foi considerada a data - estas informações são concatenadas na forma de texto para enfim serem publicadas no *Broker* http://test.mosquitto.org/, com o tópico `MicroEEL/VSM` e cada ESP32 apresentará uma identificação `RF_ID_N`, no qual, o termo `N` representa o código de cada microcontrolador para não haver conflito no uso do *Broker*. Concluída essa etapa a execução do código retorna à subrotina principal para restabelecer o `LOOP()` ou sair do programa caso o microcontrolador seja desligado.

4.2 Leds e sinalizações

Os LEDs do protótipo foram programados de maneira que se possa identificar quaisquer problemas com o procedimento de leitura das *tags* por meio de sinalizações para o usuário/operador do sistema. Abaixo (Tab. 2) são mostrados os sinais emitidos pelos LEDs e seus respectivos significados.

Tabela 2 - Tabela de sinalizações

LED azul piscando	O protótipo está conectado com a internet e com o brocker
LED verde aceso após leitura de cartão	Leitura dos dados do cartão RFID foi feita com sucesso
LED vermelho aceso após leitura do cartão	Indica problema na leitura do cartão RFID
LED azul aceso constantemente	Protótipo não está conseguindo acessar a rede ou o *Broker*

Fonte: Autoria própria.

4.3 Conexão *WiFi* e MQTT e sincronização de data/hora *NTP*Client

A primeira etapa consiste em realizar a conexão com a Internet através da subrotina `conectaWiFi()` como pode ser visto no segmento de código abaixo (Fig. 10). Se a conexão estiver estabelecida (`WL_CONNECTED`) a execução sai da subrotina. No entanto, se este teste falhar, a conexão é restabelecida e o monitor serial apresenta a descrição do processo de reconexão e o endereço IP fornecido pelo microcontrolador. Aqui também estuda-se, como uma implementação futura, a utilização de um *display* LCD para apresentar tais informações.

Figura 10 - Código de conexão com a rede *WiFi*

```
void conectaWiFi() {
  if (WiFi.status() == WL_CONNECTED)
       return;
  else {

       Serial.println();
       Serial.print("Conectando-se a ");
       Serial.print(ssid);
       WiFi.begin(ssid, password);

       while (WiFi.status() != WL_CONNECTED) {
       led(1);
       Serial.print(".");
       }

       Serial.println("");
       Serial.print("WiFi conectada no endereço de IP: ");
       Serial.println(WiFi.localIP());
  }
}
```

Fonte: Autoria própria.

Em seguida, o código de conexão com o *Broker* será mostrado na figura 11. Como pode ser visto, o código começa com uma estrutura de repetição que verifica a conexão com o protocolo MQTT. Em caso negativo, o código tenta reconectar-se novamente usando o *Broker* http://test.mosquitto.org/ fornecendo mensagem através do monitor serial em caso de sucesso na conexão ou falha tentando se reconectar novamente.

Feita a conexão com *WiFi*, é realizada uma sincronização do microcontrolador com o protocolo *NTP*Client para receber a data e hora do servidor que irá permitir a correta tomada de dados em sincronia de tempo. O código que mostra tal conexão pode ser observado através da subrotina `conectaNTP()` mostrado na figura 12. Esta calibração é importante, pois a partir desta haverá a sincronização do horário de todos os ESPs conectados à linha de produção. Ainda na Figura 12, o código foi modularizado com padrões tanto em relação às variáveis criadas como em relação às subrotinas para facilitar a compreensão do desenvolvedor que for usar ou adaptar esta solução às suas respectivas necessidades. Pode-se notar nesse procedimento que após inicializar o protocolo, uma atualização é estabelecida em dois estágios, no qual, caso, não haja sucesso na primeira, uma etapa força a atualização até o sucesso da atualização.

Figura 11 - Código de conexão com *Broker*

```
const char* BROKER_MQTT = "test.mosquitto.org";

void conectaMQTT() {
    while (!MQTT.connected()) {
       Serial.println(" ");
       Serial.print("Conectando ao Broker MQTT: ");
       Serial.println(BROKER_MQTT);
       if (MQTT.connect(ID_MQTT))
             Serial.println("Conectado ao Broker com sucesso!");

       else {
             Serial.println("Nao foi possivel se conectar ao broker.");
             Serial.println("Nova tentativa de conexão em 10 segundos.");
             led(1);
       }
    }
}
```

Fonte: Autoria própria.

Figura 12 - Código de conexão com *Broker*.

```
void conectaNTP()
{
       Serial.println();
       Serial.println("Sincronizando data/tempo com Protocolo NTPClient...");

       //Inicializa o client NTP
       ntpClient.begin();

       //Espera pelo primeiro update online
       Serial.println("Esperando pela primeira atualizacao...");
       while(!ntpClient.update())
       {
       Serial.print(".");
       ntpClient.forceUpdate();
       delay(500);
       }
       Serial.println("Atualizacao completada!");
}
```

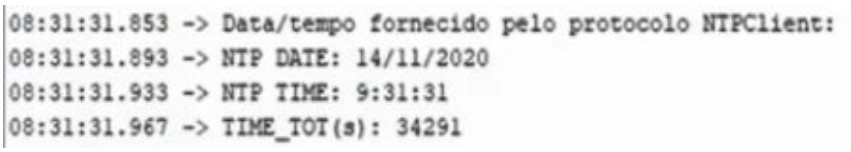
Fonte: Autoria própria.

4.4 Leitura do Tag RFID

Ao passar-se o cartão, o LED verde acende para indicar que as leituras foram feitas corretamente (Fig 13). A cada passagem das *tags* pelos sensores, são obtidos dados em tempo real, tanto dos processos, quanto do período de estocagem dos produtos. Estes dados são

convertidos em segundo no tempo total para correta sincronia e operação de cada ESP32 na linha de produção.

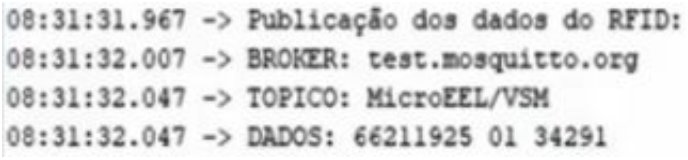
Figura 13 - Informações relativas ao tempo de aquisição dos dados do cartão RFID



Fonte: Autoria própria.

Os dados são publicados no formato: código do cartão, código do ESP32, tempo convertido em segundos no qual foi realizada a leitura. Estas informações são dispostas no monitor serial do IDE do Arduino. Abaixo são mostradas as informações relativas aos dados da *tag* que foi lido com RFID (Fig. 14).

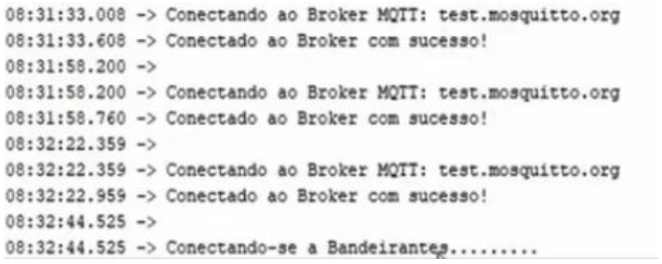
Figura 14 - Disposição dos dados da *tag* lidos no momento de sua identificação



Fonte:autoria própria.

Após finalizar a aquisição de informações do primeiro cartão lido, o protótipo continua tentando se manter conectado ao *Broker*, sendo um padrão do *script* de códigos, que ao finalizar uma rotina de leitura de informações da *tag*, este volta a tentar se conectar com a Internet e com o *Broker* (Fig. 15).

Figura 15 - Sucessivas tentativas de se conectar a rede *WiFi* e ao *Broker* até que uma leitura de cartão RFID interrompa o processo



Fonte: Autoria própria.

A partir do momento em que o protótipo encontra a rede, na qual, possa se conectar, este estabelece conexão novamente e volta à rotina de conectar-se ao *Broker*. Como pode ser visto através do código mostrado na Figura 16, primeiramente é coletada informações da *tag* lido do sensor de rádio frequência RFID. Em seguida, algumas variáveis como `Key` e `Buffer` são inicializadas para garantir a chamada da API para a biblioteca `MFRC522.h`, que conecta ao leitor. Nesse processo de inicialização é mostrado também a função randômica que será responsável por gerar a variabilidade de processo artificial (variável `rd`). Em seguida, inicia-se a autenticação da leitura da *tag* e se o parâmetro `STATUS` não retornar sucesso, mensagens de erros juntamente (monitor serial) com o LED vermelho são acionados. Caso contrário, é realizada a leitura do cartão através da API `mfrc522.MIFARE_Read()` que ao retornar sucesso na operação, fornece os valores lidos com a *tag* acendendo o LED verde. Mais uma vez a verificação da variável `STATUS` da leitura é avaliando tornando o método de verificação mais consistente. Por último, os dados da *tag* são concatenados através da conversão de dados do tipo hexadecimal para string e passados para a subrotina `enviar_dados()` que passará os parâmetros identificação da *tag*, identificação do microcontrolador e hora atual. Nas últimas linhas são apresentados o código que simula os vários microcontroladores na(s) linha(s) de produção. Como pode ser visto todo o processo de conexão com a rede *WiFi* e *Broker* são restabelecidos. Além disso, é disparado um comando `delay()` para criar o tempo de estoque e processo e, consequentemente, uma variabilidade artificial.

Figura 16 - Código para leitura da *tag* RFID

```
void ler_RFID()
{
//imprime os detalhes técnicos do cartão/tag
  Serial.println("Dados de leitura do RFID:");
  mfrc522.PICC_DumpDetailsToSerial(&(mfrc522.uid));

//Prepara a chave - todas as chaves estão configuradas para FFFFFFFFFFFFh
(Padrão de fábrica).
  for (byte i = 0; i < 6; i++)
       key.keyByte[i] = 0xFF;

  //buffer para colocar os dados lidos
  byte buffer[SIZE_BUFFER] = {0};

  //bloco que faremos a operação
  byte bloco = 1;
  byte tamanho = SIZE_BUFFER;
```

```
  long randNumber = random(5,10);
  float rd = randNumber/10.;
  Serial.println();
  Serial.print("Valor randômico: ");
  Serial.println(rd);

  //faz a autenticação do bloco que vamos operar
  status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, bloco,
&key, &(mfrc522.uid)); //line 834 of MFRC522.cpp file
  if (status != MFRC522::STATUS_OK) {

       Serial.print(F("Authentication failed: "));
       Serial.println(mfrc522.GetStatusCodeName(status));
       digitalWrite(pinVermelho, HIGH);
       delay(1000);
       digitalWrite(pinVermelho, LOW);
       return;
  }

  //faz a leitura dos dados do bloco
  status = mfrc522.MIFARE_Read(bloco, buffer, &tamanho);
  if (status != MFRC522::STATUS_OK) {

       Serial.print(F("Reading failed: "));
       Serial.println(mfrc522.GetStatusCodeName(status));

       digitalWrite(pinVermelho, HIGH);
       delay(1000);
       digitalWrite(pinVermelho, LOW);
       return;

  } else {

       digitalWrite(pinVerde, HIGH);
       delay(1000);
       digitalWrite(pinVerde, LOW);
  }

  //Mostra UID na serial
  String id_carta = "";
  for (byte i = 0; i < mfrc522.uid.size; i++)
       id_carta.concat(String(mfrc522.uid.uidByte[i], HEX));

  enviar_dados(id_carta.c_str(),"01",getTime().c_str());   //ESP32 código 01

  //SIMULAÇÃO DA BANDEJA PASSANDO POR VÁRIOS RFIDs NA ESTEIRA DO CHÃO DE FÁBRICA
  conectaWiFi(); conectaMQTT(); delay(rd*1200);
  enviar_dados(id_carta.c_str(),"02",getTime().c_str());   //ESP32 código 02
  conectaWiFi(); conectaMQTT(); delay(rd*5500);
  enviar_dados(id_carta.c_str(),"03",getTime().c_str());   //ESP32 código 03
  conectaWiFi(); conectaMQTT(); delay(rd*1000);
```

```
  enviar_dados(id_carta.c_str(),"04",getTime().c_str());   //ESP32 código 04
  conectaWiFi(); conectaMQTT(); delay(rd*2100);
  enviar_dados(id_carta.c_str(),"05",getTime().c_str());   //ESP32 código 04
  conectaWiFi(); conectaMQTT(); delay(rd*1100);
  enviar_dados(id_carta.c_str(),"06",getTime().c_str());   //ESP32 código 04
  conectaWiFi(); conectaMQTT(); delay(rd*1300);
  enviar_dados(id_carta.c_str(),"07",getTime().c_str());   //ESP32 código 04
}
```

Fonte: Autoria própria.

Por último, a subrotina `enviar_dados()` realiza todo o tratamento na cadeia de caracteres concatenando a identificação da *tag*, identificação do microcontrolador que está enviando e a hora atual do sistema e, consequentemente, envia os dados ao Broker (Fig. 17). Como pode ver a função `remover espaços()` retira os espaços vazios do código de identificação da *tag* para facilitar a leitura do servidor coletor que fará a assinatura no *Broker*. Como de praxe, todos os indicadores de execução do código são apresentados através do monitor serial.

Figura 17 - Código para publicação dos dados no *Broker*

```
void enviar_dados(const char* id_carta, const char* id_esp32, const char*
tempo_ss) {

  char tag[strlen_P(id_carta) + strlen_P(id_esp32) + strlen_P(tempo_ss)+2] = "";
  const char* espaco_v = " ";

  strncat_P(tag,id_carta,strlen_P(id_carta));
  removerSpacos(tag);
  strncat_P(tag,espaco_v,strlen_P(espaco_v));
  strncat_P(tag,id_esp32,strlen_P(id_esp32));
  strncat_P(tag,espaco_v,strlen_P(espaco_v));
  strncat_P(tag,tempo_ss,strlen_P(tempo_ss));

  Serial.println("Publicação dos dados do RFID:");
  Serial.print("BROKER: ");
  Serial.println(BROKER_MQTT);
  Serial.print("TOPICO: ");
  Serial.println(TOPIC_PUBLISH);
  Serial.print("DADOS: ");
  Serial.println(tag);
  MQTT.publish(TOPIC_PUBLISH, tag);
}
```

Fonte: Autoria própria.

Rotinas auxiliares a todos esses processos são apresentados a seguir (Fig. 18 e 19). Tem-se as subrotinas para remover os espaços de uma cadeia de caracteres e a subrotina `getDate()`, que retorna tanto a data como a hora através de uma estrutura de dados que foi criada no presente código para atender as passagens de parâmetros da API do NTPClient (Fig. 18).

Figura 18 - Escopo dos códigos auxiliares

```
//Fuso Horário, no caso horário de verão de Brasília
int timeZone = -2;

//Struct com os dados do dia e hora
struct Date{
	int dayOfWeek;
	int day;
	int month;
	int year;
	int hours;
	int minutes;
	int seconds;
};

//Socket UDP que a lib utiliza para recuperar dados sobre o horário
WiFiUDP udp;

//Objeto responsável por recuperar dados sobre horário
NTPClient ntpClient(
	udp,                        //socket udp
	"0.br.pool.ntp.org",        //URL do servidor NTP
	timeZone*3600,              //Deslocamento do horário em relação ao GMT 0
	60000);                     //Intervalo entre verificações online

//Nomes dos dias da semana
char* dayOfWeekNames[] = {"Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday"};
```

Fonte: Autoria própria.

Figura 19 - Códigos auxiliares

```
void removerSpacos(char str[]) {
	int j = 1;
	int t = strlen(str);
	for (int i=1; i<t; i++) {
	if (!isWhitespace(str[i])) {
		str[j] = str[i];
		j++;
	}
	}
```

```
        str[j] = '\0';
}

Date getDate()
{
        //Recupera os dados de data e horário usando o client NTP
        char* strDate = (char*)ntpClient.getFormattedDate().c_str();

        //Passa os dados da string para a struct
        Date date;
        sscanf(strDate, "%d-%d-%dT%d:%d:%dZ",
                        &date.year,
                        &date.month,
                        &date.day,
                        &date.hours,
                        &date.minutes,
                        &date.seconds);

        //Dia da semana de 0 a 6, sendo 0 o domingo
        date.dayOfWeek = ntpClient.getDay();
        return date;
}

String getTime(){

  Date date = getDate();
  unsigned long time = 3600.*date.hours+60.*date.minutes+date.seconds;

  Serial.println();
  Serial.println("Data/tempo fornecido pelo protocolo NTPClient:");
  Serial.print("NTP DATE: ");
  Serial.print(date.day);
  Serial.print("/");
  Serial.print(date.month);
  Serial.print("/");
  Serial.println(date.year);

  Serial.print("NTP TIME: ");
  Serial.print(date.hours);
  Serial.print(":");
  Serial.print(date.minutes);
  Serial.print(":");
  Serial.println(date.seconds);

  Serial.print("TIME_TOT(s): ");
  Serial.println(time);
  Serial.println();

  return String(time, DEC);
}
```

Fonte: Autoria própria.

4.5 Assinatura do Servidor em Python

Quando o sensor lê a tag na linha de produção, esta informação é publicada através do ESP32 para publicação no *Broker* usando o protocolo MQTT. O código que estabelece essa comunicação pode ser visto na Figura 20. A biblioteca que permite essa comunicação com o Broker é a `paho.mqtt.client` que deve ser baixada através do próprio repositório Python através do comando `pipe`. Neste código foram definidos dois métodos de classes `on_connect()` e `on_message()` e tais métodos são sobrescritos através da classe `mqtt.Client()` fazendo com que o próprio servidor *Broker* use tais métodos. No método `on_connect()` é que a assinatura é feita para o mesmo tópico gerado na publicação do microcontrolador. Em seguida, ainda dentro do escopo `try-except` (tratamento de erro por exceção), o comando `client.connect()` mostra a conexão estabelecida através da identificação do *Broker* e a porta de comunicação e o comando `client.loop_forever()` tem-se a verificação toda vez que uma publicação é feita no *Broker* para o tópico inicializado. Uma apresentação mostrando essa iteração entre o microcontrolador realizando publicações e o servidor coletor em Python assinando, pode ser detalhada através do vídeo: https://www.youtube.com/watch?v=iNQ7zz7R2L0&feature=youtu.be, publicado pelos presentes autores. No método `on_message()` é capturada a mensagem assinada no *Broker* através do parâmetro *msg.payload*, no qual, o seu valor é atribuído à uma variável `String` ou cadeia de caracteres. Esse é o código básico para coletar informações do *Broker*, no entanto, é necessário tratar a variável do tipo `String` com nome `MensagemRecebida`.

Vale lembrar que através do Python será necessário gerar o arquivo CSV para criar os painéis de controle ou *dashboard*. O código que faz essa conversão pode ser visto através da Figura 21. Primeiramente, os dados são coletados em uma matriz, contudo, para se calcular as variações temporais tanto no estoque como nos processos é necessário usar duas medidas consecutivas de leitura do microcontrolador. Por exemplo, de acordo com a Figura 3 é necessário trabalhar com o ESP32 com identificação 01 e 02 para calcular o tempo do Estoque 01, o ESP32 com identificação 02 e 03 para calcular o tempo do Processo 01. Para calcular esses tempos de processo e estoque foi usado um controle de códigos de identificação par ou ímpar para diferenciar entre variações de tempo de estoque e processo. Através do método `.to_csv()` os dados nas matrizes são salvos de forma a agregar dados ao invés de criar arquivos novos, a cada vez que o programa coletor é executado. Através desses arquivos em formato CSV é possível atualizar os dados no *dashboard* no *Power BI*.

Como será explicado mais a frente é possível configurar o *Power BI* para atualizar os dados com uma determinada frequência.

Figura 20 - Códigos em Python para assinar o *Broker*

```
import paho.mqtt.client as mqtt
import sys

#definições:
Broker = "test.mosquitto.org"
PortaBroker = 1883
KeepAliveBroker = 60
TopicoSubscribe = "MicroEEL/VSM"

#Callback - conexão ao broker realizada
def on_connect(client, userdata, flags, rc):
	print("[STATUS] Conectado ao Broker. Resultado de conexão: "+str(rc))
	client.subscribe(TopicoSubscribe)

#Callback - mensagem recebida do broker
def on_message(client, userdata, msg):
	MensagemRecebida = str(msg.payload)

	print("[MSG RECEBIDA] Tópico: "+msg.topic+" / Mensagem:
"+MensagemRecebida)
try:
	print("[STATUS] Inicializando MQTT...")
	#inicializa MQTT:
	client = mqtt.Client()
	client.on_connect = on_connect
	client.on_message = on_message

	client.connect(Broker, PortaBroker, KeepAliveBroker)
	client.loop_forever()
except KeyboardInterrupt:
	print ("\nCtrl+C pressionado, encerrando aplicação e saindo...")
	sys.exit(0)
```

Fonte: Autoria própria.

Figura 21- Tratamento da mensagem em Python para criar o arquivo CSV

```
	MensagemRecebida = str(msg.payload)

	print("Topico: " + msg.topic + " / Mensagem: " + MensagemRecebida)
	dado_recebido = [str(MensagemRecebida.split()[0].replace("b'", '')),
			int(MensagemRecebida.split()[1]),
			int(MensagemRecebida.split()[2].replace("'", ''))]

	# Criação da matriz principal (recebe os dados)
```

```
	for c in range(0, len(matriz)):
	if matriz[c][0] == dado_recebido[0] and dado_recebido[1] == 1:
		del(matriz[c])

	for c in range(0, len(estoque)):
	if estoque[c][0] == dado_recebido[0] and dado_recebido[1] == 1:
		del(estoque[c])

	for c in range(0, len(processo)):
	if processo[c][0] == dado_recebido[0] and dado_recebido[1] == 1:
		del(processo[c])

	if not matriz and dado_recebido[1] == 1:
	matriz.append([dado_recebido[0], dado_recebido[2]])
	else:
	for i in matriz:
		if i[0] == dado_recebido[0]:
		key = 1
		try:
			i[dado_recebido[1]] = dado_recebido[2]
		except:
			i.append(0)
			i[dado_recebido[1]] = dado_recebido[2]
	if key == 0 and dado_recebido[1] == 1:
		matriz.append([dado_recebido[0], dado_recebido[2]])

	print(matriz)

	# Criação da matriz estoque e processo
	if not estoque and not processo:
	estoque.append([matriz[0][0]])
	processo.append([matriz[0][0]])
	else:
	key = 0
	for a in range(0, len(matriz)):
		for b in range(0, len(processo)):
		if processo[b][0] == matriz[a][0] and len(processo[b]) * 2 + 2 ==
len(matriz[a]):
			if len(matriz[a]) % 2 == 0 and len(matriz[a]) > 2:
				processo[b].append(matriz[a][len(matriz[a]) - 1] -
matriz[a][len(matriz[a]) - 2])
				key = 1
		for b in range(0, len(estoque)):
		if estoque[b][0] == matriz[a][0] and len(estoque[b]) * 2 + 1 ==
len(matriz[a]):
			if len(matriz[a]) % 2 == 1 and len(matriz[a]) > 2:
				estoque[b].append(matriz[a][len(matriz[a]) - 1] -
matriz[a][len(matriz[a]) - 2])
				key = 1
	if key == 0:
		estoque.append([matriz[len(matriz) - 1][0]])
```

```
            processo.append([matriz[len(matriz) - 1][0]])

      for i in matriz:
      if i[0] == dado_recebido[0] and dado_recebido[1] > 1:
            if len(i) % 2 == 1 and len(i) > 2:
            for n in estoque:
                  if n[0] == dado_recebido[0]:
                        x = [[n[0], len(n) - 1, n[-1], dado_recebido[2]]]
                        if x not in estoque_matriz and type(x[0][2]) != str:
                        estoque_matriz.append(x)
                        df_x = pd.DataFrame(x)
                        print(estoque)
                        with open('c:\projetoMicro\estoque.csv', 'a') as f:
                              df_x.to_csv(f, header=False, index=False)

            elif len(i) % 2 == 0 and len(i) > 2 and dado_recebido[1] > 2:
            for n in processo:
                  if n[0] == dado_recebido[0]:
                        y = [[n[0], len(n) - 1, n[-1], dado_recebido[2]]]
                        if y not in processo_matriz and type(y[0][2]) != str:
                        processo_matriz.append(y)
                        df_y = pd.DataFrame(y)
                        print(processo)
                        with open('c:\projetoMicro\processo.csv', 'a') as f:
                              df_y.to_csv(f, header=False, index=False)
```

Fonte: Autoria própria.

4.6 Geração de arquivo CSV e PowerBI

O *dashboard* a seguir (Fig. 22) mostra todas as informações necessárias que caracterizam um Mapeamento de Fluxo de Valor Remoto (vsmR), como, a eficiência do ciclo, a quantidade de valor agregado (no qual o produto está sofrendo processo de engenharia de qualidade ou de produção), valor não agregado (tempo de estoque em que não há atribuição de valor ao produto) e, por último, também se observa a variabilidade dos tempos de estoque e processo.

Os dados dispostos informam sobre os comportamento temporal de processos e estoques conforme mostra a Figura 22. Na parte superior à direita estão localizados os componentes de segmentação de dados, podendo segmentar pelo número de identificação da tag ou pela identificação de processos e estoques. Abaixo destes, tem os gráficos de barras para mostrar os tempos de processos e estoques de cima para baixo, respectivamente. As três colunas indicam tempo mínimo, médio e máximo. A linha cortando na horizontal mostra o *Takt-Time* que indica o limite máximo que o processo deve ter para garantir que a demanda do cliente seja atendida. Como pode ser visto (Fig. 22) que o tempo do processo máximo está

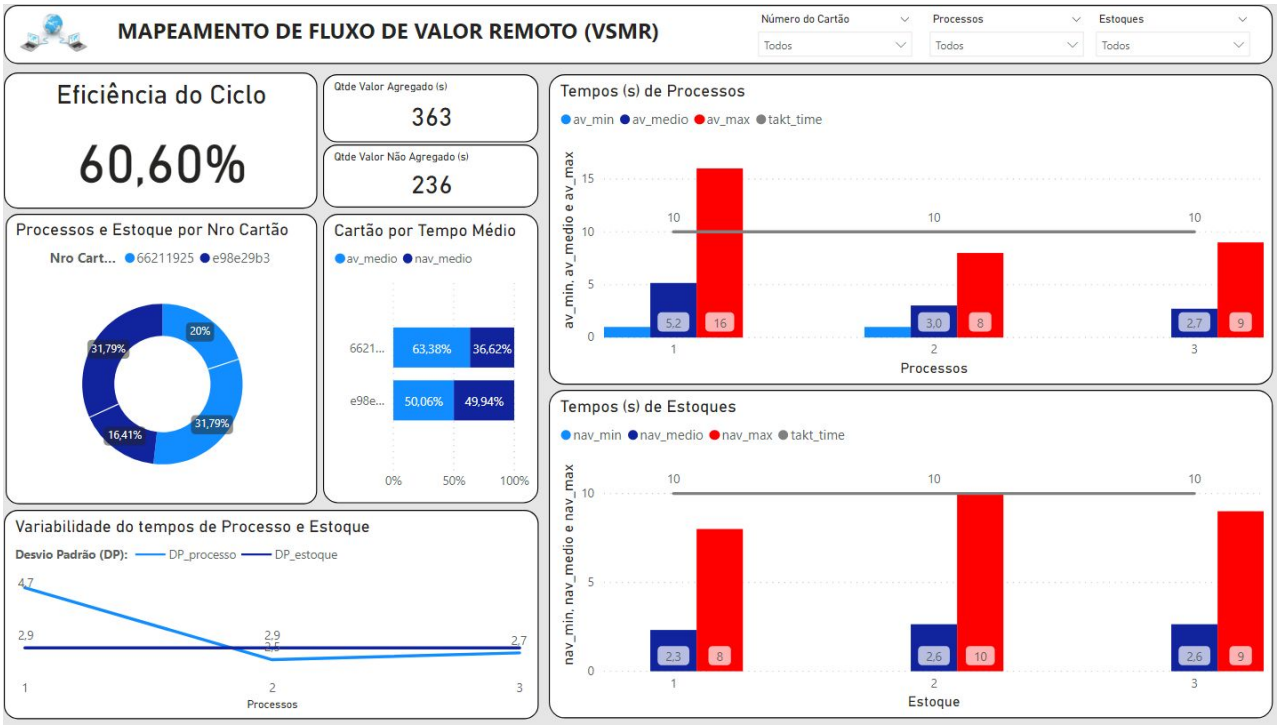
acima do *Takt-Time* que caracteriza que existe um ou mais processos que estão comprometendo o tempo de produção da linha. Esses valores foram gerados de forma proposital para compreender a importância dessa ferramenta remota para garantir a "saúde" do sistema de produção. Em relação ao estoque (Fig. 22) , todos os tempos estão abaixo do *Takt-Time*, apesar do estoque 2 apresentar um tempo muito próximo desse indicador o que sugere uma verificação desse tempo de estoque e medidas que possam minimizar tal tempo como, por exemplo, a mudança do *layout* ou mesmo a aplicação de mais operadores para atuar nesse estágio de estoque.

Olhando para a parte superior esquerda é possível ver cartões indicando a eficiência do ciclo que é um indicador para comparar valor agregado e não agregado e que pode revelar a evolução do processo de produção ao longo do tempo como as ações operacionais impactam na melhoria da produção. Ao lado deste cartão tem o tempo total em segundos do valor agregado e não agregado. Outras medidas também são mostradas abaixo desses cartões, contudo, a principal medida para atender o conceito *Six Sigma* é diminuir a variabilidade do processos conforme mostra o gráfico na parte inferior à esquerda.

Por exemplo, com o uso de filtros simples ou múltiplos, clicando nos gráficos com o auxílio da tecla *SHIFT* é possível selecionar, por exemplo, a variabilidade de qual processo se quer analisar. Além disso, para implementações futuras será incorporada a data do processo e desta forma seria possível selecionar um período de análise em relação a data e o tempo. Todos esses recursos são extremamente importantes para se estabelecer uma torre de controle do seu processo de produção e com o recurso de gerenciar a torre remotamente dá ao gestor uma maior mobilidade.

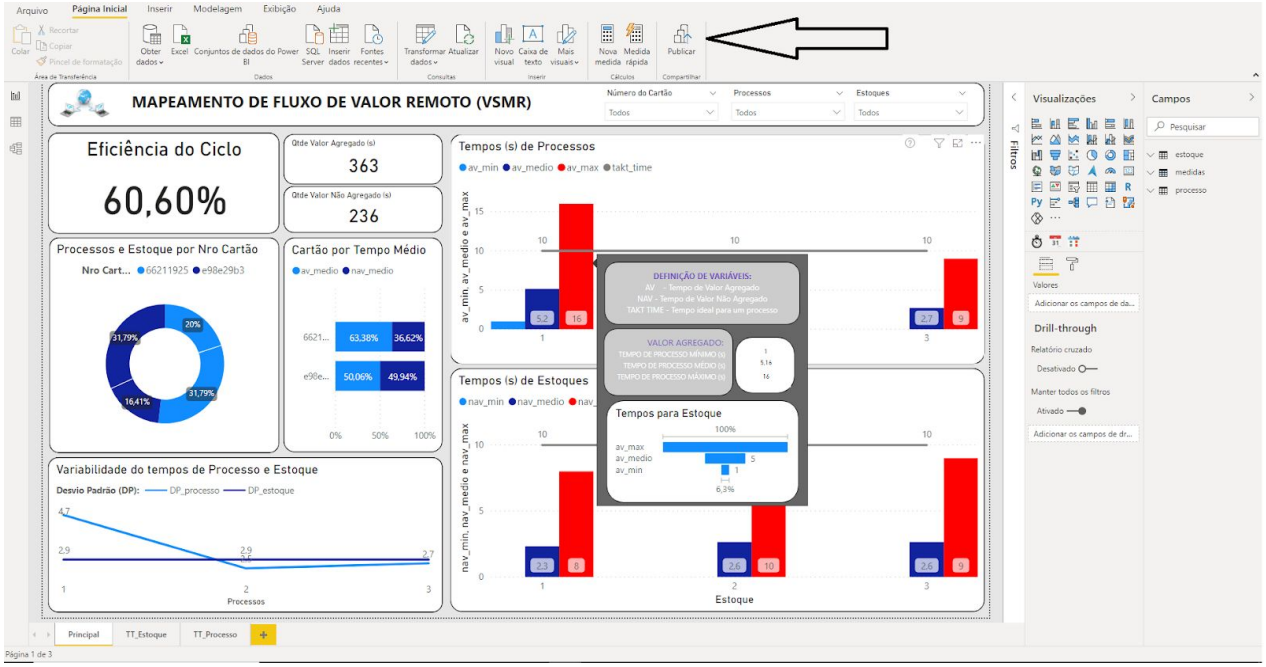
A Figura 23 mostra mais recursos do *dashboard* como os *tooltips* que são mostrados quando se passa o mouse por cima dos gráficos de barras mostrando tanto o processo como o estoque. Através desses *tooltips* pode-se ver informações detalhadas de variáveis, outros gráficos e dados que ajudem em uma análise mais detalhada conforme a necessidade do cliente. Em adicional, ainda na Figura 23, pode-se ver uma seta apontando para a função de publicação e será essa função que publica o *dashboard* na Nuvem da *Microsoft* e que torna o seu painel de controle acessível remotamente. Também existe a função de publicar no formato de página *WEB* o que agrega mais um recurso para a presente proposta de gerenciar as empresas remotamente e com uso de *Big Data*, *Machine* e *Deep Learning*.

Figura 22- *Dashboard* do mapeamento de fluxo de valor remoto (vsmR)



Fonte: Autoria própria.

Figura 23 - *Dashboard* do mapeamento de fluxo de valor remoto (vsmR) evidenciando a função de publicação e o *tooltip* com informações detalhadas



Fonte: Autoria própria.

5. CONCLUSÃO

Já é de conhecimento de todos a grande importância do acompanhamento em tempo real da linha de produção nas fábricas e indústrias. Diante disso, esse trabalho foi realizado com o intuito de automatizar esse acompanhamento, gerando informações relevantes e que podem ser acompanhadas de forma remota.

Para isso, foram utilizados diversos conceitos e aplicações práticas referentes à automação, microcontroladores e programação, onde foi necessário conectá-los para que todos trabalhassem juntos e de forma sincronizada. Foi utilizado o ESP32 juntamente com RFID para o monitoramento do processo e do estoque no chão de fábrica, dados sobre o tempo que o produto demorou em cada etapa foram coletados e publicados no *Broker* MQTT, com isso foi possível assinar esses dados da nuvem com um computador coletor (Python), o qual tratou as informações recebidas gerando arquivos CSV para análise de dados. Por fim, foi prezado neste trabalho o fornecimento final das informações de maneira resumida e com visualizações fáceis e interativas, logo, os arquivos foram analisados em *Power BI*.

Sendo assim, conclui-se a completa aplicação prática e de forma efetiva de todos os conceitos citados, resultando, portanto, no sucesso de obter o monitoramento completo da linha de produção atualizado em tempo real e de forma remota.

5.1 Implementações futuras para o protótipo piloto da *HManufacturing*

- Implementar controle por data para avaliar processos de produção que são realizados que duram mais do que 24 horas;
- Trabalhar com várias linhas de produção de forma simultânea;
- Devido a essas duas implementações atribuir ao *dashboard* tanto a segmentação de dados por data, como por linha de produção;
- Controle dos microcontroladores ESP32 através do protocolo de comunicação HTTP;
- Uso de *WifiManager* para facilitar a configuração da rede local em que os microcontroladores serão conectados.

5.2 Agradecimentos

Agradecimentos a Vinicius de Souza Meirelles e ao professor Carlos Yujiro Shigue.

REFERÊNCIAS

ATHOS ELECTRONICS. ESP32 – Especificações e projetos. [*S. l.*], 9 mar. 2019. Disponível em: https://athoselectronics.com/esp32/. Acesso em: 6 dez. 2020.

DEV MEDIA. Microsoft Power BI: primeiros passos. Microsoft Power BI: primeiros passos, [*s. l.*], p. 1-1, 13 abr. 2017. Disponível em: https://www.devmedia.com.br/microsoft-power-bi-primeiros-passos/38128. Acesso em: 3 dez. 2020.

ESPRESSIF. ESP32. [*S. l.*], entre 2016 e 2020. Disponível em: https://www.espressif.com/en/products/socs/esp32. Acesso em: 6 dez. 2020.

ISIXSIGMA. PROCESS CYCLE EFFICIENCY (PCE). [*S. l.*], 7 nov. 2018. Disponível em: https://www.isixsigma.com/dictionary/process-cycle-efficiency-pce/. Acesso em: 6 dez. 2020.

LUCIDCHART. O que é mapeamento de fluxo de valor?. [*S. l.*], 2017. Disponível em: https://www.lucidchart.com/pages/pt/o-que-e-mapeamento-de-fluxo-de-valor. Acesso em: 3 dez. 2020.

MCKINNEY, Wes. Python para análise de dados: Tratamento de dados com Pandas, NumPy e IPython. 1ª. ed. [*S. l.*]: Novatec, 2018. 616 p.

MONTEIRO , Fernando; PACHECO , Gabriel; LIMA , Lucas. RFID. Universidade Federal do Rio de Janeiro, Departamento de Eletrotécnica - Engenharia de Controle e Automação, 2010. Disponível em: https://athoselectronics.com/esp32/. Acesso em: 6 dez. 2020.

NORTEGUBISIAN. Value Stream Mapping (VSM). [*S. l.*], 24 out. 2018. Disponível em: https://www.nortegubisian.com.br/blog/value-stream-mapping-vsm. Acesso em: 3 dez. 2020.

NTP (Brasil). O NTP. [*S. l.*], 22 jun. 2015. Disponível em: https://ntp.br/ntp.php. Acesso em: 6 dez. 2020.

PYSCIENCE-BRASIL. Python: O que é? Por que usar? Portal Wikidot. Disponível em: http://pyscience-brasil.wikidot.com/python:python-oq-e-pq.[*S. l.*], entre 2013 e 2020. Acesso em: 6 dez. 2020

PYTHON. The Python Tutorial. [*S. l.*], entre 2001 e 2020. Disponível em: https://docs.python.org/3/tutorial/index.html. Acesso em: 3 dez. 2020.

YUAN, Michael. Conhecendo o MQTT: Por que o MQTT é um dos melhores protocolos de rede para a Internet das Coisas?, artigos IBM, 3 out. 2017. Disponível em: https://developer.ibm.com/br/technologies/iot/articles/iot-mqtt-why-good-for-iot/. Acesso em: 6 dez. 2020.