INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DO RIO GRANDE DO NORTE, CAMPUS NATAL - ZONA NORTE.

DAVI FREIRE PEREIRA DA SILVA
ISAQUE DANIEL JORGE DA SILVA
MATHEUS LUCCA FERREIRA DA SILVA

CONTROLADOR DE ESTOQUE ELETRÔNICO PARA LABORATÓRIOS

NATAL/RN
2021

DAVI FREIRE PEREIRA DA SILVA
ISAQUE DANIEL JORGE DA SILVA
MATHEUS LUCCA FERREIRA DA SILVA

CONTROLADOR DE ESTOQUE ELETRÔNICO PARA LABORATÓRIOS

Trabalho de Conclusão de Curso realizado por alunos do Curso de Nível Médio em Eletrônica, do Instituto de Educação, Ciência e tecnologia do Rio Grande do Norte, em cumprimento às exigências legais como requisito para a obtenção do título de Técnico em Eletrônica.
Orientador: Arthur Salgado de Medeiros

NATAL/RN
2021

DAVI FREIRE PEREIRA DA SILVA
ISAQUE DANIEL JORGE DA SILVA
MATHEUS LUCCA FERREIRA DA SILVA

CONTROLADOR DE ESTOQUE ELETRÔNICO PARA LABORATÓRIOS

Trabalho de Conclusão de Curso realizado por alunos do Curso de Nível Médio em Eletrônica, do Instituto de Educação, Ciência e tecnologia do Rio Grande do Norte, em cumprimento às exigências legais como requisito para a obtenção do título de Técnico em Eletrônica.

Trabalho de Conclusão de Curso apresentado em: XX/03/2021

Prof. Arthur Salgado de Medeiros, - Professor Orientador
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte
Matrícula: 1829355

Prof. Daniel Guerra, - Coordenador do Curso Técnico Integrado em Eletrônica
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte
Matrícula: 1281383

AGRADECIMENTOS

Primeiramente, digo que essa longa jornada composta por quatro longos anos nos tornaram as pessoas que somos hoje, todo o conhecimento adquirido nesse tempo inesquecível de nossas vidas será utilizado em nossos futuros acadêmico e profissionais. Queremos agradecer a Deus e aos nossos familiares e amigos que estiveram com a gente nos apoiando nos momentos mais difíceis tanto como nos agradáveis.

Em segundo lugar, devemos agradecer por fazer parte da família “Eletronika” e aos nossos professores que se esforçaram e se dedicaram a nos passar aprendizados, em especial Daniel Guerra, que esteve junto conosco praticamente durante toda a nossa jornada, ao nosso antigo orientador Alysson Holanda, que infelizmente foi remanejado e deixou o nosso IF. Ao professor Arthur Salgado, que nos deu a devida orientação para o término do trabalho de conclusão de curso, substituindo Alysson, e ao professor Ickson Barbosa, que foi nosso orientador no Projeto integrador e nos forneceu ensinos indispensáveis para esse projeto.

E por último, aos nossos colegas de classe que dividiram nossas experiências acadêmicas com a gente, construindo grandes laços que serão sempre lembrados, inclusive os que ficaram para trás nessa jornada como Ray Torres, que mesmo assim continuo mantendo os laços fortes entre nós. E agradecer em especial para Adrian Cruz e Talison Fabio, que apoiaram e auxiliaram esse projeto diversas vezes e que sempre foram grandes companheiros não só nas horas de entretenimento como nas horas mais necessárias.

RESUMO

O projeto trata-se de uma forma de gerenciamento de estoque, por meio de um dispositivo de contagem de itens e através do armazenamento de dados em um servidor, traduzindo assim em um maior controle no estoque, de qualquer tipo que ele seja. A ideia surgiu de forma espontânea ao ver a maneira em que estava organizada a sala da Coordenação de Laboratórios, onde estão armazenados componentes eletrônicos e objetos de manipulação para os cursos técnicos de Eletrônica e Informática. A sala de condenação tem uma grande representatividade no setor, pois é o local em que alunos dos cursos técnicos conseguem retirar componentes e manipulá-los para a confecção de circuitos e placas, porém ela apresenta um problema na organização de seu estoque de itens, já que a forma de retirada e organização dos objetos é convencional, não trazendo mobilidade, flexibilidade e acima de tudo segurança na retirada dos itens pelos alunos, por isso a necessidade da construção de um dispositivo de gerenciamento de estoques.

Palavras-Chave: Sistema, RFID, Programação, Dados, Controle de Estoque.

ABSTRACT

The project is a form of inventory management, through a device for counting items and storing data on a server, thus translating into greater control over inventory, of whatever type it is. The idea came about spontaneously when seeing the way in which the laboratory coordination room was organized, where electronic components and manipulation objects are stored for technical courses in electronics and computers. The condemnation room is highly representative in the sector, as it is the place where students of technical courses are able to remove components and manipulate them for the manufacture of circuits and plates, but it presents a problem in the organization of its stock of items, since the way of removing and organizing objects is conventional, not bringing mobility, flexibility and, above all, security in the removal of items by students. Hence the need to build an inventory management device.

Keywords: System, RFID, Programming, Data, Inventory Control.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

É sabido que, em uma empresa de qualquer porte, o estoque de produtos é um setor que demanda muito esforço, com o objetivo de assegurar a quantidade e integridade dos produtos. Com a reserva de componentes do Instituto Federal não deveria ser diferente, porém quando vamos para a realidade, é visto com clareza que não há uma forma objetiva e sistemática que traga segurança e ordem para com os itens da Coordenação de Laboratório do IFRN Campus Natal Zona Norte.

Vale destacar que a Coordenação de Laboratórios recebe mais de 100 estudantes todos os dias, já que há uma necessidade de retirada dos componentes eletrônicos. A forma convencional de organização consiste em apenas obter o componente que o aluno deseja e assinar um termo de responsabilidade em um papel, com todos os componentes pegos pelo aluno. Entretanto, é perceptível o quão falho é esse sistema, já que pode haver qualquer erro na escrita do papel, o papel pode não ser conferido pelo responsável da coordenação e entre diversas outras possibilidades que um sistema não objetivo de organização apresenta.

Visto isso, é nítido que o sistema da coordenação de laboratório precisa ser melhorado, com a implementação de um meio que possa guardar a quantidade de itens e diminuir as perdas em geral, assim gerenciando de forma eficiente o balanço do início e do fim do dia, trazendo mais mobilidade e segurança tanto para os alunos como para a coordenação, e também não retirando o fator responsabilidade dos alunos. Por isso a necessidade do dispositivo controlador de estoque, que irá atuar no gerenciamento de itens da COLAB, coordenação de laboratórios.

O presente trabalho objetiva a criação de um dispositivo eletrônico focado no gerenciamento de estoque, baseado em uma plataforma microcontrolada e auxiliada por um sistema RFID. Há também o armazenamento desses dados em um servidor, o que torna viável a visualização das quantidades pelo responsável da coordenação. As formas utilizadas para a realização deste projeto serão

apresentadas no decorrer deste relatório, assim como as possíveis melhorias e aplicações futuras.

2. OBJETIVOS

2.1. OBJETIVOS GERAIS;

Desenvolver um dispositivo para sistematizar e agilizar o gerenciamento de estoques em geral; tornando possível um melhor aprimoramento para a gerência de reservas para empresas, escolas e centro de armazenamento, possibilitando uma menor taxa de erro humano na manipulação de estoques.

2.2. OBJETIVOS ESPECIFICOS;

- Auxiliar em futuros " Inventários de estoque ”: Possibilitando um melhor controle de entrada e saída, tendo assim uma maior exatidão do estoque em tempo real.
- Prevenir extravio de componentes: Já que para cada movimentação/uso do sistema será necessário utilizar a matrícula, sendo assim mais fácil identificar com quem/onde houve o desvio do caminho.
- Economizar papel: Não sendo necessário assinar papelada, o próprio sistema já organiza tudo.
- Otimizar o tempo de trabalho: Um sistema automatizado permite que não haja perda de tempo por qualquer erro humano.

3. JUSTIFICATIVA

Uma má gestão de estoque pode fazer com que grandes planejamentos sejam comprometidos: o controle e gerenciamento de estoque não é apenas uma atividade isolada, pelo contrário, seu funcionamento tem impactos diretos no capital de giro, ou seja impacta diretamente no poder financeiro das empresas e instituições. Sendo assim controlar o estoque de maneira eficaz é uma tarefa essencial nas empresas e instituições.

O controle de estoque é o ato de acompanhar a quantidade de produtos que a empresa tem, além de verificar quais produtos estão para vendas ou sendo utilizados internamente (que pode variar de ferramentas e acessórios a insumos

para fabricação). Em um contexto escolar, por exemplo, é o ato de acompanhar a quantidade de produtos de limpeza, merenda, ou material de apoio que a escola possui, tanto para uso interno, quanto para reserva (guardar para utilizar em outro período).

Este trabalho é a implementação de uma tecnologia que permite acompanhar o controle e gerenciamento de estoque com maior facilidade, tornando a visualização de dados mais simples e prática, tendo como benefício uma maior facilidade organizacional e gerencial, o que reduz o número de desperdício, furtos, assim o número de, desperdício, furtos, excesso de produtos, avarias, vencimentos, produtos obsoletos ou parados, entre outros. Evitar esses contratempos equivale a diminuir gastos, tanto de tempo, como financeiro.

Além disso, a ideia do gerenciador de estoque é aplicável não só em ambientes industriais, visto que a ideia se estende a várias situações. Uma das aplicações pensadas inicialmente foi em ambiente escolar, justamente por buscar uma melhoria na COLAB, do IFRN Campus Natal Zona Norte, porém suas aplicações podem ser úteis em ambientes hospitalares, agrícolas, entre outros. Em outras palavras, a aplicação é eficaz, prática e útil em diversos contextos e ambientes, possuindo assim uma alta versatilidade. Ademais, por possuir um baixo custo de fabricação, o controlador de estoque torna-se uma ferramenta viável e com baixo custo e alto benefício.

4. FUNDAMENTAÇÃO TEÓRICA

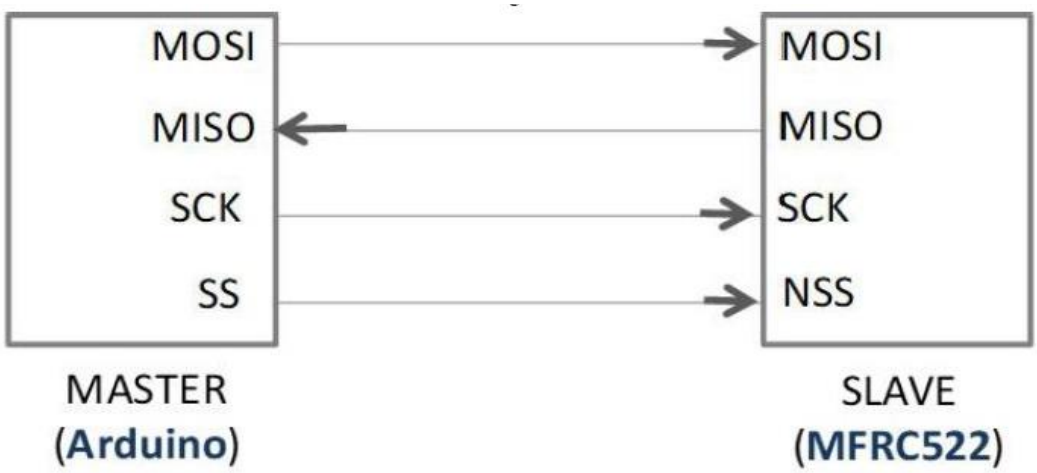
4.1. RFID e seu funcionamento

RFID é a sigla para *Radio Frequency Identification* (Identificação por Radiofrequência), ou seja, trata-se de um sistema que captura a frequência do sinal de rádio para a realização de tal tarefa. A captura desses dados é feita pelo módulo RFID(MFRC522) que capta os dados das etiquetas ou cartões, onde apresentam na sua constituição um circuito de transmissão de informações. Há dois tipos de etiquetas RF (radiofrequência), passiva e ativa: a primeira trata-se basicamente de um circuito que já detém gravado uma informação de fábrica; a ativa, por sua vez, contém um sistema mais aprimorado, em que cada cartão possui uma bateria própria, assim como um armazenamento de 32 KB, além de um alcance maior do que a etiqueta passiva.

A forma de comunicação do módulo RFID com o Arduino é realizada enviando bits de dados através de fios fisicamente conectados entre o módulo. Esse tipo de comunicação é chamado de SPI, *Serial Peripheral Interface* (Interface Periférica Serial), utilizada para transferência de dados em pequenas distâncias. A transferência é realizada bit a bit através de pulsos de voltagem. Como o Arduino opera com 5 Volts, um bit 0 é comunicado com um pulso curto de 0 Volt e um bit 1 é comunicado por um pulso curto de 5 Volts.

A comunicação SPI é baseada na ideia de “mestre-escravo”, pois funciona de uma forma que o mestre (Arduino) controla o escravo (RFID),pois toda a comunicação parte do princípio de trocar os dados do mestre com os escravos, bit por bit. O sinal de SS (Seleção do escravo ou em inglês *Slave selection*) da SPI funciona como um sinal ativo em nível baixo, nível baixo, em que o dispositivo é selecionado quando este pino se encontra em nível baixo. A Figura 1 abaixo demonstra de forma simples a função de cada registrador e suas caracterizações como “mestre” ou “escravo “.

Figura 1 - Esquemático da comunicação SPI.



Fonte: Retirado do Relatório do Projeto Integrador 2019.

O esquema de comunicação SPI é composto pelas seguintes portas:

- MOSI (*Master Output/Slave Input*): Porta para o *Master* enviar dados para o dispositivo *Slave.*
- MISO (*Master Input/Slave Output*): Porta para o dispositivo *Slave* enviar dados para o *Master.*

- SCK (*Serial Clock*): sinal de *Clock*. SS ou CS (*Slave Select* ou Chip *Select*): Porta para o *Master* selecionar o dispositivo *Slave.*

Com módulo MFRC 522 operando como S*lave*, o pino de *Slave Select* (SS) será o SDA. Em linhas gerais, os passos da comunicação SPI são os seguintes:

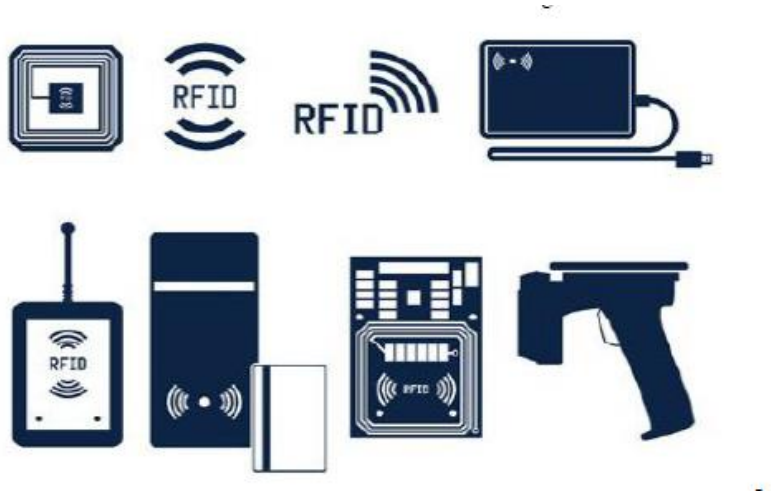
- O *Master* (Arduino) envia o sinal de *Clock*;
- O *Master* (Arduino) coloca a linha SS (SDA no MFRC522) em 0 V para ativar a seleção do *Slave* (MFRC522);
- O *Master* (Arduino) envia dados (sequência de bits) através da linha MOSI e o S*lave* (MFRC522) lê/recebe os bits transmitidos;
- Se uma resposta é necessária, o S*lave* (MFRC522) retorna dados (sequência de bits) para o *Master* (Arduino) através da linha MISO e o *Master* (Arduino) lê/recebe os bits transmitidos.

4.2. Funções do RFID

A partir de suas etiquetas RF (Radiofrequência), é possível aplicar o RFID em diversas áreas do cotidiano, como por exemplo o uso nos cartões de passagem em ônibus, metrôs e trens utilizando as *Tags* para armazenar dados como saldo, identificação do usuário e outras diversas informações. Um exemplo de aplicação que nos últimos anos se tornou cada vez mais comum, principalmente em grandes centros urbanos como São Paulo e Rio de Janeiro, consiste no pagamento de pedágio via um cartão com chip RFID.

Outra utilização frequente do cartão, é o cadastramento e identificação de funcionários em empresas e centro educativos. Também pode-se citar o rastreamento de animais, destravamento de automóveis, controle de grandes quantidades de estoque e até mesmo futuramente usado em leituras de códigos de barras. Portanto, a maior parte da utilização do sistema RFID é para registro de informações e identificação de dados, como já foi mostrado anteriormente. Representado a seguir na Figura 2.

Figura 2 - Ilustrativa das funções do RFID.



Fonte: Retirado do Relatório do Integrador.

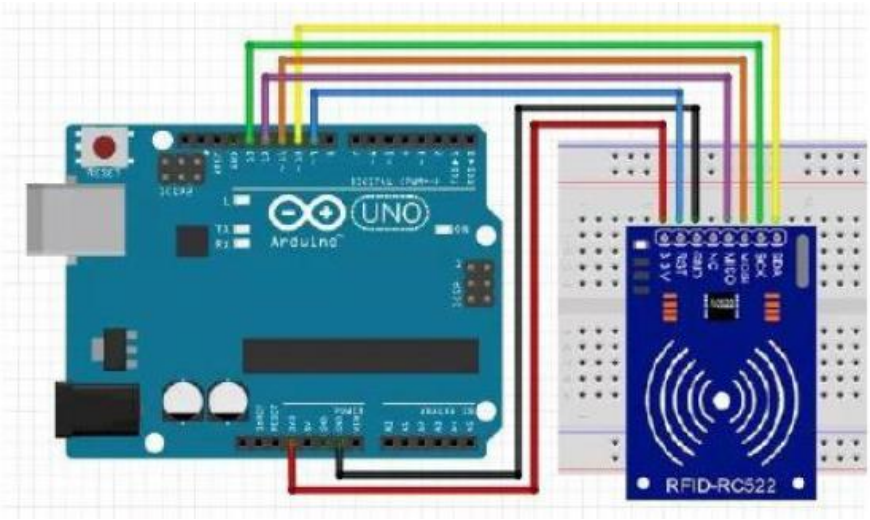
4.3. Arduino

O Arduino Uno é um dispositivo de prototipagem, baseado no microcontrolador ATmega328p. O dispositivo contém 14 pinos, em que 6 desses podem ser utilizados como saídas de PWM, além de entradas analógicas, com um cristal oscilador de 16MHz. O Arduino foi criado por cinco pesquisadores: Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino e David Mellis, com o objetivo de criar uma placa para prototipagem de custo baixo, além de uma linguagem de programação com relativa facilidade.

Pensando nas vantagens do Arduino “baixo custo, alta taxa de mobilidade e facilidade com a linguagem de programação, C ou C++”, a placa foi utilizada na prototipagem do projeto, tornando possível o desenvolvimento pela sua facilidade de manejo, além de possuir as conexões necessárias para a execução da comunicação do RFID.

Como já demonstrado anteriormente sobre o sistema RFID e suas formas de comunicação, é possível interpretar bem as formas de conexão do Arduino com o módulo RFID. Como demonstrado na Figura 3, os pinos estão conectados com as portas do Arduino para a realização da comunicação, para que haja a transmissão dos dados do dispositivo de transmissão de rádio frequência para a placa de prototipagem.

Figura 3 - Esquemático do Arduino.



Fonte: Autoria Própria – Tinkercad 2019.

O esquema de ligações consiste no seguinte:

Conexão Vermelha: Ligação para a porta de 3,3V.

Conexão Cinza: Ligação Para o *GND (graduated neutral density filter)* e conexão para a porta de *Reset.*

Conexão Laranja: Ligação da porta SDA a para porta A4.

Conexão Amarela: Ligação da porta SCK para a porta D13.

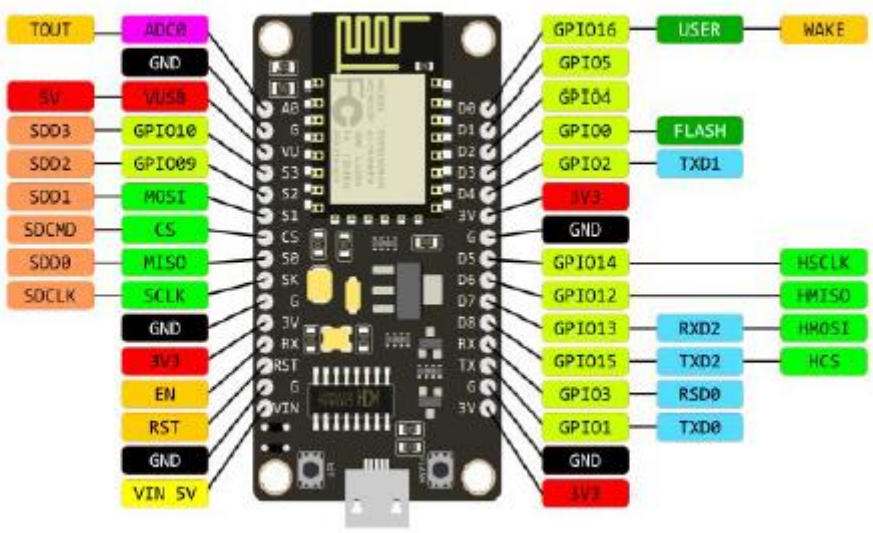
Conexão Azul: Ligação da porta MOSI para a porta D11.

Conexão Verde: Ligação da porta MISO para a porta D12.

4.4. ESP 8862

O ESP 8862 é um microcontrolador que se comunica através de Wi-Fi, sem a necessidade de uma rede com fios. Isso torna-o mais especial que o Arduino, pois o modelo clássico do Arduino UNO não tem a capacidade de se conectar a rede Wi-Fi sem fio. A necessidade de utilizar o ESP como mais um microcontrolador no projeto é pela carência de conexão do Arduino com a Wi-Fi. A necessidade de comunicação entre os dados recebidos pelo Arduino do RFID para enviar ao servidor torna imprescindível o uso do microcontrolador ESP 8862. A Figura 4 é um esquema das suas respectivas entradas.

Figura 4 - ESP 8862 e suas conexões.



Fonte: GitHub – Jotajr/esp8266-exemplos.

4.5. MIT app inventor

"O MIT App Inventor é uma ferramenta de programação baseada em blocos que permite que qualquer um comece a programar e construir aplicativos totalmente funcionais para dispositivos Android utilizando apenas um navegador *Web."* (MIT APP INVENTOR,2021). Essa ferramenta é disponibilizada pelo MIT (*Massachusetts Institute of Technology*), como o próprio nome diz. É possível acompanhar o desenvolvimento por um telefone conectado; além disso, os servidores do MIT App Inventor armazenam o trabalho em nuvem e ajudam a manter o controle dos seus projetos, concluídos ou em desenvolvimento. Para ter acesso ao MIT App Inventor, basta apenas acessar o portal web (https://appinventor.mit.edu/). Em seu portal, além das ferramentas de programação, são oferecidos tutoriais, apoio, suporte e a oportunidade de colaborar com o site.

4.6. Possibilidades do App Inventor

"Criar aplicativos para o seu telefone é divertido, e o App Inventor promove a exploração e a descoberta. Basta abrir o App Inventor em um navegador web, conectar seu telefone, e começar a montar blocos e você poderá interagir

imediatamente com o aplicativo que você está construindo no telefone.”(MIT APP INVENTOR, 2021).

As possibilidades da utilização do App inventor são infinitas, principalmente em ambiente acadêmico. Por ter uma linguagem de programação de blocos simples e intuitiva, o aprendizado de seu manuseio torna-se menos complexo que a maioria das outras plataformas de programação.

4.6.1. Protótipos

Protótipos são incompletos e não versões completas de suas ideias. Dito isto, o App Inventor pode servir como um bloco de notas eletrônico para desenvolvimento de aplicativos móveis. “Quando se tiver qualquer ideia, em vez de fazer anotações ou deixá-lo de lado, basta você começar um projeto de teste e salva-lo. E por ele arquivar seus projetos em nuvens seu armazenamento torna-se muito mais prático e seguro.”(MIT APP INVENTOR, 2021).

4.6.2. Desenvolver aplicativos completos

Essa ferramenta fornece todos os principais fundamentos de programação na construção de blocos para a aplicação, mas de maneira mais didática. Apesar de ser voltado para o ambiente escolar, escolar ou para aprendizagens de maneira individual, ele é uma ferramenta totalmente utilizável em construções de aplicativos completos.

No caso deste projeto, “utilizamos ele para fazer protótipos para o nosso projeto, um aplicativo que permitisse sua utilização para registrar e visualizar itens em uma lista no servidor”. A parte do aplicativo, que foi batizada de leitura e escrita, e o mesmo foi totalmente desenvolvido no MIT App Inventor.

5. METODOLOGIA

5.1. Estudando o Arduino e sua comunicação Serial

A placa utilizada no projeto foi o Arduino UNO, o que necessitou de um estudo mais aprofundado da sua linguagem de programação, para que fosse possível as primeiras aplicações simples e testes iniciais com sua IDE (*Integrated*

Development Environment), Ambiente de Desenvolvimento Integrado. Os primeiros contatos com as funções de leitura e escrita do RFID tinham que ter uma alta compreensão para poder aplicá-las no projeto, por isso foi necessária uma grande parcela de tempo na compreensão de códigos que possuem relações com o módulo RFID.

Outra área dos estudos iniciais foi o estudo da comunicação serial, pois era preciso entender como a tela serial básica do Arduino teria que ser utilizada para mostrar as informações dos cartões RFID e como aplicar a tela serial no nosso código de maneira conjunta com as informações recebidas do módulo MFRC 522.

5.2. Os códigos

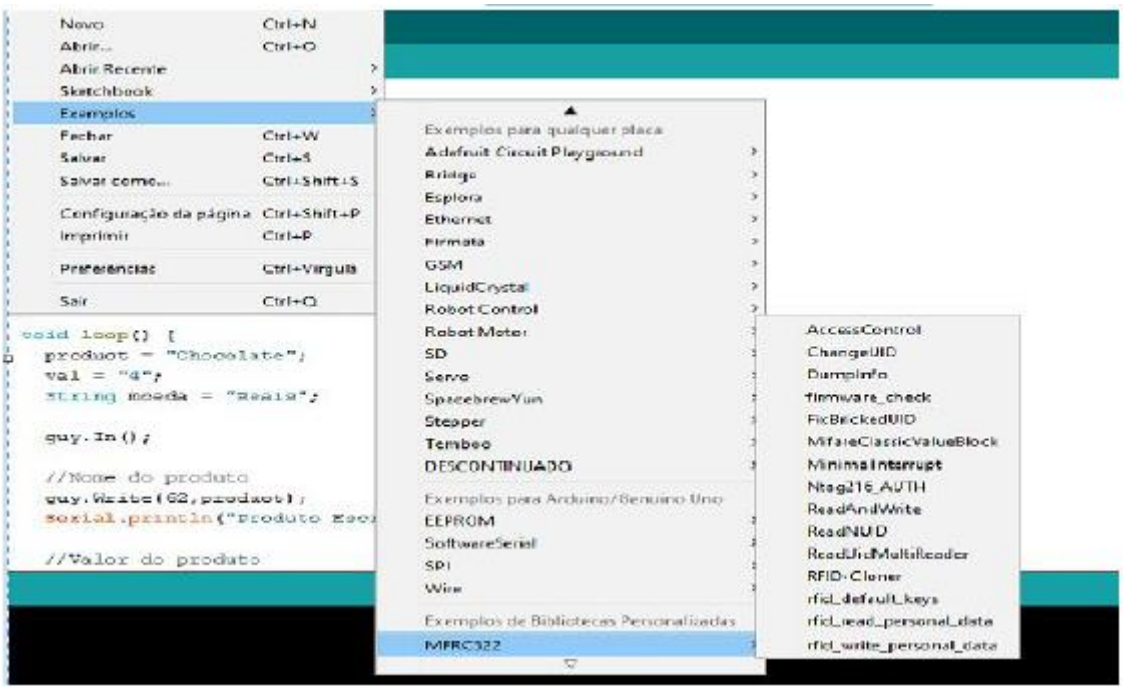
5.2.1. A organização dos Códigos.

Os códigos para desenvolver a aplicação foram divididos em duas partes, escrita e leitura. A primeira consiste em inserir uma informação no cartão RFID. Assim, o cartão torna-se um “ produto ”, e nele inserir dados como nome, quantidade, peso, preço entre outras. Por fim, a leitura é a divisão do código cuja função é ler as informações contidas nos cartões e enviar os dados para a tela serial da IDE do Arduino. A divisão do código em duas partes permite a organização das ideias e delimitação melhor dos objetivos finais com o código.

5.2.1.1. Primeiras linhas de código.

Os primeiros códigos utilizados para a compreensão do funcionamento do módulo RFID foram as duas bibliotecas que a própria IDE do Arduino disponibiliza. A biblioteca usada foi a MFRC 522, com dois exemplos usados para estudo e modificação: os exemplos usados foram *rfid_read_personal_data* para escrita e o exemplo *rfid_write_personal_data* para leitura. Os dois exemplos foram apenas usados para interpretação e compreensão da comunicação serial, escrita e leitura dos cartões RFID, juntamente com o entendimento do funcionamento do módulo RFID. A Figura 5 ilustra a biblioteca utilizada.

Figura 5 - Biblioteca usada e seus exemplos.



Fonte: Autoria Própria, 2019.

5.2.1.2. Entendimento do código em função dos Blocos.

Os cartões RF (Radiofrequência) apresentam em si um circuito que armazena as informações através de dados inseridos. O circuito que constitui no cartão é composto por blocos, que são às estruturas de armazenamento do cartão (*tags*). Esses blocos são:

- Bloco de RF: Composto por sistemas de modulação e de modulação do sinal, regulador de tensão, *Power On Reset* (POR) e gerador de *clock*.
- Anticolisão: Atendendo a ISO-14443, esse sistema permite fazer a seleção de um único cartão, mesmo que vários estejam dentro do campo do leitor.
- Unidade Lógica: Responsável por controlar os processos lógicos dentro do cartão, como solicitar autenticação, leitura e escrita na memória, controlar acesso aos dados, gerenciar os comandos, etc.
- Unidade de criptografia: Responsável pela autenticação da comunicação através do sistema CRYPTO1.
- A memória interna do chip ou EEPROM;

Os cartões são constituídos por uma memória EEPROM interna, em que as capacidades mais comuns são de 1KB, 2KB e 4KB. Internamente, as memórias são divididas em setores, que por sua vez são divididos em blocos, em que cada bloco possui 16 *bytes*. Vale destacar que alguns espaços de memória no cartão são para uso específico de funções do chip. A estrutura de armazenamento desses espaços específicos é distribuída em 3 B*locks* principais: *Manufacturer Block, Data Blocks* e *Sector Trailer.*

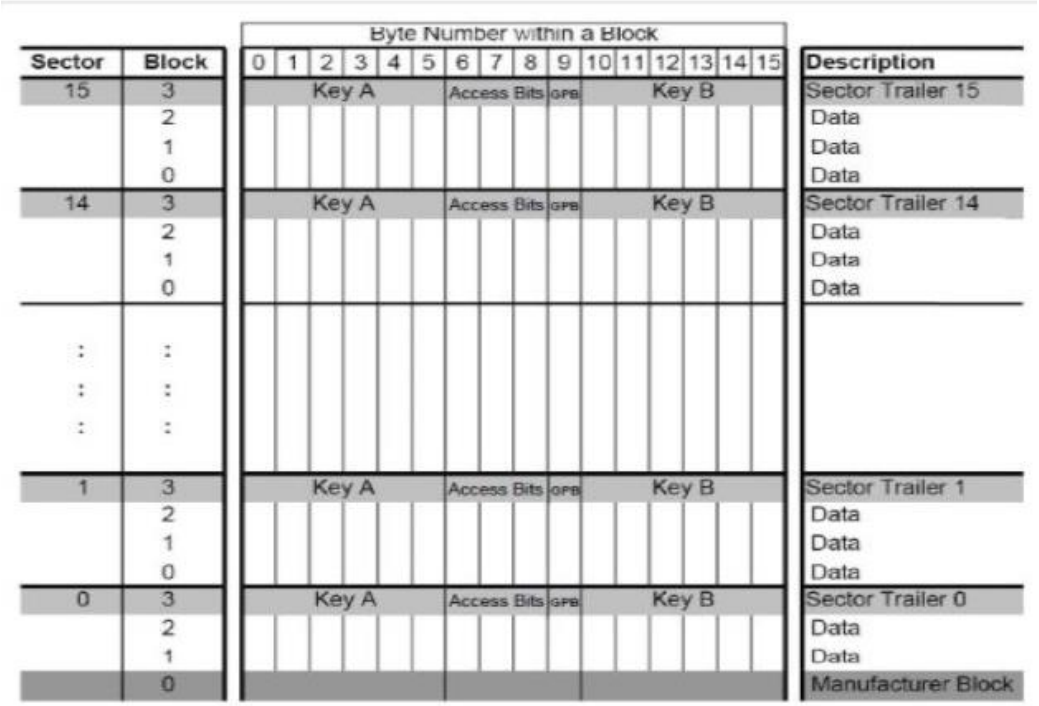
Como já dito anteriormente, os cartões já possuem dados que são do fabricante. Essas informações podem ser: ID do chip, dados de fabricação e/ou quantidade de espaço de memórias. Essas informações estão contidas exatamente no primeiro bloco, chamado de *Manufacturer Block* ,em que cada chip apresenta um número de série próprio, podendo ser de 4 ou 7 *bytes*, indicado pelos primeiros *bytes* no *Manufacturer Block.*

Além disso, os 3 primeiros blocos de cada setor (0 a 2) são para armazenamento de dados, podendo ser configurados como *value blocks* ou *read/write blocks* . O último, como o nome sugere, são blocos para leitura e escrita de dados; e os *value blocks* foram criados para serem implementados em sistemas que envolvem contagem de crédito, pois tem como funções o incremento ou decremento de valores.

O último bloco de cada setor é chamado *sector trailer* , que é fundamental para o funcionamento do cartão, pois armazena as chaves de acesso para os demais blocos daquele setor e também as condições de acesso de cada bloco daquele setor. Isso significa que ele determina a entrada para acesso de cada época de memória.

Para ter acesso aos blocos dos setores é necessário fazer a autenticação. O *sector trailer* contém duas chaves, *Key A* e *Key B*, em que *Key B* é opcional. Uma dessas chaves deve ser usada para fazer a autenticação. Após isso, os dados dos blocos daquele setor podem ser acessados. Esse acesso é realizado pelo *Access Condition*. O campo *Access Condition* é composto por 4 bytes, armazenando as condições de acesso e o modo de funcionamento de cada bloco do seu setor. A Figura 6 mostra a divisão dos blocos do cartão RFID.

Figura 6 - Estrutura de armazenamento de uma *tag* RFID.



Fonte: Embarcados.

Visto isso, é perceptível que há uma forma de acessar tais blocos. O código para a leitura e escrita nos cartões necessitavam apresentar variáveis que acessem esses blocos para efetuar a leitura ou escrita, utilizando o mecanismo do *Data Block* e modificando os 3 primeiros bits (0,1 e 2)para que ocorra a escrita ou leitura da informação contida no *Sector Trailer*. Portanto, o código deste trabalho baseia-se em toda a teoria que se aplica ao chip que está contido no cartão RFID.

5.3. Comunicação entre o módulo e as *tags* RFID.

Como já comentado, os cartões RFID apresentam um chip que constitui em um circuito onde as informações são gravadas. Porém este circuito também realiza a forma de comunicação entre o módulo e seu chip. O que ocorre é um *request* (pedido) feito pelo módulo MFRC 522 em que o chip do cartão responde com seu número de identificação. Com o módulo já conectado ao cartão, o leitor transmite, por meio de troca de bits, informações para qual local da memória acessar, utilizar a chave configurada para fazer a autenticação do bloco de memória. Depois da autenticação, todas as informações trocadas são encriptadas pela CRYPTO1.

5.4. Etapas da produção de uma biblioteca

Haja vista todas as especificações da comunicação e acesso para a leitura e escrita dos cartões RFID, além das funções do projeto para o gerenciamento de estoque, fica evidente a necessidade da criação de uma biblioteca, onde ficam armazenados todos os códigos e funções para o desenvolvimento do ecossistema de gerenciamento de estoques. As Figuras 7 e 8 a seguir mostram o código comentado. As etapas do desenvolvimento de uma biblioteca são:

- Um arquivo *header* com extensão “.h” que contém a declaração de todas as funções e variáveis da biblioteca.
- O arquivo *Source* com extensão “.cpp” que contém a lógica das funções.
- O arquivo *keywords*.txt, específico da IDE do Arduino, que é necessário para atribuir uma coloração às funções da biblioteca enquanto se escreve o código.

Figura 7 - Arquivo.H, declaração das funções.

```
#ifndef BluExodia_h
#define BluExodia_h

#include <Arduino.h>
#include <SPI.h>
#include <MFRC522.h>

class BluExodia{
  public:
    BluExodia(); //Instancia, não tem nenhum comando nem nada

    void Hearth(); //Aplicada na void setup, inicia funções necessarias
    void In();
    void Clean(int block); //Limpa o bloco do cartão selecionado no parametro
    void ReadToSerial(int block); //Ler o bloco selecionado no parametro e escreve no Serial
    void Write(int block, String guy); //Escreve uma String num bloco selecionado
    void Dump(); //Imprime cada bloco do cartão no Serial
    int InGame(); //Função que retorno 1 se tiver um cartão no leitor e 0 no oposto
    String ReadToStr(int block); //Retorna em String o que escrito no bloco selecionado
```

Fonte: Autoria própria, 2019.

Figura 8 - Um exemplo do arquivo.CPP.

```
void BluExodia::Clean(int block){

  //Preparação para chave
    for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
  //----------------------------------------

  //Variaveis necessárias
    byte buffer[34];
    byte len;
```

Fonte: Autoria própria, 2019.

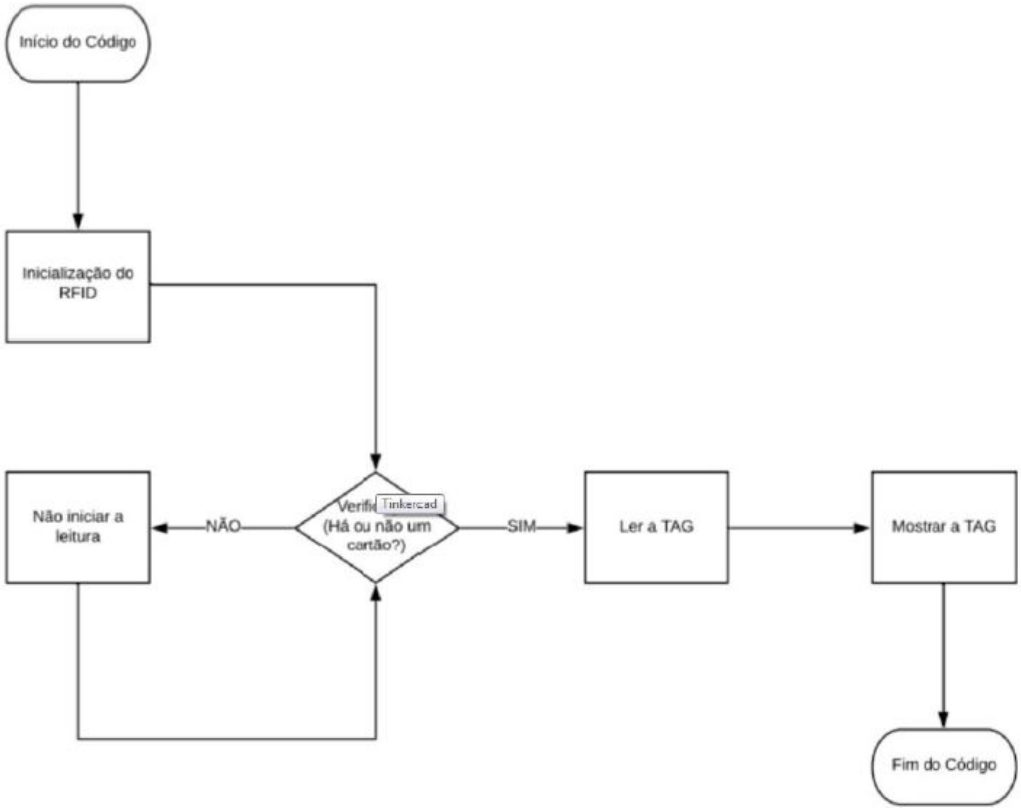
É válido destacar que a biblioteca *BluExodia* e todas as funções que ela contém foram desenvolvidas pelo próprio grupo. É autoria do nosso grupo. A produção da biblioteca teve como base os exemplos disponíveis na própria IDE do Arduino, em que os exemplos utilizados como base foram *rifd_read_personal_data, rfid_wirte_personal_data* e *access control* . Os códigos dos arquivos *.cc* e *.h* estão respectivamente nos apêndices A e B.

5.5. Produção do código para leitura.

5.5.1. Introdução ao código.

O código de leitura baseia se na ideia de pegar a informação armazenada em determinado bloco e mostrá-lo na tela serial do próprio Arduino. Isso é realizado a partir de quatro funções. A primeira função é “ *guy.Hearth*() ”, que tem como inicia o *Void setup* do Arduino, onde estão armazenados os parâmetros da comunicação serial. A segunda função é “ *guy.in*() ”, que verifica a presença ou não de um cartão RFID no módulo. A terceira função é a “ *guy.ReadToSerial*() “, que realiza a leitura do bloco selecionado para a leitura e envia a informação lida para a tela serial. A quarta função é “ *guy.end*() ”, que encerra o código de leitura localizado no apêndice C. A Figura 9 apresenta o fluxograma que informa a funcionalidade do código.

Figura 9 - Fluxograma do código.



Fonte: Autoria Própria – Lucid Chart, 2019.

5.5.1.1. Explicando o funcionamento do código.

No começo de todo código é necessário a inclusão da biblioteca básica *“<BluExodia.h> ”* através de um *#include* , onde estão declaradas as funções utilizadas no código. No início do código, é necessário realizar o *set (*defini*r)* de todos os parâmetros para a comunicação serial. Esses parâmetros são a velocidade da troca de Bits entre monitor serial e as informações recebidas; a velocidade usada é a padrão, 9600bps (bits por segundos). Os outros dois parâmetros são a inicialização do protocolo SPI, que corresponde à forma de comunicação do módulo RFID e o Arduino, e a inicialização da função *mfrc522.PCD_Init(),* que permite o acesso aos blocos dos cartões.

A próxima função a entrar em funcionamento é “ *guy.in()* ”, que realiza a verificação da presença ou não dos cartões a serem lidos. A terceira função é a leitura dos blocos em que estão armazenadas as informações: por uma questão de casualidade, todos os blocos dos cartões lidos são os “ 60,61 e 62 ”. Quem realiza a atividade de ler as informações contidas nos blocos é a função

“*guy.ReadToserial*”, que além de fazer a leitura também transmite todas as informações para o monitor serial. A última função “*guy.end*”, é responsável por encerrar o código. Segue o código fonte na Figura 10:

Figura 10 - Código de Leitura.

```
#include <BluExodia.h>

BluExodia guy;

void setup() {
  guy.Hearth();
  Serial.println("Insira cartão");
}

void loop() {

  guy.In();
  guy.ReadToSerial(62);
  //Serial.println("Informação do Produto limpa");

  guy.ReadToSerial(61);
  //Serial.println("Informação do valor limpa");

  guy.ReadToSerial(60);
  //Serial.println("Informação da moeda limpa");

  guy.end();

  Serial.println("\nRFID Pronto :)");
  Serial.println("Insira outro cartão...\n");

  }
```

Fonte: Autoria própria, 2019.

5.6. Explicando a programação de cada função.

5.6.1. Função *guy.hearth*

A “ *guy.hearth()* ” objetiva indicar os parâmetros de comunicação. Segue o código fonte na Figura 11:

Figura 11 - Programação da função *guy.hearth.*

```
void BluExodia::Hearth(){

  Serial.begin(9600);
  SPI.begin();
  mfrc522.PCD_Init();
```

Fonte: Autoria própria, 2019.

5.6.2. Função *guy.in*

A função “ *guy.in*() “ utiliza o comando *Do while* , uma estrutura de repetição com o intuito de verificar a presença de cartões. Isso é feito com os parâmetros de presença do cartão *frc522.PICC_IsNewCardPresent()* e *mfrc522.PICC_ReadCardSerial().* Segue imagem do código fonte na Figura 12:

Figura 12 - Função “ guy.in()”

```
void BluExodia::In(){

  do{
    while ( ! mfrc522.PICC_IsNewCardPresent()) {

      }
    }while (! mfrc522.PICC_ReadCardSerial());

    return;

}
```

Fonte: Autoria própria, 2019.

5.6.3. Função *guy.ReadtoSerial*

A função *guy.ReadtoSerial()* é de fato a mais trabalhosa, pois além acessar e ler o bloco, envia as informações para a tela serial, efetuando a conversão dos dados de hexadecimal para binário de 8 bits, pois é dessa forma que a informação será armazenada.

Primeiramente, prepara-se a chave de acesso para obter acesso ao *Sector Trailer* através de um laço *for.* Em seguida, efetua-se a autenticação do módulo RFID com o cartão através do comando *IF ,* que define se a leitura foi realizada corretamente ou se houve falha na comunicação. Todos os dados da leitura são armazenados em uma variável chamada *buffer* , e enviados para a tela serial do Arduino.

De maneira geral, o código autentifica o acesso dos blocos de armazenamento e lê as informações presentes nesses blocos, além de enviar para a tela serial. Segue o código fonte explicado, na Figura 13:

Figura 13 - Função *read* serial

```
void BluEmodia::ReadToSerial(int block){
  //PreparaÃ§Ã£o para chave
     for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;

    //Variaveis q ira se precisar
    byte len;

  //-------------------------------------------

  len = 18; //Aparentemente len deve ser igual ao tamanho dos Buffers

  //-----------------Pega e ver se ta certo
  byte buffer1[18];
  status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block, &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_Read(block, buffer1, &len);
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("Reading failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }
  //-----------------------------------



  //----------Imprime o nome
  for (uint8_t i = 0; i < 16; i++)
  {
    if (buffer1[i] != 32)
    {
      Serial.write(buffer1[i]);
    }
  }
  //-----------------------------------
```

Fonte: Autoria própria, 2019.

5.6.4. Função *guy.end*

A quarta e última função, apenas define o fim do código, interrompendo a criptografia e a autenticação dos dados que são transmitidos entre o módulo e o cartão. Os comandos que realizam essa tarefa são *mfrc522.PCD_StopCrypto1()* e *mfrc522.PICC alta(),*. Segue o código fonte explicado, na Figura 14:

Figura 14 - Função *guy.end*

```
void BluExodia::end(){

  mfrc522.PICC_HaltA();
  mfrc522.PCD_StopCrypto1();

}
```

Fonte: Autoria própria, 2019.

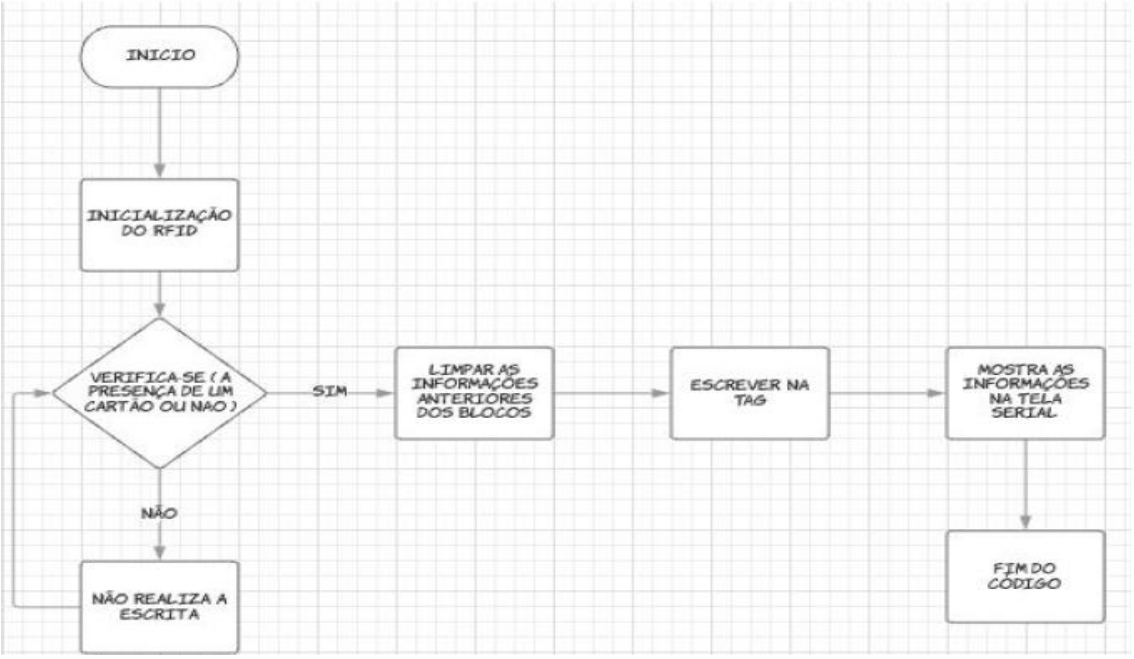
5.7. Produção do código para a escrita.

5.7.1. Operação do código.

O código de leitura também se baseia na estrutura de inserção de dados no cartão *Mifare*, onde é necessário acessar o *data block*, local em que é inserida a configuração de leitura ou escrita. O código é composto por quatro funções, além de conter 3 variáveis de tipo *string,* que informam o nome do produto, valor e a moeda usada no sistema. A primeira função usada é a *guy.Hearth()* , necessária para iniciar o *void setup* e executar os parâmetros de comunicação Serial e SPI. A função *guy.In()* verifica a presença de cartão no módulo MFRC 522. O terceiro bloco de instrução, *guy.Write()*, é responsável pela escrita dos dados nos espaços de memória nos cartões. O último bloco de comando é usado para o encerramento do código, *guy.end().*

De forma geral, o código funciona primeiramente identificando a chave de acesso para os blocos que vão armazenar as informações dos produtos. Segue o fluxograma que explica o algoritmo, na Figura 15: o código de escrita está presente no apêndice D.

Figura 15 - Fluxograma do código de escrita.



Fonte: Autoria Própria – Lucid Chart, 2019.

5.8. Explicando o funcionamento do código.

O código inicia-se com a declaração das variáveis *product, val* e moeda, que recebem a atribuição do nome, valor e moeda, características que são usadas para a identificação dos produtos. A primeira função do código executa o *set* de todos os parâmetros para a comunicação serial. Esses parâmetros são, a velocidade da troca de Bits entre monitor serial e as informações recebidas. Os outros dois parâmetros são, a inicialização do protocolo SPI, forma de comunicação do módulo RFID e o Arduino e a inicialização da função *mfrc522.PCD_Init()* que permite o acesso aos blocos dos cartões. Esse *set* de parâmetros é realizado pela função *guy.hearth().*

“A verificação da presença dos cartões é feita pelo bloco *guy.in()*. A função que realiza escrita nos cartões nos blocos 60,61 e 62 é a *guy.write*. O bloco de instrução *guy.end()*, responsável em finalizar o código”. Além disso, as linhas que fazem parte do parâmetro Serial.print são responsáveis por demonstrar que a escrita ocorrei corretamente, exibindo as informações na tela serial. Segue o código fonte explicado, na Figura 16:

Figura 16 - Código de escrita

```
#include <BluEmodia.h>

String product;
String val;

BluEmodia guy;

void setup() {
  guy.Hearth();
  Serial.println("Insira o cartão...");
}

void loop() {
  product = "Chocolate";
  val = "4";
  String moeda = "Reais";

  guy.In();

  //Nome do produto
  guy.Write(62,product);
  Serial.println("Produto Escrito");

  //Valor do produto
  guy.Write(61,val);
  Serial.println("Valor do produto Escrito");

  //Moeda utilizado
  guy.Write(60,moeda);
  guy.end();


  Serial.println("Unidade da Moeda Escrita");

  Serial.println("\nRFID Pronto :)");
  Serial.println("Insira outro cartão...\n");


}
```

Fonte: Autoria própria, 2019.

5.9. Explicando a programação de cada função.

5.9.1. Função *guy.write*

O bloco de instrução *guy.write* é composto pelos comandos responsáveis pela escrita dos cartões. O primeiro bloco faz a preparação para acessar o *data block* e o segundo bloco realiza a contagem de letras para verificar o tamanho do nome que será escrito, utilizando o comando *guy.length().* O terceiro bloco de comando realiza a comunicação entre o módulo e o cartão, efetuando um *set* na chave *Key A*. Após a autenticação da transmissão de informações ocorre a escrita no *Sector Trailer.* Vale destacar que antes da inicialização da função *guy.Write*, são iniciados os blocos *guy.Heart()* e *guy.In()*. Segue o código fonte explicado, na Figura 17:

Figura 17 - Código da função *guy.write*

```
void BluExodia::Write(int block, String guy){

  //Preparação para chave
    for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
  //-------------------------------------------

  //Variaveis necessárias
    byte buffer[34];
    byte len;


  //ESCRITA
  //----------Ler o que o cabra enviou
    len = guy.length();
    for (byte i = len; i < 30; i++){
      buffer[i] = 0;     // pad with spaces
    }
    guy.getBytes(buffer,len+1);
  //-------------------------------------------


  //----------Prepara para escrever no bloco e ver se ta certo
    //Serial.println(F("Authenticating using key A..."));
    status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH
    if (status != MFRC522::STATUS_OK) {
      Serial.print(F("Autenticação = Deu ruim "));
      Serial.println(mfrc522.GetStatusCodeName(status));
      return;
    }
```

Fonte: Autoria própria, 2019.

5.9.2. Função *guy.clean*

É válido ressaltar que, antes da realização da escrita, efetua-se a limpeza do cartão, ou seja, apagam-se as demais informações que antes estavam armazenadas nos blocos de memória. A função responsável por isso é chamada de *guy.clean*.

5.9.3. Integração com o sistema de organização

Os códigos de escrita e leitura foram unidos para integração com o código de organização do sistema de estoque. Para isso, utilizou-se uma tela LCD para melhor visualização dos dados, além do envio dos dados para o servidor, que os armazena e dispõem as informações em forma de tabela O programa foi feito com código no *Mysql*.

O código final da contagem de estoque utiliza a biblioteca *bluexodia* , de autoria própria, além das bibliotecas *LiquidCrystal.h* e *MFRC522.h,* responsáveis

respectivamente pela interação com a tela LCD e comunicação com o módulo RFID. A mecânica utilizada para realizar a troca de “ compra ” e “ venda ” de produtos é o simples aperto do botão: quando o botão envia zero para o Arduino, retira-se um produto no servidor; ao pressionar 1, adiciona-se mais um item do produto. O bloco de instrução que realiza essa mecânica é o *IF*. O parâmetro *Serial.print ("http://192.168.43.144/RFID/quantidade.php?nome=")* é responsável pela comunicação do código com o servidor. O código completo da contagem de estoque está no apêndice E. Segue o código fonte explicado, na Figura 18.

Figura 18 - Parte do código de contagem de estoque.

```
guy.In();

botao = digitalRead(but);

    if(botao==LOW){

      //guy.In();

      guy.Readf(62);
      String tProd = guy.Namef();
      guy.Readf(61);
      String tVal = guy.Namef();

      //Serial.print(" UID: ");
      //guy.ReadToSerial(0);


      lcd.clear();
      lcd.print((String)"Produto: "+tProd);
      lcd.setCursor(0,1);
      lcd.print((String)"Preco:" + tVal);

      //tProd.replace(" ","");
      Serial.print("http://192.168.43.144/RFID/compra.php?nome=");
      Serial.println(tProd);

      guy.end();
```

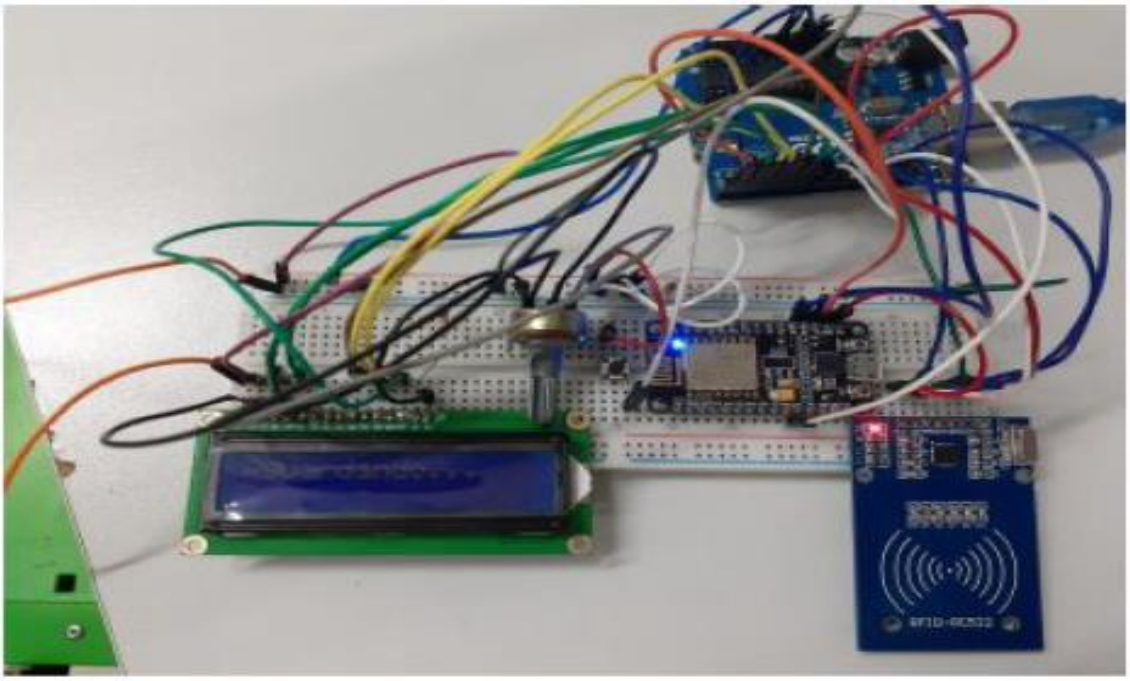
Fonte: Autoria própria, 2019.

5.9.4. Integração com ESP 8862 e protótipo.

Após a realização dos códigos no Arduino, o próximo passo é a comunicação com o servidor que estaria ligado com o APP para monitoramento do estoque. Isso foi realizado através de um dispositivo chamado *ESP 8862*, um microcontrolador em que é possível conectar-se à internet, já que no módulo do Arduino não há integração com *Wi-Fi*. Isso foi feito através do envio de linhas de *URL* do Arduino-Esp para o servidor criado, em que informações como quantidade, preço e nome são enviadas e organizadas na interface do aplicativo.

Foi possível realizar a comunicação serial do Arduino com o *ESP 8862* através das as portas *RX (0)* e *TX(1),* que permitem que o módulo Arduino faça comunicação com outros dispositivos ou até mesmos computadores, pois estas portas têm como função principal fazer o *upload* de dados para outros dispositivos, neste caso o *ESP*. O último passo consiste na captação das informações do *ESP* para o servidor, feito através do código no *Mysql,* em que pacotes de informações como quantidade, preços e nome do produto eram enviados, se foi efetuada uma compra do produto ou não. Feito isso, o projeto estava totalmente finalizado. A Figura 19 ilustra o protótipo.

Figura 19 - Protótipo do projeto.



Fonte: Autoria própria, 2019.

5.10. Aplicativo para registro de produtos.

5.10.1. Plataforma.

Utilizando a plataforma *App Inventor*, disponibilizado pelo MIT (*Massachusetts Institute of Technology*) de forma online, desenvolveu-se um aplicativo programado em blocos, capaz de gravar no servidor informações sobre um determinado produto em estoque, registrando nome, quantidade e até mesmo o preço. Batizou-se o aplicativo de Leitura e Escrita.

Figura 20 - App Inventor, Projeto Leitura e Escrita.

Fonte: Autoria própria, 2019.

O App Inventor inicia na tela de edição de ecrãs, onde se pode configurar e adicionar componentes ao layout.

5.10.2. LAYOUT

O layout é composto por legendas, caixas de texto, botões, blocos de organização horizontal e um visualizador de listas.

Figura 21 - Layout.

.

Fonte: Autoria própria, 2019.

Uma breve explicação sobre os itens na tela do aplicativo:

REGISTRAR ITEM: (adicional/opcional) legenda título.

Organização horizontal 1: contém legenda Nome e preço, seguidas por uma caixa de texto vazia em que deve se inserir o nome e preço do produto, respectivamente.

Organização horizontal 2: contém a legenda Quantidade (x), seguida por um botão “-” que subtrai uma unidade do valor atual. Na sequência, uma caixa de texto vazia informa a quantidade, seguida por um botão “+” que soma uma unidade ao valor atual.

Registrar produto: botão inicialmente desativado, que será utilizado para realizar o registro do produto no servidor.

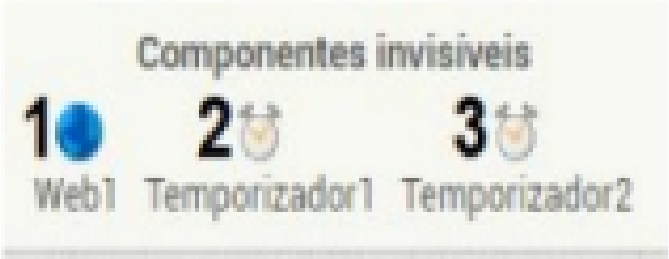
Visualizador de listas: local que armazena a relação dos produtos em estoque; contém uma barra de pesquisa.

Organização horizontal 3: possui caráter adicional/opcional; apresenta outra organização horizontal, que possui apenas uma legenda Amostra com um posicionamento diferente (esquerdo-superior); contém uma caixa de texto desativada e de fundo transparente, onde é possível visualizar um produto selecionado.

Atualizar: botão que atualiza o visualizador de listas com as informações atuais do servidor.

Alguns componentes necessários para o funcionamento dos aplicativos não podem ser visualizados na tela, sendo eles representados na Figura 22.

Figura 22 - Componentes invisíveis.



Fonte: Autoria própria, 2019.

Esses são os componentes que utilizamos no desenvolvimento do layout do aplicativo.

1- *Web1:* *permite a comunicação com um site/servidor através da* URL*.*

2- Temporizador 1: adicional/opcional: utilizado para verificar se as caixas de texto foram preenchidas.

3- Temporizador 2: utilizado na atualização da lista.

5.10.3. BLOCOS(PROGRAMAÇÃO).

A programação em blocos permitiu estabelecer, de forma simples e intuitiva, a interação entre os componentes, de modo a alcançar a finalidade desejada para o aplicativo.

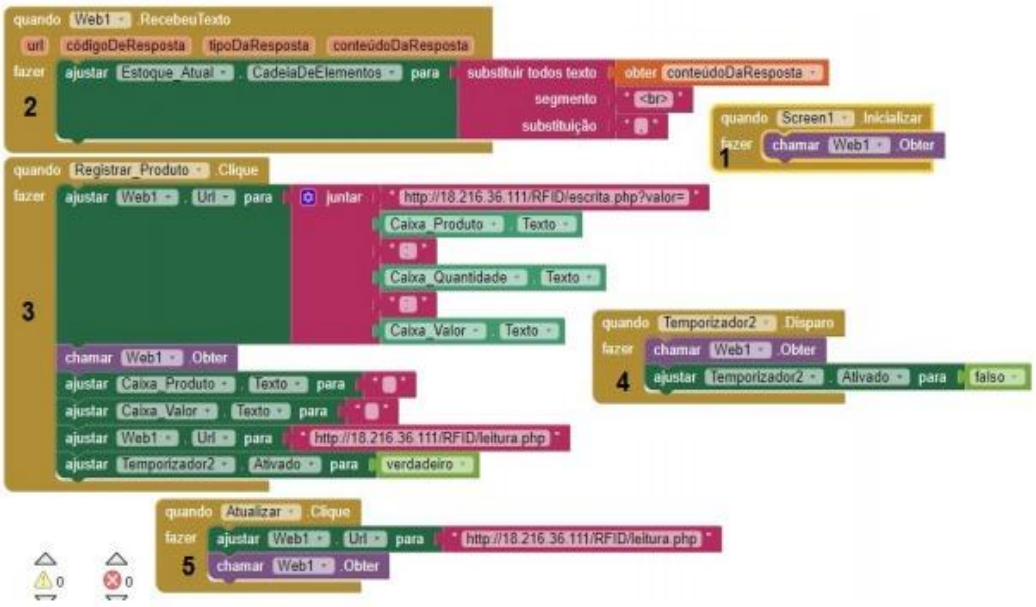
Figura 23 - Blocos principais.

Fonte: Autoria própria, 2019.

Ao clicar no ícone “Blocos” no lado superior direito da tela do App Inventor, onde está demarcado com um círculo vermelho, aparece a tela de blocos, onde pode ser realizada a programação.

A programação deste trabalho consiste em 5 blocos principais, como será observado em seguida.

Figura 24 - Blocos numerados.



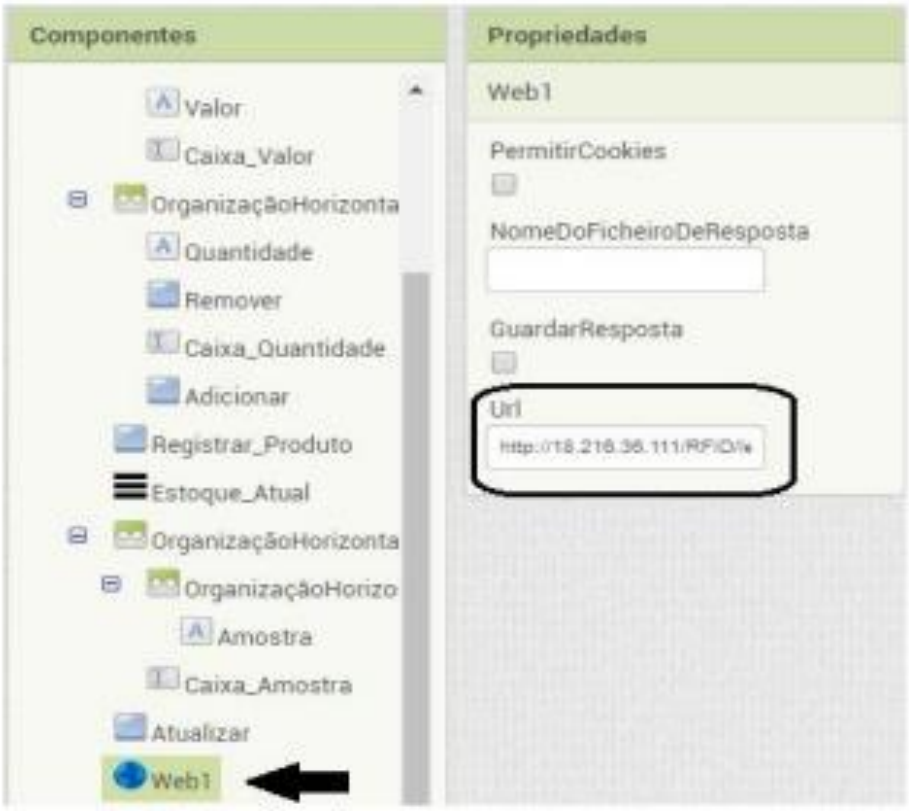
Fonte: Autoria própria, 2019.

Este primeiro bloco faz com que, ao iniciar a Screen1 (que é a tela inicial), chame-se a função ".Obter" do componente *Web1*. Esta função obtém os dados da *URL* definida, que deve ser correspondente à leitura do servidor.

Figura 25 - Primeiro bloco.

Fonte: Autoria própria, 2019.

Figura 26 - URL.



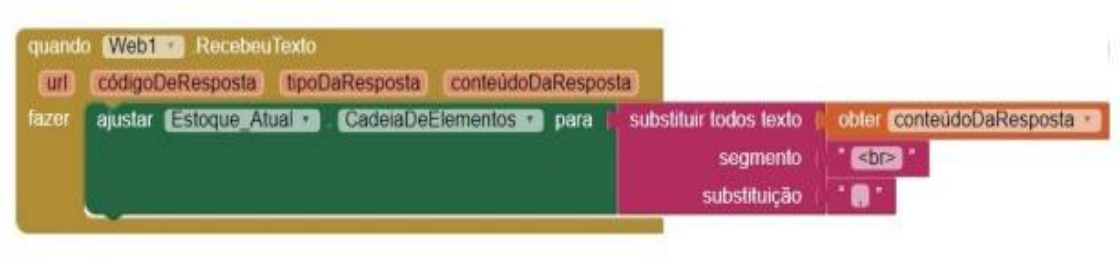
.

Fonte: Autoria própria, 2019.

Ao clicar no componente Web na área de componentes do editor de ecrãs, pode-se definir a *URL* inicial na área de propriedades.

O segundo bloco permite visualizar os itens do servidor no visualizador de listas. Quando o componente Web1 obtiver um texto como resposta, o parâmetro *Cadeia De Elementos* do visualizador de listas(aqui renomeado como *Estoque Atual*) será alterado para o *Conteúdo Da Resposta*, com o detalhe de que deve se substituir os “
” por “,” (vírgula). Isso deve ser feito pois a resposta obtida é lida em HTML, em que os itens do servidor estão separados por “
”; no entanto, o parâmetro *Cadeia De Elementos* entende que há itens distintos somente quando estes estão separados por vírgula. Fazendo tal substituição, evita-se que o visualizador de listas entenda todas as linhas de texto do servidor como um único item.

Figura 27 - Segundo bloco.



Fonte: Autoria própria, 2019.

Para maior esclarecimento, as Figuras 28 e 29 ilustram como era e como ficou na prática, depois que ocorreu a substituição dos “
” para “,”(virgula).

Figura 28 - Formatação original do servidor.

```
<html>

<head>
Estoque
<head>

<body>
Shampoo - R$ 10.50 x 15 <br> Salgadinho - R$ 5.00 x 23 <br> Laranja - R$ 3.50 x 36
<body>

<html>
```

Fonte: Autoria própria, 2019.

Figura 29 - Formatação alterada.

```
<html>

<head>
Estoque
<head>

<body>
Shampoo - R$ 10.50 x 15,Salgadinho - R$ 5.00 x 23,Laranja - R$ 3.50 x 36
<body>

<html>
```

Fonte: Autoria própria, 2019.

O terceiro bloco faz com que, quando o botão *Registrar Produto* for pressionado, o parâmetro *URL* do componente *Web1* seja alterado para o endereço de escrita unindo eles aos textos preenchidos nas caixas de texto referentes ao nome, quantidade e preço, que devem ser unidos nesta ordem e separados por “ : ” (dois pontos), devido a uma característica da programação

do próprio servidor. Este endereço permite que qualquer texto adicionado a ele após o caractere “ = ” seja escrito no servidor, por isso unimos as caixas de texto contendo as informações do produto. Em seguida, chamamos novamente a função *“.Obter”* do componente *Web1* para que na nossa *URL* seja acessada, gravando as informações no servidor.

A seguir, redefiniu-se os parâmetros *Texto* das caixas de texto referentes ao nome e ao valor do produto para espaços vazios, permitindo que novas informações possam ser escritas. Ao final do bloco, retornamos o parâmetro *URL* do componente *Web1* para o nosso endereço inicial de leitura. E então alteramos o parâmetro *Ativado* do nosso *Temporizador2* (será usado no bloco 4) para *verdadeiro* (ele deve começar desativado).

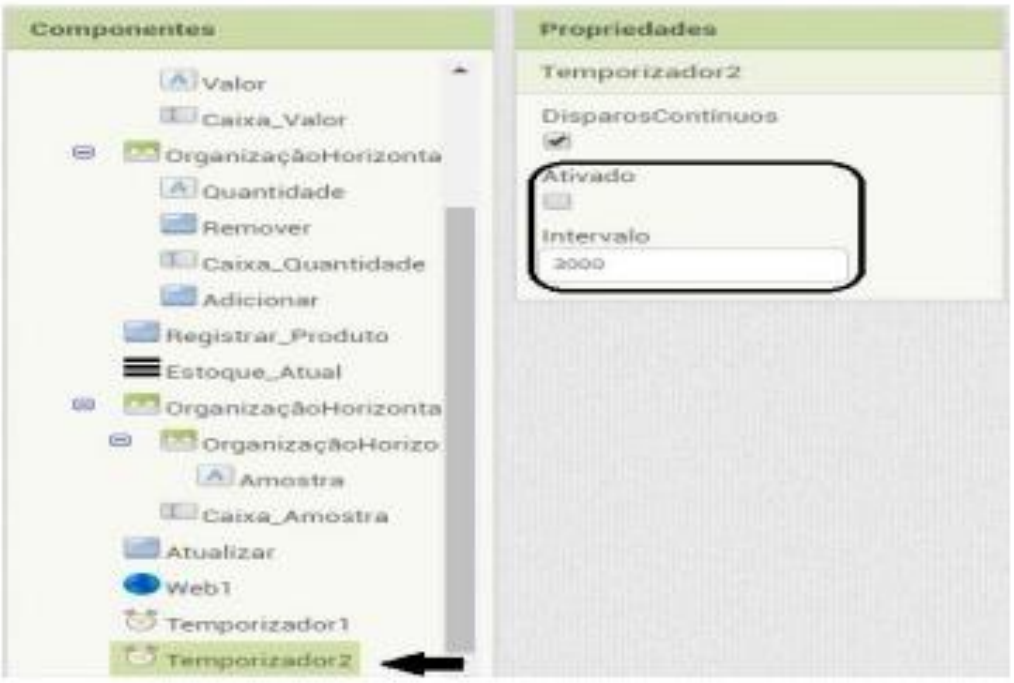
Figura 30 - Terceiro bloco.

Fonte: Autoria própria, 2019.

Figura 31 - Botão registrar produto.

Fonte: Autoria própria, 2019.

Figura 32 - Temporizador2, propriedades.



.Fonte: Autoria própria, 2019.

Ao clicar no componente *Temporizador2* na área de componentes do editor de ecrãs, pode-se definir seu intervalo de tempo em ms (milissegundos) e se este irá começar ativado ou não.

O quarto bloco permite que o nosso visualizador de listas seja atualizado com as novas informações do servidor. Quando o *Temporizador2* dispara, de acordo com o intervalo pré-definido, chamamos mais uma vez o componente *Web1* ativando sua função *".Obter"* e logo em seguida definindo o parâmetro Ativado do *Temporizador2* para *falso*. Este bloco é necessário porque quando chamamos o *Web1* no bloco anterior, ele ainda está com a *URL* de escrita, que após usarmos a função *".Obter"* retorna uma mensagem de confirmação, que nos será mostrada graças ao bloco 2. A fim de manter esta mensagem em tela apenas por alguns segundos, atrelamos a nova chamada do *Web1*(que já voltou à sua *URL* de leitura, ao final do bloco 3) a este temporizador, para que, após o intervalo determinado, o visualizador volte a nos mostrar os itens do estoque ao invés da mensagem. Isto, mais uma vez, é garantido pelo segundo bloco.

Figura 33 - Bloco quatro.

Fonte: Autoria própria, 2019.

Figura 34 - Mensagem Array.

Fonte: Autoria própria, 2019.

A Figura 34 mostra a mensagem exibida após o componente Web1 ser chamado com a URL de escrita.

O quinto e último dos blocos principais garante que o visualizador de listas seja atualizado com as informações atuais do estoque a qualquer momento, para os casos em que alterações no servidor são feitas por fora do aplicativo. Quando o botão *Atualizar* for pressionado, o parâmetro *URL* do componente *Web1* será alterado para o nosso endereço de leitura e então a função *".Obter"* do *Web1* nos trará a resposta a ser exibida no visualizador de listas, graças ao bloco 2.

Figura 35 - Bloco cinco.

Fonte: Autoria própria, 2019.

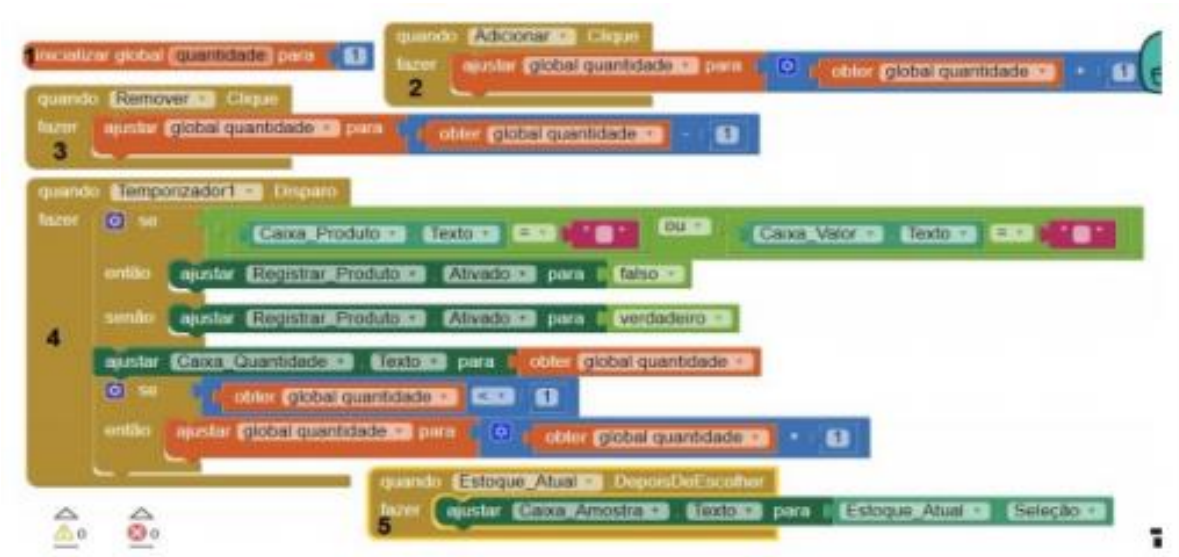
Figura 36 - Botão atualizar.

Fonte: Autoria própria, 2019.

5.10.4. BLOCOS AUXILIARES

Estes blocos, como o próprio nome evidencia, auxiliam os nossos blocos principais da programação, manipulando variáveis e realizando operações.

Figura 37 - Blocos auxiliares.



Fonte: Autoria própria, 2019.

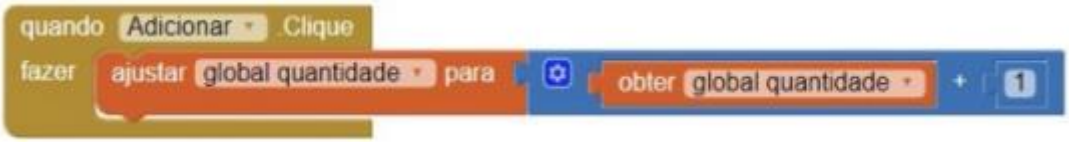
Este primeiro bloco cria uma variável que corresponde ao valor da quantidade do produto. Inicialmente definimos seu valor como sendo igual a1, mas este valor poderá ser acrescentado ou subtraído, segundo os blocos seguintes.

Figura 38 - Bloco auxiliar 1.

Fonte: Autoria própria, 2019.

O segundo bloco soma uma unidade ao valor atual da nossa variável. Ao clicar no botão Adicionar, o valor da variável *quantidade* será alterado para o seu valor atual + 1.

Figura 39 - Bloco auxiliar 2.



Fonte: Autoria própria, 2019.

Figura 40 - Botão Adicionar.

Fonte: Autoria própria, 2019.

O terceiro bloco subtrai uma unidade do valor atual da variável. Ao clicar no botão Remover, o valor da variável *quantidade* será alterado para o seu valor atual - 1.

Figura 41 - Bloco auxiliar 3.

Fonte: Autoria própria, 2019.

Figura 42 - Botão Remover.

Fonte: Autoria própria, 2019.

O quarto bloco garante que o botão *Registrar Produto* só possa ser seja habilitado apenas quando ambas as caixas de texto, referentes ao nome e ao preço do produto, forem preenchidas com alguma informação. Também define que o texto da caixa de *Texto* referente à *quantidade* será igual ao valor atual da variável *quantidade* e que este valor não poderá ser inferior a 1.

Quando o *Temporizador1* pré-definido para intervalos de 1 “ms(milissegundo)”, disparar, se o parâmetro *Texto* de uma das caixas referentes ao nome ou ao preço do produto estiver vazio, o parâmetro *Ativado* do botão *Registrar Produto* será alterado para *falso;* senão, será alterado para *verdadeiro*. O restante do bloco define que o parâmetro *Texto* da nossa *Caixa Quantidade* deverá ser igual ao valor atual da nossa variável global *quantidade*. E que, se este for menor que uma unidade, automaticamente soma-se uma unidade ao valor atual.

Figura 43 - Bloco auxiliar 4.

Fonte: Autoria própria, 2019.

Este quinto e último dos blocos adicionais/opcionais nos permite isolar em uma caixa de texto o atual item selecionado no visualizador de listas. Depois que um item é escolhido em *Estoque Atual* (nosso visualizador de listas), o parâmetro *Texto* da nossa *Caixa Amostra* é definido como igual ao parâmetro *Seleção* do nosso *Estoque Atual*.

Figura 44 - Bloco auxiliar 5.

Fonte: Autoria própria, 2019.

5.10.5. Síntese do Aplicativo

Nosso aplicativo é capaz de ler e escrever em um servidor destinado a representar o estoque de uma loja. O design é simples e intuitivo. Visualizar e buscar um item na lista é tão fácil quanto definir e registrar um novo item. Além disso, como a ferramenta (ou plataforma) usada em sua criação é de fácil acesso e rápida para fazer alterações e testes, outras funcionalidades podem ser atribuídas futuramente sem muita dificuldade.

6. RESULTADOS E DISCUSSÕES

Antes de tudo, vale ressaltar que os objetivos gerais do nosso projeto foram alcançados, e o protótipo do projeto é totalmente funcional. Utilizando apenas os cartões *Mifare*, o módulo RFID e um smartphone com o aplicativo desenvolvido Leitura e Escrita, conseguimos simular perfeitamente o controle de um estoque de maneira organizada. A Figura 19 resume a montagem do projeto.

Com exceção das bibliotecas disponibilizadas pela IDE de Arduino, todo o algoritmo foi autoria própria. Inclusive, uma das bibliotecas desenvolvidas, a *BluExodia*, foi disponibilizada no *GitHub*, uma comunidade de compartilhamento de códigos de programação. O aplicativo, bem como toda sua programação em blocos e layout, também foi projetado pela equipe.

Devido ao cenário atual da nossa sociedade brasileira e de todo mundo, a pandemia do novo Coronavírus (*COVID-19*) e a necessidade de isolamento social desde o mês de março de 2020, não foi possível efetuar novas melhorias e fazer atualizações do nosso projeto, já que para isso seriam necessárias ferramentas encontradas em nossos laboratórios.

Chegamos a conclusão de que este projeto pode ser extremamente versátil e facilmente alterado, permitindo que sua ideia seja remodelada e aplicada em diversas áreas, como as áreas da saúde, industrial, automobilística e educacional. O que proporciona uma fácil continuidade para trabalhos e projetos futuros desenvolvidos pelas próximas gerações do nosso IFRN Campus Natal Zona Norte.

7. CONSIDERAÇÕES FINAIS

Primeiramente, concluiu-se com sucesso a realização do circuito eletrônico que possibilita a organização do estoque de itens, criado especificamente para ajudar na estruturação da retirada dos componentes eletrônicos que são encontrados na coordenação de laboratórios. As maiores dificuldades encontradas foram nas criações dos códigos e bibliotecas, pois apesar das bases criadas em programação pelas matérias do curso de Eletrônica, as etapas necessárias para a formação do projeto precisavam de estudos mais avançados na programação dos códigos, tanto para o Arduino e o servidor quanto em relação à criação do APP.

É perceptível que a construção do protótipo torna viável a estrutura física do projeto, porém é necessária a realização de uma PCI (placa de circuito impresso), para que o circuito seja mais fácil de ser manuseado, além da necessidade de testes na Coordenação de Laboratório, com o intuito de comprovação definitiva da utilização do controlador de estoque no campus do IFRN Zona Norte. Apesar disso, esta etapa pode ficar para projetos futuros da próxima geração de concluintes de nossa instituição.

Os testes realizados com a montagem novamente do projeto não foram possíveis, haja vista a situação vivida atualmente em decorrência da pandemia do Coronavírus, o que traz a necessidade de novos testes no servidor e integração com o ESP8266. Contudo, através deste relatório, demonstrou-se a viabilidade do projeto e de como sua construção foi realizada, tanto de forma teórica, através do estudo dos cartões, módulo e códigos, quanto pela prática, por meio do desenvolvimento da biblioteca e montagem do circuito.

8. REFERÊNCIAS

Embarcados. RFID – Cartões *Mifare*, Disponível em: <https://www.embarcados.com.br/rfid-cartoes-mifare/> - acesso em 02/03/2021.

Tecmundo, Como Funciona a RFID Disponível em: <https://www.tecmundo.com.br/tendencias/2601-como-funciona-a-rfid-.htm> - acesso em 02/03/2021.

MIT APP INVENTOR. Página Inicial, Disponível em: <https://appinventor.mit.edu> - acesso em 02/03/2021.

Wiki IFPR. Laboratório: Introdução ao App Inventor, Disponível em :< http://wiki.foz.ifpr.edu.br/wiki/index.php/Laborat%C3%B3rio:_Introdu%C3%A7%C3%A3o_ao_App_Inventor> - acesso em 02/03/2021.

Apêndice A - Arquivo. CPP

```
#include "BluExodia.h"
#define RST_PIN 9
#define SS_PIN 10
MFRC522 mfrc522(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;
MFRC522::StatusCode status;
BluExodia::BluExodia(){
}
void BluExodia::Hearth(){
Serial.begin(9600);
SPI.begin();
mfrc522.PCD_Init();
}
void BluExodia::In(){
do{
while ( ! mfrc522.PICC_IsNewCardPresent()) {
//Um grande e magnifico nada :)
}
}while (! mfrc522.PICC_ReadCardSerial());
return;
}
void BluExodia::Clean(int block){
//Preparação para chave
for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
//-----------------------------------------
//Variaveis necessárias
```

```
byte buffer[34];
byte len;
//ESCRITA
//----------Ler o que o cabra escreve no Serial
len = 0;
for (byte i = len; i < 30; i++) buffer[i] = 0; // pad with spaces
//-----------------------------------------
//----------Prepara para escrever no bloco e ver se ta certo
status =
mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block,
&key, &(mfrc522.uid));
if (status != MFRC522::STATUS_OK) {
Serial.print(F("PCD_Authenticate() failed: "));
Serial.println(mfrc522.GetStatusCodeName(status));
return;
}
else Serial.println(F("PCD_Authenticate() success: "));
//-----------------------------------------
//++++++++++++++++++++++++
//----------Bloco que escreve o que ta no bloco anterior
status = mfrc522.MIFARE_Write(block, buffer, 16);
if (status != MFRC522::STATUS_OK) {
Serial.print(F("MIFARE_Write() failed: "));
Serial.println(mfrc522.GetStatusCodeName(status));
return;
}
else Serial.println(F("MIFARE_Write() success: "));
//-----------------------------------------
//-----------------------------------------
Serial.print("Limpo");
}
void BluExodia::ReadToSerial(int block){
//Preparação para chave
for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
//Variaveis q ira se precisar
byte len;
//-----------------------------------------
len = 18; //Aparentemente len deve ser igual ao tamanho dos Buffers
//-----------------Pega e ver se ta certo
byte buffer1[18];
status =
mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block,
&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_Read(block, buffer1, &len);
if (status != MFRC522::STATUS_OK) {
Serial.print(F("Reading failed: "));
Serial.println(mfrc522.GetStatusCodeName(status));
return;
}
//-------------------------------------
//----------Imprime o nome
for (uint8_t i = 0; i < 16; i++)
{
if (buffer1[i] != 32)
{
```

```
Serial.write(buffer1[i]);
}
}
//--------------------------------------
}
void BluExodia::Write(int block, String guy){
//Preparação para chave
for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
//-------------------------------------------
//Variaveis necessárias
byte buffer[34];
byte len;
//ESCRITA
//----------Ler o que o cabra enviou
len = guy.length();
for (byte i = len; i < 30; i++){
buffer[i] = 0; // pad with spaces
}
guy.getBytes(buffer,len+1);
//-------------------------------------------
//----------Prepara para escrever no bloco e ver se ta certo
//Serial.println(F("Authenticating using key A..."));
status =
mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block,
&key, &(mfrc522.uid));
if (status != MFRC522::STATUS_OK) {
Serial.print(F("Autenticação = Deu ruim "));
Serial.println(mfrc522.GetStatusCodeName(status));
return;
}
//else Serial.println(F("Autenticação = Xuxu beleza "));
//-------------------------------------------
//++++++++++++++++++++++++
//----------Bloco que escreve o que ta no bloco anterior
status = mfrc522.MIFARE_Write(block, buffer, 16);
if (status != MFRC522::STATUS_OK) {
Serial.print(F("Escrita = Ih, deu ruim :( "));
Serial.println(mfrc522.GetStatusCodeName(status));
return;
}
else Serial.println(F("Escrita = Nice :) "));
//-------------------------------------------
//----------Bloco de encerramento
//mfrc522.PICC_HaltA(); // Halt PICC
//mfrc522.PCD_StopCrypto1(); // Stop encryption on PCD
//-------------------------------------------
Serial.print("Fim da Escrita");
Serial.println(" ");
}
void BluExodia::Dump(){
mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
}
int BluExodia::InGame(){
return mfrc522.PICC_IsNewCardPresent();
}
String BluExodia::ReadToStr(int block){
//Preparação para chave
for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
//Variaveis q ira se precisar
byte len;
//-------------------------------------------
```

```
len = 18; //Aparentemente len deve ser igual ao tamanho dos Buffers
//-----------------Pega e ver se ta certo
byte buffer1[len];
status =
mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block,
&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_Read(block, buffer1, &len);
if (status != MFRC522::STATUS_OK) {
Serial.print(F("Reading failed: "));
Serial.println(mfrc522.GetStatusCodeName(status));
return;
}
//-------------------------------------
//----------Armazena o nome na String
String guy="";
for (int i = 0; i < 16; i++)
{
if (buffer1[i]!=0)
{
guy= guy+String(buffer1[i],HEX)+" ";
}
}
//-------------------------------------
return guy;
}
void BluExodia::Readf(int block){
//Preparação para chave
for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
//Variaveis q ira se precisar
byte len;
//-----------------------------------------
len = 18; //Aparentemente len deve ser igual ao tamanho dos Buffers
//-----------------Pega e ver se ta certo
byte buffer1[18];
status =
mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block,
&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_Read(block, buffer1, &len);
if (status != MFRC522::STATUS_OK) {
Serial.print(F("Reading failed: "));
Serial.println(mfrc522.GetStatusCodeName(status));
return;
}
//-------------------------------------
//----------Buffer salvo
for(int i=0; i<16;i++){
slifer[i]= buffer1[i];
}
//-----------------------
delay(100);
```

```
}
char BluExodia::Bufferf(int f){
return slifer[f];
}
String BluExodia::Hexf(){
String guy="";
for (int i = 0; i < 16; i++)
{
if (slifer[i]!=0)
{
guy= guy+String(slifer[i],HEX)+" ";
}
}
return guy;
}
String BluExodia::Namef(){
String guy="";
for (int i = 0; i < 16; i++)
{
if (slifer[i]!=0)
{
guy= guy+String(slifer[i]);
}
}
return guy;
}
//
//Funções W
//
void BluExodia::wClean(int block){
//Preparação para chave
for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
//-------------------------------------------
//Detecta se tem um cartão no leitor, se sim, prosegue
do{
while ( ! mfrc522.PICC_IsNewCardPresent()) {
//Um grande e magnifico nada :)
}
}while (! mfrc522.PICC_ReadCardSerial());
//-------------------------------------------
//Variaveis necessárias
byte buffer[34];
byte len;
//ESCRITA
//----------Ler o que o cabra escreve no Serial
len = 0;
for (byte i = len; i < 30; i++) buffer[i] = 0; // pad with spaces
//-------------------------------------------
//----------Prepara para escrever no bloco e ver se ta certo
status =
mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block,
&key, &(mfrc522.uid));
if (status != MFRC522::STATUS_OK) {
Serial.print(F("PCD_Authenticate() failed: "));
Serial.println(mfrc522.GetStatusCodeName(status));
return;
}
else Serial.println(F("PCD_Authenticate() success: "));
//-------------------------------------------
```

```
//++++++++++++++++++++++++
//----------Bloco que escreve o que ta no bloco anterior
status = mfrc522.MIFARE_Write(block, buffer, 16);
if (status != MFRC522::STATUS_OK) {
Serial.print(F("MIFARE_Write() failed: "));
Serial.println(mfrc522.GetStatusCodeName(status));
return;
}
else Serial.println(F("MIFARE_Write() success: "));
//----------------------------------------
//----------Bloco de encerramento
mfrc522.PICC_HaltA(); // Halt PICC
mfrc522.PCD_StopCrypto1(); // Stop encryption on PCD
//----------------------------------------
Serial.print("Limpo");
}
void BluExodia::wReadToSerial(int block){
//Preparação para chave
for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
//Variaveis q ira se precisar
byte len;
//----------------------------------------
//Detecta se tem um cartão no leitor, se sim, prosegue
do{
while ( ! mfrc522.PICC_IsNewCardPresent()) {
//Um grande e magnifico nada :)
}
}while (! mfrc522.PICC_ReadCardSerial());
//----------------------------------------
len = 18; //Aparentemente len deve ser igual ao tamanho dos Buffers
//----------------Pega e ver se ta certo
byte buffer1[18];
status =
mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block,
&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_Read(block, buffer1, &len);
if (status != MFRC522::STATUS_OK) {
Serial.print(F("Reading failed: "));
Serial.println(mfrc522.GetStatusCodeName(status));
return;
}
//-------------------------------------
//----------Imprime o nome
for (uint8_t i = 0; i < 16; i++)
{
if (buffer1[i] != 32)
{
Serial.write(buffer1[i]);
}
}
//-------------------------------------
delay(500);
mfrc522.PICC_HaltA();
```

```
}


mfrc522.PCD_StopCrypto1();
}
void BluExodia::wWrite(int block, String guy){
//Preparação para chave
for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
//-------------------------------------------
//Detecta se tem um cartão no leitor, se sim, prosegue
do{
while ( ! mfrc522.PICC_IsNewCardPresent()) {
//Um grande e magnifico nada :)
}
}while (! mfrc522.PICC_ReadCardSerial());
//-------------------------------------------
Serial.println(F("Cartão Detectado!"));
//Variaveis necessárias
byte buffer[34];
byte len;
//ESCRITA
//----------Ler o que o cabra enviou
len = guy.length();
for (byte i = len; i < 30; i++){
buffer[i] = 0; // pad with spaces
}
guy.getBytes(buffer,len+1);
//-------------------------------------------
//----------Prepara para escrever no bloco e ver se ta certo
//Serial.println(F("Authenticating using key A..."));
status =
mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block,
&key, &(mfrc522.uid));
if (status != MFRC522::STATUS_OK) {
Serial.print(F("Autenticação = Deu ruim "));
Serial.println(mfrc522.GetStatusCodeName(status));
return;
}
//else Serial.println(F("Autenticação = Xuxu beleza "));
//-------------------------------------------
//++++++++++++++++++++++++
//----------Bloco que escreve o que ta no bloco anterior
status = mfrc522.MIFARE_Write(block, buffer, 16);
if (status != MFRC522::STATUS_OK) {
Serial.print(F("Escrita = Ih, deu merda :( "));
Serial.println(mfrc522.GetStatusCodeName(status));
return;
}
else Serial.println(F("Escrita = Nice :) "));
//-------------------------------------------
//----------Bloco de encerramento
mfrc522.PICC_HaltA(); // Halt PICC
mfrc522.PCD_StopCrypto1(); // Stop encryption on PCD
//-------------------------------------------
Serial.print("Fim da Escrita");
Serial.println(" ");
}
void BluExodia::wDump(){
do{
while ( ! mfrc522.PICC_IsNewCardPresent()) {
//Um grande e magnifico nada :)
```

```
}while (! mfrc522.PICC_ReadCardSerial());
mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
}
String BluExodia::wReadToStr(int block){
//Preparação para chave
for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
//Variaveis q ira se precisar
byte len;
//-----------------------------------------
//Detecta se tem um cartão no leitor, se sim, prosegue
do{
while ( ! mfrc522.PICC_IsNewCardPresent()) {
//Um grande e magnifico nada :)
}
}while (! mfrc522.PICC_ReadCardSerial());
//-----------------------------------
len = 18; //Aparentemente len deve ser igual ao tamanho dos Buffers
//----------------Pega e ver se ta certo
byte buffer1[len];
status =
mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block,
&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_Read(block, buffer1, &len);
if (status != MFRC522::STATUS_OK) {
Serial.print(F("Reading failed: "));
Serial.println(mfrc522.GetStatusCodeName(status));
return;
}
//-------------------------------------
//----------Armazena o nome na String
String guy="";
for (int i = 0; i < 16; i++)
{
if (buffer1[i]!=0)
{
guy= guy+String(buffer1[i],HEX)+" ";
}
}
//-------------------------------------
mfrc522.PICC_HaltA();
mfrc522.PCD_StopCrypto1();
return guy;
}
void BluExodia::wReadf(int block){
//Preparação para chave
for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
//Variáveis que irá se precisar
byte len;
//-----------------------------------------
//Detecta se tem um cartão no leitor, se sim, prossegue
do{
while ( ! mfrc522.PICC_IsNewCardPresent()) {
//Um grande e magnifico nada :)
}
```

```
}while (! mfrc522.PICC_ReadCardSerial());
//-----------------------------------------
len = 18; //Aparentemente len deve ser igual ao tamanho dos Buffers
//-----------------Pega e ver se ta certo
byte buffer1[18];
status =
mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block,
&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_Read(block, buffer1, &len);
if (status != MFRC522::STATUS_OK) {
Serial.print(F("Reading failed: "));
Serial.println(mfrc522.GetStatusCodeName(status));
return;
}
//-------------------------------------
//----------Buffer salvo
for(int i=0; i<16;i++){
slifer[i]= buffer1[i];
}
//-----------------------
delay(100);
mfrc522.PICC_HaltA();
mfrc522.PCD_StopCrypto1();
}
void BluExodia::end(){
mfrc522.PICC_HaltA();
mfrc522.PCD_StopCrypto1();
}
```

Apêndice B - Arquivo.h

```
#ifndef BluExodia_h
#define BluExodia_h
#include <Arduino.h>
#include <SPI.h>
#include <MFRC522.h>
```

```
class BluExodia{
public:
BluExodia(); //Instância, não tem nenhum comando nem nada
void Hearth(); //Aplicada na void setup, inicia funções necessárias
void In();
void Clean(int block); //Limpa o bloco do cartão selecionado no parâmetro
void ReadToSerial(int block); //Ler o bloco selecionado no parâmetro e escreve
no Serial
void Write(int block, String guy); //Escreve uma String num bloco selecionado
void Dump(); //Imprime cada bloco do cartão no Serial
int InGame(); //Função que retorne 1 se tiver um cartão no leitor e 0 no oposto
String ReadToStr(int block); //Retorna em String o que escrito no bloco
selecionado
void Readf(int block); //Ler o bloco do cartão e armazena numa variável Slifer
char Bufferf(int f); //retorna apenas um caracter do slifer, sendo F o vetor
String Hexf(); //Retorna uma string com o que está armazenado em Slifer em
formato Hexadecimal
String Namef(); //O mesmo da Hexf(de cima), mas retorna decodificado
//Os mesmos códigos de cima, porém com linhas de comando que esperam ter
um cartão no leitor para seguir o código
//No caso, se você colocar uns do comandos de baixo no script, o arduino só
avança no código se completar eles
//Diferente de cima, que se caso não houver cartão no módulo, ele ignora e pula o
comando
void wClean(int block);
void wReadToSerial(int block);
```

```
void wWrite(int block, String guy);

void wDump();

String wReadToStr(int block);

void wReadf(int block);

void end();

private:

char slifer[16]; //Variavel que armazena o bloco lido na função Readf

//As funçõs Hexf, Bufferf e Namef retornam o que está armazenado nesta

variavel

};

#endif
```

Apêndice C - Código de Leitura

```
#include <BluExodia.h>

BluExodia guy;

void setup() {

guy.Hearth();

Serial.println("Insira cartão");

}

void loop() {

guy.In();

guy.ReadToSerial(62);

//Serial.println("Informação do Produto limpa");

guy.ReadToSerial(61);

//Serial.println("Informação do valor limpa");
```

```
guy.ReadToSerial(60);

//Serial.println("Informação da moeda limpa");

guy.end();

Serial.println("\nRFID Pronto :)");

Serial.println("Insira outro cartão...\n");

}
```

Apêndice D - Código de Escrita

```
#include <BluExodia.h>
String product;
String val;
BluExodia guy;
void setup() {
guy.Hearth();
Serial.println("Insira o cartão...");
}
void loop() {
product = "Chocolate";
val = "4";
String moeda = "Reais";
guy.In();
//Nome do produto
guy.Write(62,product);
Serial.println("Produto Escrito");
```

```
//Valor do produto
guy.Write(61,val);
Serial.println("Valor do produto Escrito");
//Moeda utilizado
guy.Write(60,moeda);
guy.end();
Serial.println("Unidade da Moeda Escrita");
Serial.println("\nRFID Pronto :)");
Serial.println("Insira outro cartão...\n"); }
```

Apêndice E - Código da Contagem de Estoque

```
#include <LiquidCrystal.h>

#include <MFRC522.h>

#include <BluExodia.h>

LiquidCrystal lcd(7,6,5,4,3,2);

BluExodia guy;

#define quantEs 5 //<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< Quantidade de

produtos que serão registrados

#define but 8

int start;

int ProdutoLidos;

int cont[quantEs];//Vetor que guarda a quantidade lida de cada produto

String estoqProd[quantEs]; //Vetor que guarda o nome dos produtos lidos
```

```
for(int c=0;c<quantEs;c++){
  StringestoqValor[quantEs];//Vetor que guarda o valor dos produtos lidos
  int botao;
  void setup() {
  pinMode(but,INPUT);
  lcd.begin(16,2);
  guy.Hearth();
  lcd.print("Contador");
  //Serial.print("Contador\n");
  lcd.setCursor(0,1);
  lcd.print("de Estoque");
  //Serial.print("de Estoque\n");
  delay(1000);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print((String)quantEs+" produtos para");
  lcd.setCursor(0,1);
  lcd.print("serem contados..");
  //Serial.print((String)quantEs+" produtos para\n");
  //Serial.print("serem contados..");
  delay(700);
  ProdutoLidos = 0;
  start=0;
  }
  void telaInf(){
  /*Função que mostra as informações do estoque atualmente
  enquanto não há cartão no leitor
  */
  String g[quantEs];
  String v[quantEs];
```

```
g[c] = estoqProd[c];
g[c].remove(3); //Armazena numa variavel apenas as três primeiras letras do
produto
v[c] = cont[c];
v[c].remove(3);
}
lcd.print((String)g[0]+" "+g[1]+" "+g[2]+" "+g[3]);
lcd.setCursor(0,1);
lcd.write(' ');
lcd.print((String)v[0]+" "+v[1]+" "+v[2]+" "+v[3]);
delay(400);
/*Serial.print((String)g[0]+" "+g[1]+" "+g[2]+" "+g[3]+"\n");
Serial.write(' ');
Serial.print((String)v[0]+" "+v[1]+" "+v[2]+" "+v[3]+"\n\n");*/
/*Caso já tenha sido registrado todos os produtos previstos na expectativa,
um 'X' fica na borda direita do lcd. */
if(ProdutoLidos>=quantEs){
lcd.setCursor(15,0);
lcd.write('X');
lcd.setCursor(15,1);
lcd.write('X');
}
}
void loop() {
//guy.ln();
//botao = digitalRead(but);
lcd.clear();
```

```
//Serial.print("\n----------------\n");
//Verifica se algum produto já foi registrado
if(estoqProd[0]==estoqProd[(quantEs-1)]){
lcd.print("Aguardando...");
//Serial.print("Aguardando...\n");
} else{
telaInf();
}
//------------------------
bool newProd = false; //Variavel para saber se o o produto que será inserido é
novo
/*No caso, o vetor 1 de cont[] guarda a quantidade de produtos lidos do vetor 1
de EstoqProd[]
*Cada produto novo se registrará como um novo vetor no EstoqProd[]
*e caso outro igual passe, apenas cont[] se somará mais 1.*/
guy.In();
botao = digitalRead(but);
if(botao==LOW){
//guy.In();
guy.Readf(62);
String tProd = guy.Namef();
guy.Readf(61);
String tVal = guy.Namef();
//Serial.print(" UID: ");
//guy.ReadToSerial(0);
lcd.clear();
lcd.print((String)"Produto: "+tProd);
lcd.setCursor(0,1);
```

```
lcd.print((String)"Preco:" + tVal);
//tProd.replace(" ","");
Serial.print("http://192.168.43.144/RFID/compra.php?nome=");
Serial.println(tProd);
guy.end();
delay(3000);
return;
} else{
//Bloco que ler o cartão no leitor e guarda a leitura dos blocos em variaveis
temporariais
//guy.ln();
guy.Readf(62);
String tProd = guy.Namef();
guy.Readf(61);
String tVal = guy.Namef();
guy.end();
//----------------------
Serial.print("http://192.168.43.144/RFID/quantidade.php?nome=");
Serial.println(tProd);
int aux = 0;
while(!Serial.available())
{
delay(1);
aux++;
if (aux > 4000)
break;
}
```

```
int quantidade = 0;
if (Serial.available())
quantidade = Serial.read();
lcd.clear();
lcd.print((String)"Produto: "+tProd);
lcd.setCursor(0,1);
lcd.print((String)"Quant.: " + (String)quantidade);
delay(3000);
//Bloco que verifica se caso o produto no leitor já tenha sido registrado em
estoqProd[]
/*if(tProd!="" && tVal!=""){
int verif = 0;
for(int c=0;c<quantEs;c++){
if(tProd.equals(estoqProd[c]) ){
cont[c]++;
lcd.clear();
lcd.print((String)"Produto: "+estoqProd[c]+"+1");
verif=1;
}
if(c==(quantEs-1) && verif==0 ){
newProd=true;
lcd.clear();
lcd.print("Produto NOVO: ");
verif=0;
}
}
```

```
}else{
lcd.clear();
lcd.print("Erro, cartão inválido...");
delay(800);
return;
}*/
//----------------------
//Caso o produto não tenha sido registrado, registrasse no proximo vetor vazio
if(newProd==true){
lcd.setCursor(0,1);
if(ProdutoLidos<quantEs){
estoqProd[ProdutoLidos] = tProd;
estoqValor[ProdutoLidos] = tVal;
cont[ProdutoLidos] = 1;
ProdutoLidos++;
lcd.print("Adicionado :)\n");
newProd=false;
}else{
lcd.print("Limite Atingido");
newProd=false;
}
}
//----------------------
delay(800);
tProd="";
tVal="";
}
}
```