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

ANDRE VITOR SOUZA MATOS

SISTEMA DE IDENTIFICAÇÃO POR RADIOFREQUÊNCIA PARA SUPERVISÃO E CONTROLE DA MANUTENÇÃO DE EQUIPAMENTOS INDUSTRIAIS

Recife
2024

ANDRE VITOR SOUZA MATOS

SISTEMA DE IDENTIFICAÇÃO POR RADIOFREQUÊNCIA PARA SUPERVISÃO E CONTROLE DA MANUTENÇÃO DE EQUIPAMENTOS INDUSTRIAIS

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

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

Coorientador: Prof. Dr. Lauro Rodrigo Gomes da Silva Lourenço Novo

Recife
2024

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

Matos, Andre Vitor Souza.
Sistema de identificação por radiofrequência para supervisão e controle da manutenção de equipamentos industriais / Andre Vitor Souza Matos. - Recife, 2024.
80 p. : il., tab.

Orientador(a): Douglas Contente Pimentel Barbosa
Cooorientador(a): Lauro Rodrigo Gomes da Silva Lourenço Novo
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia de Energia - Bacharelado, 2024.
Inclui referências, apêndices.

1. RFID. 2. Identificação. 3. Microcontrolador. 4. Manutenção. 5. Supervisão. I. Barbosa, Douglas Contente Pimentel. (Orientação). II. Novo, Lauro Rodrigo Gomes da Silva Lourenço. (Coorientação). IV. Título.

620 CDD (22.ed.)

ANDRE VITOR SOUZA MATOS

SISTEMA DE IDENTIFICAÇÃO POR RADIOFREQUÊNCIA PARA SUPERVISÃO E CONTROLE DA MANUTENÇÃO DE EQUIPAMENTOS INDUSTRIAIS

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Aprovado em: 19/03/2024.

BANCA EXAMINADORA

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

_____
Prof. Dr. Lauro Rodrigo Gomes da Silva Lourenço Novo (Coorientador)
Universidade Federal de Pernambuco

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

_____
Profa. M.Sc Camila Mendes Bandeira (Examinador Interno)
Universidade Federal de Pernambuco

Este trabalho é dedicado à memória de Eva Alves de Souza, uma mãe exemplar cuja vida foi dedicada a proporcionar a melhor educação aos seus filhos e a realizar o sonho de vê-los formados.

AGRADECIMENTOS

Agradeço imensamente a toda minha família pelo apoio e incentivo que me proporcionaram ao longo de toda a minha vida. Expresso minha gratidão ao meu pai, Juscelino Miranda de Matos, que se dedicou na minha criação mesmo nos momentos mais difíceis, sempre acreditando e investindo em minha educação e formação. Agradeço também à minha irmã mais velha, Mikaella Matos, pelos ensinamentos, conselhos e apoio incondicional. Também sou grato ao meu irmão, Manoel Neto, por todo o apoio e orientação, sendo não somente meu tutor na vida, mas também meu melhor amigo. E não posso deixar de mencionar minha irmã, Sara Matos, por todo apoio emocional, incentivo e acolhimento, que me permitiu ingressar nesta faculdade.

Agradeço profundamente ao Prof. Dr. Lauro Rodrigo Gomes da Silva Lourenço Novo, que além de atuar como coorientador deste trabalho, proporcionou-me oportunidades únicas ao longo do curso, me orientando não somente nos projetos e monitorias, mas também para meu desenvolvimento pessoal. Expresso também minha gratidão ao Prof. Dr. Douglas Contente Pimentel Barbosa, que aceitou orientar este trabalho com entusiasmo e dedicação.

Sou imensamente grato a todos os meus amigos, especialmente àqueles que conheci durante o curso, por tornarem os dias mais exaustivos e estressantes em momentos mais leves e divertidos. Espero poder levar cada um de vocês comigo ao longo da vida. Além disso, gostaria de expressar minha profunda gratidão a Maria Clara, que esteve ao meu lado nos momentos mais difíceis desta jornada, me apoiando e incentivando em todas as oportunidades, sendo sempre minha melhor amiga.

Agradeço à Universidade Federal de Pernambuco, ao Curso de Engenharia Elétrica e a todos os professores, coordenadores e funcionários que de alguma forma auxiliaram nesta jornada. Serei eternamente grato.

RESUMO

A tecnologia *Radio Frequency Identification* (RFID) é uma forma prática de realizar a identificação e rastreamento de objetos, por meio de sinais de radiofrequência (RF), que são captados por etiquetas que respondem a esses sinais com codificação própria. Os sistemas RFID podem realizar a identificação de um objeto a certa distância de forma rápida, sem a necessidade de diretividade entre este e o dispositivo leitor (*reader*). Este trabalho consiste na implementação de um sistema dedicado para realizar leituras, por meio da tecnologia RFID e de etiquetas de RF, conhecidas como *pit tags,* destinadas ao monitoramento de sistemas de manutenção de equipamentos. Além disso, o referido sistema deve registrar os dados de interesse de cada identificação (ID da *tag* e do leitor, data e horário de leitura) em plataformas disponíveis para o usuário, como computadores e *smartphones*. Estes sistemas são compostos em geral por módulos RFID (RFID-RC522 antenas do tipo *loop*), um dispositivo microcontrolador (ESP32), e alimentação elétrica.

Palavras-chave: RFID; Identificação; Microcontrolador; Manutenção; Supervisão.

ABSTRACT

Radio Frequency Identification (RFID) technology is a practical way of identifying and tracking objects, using radio frequency (RF) signals, that are captured by tags that answer to these signals with their own coding. RFID systems can quickly identify an object at a certain distance, with no need directivity to the reader. This work consists of implementing a dedicated system to track, using RFID technology and RF tags, known as pits tags, intended for monitoring equipment maintenance systems. Furthermore, this system must record the data of interest for each identification (tag and reader ID, date and time of reading) on the user available platforms, such as computers and smartphones. These systems are generally composed of modules RFID modules (RFID-RC522 and loop antennas), a microcontroller device (ESP32), and DC power supply.

Keywords: RFID; Identification; Microcontroller; Maintenance; Supervision.

LISTA DE ILUSTRAÇÕES

Figura 1. Exemplo de sistema RFID.....18
Figura 2. Fotografia do modulo RFID-RC522.....20
Figura 3. Antena do modulo RFID-RC522. 21
Figura 4. Etiqueta RFID Mifare.....22
Figura 5. Diagrama da estrutura da memória de um cartão *Mifare* 1 kB.....23
Figura 6. Gráfico da transmissão serial síncrona da palavra binária 01000011.....24
Figura 7. Comunicação SPI.....25
Figura 8. Fotografia do microcontrolador ESP-WROOM-32.....26
Figura 9. *Pinout* do microcontrolador ESP-WROOM-32. 26
Figura 10. Fotografias do ESP-32 e RFID-RC522 com as interconexões e funções. 30
Figura 11. Fotografias do protótipo. 31
Figura 12. Pseudocódigo do teste de gravação de dados em uma etiqueta.....32
Figura 13. Resultado do teste gravação de dados na etiqueta. 32
Figura 14. Pseudocódigo do teste da leitura de dados em uma etiqueta.....33
Figura 15. Resultado do teste leitura de dados na etiqueta. 33
Figura 16. Importar as bibliotecas, e configurar os módulos e variáveis globais.....35
Figura 17. Pseudocódigo das configurações realizadas na função *setup*.....35
Figura 18. Pseudocódigo da função *loop*.....36
Figura 19. Função Tela Inicial. 38
Figura 20. Função Informar Dados do Operador.....38
Figura 21. Função Definir operação. 39
Figura 22. Função Ler última manutenção. 40
Figura 23. Função Informar dados de manutenção.....41
Figura 24. Função Gravação da manutenção. 42
Figura 25. Função Gerar comprovante. 44
Figura 26. Pseudocódigo da função Ler etiqueta.....44
Figura 27. Pseudocódigo da função Gravar etiqueta. 45
Figura 28. Sequência de telas à função: preenchimento dos dados do operador.....47
Figura 29. Sequência de telas à função: leitura da última manutenção. 49
Figura 30. Sequência de telas à função: gravação da última manutenção. 49

Figura 31. Sequência de telas à função: gravação da última manutenção. 50

LISTA DE TABELAS

Tabela 1. Organização da EEPROM de chips MIFARE. 22

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
CI	Circuito Integrado
EEPROM	Electrically-Erasable Programmable Read-Only Memory
HF	*High Frequency*
LF	*Low Frequency*
PDF	*Portable Document Format*
PWM	*Pulse Width Modulation*
QR	*Quick Response code*
RF	*Radio Frequency*
RFID	*Radio Frequency Identification*
SPI	*Serial Peripheral Interface*
UHF	*Ultra-High Frequency*
VSWR	*Voltage Standing Wave Ratio*

SUMÁRIO

1 INTRODUÇÃO 13

1.1 OBJETIVOS 15

1.1.1 Geral 15

1.1.2 Específicos 15

1.2 ORGANIZAÇÃO DO TRABALHO 15

2 FUNDAMENTAÇÃO TEÓRICA 17

2.1 MANUTENÇÃO EM EQUIPAMENTOS ELÉTRICOS 17

2.2 TECNOLOGIA *RADIO FREQUENCY IDENTIFICATION* 18

2.2.1 Modulo RFID-RC522 20

2.3 MICROCONTROLADORES 25

2.3.1 ESP-32 25

3 DESENVOLVIMENTO DO TRABALHO 28

3.1 FUNCIONAMENTO DO SISTEMA DE MANUTENÇÃO 28

3.2 MONTAGEM E TESTES DO PROTÓTIPO E SUA PROGRAMAÇÃO 30

3.2.1 Testes da conexão do sistema e identificação da etiqueta 31

3.2.1.1 Teste da gravação de dados em uma etiqueta 31

3.2.1.2 Teste da leitura de dados em uma etiqueta 32

3.2.2 Programações desenvolvidas para a operação do sistema 34

3.2.2.1 Configurações e função loop 34

3.2.2.2 Tela Inicial 37

3.2.2.3 Definir operação 39

3.2.2.4 Ler última manutenção: parte gráfica 40

3.2.2.5 Informar dados de manutenção 41

3.2.2.6 Gravação da manutenção: parte gráfica 42

3.2.2.7 Gerar comprovante 43

3.2.2.8 Ler etiqueta 43

3.2.2.9 Gravar etiqueta 45

3.3 TESTE DO SISTEMA COMPLETO 46

4 CONCLUSÕES E PROPOSTAS DE CONTINUIDADE 51

REFERÊNCIAS 52

APÊNDICE A – FLUXOGRAMA DA OPERAÇÃO DO SISTEMA 54

APÊNDICE B – ALGORITMO PARA GRAVAÇÃO DA ETIQUETA 55

APÊNDICE C – ALGORITMO PARA LEITURA DA ETIQUETA 57

APÊNDICE D – COMPROVANTE DE LEITURA DA ÚLTIMA MANUTENÇÃO 59

APÊNDICE E – COMPROVANTE DE GRAVAÇÃO DA MANUTENÇÃO ATUAL 60

APÊNDICE F - ALGORITMO COMPLETO DO SISTEMA DE MANUTENÇÃO 61

1 INTRODUÇÃO

Uma das atividades de grande interesse da área de engenharia é o controle da manutenção realizada sobre os equipamentos elétricos. Pode-se, porém, realizar este controle através da aplicação de sistemas eletrônicos que utilizam plataformas computacionais com a finalidade de identificação individual destes equipamentos, e gerenciamento das informações destes.

Os processos produtivos das empresas estão fortemente dependentes de suas máquinas e equipamentos, para estabelecer e sustentar uma vantagem competitiva sobre seus concorrentes (PIECHNICKI, 2011). A manutenção tem a premissa de garantir a disponibilidade da função dos equipamentos, de modo a atender a produção com segurança, preservação do meio ambiente e custo adequado (KARDEC, 2002).

Os dispositivos RFID possibilitam a identificação rápida e precisa de suas etiquetas, as quais carregam identificações únicas. Além disso, alguns modelos permitem a gravação de informações em sua memória interna. Quando conectados a *smartphones* e computadores, esses dispositivos oferecem um poderoso recurso de rastreamento e identificação. Portanto, representa uma opção tecnológica para equipes de manutenção de equipamentos industriais.

A origem da tecnologia de *Radio Frequency Identification* (RFID) esteve inicialmente ligada a fins militares, mais precisamente aos sistemas de radares utilizados na Segunda Guerra Mundial. Em 1973, começaram a ser solicitados os primeiros pedidos de patente para produtos que utilizavam essa tecnologia. Posteriormente, surgiram organizações com o objetivo de padronizar a utilização da tecnologia de RFID no mercado. A partir da década de 90, o avanço da eletrônica e o desenvolvimento de componentes em larga escala estimularam o surgimento de diversas aplicações comerciais utilizando RFID (ROBERTI e VIOLINO, 2005).

Devido à sua estrutura simples, à variedade de formatos de componentes e ao rápido processo de identificação, a tecnologia de RFID revolucionou a indústria. Outras tecnologias de identificação, como o código de barras e o *Quick Response* (QR) *code*, necessitam que os itens sejam identificados individualmente a uma curta distância. Dependendo da quantidade de objetos, esses processos de identificação podem ser lentos e podem envolver várias pessoas. Por outro lado, com o RFID, *tags*

(etiquetas) podem ser identificadas simultaneamente e a distâncias maiores, reduzindo custos de pessoal e tempo de operação.

Os sistemas de RFID podem utilizar diferentes frequências, em faixas liberadas para a comunicação entre seus principais componentes: a etiqueta (*tag*) e o leitor. A escolha da frequência dependerá da aplicação, e considerará o custo e as propriedades eletromagnéticas da faixa utilizada, tais como: alcance, desempenho próximo a líquidos e metais, e a taxa de transmissão (ROGER, 2023), (NETO, 2010), (LOUREIRO, SOUZA e LOPES, 2015).

A *tag* é o dispositivo responsável por armazenar informações do objeto no qual está anexada. O leitor alimenta a *tag*, recebe as informações enviadas por ela e as comunica ao controlador. O controlador geralmente é um computador no qual está sendo executada uma aplicação com um banco de dados e um *software* de controle. O leitor, portanto, se destaca por centralizar a comunicação, sendo um intermediador entre as *tags* e o controlador (NETO, 2010).

Devido a características como confiabilidade, facilidade de operação, e dimensões reduzidas, os dispositivos de RFID, possuem, portanto, a tecnologia escolhida para aplicação neste Trabalho de Conclusão de Curso (TCC).

O sistema de RFID escolhido é acessado por quaisquer dispositivos que possuam conexão com a rede Wi-Fi gerada pelo microcontrolador interno (ESP-32), e permite gravar dados na memória interna de cada etiqueta, a qual é vinculada a cada equipamento individualmente, permitindo a extração das informações da última manutenção realizada, e a inserção das informações referentes à manutenção, dentre elas: o tipo, data/horário e o responsável pela manutenção. Para a comunicação com o usuário, o sistema disponibiliza interface gráfica, e ao final de cada acesso gera os comprovantes de manutenção no formato *Portable Document Format* (PDF).

1.1 OBJETIVOS

1.1.1 Geral

Este Trabalho de Conclusão de Curso tem como objetivo principal a realização de um sistema eletrônico de identificação de equipamentos industriais e seu controle de manutenção, utilizando a tecnologia RFID e auxiliado por computador portátil, como *notebook*, *tablet ou smartphone*.

1.1.2 Específicos

O presente Trabalho possui os seguintes objetivos específicos:

- Realizar um embasamento teórico sobre os conceitos e as tecnologias utilizadas no desenvolvimento do sistema;
- Montagem e programação de um sistema de controle de manutenção que seja capaz de:
    - Gravar informações sobre a manutenção atual na etiqueta vinculado ao equipamento;
    - Ler informações da última manutenção disponível na etiqueta vinculado ao equipamento;
    - Permitir a geração do comprovante de leitura e de gravação das etiquetas, salvos em formato PDF, no dispositivo conectado ao sistema.

1.2 ORGANIZAÇÃO DO TRABALHO

O trabalho está organizado, a partir do seu Capítulo 2, como se segue:

- No Capitulo 2 se encontra a fundamentação teórica, no qual são apresentados os principais conceitos e tecnologias utilizados no desenvolvimento do trabalho;

- No Capitulo 3 é discorrido o desenvolvimento do sistema, apresentando o algoritmo criado, a montagem do sistema, os testes realizados e os resultados adquiridos;
- No Capitulo 4 é feita a conclusão deste Trabalho, com a apresentação dos resultados obtidos e sua confrontação com os objetivos previstos, além de sugestões de melhorias e avanços que podem ser realizados no futuro.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são apresentados os fundamentos teóricos dos conteúdos de interesse deste trabalho, os quais abrangem: 1) Manutenção em equipamentos elétricos: ênfase nos tipos de manutenção em equipamentos industriais, manutenção corretiva, preventiva, preditiva e prescritiva; 2) Tecnologia *Radio Frequency Identification*: configuração básica, frequências de operação e o módulo leitor RC-522; 3) Microcontroladores: definição, características e o ESP-32.

2.1 MANUTENÇÃO EM EQUIPAMENTOS ELÉTRICOS

A manutenção em equipamentos elétricos é de fundamental importância para o funcionamento e lucro de uma indústria. A falta de manutenção pode causar uma série de imprevistos ao longo do tempo, como a interrupção de processos, aumento nos custos operacionais, diminuição da vida útil dos equipamentos, além dos riscos de acidentes que podem envolver os colaboradores a danos físicos ou até risco de vida (SANTOS, 2021).

De acordo com a Associação Brasileira de Normas Técnicas (ABNT), a manutenção é a combinação de todas as ações técnicas e administrativas, incluindo as de supervisão, destinadas a manter ou recolocar um item em um estado no qual possa desempenhar uma função requerida (SOARES, 2023).

Existem quatro tipos de manutenção que uma indústria pode realizar em seus equipamentos, os quais são apresentados a seguir (SOARES, 2023):

1) Manutenção corretiva: Ocorre no momento seguinte à identificação de um problema no equipamento, tendo o objetivo de devolver ao equipamento as condições de funcionamento. A manutenção corretiva é normalmente a mais onerosa, e pode ser responsável pela paralisação do equipamento/indústria;
2) Manutenção preventiva: Ocorre periodicamente para garantir a eficiência e a segurança do equipamento, preventivamente à ocorrência das falhas. Revisões periódicas, lubrificações, calibrações e vistorias são exemplos deste tipo de manutenção;

3) Manutenção preditiva: Ocorre para predizer a condição atual do equipamento por meio de monitoramento de rotina realizado, por exemplo, em tempo real, e que normalmente se utiliza de testes e medições de parâmetros físicos;

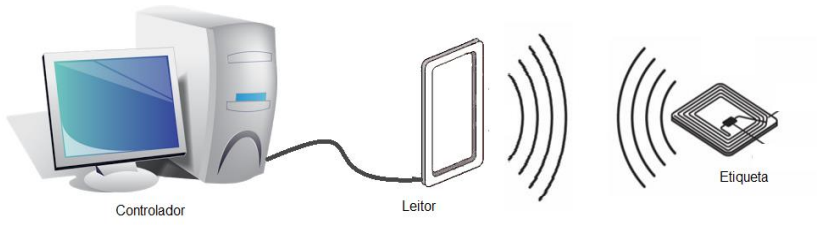
4) Manutenção prescritiva: Surgiu com a era da indústria 4.0 utilizando a inteligência artificial para identificar falhas potenciais. Mais do que apenas prever o que vai acontecer, esse tipo de monitoramento também sugere reparos baseados nos dados coletados.

2.2 TECNOLOGIA *RADIO FREQUENCY IDENTIFICATION*

A Identificação via Rádio Frequência (RFID) é uma tecnologia utilizada para identificar, rastrear, gerenciar e catalogar produtos diversos, documentos, animais e até mesmo pessoas, sem a necessidade de contato físico ou um campo visual. Algumas de suas principais aplicações é a identificação de animais domésticos e destinados à agropecuária, por exemplo (NETO, 2010).

O sistema de RFID requer um módulo leitor RF, composto por antena(s), *microchips* (*pit tags*/chaveiros/cartões), e uma plataforma eletrônica (microcomputadores/ microcontroladores). A comunicação entre o módulo RFID e as etiquetas, é dada por transmissão serial com Modulação por Largura de Pulso (do inglês *Pulse Width Modulation* - PWM). A *pit tag* possui um Circuito Integrado (CI) que é acionado ao receber o sinal eletromagnético operando na frequência de interesse para a identificação desta. A Figura 1 apresenta um sistema de etiquetagem controlada por RFID, e seus elementos básicos, dentre eles: o controlado (computador), leitor RFID, e a etiqueta (indivíduo, *tag*).

Figura 1. Exemplo de sistema RFID.



Fonte: (LOUREIRO, SOUZA e LOPES, 2015)

Para se comunicar com as *tags*, o leitor utiliza uma antena (ou um arranjo de antenas) que é fabricada em diversos tamanhos e formatos, e possui parâmetros que devem garantir que ela seja a mais adequada à aplicação. Alguns dos principais parâmetros dessas antenas são: o ganho, a polarização, o diagrama de radiação, a impedância de entrada, a faixa de frequências, e o *Voltage Standing Wave Ratio* (VSWR). O ganho determina a capacidade da antena de transmitir em uma determinada direção. A polarização é a direção do vetor campo elétrico da onda eletromagnética associada à antena. O diagrama de radiação é uma representação gráfica da energia transmitida pela antena. A impedância de entrada é a impedância que a antena apresenta em sua entrada de alimentação. A faixa de frequências é o conjunto de frequências para o qual a antena irradia dentro de suas características de projeto. Por fim, o VSWR é um indicador da quantidade de sinal refletido no transmissor dado pelo parâmetro de tensão elétrica. Os valores dos parâmetros elétricos da antena dependem das características constitutivas do condutor elétrico: comprimento, diâmetro e material de fabricação (BALANIS, 2005).

Diversos tipos de módulos e aparelhos operantes em frequências distintas podem ser destinados a diferentes tarefas de identificação (ROGER, 2023). Os módulos leitores de RF são utilizados para identificação de proximidades, porém podem-se manipular esses equipamentos para alcances de leitura mais elevados. O sistema de RFID possui o seguinte funcionamento básico: o aparelho leitor de *tags* RFID envia através de uma antena, sinais de radiofrequência, e assim, a etiqueta atingida pela radiação é então acoplada eletromagneticamente à antena do leitor, e seus dados são armazenados neste (LOUREIRO, SOUZA e LOPES, 2015).

As faixas de frequências utilizadas em RFID são: *Low Frequency* (LF), *High Frequency* (HF) e a *Ultra High Frequency* (UHF) (ROGER, 2023).

A região de *Low Frequency* trabalha entre 100 kHz e 300 kHz, em que a mais comumente utilizada é a frequência de 125 kHz. Suas principais características são os baixos valores de: raio de alcance de leitura, custo, velocidade de leitura, e interferência a ondas de rádio. Adicionalmente, possui alta penetração em materiais metálicos e líquidos (ROGER, 2023).

A faixa *High Frequency* está entre 3 MHz e 30 MHz, principalmente a frequência de 13,56 MHz, e todas suas características são intermediárias entre o *Low Frequency*, e Ultra High Frequency (ROGER, 2023).

A *Ultra High Frequency* trabalha na faixa de 300 MHz a 3 GHz, principalmente entre 900-915 MHz, e oferece longo alcance e alta velocidade de leitura, mas possui baixa penetração em líquidos e metais e está sujeita à interferência de ondas de rádio (ROGER, 2023).

2.2.1 Modulo RFID-RC522

O módulo RC522 é um CI capaz de ler e escrever em *pit tags* segundo a norma ISO/IEC 14443, em uma frequência nominal de 13,56 MHz (GUIMARÃES, 2021). A Figura 2 apresenta uma fotografia do módulo RFID-RC522 e de duas *pits tags*, uma em forma de chaveiro, e outra em forma de cartão.

Figura 2. Fotografia do modulo RFID-RC522.

Fonte: (GUIMARÃES, 2021).

As características principais do módulo RC522 são:

- Alimentação típica: 3,3V;
- Interface SPI: Velocidade de até 10Mbits/s;

- Interface I2C: Pode operar no “*Fast mode*”(400 kbits/s) ou no modo “*High-speed*” (3400 kbits/s);
- Interface UART: Velocidade de até 1228,8 kbits/s;
- Consumo de corrente:
    - Operação normal: 13 a 26mA;
    - Modo baixo consumo: 10µA.
- Alcance de 3 cm em média.

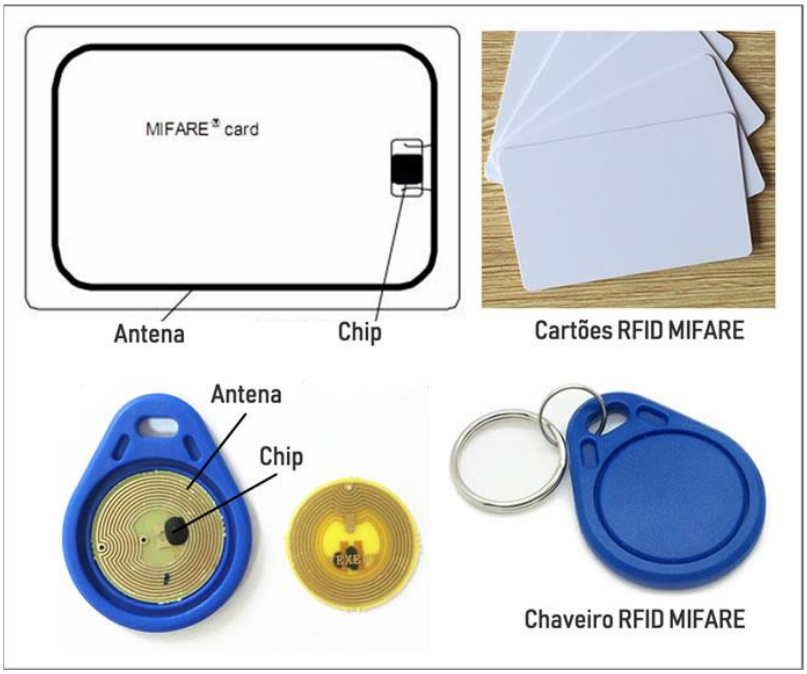
A antena do leitor já vem integrada ao CI do módulo RC522, por meio de trilhas na placa de circuito impresso, como apresentado na Figura 3 (GUIMARÃES, 2021).

As etiquetas da mesma fabricante deste módulo, apresentadas na Figura 4, permitem não somente a leitura de um código único, mas também permite gravar dados em sua memória interna, do tipo *Electrically-Erasable Programmable Read-Only Memory* (EEPROM), que permanecerão na etiqueta mesmo sem alimentação elétrica constante.

Figura 3. Antena do modulo RFID-RC522.

Fonte: (GUIMARÃES, 2021)

Figura 4. Etiqueta RFID Mifare.



Fonte: (SIMÕES, 2019).

As capacidades de memória EEPROM mais comuns dos sistemas MIFARE são de 1, 2 e 4 kBytes, e estas possuem organização própria, como apresentado na Tabela 1 (SIMÕES, 2019).

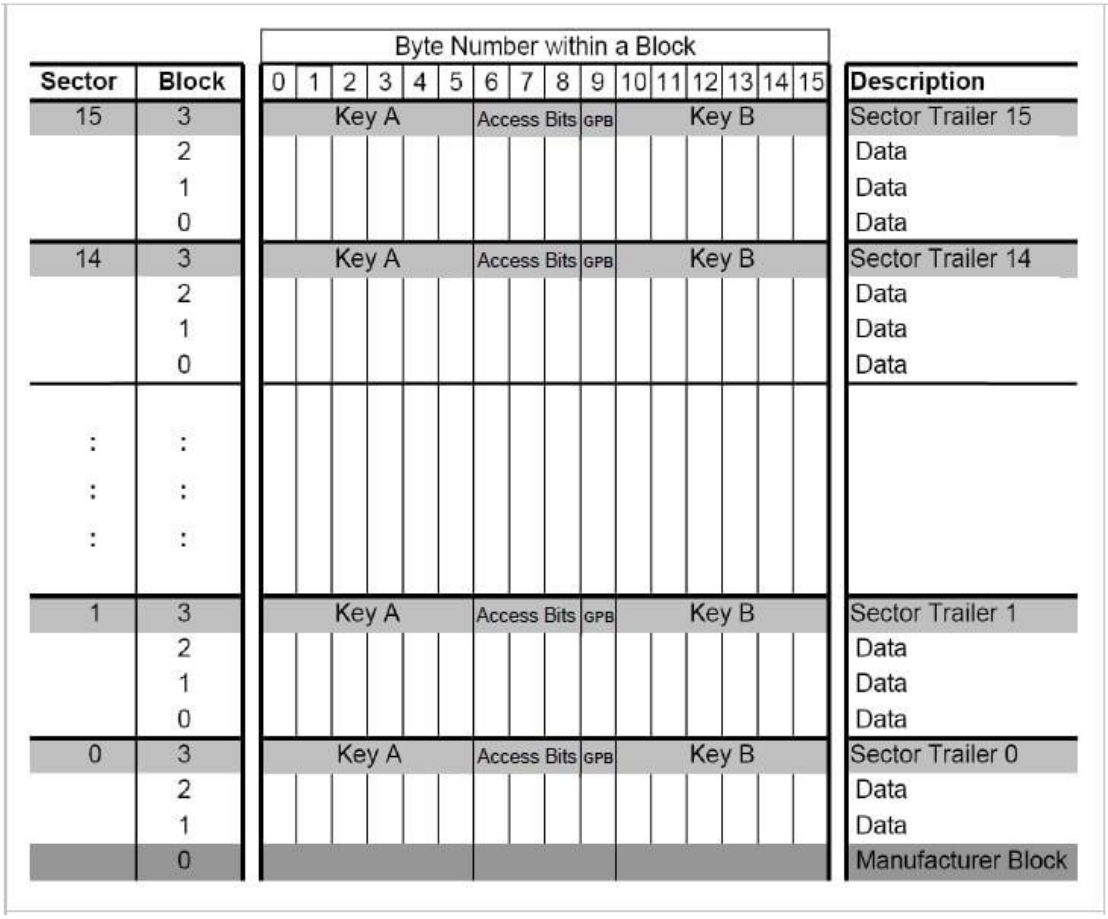
Tabela 1. Organização da EEPROM de chips MIFARE.

Capacidade de armazenamento de dados (kBytes)	Quantidade de setores	Quantidade de blocos por setor
1	16	4
2	32	4
4	40	4 ou 16

Fonte: Adaptada de (SIMÕES, 2019).

As memórias são divididas em setores, e estes contêm 4 blocos de 16 bytes de memória cada. A estrutura da memória é dividida em: "*Manufacture Block"* (bloco do fabricante), "*Data Block"* (Bloco de dados) e "*Sector Trailer*" (SIMÕES, 2019). A Figura 5 mostra um diagrama da estrutura interna da memória de um cartão *Mifare* de 1 kB, destacando os blocos mencionados anteriormente.

Figura 5. Diagrama da estrutura da memória de um cartão *Mifare* 1 kB.



Fonte: (ALVES, 2017)

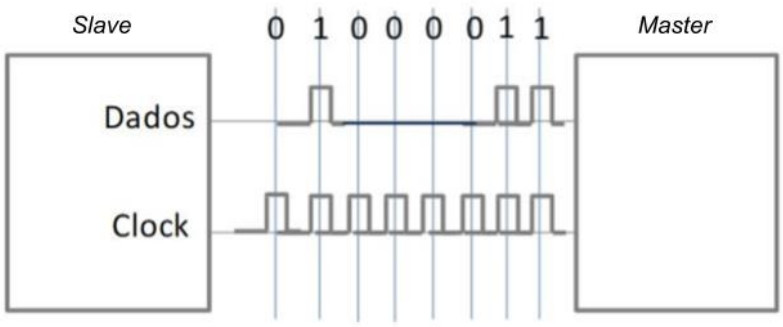
O “*Manufacturer Block*” é o primeiro bloco do primeiro setor, reservado para armazenar informações do fabricante. Os três primeiros blocos de cada setor (0, 1 e 2) são para armazenamento de dados (exceto o bloco 0 do setor 0), podendo ser configurados como “*read/write blocks*” ou “*value blocks*”. Os “*read/write blocks*” são blocos para leitura e escrita de dados. Já os “*value blocks*” foram destinados ao uso em sistemas que envolvem contagem de crédito, pois também oferecem as funções de incremento e decremento de valores, além de restauração e transferência. O último bloco de cada setor é chamado de “*Sector Trailer*” e armazena as chaves de acesso aos demais blocos daquele setor, além das condições de acesso de cada bloco daquele setor (SIMÕES, 2019).

A comunicação do módulo RFID-RC522 utiliza o protocolo SPI (*Serial Peripheral Interface*), que é um protocolo de comunicação síncrona, e o conceito de *master/slave*. O *Master* é o dispositivo controlador, enquanto o *Slave* é o seu leitor controlado. Este

protocolo permite comunicações com velocidades de até 10 Mbps (Megabits por segundo) (SIMÕES, 2019).

A Figura 6 apresenta um gráfico com o sincronismo dado pelo sinal de *clock* da comunicação serial que utiliza o protocolo SPI, por exemplo entre *Master* e *Slave*.

Figura 6. Gráfico da transmissão serial síncrona da palavra binária 01000011.



Fonte: Adaptado de (SIMÕES, 2019)

Os sinais presentes na comunicação SPI entre *Master* e *Slave* são apresentados na Figura 7, e são responsáveis pelas funções descritas a seguir:

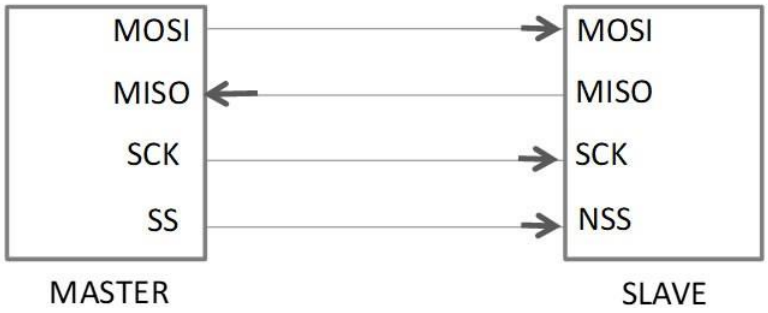
- MOSI (*MASTER OUTPUT/SLAVE INPUT*): Envio dos dados do *Master* para o *Slave*;
- MISO (*MASTER INPUT/SLAVE OUTPUT*): Envio de dados do *Slave* para o *Master*;
- SCK (*SERIAL CLOCK*): Sinal de *Clock* da comunicação serial;
- SS ou CS (*SLAVE SELECT* ou *CHIP SELECT*): Escolha do *Master* para selecionar o dispositivo *Slave.*

De forma resumida, os passos da comunicação SPI são os seguintes:

- O *Master* envia o sinal de *Clock*;
- O *Master* coloca a conexão SS (SDA no RC522) em 0V para ativar a seleção do *Slave* (RC522);

- O *Master* envia dados (sequência de bits) através da conexão MOSI e o *Slave* lê os bits transmitidos;
- Se uma resposta é necessária, o *Slave* retorna dados para o Master através da conexão MISO e o *Master* lê os bits transmitidos.

Figura 7. Comunicação SPI.



Fonte: Adaptado de (SIMÕES, 2019)

2.3 MICROCONTROLADORES

Um microcontrolador consiste em um único circuito integrado que reúne um núcleo de processador, memórias voláteis e não voláteis, e diversos periféricos de entrada e saída de dados que têm poder de serem programados (IEEE RAS UFCG, 2020). O uso mais comum para microcontroladores costuma ser em sistemas embarcados, pois são capazes de produzir uma sequência de tarefas preestabelecidas, que são controladas por estes dispositivos. As maiores vantagens deste equipamento são o baixo custo de projeto e um consumo mínimo de energia (MELLO, 2022).

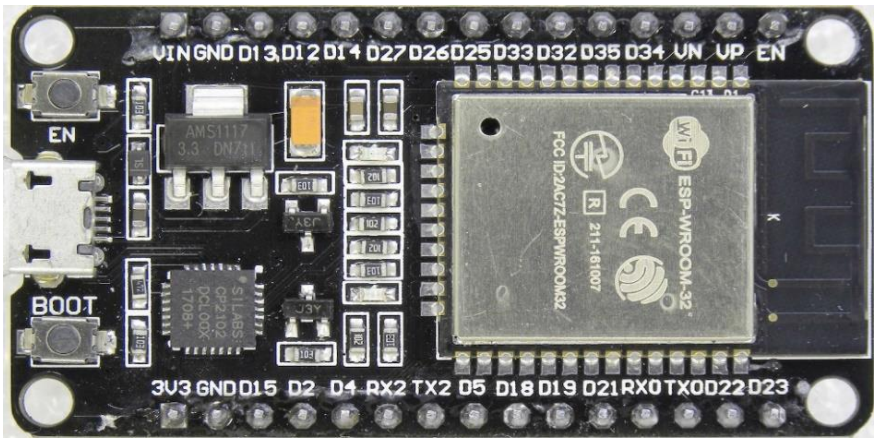
2.3.1 ESP-32

O ESP 32 é uma plataforma eletrônica concorrente ao Arduino, produzido pela empresa *Espressif Systems*, a qual possui um processador *Tensilica Xtensa LX6 dual core*, com até 240 MHz de velocidade, e módulos WiFi e *Bluetooth Low Energy*. Sua

programação pode ser feita na mesma linguagem e *software* que qualquer Arduino (LOBODAROBOTICA, 2021).

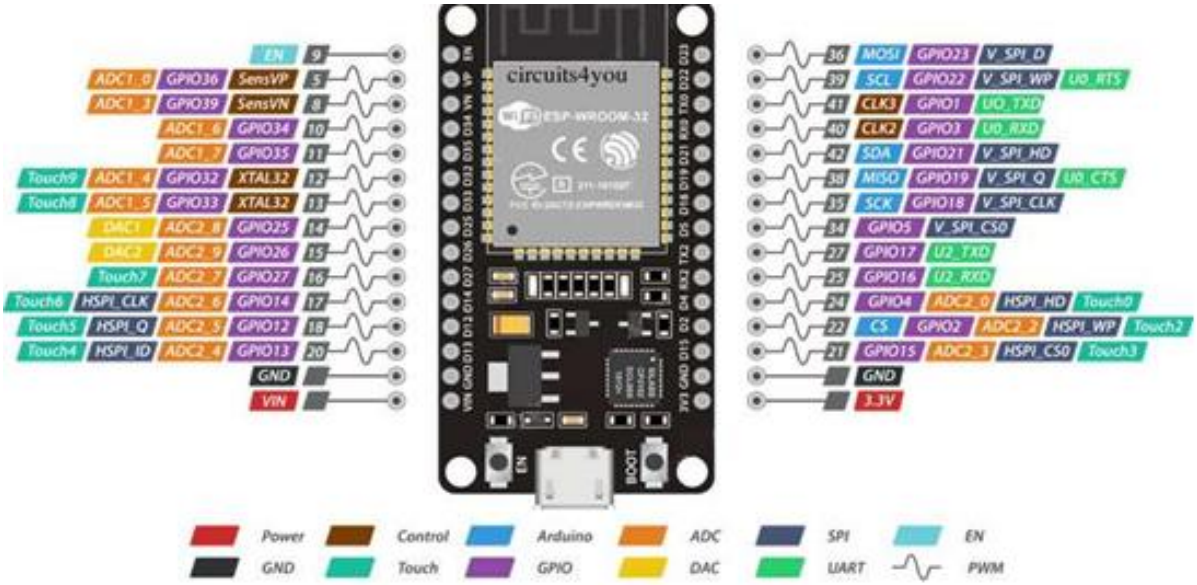
A Figura 8 e a Figura 9 exibem a fotografia do ESP-WROOM-32, e a descrição da *pinout* do ESP-WROOM-32, respectivamente.

Figura 8. Fotografia do microcontrolador ESP-WROOM-32.



Fonte: (THAKUR, 2018)

Figura 9. *Pinout* do microcontrolador ESP-WROOM-32.



Fonte: (THAKUR, 2018)

As características principais do ESP-WROOM-32 são descritas a seguir:

- Memória ROM interna de 448 kBytes;
- Memória RAM estática interna de 520 kBytes;
- Memórias externas: 4x (16MB @ *Flash* e SRAM);
- Aceleradores de *hardware*: AES, SHA, RSA, ECC;
- *Real Time Clock* com 16 kB de SRAM;
- WIFI 802.11 b/g/n (2.4 GHz), até 150 Mbps;
- *Bluetooth* v4.2 BR/EDR e LE (*low energy*);
- Dois grupos de *Timers* – 4 *timers* de 64 Bits;
- 1 kB de Fusíveis eletrônicos (segurança);
- Alimentação de 2,3 V a 3,6 V, e corrente 0,24 A;
- 34 × Portas programáveis GPIOs;
- Conversores ADC SAR 12-bits e DAC de 8-bits;
- 10 sensores de toque, e sensor de Temperatura;
- 4 interfaces SPI e 2 I2S – clock até 40 MHz;
- 2 interfaces I2C, e 3 seriais UART – até 5 Mbps;
- 1 *Host* (SD/eMMC/SDIO)-controle de SD *Card*

3 DESENVOLVIMENTO DO TRABALHO

Neste capítulo, descreve-se o processo de desenvolvimento e construção do sistema de manutenção, abrangendo desde sua concepção, suas funcionalidades, seus testes preliminares e de validação. As atividades desenvolvidas envolveram a montagem do protótipo funcional, seus testes básicos de leitura e escrita em etiquetas, a programação das funções do sistema e, por fim, o teste de validação do sistema completo, onde é realizada a simulação da leitura da última manutenção realizada e a gravação de uma nova manutenção.O desenvolvimento do sistema ocorreu em etapas, previstas no fluxograma disponível no APÊNDICE A, o qual foi utilizado para o desenvolvimento do código completo do sistema apresentado no APÊNDICE F.

3.1 FUNCIONAMENTO DO SISTEMA DE MANUTENÇÃO

O sistema desenvolvido tem como objetivo permitir que o operador ou supervisor de uma indústria realize o controle de manutenção dos equipamentos por meio de sua identificação por etiquetas eletrônicas, colocadas na parte externa ou próxima ao equipamento, onde ficarão disponíveis os dados da última manutenção para leitura e para gravação de novas informações.

Para acesso ao sistema, o microcontrolador ESP-32 utilizado deve gerar uma rede Wi-Fi, onde o dispositivo do operador ou supervisor deva se conectar. Por meio do navegador do dispositivo, o usuário do sistema deve se conectar ao IP: 192.168.4.1 e acessar a interface do sistema, disponibilizada pelo ESP-32.

O fluxograma do sistema, a partir da conexão do usuário, é apresentado no APÊNDICE A. As principais funções disponíveis são a de ler os dados da última manutenção e a de gravar os respectivos dados da manutenção atual. Para qualquer tipo de operação é necessário que o usuário inicie informando ao sistema o seu nome e sua matrícula na empresa, tornando toda operação realizada rastreável.

Ao selecionar a opção para ler a última manutenção de um equipamento, é necessário que a antena do sistema seja aproximada da etiqueta vinculada a aquele equipamento. A partir da detecção e da leitura da etiqueta, as informações presentes

sobre a última manutenção devem ser apresentadas ao usuário, juntamente com a identificação da etiqueta e os dados do usuário que realizou a leitura, com a opção de salvar estas informações em comprovante no formato PDF no próprio dispositivo.

Na opção para gravação de nova manutenção, afim de que não sejam perdidas as informações presentes antes da nova gravação, é realizada de forma obrigatória a função de leitura da etiqueta antes do preenchimento dos novos dados. Após a leitura, o usuário deve preencher um formulário com as informações da nova manutenção, e após salvar essas informações, a antena do sistema deve ser aproximada da etiqueta, realizando a gravação na sua memória interna. No fim da operação será gerado um comprovante em PDF contendo as antigas e novas informações da manutenção, com a opção de salvá-lo no próprio dispositivo.

As informações gravadas na etiqueta após cada manutenção são:

- Nome do operador que realizou a manutenção;
- Matrícula do operador que realizou a manutenção;
- Tipo de manutenção realizada;
- Código dos itens que foram substituídos durante a manutenção;
- Código dos itens que continham avaria no equipamento;
- Data e hora que a manutenção foi realizada.

As informações são salvas na memória dinamicamente, ou seja, não há espaços específicos dentro da memória interna para gravação de cada informação, a qual é feita sequencialmente (serial) na etiqueta, com as informações separadas internamente pelo caractere ‘#’, o qual não é apresentado/disponibilizado ao usuário para evitar erros de interpretação dos dados pela linguagem interna do dispositivo.

As informações salvas na etiqueta devem possuir valor máximo de 752 *bytes*, verificado para evitar que o usuário tente salvar mais informações que o permitido.

Devido à possibilidade de registro de observações pertinentes à manutenção e que não possam ser gravadas na etiqueta, o formulário de manutenção apresenta um espaço de observação apresentado no comprovante PDF.

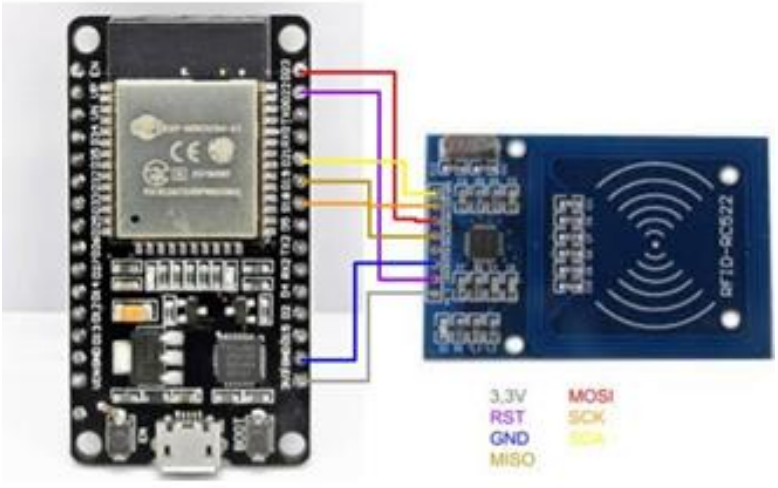
3.2 MONTAGEM E TESTES DO PROTÓTIPO E SUA PROGRAMAÇÃO

Os dispositivos previstos para o desenvolvimento do sistema proposto precisam ser previamente montados e testados para familiarização de suas funcionalidades com vistas à aplicação requerida. A partir daí, a programação da operação do referido sistema deve ser desenvolvida no microcontrolador utilizado para a realização do desejado controle da manutenção dos equipamentos.

A Figura 10 apresenta fotografia do microcontrolador ESP-32 e o leitor RFID-RC522, com indicações de suas interconexões. A Figura 11 exibe fotografias do sistema montado em laboratório, e da etiqueta utilizada nos testes.

A programação do ESP-32 foi desenvolvida no *software* Arduino IDE de acordo com o funcionamento do sistema, onde são utilizadas as bibliotecas de comunicação com os módulos RFID-RC522 e o Wi-Fi nativo do ESP-32, o qual é configurado para gerar uma rede administrada pelo próprio microcontrolador com as credenciais “Sistema de manutenção” e senha “123456789”. O ESP-32 disponibiliza no IP 192.168.4.1 a interface de usuário, desenvolvidas em HTML, CSS e Javascript. A alimentação do sistema RFID é de 5 V, fornecida por um circuito eletrônico regulador de tensão, alimentado pela placa de expansão do ESP-32 e pilha de 9 V.

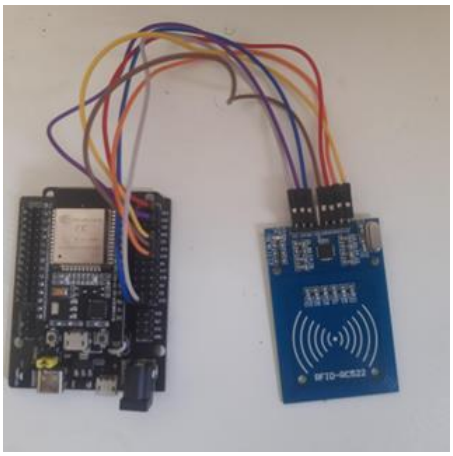
Figura 10. Fotografias do ESP-32 e RFID-RC522 com as interconexões e funções.



Fonte: Adaptado de (THAKUR, 2018) e (GUIMARÃES, 2021).

Figura 11. Fotografias do protótipo.

(a) ESP-32 e RFID-RC522 interconectados.



(b) Etiquetas (*tags*) utilizadas.

Fonte: O Autor.

3.2.1 Testes da conexão do sistema e identificação da etiqueta

Para o desenvolvimento da programação dos algoritmos do sistema foi realizada a separação dos códigos das duas principais funcionalidades, realizando testes para as funções de leitura dos dados e de gravação de dados numa etiqueta.

3.2.1.1 Teste da gravação de dados em uma etiqueta

O teste de gravação de dados em uma etiqueta tem como objetivo preencher um bloco da memória com a informação “Mensagemgravada”.

Figura 12 apresenta o pseudocódigo do algoritmo desenvolvido para os testes de gravação de dados numa etiqueta (Algoritmo 1), o qual é apresentado no APÊNDICE B.

Ao aproximar a etiqueta da antena, o resultado é obtido pelo monitor serial do *software* Arduino IDE (Figura 13), onde é possível visualizar o ID da etiqueta, seu tipo e a mensagem de confirmação da gravação. Para a gravação em vários blocos é necessária a mudança gradual do bloco de escrita enquanto houver informações a serem armazenadas, evitando assim os blocos específicos de configuração do setor.

Figura 12. Pseudocódigo do teste de gravação de dados em uma etiqueta.

```
01: Importe as bibliotecas MFRC522.h e SPI.h.
02: Defina as portas utilizadas para conexão com o RC522.
03: Configure o módulo RC522.
04: Grave a mensagem “Mensagem gravada” em uma variável.
Início da função setup():
05:    Inicie a comunicação Serial.
06:    Inicie a comunicação SPI.
07:    Inicie o módulo RC522.
Fim da função setup().
Início da função loop():
08:    Se for detectado uma nova etiqueta e é possível ler então:
09:       Apresente as informações da etiqueta na comunicação Serial.
10:       Configure variáveis de gravação.
11:       Se a autenticação não apresentar erro então:
12:          Grave a mensagem na etiqueta.
13:          Se a gravação não apresentar erro então:
14:             Apresente a mensagem de confirmação na comunicação Serial.
15:          Senão:
16:             Apresente a mensagem de erro na comunicação Serial.
17:       Senão:
18:          Apresente a mensagem de erro na comunicação Serial.
19:    Reinicie a comunicação com o leitor RC522.
Fim da função loop().
```

Fonte: O Autor.

Figura 13. Resultado do teste gravação de dados na etiqueta.

```
Card UID: 13 3A D7 21
Card SAK: 08
PICC type: MIFARE 1KB
MIFARE_Write () success:
```

Fonte: O Autor.

3.2.1.2 Teste da leitura de dados em uma etiqueta

O teste de leitura de dados em uma etiqueta tem como objetivo realizar a leitura das informações presentes em um bloco da memória. A Figura 14 apresenta o pseudocódigo do algoritmo desenvolvido para os testes de leitura de dados numa etiqueta (Algoritmo 2), o qual é apresentado no APÊNDICE C.

A partir do teste apresentado na seção 3.2.1.1, espera-se a leitura da informação “Mensagemgravada” na etiqueta. Ao aproximá-la da antena, o resultado foi apresentado pelo monitor Serial do *software* Arduino IDE (Figura 15), condizente com o esperado para as informações de ID da etiqueta, seu tipo e mensagem lida.

Figura 14. Pseudocódigo do teste da leitura de dados em uma etiqueta

```
01: Importe as bibliotecas MFRC522.h e SPI.h.
02: Defina as portas utilizadas para conexão com o RC522.
03: Configure o módulo RC522.
Início da função setup():
04:    Inicie a comunicação Serial.
05:    Inicie a comunicação SPI.
06:    Inicie o módulo RC522.
Fim da função setup().
Início da função loop():
07:    Se for detectado uma nova etiqueta e é possível ler então:
08:       Apresente as informações da etiqueta na comunicação Serial.
09:       Configure variáveis de leitura.
10:       Se a autenticação não apresentar erro então:
11:          Leia a mensagem presente na etiqueta.
12:          Se a leitura não apresentar erro então:
13:             Apresente a mensagem de confirmação na comunicação Serial.
14:             Apresente as informações lidas
15:          Senão:
16:             Apresente a mensagem de erro na comunicação Serial.
17:       Senão:
18:          Apresente a mensagem de erro na comunicação Serial.
19:    Reinicie a comunicação com o leitor RC522.
Fim da função loop().
```

Fonte: O Autor.

Figura 15. Resultado do teste leitura de dados na etiqueta.

```
Card UID: 13 3A D7 21
Card SAK: 08
PICC type: MIFARE 1KB

Dados bloco [1]: Mensagemgravada
```

Fonte: O Autor.

3.2.2 Programações desenvolvidas para a operação do sistema

O algoritmo completo da programação do sistema de manutenção, apresentado no APÊNDICE F, baseia-se no fluxograma desenvolvido para a operação deste, o qual é apresentado no APÊNDICE A, cujas funções/blocos referem-se a:

- Tela Inicial: Permite que o usuário selecione entre o preenchimento dos dados do operador e o menu “Definir operação”, sendo este último disponível somente com a entrada de dados cadastrais válidos do operador;
- Informar dados do operador: Permite ao usuário preencher os dados do operador por meio de um formulário e enviá-los ao sistema;
- Definir Operação: Permite ao usuário selecionar o tipo de operação que se deseja realizar;
- Ler etiquetas: Realiza a leitura dos dados presentes na memória de uma etiqueta;
- Informar dados de manutenção: Permite ao usuário preencher os dados da manutenção para gravação na etiqueta;
- Gravar etiqueta: Realiza a gravação dos dados da manutenção atual enviados pelo operador;
- Gerar comprovante de leitura da manutenção: Gera um comprovante de leitura, permitindo ao usuário armazenar em seu dispositivo;
- Gerar comprovante de gravação da manutenção: Gera um comprovante de gravação, permitindo ao usuário armazenar em seu dispositivo.

3.2.2.1 Configurações e função loop

Inicialmente, deve ser realizada a configuração do ESP-32 por meio das importações das bibliotecas, definições de portas, inicialização dos módulos e criação de variáveis globais.

A Figura 16 apresenta o pseudocódigo (Algoritmo 3) das configurações de entrada do sistema, tais como: Importação de bibliotecas, configuração de módulos e variáveis globais. As bibliotecas utilizadas são as necessárias para comunicação e

configuração do WiFi como *Access Point*, conexão com o módulo RC522 e para comunicação via protocolo SPI. Após a importação destas bibliotecas, é realizada a configuração da conexão com o módulo RC522, onde são definidas as portas utilizadas, e a comunicação Wi-Fi configurando credenciais e modo de comunicação.

Figura 16. Importar as bibliotecas, e configurar os módulos e variáveis globais.

```
1:Importe as bibliotecas: WiFi.h; WiFiAP e SPI.h.
2:Defina dos pinos utilizados e constantes
3:Configure do módulo RFID-RC522
4:Defina das credenciais do WiFi
5:Crie as variáveis globais responsáveis pela verificação do preenchimento dos dados e
armazenar os dados recebidos
```

Fonte: O Autor.

A função *setup* é a primeira a ser executada de forma automática pelo ESP-32, e é responsável por iniciar as configurações do programa. A Figura 17 apresenta o pseudocódigo (Algoritmo 4) da função *setup* utilizada no programa, onde é realizada: A configuração da comunicação WiFi, definindo seu o IP, *gateway* e *subnet*, além da inicialização da rede WiFi como *Access Point*; A inicialização da comunicação via protocolo SPI e a inicialização do módulo RC522.

Figura 17. Pseudocódigo das configurações realizadas na função *setup*.

```
1:Definir o IP estático, gatway e subnet.
2:Configurar e iniciar a rede WiFi.
3:Iniciar comunicação SPI.
4:Iniciar módulo RC522.
```

Fonte: O Autor.

A função loop é a função principal do ESP-32, enquanto o dispositivo estiver em funcionamento esta função deve ser executada infinitamente. A Figura 18 apresenta o pseudocódigo (Algoritmo 5) da função loop utilizada nesta aplicação, com o principal objetivo realizar a dinâmica entre as funções e a aquisição de dados.

Figura 18. Pseudocódigo da função *loop*.

```
01: Cria objeto WiFiClient Client
02: Se existe uma requisição HTTP no objeto Client então:
03:    Armazenar o texto da requisição em uma variável Requisição.
04:    Se existe informações enviadas via método POST então:
05:       Grave essas informações no texto da requisição.
06:    Armazenar o texto entre o “GET /” e “HTTP” em uma variável Requisição.
07:    Se Requisição estiver vazia ou Requisição conter “TelaInicial” então:
08:       Chamar a função TelaInicial(Client).
09:    Senão se Requisição conter “DadosOperador” então:
10:       Se no texto da requisição conter “Nome” então:
11:          Armazenar o texto do nome do operador e da matricula do operador nas
respectivas variáveis.
12:          Substituir o caractere “#” por espaço.
13:          Decodificar o texto chamando a função DecodeURL(Texto).
14:          Modificar variável de verificação dos dados de operador para verdadeiro.
15:       Chamar a função TelaDadosOperador(Client).
16:    Senão se Requisição conter “RealizarOperacao” então:
17:       Chama função LimpaVariaveis().
18:       Chama função TelaRealizaOperacao(Client).
19:    Senão se Requisição conter “LerManutenção” então:
20:       Chama função LerManutencao(Client, falso).
21:    Senão se Requisição conter “Comprovante” então:
22:       Chama função TelaComprovante(Client).
23:    Senão se Requisição conter “RealizarManutencao” então:
24:       Chama função LerManutencao(Client, verdadeiro).
25:    Senão se Requisição conter “FormularioManutencao” então:
26:       Se no texto da requisição conter “Tipo” então:
27:          Armazenar o texto dos dados de manutenção nas respectivas variáveis.
28:          Substituir o caractere “#” por espaço.
29:          Decodificar o texto chamando a função DecodeURL(Texto).
30:          Modificar variável de verificação dos dados de manutenção para verdadeiro
31:       Chama a função FormularioManutenção(Client)
32:    Senão se Requisição conter “GravarTag” então:
33:       Chama a função TelaGravarTag(Client).
34:    Senão:
35:       Apresentar mensagem com texto “Requisição Invalida”.
36: Limpar memória.
37: Reinicie a comunicação com o leitor RC522.
```

Fonte: O Autor.

Inicialmente o algoritmo gera um objeto chamado *Client*, responsável pela comunicação WiFi, e aguarda a requisição HTTP realizada pelo usuário, o texto da requisição é tratado, e por meio do método GET e POST é adquirido os parâmetros enviados pelo usuário. Ao analisar os parâmetros enviados, é chamada a função responsável pela respectiva tela solicitada juntamente com o parâmetro *Client*, apresentando uma mensagem de erro caso o parâmetro não seja válido.

Os dados enviados via formulário pelo usuário, como o nome, matrícula e dados da manutenção para gravação são enviados ao microcontrolador por meio da requisição POST, sendo necessário o tratamento destas informações, como a substituição do caractere “#” por um espaço e a decodificação completa do código URL por meio da função decodeURL. Após o preenchimento das informações, as variáveis globais responsáveis pela verificação da entrada de dados recebem o valor “verdadeiro”, de acordo com a informação que foi enviada.

Quando o usuário realizar a solicitação para a tela responsável pela função “Definir Operação”, a função “Limpa Variaveis” é chamada para evitar que os dados anteriores de uma leitura ou de uma gravação interfiram em uma nova operação.

3.2.2.2 Tela Inicial

A função Tela Inicial é chamada sempre que o usuário realizar uma requisição vazia ou de “/TelaInicial”, e esta representa o bloco Tela Inicial no fluxograma apresentado no APÊNDICE A. O pseudocódigo (Algoritmo 6) desta função é apresentado na Figura 19 juntamente com um exemplo de tela correspondente.

Para o início da construção de todas as telas do sistema é realizada a montagem do cabeçalho em HTML que contém além das configurações da tela, a folha de estilo dos elementos em CSS. Para evitar repetição dentro do código, foi criada separadamente uma função auxiliar contendo somente o cabeçalho.

Depois de realizar a construção do cabeçalho, são verificados os dados informados pelo operador, por meio da variável de controle, e apresentando-os ao usuário. A partir daí é exibido o botão de acesso à tela “Realizar a Operação”. Informar dados do operador

A função Informar dados do operador visa à construção de tela para aquisição dos dados do operador, sendo eles o nome e a matrícula deste, e é representada pelo bloco “Informar dados do operador” do fluxograma (APÊNDICE A). A tela construída apresenta um formulário solicitando as informações necessárias, que após ser preenchida e enviada, é recuperada por meio do método POST.

A Figura 20 apresenta o pseudocódigo (Algoritmo 7) da função “Informar dados do operador”, e a imagem da tela gerada por esta função.

Figura 19. Função Tela Inicial.

(a) Pseudocódigo do Algoritmo 6.

Parâmetros: WiFiClient Client

1: Construir o cabeçalho HTML.
2: **Se** Os dados do operador foram preenchidos **então:**
3: Apresentar dados do operador.
4: Apresentar botão “Realizar Operação”.
5: Apresentar botão “Preencher dados do operador”
6: Finalizar página HTML

(b) Tela com os dados inseridos do usuário.

Fonte: O Autor.

Figura 20. Função Informar Dados do Operador.

(a) Pseudocódigo do Algoritmo 8

Parâmetros: WiFiClient Client

1: Construir o cabeçalho HTML.
2: **Se** Os dados do operador foram preenchidos **então:**
3: Apresentar dados do operador.
4: Apresentar formulário de envio.
5: Apresentar botão “Voltar”, que redireciona para a função Tela Inicial.
6: Finaliza página HTML

(b) Tela de inserção dos dados do operador.

Fonte: O Autor.

3.2.2.3 Definir operação

A função “Definir operação” é representada pelo bloco “Definir Operação” do fluxograma apresentado no APÊNDICE A. O objetivo desta função é gerar uma página HTML para a escolha da operação que deve ser realizada pelo usuário.

Para evitar erros na utilização do programa, é utilizada a variável de controle responsável por verificar as informações do operador como condição para que a função seja executada corretamente. Caso não sejam preenchidas as informações do operador, a referida função deve retornar uma mensagem de erro, solicitando o preenchimento dos dados antes de utilizar desta função.

A Figura 21 apresenta o pseudocódigo (Algoritmo 9) da função “Definir operação”, e a imagem da tela gerada por esta função.

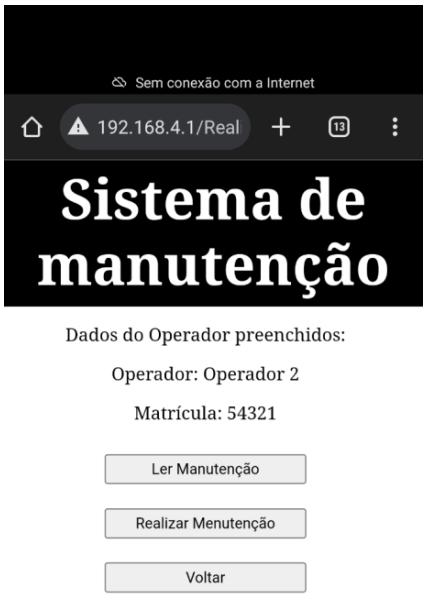
Figura 21. Função Definir operação.

(a) Pseudocódigo do Algoritmo 9.

Parâmetros: WiFiClient Client

01: **Se** os dados do operado foram preenchidos **então:**
02: Construir o cabeçalho em HTML.
03: Apresentar dados do operador.
04: Apresentar botão “Ler manutenção”.
05: Apresentar botão “Gravar manutenção”.
06: Apresentar botão “Voltar”, que redireciona para a função Tela Inicial.
07: Finaliza página HTML
08: **Senão:**
09: Construir o cabeçalho em HTML.
10: Apresentar mensagem de erro.
11: Apresentar botão “Voltar”, que redireciona para a função Tela Inicial.
12: Finaliza página HTML

(b) Tela de escolha da operação.



Fonte: O Autor.

3.2.2.4 Ler última manutenção: parte gráfica

A tela da função “Ler última manutenção” possui como objetivos: apresentar uma tela onde é realizada a solicitação para aproximação do equipamento na etiqueta; chamar a função responsável pela leitura; apresentar a mensagem de confirmação quando a etiqueta for lida com sucesso; e redirecionar a geração do comprovante ou do formulário de manutenção, dependendo da operação escolhida. Para identificação da operação solicitada, uma variável do tipo booleana é enviada, referente à gravação da manutenção. A função “Ler última manutenção” está representada pelos blocos “Ler etiqueta” e “Etiqueta foi lida?” no APÊNDICE A.

A Figura 22 apresenta o pseudocódigo (Algoritmo 10) da função “Ler última manutenção”, e a imagem da tela gerada por esta função.

Figura 22. Função Ler última manutenção.

(a) Pseudocódigo do Algoritmo 10.

Parâmetros: WiFiClient Client, boolean Manutencao

```
01: Se os dados do operado foram preenchidos então:
02:     Construir o cabeçalho em HTML.
03:     Apresentar dados do operador.
04:     Apresentar mensagem “Aproxime o equipamento da etiqueta”
05:     Chamar a função LerTag().
06:     Esperar a execução da função.
07: Apresentar mensagem “Etiqueta Lida”.
08:  Se Manutenção tem valor falso então:
09: Apresentar o botão “Gerar Comprovante”.
10:     Senão:
11: Apresentar o botão “Preencher formulário”.
12:     Finalizar página HTML.
13: Senão:
14:     Construir o cabeçalho em HTML.
15:     Apresentar mensagem de erro.
16: Apresentar botão “Voltar”, que redireciona para a função Tela Inicial.
17:     Finaliza página HTML
```

(b) Tela para leitura da etiqueta com a última manutenção gravada.

Fonte: O Autor.

3.2.2.5 Informar dados de manutenção

A função “Informar dados de manutenção” é responsável pela aquisição dos dados de manutenção que serão gravadas na etiqueta, e está representada pelo bloco “Informar dados de manutenção” presente no fluxograma (APÊNDICE A).

As informações solicitadas no formulário disponibilizado por esta função são: Tipo de manutenção; Código dos itens substituídos na manutenção; Código dos itens que contém avaria; Dia e hora da manutenção; e Observações. O campo “Observações” é destinado a informações adicionais da manutenção, as quais não são gravadas na etiqueta, porém estão presentes no seu comprovante de gravação.

O botão para realizar a gravação só se torna disponível ao usuário após o preenchimento e o envio dos dados de manutenção com o clique no botão “enviar”.

A Figura 23 apresenta o pseudocódigo da função “Informar dados de manutenção” (Algoritmo 11), e a imagem da tela gerada por esta função.

Figura 23. Função Informar dados de manutenção.

(a) Pseudocódigo do Algoritmo 11.

Parâmetros: WiFiClient Client

```
01: Se os dados do operado foram preenchidos então:
02:     Construir o cabeçalho em HTML.
03:     Apresentar dados do operador.
04:     Se os dados de manutenção foram preenchidos então:
05:         Apresentar dados de manutenção
06:         Apresentar o botão “Gravar na etiqueta”.
07:         Apresentar texto de modificação dos dados.
08:     Senão:
09:         Apresentar texto para inserir os dados.
10:     Apresenta o formulário para envio dos dados de manutenção.
11:     Apresentar botão “Voltar”, que redireciona para a função Tela Inicial.
12:     Finalizar página HTML.
13: Senão:
14:     Construir o cabeçalho em HTML.
15:     Apresentar mensagem de erro.
16:     Apresentar botão “Voltar”, que redireciona para a função Tela Inicial.
17:     Finaliza página HTML
```

(b) Tela de preenchimento de dados da manutenção

Fonte: O Autor.

3.2.2.6 Gravação da manutenção: parte gráfica

A tela da função “Gravação da manutenção” possui aspecto similar à tela da função de leitura, apresentado na seção 3.2.2.4. Seus principais objetivos são: Apresentar uma tela onde é realizada a solicitação para aproximação do equipamento na etiqueta; chamar a função responsável pela gravação; e apresentar a mensagem de confirmação quando a informação for gravada na etiqueta com sucesso e o redirecionamento para geração do comprovante.

A função “Gravação da manutenção” está representada pelos blocos “Gravar etiqueta” e “Foi gravado na etiqueta”, apresentados no APÊNDICE A. A Figura 24 apresenta o pseudocódigo da tela da função “Gravação da manutenção” (Algoritmo 12), e a imagem da tela gerada por esta função.

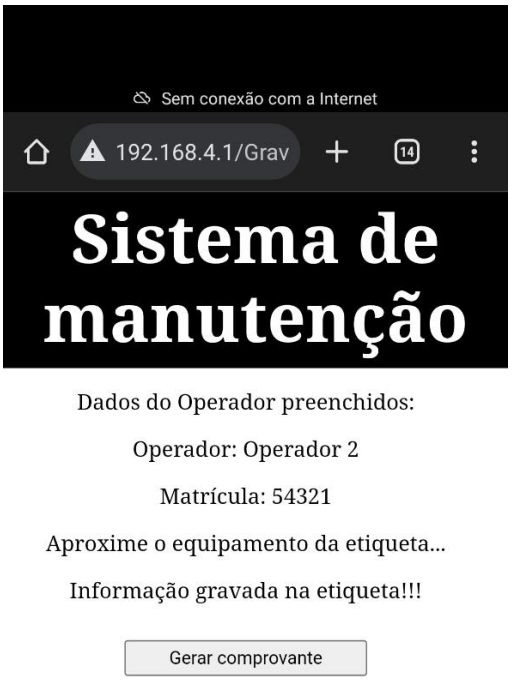
Figura 24. Função Gravação da manutenção.

(a) Pseudocódigo do Algoritmo 12

Parâmetros: WiFiClient Client

01: **Se** os dados do operado foram preenchidos **então:**
02: Construir o cabeçalho em HTML.
03: Apresentar dados do operador.
04: Apresentar mensagem “Aproxime o equipamento da etiqueta”
05: Chamar a função **EscreverTag()**.
06: Esperar a execução da função.
07: Apresentar mensagem “Etiqueta Lida”.
08: Apresentar o botão “Gerar Comprovante”.
09: Finalizar página HTML.
10: **Senão:**
11: Construir o cabeçalho em HTML.
12: Apresentar mensagem de erro.
13: Apresentar botão “Voltar”, que redireciona para a função Tela Inicial.
14: Finaliza página HTML

(b) Tela para gravação da etiqueta



Fonte: O Autor.

3.2.2.7 Gerar comprovante

A função “Gerar comprovante” é responsável pela geração da interface gráfica e compilação dos dados para emissão de comprovante, tanto para leitura da última manutenção, como para a gravação da manutenção atual. Esta função representa todos os blocos do tipo: “Gerar comprovante de...” no APÊNDICE A, cujo tipo a ser gerado, é identificado por meio de uma variável global de controle que possui seu valor definido na solicitação da tela de comprovante.

O comprovante de leitura apresenta as informações do operador que realizou a leitura e os dados presentes na etiqueta. O comprovante de manutenção apresenta as informações do formulário e dos dados do operador que realizou a manutenção, e também as informações contidas na etiqueta antes da gravação.

A geração do arquivo em PDF é realizada por um código em Javascript presente na construção da tela em HTML, onde é vinculado o comprovante de impressão ao botão localizado na tela.

A Figura 25 apresenta o pseudocódigo (Algoritmo 13) responsável pela construção da tela de comprovante, e a imagem da tela gerada por esta função.

3.2.2.8 Ler etiqueta

A função “Ler etiqueta” é a responsável pela comunicação e leitura das informações presentes na memória da etiqueta. Em união com a parte gráfica, apresentada na seção 3.2.2.4, representa o comportamento dos blocos “Ler etiqueta”, apresentados no APÊNDICE A.

O algoritmo percorre e lê todas as informações presentes nos blocos de informação, salvando tanto o ID como os valores lidos em variáveis globais para utilização pelo resto do programa. Os blocos de acesso aos setores são ignorados, e identificados pelos blocos com valores múltiplos de quatro menos um e o bloco zero.

A Figura 26 apresenta o pseudocódigo (Algoritmo 14) desta função.

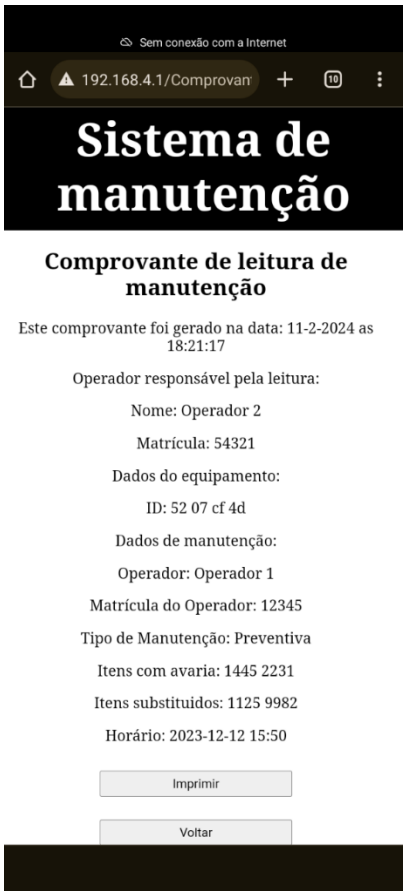
Figura 25. Função Gerar comprovante.

(a) Pseudocódigo do Algoritmo 13.

Parâmetros: WiFiClient Client

01: Construir o cabeçalho em HTML.
02: **Se** o comprovante for de leitura **então:**
03: Apresenta a data e hora em que foi gerado o comprovante, configurado em Javascript.
05: Apresentar dados do operador.
06: Apresentar dados da etiqueta
07: Apresentar botão “Voltar”, que redireciona para a função Definir Operação.
08: **Senão se** o comprovante for de manutenção **então:**
09: Apresenta a data e hora em que foi gerado o comprovante, configurado em Javascript.
10: Apresentar dados do operador.
11: Apresentar dados lidos da etiqueta.
12: Apresentar dados escritos na etiqueta.
13: Apresentar botão “Voltar”, que redireciona para a função Definir Operação.
14: Apresentar botão “Imprimir”, configurado em Javascript para geração do PDF.
15: Apresentar botão “Voltar”, que redireciona para a função Definir Operação.
16: Finaliza página HTML

(b) Tela de comprovante de leitura.



Fonte: O Autor.

Figura 26. Pseudocódigo da função Ler etiqueta

```
01: Para bloco de 1 até 63 faça:
02: Configure variável etiqueta como não lida
03:    Enquanto a etiqueta não for lida faça:
04:       Se existe uma nova detecção de etiqueta e é possível ler então:
05:          Configure as variáveis para leitura.
06:          Se a autenticação não retornar erro então:
07:             Enquanto a leitura não der erro e o bloco for menor que 63 faça:
08:                Se não for um bloco de acesso então:
09:                   Se a leitura não de erro então:
10:                      Se for a primeira leitura então:
11:                         Grave o ID da etiqueta.
12:                      Grave a informação do bloco.
13:                      Vá para o próximo bloco.
14:                      Configure variável etiqueta como lida.
15:                Senão:
16:                   Pule este bloco.
17:             Reinicie a comunicação com o leitor RC522.
```

Fonte: O Autor.

3.2.2.9 Gravar etiqueta

A função “Gravar etiqueta” é responsável pela comunicação e gravação dos dados enviados pelo operador na memória da etiqueta. Em união com a parte gráfica, apresentada na seção 3.2.2.6, esta função representa o comportamento dos blocos “Gravar etiqueta”, apresentado no APÊNDICE A. A Figura 27 apresenta o pseudocódigo (Algoritmo 15) da função “Gravar na etiqueta”.

Inicialmente, o algoritmo verifica se não foi ultrapassado o limite de 752 caracteres referentes aos dados enviados para a gravação, com o acréscimo dos caracteres “#” responsáveis por separar as informações inseridas. Em seguida, o algoritmo realiza a gravação de todos os caracteres enviados pelo operador sequencialmente, finalizando sua execução após o envio do último caractere.

Figura 27. Pseudocódigo da função Gravar etiqueta.

```
01: Concatene as informações para gravação em uma única variável, separadas pelo
caractere “#”
02: Se a variável para envio não tiver mais caracteres que 752 então:
03:   Para bloco de 1 até 63 faça:
04:     Configure variável etiqueta como não lida.
05:     Enquanto a etiqueta não for lida faça:
06:       Se existe uma nova detecção de etiqueta e é possível ler então:
07:         Configure as variáveis para gravação.
08:         Se a autenticação não retornar erro então:
09:           Enquanto a escrita não der erro e o bloco for menor que 63 faça:
10:             Se não for um bloco de acesso então:
11:               Se a variável de envio ainda contém informações
12:                 Se a escrita não de erro então:
13:                   Envie os primeiros 16 caracteres da variável para envio.
14:                   Retire os primeiros 16 caracteres da variável para envio.
15:                   Vá para o próximo bloco.
16:                   Configure variável etiqueta como lida.
17:             Senão:
18:               Pule este bloco.
19:           Reinicie a comunicação com o leitor RC522.
```

Fonte: O Autor.

3.3 TESTE DO SISTEMA COMPLETO

A partir das programações desenvolvidas para o sistema de manutenção proposto, foi realizado um teste com o sistema completo, onde inicialmente deve ser cadastrado um operador com os seguintes dados:

- Nome: Operador 2;
- Matrícula: 54321.

Após o cadastro deve ser realizada a leitura da última manutenção presente na etiqueta, os dados esperados de leitura são:

- ID da Tag: 52 07 cf 4d (etiqueta vinculado a um Transformador 3F 500 MVA);
- Operador que realizou a manutenção: Operador 1;
- Matrícula do operador que realizou a manutenção: 12345;
- Tipo de Manutenção: Preventiva;
- Itens com avaria: 1445 2231;
- Itens substituídos: 1125 9982;
- Horário: 12 de dezembro de 2023, às 15:50.

Por fim, deve ser realizada a gravação de uma nova manutenção, fornecendo os seguintes dados de manutenção:

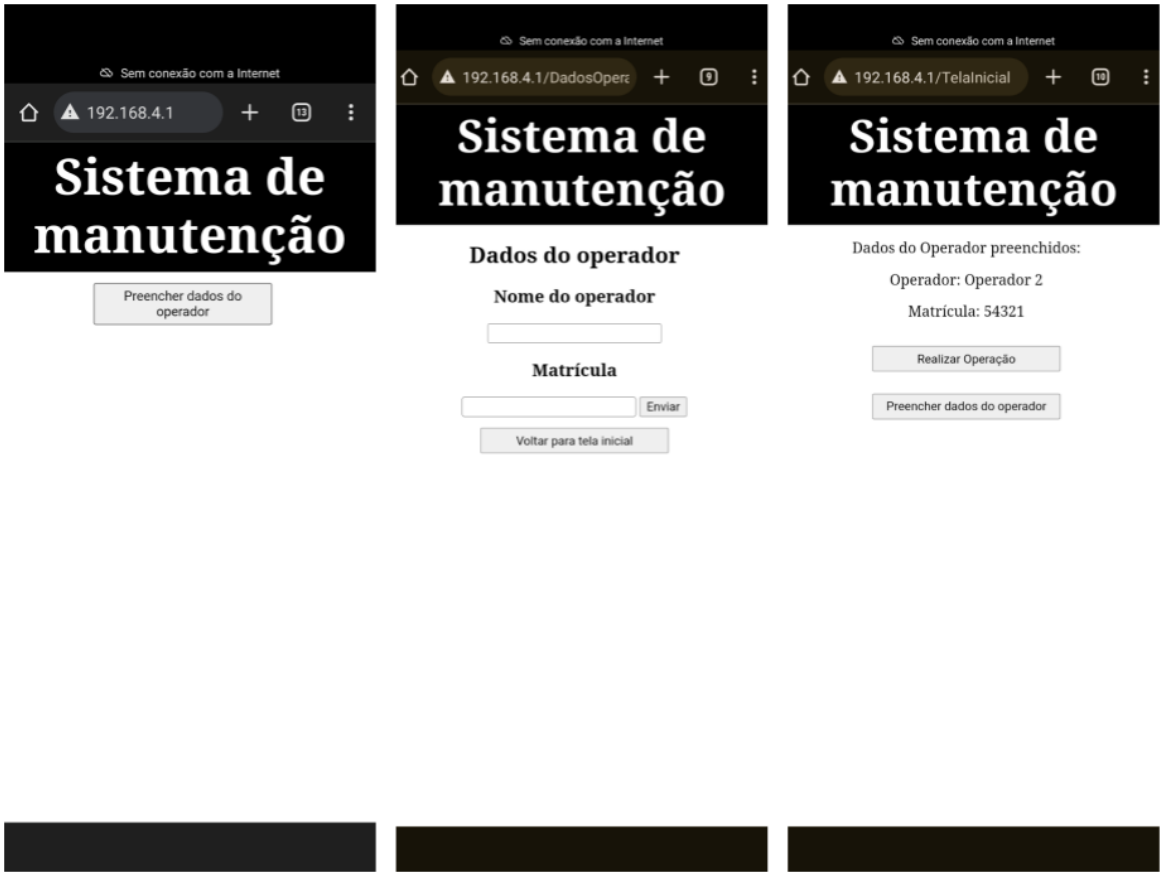
- Nome do operador que realizou a manutenção: Operador 2;
- Matrícula do operador que realizou a manutenção: 54321;
- Tipo: Corretiva;
- Itens Substituídos: 1772 9912;
- Itens Avariados: 7712 4472;
- Horário da manutenção: 01 de janeiro de 2024, às 21:00;
- Observações: Marcas de ferrugem.

Como resultado, o sistema deve gerar dois comprovantes de operação, um de leitura da última manutenção e outro da gravação da manutenção atual, de acordo com as informações fornecidas e solicitadas.

Na página inicial, caso o usuário não tenha preenchido as informações solicitadas, como o nome e a matrícula do operador, o usuário somente terá a opção de clicar no botão “Preencher os dados do operador”. A partir do preenchimento e envio das informações do usuário, estas ficarão fixas no cabeçalho da página permitindo ao usuário realizar as opções de leitura e gravação de manutenção.

A Figura 28 apresenta a tela principal antes do preenchimento dos dados, a tela de preenchimento das informações e a tela principal com as informações preenchidas de acordo com os dados fornecidos, respectivamente. A última tela possui o botão “Preencher dados do operador” em caso de correção de informações.

Figura 28. Sequência de telas à função: preenchimento dos dados do operador.



Fonte: O Autor.

Para leitura de uma manutenção, a partir da tela “Definir Operação”, o usuário será redirecionado para tela de leitura, onde será solicitado aproximar o equipamento na etiqueta para a aquisição das informações. Ao realizar a leitura, a página será automaticamente atualizada confirmando operação e permitindo a geração do comprovante de leitura da última manutenção, podendo gerar um PDF e salvar no próprio dispositivo do operador.

A Figura 29 apresenta a sequência de telas responsáveis pela função de leitura da última manutenção, onde estão presentes, no comprovante de leitura gerado, os dados previstos para a leitura da etiqueta neste teste do sistema completo.

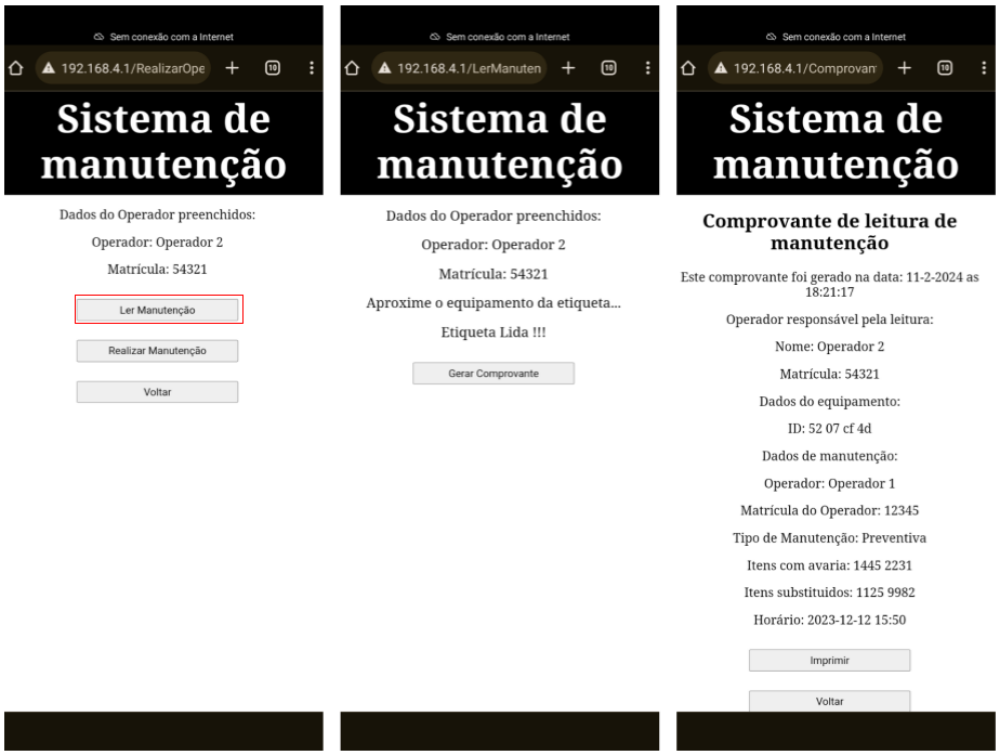
Para a gravação de uma manutenção, a partir da tela “Definir Operação”, o usuário será redirecionado para tela de leitura, onde o será solicitado aproximar o equipamento da etiqueta para aquisição das informações que serão substituídas. Ao realizar a leitura, a página será automaticamente atualizada confirmando a operação e permitindo ao usuário preencher as informações da manutenção.

A Figura 30 apresenta a sequência de telas citadas. A partir do preenchimento e do envio dos dados da manutenção de acordo com as informações definidas previamente, o usuário poderá selecionar a opção de gravar na etiqueta, onde será gerada uma tela solicitando ao usuário aproximar o equipamento da etiqueta para gravação dos dados. Ao realizar a gravação, o usuário poderá visualizar e gerar o PDF do comprovante de gravação de manutenção, que apresenta os dados que foram substituídos e as novas informações da manutenção gravadas na etiqueta. A Figura 31 apresenta a tela para a função de gravação da última manutenção.

Conforme exibido nas telas presentes da Figura 28 à Figura 31, o teste do sistema completo obteve resultados satisfatórios, com todos os parâmetros e telas obtidas coincidentes com aqueles previamente designados, e a sequência das funções e suas telas correspondentes estando em conformidade com a operação do sistema de manutenção definida pelo Fluxograma constante noAPÊNDICE A.

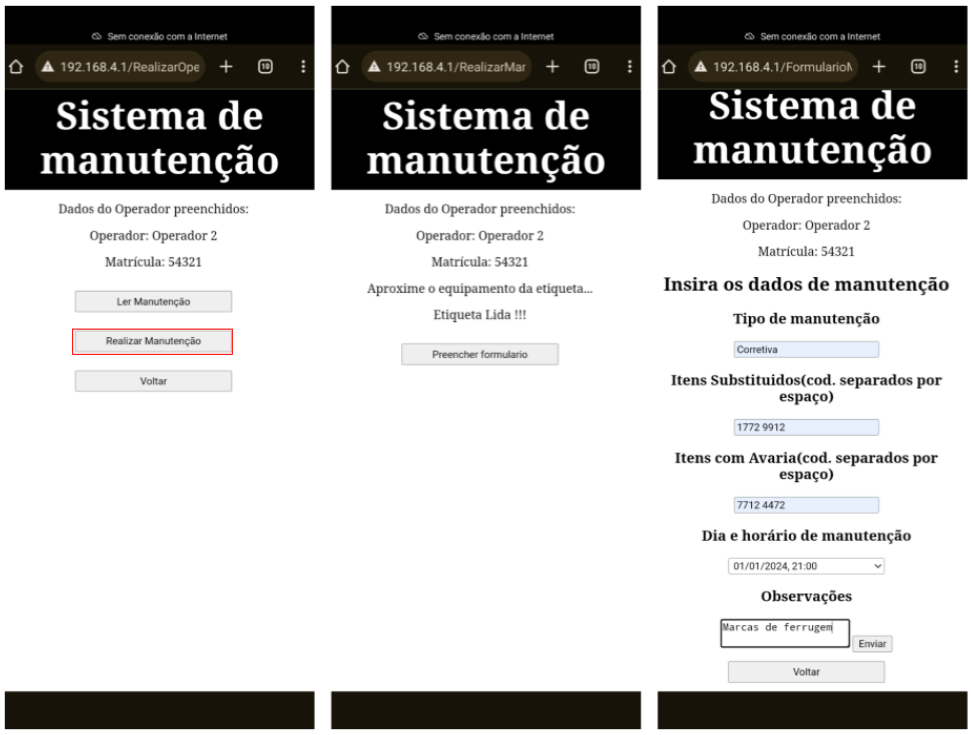
Adicionalmente, O APÊNDICE D e APÊNDICE E apresentam, respectivamente, o comprovante de leitura da última manutenção e o comprovante de gravação da manutenção atual, geradas como resultado deste teste.

Figura 29. Sequência de telas à função: leitura da última manutenção.



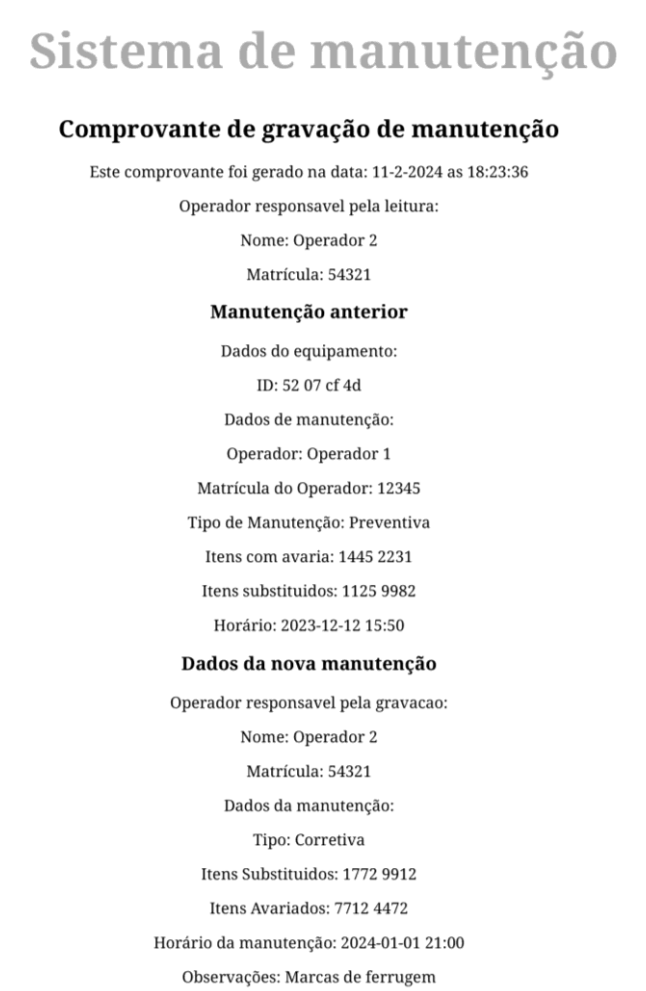
Fonte: O Autor.

Figura 30. Sequência de telas à função: gravação da última manutenção.



Fonte: O Autor.

Figura 31. Sequência de telas à função: gravação da última manutenção.



Fonte: O Autor.

4 CONCLUSÕES E PROPOSTAS DE CONTINUIDADE

O presente Trabalho de Conclusão de Curso apresentou o desenvolvimento de um sistema de controle de manutenção de equipamentos elétricos ou industriais utilizando a tecnologia RFID e plataformas eletrônicas (microcontroladores) de baixo custo de aquisição e complexidade de desenvolvimento.

A montagem do protótipo do sistema ocorreu de forma satisfatória, onde foram realizados testes iniciais para familiarização das funcionalidades dos dispositivos eletrônicos utilizados, requisitadas para implementação do referido sistema.

O teste do sistema completo apresentou seu comportamento e progressão de telas de acordo com a lógica desenvolvida pelo fluxograma apresentado no APÊNDICE A, sendo capaz de realizar a identificação de equipamentos industriais e seu controle de manutenção, alcançando com êxito os objetivos específicos previstos, possibilitando a leitura da última manutenção, gravação uma nova manutenção e geração de comprovantes que possam ser salvos no dispositivo do usuário em formato PDF.

O sistema foi elaborado para ser o mais abrangente possível, prevendo limitações como a falta de uma rede Wi-Fi no local da manutenção. Para avanços posteriores é recomendado um estudo detalhado do local de aplicação para contornar esta falta de conexão, além do desenvolvimento de um algoritmo mais complexo, que preencha de forma automática as informações em um banco de dados, planilha ou *software* de gerenciamento de manutenção utilizado pela indústria ou onde o sistema possa ser utilizado.

REFERÊNCIAS

ALLINBEST. 134.2K Animal Tag FDX-B ISO11784 Reader Module. **allinbest**, 2017. Disponível em: <https://allinbest.com/blog/1342k-animal-tag-fdxb-iso11784-reader-module/>. Acesso em: 09 Julho 2023.

ALVES, Josemar. RFID - Cartões Mifare. **EMBARCADOS**, 2017. Disponível em: <https://embarcados.com.br/rfid-cartoes-mifare/>. Acesso em: 18 Fevereiro 2024.

BALANIS, C. A. **Antenna Theory:** Analysis and Design. 3rd Ed. ed.

GUIMARÃES, Fábio. Módulo RFID RC522 com Arduino. **Mundo Projetado**, 2021. Disponível em: <https://mundoprojetado.com.br/modulo-rfid-rc522/>. Acesso em: 23 Novembro 2023.

IEEE RAS UFCG. O Que É Um Microcontrolador. **RAS UFCG**, 2020. Disponível em: <https://edu.ieee.org/br-ufcgras/o-que-e-um-microcontrolador/>. Acesso em: 09 Julho 2023.

KARDEC, Alan. **Gestão Estratégica e Manutenção**.

LOBODAROBOTICA. O Que É ESP32? Pra Que Serve? Quando Usar? **LOBODAROBOTICA**, 2021. Disponível em: <https://lobodarobotica.com/blog/o-que-e-esp32-pra-que-serve-quando-usar/>. Acesso em: 09 Julho 2023.

LOUREIRO, Gabriel D. S. M.; SOUZA, Isabella Q. D.; LOPES, Marcelle G. D. M. RFID: Tecnologia. **UFRJ**, 2015. Disponível em: <https://www.gta.ufrj.br/grad/15_1/rfid/tecnologia.html>. Acesso em: 23 Julho 2023.

MELLO, Marcio. O que é um Microcontrolador, para que serve e principais usos. **Victor Vision**, 2022. Disponível em: <https://victorvision.com.br/blog/o-que-e-um-microcontolador/>. Acesso em: 30 Agosto 2023.

NETO, Manoel Sátiro de M. **Tecnologia RFID e suas aplicações**. Universidade Federal de Campina Grande. Campina Grande. 2010.

PIECHNICKI, Ademir S. **METODOLOGIAS PARA IMPLANTAÇÃO E DESENVOLVIMENTO DE SISTEMAS DE GESTÃO DA MANUTENÇÃO: AS MELHORES PRÁTICAS**. UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ, CAMPUS PONTA GROSSA. PONTA GROSSA, p. 13. 2011.

PRIORITY1DESIGN. Dual animal tag rfid data logger. **Priority1Design**, 2007. Disponível em: <https://www.priority1design.com.au/animal_tag_rfid_data_logger.html>. Acesso em: 09 Julho 2023.

PRIORITY1DESIGN. FDX-B Animal Identification Protocol description. **Priority1Design**, 2007. Disponível em: <https://www.priority1design.com.au/fdx-b_animal_identification_protocol.html>. Acesso em: 09 Julho 2023.

ROBERTI, Mark; VIOLINO, Bob. The History of RFID Technology. **RFID Journal**, 2005. Disponível em: <https://www.rfidjournal.com/the-history-of-rfid-technology>. Acesso em: 17 Fevereiro 2024.

ROGER. Frequencia LF, HF e UHF: Qual é a diferença. **RFIDFuture.**, 2023. Disponível em: <https://www.rfidfuture.com/pt/lf-hf-and-uhf-frequency-whats-the-difference.html>. Acesso em: 09 Julho 2023.

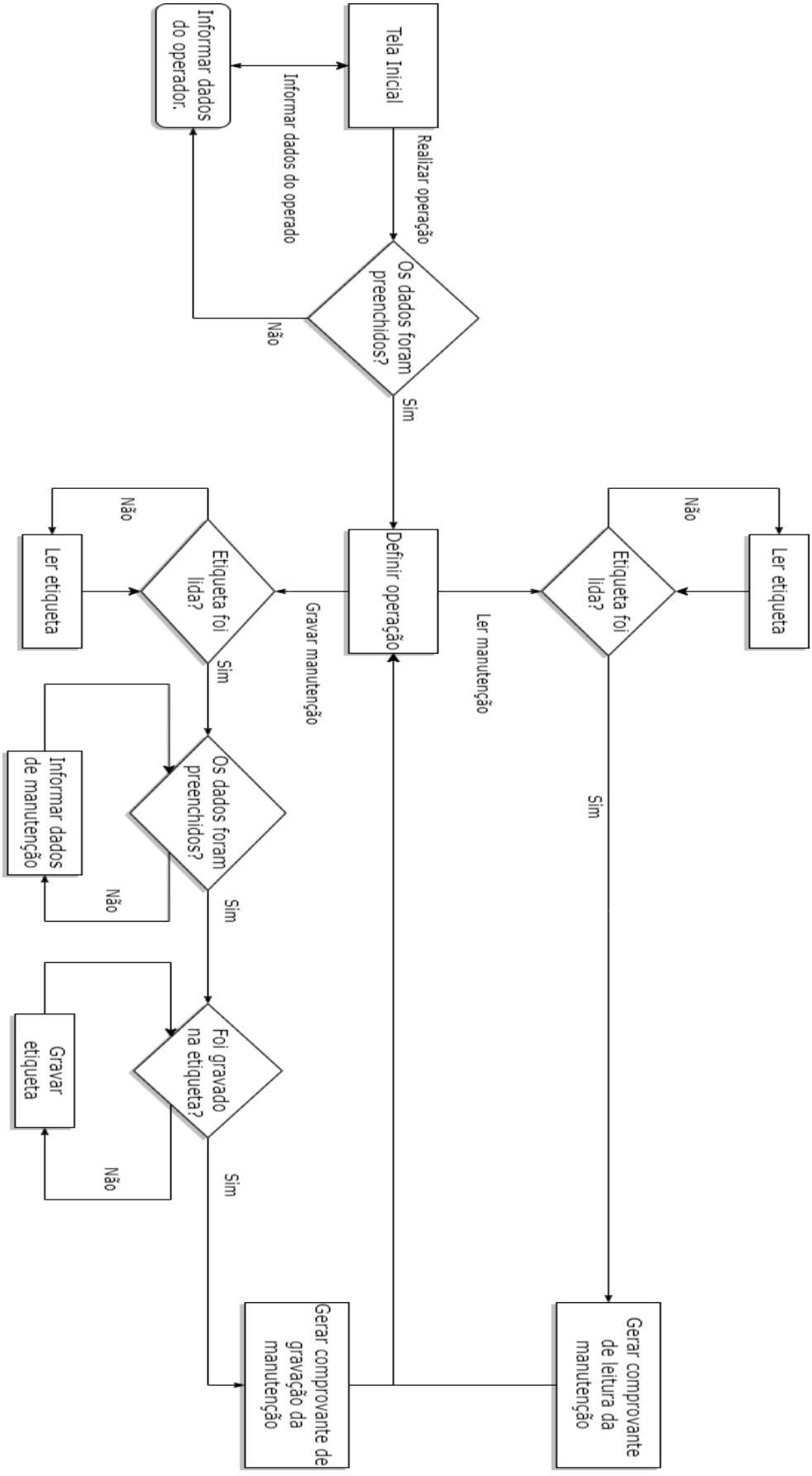
SANTOS, Luan. Saiba o que é e qual é a importância da manutenção elétrica. **Keepfy**, 2021. Disponível em: <https://keepfy.com/blog/manutencao-eletrica-qual-sua-importancia/>. Acesso em: 15 Janeiro 2024.

SIMÕES, Haroldo M. C. CARTÕES RFID E O MÓDULO MFRC522. **Módulo Eletrônica**, 2019. Disponível em: <https://blog.moduloeletronica.com.br/cartoes-rfid-e-o-modulo-mfrc522/>. Acesso em: 15 Janeiro 2024.

SOARES, Isadora. Tipos de Manutenção: guia completo e atualizado. **COBLI**, 2023. Disponível em: <https://www.cobli.co/blog/tipos-de-manutencao/>. Acesso em: 15 Janeiro 2024.

THAKUR, Manoj R. ESP32 DevKit ESP32-WROOM GPIO Pinout. **circuits4you**, 2018. Disponível em: <https://circuits4you.com/2018/12/31/esp32-devkit-esp32-wroom-gpio-pinout/>. Acesso em: 24 Março 2024.

APÊNDICE A – FLUXOGRAMA DA OPERAÇÃO DO SISTEMA



APÊNDICE B – ALGORITMO PARA GRAVAÇÃO DA ETIQUETA

```
#include <MFRC522.h>


#include <SPI.h>
#define SS_PIN 21
#define RST_PIN 22
#define SIZE_BUFFER 18
#define MAX_SIZE_BLOCK 16
MFRC522::MIFARE_Key key;
MFRC522::StatusCode status;
MFRC522 mfrc522(SS_PIN, RST_PIN);
String mensagem = “Mensagem gravada”;
char valortag[MAX_SIZE_BLOCK];
byte buffer[MAX_SIZE_BLOCK]= “”;
void setup() {
  Serial.begin(115200);
  Serial.println();
  SPI.begin();
  mfrc522.PCD_Init();
}
void loop() {
  if ( ! mfrc522.PICC_IsNewCardPresent()){
    return;
  }
  if ( ! mfrc522.PICC_ReadCardSerial()){
    return;
  }
  mfrc522.PICC_DumpDetailsToSerial(&(mfrc522.uid));
  for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
  byte bloco = 1;
  byte tamanho = SIZE_BUFFER;
  mensagem.toCharArray((char*)buffer, MAX_SIZE_BLOCK);
  for(byte i= mensagem.length(); i < MAX_SIZE_BLOCK; i++){
    buffer[i] = ‘ ‘;
  }
  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));
    return;
  }
  status = mfrc522.MIFARE_Write(bloco, buffer, MAX_SIZE_BLOCK);
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F(“MIFARE_Write() failed: “));
```

```
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }else{
    Serial.println(F("MIFARE_Write() success: "));
  }
  mfrc522.PICC_HaltA();
  mfrc522.PCD_StopCrypto1();
}
```

APÊNDICE C – ALGORITMO PARA LEITURA DA ETIQUETA.

```
#include <MFRC522.h>
#include <SPI.h>
#define SS_PIN 21
#define RST_PIN 22
#define SIZE_BUFFER 18
#define MAX_SIZE_BLOCK 16
MFRC522::MIFARE_Key key;
MFRC522::StatusCode status;
MFRC522 mfrc522(SS_PIN, RST_PIN);
char valortag[MAX_SIZE_BLOCK];
byte buffer[MAX_SIZE_BLOCK]= “”;
void setup() {
  Serial.begin(115200);
  Serial.println();
  SPI.begin();
  mfrc522.PCD_Init();
}
void loop() {
  if ( ! mfrc522.PICC_IsNewCardPresent()){
    return;
  }
  if ( ! mfrc522.PICC_ReadCardSerial()){
    return;
  }
  mfrc522.PICC_DumpDetailsToSerial(&(mfrc522.uid));
  for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
  byte buffer_read[SIZE_BUFFER] = {0};
  byte bloco = 1;
  byte tamanho = SIZE_BUFFER;
  status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, bloco, &key,
&(mfrc522.uid));
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F(“Authentication failed: “));
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }
  status = mfrc522.MIFARE_Read(bloco, buffer_read, &tamanho);
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F(“Reading failed: “));
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }
  Serial.print(F(“\nDados bloco [“));
  Serial.print(bloco);
  Serial.print(F(“]: “));
```

```
  for (uint8_t i = 0; i < MAX_SIZE_BLOCK; i++)
  {
      Serial.write(buffer_read[i]);
  }
  Serial.println(“ “);
  mfrc522.PICC_HaltA();
  mfrc522.PCD_StopCrypto1();
}
```

APÊNDICE D – COMPROVANTE DE LEITURA DA ÚLTIMA MANUTENÇÃO

Sistema de manutenção

Comprovante de leitura de manutenção

Este comprovante foi gerado na data: 11-2-2024 as 18:25:38

Operador responsável pela leitura:

Nome: Operador 2

Matrícula: 54321

Dados do equipamento:

ID: 52 07 cf 4d

Dados de manutenção:

Operador: Operador 1

Matrícula do Operador: 12345

Tipo de Manutenção: Preventiva

Itens com avaria: 1445 2231

Itens substituidos: 1125 9982

Horário: 2023-12-12 15:50

Imprimir

Voltar

APÊNDICE E – COMPROVANTE DE GRAVAÇÃO DA MANUTENÇÃO ATUAL

Sistema de manutenção

Comprovante de gravação de manutenção

Este comprovante foi gerado na data: 11-2-2024 as 18:23:36

Operador responsavel pela leitura:

Nome: Operador 2

Matrícula: 54321

Manutenção anterior

Dados do equipamento:

ID: 52 07 cf 4d

Dados de manutenção:

Operador: Operador 1

Matrícula do Operador: 12345

Tipo de Manutenção: Preventiva

Itens com avaria: 1445 2231

Itens substituidos: 1125 9982

Horário: 2023-12-12 15:50

Dados da nova manutenção

Operador responsavel pela gravacao:

Nome: Operador 2

Matrícula: 54321

Dados da manutenção:

Tipo: Corretiva

Itens Substituidos: 1772 9912

Itens Avariados: 7712 4472

Horário da manutenção: 2024-01-01 21:00

Observações: Marcas de ferrugem

APÊNDICE F - ALGORITMO COMPLETO DO SISTEMA DE MANUTENÇÃO

Para acessar o repositório contendo o algoritmo completo do sistema de manutenção, clique no link:

Repositório do Sistema de Manutenção por RFID.

```
// Importando bibliotecas
#include <WiFi.h>
#include <WiFiClient.h>
#include <WiFiAP.h>
#include <MFRC522.h>
#include <SPI.h>
// Definindo constantes de pinos e buffers do RC522
#define SS_PIN 21
#define RST_PIN 22
#define SIZE_BUFFER 18
#define MAX_SIZE_BLOCK 16
// Configurando RFID
MFRC522::MIFARE_Key key;
MFRC522::StatusCode status;
MFRC522 mfrc522(SS_PIN, RST_PIN);
// Configurando Wifi
const char *ssid = "Sistema de Manutencao";
const char *password = "123456789";
WiFiServer server(80);
/* Definindo variáveis globais, sendo elas:
  DadosOperador(booleano) -> Indica ao sistema se os dados do operador foram
preenchidos ao menos uma vez.
  DadosRealizarManutencao(booleano) -> Indica ao sistema se os dados de manutenção
foram preenchidos,
    permitindo a gravação das informações na etiqueta.
  TextoPOST(String) -> Recebe todas as informações enviadas pelo método POST.
  NomeOperador(String) -> Recebe o nome do operador que está utilizando o sistema.
  MatriculaOperador(String) -> Recebe a matrícula do operador que está utilizando
o sistema.
  TipoOperacao(String) -> Recebe a informação do tipo de manutenção que foi
realizada no equipamento.
  ItensSubstituidos(String) -> Recebe a informação dos itens substituídos durante
a manutenção.
  ItensAvariados(String) -> Recebe a informação dos itens que apresentaram avaria
durante a manutenção.
  Horario(String) -> Recebe todas as informações de data em que ocorreu a
manutenção.
```

```
  OBS(String) -> Recebe o texto de observações informados pelo operador, essa
informação não será
    gravada na etiqueta, somente apresentado no comprovante.
  Tag(String) -> Recebe todas as informações lidas da etiqueta.
  IDTag(String) -> Recebe o ID único da etiqueta lida.
  valortag(Vetor de 16 caracteres) -> Armazena as informações de leitura da
etiqueta, armazenando
    após a leitura de cada bloco na variável Tag.
  TipoComprovante(inteiro) -> Indica a tela de comprovante qual deve ser o tipo de
comprovante gerado, valendo:
    0: erro.
    1: leitura.
    2: manutenção.
  buffer(Vetor de 16 bytes) -> Recebe inicialmente os dados da etiqueta.
  */
boolean DadosOperador = false;
boolean DadosRealizarManutencao = false;
String TextoPOST, NomeOperador, MatriculaOperador, TipoOperacao,
ItensSubstituidos, ItensAvariados, Horario, OBS, Tag, IDTag;
char valortag[MAX_SIZE_BLOCK];
int TipoComprovante = 0;
byte buffer[MAX_SIZE_BLOCK] = "";

void setup() {
  /*Função setup(), responsável pela inicialização do sistema.*/
  //Inicialização da serial para verificação dos processos ao ser conectado com um
computador
  Serial.begin(115200);
  Serial.println();
  Serial.println("Configuring access point...");
  //Inicialização do WiFi, configuração da rede como ponto de acesso e definição
do IP, Gateway e subnet
  IPAddress staticIP(192, 168, 4, 2);
  IPAddress gateway(192, 168, 4, 1);
  IPAddress subnet(255, 255, 255, 0);
  if (!WiFi.softAP(ssid, password)) {
    log_e("Soft AP creation failed.");
    while (1)
      ;
  }
  WiFi.config(staticIP, gateway, subnet);
  IPAddress myIP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(myIP);
  server.begin();
  Serial.println("Server started");
  //Inicialização da comunicação SPI e do modulo RC522
  SPI.begin();
```

```
  mfrc522.PCD_Init();
}
void loop() {
  /*Função loop():
    Objetivo: Esta função é a principal do sistema, e deve rodar infinitamente os
comandos dentro dela...
      O principal objetivo é verificar a conexão de um dispositivo ao sistema, e
caso tenha conexão realizar
      o tratamento do requerimento HTTP e o redirecionamento para tela solicitada.
  */
  WiFiClient client = server.available();
  //Verifica se exite alguma conexão com um dispositivo.
  if (!client) {
    return;
  }
  Serial.println("New Client.");
  //Espera o requerimento chegar.
  while (!client.available()) {
    delay(1);
  }
  //Lê o requerimento HTTP
  String req = client.readStringUntil('\r');
  Serial.println(req);
  //Verificando se existe dados via POST
  if (req.indexOf("POST") != -1) {
    TextoPOST = "";
    while (client.available()) {
      String line = client.readStringUntil('\r');
      //Verifica se existe dados do formulário de dados do operador
      if(line.indexOf("Nome") !=-1){
        TextoPOST = line.substring(line.indexOf("Nome"));
        Serial.print("texto: " + TextoPOST);
      }
      //Verifica se existe dados do formulário de manutenção
      if(line.indexOf("Tipo") !=-1){
        TextoPOST = line.substring(line.indexOf("Tipo"));
        Serial.print("texto: " + TextoPOST);
      }
    }
  }
  //Quebra o requerimento HTTP, salvando somente a parte que será utilizada
  req = req.substring(req.indexOf("/") + 1, req.indexOf("HTTP") - 1);
  Serial.println(req);
  /*Trata o requerimento HTTP da seguinte forma
    Caso não tenha nenhuma informação ou o texto "TelaInicial" -> Chama a tela
inicial.
    Caso apresente o texto "DadosOperador" -> Chama a tela para preenchimento dos
dados do operador.
```

```
    Caso apresente o texto "RealizarOperacao" -> Chama a tela para escolha da
operação que deve ser realizada.
    Caso apresente o texto "LerManutencao" ->  Chama a tela para leitura de uma
etiqueta para a operação ler manutenção.
    Caso apresente o texto "Comprovante" ->  Chama a tela para geração do
comprovante de leitura ou manutenção.
    Caso apresente o texto "RealizarManutencao" -> Chama a tela para leitura de
uma etiqueta para a operação realizar manutenção.
    Caso apresente o texto "FormularioManutencao" -> Chama a tela para
preenchimento dos dados de manutenção.
    Caso apresente o texto "GravarTag" ->  Chama a tela para gravação da etiqueta.
    Caso apresente nenhuma das opções acima -> Chama uma tela de erro.
  */
  if (req.length() == 0 || req.indexOf("TelaInicial") != -1) {
    TelaInicial(client);
  } else if (req.indexOf("DadosOperador") != -1) {
    //Verifica se existem informações via POST
    if (TextoPOST.indexOf("Nome") != -1) {
      //Separa as informações e realiza o tratamento.
      if (TextoPOST.substring(TextoPOST.indexOf("Nome=") + 5,
TextoPOST.indexOf("&")) != "") {
        NomeOperador = TextoPOST.substring(TextoPOST.indexOf("Nome=") + 5,
TextoPOST.indexOf("&"));
        NomeOperador.replace("%23", " ");
        NomeOperador = DecodeURL(NomeOperador);
      }
      if (TextoPOST.substring(TextoPOST.indexOf("Matricula=") + 10) != "") {
        MatriculaOperador = TextoPOST.substring(TextoPOST.indexOf("Matricula=") +
10);
        MatriculaOperador.replace("%23", " ");
        MatriculaOperador = DecodeURL(MatriculaOperador);
      }
      //Somente troca o valor da variável DadosOperador caso tanto o nome como a
matricula estiverem preenchidas
      if (NomeOperador.length() != 0 && MatriculaOperador != 0) {
        DadosOperador = true;
      }
    }
    //Limpa variável TextoPOST
    TextoPOST = "";
    TelaDadosOperador(client);
  } else if (req.indexOf("RealizarOperacao") != -1) {
    //Chama LimpaVariaveis, para que uma operação anterior não interfira em uma
nova operação
    LimpaVariaveis();
    TelaDefinirOperacao(client);
  } else if (req.indexOf("LerManutencao") != -1) {
```

```
    //O parâmetro false é utilizado para identificar que a leitura vem da operação
ler manutenção
    TelaLerManutencao(client, false);
  } else if (req.indexOf("Comprovante") != -1) {
    TelaComprovante(client);
  } else if (req.indexOf("RealizarManutencao") != -1) {
    //O parâmetro true é utilizado para identificar que a leitura vem da operação
realizar manutenção
    TelaLerManutencao(client, true);
  } else if (req.indexOf("FormularioManutencao") != -1) {
    //Verifica se existem informações via POST
    if (TextoPOST.indexOf("Tipo") != -1) {
      //Separa as informações e realiza o tratamento.
      if (TextoPOST.substring(TextoPOST.indexOf("Tipo=") + 5,
TextoPOST.indexOf("&")) != "") {
        TipoOperacao = TextoPOST.substring(TextoPOST.indexOf("Tipo=") + 5,
TextoPOST.indexOf("&"));
        TipoOperacao.replace("%23", " ");
        TipoOperacao = DecodeURL(TipoOperacao);
      }
      if (TextoPOST.substring(TextoPOST.indexOf("ItensS=") + 7,
TextoPOST.indexOf("&ItensA=")) != "") {
        ItensSubstituidos = TextoPOST.substring(TextoPOST.indexOf("ItensS=") + 7,
TextoPOST.indexOf("&ItensA="));
        ItensSubstituidos.replace("%23", " ");
        ItensSubstituidos = DecodeURL(ItensSubstituidos);
      }
      if (TextoPOST.substring(TextoPOST.indexOf("ItensA=") + 7,
TextoPOST.indexOf("&Horario=")) != "") {
        ItensAvariados = TextoPOST.substring(TextoPOST.indexOf("ItensA=") + 7,
TextoPOST.indexOf("&Horario="));
        ItensAvariados.replace("%23", " ");
        ItensAvariados = DecodeURL(ItensAvariados);
      }
      if (TextoPOST.substring(TextoPOST.indexOf("Horario=") + 8) != "") {
        Horario = TextoPOST.substring(TextoPOST.indexOf("Horario=") + 8,
TextoPOST.indexOf("&Obs="));
        Horario.replace("T", " ");
        Horario.replace("%3A", ":");
      }
      if (TextoPOST.substring(TextoPOST.indexOf("Obs=") + 4) != "") {
        OBS = TextoPOST.substring(TextoPOST.indexOf("Obs=") + 4);
        OBS.replace("%23", " ");
        OBS = DecodeURL(OBS);
      }
      //Limpa variável TextoPOST
      TextoPOST = "";
      DadosRealizarManutencao = true;
```

```
    }
    FormularioManutencao(client);
  } else if (req.indexOf("GravarTag") != -1) {
    TelaGravarTag(client);
  } else {
    client.println("Requisicao Invalida");
  }
  //Realiza procedimento para encerrar a comunicação HTTP e se preparar para uma
nova requisição
  client.flush();
  client.stop();
  //Finaliza os processos do modulo RC522
  mfrc522.PICC_HaltA();
  mfrc522.PCD_StopCrypto1();
}
void TelaInicial(WiFiClient client) {
  /*Função TelaInicial(WiFiClient client)
    Objetivo:
      Construir a tela inicial do sistema.
    Parâmetros:
      WiFiClient client -> Dispositivo conectado, onde será construído a tela.
    Retorna:
      Nada.
  */
  //Constrói o cabeçalho e o css da pagina
  Cabecalho(client);
  //Apresenta os dados e o botão de realizar operação caso os dados do operador já
estiverem preenchidos
  if (DadosOperador == true) {
    DadosOperadorPreechido(client);
    client.println("<a href=\"RealizarOpera&ccedil;&atilde;o\"><button>Realizar
Operacao</button></a>");
  }
  client.println("<a href=\"DadosOperador\"><button>Preencher dados do
operador</button></a>");
  client.println("</body>");
  client.println("</html>");
  client.println();
}
void TelaDadosOperador(WiFiClient client) {
  /*Função TelaDadosOperador(WiFiClient client)
    Objetivo:
      Construir a tela de formulário para preenchimento dos dados do operador.
    Parâmetros:
      WiFiClient client -> Dispositivo conectado, onde será construído a tela.
    Retorna:
      Nada.
  */
```

```
  //Constrói o cabeçalho e o css da pagina
  Cabecalho(client);
  //Apresenta os dados do operador caso já estiverem preenchidos
  if (DadosOperador == true) {
    DadosOperadorPreechido(client);
  }
  //Constrói formulário dos dados do operador
  client.println("<form method=\"POST\">");
  client.println("<h2>Dados do operador</h2>");
  client.println("<h3>Nome do operador</h2><input type=\"text\" name=\"Nome\"/>");
  client.println("<h3>Matr&iacute;cula</h2><input type=\"text\"
name=\"Matricula\">");
  client.println("<input type=\"submit\" id=\"meu-submit\" value=\"Enviar\"
/></form>");
  client.println("<a href=\"TelaInicial\"><button>Voltar para tela
inicial</button></a>");
  client.println("</body>");
  client.println();
}
void TelaDefinirOperacao(WiFiClient client) {
  /*Função TelaDefinirOperacao(WiFiClient client)
    Objetivo:
      Construir a tela de menu para escolha da operação que será realizada
    Parâmetros:
      WiFiClient client -> Dispositivo conectado, onde será construído a tela.
    Retorna:
      Nada.
  */
  //Constrói a tela caso os dados do operador estejam preenchidos, caso contrário
constrói uma tela de erro
  if (DadosOperador == true) {
    //Constrói o cabeçalho e o css da pagina
    Cabecalho(client);
    //Apresenta os dados do operador
    DadosOperadorPreechido(client);
    //Constrói menu
    client.println("<a href='LerManutencao'><button>Ler
Manuten&ccedil;&atilde;o</button></a>");
    client.println("<a href='RealizarManutencao'><button>Realizar
Manuten&ccedil;&atilde;o</button></a>");
    client.println("<a href='TelaInicial'><button>Voltar para tela
inicial</button></a>");
    client.println("</body>");
    client.println();
  } else {
    //Constrói o cabeçalho e o css da pagina
    Cabecalho(client);
    //Constrói mensagem de erro e botão de voltar
```

```
    client.println("<h2>Preencha os dados de operador!!</h2>");
    client.println("<a href='TelaInicial'><button>Voltar</button></a>");
    client.println("</body>");
  }
}
void TelaLerManutencao(WiFiClient client, boolean manutencao) {
  /*Função TelaLerManutencao(WiFiClient client, boolean manutencao)
    Objetivo:
      Construir a tela de leitura e chamar a função responsável pela leitura.
    Parâmetros:
      WiFiClient client -> Dispositivo conectado, onde será construído a tela.
      boolean manutencao -> Informa se a tela foi chamada na operação de leitura
ou de realizar manutenção.
        true : realizar manutenção
        false : leitura da última manutenção.
    Retorna:
      Nada.
  */
  //Constrói a tela caso os dados do operador estejam preenchidos, caso contrário
constrói uma tela de erro
  if (DadosOperador == true) {
    //Constrói o cabeçalho e o css da página.
    Cabecalho(client);
    //Apresenta os dados do operador.
    DadosOperadorPreechido(client);
    //Solicita a aproximação da etiqueta e chama a função LerTag.
    client.println("<p>Aproxime o equipamento da etiqueta...</p>");
    LerTag();
    //A partir a construção da tela só vai ocorrer após a leitura da etiqueta.
    client.println("<p>Etiqueta Lida !!!</p>");
    //Cria o botão da próxima página de acordo com o valor da variável manutenção
    if (manutencao == false) {
      client.println("<a href='Comprovante'><button>Gerar
Comprovante</button></a)");
      TipoComprovante = 1;
    } else {
      client.println("<a href='FormularioManutencao'><button>Preencher
formulario</button></a)");
    }
    client.println("</body>");
    client.println();
  } else {
    //Constrói o cabeçalho e o css da página.
    Cabecalho(client);
    //Constrói mensagem de erro e botão de voltar.
    client.println("<h1>Sistema de manuten&ccedil;&atilde;o</h1>");
    client.println("<h2>Preencha os dados de operador!!</h2>");
    client.println("<a href='TelaInicial'><button>Voltar</button></a>");
```

```
  }
}
void TelaComprovante(WiFiClient client) {
  /*Função TelaComprovante(WiFiClient client)
    Objetivo:
      Construir a tela de comprovante de leitura ou de gravação.
    Parâmetros:
      WiFiClient client -> Dispositivo conectado, onde será construído a tela.
    Retorna:
      Nada.
  */
  //Constrói o cabeçalho e o css da página.
  Cabecalho(client);
  /*Identifica por meio da variável global TipoComprovante se o comprovante é de
leitura ou de gravação:
    TipoComprovante == 1: Comprovante de leitura.
    TipoComprovante == 2: Comprovante de gravação.
  */
  if (TipoComprovante == 1) {
    client.println("<h2>Comprovante de leitura de manuten&ccedil;&atilde;o</h2>");
    //Apresenta o horário que o comprovante foi gerado com auxílio do código
javascript no final da página.
    client.println("<p id=\"horario\">");
    //Apresenta os dados do responsável pela leitura.
    client.println("<p>Operador respons&aacute;vel pela leitura:</p>");
    client.println("<p>Nome: " + NomeOperador + "</p>");
    client.println("<p>Matr&iacute;cula: " + MatriculaOperador + "</p>");
    client.println("<p>Dados do equipamento: </p>");
    //Apresenta os dados do equipamento e da última manutenção.
    client.println("<p>ID:" + IDTag + "</p>");
    client.println("<p>Dados de manuten&ccedil;&atilde;o: </p>");
    client.println("<p>Operador: " + Tag.substring(0, Tag.indexOf("#")) + "</p>");
    Tag = Tag.substring(Tag.indexOf("#") + 1);
    client.println("<p>Matr&iacute;cula do Operador: " + Tag.substring(0,
Tag.indexOf("#")) + "</p>");
    Tag = Tag.substring(Tag.indexOf("#") + 1);
    client.println("<p>Tipo de Manuten&ccedil;&atilde;o: " + Tag.substring(0,
Tag.indexOf("#")) + "</p>");
    Tag = Tag.substring(Tag.indexOf("#") + 1);
    client.println("<p>Itens substituidos: " + Tag.substring(0, Tag.indexOf("#"))
+ "</p>");
    Tag = Tag.substring(Tag.indexOf("#") + 1);
    client.println("<p>Itens com avaria: " + Tag.substring(0, Tag.indexOf("#")) +
"</p>");
    Tag = Tag.substring(Tag.indexOf("#") + 1);
    client.println("<p>Hor&aacute;rio: " + Tag.substring(0, Tag.indexOf("#")) +
"</p>");
    Tag = "";
```

```
    TipoComprovante = 0;
  } else if (TipoComprovante == 2) {
    client.println("<h2>Comprovante de grava&ccedil;&atilde;o de
manuten&ccedil;&atilde;o</h2>");
    //Apresenta o horário que o comprovante foi gerado com auxílio do código
javascript no final da página.
    client.println("<p id=\"horario\">");
    //Apresenta os dados do responsável pela leitura.
    client.println("<p>Operador responsavel pela leitura:</p>");
    client.println("<p>Nome: " + NomeOperador + "</p>");
    client.println("<p>Matr&iacute;cula: " + MatriculaOperador + "</p>");
    client.println("<h3>Manuten&ccedil;&atilde;o anterior</h3>");
    //Apresenta os dados do equipamento e da última manutenção antes da gravação.
    client.println("<p>Dados do equipamento: </p>");
    client.println("<p>ID:" + IDTag + "</p>");
    client.println("<p>Dados de manuten&ccedil;&atilde;o: </p>");
    client.println("<p>Operador: " + Tag.substring(0, Tag.indexOf("#")) + "</p>");
    Tag = Tag.substring(Tag.indexOf("#") + 1);
    client.println("<p>Matr&iacute;cula do Operador: " + Tag.substring(0,
Tag.indexOf("#")) + "</p>");
    Tag = Tag.substring(Tag.indexOf("#") + 1);
    client.println("<p>Tipo de Manuten&ccedil;&atilde;o: " + Tag.substring(0,
Tag.indexOf("#")) + "</p>");
    Tag = Tag.substring(Tag.indexOf("#") + 1);
    client.println("<p>Itens substituidos: " + Tag.substring(0, Tag.indexOf("#"))
+ "</p>");
    Tag = Tag.substring(Tag.indexOf("#") + 1);
    client.println("<p>Itens com avaria: " + Tag.substring(0, Tag.indexOf("#")) +
"</p>");
    Tag = Tag.substring(Tag.indexOf("#") + 1);
    client.println("<p>Hor&aacute;rio: " + Tag.substring(0, Tag.indexOf("#")) +
"</p>");
    Tag = "";
    TipoComprovante = 0;
    //Apresenta os dados da última manutenção após a gravação.
    client.println("<h3>Dados da nova manuten&ccedil;&atilde;o</h3>");
    client.println("<p>Operador responsavel pela gravacao:</p>");
    client.println("<p>Nome: " + NomeOperador + "</p>");
    client.println("<p>Matr&iacute;cula: " + MatriculaOperador + "</p>");
    client.println("<p>Dados da manuten&ccedil;&atilde;o: </p>");
    client.println("<p>Tipo: " + TipoOperacao + "</p>");
    client.println("<p>Itens Substituidos: " + ItensSubstituidos + "</p>");
    client.println("<p>Itens Avariados: " + ItensAvariados + "</p>");
    client.println("<p>Hor&aacute;rio da manuten&ccedil;&atilde;o: " + Horario +
"</p>");
    client.println("<p>Observa&ccedil;&otilde;es: " + OBS + "</p>");
    TipoComprovante = 0;
  }
```

```
  //Cria botão para imprimir, que utiliza o javascript no final da página.
  client.println("<button id='imprimir'>Imprimir</button>");
  client.println("<a href='RealizarOperacao'><button>Voltar</button></a>");
  //Inicio do javascript.
  client.println("<script>");
  //Criando o comportamento de imprimir a tela para o botão de imprimir.
  client.println("const btn_imprimir = document.getElementById(\"imprimir\")");
  client.println("btn_imprimir.addEventListener(\"click\",(evt)=>{window.print()})
");
  //Criando o comportamento de apresentar o horário na tela
  client.println("function mostrarHorario() {");
  client.println("var dataAtual = new Date();");
  client.println("var hora = dataAtual.getHours();");
  client.println("var minuto = dataAtual.getMinutes();");
  client.println("var segundo = dataAtual.getSeconds();");
  client.println("var dia = dataAtual.getDate();");
  client.println("var mes = dataAtual.getMonth() + 1;");
  client.println("var ano = dataAtual.getFullYear();");
  client.println("var horaAtualString = dia +'-'+ mes + '-' + ano + ' as ' + hora
+ ':' + minuto + ':' + segundo;");
  client.println("document.getElementById(\"horario\").innerText = \"Este
comprovante foi gerado na data: \" + horaAtualString;");
  client.println("}");
  client.println("mostrarHorario()");
  client.println("</script>");
  client.println("</body>");
}
void FormularioManutencao(WiFiClient client) {
  /*Função FormularioManutencao(WiFiClient client)
    Objetivo:
      Construir a tela de formulário de manutenção para gravar na etiqueta
    Parâmetros:
      WiFiClient client -> Dispositivo conectado, onde será construído a tela.
    Retorna:
      Nada.
  */
  //Constrói o cabeçalho e o css da página.
  Cabecalho(client);
  //Constrói a tela caso os dados do operador estejam preenchidos, caso contrário
constrói uma tela de erro.
  if (DadosOperador == true) {
    DadosOperadorPreechido(client);
    //Apresenta os dados do formulário para gravação caso o formulário já tenha
sido preenchido ao menos uma vez.
    if (DadosRealizarManutencao == true) {
      client.println("<p>Dados da manuten&ccedil;&atilde;o preechido:</p>");
      client.println("<p>Tipo: " + TipoOperacao + "</p>");
      client.println("<p>Itens Substituidos: " + ItensSubstituidos + "</p>");
```

```
      client.println("<p>Itens Avariados: " + ItensAvariados + "</p>");
      client.println("<p>Horario da manuten&ccedil;&atilde;o: " + Horario +
"</p>");
      client.println("<p>Observa&ccedil;&otilde;es: " + OBS + "</p>");
      client.println("<a href='GravarTag'><button>Gravar na
etiqueta</button></a>");
      client.println("<h2>Modifique os dados de manuten&ccedil;&atilde;o</h2>");
    } else {
      client.println("<h2>Insira os dados de manuten&ccedil;&atilde;o</h2>");
    }
    //Constrói o formulário.
    client.println("<form method=\"POST\">");
    client.println("<h3>Tipo de manuten&ccedil;&atilde;o</h3><input type='text'
name='Tipo' placeholder='Tipo de Manuten&ccedil;&atilde;o'/>");
    client.println("<h3>Itens Substituidos(cod. separados por
espa&ccedil;o)</h3><input type='text' name='ItensS' placeholder='Itens
Substituidos'>");
    client.println("<h3>Itens com Avaria(cod. separados por
espa&ccedil;o)</h3><input type='text' name='ItensA' placeholder='Itens com
Avaria'>");
    client.println("<h3>Dia e hor&aacuterio de manuten&ccedil;&atilde;o</h3><input
type='datetime-local' name='Horario'>");
    client.println("<h3>Observa&ccedil;&otilde;es</h3><textarea
name='Obs'></textarea>");
    client.println("<input type='submit' id='meu-submit' value='Enviar'
/></form>");
    client.println("<a href='RealizarOperacao'><button>Voltar</button></a>");
    client.println("</body>");
    client.println();
  } else {
    //Constrói mensagem de erro e botão de voltar.
    client.println("<h1>Sistema de manuten&ccedil;&atilde;o</h1>");
    client.println("<h2>Preencha os dados de operador!!</h2>");
    client.println("<a href='TelaInicial'><button>Voltar</button></a>");
  }
}
void TelaGravarTag(WiFiClient client) {
  /*Função TelaGravarTag(WiFiClient client)
    Objetivo:
      Construir a tela de gravação de manutenção na etiqueta.
    Parâmetros:
      WiFiClient client -> Dispositivo conectado, onde será construído a tela.
    Retorna:
      Nada.
  */
  //Constrói a tela caso os dados do operador estejam preenchidos, caso contrário
constrói uma tela de erro.
  if (DadosOperador == true) {
```

```
    //Constrói o cabeçalho e o css da página.
    Cabecalho(client);
    //Apresenta os dados do operador.
    DadosOperadorPreechido(client);
    //Solicita a aproximação da etiqueta e chama a função EscreverTag.
    client.println("<p>Aproxime o equipamento da etiqueta...</p>");
    EscreverTag();
    //A partir a construção da tela só vai ocorrer após a gravação da etiqueta.
    client.println("<p>Informa&ccedil;&atilde;o gravada na etiqueta!!!</p>");
    DadosRealizarManutencao = false;
    //Constrói botão para geração do comprovante.
    client.println("<a href='Comprovante'><button>Gerar
comprovante</button></a>");
    TipoComprovante = 2;
    client.println("</body>");
  } else {
    //Constrói o cabeçalho e o css da página.
    Cabecalho(client);
    //Constrói mensagem de erro e botão de voltar.
    client.println("<h1>Sistema de manuten&ccedil;&atilde;o</h1>");
    client.println("<h2>Preencha os dados de operador!!</h2>");
    client.println("<a href='TelaInicial'><button>Voltar</button></a>");
  }
}
void Cabecalho(WiFiClient client) {
  /*Função Cabecalho(WiFiClient client)
    Objetivo:
      Construir o cabeçalho em html, contando a folha de estilos utilizadas em
todas as telas.
    Parâmetros:
      WiFiClient client -> Dispositivo conectado, onde será construído a tela.
    Retorna:
      Nada.
  */
  client.println("HTTP/1.1 200 OK");
  client.println("Content-type:text/html");
  client.println("");
  client.println("<!DOCTYPE html>");
  client.println("<html>");
  client.println("<head>");
  client.println("<meta charset=\"uft-8\" />");
  client.println("<meta name=\"viewport\" content=\"width=device-width\" />");
  client.println("<title>Tela Inicial</title>");
  client.println("<style>");
  client.println("body{");
  client.println("position: absolute;");
  client.println("top: 0px;");
  client.println("left: 0px;");
```

```
  client.println("width: 100%;");
  client.println("margin: 0;");
  client.println("padding: 0;");
  client.println("text-align: center;");
  client.println("}");
  client.println("h1{");
  client.println("background-color: black;");
  client.println("color: white;");
  client.println("margin:0px;");
  client.println("width: 100%;");
  client.println("padding: 2%;");
  client.println("font-size: 50px;");
  client.println("}");
  client.println("button{");
  client.println("width: 50%;");
  client.println("padding: 1%;");
  client.println("margin-top: 3%;");
  client.println("margin-bottom: 3%;");
  client.println("}");
  client.println("</style>");
  client.println("</head>");
  client.println("<body>");
  client.println("<h1>Sistema de manuten&ccedil;&atilde;o</h1>");
}
void DadosOperadorPreechido(WiFiClient client) {
  /*Função DadosOperadorPreechido(WiFiClient client)
    Objetivo:
      Construir a apresentação dos dados do operador, utilizado na maioria das
telas.
    Parâmetros:
      WiFiClient client -> Dispositivo conectado, onde será construído a tela.
    Retorna:
      Nada.
  */
  client.println("<p>Dados do Operador preenchidos:</p>");
  client.println("<p>Operador: " + NomeOperador + "</p>");
  client.println("<p>Matr&iacute;cula: " + MatriculaOperador + "</p>");
  client.println("<p></p>");
}
void LerTag() {
  /*Função LerTag()
    Objetivo:
      Realizar a comunicação com o modulo RC522 e realizar a leitura de todas as
informações da etiqueta.
    Parâmetros:
    Retorna:
      Nada.
  */
```

```
  //Limpa a variável global Tag antes da leitura
  Tag = "";
  //Repete o código para todos os blocos da memória interna da etiqueta.
  for (byte bloco = 1; bloco < 64; bloco++) {
    //Inicia a variável taglida como false
    boolean taglida = false;
    //Enquanto a etiqueta não for lida o código deve se repetir.
    while (taglida == false) {
      //Verifica se existe uma etiqueta presente e se é possível realizar a
leitura.
      if (mfrc522.PICC_IsNewCardPresent() && mfrc522.PICC_ReadCardSerial()) {
        mfrc522.PICC_DumpDetailsToSerial(&(mfrc522.uid));
        //Prepara as variaveis para leitura.
        for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
        byte buffer_read[SIZE_BUFFER] = { 0 };
        byte tamanho = SIZE_BUFFER;
        boolean erro = false;
        //Tenta autenticar a etiqueta.
        status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, bloco,
&key, &(mfrc522.uid));
        //Se a autenticação der erro, apresente na Serial, caso contrário realize
a leitura.
        if (status != MFRC522::STATUS_OK) {
          Serial.print(F("Authentication failed: "));
          Serial.println(mfrc522.GetStatusCodeName(status));
        } else {
          //Enquanto não chegar no último bloco ou de erro, tente ler as
informações da etiqueta
          while (erro == false && bloco < 64) {
            //Verifica se o bloco é um bloco de gravação e leitura.
            if ((bloco + 1) % 4 != 0) {
              //Realiza a leitura.
              status = mfrc522.MIFARE_Read(bloco, buffer_read, &tamanho);
              //Verifica se a leitura deu erro.
              if (status != MFRC522::STATUS_OK) {
                Serial.print(F("Reading failed: "));
                Serial.println(mfrc522.GetStatusCodeName(status));
                //Muda a variável erro para verdadeiro e volta um bloco.
                erro = true;
                bloco = bloco - 1;
              } else {
                Serial.print(F("\nDados bloco ["));
                Serial.print(bloco);
                Serial.print(F("]: "));
                //Grava os dados lidos na variável global valortag;
                for (uint8_t i = 0; i < MAX_SIZE_BLOCK; i++) {
                  valortag[i] = buffer_read[i];
                }
```

```
                //Grava o ID da etiqueta caso seja o primeiro bloco.
                if (bloco == 1) {
                  for (byte i = 0; i < mfrc522.uid.size; i++) {
                    Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
                    Serial.print(mfrc522.uid.uidByte[i], HEX);
                    IDTag.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : "
"));
                    IDTag.concat(String(mfrc522.uid.uidByte[i], HEX));
                  }
                }
                //Grava os dados lidos em String e grava na variável global Tag.
                String str = (char *)valortag;
                str = str.substring(0, 15);
                Serial.println(str);
                Tag += str;
                //Muda o valor da variável taglida para verdadeiro.
                taglida = true;
                //Avança para o próximo bloco.
                bloco++;
              }
            } else {
              //Caso não seja um bloco de leitura e gravação, pule o bloco.
              bloco++;
            }
          }
        }
        //Finalize a comunicação com o modulo RC522
        mfrc522.PICC_HaltA();
        mfrc522.PCD_StopCrypto1();
      }
    }
  }
  return;
}
void EscreverTag() {
  /*Função EscreverTag()
    Objetivo:
      Realizar a comunicação com o modulo RC522 e realizar a gravação de todas as
informações da etiqueta.
    Parâmetros:
    Retorna:
      Nada.
  */
  //Concatena todas as informações que devem ser gravadas na etiqueta adicionando
um # entre elas.
  String EnvioTag = NomeOperador + "#" + MatriculaOperador + "#" + TipoOperacao +
"#" + ItensSubstituidos + "#" + ItensAvariados + "#" + Horario;
```

```
  //Verifica se o tamanho dos dados gravados não ultrapassa o valor máximo
permitido.
  if (EnvioTag.length() > 752) {
    Serial.println("os dados tem mais de 752 bytes");
    return;
  }
  /*Adiciona espaços em branco caso tenha mais informações gravadas na etiqueta
que informações que vão ser gravadas.
  Caso seja verdade, as informações que vão ser gravadas recebem espaços em branco
afim de deixar a etiqueta somente
  com informações referentes a essa manutenção, sem pedaços de informações de
outras manutenções.
  */
  while (EnvioTag.length() < Tag.length()) {
    EnvioTag += " ";
  }
  Serial.println(EnvioTag);
  //Repete o código para todos os blocos da memória interna da etiqueta.
  for (byte bloco = 1; bloco < 64; bloco++) {
    //Inicia a variável taglida como false
    boolean taglida = false;
    //Enquanto a etiqueta não for lida e ainda tiver informações a serem gravadas.
    while (taglida == false && EnvioTag.length() > 0) {
      //Verifica se existe uma etiqueta presente e se é possível realizar a
leitura.
      if (mfrc522.PICC_IsNewCardPresent() && mfrc522.PICC_ReadCardSerial()) {
        mfrc522.PICC_DumpDetailsToSerial(&(mfrc522.uid));
        //Prepara as variáveis para leitura.
        for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
        //Tenta autenticar a etiqueta.
        status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, bloco,
&key, &(mfrc522.uid));
        //Se a autenticação der erro, apresente na Serial, caso contrário realize
a gravação.
        if (status != MFRC522::STATUS_OK) {
          Serial.print(F("PCD_Authenticate() failed: "));
          Serial.println(mfrc522.GetStatusCodeName(status));
        } else {
          boolean erro = false;
          String aux;
          //Enquanto não chegar no último bloco ou de erro, tente gravar as
informações da etiqueta
          while (erro == false && bloco < 64) {
            //Verifica se o bloco é um bloco de gravação e leitura.
            if ((bloco + 1) % 4 != 0) {
              //Caso as informações restantes para gravação tenham menos de 16
caracteres.
              if (EnvioTag.length() < 16) {
```

```
                //Transforme a String para vetor de caracteres.
                EnvioTag.toCharArray((char *)buffer, MAX_SIZE_BLOCK);
                //Preencha o vetor com espaços até ter 16 caracteres.
                for (byte i = EnvioTag.length(); i < MAX_SIZE_BLOCK; i++) {
                  buffer[i] = ' ';
                  taglida = true;
                  erro = true;
                }
              } else {
                //Grave os próximos 16 caracteres das informações para gravação em
uma String auxiliar.
                aux = EnvioTag.substring(0, MAX_SIZE_BLOCK);
                //Transforme a String para vetor de caracteres.
                aux.toCharArray((char *)buffer, MAX_SIZE_BLOCK);
              }
              //Realize a gravação das informações.
              status = mfrc522.MIFARE_Write(bloco, buffer, MAX_SIZE_BLOCK);
              //Verifique se gravação ocorreu corretamente.
              if (status != MFRC522::STATUS_OK) {
                Serial.print(F("MIFARE_Write() failed: "));
                Serial.println(mfrc522.GetStatusCodeName(status));
                //Em caso de erro, mude o valor da variável erro para verdadeiro e
volte um bloco.
                bloco = bloco - 1;
                erro = true;
              } else {
                Serial.println(F("MIFARE_Write() success: "));
                if (EnvioTag.length() < 16) {
                  EnvioTag = "";
                } else {
                  EnvioTag = EnvioTag.substring(MAX_SIZE_BLOCK - 1);
                }
                //Avance um bloco e mude o valor da variável taglida para
verdadeiro.
                bloco++;
                taglida = true;
              }
            } else {
              //Avance um bloco.
              bloco++;
            }
          }
        }
        mfrc522.PICC_HaltA();
        mfrc522.PCD_StopCrypto1();
      }
    }
  }
```

```
}
void LimpaVariaveis() {
  /*Função LimpaVariaveis()
    Objetivo:
      Limpa as variáveis que são utilizadas durante uma operação,
      afim de não interferir na próxima operação realizada pelo usuário.
    Parâmetros:
    Retorna:
      Nada.
  */
  TipoOperacao = "";
  ItensSubstituidos = "";
  ItensAvariados = "";
  Horario = "";
  OBS = "";
  Tag = "";
  IDTag = "";
  DadosRealizarManutencao = false;
}
String DecodeURL(String str) {
  /*Função DecodeURL(String str)
    Objetivo:
      Decodifica um texto em padrão url.
    Parâmetros:
      String str -> Texto a ser decodificado.
    Retorna:
      String -> Texto decodificado.
  */
  //Cria uma variável que irá receber o texto decodificado.
  String encodedString = "";
  char c;
  char code0;
  char code1;
  //Repita o código para cada caractere do texto codificado.
  for (int i = 0; i < str.length(); i++) {
    //Grava o caractere atual na variável c
    c = str.charAt(i);
    //Troca + por espaço
    if (c == '+') {
      encodedString += ' ';
    } else if (c == '%') {
      //Caso seja um caractere codificado, pegue os dois próximos caracteres e
chame a função h2int.
      i++;
      code0 = str.charAt(i);
      i++;
      code1 = str.charAt(i);
      //Concatena o retorno da função h2int para code0 e code1 na variável c.
```

```
      c = (h2int(code0) << 4) | h2int(code1);
      encodedString += c;
    } else {
      encodedString += c;
    }
  }
  //Retorne o texto decodificado.
  return encodedString;
}
unsigned char h2int(char c) {
  /*Função h2int(char c)
    Objetivo:
      Transformar valores em hexadecimal para decimal.
    Parâmetros:
      char c -> Valor que vai ser transformado.
    Retorna:
      unsingned char -> Valor transformado.
  */
  //Se o valor hexadecimal estiver entre 1 e 9, retorne ele.
  if (c >= '0' && c <= '9') {
    return ((unsigned char)c - '0');
  }
  //Se o valor hexadecimal estiver entre a e f, retorne ele subtraindo 'a' e
somando 10.
  if (c >= 'a' && c <= 'f') {
    return ((unsigned char)c - 'a' + 10);
  }
  //Se o valor hexadecimal estiver entre A e F, retorne ele subtraindo 'a' e
somando 10.
  if (c >= 'A' && c <= 'F') {
    return ((unsigned char)c - 'A' + 10);
  }
  return (0);
}
```