UNIVERSIDADE FEDERAL DO PARANÁ
SETOR DE TECNOLOGIA
CURSO DE ENGENHARIA ELETRICA

JEAN ALVES DA COSTA

CONTROLADOR DE ACESSO PARA AMBIENTES COM GERENCIAMENTO REMOTO

CURITIBA
2019

CONTROLADOR DE ACESSO PARA AMBIENTES COM GERENCIAMENTO REMOTO

Trabalho de Conclusão de Curso de Graduação em Engenharia Elétrica da Universidade Federal do Paraná, apresentado como requisito parcial a obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Bruno Pohlot Ricobom

CURITIBA

2019

AGRADECIMENTOS

Primeiramente agradeço a minha família que forneceu apoio e auxílio em todo o período da graduação.

Aos amigos que conheci e compartilharam todos os momentos desta jornada em minha vida.

RESUMO

Quando um determinado ambiente possui um acesso controlado, é possível instalar um dispositivo para verificar a identificação do usuário. De forma a permitir ou não a entrada do indivíduo no ambiente alguns métodos podem ser utilizados, tais como: leitores de biometria, senhas numéricas, dispositivos de radiofrequência, entre outros. Dentre as soluções existentes no mercado é possível encontrar determinados controladores de acesso que podem se conectar em uma rede de computadores, de forma a fornecer informações a pessoa que irá administrar os controladores de acesso. O objetivo deste trabalho é apresentar um modelo de controlador de acesso composto por dois módulos, de tal forma que existirá um modulo mestre e outro escravo com o objetivo de dificultar danos e furtos. Também compõe o escopo do projeto uma conexão sem fio do módulo mestre com uma interface gráfica instalada no computador do administrador. As ferramentas empregas constituem-se de um microcontrolador que é capaz de estabelecer uma conexão com uma rede sem fio. Foi possível ao final do projeto exibir dados importantes na interface gráfica como estado da porta, histórico de entrada no ambiente, nível da bateria do circuito de emergência e enviar comandos para o módulo escravo. O projeto dispõe de dois principais diferencias quando comparados aos produtos existentes no mercado, uma conexão em uma rede sem fio de computadores e a divisão do hardware de processamento entre módulo mestre e escravo, é o módulo escravo que faz a interface de identificação da usuário que deseja entrar no ambiente, caso ele seja danificado a porta não abre, pois o controle é feito pelo módulo mestre.

Palavras-chave: Controlador de Acesso, rede sem fio, mestre, escravo.

ABSTRACT

When a particular environment has controlled access, a device can be installed to verify user identification. In order to allow the individual to enter the environment or not, some methods may be used, such as: biometric readers, numeric passwords, radio frequency devices, among others. Among the existing solutions on the market it is possible to find certain access controllers that can connect to a computer network in order to provide information to the person who will administer the access controllers. The objective of this work is to present an access controller model composed of two modules, such that there will be a master module and a slave module in order to make damage and theft more difficult. The scope of the project also comprises a master module wireless connection with a graphical interface installed on the administrator's computer. The tools employed are a microcontroller capable of establishing a connection to a wireless network. It was possible at the end of the project to display important data in the graphical interface such as door status, environment entry history, emergency circuit battery level and sending commands to the slave module. The project has two main differences when compared to existing products in the market, a connection in a wireless computer network and the division of the processing hardware between master and slave module, it is the slave module that makes the user identification interface that want to enter the environment, if it is damaged the door does not open because the control is done by the master module.

Keywords: Access Controller, wireless network, master, slave.

LISTA DE FIGURAS

LISTA DE SIGLAS

GUI – Grafic User Interface
IEEE – Institute of Electrical and Electronics Engineers
UART – Universal Asynchronous Receiver/Transmitter
GPIO – General Purpose Input/Output
SPI – Serial Peripheral Interface
RFID – Radio-Frequency IDentification
A/D – Analogic to Digital converter (Conversor analógico digital)
STA – Station
AP – Acess Point
SSID – Service Set Identifier

SUMÁRIO

1 INTRODUÇÃO

O campo de estudo que envolve o projeto é área de controle de acesso de ambientes. Há várias vantagens em obter o gerenciamento de um determinado local, dentre elas:

a. Monitorar entrada e saída de funcionários e visitantes;
b. Identificação de todos os ingressantes;
c. Tornar os ambientes mais seguros, privados e funcionais;
d. Controle de trânsito de pessoas;
e. Economia para contratação de mão de obra.

Os sistemas existentes atualmente envolvem: senhas, biometrias, cartões magnéticos, entre outros que derivados. As soluções encontradas no mercado para dispositivos controladores de acesso estão limitadas a um único módulo, que realiza o controle e processamento. Desta forma toda unidade eletrônica está exposta na localidade externa do ambiente, não representando uma eficácia total no sistema de segurança.

Quando o controlador de acesso é formado por apenas um sistema um individuo que não tenha acesso ao ambiente e deseje adentrar pode danificar o módulo de tal forma que o relé responsável por acionar a trava eletromagnética não atue, o que implica diretamente na abertura da porta.

O objetivo do projeto é criar um protótipo composto por dois módulos, um principal e um secundário, seguindo a analogia de sistemas mestre-escravo. Neste sistema o módulo principal (mestre) está localizado na parte interna do ambiente e está conectado fisicamente com o módulo secundário (escravo) que está localizado na parte externa do ambiente.

O projeto contempla também um novo recurso para estabelecer uma comunicação através de uma rede sem fio entre o módulo mestre em cada ambiente. Com o dispositivo conectado em uma rede wifi é possível alimentar uma interface gráfica que forneça informações a um administrador dos controladores de acesso. Onde este individuo precisa ter conhecimentos básicos de rede de computadores, sabendo trabalhar com conceitos como endereço IP, máscara de rede e gateway.

Nesta interface gráfica o administrador terá informações como o estado da porta, se ela está aberta ou fechada, qual o nível da bateria que supre o sistema em uma eventual queda de energia, um arquivo informando o histórico de acesso do ambiente onde é possível descobrir quais usuários tiveram acesso ao ambiente e em quais datas e horários. O escopo do projeto também engloba a abertura remota de uma determinada porta.

A base da pesquisa é oferecer uma solução com melhor desempenho em sistemas de segurança para controle de acesso, de tal forma que um indivíduo externo não tenha acesso para manipular os sistemas de hardware e software. Os critérios de validação formam a base de verificação do indivíduo. Os critérios de verificação da identificação se resumem em:

> “Algo que só o indivíduo sabe: senhas numéricas ou alfanuméricas...”
>
> “Algo que só o indivíduo possui: objetos físicos únicos...”
>
> “Algo que só o indivíduo é: características biométricas...”
>
> (GALHARDO,2011)

Atualmente os sistemas envolvem: senhas, biometria, cartões magnéticos, entre outros.

De forma a restringir o escopo do projeto para possibilitar resultados dentro do prazo, os critérios de verificação escolhidos são dois, o primeiro é uma senha de até seis dígitos onde o usuário precisa digitar no teclado e o outro é uma tag RFID (chave ou cartão) que o usuário deve aproximar do leitor.

As questões investigadas por este trabalho são: é possível desenvolver um sistema de controle de acesso de ambiente composto por dois módulos? Quais são os ganhos efetivos em termos de segurança? É possível estabelecer uma comunicação sem fio eficaz entre o sistema embarcado e a interface gráfica? É possível gerenciar estes dados através de uma rede sem fio?

1.1 JUSTIFICATIVA

A hipótese que fundamenta a pesquisa é a suposição que dividir um sistema eletrônico comumente unificado, em outro formado por dois módulos que seguem o modelo mestre-escravo torna o sistema mais seguro. A fragilidade dos controladores existente no mercado existe devido ao fato que todo o hardware está construindo em um único módulo, o que implica diretamente que o atuador que precisa fornecer a corrente para a bobina que está na trava eletromagnética pode ser danificado.

É importante destacar que a conexão de rede destes controladores é feita via conexão física, implicando em um custo para realizar o cabeamento e outras alterações como passagem de caneletas e mudanças na planta civil do usuário.

Na figura 1 é exibido dois controladores de fabricantes distintos que possuem as limitações já citadas acima.

Modelo: MD 5705
Fabricante: MADIS

Fabricante: HENRY
Modelo: Primme SF Acesso

Figura 1: Modelos MD5705 e Prime SF Acesso.
Fonte: Madis, 2019. Henry Equipamentos e Sistemas, 2019

1.1.2 INFRAESTRUTURA

A infraestrutura utilizada nos controladores apresentados está na figura 2 abaixo, onde existe a tubulação da alimentação de corrente alternada e a tubulação de dados onde está o cabo para conexão no módulo. Na parte superior está localizada a trava magnética (bobina) responsável pela abertura ou não da porta, na região inferior é possível visualizar os módulos de leitura responsáveis pela identificação do usuário.

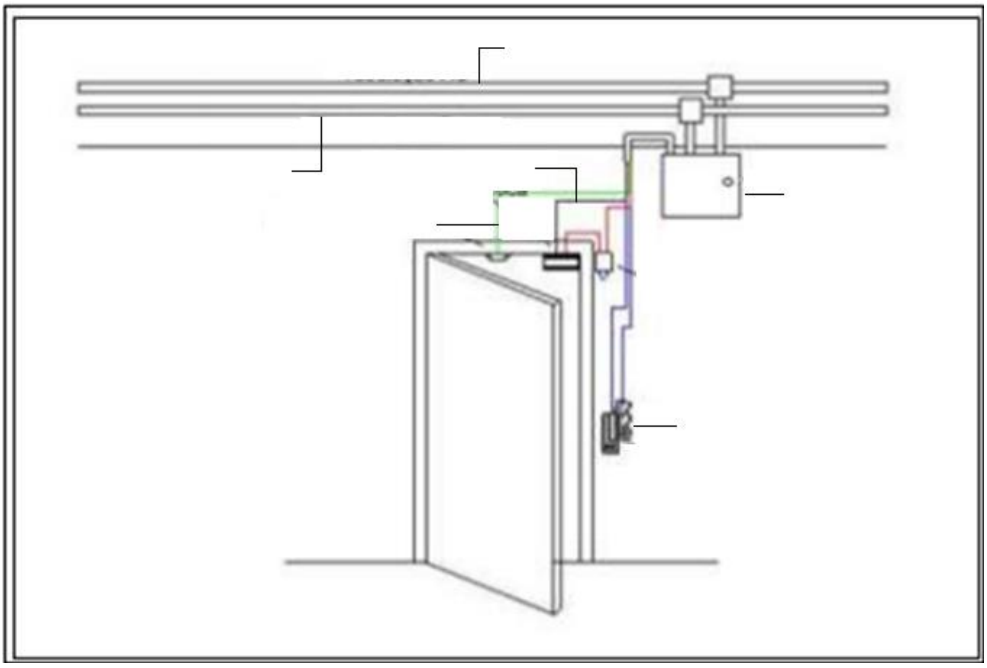

Figura 2: Infraestrutura

Fonte: Galhardo,2011, p. 40.

1.1.3 ARQUITETURA BÁSICA DE SISTEMAS DE CONTROLE DE ACESSO

Uma estrutura padrão de redes para gerenciamento de sistemas de controle de acesso é apresentada na figura 3 abaixo, é necessário um servidor onde é armazenado o sistema de banco de dados que informam quais são os usuários cadastrados, também está presente um protocolo de comunicação TCP/IP com as controladoras.

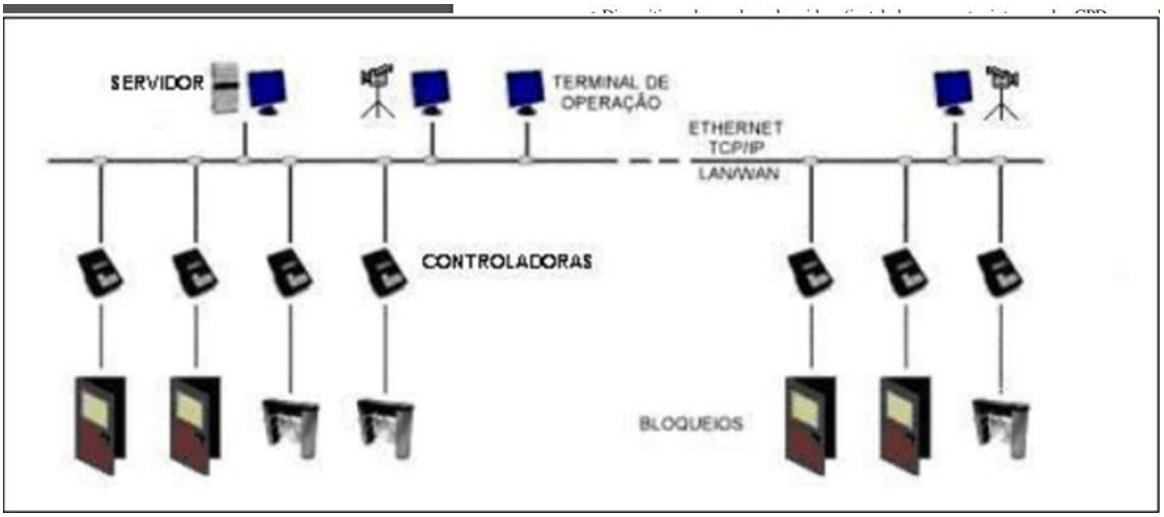

Figura 3: Arquitetura

Fonte: Galhardo,2011, p. 15.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Criar um dispositivo conectado a uma rede sem fio capaz de controlar o acesso de um determinado local, composto por dois módulos, onde haja um módulo principal instalado na parte interna do ambiente e outro na área externa.

1.2.2 Objetivos Específicos

Os objetivos específicos do trabalho são:

a) Desenvolver módulo de processamento;

b) Construir interface gráfica;

c) Elaborar hardware de interface composto por módulo externo, teclado, RFID e display;

d) Desenvolver comunicação sem fio;

e) Construir circuitos de proteção, acionamento e leitura.

A figura quatro abaixo é apresentado um diagrama de blocos do funcionamento esperado do projeto, com o módulo externo(escravo) realizando a interface (display, teclado, RFID) com o usuário e está conectado ao módulo principal (mestre) que é o responsável por fornecer as instruções para o escravo e verificar quais usuários tem acesso ao ambiente. O módulo mestre está conectado em uma rede sem fio e alimenta uma interface gráfica para um eventual administrador do sistema.

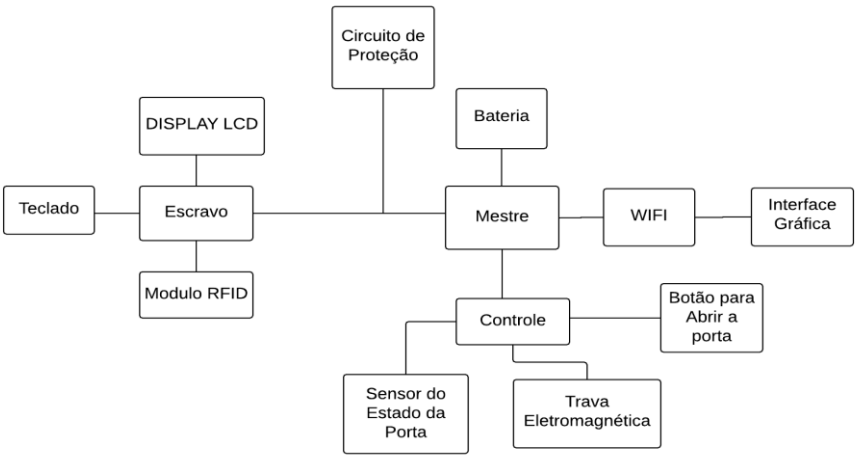

Figura 4: Diagrama de Blocos.

Fonte: O autor, 2019.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 GUI (GRAFIC USER INTERFACE)

Uma interface gráfica é a ferramenta que permite que um usuário interaja com um programa construído a partir de uma linguagem de programação como C, C++, Java, PHP, Python, entre outras. É a GUI que trata eventos como entrada de informações, cliques do mouse, tecla pressionadas e exibe informações que o programador acredite ser importante para conhecimento do usuário. A linguagem de programação escolhida para o projeto foi o Python, devido a vantagens como:

i. Programação Orientada a Objeto:

A orientação a objeto é fundamentada em classes, objetos e métodos. Para (Ricarte,2001) "*um objeto é uma instância de classe que determinam qual informação o objeto contém e como ele pode ser manipulado"*. Através desta linguagem é possível usar conceitos como herança, onde métodos e objetos de uma determinada classe podem ser utilizados em outras classes. Também é possível usar encapsulamento onde toda a informação necessária para manipular um objeto está contida em um único elemento. Outro elemento disponível na orientação a objeto é o polimorfismo que implica que duas ou mais classes podem empregar o mesmo método de uma terceira classe, mas com aplicações diferentes.

ii. Alto nível:

Linguagens de alto nível por definição se distanciam da linguagem Assembly, o que para esta aplicação é uma vantagem, onde o autor não precisa trabalhar com instruções de processador ou registrador;

iii. Software livre:

A licença para o Python é apoiada pela GNU (General Public Licence) facilitando a reutilização e distribuição dos códigos.

iv. Indentação:

Para separação de blocos de código é necessário apenas um recuo em relação a margem, não é necessário o uso de chaves como em outras linguagens;

v. Tipagem dinâmica:

Não é obrigatório fazer a inicializar do tipo (int, char, float) de uma variável ao declarar ela no escopo, basta atribuir um valor ou caractere a ela.

Uma comparação direta entre as linguagens Python e C feita por < http://www.inf.ufes.br/~vitorsouza/wp-content/uploads/teaching-lp-20142-seminario-python.pdf> está abaixo:

“Em C:

```
#include <stdio.h>
int main(){
char nome[200];
printf("Digite seu nome: ");
scanf("%s", nome);
printf("\n %s, Seja bem vindo(a)\n", nome);
return 0;
} “”
```

“Python:

```
nome = raw_input('Digite seu nome: ')
print ("\n%s, Seja bem vindo(a) :)\n" % nome); “
```

Como pode ser observado o Python possui uma sintaxe mais direta e simples o que facilita a construção e reutilização do código

2.1 TKINTER

O Tkinter é uma biblioteca do Python que permite a estruturação de uma GUI, ela é nativa da linguagem o que facilita o uso de outras ferramentas. A base de organização do Tkinter é a estrutura que segue analogia de containers, onde todas as informações (janelas, botões, textos) devem estar dentro de um container. Os principais conceitos durante o desenvolvimento com Tkinter são:

i. Container: Como já foi citado toda a organização da GUI com o Tkinter é feita através de container, onde todos os objetos das classes precisam estar em um container para serem exibidos;

ii. Widget: É um elemento que permite a interação com o usuário, como um botão, caixa de texto, ícone, etc.;

iii. Loop de Eventos: É o responsável por verificar se aconteceu algum evento, originado da comunicação com o usuário;

iv. Manipulador de Eventos: Caso uma ação relacionada a um evento aconteça, o manipulador deve executá-la, exemplo: Após um clique em botão uma mensagem é exibida.

A figura 5 ilustra como é a estrutura de container em uma janela Tkinter:

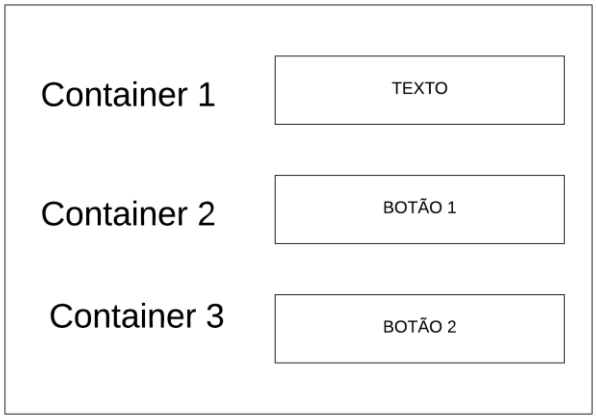

Figura 5: Conceito de Container.

Fonte: O autor, 2019.

Um exemplo de uma janela construída com o Tkinter e empregando um tratador de eventos pode ser visto na figura 6 abaixo:

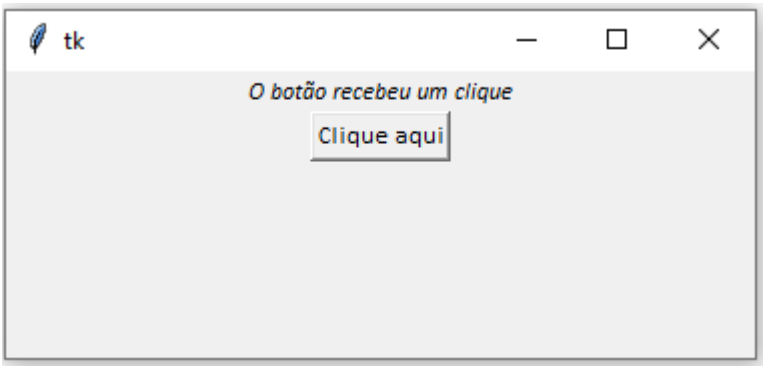

Figura 6: Exemplo de uma janela Tkinter.

Fonte: O autor, 2019.

No exemplo apresentado o texto “o botão recebeu um clique” está no primeiro container, enquanto o botão “Clique aqui” está no segundo container.

2.2 REDES DE COMPUTADORES

Redes de computadores pode ser definida como uma infraestrutura que fornece serviços para diversas aplicações (KUROSE, 2010). Desta forma uma rede conecta diversos dispositivos eletrônicos (hosts) que podem troca informações através de comutadores de pacotes (roteadores) quando houverem links de comunicação (enlaces). Dentro do campo de rede de computadores essas informações são chamadas de pacotes e a troca de pacotes é organiza pelos protocolos. Protocolo é a forma de se comunicar, podendo ser uma informação eletrônica ou humana como ilustra a figura 7 abaixo.

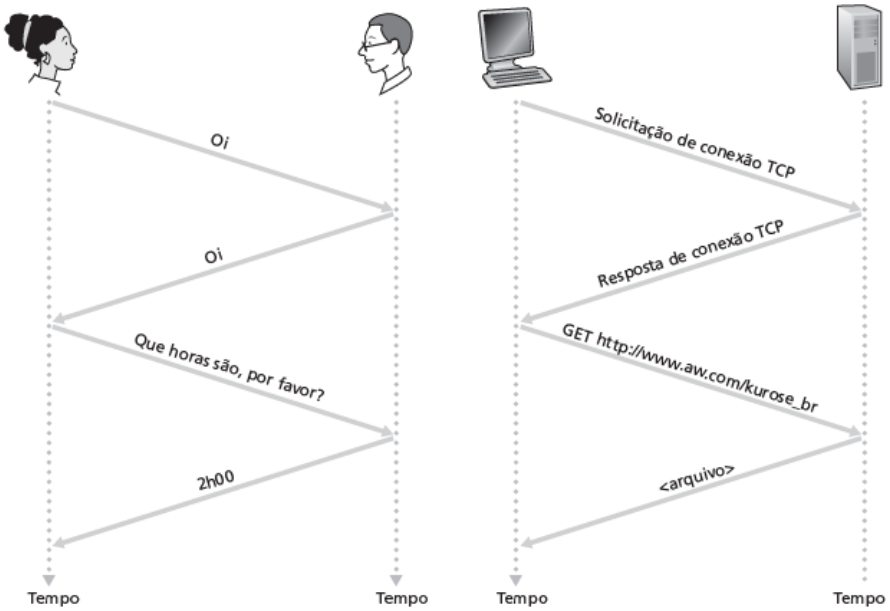

Figura 7: Protocolo

Fonte: Kurose,2010, p. 30.

Na estrutura acima apresentada uma requisição é feita e após é aguardada uma resposta. Isto ilustra a importância do protocolo, que diz como a relação pergunta resposta deve acontecer.

Dentro da esfera de redes de computadores é necessário definir um conceito denominado relação cliente e servidor. Um cliente é o sistema que solicita informações (pacotes) há um outro sistema, servidor, que deve responder de acordo com o protocolo estabelecido. Para exemplificar é apresenta a figura 8 onde um cliente, computador, faz uma requisição há uma página da internet.

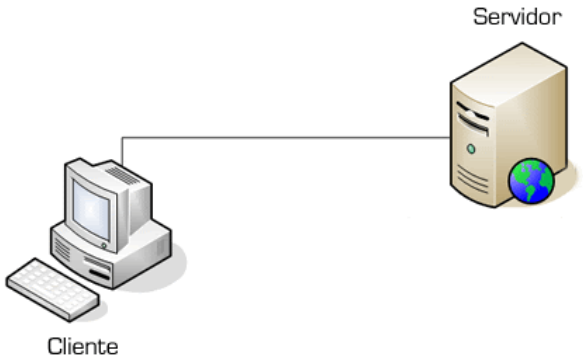

Figura 8: Cliente e Servidor

Fonte: Redes de Comunicação – 2016

Uns dos protocolos mais utilizados é o TCP (Protocolo de Controle de Transmissão), que possui as seguintes características:

i. Orientado a conexão sempre garantindo a entrega dos pacotes;
ii. É necessário estabelecer uma sincronização para estabelecer e fechar a conexão;
iii. Full Duplex: Transferência simultânea entre servidor e cliente.

A estrutura do protocolo TCP é fundamentada em um datagrama que apresentada na figura 9 abaixo:

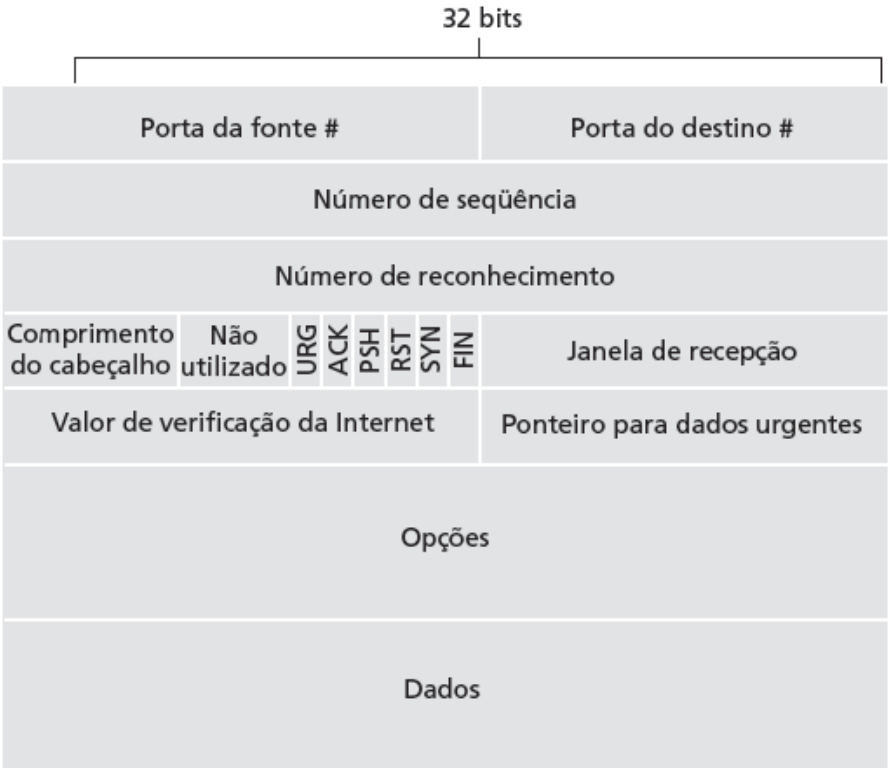

Figura 9: Datagrama TCP
Fonte: Kurose,2010, p. 247.

Dentre os campos da figura acima, é necessária destacar os campos porta de destino e fonte que são uns dos parâmetros utilizados durante a conexão sem fio entre os módulos mestre e escravo. As portas determinam qual aplicação do host usa os dados recebidos e enviados. Exemplo uma página HTTP utiliza a porta 80 para enviar e dados em um navegador web.

Dentro da conexão em rede cada cliente deve ter um endereço específico para ser possível a troca de pacotes quando há vários clientes. A figura 10 abaixo ilustra essa conexão com diversos cientes:

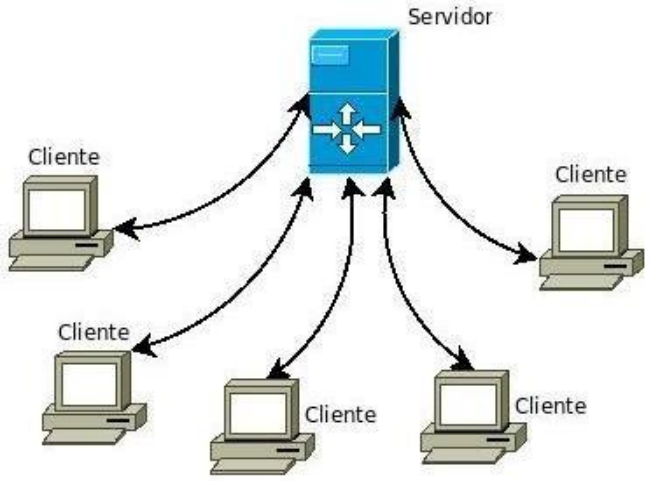

Figura 10: Conexão de vários clientes há um servidor.

Fonte: Researchate - Repositório Digital

É preciso fazer o uso de um segundo protocolo para enviar pacotes entre cliente e servidor com o TCP, este segundo protocolo é um protocolo de endereçamento denominado Protocolo de Internet (IP). Este protocolo é o responsável por fornecer um endereço dentro de uma rede para cada cliente se conectar ao servidor. Assim como o TCP, o protocolo IP também é estruturado através de um datagrama, que está na figura 11 abaixo.

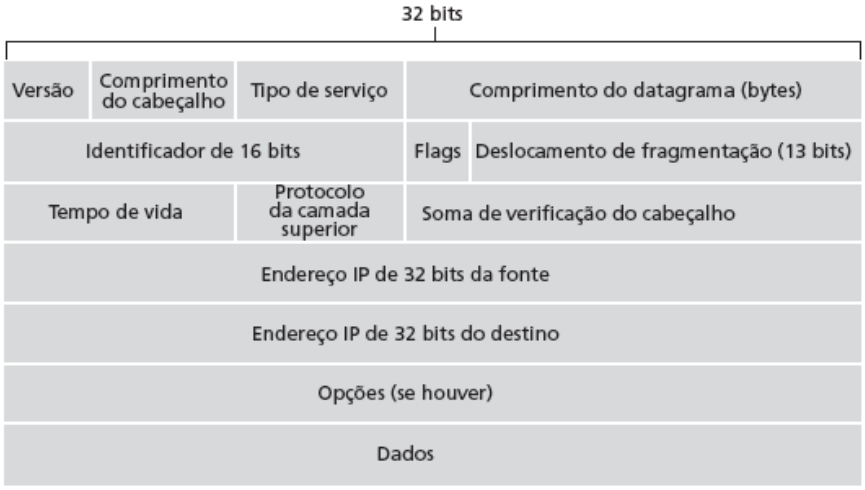

Figura 11: Datagrama IP

Fonte: Kurose,2010, p. 248.

Aqui entra o segundo parâmetro estratégico para o funcionamento da comunicação sem fio implementada no projeto, o endereço IP. Um endereço IP possui um espaço de 32 bits organizado em 4 bytes separados por pontos para determinar um endereço de rede, assim existe um total de 2^{32} endereços disponíveis. Dentro da literatura de redes existe o conceito de sub rede onde existem hosts conectados através de um roteador em uma mesma faixa de endereços. Para determinar qual a quantidade de endereços disponíveis é necessária utilizar uma máscara de rede que indica qual a quantidade de bits disponíveis que podem ser usados em uma rede. Um exemplo de cálculo de endereços é exposto abaixo:

Endereço IP: 223.1.1.0
Mascara: 255.255.255.0
Existem 255 endereços disponíveis na rede 223.1.1.x

Uma representação usual para o endereço IP e máscara juntos é colocar da seguinte forma 223.1.1.0/24 que indica que os 24 bits mais a esquerda do endereço IP definem o endereço da máscara de rede. É a máscara que define o tamanho da sub rede. Na figura 12 abaixo é possível observar três sub redes existentes que são interligados através de um roteador.

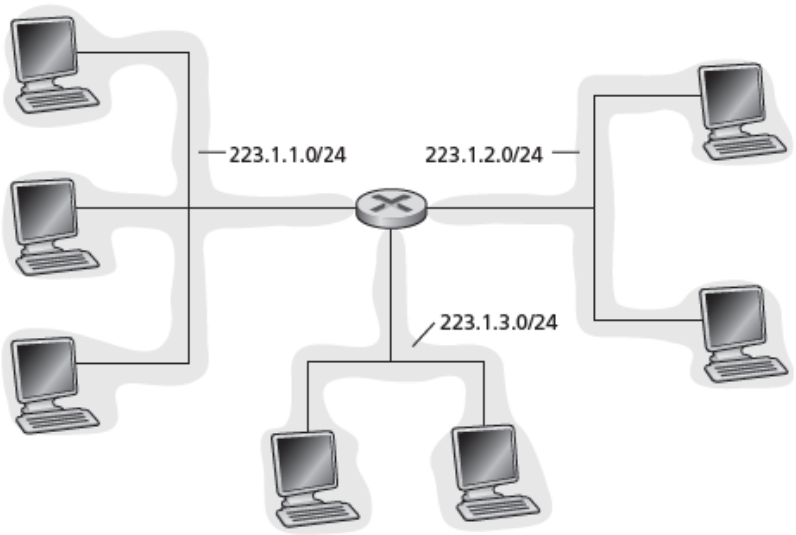

Figura 12: Sub rede.
Fonte: Kurose,2010, p. 253.

Esta forma de organização de endereços IP em 4 bytes é denominada IPv4, também existem os endereços que seguem o formato Ipv6. Quando um cliente precisa se conectar a um servidor. Dentro desta sub rede o primeiro endereço é o gateway (roteador), ele é o responsável por encaminhar pacotes para endereços que estão fora da sub rede. No exemplo abaixo o host 192.168.0.101 pode se comunicar direto com o host 192.168.0.102, mas caso algum deles deseje enviar uma requisição para outra rede, no caso do exemplo a internet, é necessário que esse pacote passe pelo gateway a figura 13 abaixo ilustra

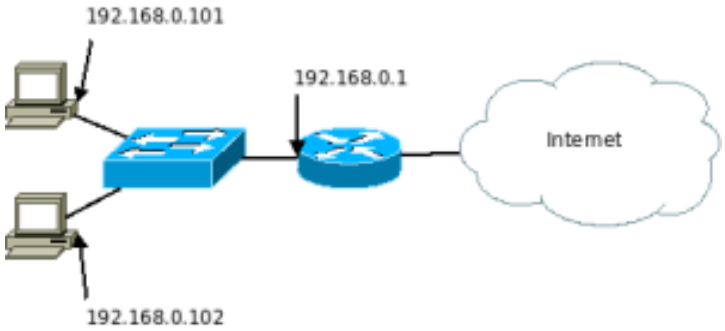

Figura 13: Funcionamento do Gateway.

Fonte: Plano em Foco – Gateway Padrão

Um determinado host pode receber um endereço IP fixo ou dinâmico através do Protocolo de Configuração Dinâmica de Hospedeiro (DHCP). No modo DHCP o roteador da rede fornece um endereço IP ao host quando ele se conecta na rede. A escolha do autor foi optar por endereços IP fixos em uma rede wifi conhecida, facilitando o desenvolvimento do projeto, como por exemplo o rastreamento de pacotes para cálculo das velocidades de transmissão.

Uma estrutura usual no trabalho com redes é a divisão do sistema no formato OSI que emprega a estrutura de camadas. O modelo OSI foi criando nos anos de 1970 e dividi as funções de uma rede de computadores em sete camadas, cada camada tem uma atribuição, por exemplo o protocolo TCP já mencionado está na camada quatro, a camada de transporte.

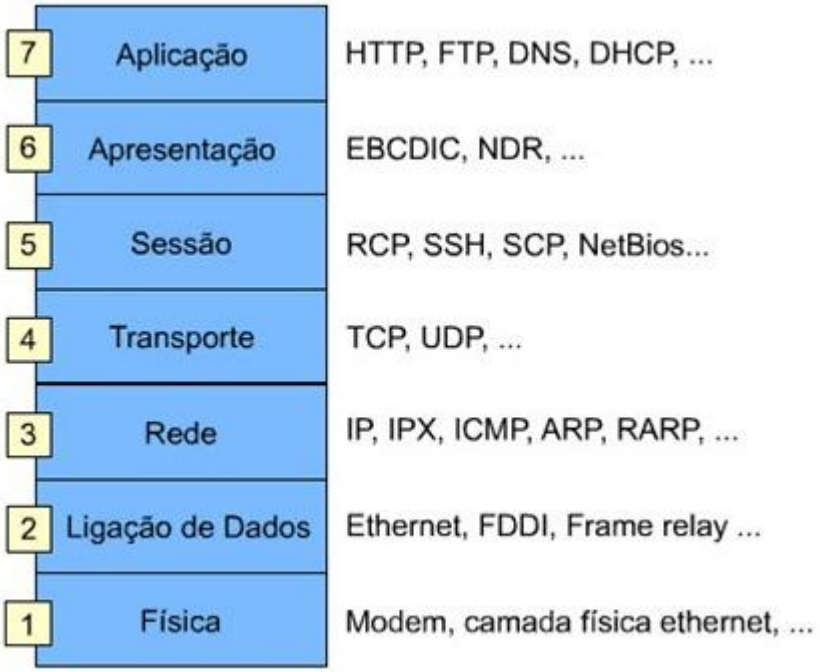

Figura 14: Modelo de Camadas.
Fonte: PPLWARE – O Modelo OSI

A 7 camadas do Modelo OSI são:

i. Aplicação: Responsável interagir com o usuário;
ii. Apresentação: Realiza a compreensão dos dados oriundos dos pacotes;
iii. Sessão: Gerencia as sessões abertas pelo usuário;
iv. Transporte: É a camada que efetivamente faz o transporte dos pacotes;
v. Rede: Controla os dados entre diferentes hosts e redes;
vi. Enlace: Detecta erros oriundos camada física;
vii. Física: Determina como dever ser a conexão física dos cabos, roteadores e outros dispositivos.

Para haver uma conexão entre a camada de aplicação onde os dados são inseridos pelo usuário, neste caso pela GUI, até a camada de transporte é necessário o uso de um dispositivo chamado socket de redes. O socket é uma A.P.I (Interface de Programação de Aplicativos) que está presente entre as camadas de transporte e aplicação, com um endereço IP e um número de porta de um servidor ou cliente o sistema operacional estabelece uma conexão. O socket é a ferramenta

que permite que após estabelecer uma conexão um determinado host possa receber e enviar informações para outro host conectado na mesma rede. A figura 15 ilustra esse processo onde um servidor ou cliente recebe informações por um processo que está na camada de aplicação e o socket carrega esses dados até a camada de transporte onde está o protocolo TCP/IP que é o responsável por encaminhar os pacotes até o destino (servidor ou cliente).

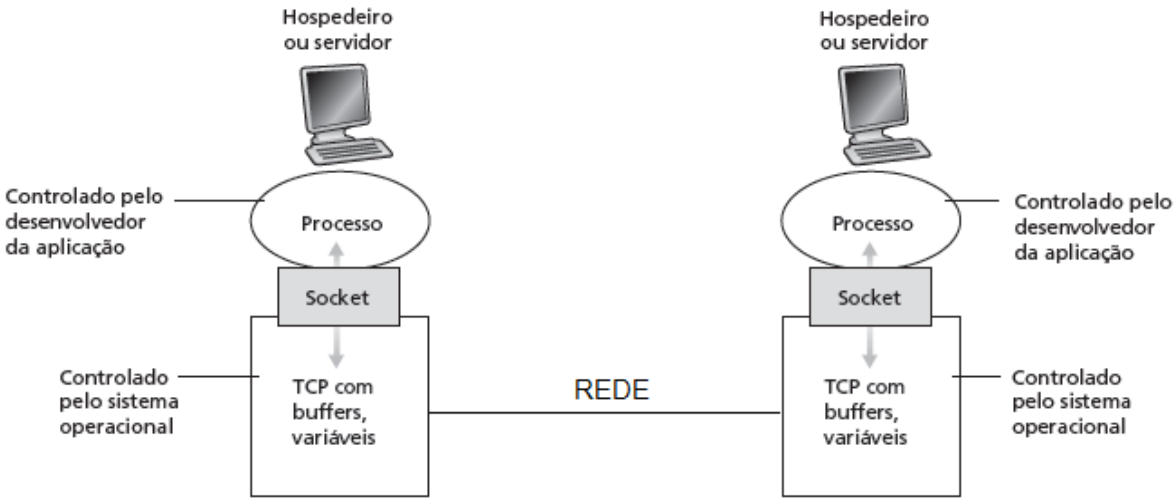

Figura 15: Socket de Redes.

Fonte: Kurose,2010, p. 66.

3 METODOLOGIA

Nesta sessão será abordado os tópicos que direcionaram o desenvolvimento do projeto:

3.1 GUI (INTERFACE GRÁFICA DO USUÁRIO)

A Interface gráfica foi construída utilizando os recursos do Tkinter através da linguagem python, a programação orientada a objeto permitiu a criação de vários objetos que são utilizados em outros pontos do código. A GUI acessa um arquivo de texto onde estão armazenados os endereços de host, para verificar quais são os endereços disponíveis. A porta utilizada pelo socket é fixa, a 25565, que não tem uma aplicação oficial.

3.2 MICROCONTROLADOR ESP8266EX

O autor optou pela escolha do microcontrolador ESP8266EX fabricado pela empresa Expressif para implementar o modulo mestre que é o responsável pela comunicação com a interface gráfica. É o ESP o responsável pela comunicação via serial com o módulo escravo. Este microcontrolador possui um conversor A/D de 10 bits, clock de até 80 MHz, UART máxima de 4Mbps, I2C e PWM. A figura 16 apresenta o chip:

Figura 15: Microcontrolador ESP8266EX

Fonte: ESP8266EX. Expressif, 2019

3.2.1 ESP 12-E

O microcontrolador ESP8266EX pode ser adaptado para um trabalhar com uma rede sem fio, nos padrões 802.11 b/g/n, sendo este um fator determinante para a escolha deste ser o microcontrolador mestre a ser utilizado no projeto, dado que um dos objetivos é implementar um controlador de acesso sem fio. Quando o ESP8266EX está encapsulado junto a com a antena pela empresa AITHINKER, o conjunto passa ser chamado de ESP 12-E. Junto a este módulo também existe um chip de memória flash de 4MB. A figura x ilusta esse conjunto e a figura 2

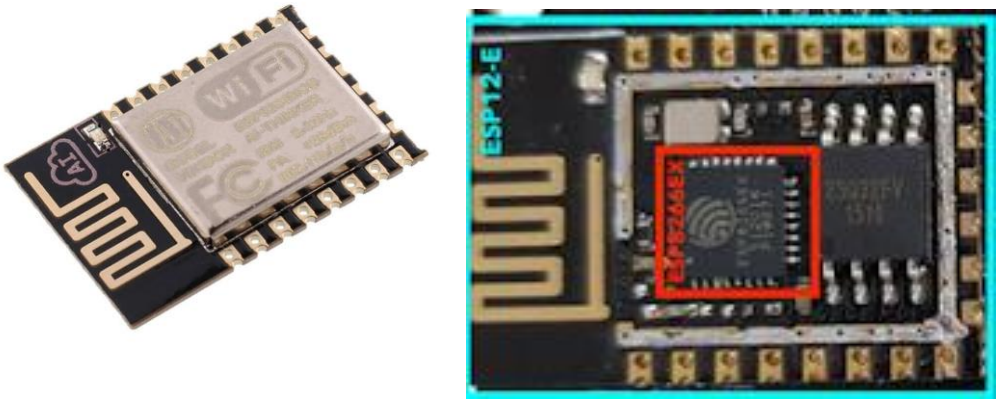

Figura 15: ESP 12-E.

Fonte: ESP 12-e. AITHINKER 2019

3.2.1 NODEMCU

Para facilitar o debug durante o desenvolvimento com o ESP8266 existe uma LauchPad intitulada NodeMCU que possuiu o conversor serial USB CH340, assim é possível fazer o download do firmware via USB.

Figura 16: NodeMCU.
Fonte: LauchPad NodeMCU. Loja Curto Circuito, 2019

3.2.2 COMUNICAÇÃO SEM FIO

A comunicação sem fio é estabelecida entre o ESP8266 e a GUI através de um socket, ambos estão na mesma rede sem fio. O ESP8266 é o servidor que irá repassar as informações para a GUI que é o cliente que faz a requisições. O ESP8266 sempre está aguardando uma conexão de um cliente. O protocolo de comunicação entre a GUI e o ESP8266 é o tcp/ip pois sempre garante a entrega dos pacotes e é orientado a conexão. O ESP8266 pode operar tanto como um ponto de acesso fornecendo uma rede sem fio para outros dispositivos se conectarem ou no modo estação onde ele se conecta em uma rede sem fio existente. Operando no modo estação é preciso configurar os endereços de rede IP, gateway e máscara e fornecer o nome e senha da rede sem fio existente para o ESP se conectar.

Para o desenvolvimento deste controlador de acesso o ESP opera no modo estação, o administrador ou o responsável pela instalação do controlador deve conhecer o nome da sua rede sem fio e senha. Também é preciso ter conhecimento de um endereço IP disponível, a mascará da rede e o gateway utilizado. A figura 17 ilustra como é a operação no modo estação.

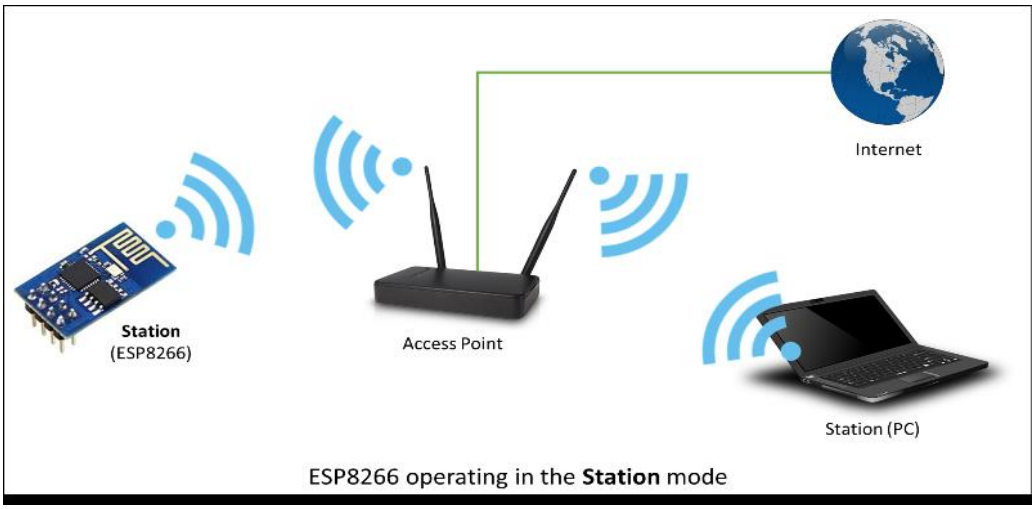

Figura 17: Modo Estação.

Fonte: Modo de Operação. NodeMCU, 2019

3.2.3 TRABALHANDO COM A MEMÓRIA EEPROM

Como definido no escopo do projeto é necessário um banco de dados onde está armazenado os nomes dos usuários e suas respectivas senhas, via teclado e RFID. Essas informações não podem ser perdidas quando o microcontrolador for resetado ou desligado. Para resolver esse problema foi optado por trabalhar com a memória EEPROM do ESP, é importante ressaltar que este microcontrolador não possui memória EEPROM nativa, a EEPROM é simulada a partir do chip de memória FLASH externo. Foi implementado uma rotina para realizar a gravação e leitura da EEPROM descrita abaixa:

O valor máximo que a EEPROM pode assumir são 4096 bytes, como cada endereço de memória ocupada um byte, o total de endereços é 4096. O valor padrão escrito em cada endereço é 255, está é uma informação importante para descobrir quais endereços já foram gravados.

O primeiro bloco de informações importantes são as configurações de rede, que assim como o banco de dados não podem ser perdidas após um reset do microcontrolador. As configurações de rede ocupam um valor conhecido, dado que os três parâmetros endereço IP, gateway e máscara possuem um tamanho fixo. O cálculo é demonstrado abaixo:

Endereço IP: 000.000.000.000 = 15 caracteres (incluindo o ponto) => 15 endereços.

Gateway: 000.000.000.000 = 15 caracteres (incluindo o ponto) => 15 endereços.

Máscara: 000.000.000.000 = 15 caracteres (incluindo o ponto) => 15 endereços.

Nome da rede sem fio: 32 caracteres é o máximo (IEEE,2016) => 32 endereços.

Senha da rede sem fio: 64 caracteres é o máximo (IEEE,2016) => 32 endereços.

Assim o total de endereços que a configuração da rede sem fio ocupa é 106, o que implica que do endereço 0 até o endereço 105 nada pode ser gravado.

O segundo bloco de informações é o tamanho do banco de dados dos usuários, como o tamanho máximo da EEPROM é 4096 endereços e as configurações de rede ocupam 106 endereços, restam 3990 endereços para serem usados pelo banco de dados. Para cada usuário foi definido que o nome terá um tamanho máximo de 36 caracteres, senha via teclado com 6 caracteres e a identificação gerada pelo sistema RFID é padrão com o tamanho de 11 caracteres, totalizando um total de 53 caracteres. O tamanho do banco de dados é calculado abaixo:

Nº de usuários = (3990 endereços disponíveis) / (53 endereços por usuário) = 75 usuários.

3.3 PROTOCOLO DE COMUNICAÇÃO SERVIDOR CLIENTE

Para que seja possível a comunicação entre o ESP e a GUI precisa existir um protocolo de comunicação para ambos compreenderem as mensagens recebidas entre servidor (ESP) e cliente (GUI). O cliente sempre está enviando uma mensagem padrão, que é “Status,Nada” e o servidor responde com as informações que ele tem armazenado no buffer interno, que são “Estado da Porta, Nível da Bateria, Usuário que abriu a porta”. É com essas informações que é possível

alimentar as informações da GUI e escrever em um arquivo de texto o histórico de usuários que entraram no ambiente. A mensagem somente é alterada quando o usuário clica no botão Abrir Porta, a mensagem passa a ser “Status,Abrir” somente para o host selecionado e após retorna para “Status,Nada”. A figura 18 exibe este funcionamento.

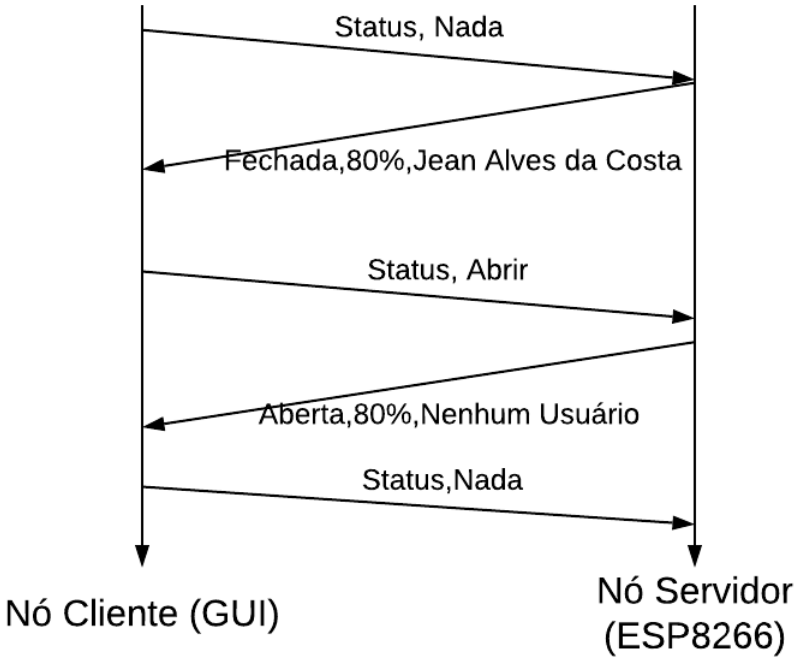

Figura 18: Protocolo entre Servidor e Cliente.
FONTE: O AUTOR, 2019

3.4 NÍVEL BATERIA, SENSOR DA PORTA E ATUADOR DA BOBINA

Duas GPIOS são usadas para atuação no controlador de acesso da porta, a primeira é referente ao sensor que indica se a porta está aberta ou fechada, desta forma essa GPIO é configurada para sempre estar em nível alto (3.3V) e quando a abertura da porta o nível da porta vai para 0V. A segunda GPIO é configurada para nível baixo (0V) e somente vai para nível alto (3.3V) quando a GUI enviar a mensagem informando para abrir a porta, esta porta é conectada há um sistema atuador que abre a porta. Ambas as informações são enviadas para a GUI com o uso do socket

A tensão de operação deste microcontrolador é de 3.3V, a entrada do ADC suporta tensões de entrada de até 1V, mas o chip já conta com um divisor resistivo interno para adequar o valor para a faixa correta. A figura 19 ilustra o divisor resistivo.

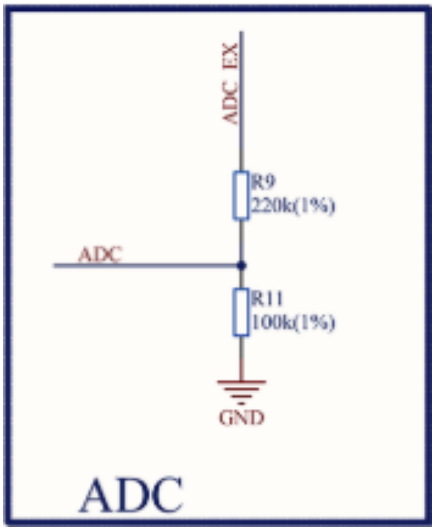

Figura 19: ADC NodeMCU.

Fonte: GitHub – NodeMCU Kit

3.5 MODULO ESCRAVO(ATMEGA2560)

A proposta inicial do projeto era usar o microcontrolador PIC18F2550 para o módulo escravo, o autor optou pela mudança para o microcontrolador ATMEGA2560 com base na experiência do mesmo com esta ferramenta. O esquemático do chip já encapsulado na launch pad Arduino está na figura 20 abaixo.

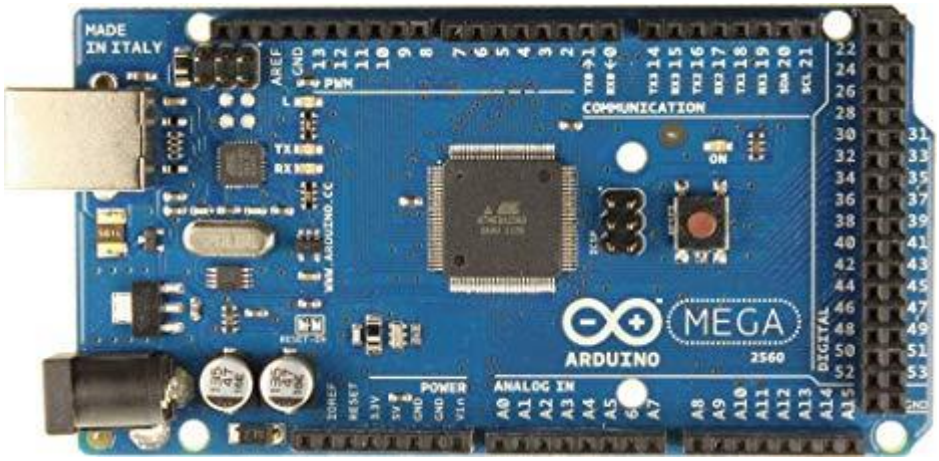

Figura 20: Arduino Mega 2560.

Fonte: Amazon – Atmega2560 Mega2560 Board

É o ATMEGA2560 o responsável pelo funcionamento do hardware de interface, ele apenas opera o teclado matricial, display gráfico e o módulo RFID, as informações captadas pelo RFID e teclado o ATMEGA envia para o ESP8266 e aguarda a resposta para exibir no display.

3.4.1 DISPLAY GRÁFICO

Foi optado pelo display gráfico 128x64, pelo custo baixo, média de R$ 60,00 (sessenta reais) e o material disponível do seu chip controlador o chip ST7920. O display está na figura 20 abaixo e a pinagem na figura 21.

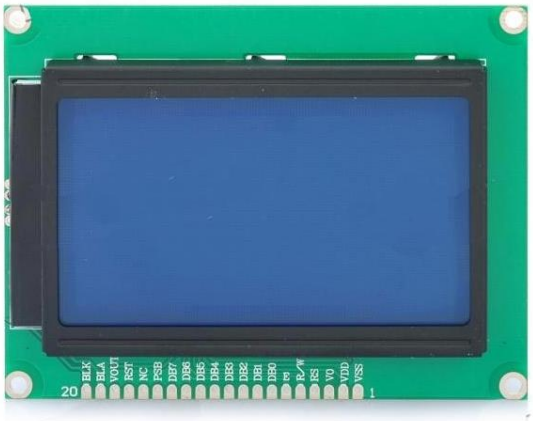

Figura 20: Display Gráfico 128x64.

Fonte: Portal Vida de Silício – Display Gráfico 128x64

Pino	Descrição
1 (Vss)	Terra da alimentação do display
2 (Vdd)	Alimentação positiva de 5 V
3 (Vo)	Ajuste do contraste do display
4 (RS)	Controle de envio de dados ou programa para o display
5 (RW)	Controle de escrita ou leitura no display
6 (E)	Envio de pulso de habilitação do display
7 – 14 (DB0 a DB7)	Barramento de dados do display
15 (CS1)	Seleção do circuito de varredura da coluna 0 a 63
16 (RST)	Reset do display
17 (Vee)	Saída de tensão para ajuste do contraste
18 (CS2)	Seleção do circuito de varredura da coluna 64 a 127
19 (K)	Cátodo do backlight
20 (A)	Ânodo do backlight

Figura 21: Display Gráfico 128x64.

Fonte: Portal Vida de Silício – Display Gráfico 128x64

3.4.2 TECLADO MATRICIAL

O teclado escolhido é o matricial de membrana 4x4. As vias são numeradas de 1 a 8 da esquerda para a direita, iniciando com as linhas e depois com as colunas. A Lógica de operação funciona com a identificação das vias ao pressionar uma tecla, por exemplo ao se pressionar a tecla “1” a via 1 (linha 1) e a via 5 (coluna 1) são interligadas possibilitando a identificação pelo ATMEGA2560.

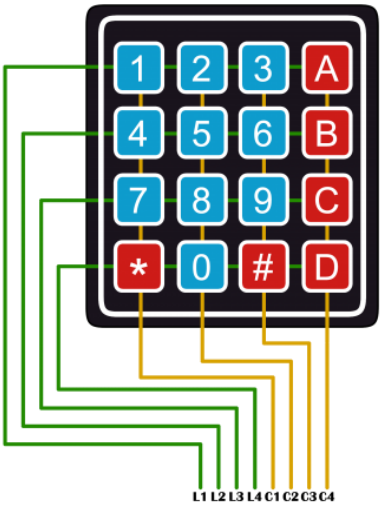

Figura 21: Teclado de Membrana 4x4.

Fonte: Portal Vida de Silício – Teclado Matricial de Membrana

3.4.3 MÓDULO RFID MFRC522

A identificação por radio frequência é fundamenta na teoria eletromagnética. Existe uma antena no módulo RFID que é capaz de detectar variações de campo magnético, então ao aproximar uma tag RFID do módulo RC522 o dispositivo é capaz de detectar a presença da tag. O formato de uma UID são 8 símbolos hexadecimais, exemplo: 83 6F 4C 79.

Figura 22: Módulo RFID MFRC522.
Fonte: Portal Vida de Silício – Teclado Matricial de Membrana.

3.6 COMUNICAÇÃO SERIAL ESP8266 E ATMEGA2560

A comunicação Serial entre o módulo mestre e o escravo segue o formato serial, onde os pinos Rx e Tx de ambos os microcontroladores são conectados. No ESP8266 é a UART0 que está ativada e sendo usada para enviar e receber os dados. No ATMEGA2560 é a UART1 que faz a comunicação com ESP. A figura 23 ilustra esta conexão:

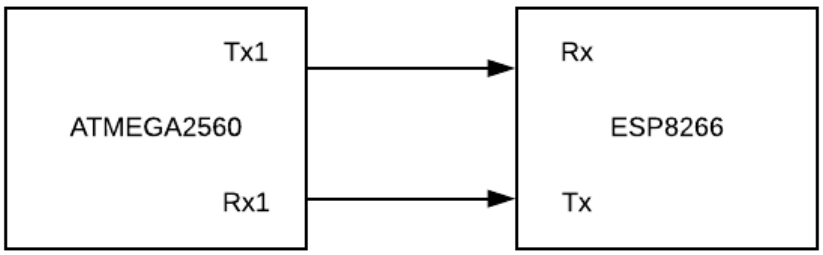

Figura 23: Conexão Serial entre os módulos mestre e escravo.
FONTE: O AUTOR, 2019

A comunicação serial até a presente data não foi totalmente implementada, ainda há problemas de sincronização entre as mensagens enviadas e recebidas. O que implica diretamente na consulta de informações pelo módulo escravo.

4 Resultados

Os resultados dos projetos podem ser descritos pelos diagramas e imagens a seguir. O primeiro diagrama explica o funcionamento da Interface Gráfica do Usuário e como ela foi implementada para obter os dados do Módulo Mestre. A GUI representa o cliente na relação servidor cliente. A construção da GUI tem inicio com a inicialização da classe Application com suas definições dos containers e suas posições geométricas, em seguida é feito uma busca de hosts que estão escritos em um arquivo de texto, é através da GUI que esses hosts são adicionados ou deletados. O próximo passo é tentar abrir um socket nos hosts cadastrados, se a conexão falhar o código continua e o loop continua. Com o socket aberto uma mensagem é enviada para o servidor, “Status.Nada”, e uma resposta no modelo “Estado da Porta,Nível Bateria, Usuário” é esperada, após a chegada da resposta o socket é fechada e o scrip alimenta um arquivo de texto chamado histórico e no fim atualiza os labels da GUI. O A classe Application é instanciada no método Main.Loop() que faz com que tudo que estiver nesta classe fique no loop de eventos.

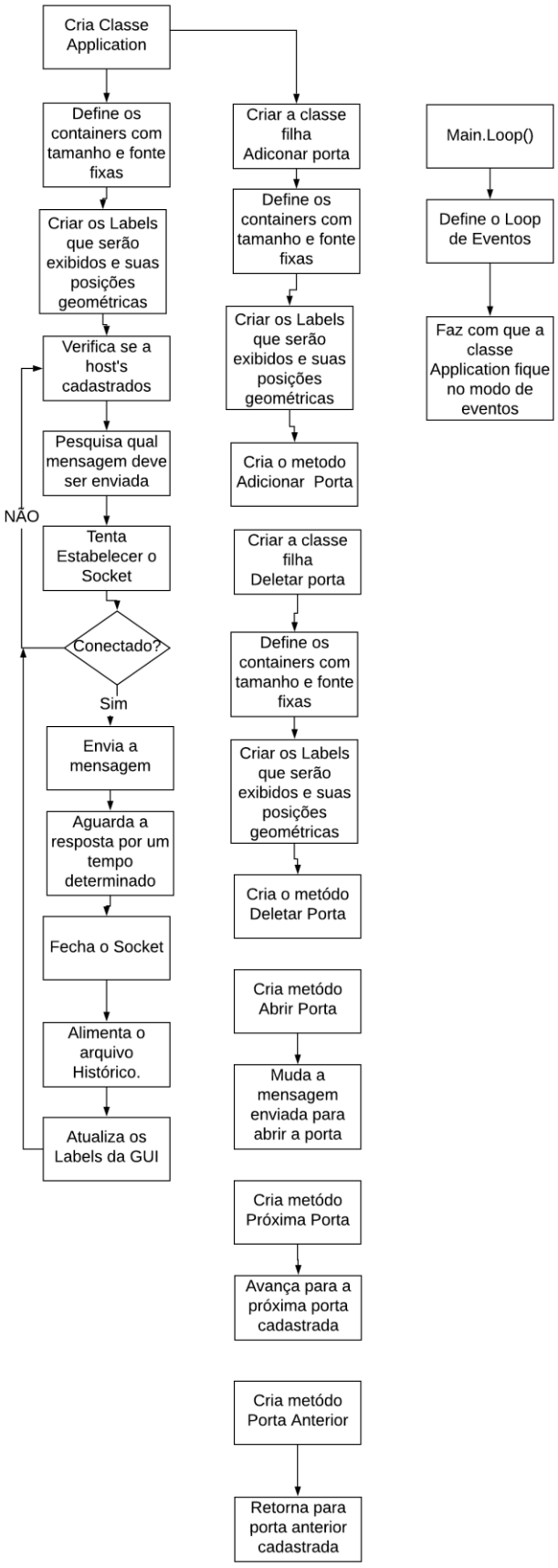

Figura 23: Diagrama GUI.

FONTE: O AUTOR, 2019

Existem duas classes filhas que podem adicionar um host ou deletar um host. Existem dois métodos que mudam a porta principal exibida na GUI, o Próxima Porta e o Porta Anterior, a porta que está logo abaixo do nome do projeto e da hora e data é a porta principal. É nela que os comandos Abrir Porta e Histórico atuam. Na figura 24 é possível observar a GUI em funcionamento com a porta ESP01 como principal e sendo exibida no painel de status. Também está no painel de status outra porta cadastrada, a ESP02, para exibir ela no menu principal é necessária clicar em próxima porta ou porta anterior.

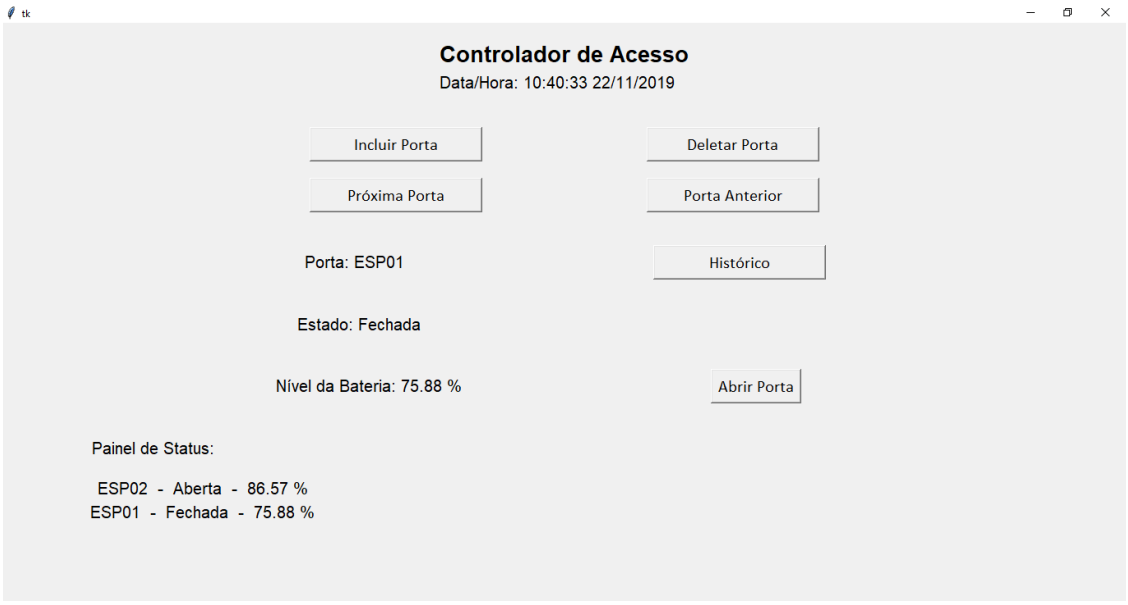

Figura 24: GUI.

FONTE: O AUTOR, 2019

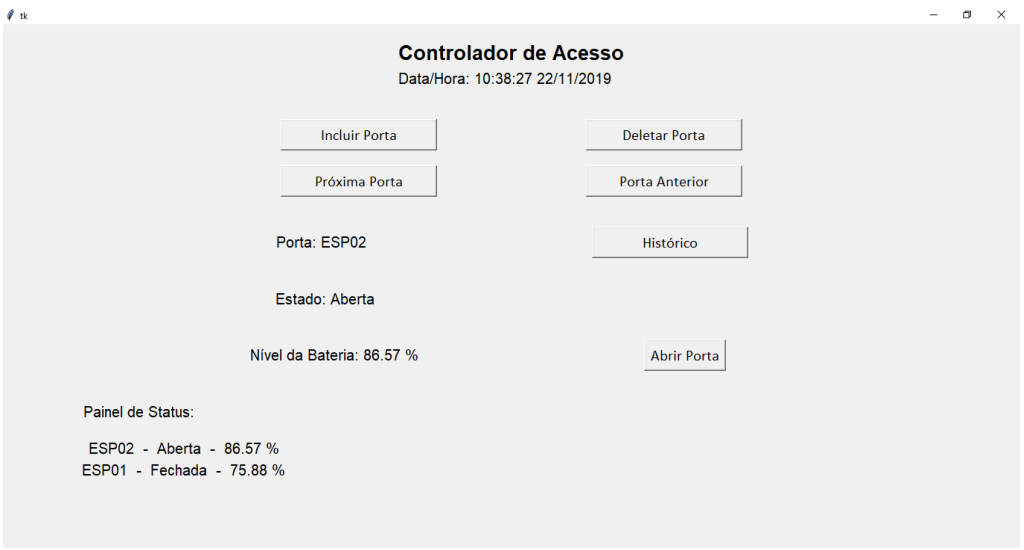

Figura 25: GUI.

FONTE: O AUTOR, 2019

As janelas que acionam os métodos Proxima Porta e Porta Anterior estão figura 26 e 27 respectivamente, é necessário entrar com o endereço IP do host e o nome que foi dado para o host. Na GUI o host é o nome da porta.

Figura 26: Cadastrar Porta.

FONTE: O AUTOR, 2019

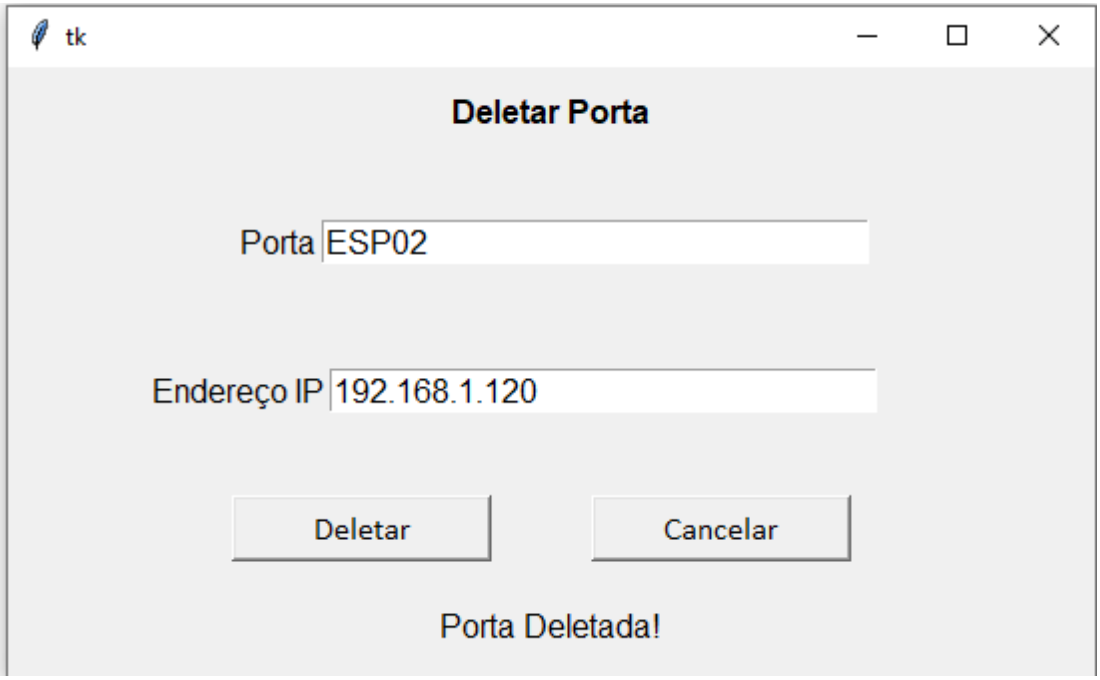

Figura 27: Deletar Porta.

FONTE: O AUTOR, 2019

A rotina do Módulo Mestre está no diagrama da figura 28 abaixo:

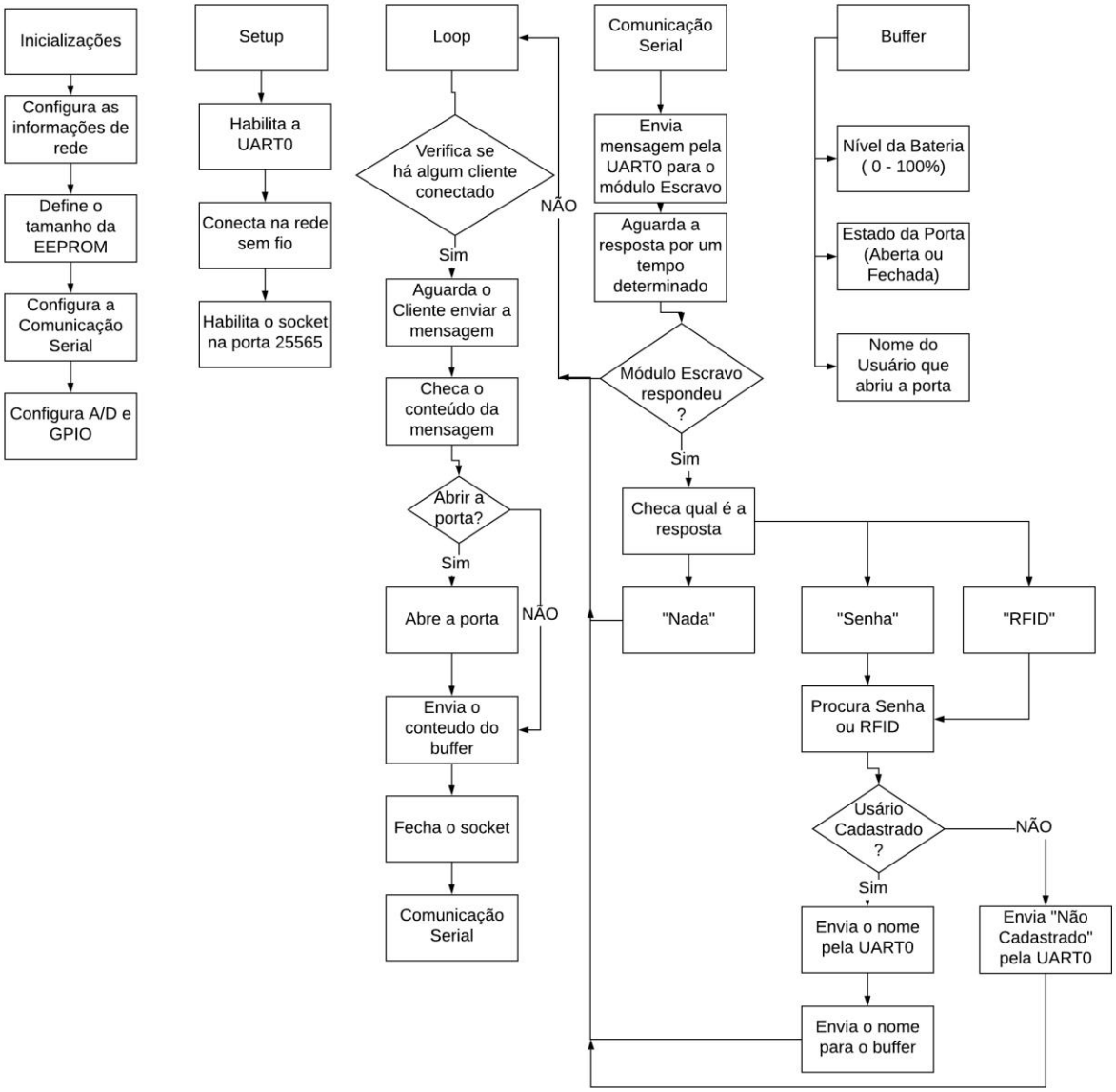

Figura 28: Rotina Módulo Mestre

FONTE: O AUTOR, 2019

A rotina aplicada no módulo escravo é descrita pelo diagrama da figura 29.

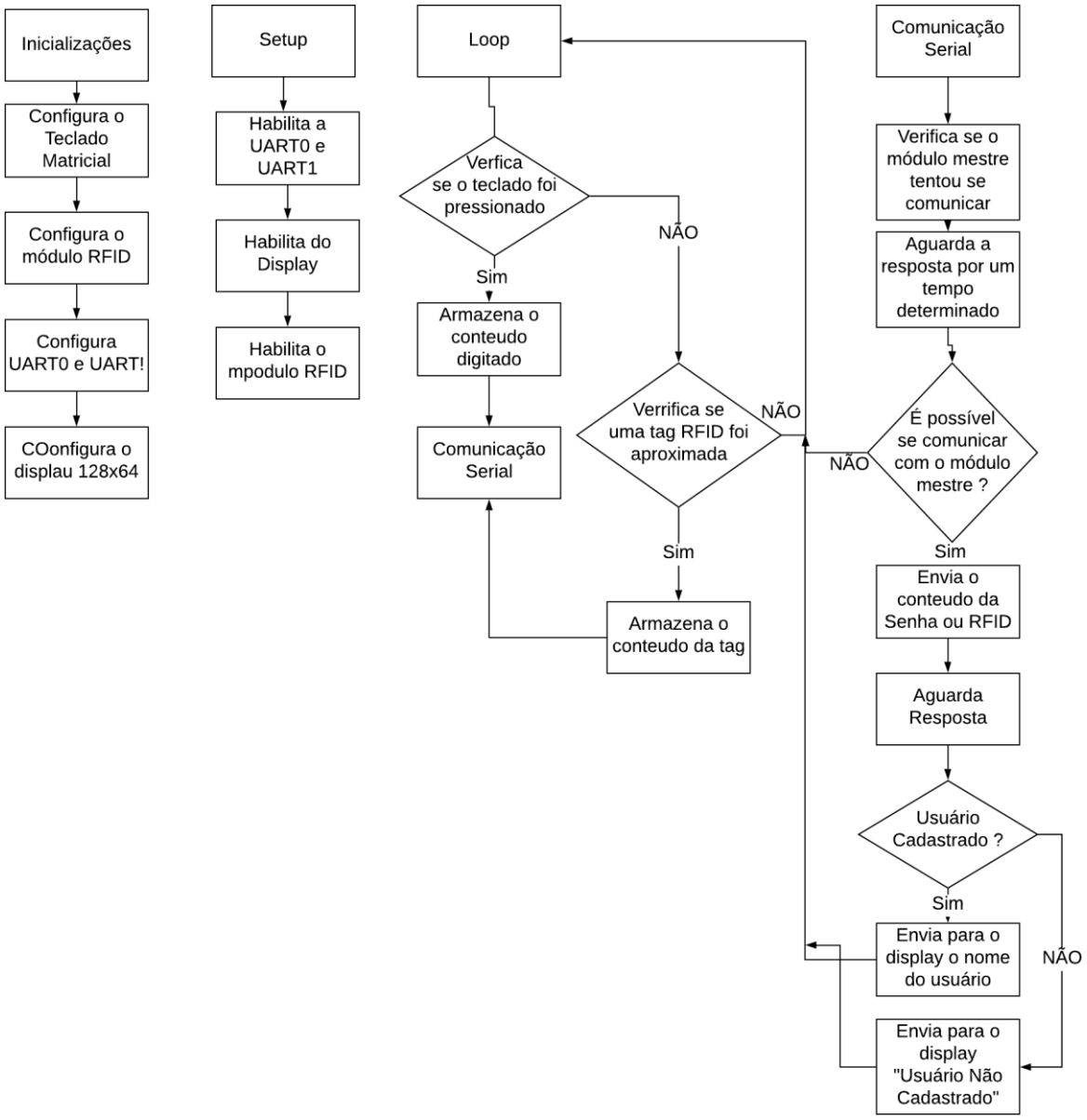

Figura 29: Rotina Módulo Escravo.

FONTE: O AUTOR, 2019

O projeto completo em funcionamento está nas próximas figuras. Na figura 30 é aproximado um cartão RFID que é identificado pelo módulo escravo e o nome é exibido no display.

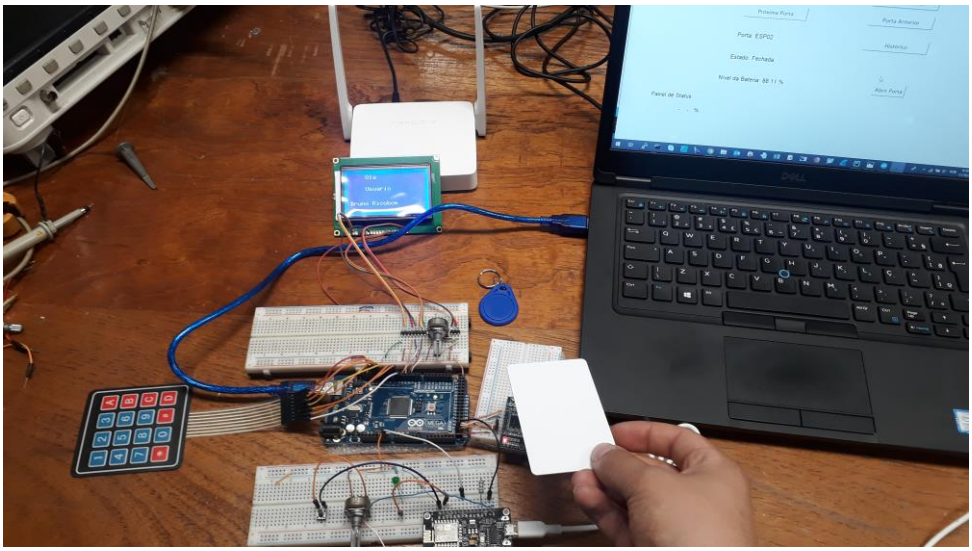

Figura 30: Aproximando uma tag

Fonte: O Autor, 2019.

Na figura 31 é possível observar o funcionamento da GUI que está recebendo os dados do módulo mestre.

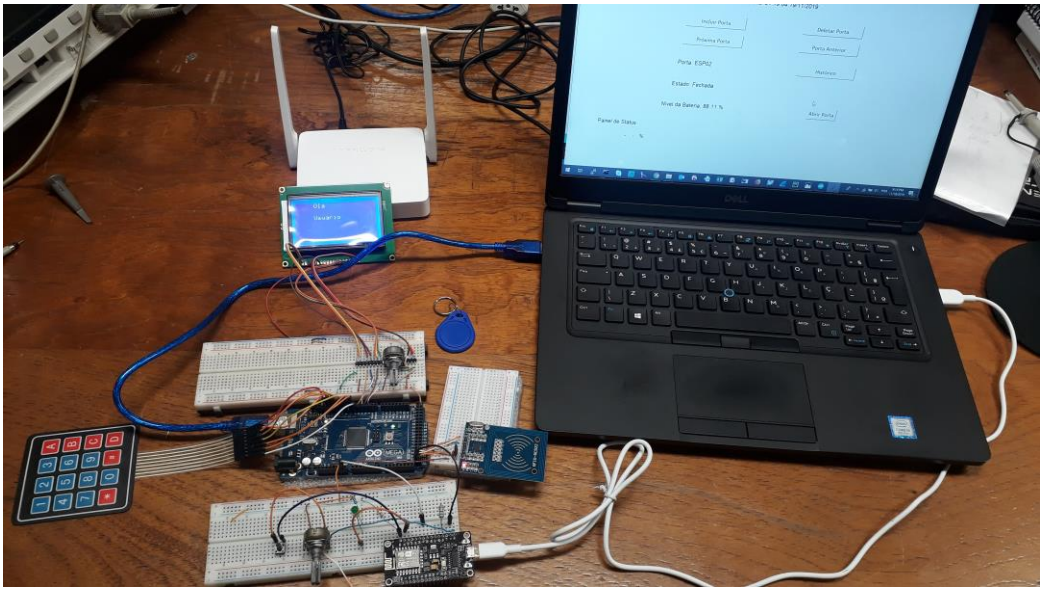

Figura 31: Visão Geral.

Fonte: O Autor, 2019.

A figura 32 é uma imagem do display ao receber o nome do usuário.

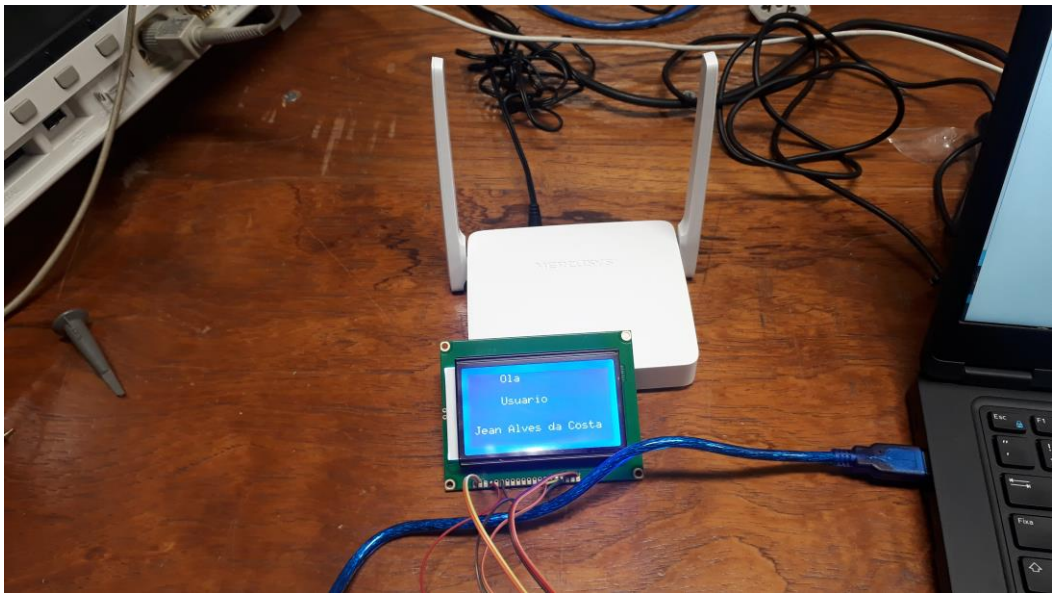

Figura 32: Displayl.

Fonte: O Autor, 2019.

Na figura 33 tem-se um foco no módulo principal. O led (1) ligado representa o atuador que deve abrir a porta, o potenciômetro (2) está sendo aplicado no conversor A/D para indicar a variação do nível da bateria do nobreak e o botão (3) para simular o estado da porta (aberta ou fechada).

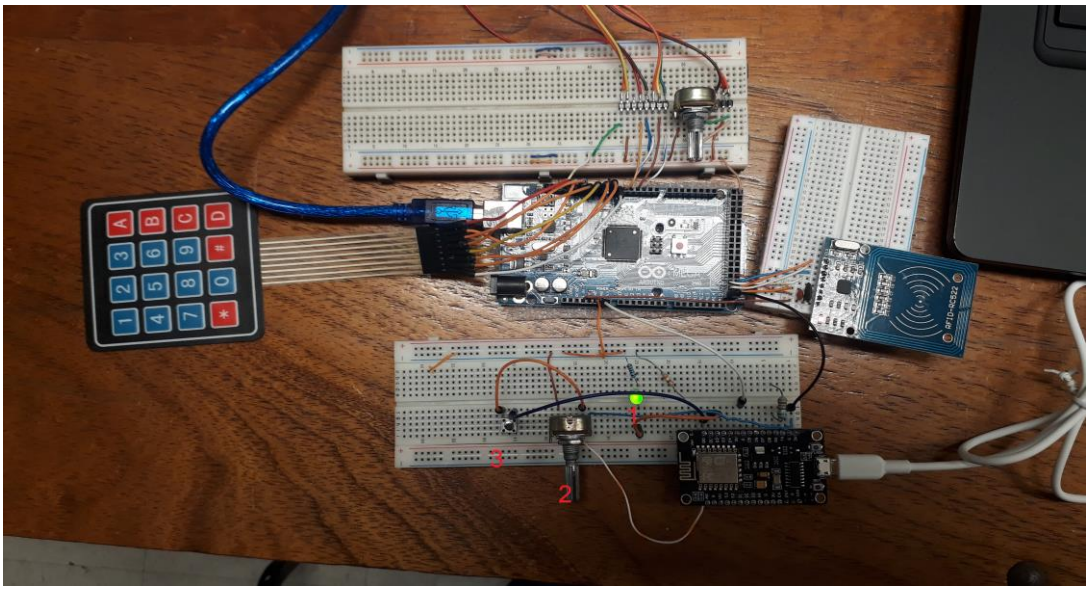

Figura 33: Módulo Mestre.

Fonte: O Autor, 2019.

5 CONCLUSÃO

A proposta do projeto foi desenvolver um controlador de acesso sem fio com gerenciamento remoto, formado por dois módulos. O projeto tem a inovação de fornecer informações a uma interface gráfica instalada em um computador conectado na mesma rede sem fio que o controlador de acesso. O propósito da divisão dos módulos é dificultar o acesso de usuários não cadastrados em um eventual furto ou roubo, assegurando que a porta do ambiente não abrirá, mesmo que o modulo externo seja completamente danificado.

A interface gráfica e a comunicação sem fio foram implementadas com sucesso, como já exposto na sessão resultados. O hardware de interface com usuário também foi implementado com êxito. A Comunicação serial ainda está em processo de desenvolvimento, devido à problemas que tem origem nas portas seriais usadas por cada microcontrolador. Espera-se que até a data da apresentação da banca todo o projeto esteja implementado.

O projeto teve uma grande contribuição na formação do autor, visto que o mesmo reuniu conceitos de diversas disciplinas, como rede de computadores, programação de microcontroladores, eletrônica analógica e digital.

6 REFERÊNCIAS

TADEU GALHARDO, Antonio. **Sistemas Eletrônicos de Controle de Acesso**. 2011. 50 f. Monografia (Engenharia Elétrica - Telecomunicações). Universidade São Francisco. São Paulo, 2011.

DOS SANTOS, Daniel L. **Controle de acesso em sistemas gerenciadores de banco de dados**. 2014. 50 f. Monografia (Especialização em Configuração e Gerenciamento de Servidores e Equipamentos de Redes). Universidade Tecnológica Federal do Paraná. Curitiba, 2014.

TONIN Fabianna S.; CITTOLIN, Guilherme F.; SOUZA, Vinicius de. **Desenvolvimento de um Sistema Web de Controle de Acesso**. 2015. 69f. Trabalho de Conclusão de Curso (Engenharia Industrial Elétrica – ênfase Automação) – Departamento Acadêmico de Eletrotécnica – Universidade Tecnológica Federal do Paraná - Curitiba, 2015.

KUROSE, J. F. e ROSS, K. - **Redes de Computadores e a Internet** - 5ª Ed., Pearson, 2010.

Anexo I – Código da GUI

```
import ipaddress
import csv
import time
from datetime import datetime
import socket
from tkinter import *   # biblioteca para importar os componentes do Tkinter


class Adicionar_Porta:

    def __init__(self, master=None):
        self.primContainer = Frame(master)
        self.fontPadrao = ("Arial", "12")
        self.primContainer["pady"] = 10
        self.primContainer.pack()

        self.seguContainer = Frame(master)
        self.seguContainer["padx"] = 20
        self.seguContainer.pack()

        self.tercContainer = Frame(master)
        self.tercContainer["padx"] = 20
        self.tercContainer.pack()

        self.quarContainer = Frame(master)
        self.quarContainer["pady"] = 20
        self.quarContainer.pack()

        self.quinContainer = Frame(master)
        self.quinContainer["padx"] = 20
        self.quinContainer.pack()

        self.titul = Label(self.primContainer, text="Cadastrar Porta")
        self.titul["font"] = ("Arial", "12", "bold")
        self.titul.pack()
        self.titul.pack(padx=(0, 0))
        self.titul.pack(pady=(0, 0))

        self.portaLabel = Label(self.seguContainer, text="Porta",
font=self.fontPadrao)
        self.portaLabel.pack(side=LEFT)
        self.portaLabel.pack(padx=(0, 0))
        self.portaLabel.pack(pady=(30, 0))

        self.porta = Entry(self.seguContainer)
        self.porta["width"] = 30
        self.porta["font"] = self.fontPadrao
        self.porta.pack(side=LEFT)
        self.porta.pack(padx=(0, 0))
        self.porta.pack(pady=(30, 0))

        self.ipLabel = Label(self.tercContainer, text="Endereço IP",
font=self.fontPadrao)
```

```
        self.ipLabel.pack(side=LEFT)
        self.ipLabel.pack(padx=(0, 0))
        self.ipLabel.pack(pady=(50, 0))

        self.ip = Entry(self.tercContainer)
        self.ip["width"] = 30
        self.ip["font"] = self.fontPadrao
        self.ip.pack(side=LEFT)
        self.ip.pack(padx=(0, 40))
        self.ip.pack(pady=(50, 0))

        self.cancelar = Button(self.quarContainer)
        self.cancelar["text"] = "Cancelar"
        self.cancelar["font"] = ("Calibri", "12")
        self.cancelar["width"] = 15
        self.cancelar["command"] = master.destroy
        self.cancelar.pack()
        self.cancelar.pack(side=RIGHT)
        self.cancelar.pack(padx=(0, 50))
        self.cancelar.pack(pady=(20, 0))

        self.cadastrar = Button(self.quarContainer)
        self.cadastrar["text"] = "Cadastrar"
        self.cadastrar["font"] = ("Calibri", "12")
        self.cadastrar["width"] = 15
        self.cadastrar["command"] = self.cadastrarporta
        self.cadastrar.pack(side=LEFT)
        self.cadastrar.pack()
        self.cadastrar.pack(padx=(40, 50))
        self.cadastrar.pack(pady=(20, 0))


        self.mens = Label(self.quinContainer, text="", font=self.fontPadrao)
        self.mens.pack()
        self.mens.pack(padx=(0, 0))
        self.mens.pack(pady=(0, 0))


    # Método adiciona porta
    def cadastrarporta(self):
            ip = self.ip.get()
            host = self.porta.get()
            arquivo = open('host.txt', 'r')            # Abra o arquivo host
            conteudo = arquivo.readlines()

            # insira seu conteúdo
            # obs: o método append() é proveniente de uma lista

            conteudo.append(ip+","+host+"\n")


            # Abre novamente o arquivo (escrita)
            # e escreva o conteúdo criado anteriormente nele.
            arquivo = open('host.txt', 'w')
            arquivo.writelines(conteudo)
            arquivo.close()
            self.mens["text"] = "Porta Cadastrada!"

    def quit(self):
```

```
        exit()

class Deletar_Porta:

    def __init__(self, master=None):
        self.priContainer = Frame(master)
        self.fonPadrao = ("Arial", "12")
        self.priContainer["pady"] = 10
        self.priContainer.pack()

        self.segContainer = Frame(master)
        self.segContainer["padx"] = 20
        self.segContainer.pack()

        self.terContainer = Frame(master)
        self.terContainer["padx"] = 20
        self.terContainer.pack()

        self.quaContainer = Frame(master)
        self.quaContainer["pady"] = 20
        self.quaContainer.pack()

        self.quiContainer = Frame(master)
        self.quiContainer["padx"] = 20
        self.quiContainer.pack()

        self.titu = Label(self.priContainer, text="Deletar Porta")
        self.titu["font"] = ("Arial", "12", "bold")
        self.titu.pack()
        self.titu.pack(padx=(0, 0))
        self.titu.pack(pady=(0, 0))

        self.port2Label = Label(self.segContainer, text="Porta",
font=self.fonPadrao)
        self.port2Label.pack(side=LEFT)
        self.port2Label.pack(padx=(0, 0))
        self.port2Label.pack(pady=(30, 0))

        self.port2 = Entry(self.segContainer)
        self.port2["width"] = 30
        self.port2["font"] = self.fonPadrao
        self.port2.pack(side=LEFT)
        self.port2.pack(padx=(0, 0))
        self.port2.pack(pady=(30, 0))

        self.ip2Label = Label(self.terContainer, text="Endereço IP",
font=self.fonPadrao)
        self.ip2Label.pack(side=LEFT)
        self.ip2Label.pack(padx=(0, 0))
        self.ip2Label.pack(pady=(50, 0))

        self.ip2 = Entry(self.terContainer)
        self.ip2["width"] = 30
        self.ip2["font"] = self.fonPadrao
        self.ip2.pack(side=LEFT)
        self.ip2.pack(padx=(0, 40))
        self.ip2.pack(pady=(50, 0))

        self.cancela = Button(self.quaContainer)
```

```
        self.cancela["text"] = "Cancelar"
        self.cancela["font"] = ("Calibri", "12")
        self.cancela["width"] = 15
        self.cancela["command"] = master.destroy
        self.cancela.pack()
        self.cancela.pack(side=RIGHT)
        self.cancela.pack(padx=(0, 50))
        self.cancela.pack(pady=(20, 0))

        self.deletar = Button(self.quaContainer)
        self.deletar["text"] = "Deletar"
        self.deletar["font"] = ("Calibri", "12")
        self.deletar["width"] = 15
        self.deletar["command"] = self.deletarporta
        self.deletar.pack(side=LEFT)
        self.deletar.pack()
        self.deletar.pack(padx=(40, 50))
        self.deletar.pack(pady=(20, 0))


        self.men = Label(self.quiContainer, text="", font=self.fonPadrao)
        self.men.pack()
        self.men.pack(padx=(0, 0))
        self.men.pack(pady=(0, 0))


    # Método adiciona porta
    def deletarporta(self):
            ip2 = self.ip2.get()
            host2 = self.port2.get()
            arquivo2 = open('host.txt', 'r')  # Abra o arquivo host
            for linha in arquivo2:
                conteudo2 = linha.split(",")  # método para quebrar cada linha em
uma lista com o argumento ()
                conteudo2.remove(host2)  # apagar conteudo pelo valor
                conteudo2.remove(ip2)  # apagar conteudo pelo valor
                arquivo2 = open('host.txt', 'w')  # Abre novamente o arquivo
(escrita)
                arquivo2.writelines(conteudo2)    #reescreve a nova lista no
arquivo, exceto o que foi deletado
            arquivo2.close()        #fecha o arquivo
            self.men["text"] = "Porta Deletada!"

    def quit(self):
        exit()

class Application:

    def __init__(self, master=None,):
        self.fontePadrao = ("Arial", "15")
        self.primeiroContainer = Frame(master)
        self.primeiroContainer["pady"] = 10
        self.primeiroContainer.pack()
        self.primeiroContainer.pack(pady=(10, 30))

        self.segundoContainer = Frame(master)
        self.segundoContainer["padx"] = 20
        self.segundoContainer.pack()
```

```
self.terceiroContainer = Frame(master)
self.terceiroContainer["padx"] = 20
self.terceiroContainer.pack()
self.terceiroContainer.pack(pady=20)

self.quartoContainer = Frame(master)
self.quartoContainer["padx"] = 20
self.quartoContainer.pack()
self.quartoContainer.pack(pady=20)


self.quintoContainer = Frame(master)
#self.quintoContainer["padx"] = 20
self.quintoContainer.pack()
self.quintoContainer.pack(pady=20)

self.sextoContainer = Frame(master)
self.sextoContainer["padx"] = 0
self.sextoContainer.pack()
self.sextoContainer.pack(pady=20)

self.setimoContainer = Frame(master)
self.setimoContainer["padx"] = 20
self.setimoContainer.pack()
self.setimoContainer.pack(pady=20)

self.oitavoContainer = Frame(master)
self.oitavoContainer["padx"] = 20
self.oitavoContainer.pack()
#self.oitavoContainer.pack(pady=20)

self.nonoContainer = Frame(master)
self.nonoContainer["padx"] = 20
self.nonoContainer.pack()
#self.nonoContainer.pack(pady=20)

self.decimoContainer = Frame(master)
self.decimoContainer["padx"] = 20
self.decimoContainer.pack()
#self.decimoContainer.pack(pady=20)

########### Strings Variaveis ################

self.data_hora_texto = StringVar()
self.IP_texto = StringVar()
self.estado_porta_texto = StringVar()
self.estado_porta_status = StringVar()
self.nivel_bateria_texto = StringVar()
self.nivel_bateria_status = StringVar()
self.nome_porta_texto = StringVar()
self.nome_porta_status = StringVar()
self.nome_usuario_texto = StringVar()
self.mensagem = StringVar()


########### Título do Projeto ################

self.titulo = Label(self.primeiroContainer, text="Controlador de Acesso")
self.titulo["font"] = ("Arial", "20", "bold")
```

```
        self.titulo.pack()

        ########### Data e Hora ###############

        self.data_hora_Label = Label(self.primeiroContainer, text="Data/Hora:",
font=self.fontePadrao)    #Nome do label
        self.data_hora_Label.pack(side=LEFT)           #posicao do label
        self.data_hora_Label.pack(padx=(0, 0))

        self.datahoralabel = Label(self.primeiroContainer, textvariable=
self.data_hora_texto, font=self.fontePadrao)       #Nome do label
        self.datahoralabel.pack(side=LEFT) #posicao do label
        self.datahoralabel.pack(padx=(0, 0))


        ########### Incluir & deletar Porta ###############
        self.incluir = Button(self.segundoContainer)
        self.incluir["text"] = "Incluir Porta"
        self.incluir["font"] = ("Calibri", "15")
        self.incluir["width"] = 20
        self.incluir["command"] = self.incluir_porta
        self.incluir.pack(side=LEFT)
        self.incluir.pack(padx = (0,100))

        self.deletar = Button(self.segundoContainer)
        self.deletar["text"] = "Deletar Porta"
        self.deletar["font"] = ("Calibri", "15")
        self.deletar["width"] = 20
        self.deletar["command"] = self.deletarporta
        self.deletar.pack(side=RIGHT)
        self.deletar.pack(padx=(100,0))

        ########### Avançar/Voltar Host ###############

        self.avancaLabel = Button(self.terceiroContainer)   #Nome do label
        self.avancaLabel["text"] = "Próxima Porta"
        self.avancaLabel["font"] = ("Calibri", "15")
        self.avancaLabel["width"] = 20
        self.avancaLabel["command"] = self.proxima_porta
        self.avancaLabel.pack(side=LEFT)
        self.avancaLabel.pack(padx=(0,100))

        self.retornaLabel = Button(self.terceiroContainer)   #Nome do label
        self.retornaLabel["text"] = "Porta Anterior"
        self.retornaLabel["font"] = ("Calibri", "15")
        self.retornaLabel["width"] = 20
        self.retornaLabel["command"] = self.anterior_porta
        self.retornaLabel.pack(side=RIGHT)
        self.retornaLabel.pack(padx=(100,0))

        ########### Nome da Porta e Botão Histórico ###############


        self.portaLabel = Label(self.quartoContainer, text="Porta:",
font=self.fontePadrao)    #Nome do label
        self.portaLabel.pack(side=LEFT)
        self.portaLabel.pack(padx=(0,0))            ###

        self.mensagem_nome_porta = Label(self.quartoContainer,
```

```
textvariable=self.nome_porta_texto, font=self.fontePadrao)
        self.mensagem_nome_porta.pack(side=LEFT)
        self.mensagem_nome_porta.pack(padx=(0,0))

        self.historico = Button(self.quartoContainer)   #Nome do label
        self.historico["text"] = "Histórico"
        self.historico["font"] = ("Calibri", "15")
        self.historico["width"] = 20
        #self.historico["command"] = self.deletarporta
        self.historico.pack(side=RIGHT)
        self.historico.pack(padx=(300,0))

        ############ Estado da Porta e Botão Banco de Dados ################

        self.estadoLabel = Label(self.quintoContainer, text="Estado:",
font=self.fontePadrao)    #Nome do label
        self.estadoLabel.pack(side=LEFT)           #posicao do label
        self.estadoLabel.pack(padx=(0,0))

        self.mensagem_estado_porta = Label(self.quintoContainer,
textvariable=self.estado_porta_texto, font=self.fontePadrao)
        self.mensagem_estado_porta.pack(side=RIGHT)
        self.mensagem_estado_porta.pack(padx=(0, 500))

        """self.bancodedados = Button(self.quintoContainer)   #Nome do label
        self.bancodedados["text"] = "Banco de Dados"
        self.bancodedados["font"] = ("Calibri", "15")
        self.bancodedados["width"] = 20
        #self.deletar["command"] = self.deletarporta
        self.bancodedados.pack(side=RIGHT)
        self.bancodedados.pack(padx=(250,0))"""

        ############ Nível Bateria e Abrir Porta ################

        self.nivelbateriaLabel = Label(self.sextoContainer, text="Nível da
Bateria:", font=self.fontePadrao)    #Nome do label
        self.nivelbateriaLabel.pack(side=LEFT)           #posicao do label
        self.nivelbateriaLabel.pack(padx=(0,0))

        self.mensagem_nivel_bateria = Label(self.sextoContainer,
textvariable=self.nivel_bateria_texto, font=self.fontePadrao)
        self.mensagem_nivel_bateria.pack(side=LEFT)

        self.mensagem_nivel_bateria_porcentagem = Label(self.sextoContainer,
text="%", font=self.fontePadrao)
        self.mensagem_nivel_bateria_porcentagem.pack(side=LEFT)

        self.abrir = Button(self.sextoContainer)
        self.abrir["text"] = "Abrir Porta"
        self.abrir["font"] = ("Calibri", "15")
        self.abrir["width"] = 10
        self.abrir["command"] = self.abrir_porta
        self.abrir.pack(side=LEFT)
        self.abrir.pack(padx=(300,60))

        self.mensagem = Label(self.sextoContainer, text="", font=self.fontePadrao)
        self.mensagem.pack(side=RIGHT)

        ############ Loop estado da porta e nível da Bateria de todas as portas
```

```
################

        self.logLabel = Label(self.setimoContainer, text="Painel de Status:",
font=self.fontePadrao)    #Nome do label
        self.logLabel.pack(side=LEFT)           #posicao do label
        self.logLabel.pack(padx=(0, 1000))

        self.mensagem_nome_porta = Label(self.oitavoContainer,
textvariable=self.nome_porta_status, font=self.fontePadrao)
        self.mensagem_nome_porta.pack(side=LEFT)
        self.mensagem_nome_porta.pack(padx=(0,0))
        self.mensagem_nome_porta_traco = Label(self.oitavoContainer, text=" - ",
font=self.fontePadrao)
        self.mensagem_nome_porta_traco.pack(side=LEFT)

        self.mensagem_estado_porta = Label(self.oitavoContainer,
textvariable=self.estado_porta_status, font=self.fontePadrao)
        self.mensagem_estado_porta.pack(side=LEFT)
        self.mensagem_estado_porta.pack(padx=(0, 0))
        self.mensagem_estado_porta_traco = Label(self.oitavoContainer, text=" - ",
font=self.fontePadrao)
        self.mensagem_estado_porta_traco.pack(side=LEFT)

        self.mensagem_nivel_bateria = Label(self.oitavoContainer,
textvariable=self.nivel_bateria_status, font=self.fontePadrao)
        self.mensagem_nivel_bateria.pack(side=LEFT)
        self.mensagem_nivel_bateria_porcentagem = Label(self.oitavoContainer,
text="%", font=self.fontePadrao)
        self.mensagem_nivel_bateria_porcentagem.pack(side=LEFT)
        self.mensagem_nivel_bateria_porcentagem.pack(padx=(0, 880))



    ############ Chamada de funções################

        self.contador = 0
        self.contador_porta = 0

        i = 0
        self.lista_send = []
        arquivo = open('host.txt', 'r')  # Abra o arquivo host
        self.lista = arquivo.readlines()  # transforma to do o conteudo do arquivo
em uma lista
        print("Lista:", self.lista)
        # self.valores = self.lista[0].split(",")  # método para quebrar cada
linha em uma lista com o argumento ()
        # print(self.valores)

        # Edit Stonoga + Jean
        self.lista_resposta = []
        self.valores = []

        self.tamanho_lista = len(self.lista)  # desconta /0
        print("Tamanho da lista:", self.tamanho_lista)
        while i < self.tamanho_lista:

            par = self.lista[i]
            _ip, _nome = par.strip().split(',')
```

```
            print("IP:",_ip, "Host", _nome)
            self.valores.append([_ip, _nome])

            self.lista_send.append("Status,Nada")  # para to do o tamanho da lista
de
            self.lista_resposta.append([0,0,0])     #inicializa a lista resposta
            i +=1
        print("Lista de status:", self.lista_send)
        print("Valores:", self.valores)

        arquivo.close()
        self.loop()


    ########### Metódos ###############


    def loop(self):
        self.ler_host()
        self.escrever_historico()

        data_atualizada = datetime.now().strftime('%H:%M:%S %d/%m/%Y')

        if data_atualizada != self.data_hora_texto:
            self.data_hora_texto.set(data_atualizada)

        self.datahoralabel.after(500, self.loop)       #tempo ms precisa ser igual
ao timeout do tcp para não perder sincronismo


    def ler_host(self):
                print(self.contador)

                # ipdaporta = self.valores[self.contador]
                # nomedaporta = self.valores[self.contador + 1]
                ipdaporta, nomedaporta = self.valores[self.contador]

                self.contador = self.contador+1
                if (self.contador >= self.tamanho_lista):
                    self.contador = 0
                self.IP_texto.set(ipdaporta)
                print(self.IP_texto.get())
                print(self.nome_porta_texto.get())
                self.estabelece_socket()


    def estabelece_socket(self):
                if (self.IP_texto != ''):  # verifica se a primeira posição da
lista não está em branco
                    PORT = 25565  # Porta que o Servidor esta
                    BUFFER_SIZE = 1024  # Define o tamanho do Buffer
                    tcp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  #
cria o objeto socket com a familia TCP
                    HOST = self.IP_texto.get()          #Define HOST como str
                    #hostname = tcp.gethostbyname_ex(HOST)
                    dest = (HOST, PORT)  # define o servidor
                    self.msg_status = self.lista_send[self.contador]
                    if self.lista_send[self.contador] == ('Status,Abrir'):
                        self.lista_send [self.contador] = ('Status,Nada')
```

```
                    msgb = str.encode(self.msg_status)  # String -> Bytes
                    print(msgb)
                    tcp.settimeout(0.5)                         #tempo precisa ser
igual do update, para não perder sincronismo,
                    conecta = tcp.connect_ex(dest)              #Abre o socket e
retorna 0 para estabelecido e erro variable para não estabelecido,
                    if (conecta == 0):
                        tcp.send(msgb)  # enviados dados para o servidor
                        #print(msgb)
                        data = tcp.recv(BUFFER_SIZE)  # armazena os dados
recebidos do servidor
                        self.resposta = bytes.decode(data)  # bytes -> string
                        print(self.resposta)
                        tcp.shutdown(socket.SHUT_RDWR)                  #desliga a
conexão, enviar e receber não é permitido
                        tcp.close()
                        self.analisa_resposta()                         #metodo
para a analisar a mensagem recebida
                    else: tcp.close()
                else: return

    def analisa_resposta(self):
            estadodaporta = self.resposta.split(",")[0]
            nivelbateria = self.resposta.split(",")[1]
            nomeusuario = self.resposta.split(",")[2]

            self.lista_resposta[self.contador_porta] = [estadodaporta,
nivelbateria, nomeusuario]
            if self.contador_porta == self.contador:
                self.atualiza_labels_GUI()

            #self.estado_porta_texto.set(estadodaporta)
            #self.nivel_bateria_texto.set(nivelbateria)
            #self.nome_usuario_texto.set(nomeusuario)
            #self.nomehistorico = str(nomeusuario)


    def escrever_historico(self):
        nome_usuario_log = self.nome_usuario_texto.get()
        estado_porta_log = self.estado_porta_texto.get()
        data_hora_log = datetime.now().strftime('%H:%M:%S %d/%m/%Y')
        arquivo3 = open('historico.txt', 'r')  # Abra o arquivo host
        conteudo3 = arquivo3.readlines()

        # insira seu conteúdo
        # obs: o método append() é proveniente de uma lista

        conteudo3.append(data_hora_log+","+nome_usuario_log+ ","
+estado_porta_log+ ","+"\n")

        # Abre novamente o arquivo (escrita)
        # e escreva o conteúdo criado anteriormente nele.
        arquivo3 = open('historico.txt', 'w')
        arquivo3.writelines(conteudo3)
        arquivo3.close()

    def abrir_porta(self):
        self.lista_send[self.contador] = ('Status,Abrir')
```

```
    def proxima_porta(self):
        self.contador_porta += 1
        if self.contador_porta >= self.tamanho_lista:
            self.contador_porta = 0
        self.atualiza_labels_GUI()

    def anterior_porta(self):
        self.contador_porta -= 1
        if self.contador_porta < 0:
            self.contador_porta = self.tamanho_lista-1
        self.atualiza_labels_GUI()

    def deletarporta(self):
        root3 = Toplevel()
        Deletar_Porta(root3)

    def incluir_porta(self):
        root2= Toplevel()
        Adicionar_Porta(root2)

    def atualiza_labels_GUI(self):
        estadodaporta, nivelbateria, _ = self.lista_resposta[self.contador_porta]
        _, nomedaporta = self.valores[self.contador_porta]

        self.estado_porta_texto.set(estadodaporta)
        self.nivel_bateria_texto.set(nivelbateria)
        #self.nome_usuario_texto.set(nomeusuario)
        self.nome_porta_texto.set(nomedaporta)

    #def atualiza_labels_status(self):



root = Tk()                     #cria um widget root Tk
root.geometry('1366x768')       #Define o tamanho fixo da tela
Application(root)               #Inicializa a interface com a classe application,
instância da classe App usando o widget root como pai:
root.mainloop()                 #Faz com que a App fique no modo de

#ctrl_porta = 0
```

Anexo II – Código do ESP8266

```
#include "ESP8266WiFi.h"
#include <Bounce2.h>
#include <stdio.h>
#include <stdlib.h>
#include <EEPROM.h>

//////////////////////Variáveis da Rede Wifi//////////////////////

const char* ssid = "Rede_ESP";
const char* password =  "genedu139mr";

WiFiServer wifiServer(25565);

String readString;
String readString2;
String estado_porta;
String writeString;
char buffer[100];
String leitura_cliente ;
int abrir_porta = 0;
int contador_timeot_ext = 0;

char* condporta = "";
char* nivelbateria="80%";
char *nomeusuario = "Nenhum usuario";
char *erro = "Erro";
char data[4];
char nome_user[37];
int aux;

const unsigned int analogInPin = A0;    // Analog input - pino analógico A0
unsigned int RAWanalogInput = 0;          // save the RAW value of A0 - armazena o valor RAW de A0
float analogInputVoltage = 0;             // save the converted A0 voltage - armazena a tensão convertida de A0
const float analogLimit = 3.3;          // Limit of Analog In. - Limite da entrada analogica
bool estado=1;
bool sensor_porta;
unsigned long start;
unsigned long end;
unsigned long end2;
```

```
Bounce debouncer = Bounce();                    //Tratar o botão que indica o estado da
Porta


//////////////////////////Variáveis da EEPROM//////////////////////////////

int tamanho_senha = 6;
int tamanho_RFID = 8;
int tamanho_nome = 36;

int inicio_senha = 45;
int incio_config_rede = 0;
int inicio_RFID = 525;
int inicio_nome_usuario = 1165;


String senha_recebida = "";     //senha do usuário via UART
String rfid_recebida = "";      //RFID do usuário via UART
String nome_usuario = "";       //nome do usuário via UART

//char* senha_recebida = "";     //senha do usuário via UART
//char* rfid_recebida = "";      //RFID do usuário via UART
//char* nome_usuario = "";       //nome do usuário via UART

int endereco;           //variável para varrer cada endereço
int digito;         //variável para varrer cada caracter da mensagem

int msg=0;

char* senha_rede = "";        //senha da rede via UART
char* nome_rede = "";         //nome da rede via UART

char* ip = "";              //IP fornecido pelo usuário via UART
char* gw = "";              //gateway fornecido pelo usuário via UART
char* msk = "";             //mascara fornecida pelo usuário via UART

String mensagem = "";             //mensagem de comunicação com o usuário

//////////////////Variáveis UART0 do ESP/////////////////

//const byte numChars = 100;
//char receivedChars[numChars];
String receivedChars;
```

```
/////////////// Funções Nivel, Bateria, Estado da Porta e Abrir Porta
////////////////////////////////////

void Nivel_Bateria(){
   RAWanalogInput = analogRead(analogInPin);;
   analogInputVoltage = (float)RAWanalogInput * (analogLimit/1024);
//analogLimit/1024 -> 10 apenas para visualizar na GUI
   dtostrf(analogInputVoltage,4, 2, nivelbateria);
   return;
 }


void Verifica_Estado()
{
    debouncer.update(); // Executa o algorítimo de tratamento;
    int value = debouncer.read(); // Lê o valor tratado do botão;
    if (value == LOW) {
       condporta = "Fechada";
        }
    else {
       estado = !estado;
       condporta = "Aberta";
       }
  return;
}

void banco_de_dados()
{
 //retorna os usuários salvos na memória do esp
}

void AbrirPorta(){
  bool sensor_porta = digitalRead(D1); // Lê o estado atual;

  if (sensor_porta == LOW) {
     digitalWrite(D1,estado);
          estado = !estado;
     //delay(500);
  }
```

```
  return;
}


////////////////////// Rotina para receber e enviar dados para GUI/////////////////////////////////////

void proc_stack()
{
  WiFiClient client = wifiServer.available();    //Obtém um cliente que está conectado ao servidor e tem dados disponíveis para leitura

       //Serial.println("Aguardando cliente...");

       if (client ) {                          //Se houver um cliente

             while (client.connected()) {                    //Enquanto houver um cliente conectado
              //Serial.println("Cliente Conectado");


                while (client.available()>0) {       //Enquanto houver dados disponíveis no servidor
                  client.setTimeout(20);
                  leitura_cliente = client.readString();



                 Serial.println(leitura_cliente);

                  Verifica_Estado();
                  Nivel_Bateria();
                  if(leitura_cliente == "Status,Nada")
                  {
                    //proc_resposta();
                       sprintf(buffer,"%s,%s,%s",condporta,nivelbateria,nomeusuario);
                       client.write(buffer);
                       Serial.println(buffer);

                  }


                  else if(leitura_cliente == "Status,Abrir")
                  {
                        AbrirPorta();
```

```
                    Verifica_Estado();
                    sprintf(buffer,"%s,%s,%s,abr",condporta,nivelbateria,nomeusuario);
                    client.write(buffer);
                    Serial.println(buffer);
                  }

                  leitura_cliente ="";
                  }
                }
                    client.stop();
                    Serial.println("Cliente Desconectado");
              }


}

////////////////////Funções para Trabalhar com EEPROM/////////////////////////////////////////

void deleta_usuario(){

}

void procura_nome_usuario(){

  for(digito = 0; digito<36;digito++)
  {
     nome_user[digito] == EEPROM.read(inicio_nome_usuario+digito+36*endereco);
  }
  nome_user[digito] = '\0';
}

int procura_rfid(){
  EEPROM.begin(4096);//Inicia a EEPROM com tamanho de 4096 Bytes.
  int aux=0;
  endereco=0;
  digito=0;

  while(endereco<80){
    while(digito < 11){
      if(rfid_recebida[digito] == EEPROM.read(inicio_RFID+digito+11*endereco))
      {
        aux =1;
      }
      else {
```

```
        aux = 0;
        break;
      }
      digito++;
    }
    endereco++;
    if(aux == 1){
      mensagem = "Usuário Cadastrado";
      msg = 1;
    }
    else {
      mensagem = "Usuário Não Cadastrado";
      msg = 0;
    }
  }
  EEPROM.end();//Fecha a EEPROM.
  return msg;

}


int procura_senha(){
  EEPROM.begin(4096);//Inicia a EEPROM com tamanho de 4096 Bytes.
  int aux=0;
  endereco=0;
  digito=0;
  while(endereco<80){
    while(digito < 6){
      if(senha_recebida[digito] == EEPROM.read(inicio_senha+digito+6*endereco))
      {
        aux =1;
      }
      else {
        aux = 0;
        break;
      }
      digito++;
    }
    endereco++;
    if(aux == 1){
      mensagem = "Usuário Cadastrado";
      msg=1;
    }
    else {
```

```
      mensagem = "Usuário Não Cadastrado";
      msg = 0;

    }
    //yield();                    // faz o feed do SW WDT, para não resetar o esp
  }
  EEPROM.end();//Fecha a EEPROM.
  Serial.println(mensagem);
  return msg;
}


void adiciona_senha_RFID_usuario(){
    EEPROM.begin(4096);//Inicia a EEPROM com 4096 bytes
    int sem_endereco=0;
    int count=0;
    endereco = 0;
    while(EEPROM.read(inicio_senha+6*endereco)!=255){          //descobrindo onde
começa um endereço disponível, 255 é o default de endereço de memória
      endereco++;
      sem_endereco=0;
      if(endereco>79){
        sem_endereco=1;
        break;
      }
    }
    /*Serial.println("sem_endereco:");
    Serial.println(sem_endereco);
    if(sem_endereco==0)
    {
        //Nao tem endereco disponivel
        //return 1;
        Serial.println("Existe um endereço disponível");
    }
    else {
      Serial.println("Não existe um endereço disponível");
    }*/


    //Serial.println("Endereco");
    //Serial.println(endereco);

    for(int count=0;count<6;count++){  //grava senha
        //Serial.println(inicio_senha+ 6*endereco + count),senha_recebida[count]);
```

```
      EEPROM.write((inicio_senha+ 6*endereco + count),senha_recebida[count]);
   }
   for(int count=0;count<11;count++){ //grava RFID
      EEPROM.write((inicio_RFID+ 11*endereco + count),rfid_recebida[count]);
    }
   for(int count=0;count<36;count++){  //grava nome
      EEPROM.write((inicio_nome_usuario+ 36*endereco + count),nome_usuario[count]);
    }

   EEPROM.end();//Fecha a EEPROM.


}

void consulta_endereco(int a, int b){        // consulta os endereços de a até
  EEPROM.begin(4096);//Inicia a EEPROM com 4096 bytes

  for (int i = a; i < b; i++)//Loop que irá mostrar no Serial monitor cada valor da EEPROM.
  {

    Serial.print(i);
    Serial.println(EEPROM.read(i));     //imprime o endereço, char para converter int -> caracter
  }
  EEPROM.end();//Fecha a EEPROM.
}



void limpa_endereco(int a, int b){         //limpa os endereços de a até b
  EEPROM.begin(4096);//Inicia a EEPROM com 4096 bytes

  for (int i = a; i < b; i++)//Loop que irá mostrar no Serial monitor cada valor da EEPROM.
  {
      if(i != 255){
      (EEPROM.write(i,255));        //imprime o endereço, char para converter int -> caracter
    }
  }

  EEPROM.end();//Fecha a EEPROM.
```

```
}

////////////////// Rotina para comunicar com a UART0 do ESP//////////////////

void comunica_externo()
{
    Serial.write("status");
    for(contador_timeot_ext = 0; contador_timeot_ext < 300 ; contador_timeot_ext++)
// timeout= 300 # tempo cada com uma instrução, clock=80MHz
    {
      if(Serial.available())
      {
        receivedChars = Serial.readString();   //Recebe a msg do modulo externo
        break;
      }
      receivedChars = "";                 ////////////////////// ??????????????????
    }

    //Proc. mensagem
    if(receivedChars == "nada")         //Compara a string diretamente sem strcmp
    {



    }
    else if(receivedChars,"config")
    {



    }
    else if(receivedChars.length() > 6)  //Se for maior que 6 caracteres é do RFID
    {  //RFID
      rfid_recebida = receivedChars;
      if(procura_rfid())
      {  //Achou um usuario
        AbrirPorta();
        procura_nome_usuario();
        Serial.write(nome_user);  //manda nome para display externo
        nomeusuario = nome_user;;  //manda para GUI-phyton
      }
      else
      {  //Nao achou usuario
        Serial.write("Usuario não encontrado");
        nomeusuario = ("Usuario não encontrado");  //manda para GUI-phyton
      }
```

```
    }
    else
    {  //Senha
       senha_recebida = receivedChars;
       if(procura_senha())
       {  //Achou um usuario
          AbrirPorta();
          procura_nome_usuario();
          Serial.write(nome_user);  //manda nome para display externo
          nomeusuario = nome_user;  //manda para GUI-phyton
       }
       else
       {  //Nao achou usuario
          Serial.write("Usuario não encontrado");
          nomeusuario = ("Usuario não encontrado");  //manda para GUI-phyton
       }
    }

}


void setup() {



  pinMode(D2, INPUT); // Configura pino D2 como entrada;
  debouncer.attach(D2); // Informa que o tratamento de debouce será feito no pino
D2;
  debouncer.interval(10); // Seta o intervalo de trepidação;

  pinMode(D1, OUTPUT); //

  Serial.begin(115200);                //Inicializa a UART0 do atmega
  Serial1.begin(115200);               //Incializa a UART1 do atmega

  IPAddress staticIP(192, 168, 1,110); // IP set to Static
  IPAddress gateway(192, 168, 1, 1);// gateway set to Static
  IPAddress subnet(255, 255, 255, 0);// subnet set to Static

  WiFi.config(staticIP, gateway, subnet);

  WiFi.begin(ssid, password);
```

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(10);
    Serial.println("Connecting..");
  }

  Serial.print("Connected to WiFi. IP:");
  Serial.println(WiFi.localIP());

  wifiServer.begin();

  yield();                    // faz o feed do SW WDT, para não resetar o esp


}




void loop() {

        proc_stack();       //Função para receber e enviar para dados para a GUI
        comunica_externo();   //Funçõa que comunica com o modulo externo
        //procura_senha();


        //senha_recebida = "123456";
        //rfid_recebida = "B6 B2 DA 2B";
        //nome_usuario = "Jean Alves da Costa";
        //adiciona_senha_RFID_usuario();
        //limpa_endereco(45,1500);
        //consulta_endereco(525,535);
        //delay(5000);



}
```

Anexo III – Código do Atmega2560

```
#include <Keypad.h> //INCLUSÃO DE BIBLIOTECA
#include "U8glib.h"   // Biblioteca para o controlador ST7920
#include <SPI.h>
#include <MFRC522.h>


//////////////////////INICIALIZAÇÃO DO TECLADO////////////////////////////////////////

const byte qtdLinhas = 4; //QUANTIDADE DE LINHAS DO TECLADO
const byte qtdColunas = 4; //QUANTIDADE DE COLUNAS DO TECLADO

//CONSTRUÇÃO DA MATRIZ DE CARACTERES
char matriz_teclas[qtdLinhas][qtdColunas] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};


int aux=0;
//String ip = "Ola";
//byte PinosqtdLinhas[qtdLinhas] = {3, 4, 5, 6}; //PINOS UTILIZADOS PELAS LINHAS
//byte PinosqtdColunas[qtdColunas] = {8, 9, 10,11}; //PINOS UTILIZADOS PELAS COLUNAS

byte PinosqtdLinhas[qtdLinhas] = {6, 7, 8, 9}; //PINOS UTILIZADOS PELAS LINHAS
byte PinosqtdColunas[qtdColunas] = {10, 11, 12,13}; //PINOS UTILIZADOS PELAS COLUNAS


Keypad meuteclado = Keypad( makeKeymap(matriz_teclas), PinosqtdLinhas, PinosqtdColunas, qtdLinhas, qtdColunas);


//////////////////Variáveis Globais////////////////////////////////////

String IP = "";
String senha_rfid="";
String rfid="";
```

```
//////////////////////Configuração do RFID//////////////////////////
#define SS_PIN 53
#define RST_PIN 49
MFRC522 mfrc522(SS_PIN, RST_PIN);   // Create MFRC522 instance.

char st[20];

//////////////////////Variáveis UART1 do Atmega/////////////////

//const byte numChars = 100;
//char receivedChars[numChars];
String receivedChars;


//////////////////////////Configuração do Display//////////////////////////////

//Configuração de Pinagem, Enable, RW, RS, RESET
U8GLIB_ST7920_128X64_1X Display(4, 3, 2 , 5);

// (0,0) É Ponto Superior esquerdo para referencia
// (128,64)

//////////////////// Rotina de Configuração da Escrita no Display//////////////////////
void Display_config() {

  Display.setFont(u8g_font_6x10);        //definir a fonte das mensagem mostradas no display
  Display.setFontRefHeightExtendedText(); //definimos a altura de referência padrão para o modelo do display utilizado, no caso o ST7920.
  Display.setDefaultForegroundColor();    //define a cor de fundo padrão
  Display.setFontPosTop();                //definido o tamanho de fonte máximo permitido para o display.
}

//////////////////////// Telas que o Display ira exibir ////////////////////////////////////////////
void Tela1() {

  Display.setFont(u8g_font_6x10);      //responsável por selecionar a fonte que será empregada na escrita
  Display.drawStr(30, 0, "Controlador");     //string entre aspas no display na posição desejada.
  Display.drawStr(60, 20, "de");
  Display.drawStr(45, 45, "Acesso");
```

```
}

void Tela2() {

  Display.setFont(u8g_font_6x10);      //responsável por selecionar a fonte que será empregada na escrita
  Display.drawStr(30, 0, "Tecla");       //string entre aspas no display na posição desejada.
  Display.drawStr(30, 20, "Pressionada");
  Display.drawStr(50, 45, IP.c_str());
}

void Tela3() {

  Display.setFont(u8g_font_6x10);      //responsável por selecionar a fonte que será empregada na escrita
  Display.drawStr(30, 0, "Ola");    //string entre aspas no display na posição desejada.
  Display.drawStr(30, 20, "Usuario");
  Display.drawStr(5, 45, senha_rfid.c_str()); //obtendo um char * usando o c_str()método String
}

void Tela4() {

  Display.setFont(u8g_font_6x10);      //responsável por selecionar a fonte que será empregada na escrita
  Display.drawStr(30, 0, "Ola");    //string entre aspas no display na posição desejada.
  //Display.drawStr(30, 20, "Usuario");
  Display.drawStr(30, 45, senha_rfid.c_str()); //obtendo um char * usando o c_str()método String
  Display.drawStr(30, 20, receivedChars.c_str()); //obtendo um char * usando o c_str()método String
}


void Tela3_1() {
  /*é instanciado a variável ASCII (do tipo char) que recebe os valores (do tipo int), e exibe o seu caractere
  correspondente por meio da intrução Display.drawStr(coluna, linha, ASCII).
  os caracateres especiais exibidos correpondem aos numeros em decimal de 32 a 127.
  Os dois for presentes na função são utilizados para imprimir os caracteres ao longo de toda a área do display,
```

sendo que o primeiro for é responsáel por pular linhas ao passo que na linhas corresponde não há mais espaço para exibir mais caracteres.

Já o segundo for, ou for interno, é responsável por definir a posição de cada caractere ao longo dos 128 pixels disponíveis do display.

A variável dec so pode ser incrementada até o valor 127, que é o valor decimal correspondente ao último caractere especial da tabela ASCII,

por isso a variável foi incrementada dentro do segundo for, que por coincidência também tem limite de 127, definido pelo numero máximo de pixels do display.*/

```
  char ASCII[2] = " ";
  int dec = 32;

  Display.setFont(u8g_font_robot_de_niro);

  for (int linha = 10; linha < 70; linha += 10) {
    for (int coluna = 2; coluna < 128; coluna += 8) {
      ASCII[0] = dec;
      Display.drawStr(coluna, linha, ASCII);
      dec ++;
    }
  }

}


void verifica_teclado(){
char tecla_pressionada = meuteclado.getKey(); //VERIFICA SE ALGUMA DAS TECLAS FOI PRESSIONADA

  if (tecla_pressionada){
    IP = tecla_pressionada;
   //Serial.println(tecla_pressionada);
  }

}

void verifica_RFID(){

   // Look for new cards
  if ( ! mfrc522.PICC_IsNewCardPresent())
  {
    return;
  }
```

```
  // Select one of the cards
  if ( ! mfrc522.PICC_ReadCardSerial())
  {
    return;
  }
  int count_rfid = 0;
  String conteudo= "";
  String a = "";
  //byte letra;
  for (byte i = 0; i < mfrc522.uid.size; i++)
  {
     //conteudo.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
     conteudo.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
     conteudo.concat(String(mfrc522.uid.uidByte[i], HEX));
  }
  Serial.println(a);
  Serial.println(conteudo);
  //Serial.print("Mensagem : ");
  conteudo.toUpperCase();
  //senha_rfid = conteudo.substring(1);


  //Serial.print(senha_rfid);
  if (conteudo.substring(1) == ("83 6F 4C 79")){
    senha_rfid = "Jean Alves da Costa";
  }
  delay(500);                    //delay para evitar de identificar + de uma vez
   if (conteudo.substring(1) == "B6 B2 DA 2B") {
        senha_rfid = "Bruno Ricobom";//UID 2 - Cartao
   }

}

////////////////// Rotina para comunicar com a UART1 do Atmega//////////////////

void comunica_esp()
{
    for(int contador_timeot_ext = 0; contador_timeot_ext < 300 ;
contador_timeot_ext++)                // timeout= 300 # tempo cada com uma instrução,
clock=80MHz
    {
       if(Serial1.available())
       {
         receivedChars = Serial1.readString();   //Recebe a msg do esp
```

```
      break;
    }
    receivedChars = "";                 ////////////////////// ??????????????????
  }

  Serial.println(receivedChars);
  //Proc. mensagem
  if(receivedChars == "Status")        //Compara a string diretamente sem strcmp
  {
   if ( senha_rfid != "") {       // verifica se uma RFID foi identificada
      Serial1.write(senha_rfid.c_str());   // Se existir uma RFID envia pela UART
   }


  for(int contador_timeot_ext = 0; contador_timeot_ext < 300 ;
contador_timeot_ext++)                // timeout= 300 # tempo cada com uma instrução,
clock=80MHz
  {
    if(Serial1.available())
    {
      receivedChars = Serial1.readString();   //Recebe a msg do esp
      break;
    }
    receivedChars = "";                 ////////////////////// ??????????????????
  }

  if(receivedChars != "")
  {
     Tela4();
  }


  }



  /*else if(receivedChars,"config")
  {


  }
  else if(receivedChars.length() > 6)  //Se for maior que 6 caracteres é do RFID
  {  //RFID
    rfid_recebida = receivedChars;
```

```
        if(procura_rfid())
        {  //Achou um usuario
           AbrirPorta();
           procura_nome_usuario();
           Serial.write(nome_user);  //manda nome para display externo
           nomeusuario = nome_user;;  //manda para GUI-phyton
        }
        else
        {  //Nao achou usuario
           Serial.write("Usuario não encontrado");
           nomeusuario = ("Usuario não encontrado");  //manda para GUI-phyton
        }

     }
     else
     {  //Senha
        senha_recebida = receivedChars;
        if(procura_senha())
        {  //Achou um usuario
           AbrirPorta();
           procura_nome_usuario();
           Serial.write(nome_user);  //manda nome para display externo
           nomeusuario = nome_user;  //manda para GUI-phyton
        }
        else
        {  //Nao achou usuario
           Serial.write("Usuario não encontrado");
           nomeusuario = ("Usuario não encontrado");  //manda para GUI-phyton
        }
     }*/

}

void setup(){
  Serial.begin(115200); // Iniciliza a UART0
  Serial1.begin(115200);

    //////////Inicializações do Display//////////////////
    if ( Display.getMode() == U8G_MODE_R3G3B2 )
    Display.setColorIndex(20);
  else if ( Display.getMode() == U8G_MODE_GRAY2BIT )
    Display.setColorIndex(1);
  else if ( Display.getMode() == U8G_MODE_BW )
    Display.setColorIndex(1);
```

```
   ////////////Inicializações do RFID/////////////////
  SPI.begin();      // Inicia  SPI bus
  mfrc522.PCD_Init();   // Inicia MFRC522

}



void loop(){
   verifica_teclado();
   verifica_RFID();
   comunica_esp();   //Função que comunica com o esp


  /*  //Tela 1
  Display.firstPage();
  do {
    Display_config();
    Tela1();
  }
  while (Display.nextPage());
  delay(300);

   //Tela 2
  Display.firstPage();
  do {
    Display_config();
    Tela2();
  }
  while (Display.nextPage());
  delay(300);*/

     //Tela 3
  Display.firstPage();
  do {
    Display_config();
    Tela3();
  }
  while (Display.nextPage());
  //delay(300);


}
```