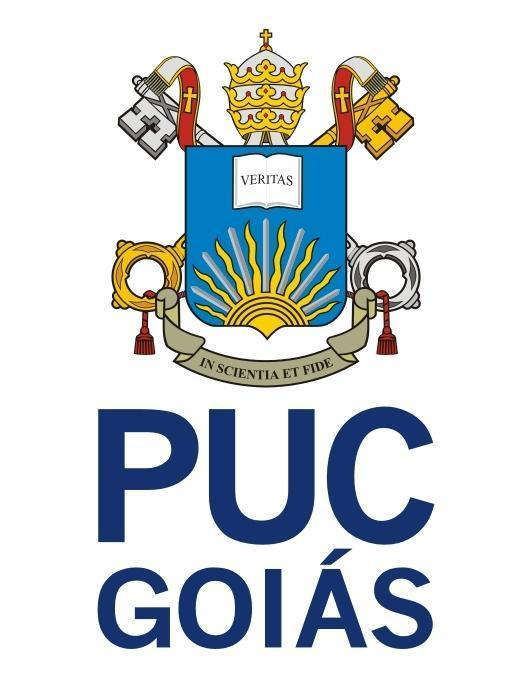
PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS

ESCOLA DE CIÊNCIAS EXATAS E DA COMPUTAÇÃO

GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO



**ESP32, RFID e MySQL a Porta Deve Abrir? Sim ou Não?**

DANILO DIAS COSTA

GOIÂNIA

2020

Danilo Dias Costa

**ESP32, RFID e MySQL a Porta Deve Abrir? Sim ou Não?**

Trabalho de Conclusão de Curso apresentado à Escola de Ciências Exatas e da Computação, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador(a):

Profº Ms. Wilmar Oliveira de Queiroz

GOIÂNIA,

2020

DANILO DIAS COSTA

**ESP32, RFID e MySQL a Porta Deve Abrir? Sim ou Não?**

Este Trabalho de Conclusão de Curso julgado adequado para obtenção do título de Bacharel em Engenharia de Computação, e aprovado em sua forma final pela Escola de Ciências Exatas e da Computação, da Pontifícia Universidade Católica de Goiás, em \_\_\_\_\_/\_\_\_\_\_/\_\_\_\_\_\_\_\_\_.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Me. Ludmilla Reis Pinheiro dos Santos

Coordenadora de Trabalho de Conclusão de Curso

Banca Examinadora:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Orientador: Prof. Me. Wilmar Oliveira de Queiroz

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Me. Angélica da Silva Nunes

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Dr. Sibelius Lellis Vieira

GOIÂNIA

2020**DEDICATÓRIA**

A Deus que me deu força e coragem para vencer todos os obstáculos. Ao meu avô Jair que mesmo não estando mais aqui me deixou ensinamentos importantes e uma forma de ver a vida que me ajudou a concluir esse trabalho, mesmo com todos os problemas e frustrações.

**AGRADECIMENTOS**

Ao professor Ms. Wilmar Oliveira de Queiroz, que teve papel fundamental na elaboração deste trabalho, com seu apoio e dedicação.

Aos meus colegas Vitor, Lucas, Higor, Marcos Vinicius, Marcos Aurélio, Lorena e Lais por me auxiliarem em vários momentos.

Ao meu amigo Marco por toda a paciência e auxílio em vários momentos da construção desse projeto.

“São nos nossos piores dias que precisamos dar nosso melhor”

Ricky Carmichael

**RESUMO**

Este trabalho tem como objetivo criar uma forma para o usuário realizar autenticação e validação de acesso a um determinado local, utilizando o paradigma de Internet das Coisas (*Internet of Things*), junto a RFID (*Radio-Frequency Identification*), ESP32 e um celular. O usuário, através da apresentação de uma etiqueta RFID realizará a validação do acesso em uma consulta ao banco de dados. Para o estudo de caso foi criado uma estrutura com banco de dados, um servidor Web Apache com *Simple HTTP Server* utilizando o *Xampp* para realizar o recebimento das informações do microcontrolador ESP32 e processá-las por meio de uma aplicação em PHP (*Personal Home Page*). Tem-se, também, o aplicativo *mobile* no qual o usuário efetuará o *login* para cadastrar ou remover novas etiquetas RFID para autenticação. Como resultado, foi possível realizar a autenticação, remoção ou adição de etiquetas de uma forma rápida.

**Palavras-Chave:** ESP32, RFID, Banco de dados.

**ABSTRACT**

This work aims to create a way for the user to perform authentication and validation of access to a given location, using the Internet of Things paradigm, along with RFID (Radio-Frequency Identification), ESP32 and a cell phone. The user, through the presentation of an RFID tag, validates the access in a query to the database. For the case study, a database structure was created, an Apache Web server with Simple HTTP Server using Xampp to receive the information from the ESP32 microcontroller and process it using a PHP application (Personal Home Page) . There is also a *mobile* application in which the user will log in to register or remove new RFID tags for authentication. As a result, it was possible to quickly authenticate, remove or add tags.

**Keywords:** ESP32, RFID, Database.

**LISTA DE FIGURAS**

[Figura 1- ESP32 19](#_Toc58919727)

[Figura 2 - Arquitetura e organização RFID 20](#_Toc58919728)

[Figura 3 - Sinal de leitura RFID 21](#_Toc58919729)

[Figura 4 - Leitura das etiquetas RFID 22](#_Toc58919730)

[Figura 5 - Datagrama mensagem *Query* 23](#_Toc58919731)

[Figura 6 - Logo Sistema Operacional Android 24](#_Toc58919732)

[Figura 7 - Conexão *Web Server* 25](#_Toc58919733)

[Figura 8 - Modelo de comunicação SPI 26](#_Toc58919734)

[Figura 9 - Modelo de comunicação SPI mestre e escravos 27](#_Toc58919735)

[Figura 10 - Diagrama Entidade Relacionamento 28](#_Toc58919736)

[Figura 11 - Fluxograma validação de *tag* 30](#_Toc58919737)

[Figura 12 - Fluxograma remoção de *tag* 31](#_Toc58919738)

[Figura 13 - Fluxograma adição de *tag* 31](#_Toc58919739)

[Figura 14 - Fluxograma *login* do usuário no sistema 32](#_Toc58919740)

[Figura 15 - Formulário em PHP do arquivo index.php 33](#_Toc58919741)

[Figura 16 - Formulário em PHP do arquivo *login*.php 34](#_Toc58919742)

[Figura 17 - Página de *login* aplicativo com erro 35](#_Toc58919743)

[Figura 18 - Página de *login* aplicativo 35](#_Toc58919744)

[Figura 19 - Painel do usuário em PHP do arquivo painel.php 36](#_Toc58919745)

[Figura 20 - Painel do usuário do aplicativo 36](#_Toc58919746)

[Figura 21 - Código PHP do arquivo logout.php 37](#_Toc58919747)

[Figura 22 - Cadastro de *tag* PHP arquivo adicionarTag.php 37](#_Toc58919748)

[Figura 23 - Página cadastrar *tag* do aplicativo 38](#_Toc58919749)

[Figura 24 - Mensagem de erro campo não preenchido 38](#_Toc58919750)

[Figura 25 - Mensagem de erro de *tag* já cadastrada 39](#_Toc58919751)

[Figura 28 - Mensagem *tag* cadastrada 39](#_Toc58919752)

[Figura 29 - Remoção de *tag* PHP arquivo removerTag.php 40](#_Toc58919753)

[Figura 30 - Página remover *tag* do aplicativo 40](#_Toc58919754)

[Figura 29 - Mensagem de erro campo não preenchido 41](#_Toc58919755)

[Figura 30 - Mensagem de erro de *tag* não existente 41](#_Toc58919756)

[Figura 31 - Mensagem de erro de usuário incorreto 42](#_Toc58919757)

[Figura 32 - Mensagem *tag* removida 42](#_Toc58919758)

[Figura 33 - Método de conexão conexao2.php 43](#_Toc58919759)

[Figura 34 - Método de conexão conexao.php 43](#_Toc58919760)

[Figura 35 - Arquivo XML página Android 44](#_Toc58919761)

[Figura 36 - Código WebView do aplicativo 45](#_Toc58919762)

[Figura 37 - Definição das informações no ESP32 46](#_Toc58919763)

[Figura 38 - Definição void setup( ) para o ESP32 47](#_Toc58919764)

[Figura 39 - Definição void loop ( ) para o ESP32 48](#_Toc58919765)

[Figura 40 - Função leituraDados() 49](#_Toc58919766)

[Figura 41 - Arquivo validacao.php 50](#_Toc58919767)

[Figura 42 - Circuito montado 51](#_Toc58919768)

[Figura 43 - Diagrama de conexão e pinagem entre componentes 52](#_Toc58919769)

[Figura 44 - Aproximação da *tag* no leitor RFID 53](#_Toc58919770)

[Figura 45 - *Tag* cadastrada no banco de dados 54](#_Toc58919771)

[Figura 46 - Reposta do servidor *tag* cadastrada 54](#_Toc58919772)

[Figura 47 - *Tag* cadastrada no banco de dados 55](#_Toc58919773)

[Figura 48 - Reposta do servidor *tag* não cadastrada 55](#_Toc58919774)

[Figura 49 - Aguardando aproximação de uma *tag* 56](#_Toc58919775)

**LISTA DE TABELAS**

[Tabela 1 – Exemplo da tabela Clientes com informações de usuários. 28](#_Toc58793119)

[Tabela 2 - Exemplo da tabela Tags com informações das *tags* 29](#_Toc58793120)

[Tabela 3 - Tabela de pinagem 52](#_Toc58793121)

**LISTA DE SIGLAS**

|  |  |
| --- | --- |
| ACK | Reconhecimento – Acknowledgement |
| ASK | Chaveamento de mudança de amplitude – Amplitude Shift Keying |
| CGI | Interface de portão comum – Common Gateway Interface |
| CPF | Cadastro de Pessoas Físicas |
| CSS | Folha de estilo em cascata – Cascading Style Sheets |
| EPC | Código eletrônico do produto – Electronic Product Code |
| HTML | Linguagem de marcação de hipertexto – HyperText Markup Language |
| HTTP | Protocolo de Transferência de Hipertexto – HyperText Transfer Protocol |
| IP | Protocolo de Internet – Internet Protocol |
| ISM | Industrial científico e médico – Industrial Scientific and Medical |
| LED | Diodo emissor de Luz – Light emitting Diode |
| MISO | Entrada mestre entrada escravo – Master Input Slave Output |
| MOSI | Saída mestre entrada escravo – Master Output Slave Input |
| PDO | Objetos de dados PHP – PHP Data Objects |
| PHP | Página pessoal – Personal Home Page |
| RAM | Memória de acesso aleatório – Random Access Memory |
| RFID | Identificação por radiofrequência – Radio Frequency Identification |
| ROM | Memória somente de leitura – Read Only Memory |
| SPI | Interface Periférica Serial – Serial Peripheral Interface |
| SQL | Linguagem de consulta padrão – Standard Query Language |
| UHF | Frequência ultra-alta – Ultra High Frequency |
| UID | Identificação única – Unique Identification |
| URL | Localizador padrão de recursos – Uniform Resource Locator |

**Sumário**

[1 INTRODUÇÃO 15](#_Toc58793128)

[1.1 Objetivo Geral 16](#_Toc58793129)

[1.2 Objetivos Específicos 16](#_Toc58793130)

[1.3 Procedimentos Metodológicos 17](#_Toc58793131)

[1.4 Estrutura da Monografia 17](#_Toc58793132)

[2 FUNDAMENTAÇÃO TEÓRICA 18](#_Toc58793133)

[2.1 Protocolo HTTP (*HyperText Transfer Protocol*) 18](#_Toc58793134)

[2.2 ESP32 19](#_Toc58793135)

[2.3 RFID (*Radio Frequency Identification*) 20](#_Toc58793136)

[2.4 Android 24](#_Toc58793137)

[2.5 MySQL 24](#_Toc58793138)

[2.6 Servidor *Web* - *Web Server* 25](#_Toc58793139)

[2.7 *Personal Home Page*, Bootstrap e *HyperText Markup Language* 26](#_Toc58793140)

[2.8 Protocolo *Serial Peripheral Interface* 26](#_Toc58793141)

[3 DESENVOLVIMENTO DO PROJETO 28](#_Toc58793142)

[3.1 Base de Dados 28](#_Toc58793143)

[3.1.1 Clientes 28](#_Toc58793144)

[3.1.2 Tags 29](#_Toc58793145)

[3.2 Ambiente Utilizado 29](#_Toc58793146)

[3.3 Funcionalidades 30](#_Toc58793147)

[3.4 Divisão de Tarefas no Sistema 32](#_Toc58793148)

[3.4.1 Login de usuário 33](#_Toc58793149)

[3.4.2 Painel do usuário 36](#_Toc58793150)

[3.4.3 Adicionar *tag* 37](#_Toc58793151)

[3.4.4 Remover *tag* 39](#_Toc58793152)

[3.4.5 Conexão com Banco de Dados 42](#_Toc58793153)

[3.4.6 Aplicativo Android 43](#_Toc58793154)

[3.4.7 ESP32 Verificação de *tag* 46](#_Toc58793155)

[4. RESULTADOS 51](#_Toc58793156)

[5 CONCLUSÃO 57](#_Toc58793157)

[5.1 Trabalhos Futuros 57](#_Toc58793158)

[5.1.1 Complementação da solução 57](#_Toc58793159)

[5.1.2 Implementações Similares 58](#_Toc58793160)

[6 REFERÊNCIAS 59](#_Toc58793161)

[ANEXO A – INSTALAÇÃO XAMPP 61](#_Toc58793162)

[ANEXO B - INSTALAÇÃO VISUAL STUDIO CODE 66](#_Toc58793163)

[ANEXO C – CONFIGURAÇÃO ARDUINO IDE PARA O ESP32 69](#_Toc58793164)

# INTRODUÇÃO

Para o controle de acesso a um usuário em um determinado meio, é utilizados diversas formas para poder controlar esse acesso. Autenticação é entendido como o ato de estabelecer ou confirmar algo como autêntico (Aurélio, 2010).

Esses sistemas podem fazer a utilização de RFID ou não. O *Radio Frequency Identification* (RFID) é frequentemente usado para rastreamento de objetos, por exemplo, controle de estoque, rastreamento de paletes (MARQUARDT; TAYLOR; VILLAR; GREENBERG, 2010). Um sistema de RFID é composto, basicamente, de uma antena, um transceptor, que faz a leitura do sinal e transfere a informação para um dispositivo leitor e, também, um transponder ou etiqueta de RF (Rádio Frequência). Essa etiqueta pode ser de dois tipos:

* Passiva – Estas etiquetas utilizam a rádio frequência do leitor para transmitir o seu sinal e normalmente têm com suas informações gravadas permanentemente quando são fabricadas.
* Ativa – As etiquetas ativas são muito mais sofisticadas e caras e contam com uma bateria própria para transmitir seu sinal sobre uma distância razoável, além de permitir armazenamento em memória RAM capaz de guardar até 32 KB.

Na ponta disso tudo existe um sistema embarcado que pode ser definindo vagamente como qualquer dispositivo que possua um processador programável de objetivo específico, que não seja um computador de uso pessoal (Wolf ;2012).

Com os já citados conceitos, este trabalho busca uma forma para tornar a solução de autenticação mais fácil, para a utilização de maior número possível de usuários e com uma forma que não traga complicações de gerenciar. Foi desenvolvido um controle de acesso de usuários e de gerência de etiquetas RFID usando o RFID, ESP32, banco de dados e aplicativo Android.

Um microcontrolador, chamado ESP 32 como segundo SOUZA (2005) um microcontrolador pode ser definido como um pequeno componente eletrônico, dotado de uma inteligência programável, utilizado no controle de processos lógicos ou ZANCO (2005), o microcontrolador é uma *Central Processing Unit* (CPU) de pequeno porte, capaz de executar um conjunto de tarefas denominadas instruções. O ESP 32 receberá os sinais do sistema RFID, processará e enviará os dados via WI-FI para o Web Server onde está o banco de dados utilizando uma requisição HTTP (*Hypertext Transfer Protocol*), o qual liberará ou não o acesso do usuário.

Segundo Deitel P., Deitel H. e Deitel A. (2015) em 2013 existiam cerca de 1,5 milhões de aparelhos com o sistema operacional *Android* ativos diariamente. Com isso, o sistema se mostra bastante utilizado, sendo assim, é mais lógico desenvolver o aplicativo para gerência de etiquetas RFID para Android. Para Silva (2019), o *Android* é uma plataforma utilizada em dispositivos móveis, dispositivos estes que podem ser *smarthphones, tablets, e-readers,* robôs etc. (DEITEL P.; DEITEL H.; DEITEL A., 2015).

## Objetivo Geral

Desenvolver um protótipo de um sistema de autenticação para liberar acesso utilizando RFID, ESP32, banco de dados e que pode ser controlado através do celular.

## Objetivos Específicos

* Desenvolver um aplicativo para gerenciar etiquetas RFID;
* Realizar a criação de um banco de dados íntegro e seguro;
* Realizar a autenticação do usuário via etiqueta RFID.

## Procedimentos Metodológicos

Este trabalho, quanto à natureza, caracteriza-se como um trabalho de aperfeiçoamento, já que busca apresentar uma forma mais simples de um usuário poder realizar o gerenciamento de etiquetas RFID para acesso a sua residência, ou qualquer outro local, e possui uma implicação prática em sua realização (WAZLAWICK, 2014). Pois, será construído um protótipo de um sistema sensorial e de autenticação para validação de acesso de um usuário.

Quanto aos objetivos se caracteriza como uma pesquisa explicativa, pois é realizada a análise de dados observados, busca suas causas e explicações.

Quanto aos procedimentos técnicos, é uma pesquisa experimental, apresentando uma forma de medir e analisar as entradas para o servidor e seu comportamento.

## Estrutura da Monografia

No Capítulo 2, será feita a fundamentação teórica sobre o http, esp32/microcontrolador, RFID (*Radio Frequency Identification*), Android, MySQL e Banco de Dados, Web Server, *Personal Home Page*, Bootstrap e *HyperText Markup Language* e Protocolo Serial Peripheral Interface com seus principais conceitos e sua importância.

No Capítulo 3, contemplará todo o desenvolvimento do trabalho.

No Capítulo 4, serão demonstrados os resultados obtidos ao longo do processo e como essa aplicação pode facilitar alguns campos.

No Capítulo 5, a conclusão sobre a aplicação e sugestões de trabalhos futuros.

# FUNDAMENTAÇÃO TEÓRICA

Neste capitulo é feita uma fundamentação teórica sobre o HTTP (*HyperText Transfer Protocol*), ESP32/microcontrolador, RFID (*Radio Frequency Identification*), Android, MySQL e Banco de Dados, Web Server, *Personal Home Page*, Bootstrap e *HyperText Markup Language* e Protocolo Serial Peripheral Interface com seus principais conceitos e sua importância.

## Protocolo HTTP (*HyperText Transfer Protocol*)

É um protocolo de transferência que roda sobre TCP (*Transmission Control Protocol*) e que possibilita que as pessoas que insiram a URL (*Uniform Resource Locator*) do seu *site* na *Web* possam ver os conteúdos e os dados que nele existem. Esse protocolo é a base da comunicação que existe em toda a *Internet* em que os *sites* e conteúdos que tragam *hiperlinks* possam ser encontrados mais facilmente pelo público por meio de um clique do mouse ou um toque na tela. O HTTP é um protocolo da camada de aplicação.

O TCP, foi projetado especificamente para oferecer um fluxo de *bytes* confiável em uma rede interligada não confiável. Uma rede interligada é diferente de uma única rede porque suas diversas partes podem ter topologias, larguras de banda, atrasos, tamanhos de pacote e outros parâmetros completamente diferentes. O TCP foi projetado para se adaptar dinamicamente às propriedades da rede interligada e ser robusto diante dos muitos tipos de falhas que podem ocorrer.

Na versão HTTP 1.0, depois que a conexão era estabelecida, uma única solicitação era enviada e uma única resposta era devolvida. Então, a conexão TCP era encerrada. Com a chegada do HTTP 1.1, são admitidas conexões persistentes. Com elas, é possível estabelecer uma conexão TCP, enviar uma solicitação e obter uma resposta, e em seguida enviar solicitações adicionais e receber respostas adicionais. Essa estratégia também é chamada reuso de conexão. A Figura 1 ilustra o funcionamento do Protocolo HTTP.

## ESP32

O ESP32 é uma série de microcontroladores de baixo custo e baixo consumo de energia, e é desenvolvido pela empresa Espressif. OESP32 apresenta-se como um meio inovador no desenvolvimento de projetos. Para sua programação é possível utilizar várias IDEs, como a do Arduino, CodeBlocks, Netbeans, Lua e muitas outras. Segundo SOUZA (2005) um microcontrolador pode ser definido como um pequeno componente eletrônico, dotado de uma inteligência programável, utilizado no controle de processos lógicos ou ZANCO (2005), o microcontrolador é uma CPU (*Central Processing Unit*) de pequeno porte, capaz de executar um conjunto de tarefas denominadas instruções. A Figura 1 ilustra um ESP-32.

Figura - ESP32



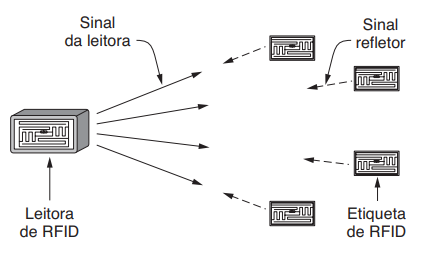
Fonte: ESP32 (ESPRESSIF DOCUMENTATION, 2020)

O ESP32 possui CPU Xtensa® Dual-Core 32-bit LX6, Memória *ROM* de 448 *KBytes*, *Clock* máximo de 240MHz, Memória *RAM* de 520 *Kbytes*, Memória Flash de 4 *MBytes*, *Wireless* padrão 802.11 b/g/n, conexão Wifi de 2.4Ghz (máximo de 150 Mbps). A Figura 3 mostra a arquitetura e a organização do ESP32.

## RFID (*Radio Frequency Identification*)

O RFID tem dois componentes-chave: etiquetas, também conhecidas como *tags*, e leitoras. As etiquetas de RFID são dispositivos pequenos e de baixo custo, que possuem um identificador EPC(*Electronic Product Code*) exclusivo de 96 *bits* e uma pequena quantidade de memória que pode ser lida e escrita pela leitora de RFID. A Figura 2 ilustra a arquitetura do RFID.

Figura - Arquitetura e organização RFID



Fonte: Tanenbaum, 2011

As etiquetas podem ser ativas, passivas ou ambas. As leitoras são muito mais poderosas que as etiquetas. Elas possuem suas próprias fontes de energia, normalmente têm várias antenas e definem quando as etiquetas enviam e recebem mensagens. A tarefa principal da leitora é descobrir os identificadores das etiquetas vizinhas. O inventário é realizado com o protocolo da camada física e o protocolo de identificação de etiqueta. A camada física define como os *bits* são enviados entre a leitora de RFID e as etiquetas.

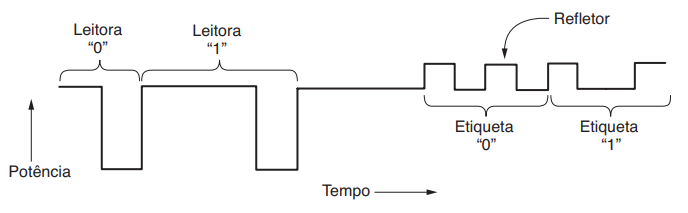
As transmissões são enviadas na banda ISM (*Industrial Scientific and Medical*) não licenciada de 902-928 Megahertz. Visto que essa banda está na faixa do UHF (*Ultra High Frequency*), as etiquetas são conhecidas como etiquetas UHF RFID. A leitora realiza um salto de frequência pelo menos a cada 400 milissegundos, para espalhar seu sinal pelo canal, para limitar a interferência e cumprir os requisitos regulamentares. A leitora e as etiquetas usam as formas de modulação ASK (*Amplitude Shift Keying*), para codificar os *bits*. Elas se alternam para enviar *bits*, de modo que o enlace seja *half-duplex*, ou seja, receba ou mande informações.

Para as etiquetas enviarem *bits* para a leitora, esta transmite um sinal de portadora fixa, que não transporta *bits*. As etiquetas pegam esse sinal para obter a energia de que precisam para funcionar, caso contrário, uma etiqueta não poderia transmitir em primeiro lugar. Para enviar dados, uma etiqueta muda se estiver refletindo o sinal da leitora. Esse método é chamado refletor.

O refletor é um modo de baixa energia para a etiqueta criar um sinal fraco por conta própria, que aparece na leitora. Para que a leitora decodifique o sinal que chega, ela precisa filtrar o sinal de saída que ela mesma está transmitindo. Como o sinal da etiqueta é fraco, elas só podem enviar *bits* para a leitora em uma taxa baixa, e as etiquetas não podem receber ou mesmo detectar transmissões de outras etiquetas.

Para enviar dados para as etiquetas, a leitora usa dois níveis de amplitude. Os *bits* são determinados como 0 ou 1, dependendo do tempo que a leitora espera antes de um período de baixa potência. A etiqueta mede o tempo entre os períodos de baixa potência e compara esse tempo com uma referência medida durante um preâmbulo. A figura 3 ilustra um o sinal de leitura do RFID.

Figura - Sinal de leitura RFID

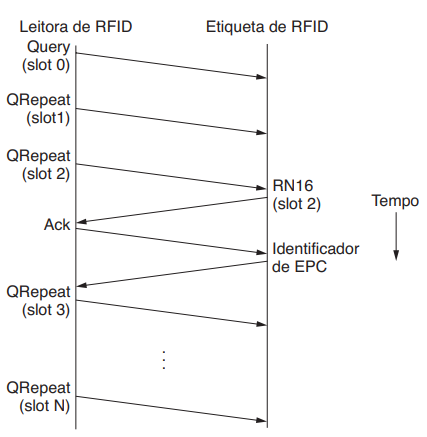


Fonte: Tanenbaum, 2011

As respostas da etiqueta consistem, alternando seu estado refletor em intervalos fixos para criar uma série de pulsos no sinal. Qualquer ponto entre um e oito períodos de pulso pode ser usado para codificar cada 0 ou 1, dependendo da necessidade de confiabilidade. Os 1s têm menos transições que os 0s. Para fazer o inventário de etiquetas vizinhas, a leitora precisa receber uma mensagem de cada etiqueta, oferecendo o identificador para ela.

O *Slotted* ALOHA é utilizado para fazer o inventário das etiquetas, funciona no seguinte formato. No primeiro *slot* (*slot* 0), a leitora envia uma mensagem *Query*para iniciar o processo. Cada mensagem *QRepeat*avança para o *slot* seguinte. A leitora também diz às etiquetas o intervalo de *slots* sobre o qual as transmissões se tornam aleatórias. O uso de um intervalo é necessário porque a leitora sincroniza as etiquetas quando inicia o processo; diferentemente das estações em uma rede Ethernet, as etiquetas não acordam com uma mensagem em um tempo escolhido. As etiquetas apanham um *slot* aleatório para responder. As etiquetas não enviam seus identificadores quando respondem inicialmente. Em vez disso, uma etiqueta envia um número aleatório curto de 16 *bits* em uma mensagem. A figura 4 ilustra como é feita a leitura de etiquetas RFID.

Figura - Leitura das etiquetas RFID



Fonte: Tanenbaum, 2011

Se não houver colisão, a leitora recebe essa mensagem e envia uma mensagem ACK (*Acknowledgement*) própria. Nesse estágio, a etiqueta terá adquirido o *slot* e envia seu identificador de EPC.

O motivo para essa troca é que os identificadores EPC são longos, de modo que colisões nessas mensagens seriam dispendiosas. Em vez disso, uma troca curta é usada para testar se a etiqueta pode usar o *slot* com segurança para enviar seu identificador. Uma vez que seu identificador tiver sido transmitido com êxito, a etiqueta temporariamente deixa de responder a novas mensagens *Query*, para que todas as etiquetas restantes possam ser identificadas.

Um problema importante é que a leitora deve ajustar o número de *slots* para evitar colisões, mas sem usar muitos *slots*, para que o desempenho não seja prejudicado. Se a leitora encontrar muitos *slots* sem respostas ou muitos *slots* com colisões, ela pode enviar uma mensagem *QAdjust* para diminuir ou aumentar o intervalo de *slots* sobre o qual as etiquetas estão respondendo.

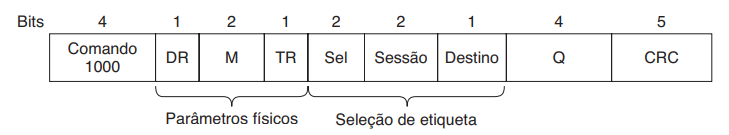
A mensagem é compacta porque as taxas de *downlink* são limitadas, de 27 kbps até 128 kbps. O campo Comando transporta o código 1000 para identificar a mensagem como uma *Query*.

Os próximos *flags*, DR, M e TR, determinam os parâmetros da camada física para transmissões da leitora e respostas da etiqueta

Em seguida vêm três campos, Sel, Sessão e Destino, que selecionam as etiquetas para responder. Assim como as leitoras precisam ser capazes de selecionar um subconjunto de identificadores, as etiquetas precisam acompanhar até quatro sessões simultâneas e se elas foram identificadas nessas sessões. Desse modo, diversas leitoras podem operar nas áreas de cobertura sobrepostas usando diferentes sessões.

O parâmetro mais importante para esse comando, Q. Esse campo identifica o intervalo de *slots* sobre o qual as etiquetas responderão, de 0 a 2Q – 1. CRC para proteger os campos de mensagem. Em 5 *bits*, ele é mais curto que a maioria dos CRCs observados até aqui, mas a mensagem *Query*é muito mais curta do que a maioria dos pacotes também. A figura 5 ilustra o datagrama de uma mensagem *Query*.

Figura - Datagrama mensagem *Query*



Fonte: Tanenbaum, 2011

## Android

Android é o sistema operacional móvel do Google, criado em 2008. Feito para aparelhos celulares, é conhecido por ser baseado no núcleo do Linux e possui um código aberto.

É gravado em Java, C, C++, XML, Assembly, Python, Shell script, Go, make, D. Seus aplicativos geralmente são desenvolvidos em Kotlin ou Java. A interface padrão do usuário no Android é baseada na manipulação direta. A resposta para a manipulação é desenhada para ser imediata e produzir uma sensação de fluidez, utilizando-se constantemente da resposta para informar o usuário sobre a conclusão do comando. A figura 6 demonstra a logo do sistema operacional Android.

Figura - Logo Sistema Operacional Android



Fonte: GOOGLE, 2020

## MySQL

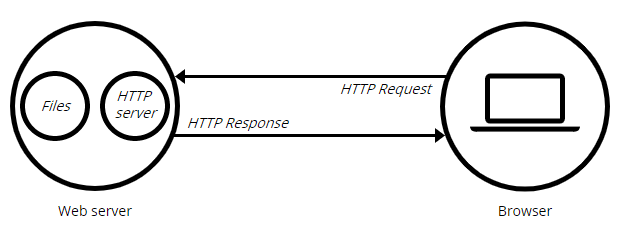
O MySQL é um [sistema de gerenciamento de banco de dados](https://pt.wikipedia.org/wiki/Sistema_de_gerenciamento_de_banco_de_dados), que utiliza a linguagem [SQL](https://pt.wikipedia.org/wiki/SQL) (*Standard Query Language*) como interface. É atualmente um dos sistemas de gerenciamento de bancos de dados mais populares. Um banco de dados é a organização e o armazenamento de informações. De forma mais simples, é o agrupamento de dados que tratam do mesmo assunto, e que precisam ser armazenados para segurança ou conferência futura. Bancos de dados podem seguir dois paradigmas, o relacional e não relacional. Os bancos de dados relacionais são criados no paradigma da orientação a conjuntos. Dessa forma, os dados que ali estão disponíveis serão armazenados em tabelas. Cada tabela terá atributos e linhas ou registros responsáveis por organizar essas informações. Bancos de dados não relacionais são responsáveis por atender a demandas para dados mistos, se misturam tabelas, imagens e mapas, por exemplo, que não poderão ser tabulados em colunas e linhas de tabela.

Bancos de dados possuem uma ou mais colunas que distinguem uma linha das demais dentro de uma tabela, sendo esta chamada de chave primária ou para relacionar com outra tabela, chamada de chave estrangeira. Essas chaves é que determinam a unicidade de cada registro dentro de uma tabela. Na chave primária conjuntos de um ou mais campos, cujos valores, considerando a combinação de valores de todos os campos da tupla (registro), nunca se repetem e que podem ser usadas como um índice para os demais campos da tabela do banco de dados. Em chaves primárias, não pode haver valores nulos nem repetição de tuplas. Podem ser simples ou compostas, na chave simples ela é formada por um único campo da tabela, sendo que este campo não pode ter dois valores ou mais registros de mesmo valor, e não pode conter registro nulo. Quanto a chave composta é formada por mais de um campo, os valores podem se repetir, mas não a combinação desses valores. Chaves estrangeiras por sua vez são utilizadas para criar os relacionamentos entre as tabelas com essa chave estrangeira, podendo facilitar as consultas e fazer cruzamento de dados através destas referências.

## Servidor Web - Web Server

Um servidor *web* inclui diversos componentes que controlam como os usuários acessam os arquivos armazenados, no mínimo um servidor HTTP. Um servidor HTTP é um *software* que compreende endereços *web* e HTTP. Um *web server* pode ser estático ou dinâmico. Um *web server* estático consiste em um computador com um servidor HTTP. É chamado "estático" porque o servidor envia seus arquivos como foram criados e armazenados ao navegador. Um *web server* dinâmico consiste em um servidor *web* estático com *software* adicional, mais comumente um servidor de aplicações e um banco de dados. É chamado "dinâmico" porque o servidor de aplicações atualiza os arquivos hospedados antes de enviá-los ao navegador através do servidor HTTP. Podendo também trabalhar de duas formas síncrono e assíncrono. Em um *web* server síncrono uma requisição é enviada, o processo remetente é bloqueado até que ocorra uma resposta, ou seja, não é possível enviar novas requisições até que a requisição atual seja finalizada, existe sincronismo entre as requisições. Já no assíncrono não existe sincronismo entre as requisições, sendo assim, pode-se enviar diversas requisições em paralelo, onde cada resposta retorna quando estiver pronta. A figura 7 ilustra a comunicação de um cliente/*browser* com um *web* server.

Figura - Conexão *Web Server*



Fonte: Developer Mozilla, 2020

## *Personal Home Page*, Bootstrap e *HyperText Markup Language*

O PHP (*Personal Home Page*) é uma linguagem interpretada livre, usada originalmente para o desenvolvimento de aplicações presentes e atuantes no lado do servidor, capazes de gerar conteúdo dinâmico para *web*. Foi criada por Rasmus Lerdof. O PHP, é uma das linguagens de programação mais usadas no mundo. O termo PHP foi criado com apenas um aglomerado de códigos CGI (*Common Gateway Interface*) – um elemento que torna a ligação física ou lógica entre dois sistemas ou servidores, descritos em uma linguagem C.

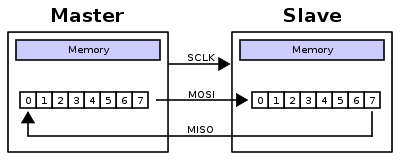
O Bootstrap é um framework *web* com código-fonte aberto para desenvolvimento de componentes de interface e *front-end* para sites e aplicações *web* usando HTML (*HyperText Markup Language*), *Cascading Style Sheets* (CSS) e JavaScript, baseado em modelos de design já prontos. Já o HTML é uma linguagem de marcação utilizada na construção de páginas na Web.

## Protocolo Serial Peripheral Interface

SPI (Serial Peripheral Interface) é um protocolo que permite a comunicação do microcontrolador com diversos outros componentes, formando uma rede. É uma especificação de interface de comunicação série síncrona usada para comunicação de curta distância, principalmente em sistemas embarcados. A interface foi desenvolvida pela Motorola e tornou-se um padrão. Os dispositivos SPI comunicam entre si em modo *full duplex* usando uma arquitetura *master-slave* (mestre-escravo) com um único mestre. O dispositivo mestre origina o sinal para a leitura e a escrita.

No protocolo SPI, quer o *master* quer o *slave* possuem internamente um registo de deslocamento que serão ligados entre si por duas linhas de dados. A figura 8 ilustra esse modelo de comunicação.

Figura - Modelo de comunicação SPI



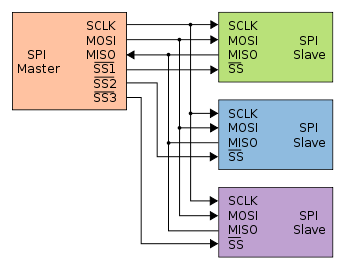
Fonte: Sousa Hélio, 2020

**MOSI** – *Master Output Slave Input*

**MISO** – *Master Input Slave Output*

Os *bits* são deslocados ao ritmo do sinal de relógio (SCLK) gerado pelo mestre. Para se poder ligar diferentes escravos, o master controla o valor presente em determinadas linhas de seleção (designadas na Figura 9 por SSx). O escravo que tiver a sua linha ativa, saberá que os dados na linha MOSI são para si, e/ou que pode gerar dados na linha MISO. Deste modo não há qualquer conflito ao nível das linhas MISO pois só uma estará ativa e as restantes estarão em alta impedância (Alta Impedância significa que um ponto em um circuito permite uma quantidade relativamente pequena de corrente através de, por unidade de tensão aplicada nesse ponto.). A figura 9 demonstra essa comunicação entre mestre e escravos.

Figura - Modelo de comunicação SPI mestre e escravos



Fonte: SOUSA Hélio, 2020

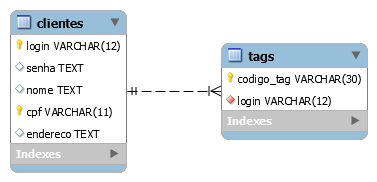
# DESENVOLVIMENTO DO PROJETO

Este capítulo trata como este projeto foi desenvolvido. Será feita a abordagem explicando o que foi feito e como em cada etapa.

## Base de Dados

O projeto possui dentro de sua estrutura, um banco de dados chamado de dados e duas tabelas uma chamada de clientes e outra de *tags*. Essas tabelas correspondem aos sistemas de *login* de usuários e autenticação de *tags*. A Figura 10 demonstra o Diagrama Entidade Relacionamento do banco de dados que foi criado.

Figura - Diagrama Entidade Relacionamento



Fonte: Autoria Própria, 2020

### Clientes

A tabela Clientes é responsável por armazenar as informações dos usuários registrados no sistema e, portanto, aptos a realizar *login* dando a permissão de adicionar e remover *tags* desde que vinculadas ao *login* que está realizando essa operação.

Para esse trabalho como informações foram escolhidas: *login*, senha, nome, cpf (Cadastro de Pessoas Físicas) e endereço. Tendo-se *login* e cpf como chaves primarias. A estrutura dessa tabela é demonstrada a partir de um modelo assim como a Tabela 1.

Tabela – Exemplo da tabela Clientes com informações de usuários.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *login* | senha | nome | cpf | endereco |
| user1 | 123456 | Joãozinho | 04125612389 | Nárnia guarda-roupa 1 |
| User2 | 654321 | Maria | 04125612301 | Nárnia guarda-roupa 2 |

Definição sobre cada coluna:

**login** – É um valor de até 12 dígitos referente ao valor decimal ou textual utilizado como forma de identificação do usuário;

**senha** – É a palavra secreta de cada usuário e será utilizada junto ao *login* para validar a identificação do usuário;

**nome** – Nome próprio do usuário;

**cpf** – É o cadastro de pessoa física do usuário;

**endereco** – Endereço do usuário.

### Tags

A tabela Tags, é responsável por armazenar o valor do identificador de uma *tag* e fazer o vínculo com o *login*. É nessa tabela que será consultado se uma *tag* tem acesso ou não.

Para esse trabalho como informações foram escolhidas: código da *tag* e *login*. Tendo-se o código da *tag* como chave primária e o *login* uma chave estrangeira que faz referência ao *login* da tabela Clientes. A estrutura dessa tabela é demonstrada a partir de um modelo assim como a Tabela 2.

Tabela - Exemplo da tabela Tags com informações das *tags*

|  |  |
| --- | --- |
| codigo\_*tag* | *login* |
| C7FFA433 | user1 |
| C7FFA435 | user2 |

Definição sobre cada coluna:

**codigo\_*tag* –** É o valor do UID da *tag*, ou seja, o valor do seu identificador;

**login** – É um valor de até 12 dígitos referente ao valor decimal ou textual utilizado como forma de identificação do usuário.

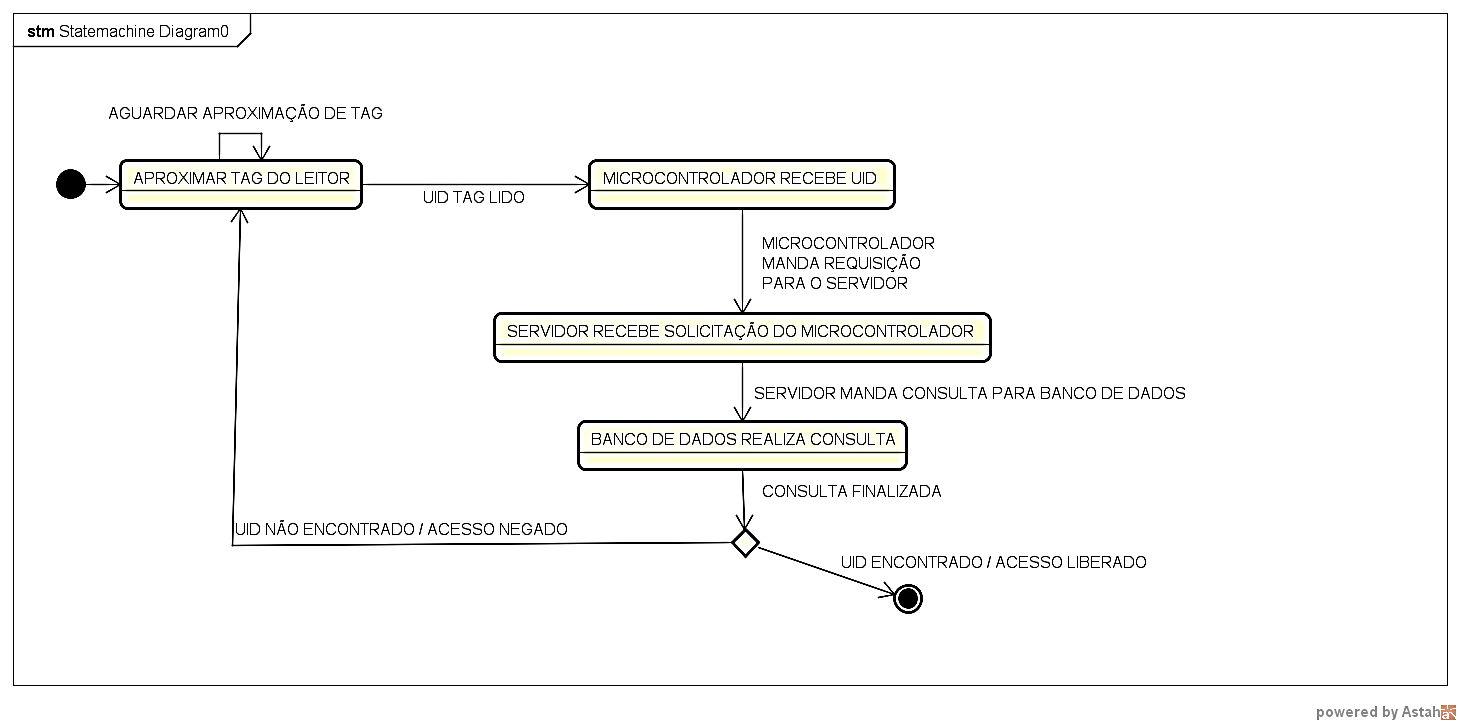
## Ambiente Utilizado

Neste trabalho, utilizou-se um microcontrolador ESP32 do modelo DEV KIT V1 que foi programado para que realizasse as tarefas desejadas. Como IDE para desenvolvimento o desenvolvimento do aplicativo para Android foi utilizado o Android Studio, Visual Studio Code para a programação do PHP, HTML e Bootstrap e Arduino IDE para programação no ESP32. Essa placa, não veio instalada nativamente na IDE do Arduino. Por isso, foi necessário a sua instalação como demonstrado no Anexo C.

## Funcionalidades

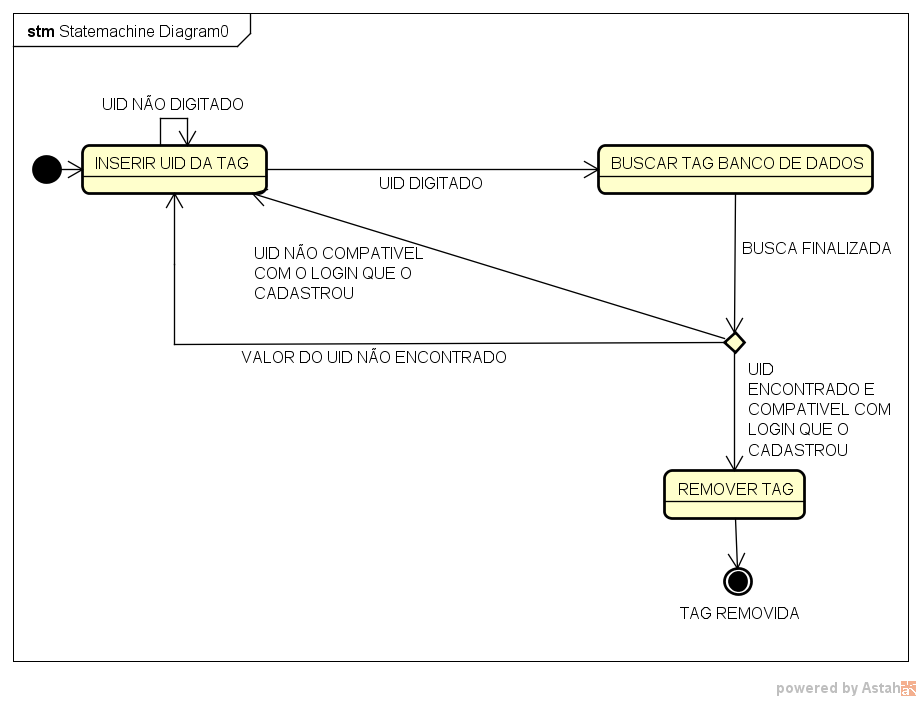
Os diagramas de fluxo, Figuras 11 a 14, demonstram as funcionalidades presentes no sistema desenvolvido:

Figura - Fluxograma validação de *tag*



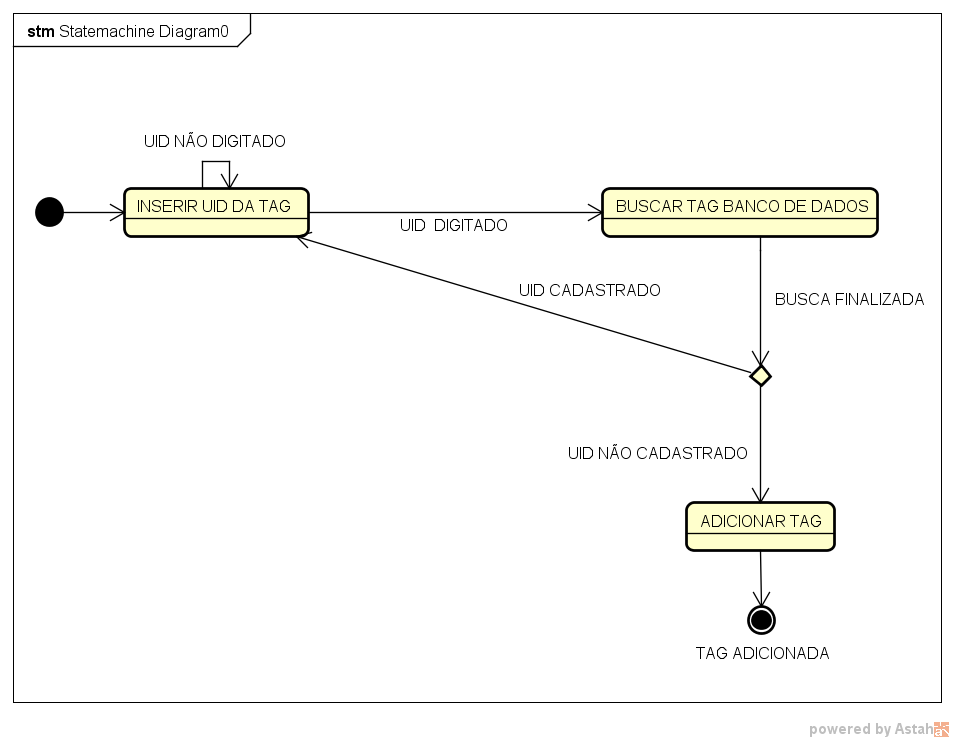
Fonte: Autoria própria, 2020

Figura - Fluxograma remoção de *tag*



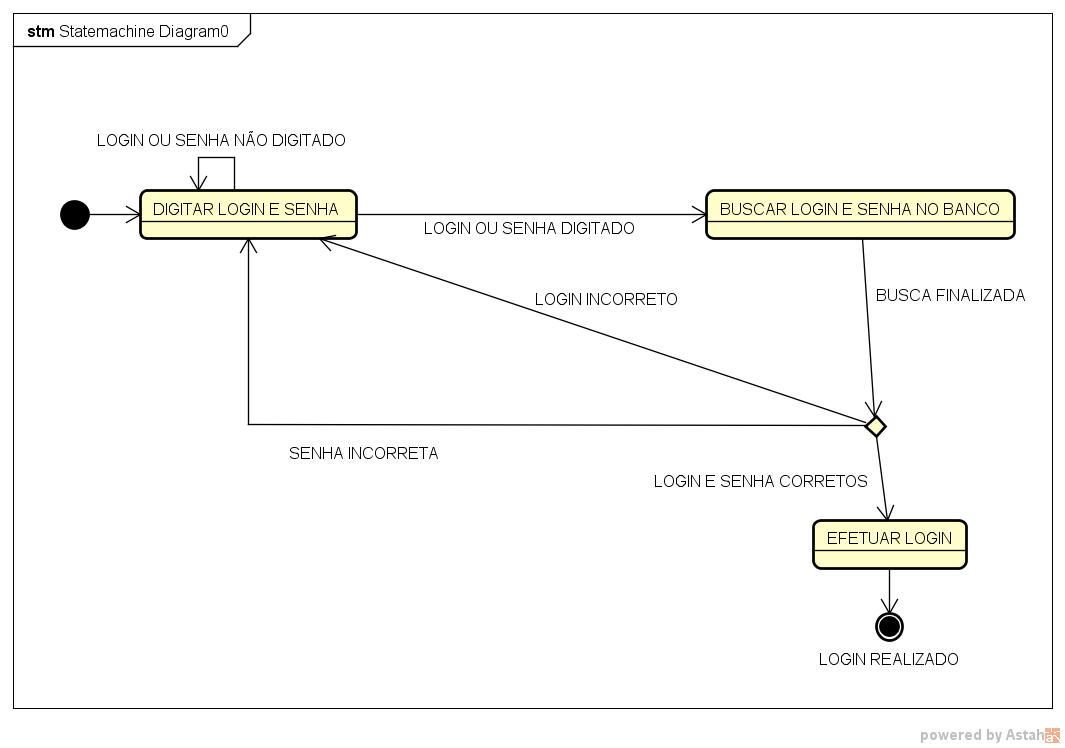
Fonte: Autoria própria, 2020

Figura - Fluxograma adição de *tag*



Fonte: Autoria própria, 2020

Figura - Fluxograma *login* do usuário no sistema



Fonte: Autoria própria, 2020

## Divisão de Tarefas no Sistema

As *interfaces* do sistema de usuários foram definidas que seriam *Web*, aproveitando o dispositivo de conexão Wi-Fi presente no ESP32 e celulares. O objetivo dessas *interfaces* é que o usuário desse sistema possa acessá-lo através do próprio aplicativo. O servidor, por sua vez, deve receber essas informações do usuário e escrevê-las ou consultá-las no banco de dados.

Visando atender esse objetivo, nas configurações do lado do servidor utilizando PHP foram criados os seguintes arquivos de execução:

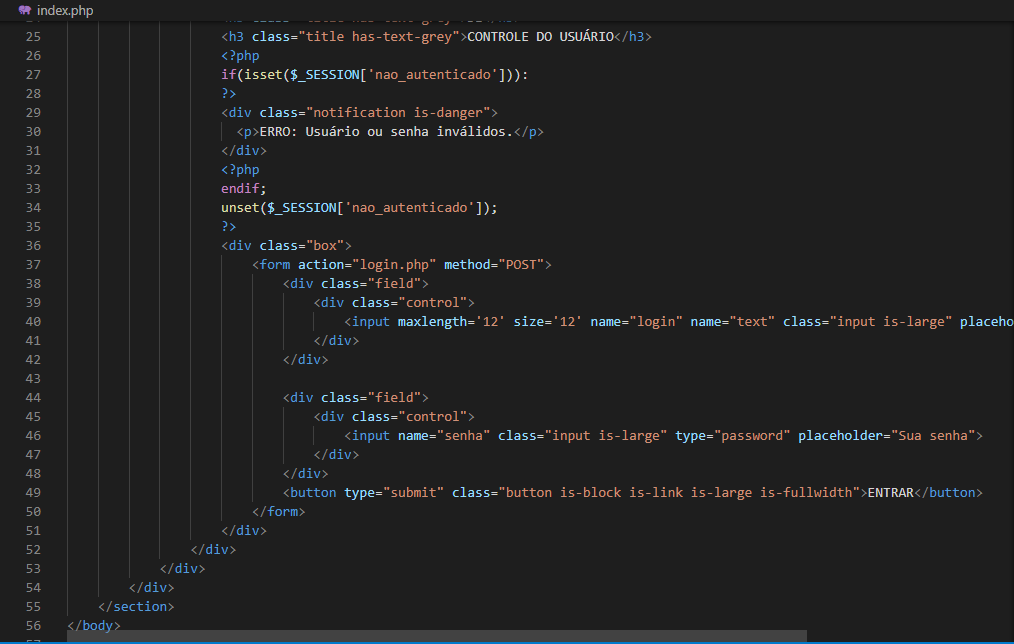
* **/index.php –** Corresponde a página inicial de *login*, o usuário irá inserir seus *login* e senha;
* **/login.php –** Corresponde a função complementar do arquivo index.php, que irá fazer a pesquisa ao banco de dados e validar *login* e senha digitados;
* **/conexao.php –** Corresponde a conexão da aplicação com o banco de dados nos arquivos adicionarTag.php e removerTag.php;
* **/conexao2.php –** Corresponde a conexão da aplicação com o banco de dados nos arquivos *login*.php e validacao.php;
* **/validaçao.php –** Corresponde a parte que irá tratar a requisição do ESP32.
* **/painel.php –** Corresponde ao menu principal da aplicação que é escolhido a opção de adicionar ou remover *tag*;
* **/adicionarTag.php –** Corresponde a página onde será digitado a *tag* que deseja adicionar;
* **/removerTag.php –** Corresponde a página que será digitado a *tag* que se deseja remover
* **/logout.php –** Corresponde a página responsável por encerrar a sessão do usuário.

O caminho raiz do servidor configurado é correspondente ao endereço IP 192.168.1.3 atribuído ao computador utilizado na rede após a sua conexão na rede. Por exemplo http://192.168.1.3/index.php, iria direcionar para o caminho “/index.php” e a página de *login* seria exibida.

### Login de usuário

Nessa página o usuário deverá preencher aos campos de *login* e senha. Utilizou-se um formulário na criação da página. A Figura 15 demonstra o código do arquivo index.php.

Figura - Formulário em PHP do arquivo index.php



Fonte: Autoria própria, 2020

Assim que é feita a tentativa de *login* o arquivo index.php chama o arquivo *login*.php que irá receber os dados inseridos e liberará ou não o acesso. A Figura 16 demonstra o código do arquivo *login*.php.

Figura - Formulário em PHP do arquivo *login*.php



Fonte: Autoria própria, 2020

Caso os dados não estejam corretos será mostrado a frase “ERRO: Usuário ou senha inválidos.”. Esses dados são verificados na tabela clientes. A Figura 17 mostra a página de *login* com erro.

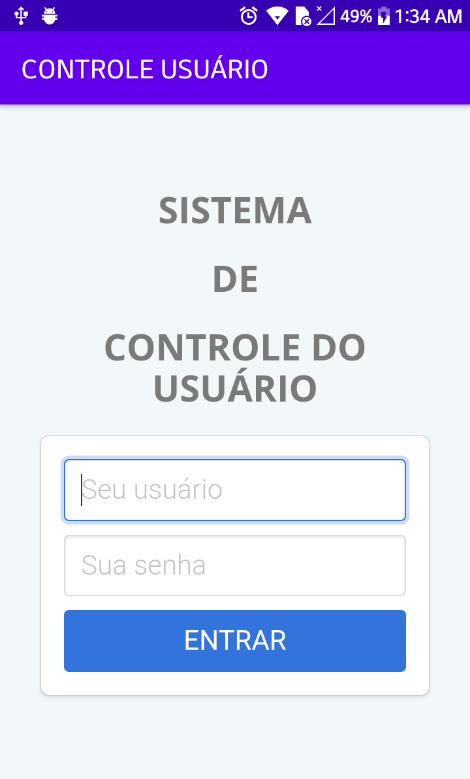
Figura - Página de *login* aplicativo com erro



Fonte: Autoria própria, 2020

A Figura 18 mostra a página de *login* no aplicativo Android.

Figura - Página de *login* aplicativo



Fonte: Autoria própria, 2020

### Painel do usuário

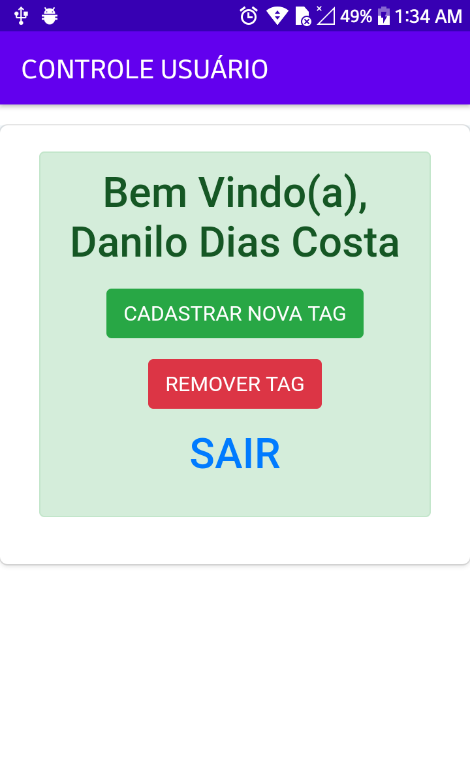
Nessa página o usuário deverá escolher qual função do sistema deseja realizar. Se deseja adicionar ou remover uma *tag* ou até mesmo sair. A Figura 19 mostra o código do arquivo painel. A Figura 21 demonstra o código do arquivo painel.php e a Figura 20 mostra a página de *login* no aplicativo Android.

Figura - Painel do usuário em PHP do arquivo painel.php



Fonte: Autoria própria, 2020

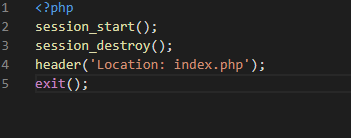
Figura - Painel do usuário do aplicativo



Fonte: Autoria própria, 2020

A Figura 21 mostra o arquivo logout.php que é responsável por encerrar a sessão do usuário, através da função “sesssion\_destroy()” e automaticamente já será redirecionado para o arquivo index.php. A Figura 21 demonstra o código do arquivo logout.php.

Figura - Código PHP do arquivo logout.php

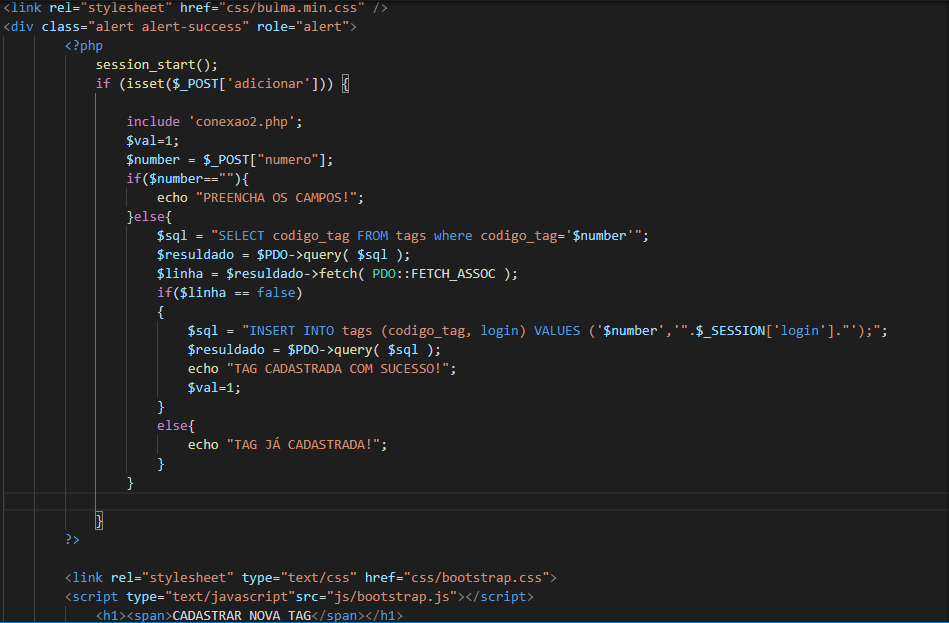


Fonte: Autoria própria, 2020

### Adicionar *tag*

Nessa página o usuário deverá digitar o UID da *tag* que deseja adicionar ou pode voltar para o painel do usuário caso tenha entrado na página por engano. Algumas validações são realizadas aqui para garantir a integridade ao banco de dados e informações alteradas nele. As validações são: se o campo foi preenchido e se o UID já não está cadastrado no banco na tabela *tags*. Caso o valor digitado obedeça a regra o valor de UID digitado será armazenado ao banco de dados. A Figura 22 demonstra o código do arquivo adicionarTag.php.

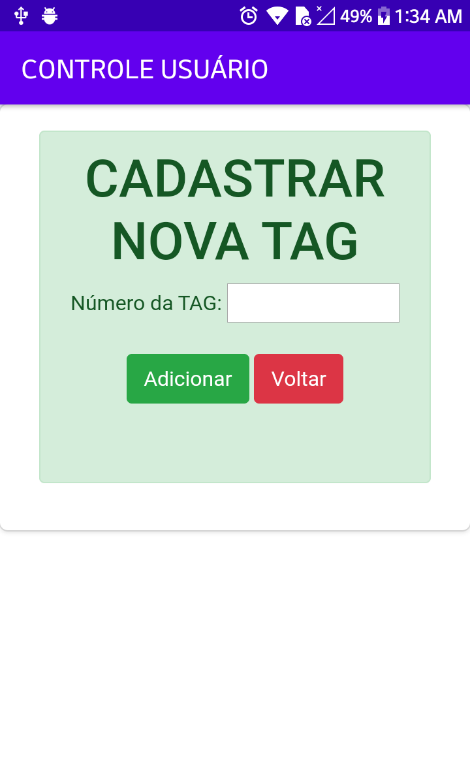
Figura - Cadastro de *tag* PHP arquivo adicionarTag.php



Fonte: Autoria própria, 2020

Figura 23 mostra a página de cadastro de *tag* no aplicativo Android.

Figura - Página cadastrar *tag* do aplicativo



Fonte: Autoria própria, 2020

A primeira verificação é se o campo foi preenchido. Caso não tenha sido será mostrado a seguinte mensagem na tela: “PREENCHA OS CAMPOS!”. Conforme mostra a Figura 24.

Figura - Mensagem de erro campo não preenchido



Fonte: Autoria própria, 2020

A segunda verificação, é se o UID já não está cadastrado no banco. Caso não tenha sido será mostrado a seguinte mensagem na tela: “TAG JÁ CADASTRADA!”. Conforme pode ser observado na Figura 25.

Figura - Mensagem de erro de *tag* já cadastrada



Fonte: Autoria própria, 2020

Caso esteja tudo certo, a *tag* será cadastrada no banco de dados, exibindo a seguinte mensagem: “TAG CADASTRADA COM SUCESSO!”. Conforme pode ser observado na Figura 26.

Figura - Mensagem *tag* cadastrada

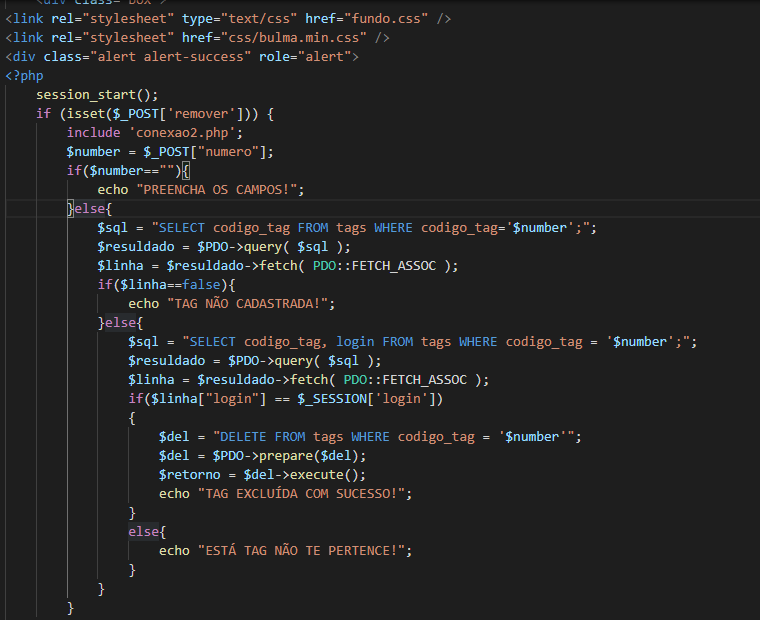


Fonte: Autoria própria, 2020

### Remover *tag*

Nessa página o usuário deverá digitar o UID da *tag* que deseja remover ou pode voltar para o painel do usuário caso tenha entrado na página por engano. Algumas validações são realizadas aqui para garantir a integridade ao banco de dados e informações alteradas nele. As validações são: se o campo foi preenchido, se o UID existe no banco na tabela *tags* e se o mesmo *login* que cadastrou é o que está removendo está *tag*. Caso o valor digitado obedeça a regra o valor de UID digitado será removido do banco de dados. A Figura 27 demonstra o código do arquivo removerTag.php.

Figura - Remoção de *tag* PHP arquivo removerTag.php



Fonte: Autoria própria, 2020

Figura 28 mostra a página de cadastro de *tag* no aplicativo Android.

Figura - Página remover *tag* do aplicativo



Fonte: Autoria própria, 2020

A primeira verificação é se se o campo foi preenchido. Caso não tenha sido será mostrado a seguinte mensagem na tela: “PREENCHA OS CAMPOS!”. Conforme pode ser observado na Figura 29.

Figura - Mensagem de erro campo não preenchido



Fonte: Autoria própria, 2020

A segura verificação é se o UID está cadastrado no banco. Caso não tenha sido será mostrado a seguinte mensagem na tela: “TAG NÃO CADASTRADA!”. Conforme pode ser observado na Figura 30.

Figura - Mensagem de erro de *tag* não existente



Fonte: Autoria própria, 2020

A terceira verificação é se o mesmo *login* que cadastrou é o que está removendo a *tag*. Caso não tenha sido será mostrado a seguinte mensagem na tela: “ESTÁ TAG NÃO TE PERTENCE!”. Conforme pode ser observado na Figura 31.

Figura - Mensagem de erro de usuário incorreto



Fonte: Autoria própria, 2020

Caso esteja tudo certo, a *tag* será excluída no banco de dados, exibindo a seguinte mensagem: “TAG EXCLUÍDA COM SUCESSO!”. Conforme pode ser observado na Figura 32.

Figura - Mensagem *tag* removida



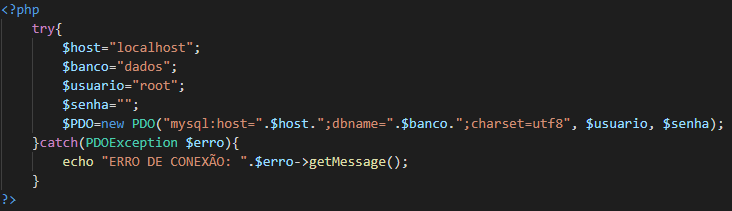
Fonte: Autoria própria, 2020

### Conexão com Banco de Dados

A conexão com o banco de dados é feita com os dois arquivos conexao.php e conexao2.php. Os dois arquivos funcionam de maneira parecida, ambos fazem a conexão com o banco de dados dando suporte em algumas partes, o que difere uma da outra é a função e a forma como é feita.

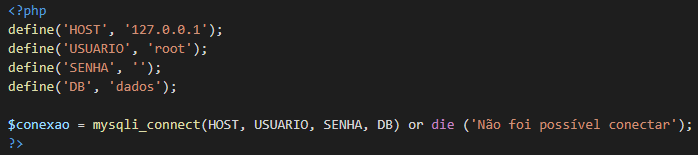
No arquivo conexao.php usa-se uma função chama mysqli\_connect que é mais antiga no PHP e indicada para buscas mais simples e leves, por isso ela é utilizada na parte de *login*. No arquivo conexao2.php é utilizado uma função chamada PDO (*PHP Data Objects*) que é mais recente na linguagem PHP e, geralmente, é usada em buscas mais complexas e pesadas oferecendo uma melhor performance e resposta, por isso foi utilizada em várias partes da aplicação. Ambas têm os mesmos parâmetros de dados sendo eles: *host* do servidor, nome do banco de dados, usuário e senha de acesso ao banco de dados. A Figura 33 demonstra o código do arquivo conexao2.php e a Figura 34 mostra o código do arquivo conexao.php.

Figura - Método de conexão conexao2.php



Fonte: Autoria própria, 2020

Figura - Método de conexão conexao.php



Fonte: Autoria própria, 2020

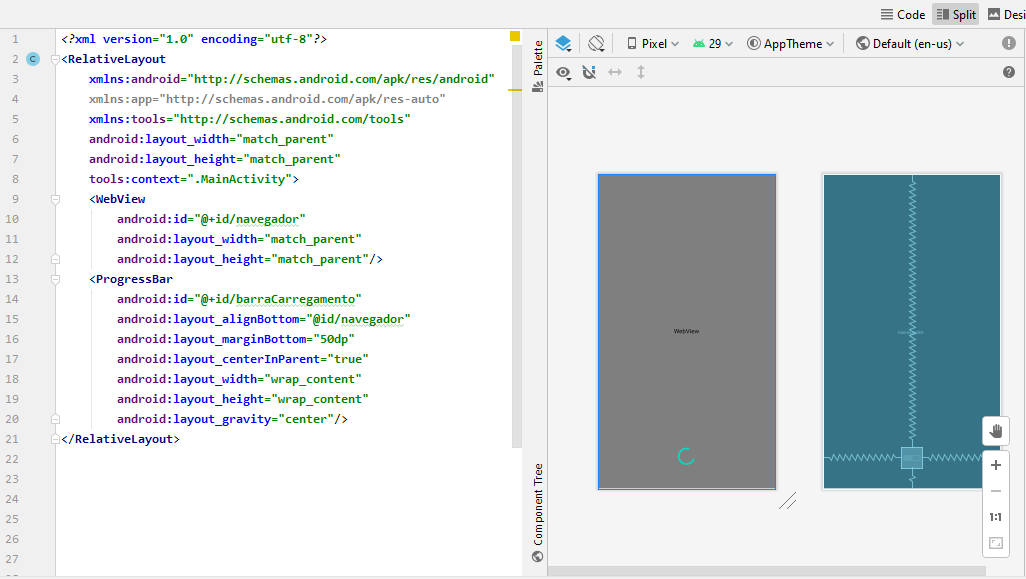
### Aplicativo Android

O aplicativo foi desenvolvido inicialmente em Android por ser o sistema operacional *mobile* mais utilizado atualmente, facilitando assim um alcance maior de possíveis usuários. Com o aplicativo espera-se dar uma maior facilidade ao usuário na hora de gerenciar suas *tags*.

Esse aplicativo consiste na utilização de uma WebView que é um componente com tecnologia do Google Chrome que permite abrir *links* no aplicativo desenvolvido sem sair dele, ou seja, permite uma simulação de aplicativo, mas utilizando um navegador nativo do sistema operacional Android. É definido a qual endereço a WebView deve acessar, se está habilitado o *zoom* e javascript para a página que deseja acessar.

A primeira parte consiste na criação da tela utilizando o XML, que é criado uma barra de progresso e a WebView e seus respectivos tamanhos na tela. A Figura 35 demonstra o código em XML desenvolvido no Android Studio.

Figura - Arquivo XML página Android



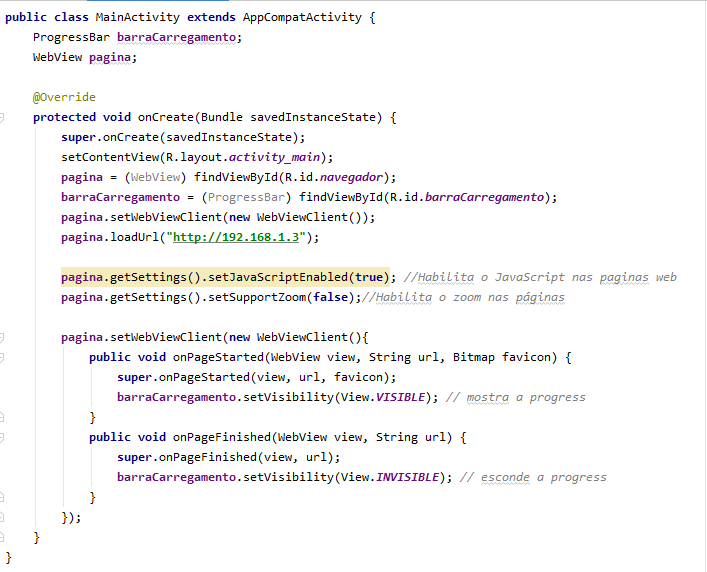
Fonte: Autoria própria, 2020

Após a definição da aparência do aplicativo, precisa-se fazer a ligação da parte da tela, com a parte do código que realizará as operações, e para isso utiliza-se então a função findViewById. Após isso é preciso criar uma nova instancia para que a WebView seja capaz de gerenciar tudo e não dependa de aplicativos externos. É utilizado para isso a função setWebViewClient o qual passa-se como parâmetro um *new* WebViewClient().

Com essa primeira parte feita precisa-se então definir qual é o endereço a ser carregado, aqui utiliza-se a função loadUrl, que passa-se como parâmetro o endereço. Na próxima etapa tem-se que definir sobre se é possível a utilização de *zoom* e de javascript pela WebView aqui é utilizado as funções setJavaScriptEnabled e setSupportZoom passando como parâmetro *true* ou *false*, verdadeiro ou falso. Verdadeiro para habilitar e falso para desabilitar.

Ao final tem-se as configurações da barra de progresso que aparecerá se o celular espera uma resposta do servidor. A Figura 36 demonstra o código no Android Studio que foi desenvolvido.

Figura - Código WebView do aplicativo



Fonte: Autoria própria, 2020

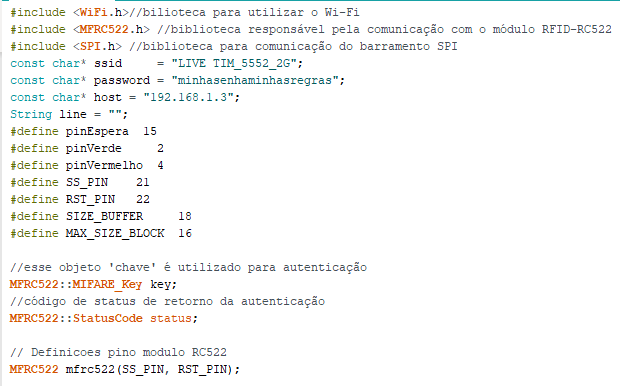
### ESP32 Verificação de *tag*

Na operação de liberação ou não de acesso, escolheu-se o ESP32 pelo poder de processamento e seus vários suportes como Wi-Fi e Bluetooth. A primeira parte no desenvolvimento, é a definição de quais bibliotecas usar, no caso utilizou-se:

* **WiFi.h:** Bilioteca para utilizada para as funções do Wi-Fi.
* **MFRC522.h:** Biblioteca responsável pela comunicação com o módulo RFID-RC522.
* **SPI.h:** Biblioteca para comunicação do barramento SPI.

Após a definição das bibliotecas é preciso definir três constantes, nome da rede, senha que o ESP32 irá se conectar e o *host* do servidor. Com isso definido passa-se então para a definição de portas a serem utilizadas, finalizando com a definição da *key* e status do RFID e a definição dos módulos dos pinos para o leitor de etiquetas usando o SPI. A figura 37 demonstra a parte do código no Arduino IDE que foi discutido.

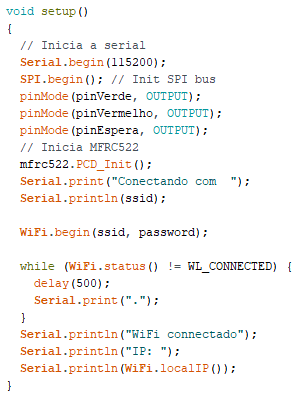
Figura - Definição das informações no ESP32



Fonte: Autoria própria, 2020

Na próxima fase, dentro do void setup() define-se taxa de transferência de *bits*, aqui utilizou-se 115200 *bits* de taxa de transmissão, inicia-se o protocolo SPI, e define-se também as saídas de pinagem para os *LEDs* (*Light-emitting Diode*) usando a função pinMode(), que recebe como parâmetro a porta e se ela é saída (*OUTPUT*) ou entrada (*INPUT*). Com isso inicia-se a conexão Wi-Fi usando WiFi.begin() que possui como parâmetro o nome e senha da rede que o ESP32 conectará. A figura 38 demonstra a parte do código no Arduino IDE que foi discutido.

Figura - Definição void setup( ) para o ESP32



Fonte: Autoria própria, 2020

Nas definições do void loop() tem-se duas funções principais, PICC\_IsNewCardPresent() e PICC\_ReadCardSerial(), que são responsáveis por realizar a leitura quando uma *tag* é aproximada do leitor. Aqui é feito também a definição do http através da função connect() que tem como parâmetros o *host* do servidor e a porta, no caso a 80 por padrão no HTTP. Com essas definições prontas tem-se então disponível a parte de comunicação com o servidor usando client.avaliable() o qual irá abrir a conexão e enviar as requisições e fica-se a client.readStringUntil() responsável por receber as respostas do servidor que são armazenadas na variável line.

Uma parte bastante interessante a qual gerou muitos problemas até ser descoberta é que a cada reposta que o servidor realiza, ele automaticamente salta a linha dando um “\n”. Após mandar a solicitação para o servidor verificar no banco de dados sobre a UID da *tag* o servidor irá retornar *true* (Verdadeiro) ou *false* (Falso). Essa parte será explicada mais adiante. A figura 39 demonstra a parte do código no Arduino IDE que foi discutido.

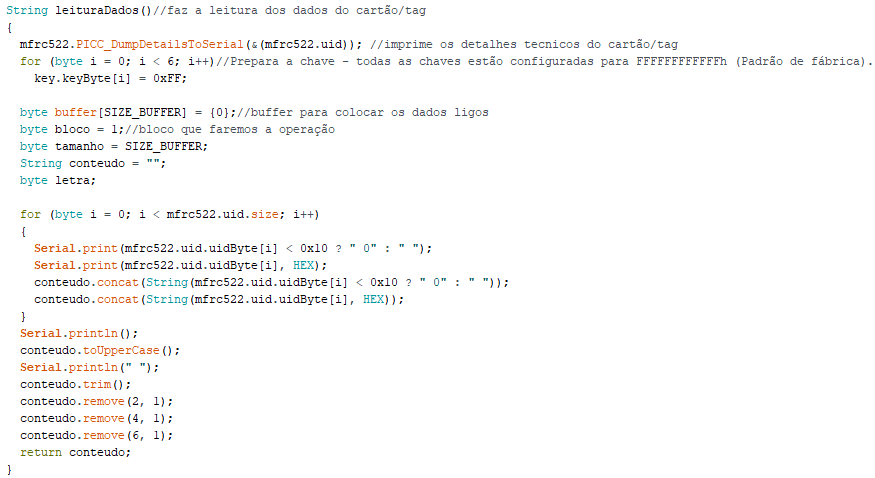
Figura - Definição void loop ( ) para o ESP32



Fonte: Autoria própria, 2020

Com a aproximação e leitura de uma *tag*, e após todos os passos realizados, será chamado a função leituraDados() a qual realiza a leitura do UID da *tag*. Como esse UID não retorna uma string limpa, torna-se necessário realizar alguns tratamentos, como retirada de espaços usando trim() e remove() para assim retornar o texto. A figura 40 demonstra a parte do código no Arduino IDE que representa a função leituraDados.

Figura - Função leituraDados()

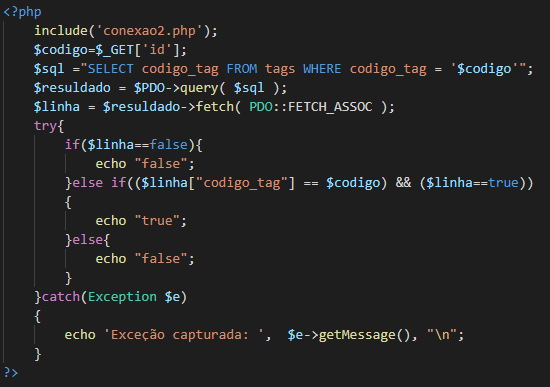


Fonte: Autoria própria, 2020

Após a conclusão dessa etapa, faz-se então uma concatenação com um URL (*Uniform Resource Locator*) que acionará no servidor o arquivo validacao.php, aqui ele recebe o valor do UID da *tag* de um método no PHP chamado GET que recebe o valor atribuído via URL, como por exemplo: **/validacao.php?id= + o valor que deseja-se enviar**. Com este valor, será feita então uma busca no banco de dados na tabela *tags*, e a resposta do servidor consiste em uma mensagem echo do PHP de *true* ou *false*.

Caso o retorno em mãos se for *true* irá acender o *LEDs* verde, manter apagado o vermelho e apagar o amarelo, caso *false* será aceso o LED vermelho, manter apagado o verde e apagar o amarelo por 3 segundos em ambas as situações. Somente após 1 minuto poderá ser tentado outra aproximação de *tag*, caso o sistema não tenha nenhuma resposta permanecem apagados os *LEDs* vermelho e verde e o amarelo aceso. A Figura 41 demonstra o código do arquivo validacao.php

Figura - Arquivo validacao.php

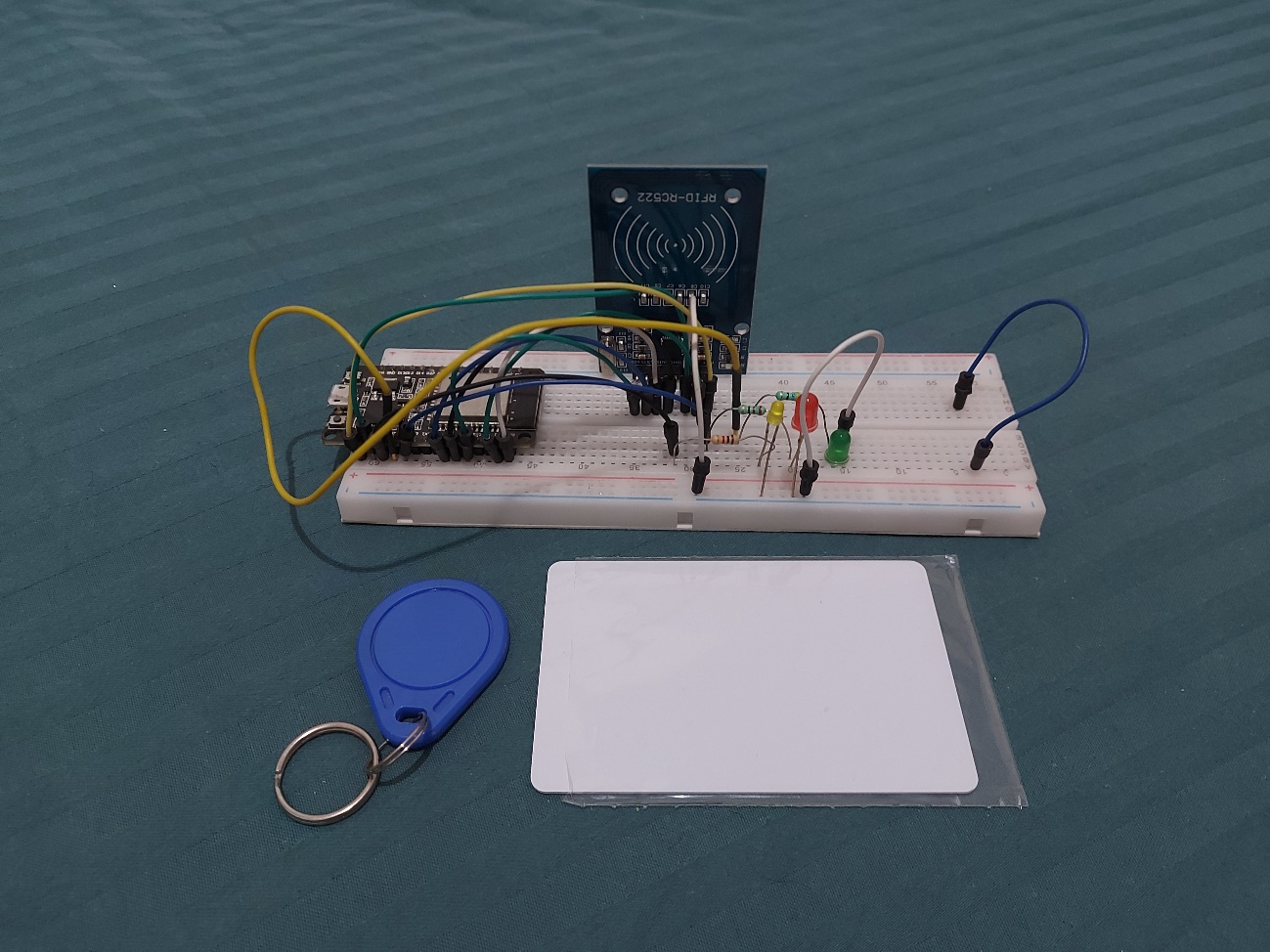


Fonte: Autoria própria, 2020

# 4. RESULTADOS

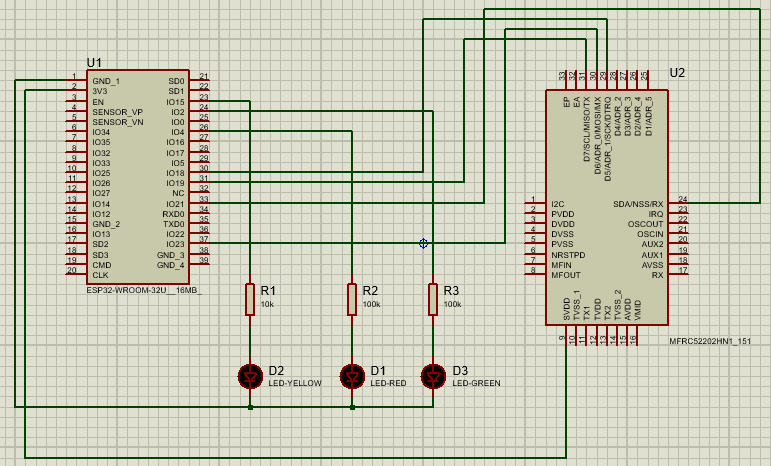
Neste capítulo são apresentados os resultados deste projeto, conforme discutido nos capítulos anteriores. A Figura 42 mostra o circuito final montado. E a pinagem de conexão entre os componentes na Figura 43.

Figura - Circuito montado



Fonte: Autoria própria, 2020

Figura - Diagrama de conexão e pinagem entre componentes



Fonte: Autoria própria, 2020

Os componentes são ligados da seguinte forma:

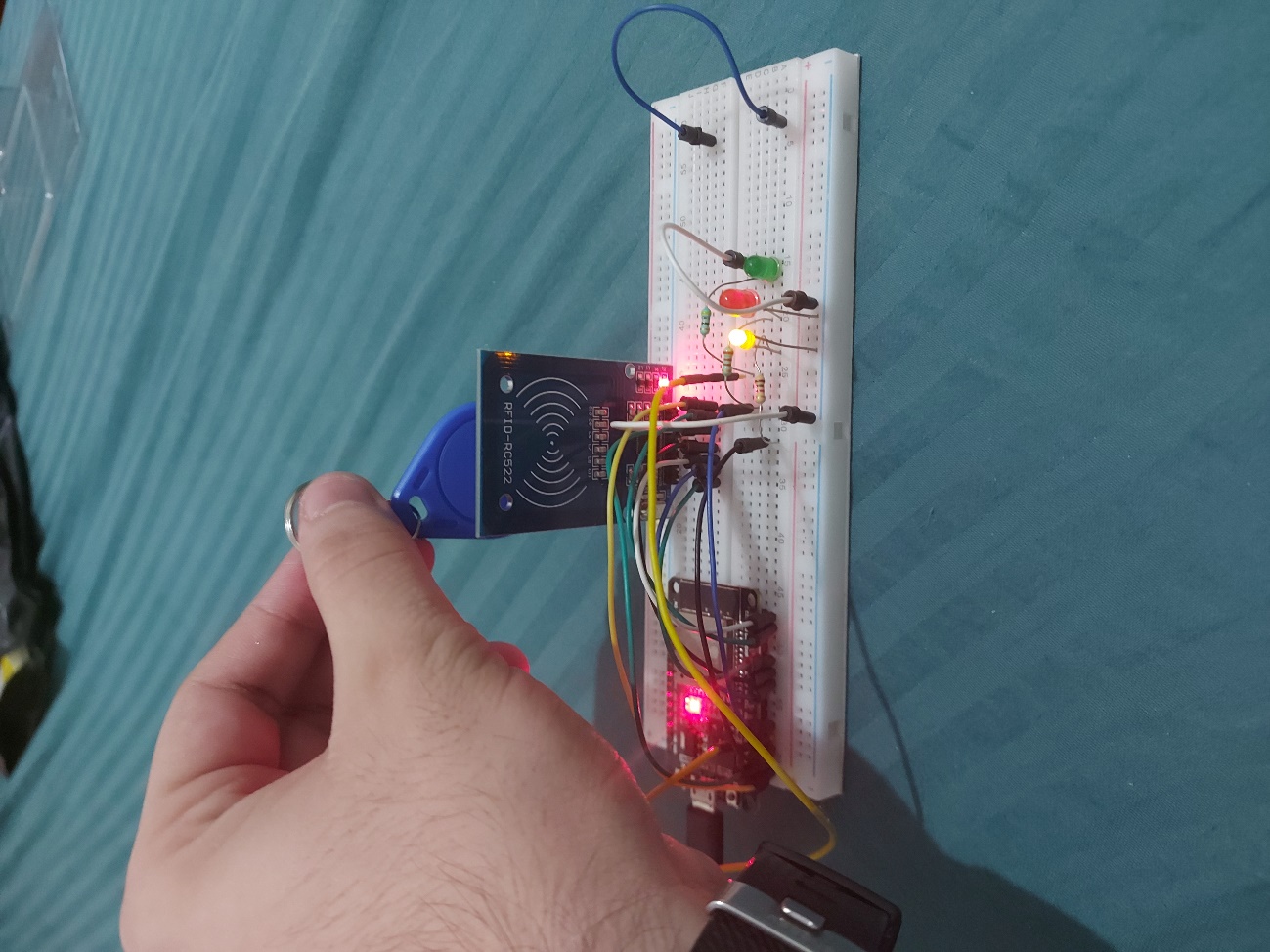
Tabela 3 - Tabela de pinagem

|  |  |
| --- | --- |
| **ESP32 (PINO)** | **RC522(PORTA)** |
| 18 | SCK |
| 19 | MISO |
| 21 | SDA |
| 22 | RST |
| 23 | MOSI |
| GND | GND |
| 3v3 | 3v3 |
| **LEDS** | |
| 15 | LED Amarelo |
| 2 | LED Verde |
| 4 | LED Vermelho |

Na etapa de projeto, foi definido que o ESP32 receberia a resposta do servidor *web*, pois o local de validação por completo da UID seria feita no mesmo, já que o microcontrolador só é responsável pela leitura do UID da *tag* e envia para o servidor *web* o qual faz a verificação no banco de dados e devolve a permissão de sim ou não para a autenticação.

A leitura será realizada com a aproximação da *tag* ao leitor como na Figura 44.

Figura - Aproximação da *tag* no leitor RFID

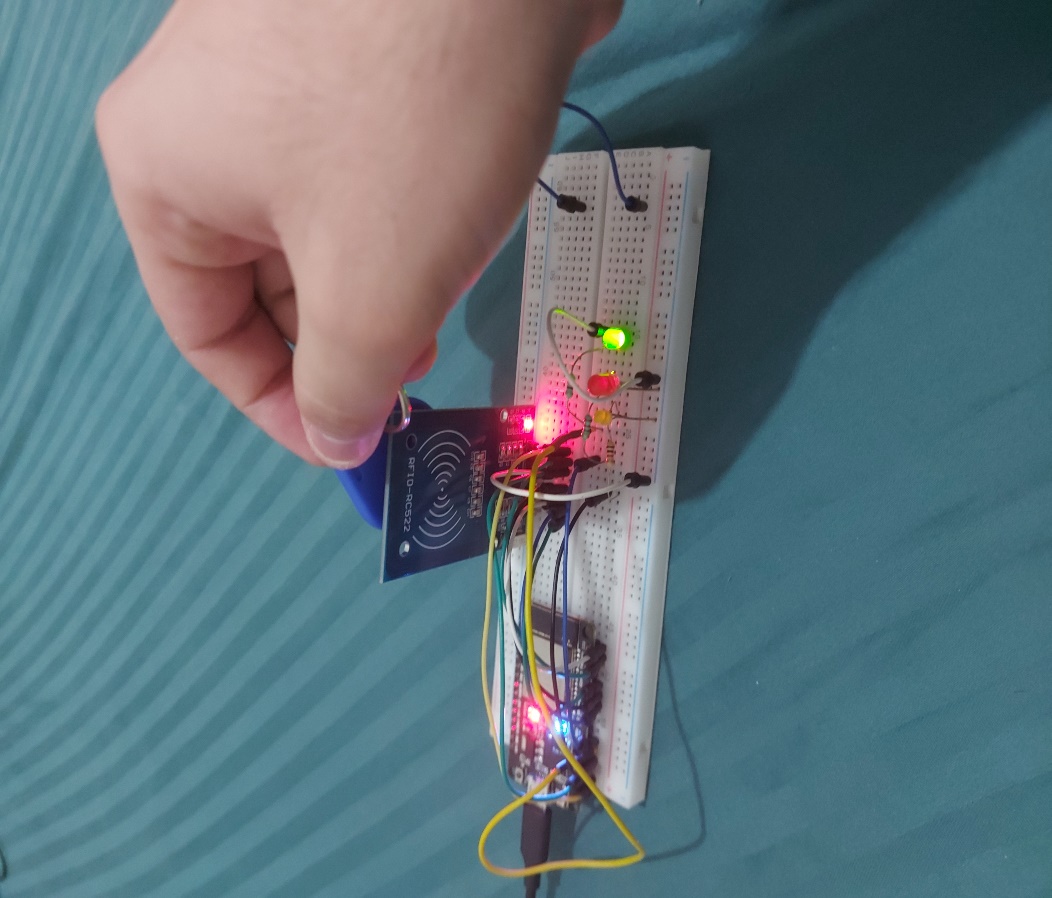


Fonte: Autoria própria, 2020

Após essa aproximação o ESP32 irá se conectar com o servidor e devolver se é permitido ou não o acesso.

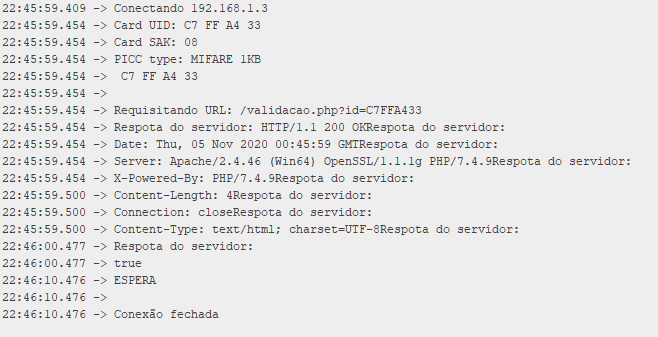
Com a aproximação de uma etiqueta válida, ou seja, cadastrada no banco de dados, o sistema irá acender o LED verde como mostrado na Figura 45, e o microcontrolador terá uma resposta do servidor dizendo *true*, como na Figura 46 usando o monitor serial do Arduino IDE, após isso para realizar a leitura de outra etiqueta será necessário um tempo de espera de 1 minuto. Esse tempo pode ser configurado a critério da aplicação.

Figura - *Tag* cadastrada no banco de dados



Fonte: Autoria própria, 2020

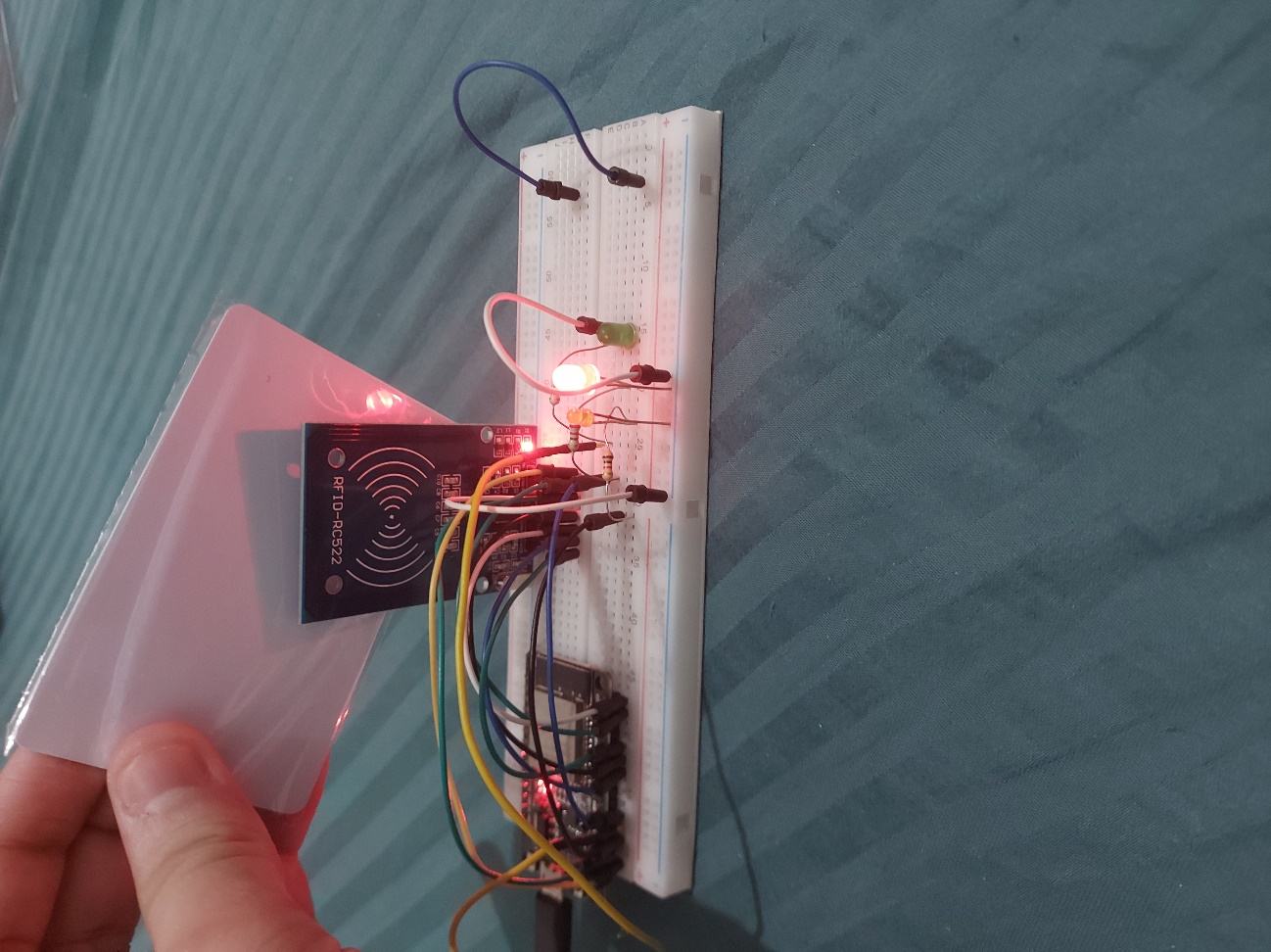
Figura - Reposta do servidor *tag* cadastrada



Fonte: Autoria própria, 2020

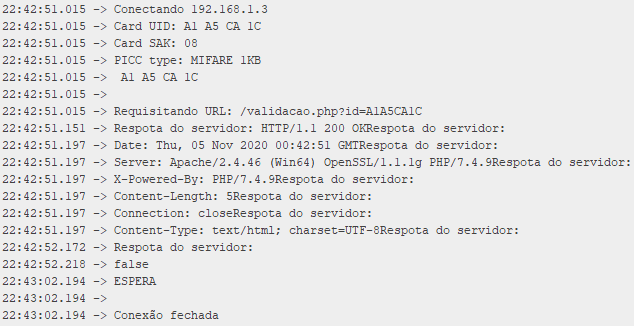
Com a aproximação de uma etiqueta não válida, ou seja, não cadastrada no banco de dados, o sistema irá acender o LEDvermelho como mostrado na Figura 47, e o microcontrolador terá uma resposta do servidor dizendo *false*, como na Figura 48 usando o monitor serial do Arduino IDE, após isso para realizar a leitura de outra etiqueta terá um tempo de espera de 1 minuto.

Figura - *Tag* cadastrada no banco de dados



Fonte: Autoria própria, 2020

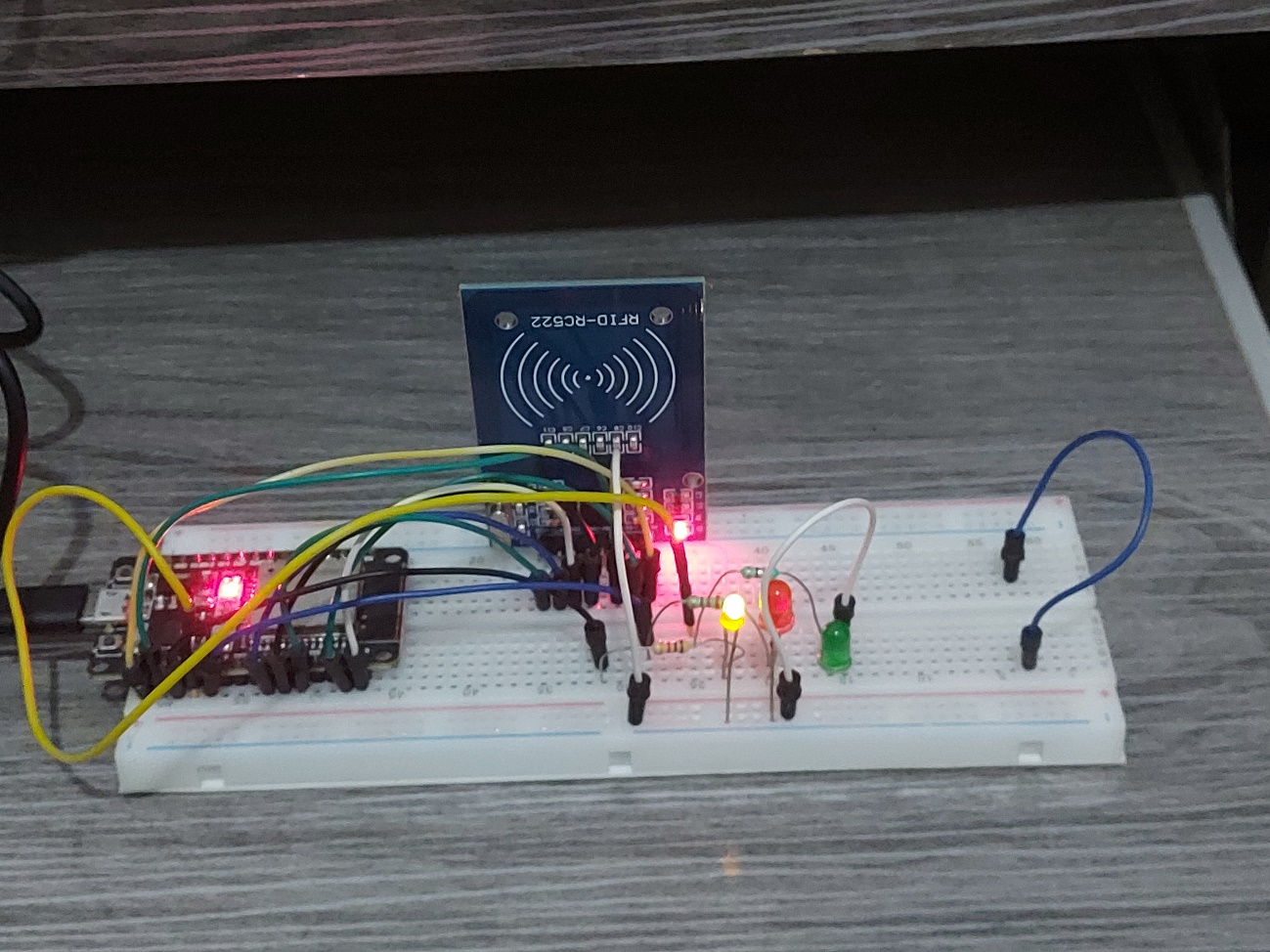
Figura - Reposta do servidor *tag* não cadastrada



Fonte: Autoria própria, 2020

Caso não tenha nenhuma leitura de uma *tag*, o sistema irá manter o LEDamarelo acesso e o servidor junto ao ESP32 ficará aguardando a aproximação de uma *tag* como mostrado na Figura 49.

Figura - Aguardando aproximação de uma *tag*



Fonte: Autoria própria, 2020

# CONCLUSÃO

O projeto conseguiu atender seu objetivo principal, ao realizar autenticação dos usuários de forma correta, tanto via *login* no site para gerenciamento das *tags*, quanto ao apresentar a *tag* no sensor para a liberação ou não do acesso. A parte de remoção e adição das *tags* também correu da forma certa.

Nesse projeto foram colocados em prática e aprimorados os conceitos de várias disciplinas. Isso possibilitou um grande aprendizado já que foi possível ver na prática e aprimorar os conhecimentos e experimentos obtidos e realizados em sala de aula.

Alguns problemas surgiram ao longo do desenvolvimento desse trabalho. O que precisou levar o trabalho a um novo patamar, porém utilizando características e fundamentos do trabalho inicial. Com isso alguns novos conhecimentos tiveram que ser obtidos como a programação na linguagem PHP.

Um grande problema foi o encaixe e utilização do protocolo SPI para conectar o ESP32 com o RFID. Como maior desafio nesse trabalho teve-se encontrar boas referências bibliográficas para auxilio de como realizar esse projeto, por algumas informações serem restritas a indústria no geral. Algumas funções dentro dessa aplicação precisaram ser desenvolvidas do zero.

Os resultados foram bem satisfatórios já que atenderam aos objetivos propostos e obteve as funcionalidades esperadas. Cada um dos componentes trabalhados nesse trabalho, apesar das dificuldades conseguiu-se sua atender ao esperado.

## Trabalhos Futuros

### Complementação da solução

Essa solução ainda pode ter várias complementações, como por exemplo:

* *Interface* para cadastro de novos usuários;
* Servidor *Web* hospedado externamente em um servidor próprio para maior escalabilidade e armazenamento de dados;
* Armazenamento de dados em servidor externo de banco de dados;
* Criptografia para guardar, cadastrar e autenticar os usuários;
* Utilização de rede cabeada;
* Geração de relatórios de cadastro/remoção de tags e acesso de usuários;
* Criação de aplicativo para IOS.

### Implementações Similares

Esse mesmo projeto, com pequenas mudanças de adaptação para um contexto diferente, pode ser utilizado em outras aplicações. Como:

* Controle de armários de academias e/ou albergues, escola, supermercados etc. para seus usuários;
* Automatização de empréstimos de livros em uma biblioteca;
* Automatização de compras em supermercados;
* Controle de uso para equipamentos compartilhados;

Controle de rebanhos em fazendas.

# REFERÊNCIAS

CIRIACO, Douglas. **Como funciona a RFID?**. Tec Mundo, 17 ago. 2009. Disponível em: https://www.tecmundo.com.br/tendencias/2601-como-funciona-a-rfid-.htm. Acesso em: 10 maio 2020.

DEITEL, Paul; DEITEL, Harvey; DEITEL, Abbey. **Android como programar: com introdução a Java**. São Paulo: Bookman, 2015.

DEITEL, Paul; DEITEL, Harvey; DEITEL, Abbey. **Android: Como Programar**. 2. ed. São Paulo: Bookman, 2015.

Dwight, Robert. **PHP: Learn PHP in 24 Hours or Less - A Beginner’s Guide To Learning PHP Programming Now (PHP, PHP Programming, PHP Course**. 1. ed. CreateSpace Independent Publishing Platform, 2016.

HCC Documentation. **HCC Embedded**. Disponível em: https://doc.hcc-embedded.com/. Acesso em: 20 abr. 2020.

LEO, Felipe Campos de. **SISTEMA DE CONTROLE DE ACESSO A RECURSOS COMPARTILHADOS**. 2019. 80 f. Monografia (Especialização) - Curso de Engenharia Eletrônica e de Computação, Escola Politécnica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2019.

MARQUARDT, Nicolai; TAYLOR, Alex S.; VILLAR, Nicolas; GREENBERG, Saul. **Rethinking RFID.** **Proceedings Of The 28th International Conference On Human Factors In Computing Systems - Chi '10**, [s.l.], p. 2307-2316, abr. 2010. ACM Press. <http://dx.doi.org/10.1145/1753326.1753674>.

McMahon, David. **Learn html and css with easy to follow-step-by-step tutorials**. 1. ed. Bluewater Publishing LLC, 2015.

OTTO, Mark; THORNTON, Jacob. **Bootstrap Documentation**. Disponível em: https://getbootstrap.com/docs/4.5/getting-started/introduction/. Acesso em: 28 nov. 2019.

SACCO, Francesco. **Comunicação SPI.** Embarcados, 05 maio 2014. Disponível em: https://www.embarcados.com.br/spi-parte-1/. Acesso em: 20 setembro 2020.

SOUSA, Hélio. **SPI e I2C**. Disponível em: https://paginas.fe.up.pt/~hsm/docencia/comp/spi-e-i2c/. Acesso em: 20 setembro 2020.

Spanulescu, Server. **ESP32 Programming for the Internet of Things - Second Edition: HTML, JavaScript, MQTT and WebSockets Solutions (Microcontrollers and IT Book 1)**. 2. ed. CreateSpace Independent Publishing Platform, 2018.

TANENBAUM, David; WETHERALL, J.; S., Andrew. **Redes de Computadores**. 5. ed. São Paulo: Pearson, 2011.

WAZLAWICK, Raul Sidnei. **Metodologia de Pesquisa para Ciência da Computação.** 2. ed. São Paulo: Elsevier, 2014.

WOLF, Marilyn. **Computer as Components**: **principles of embedded computing system design.** 3. ed. Waltham: Elsevier, 2012.

ZANCO, W. S. **Microcontroladores PIC 16F628A/648A: Uma Abordagem Prática e Objetiva.** 1ª ed. São Paulo: Érica, 2005.

# ANEXO A – INSTALAÇÃO XAMPP

**Sistema Operacional Usado:** Windows 10 Pro

**Etapa 1 – Instalação XAMPP**

Acessando o site [**www.apachefriends.org**](http://www.apachefriends.org) apresentado na Figura A.1, a instalação é feita pelo próprio .exe do Xampp por ser mais fácil e rápida.



Figura A.1 – Página para *download* do XAMPP

**Executando arquivo .exe do XAMPP**

Executar o arquivo baixado do XAMPP e clicar em “*Next*” como na Figura A.2

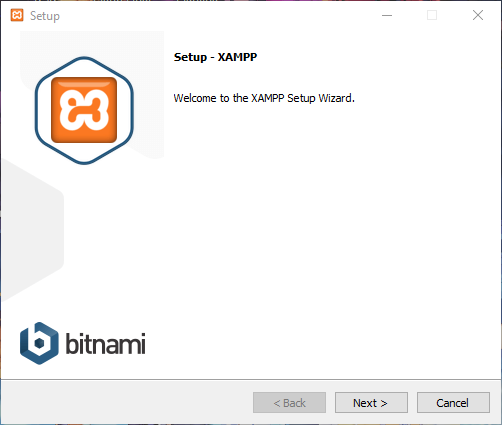


Figura A.2 – Tela inicial instalação XAMPP

**Definindo quais serviços serão utilizados**

Dos conteúdos contidos no XAMPP serão utilizados somente o MySQL, Apache, phpMyAdmin e PHP como mostrado na Figura A.3.

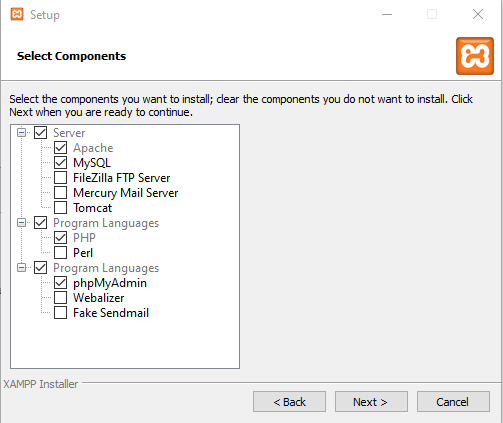


Figura A.3 – Itens a serem instalados do XAMPP

**Definir o diretório que o programa será instalado**

Definir qual pasta será instalado o XAMPP como mostrado na Figura A.4.

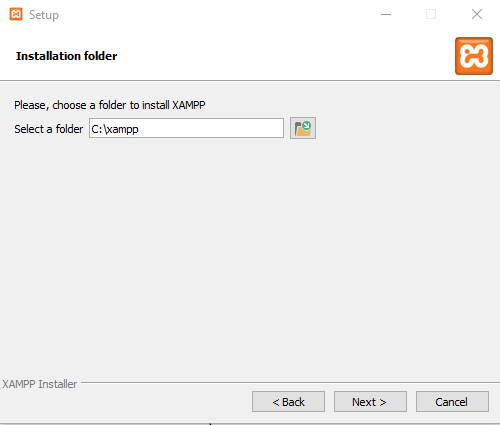


Figura A.4 – Caminho de instalação do XAMPP

**Definindo a linguagem de instalação**

Definir qual linguagem será instalado o XAMPP como mostrado na Figura A.5.

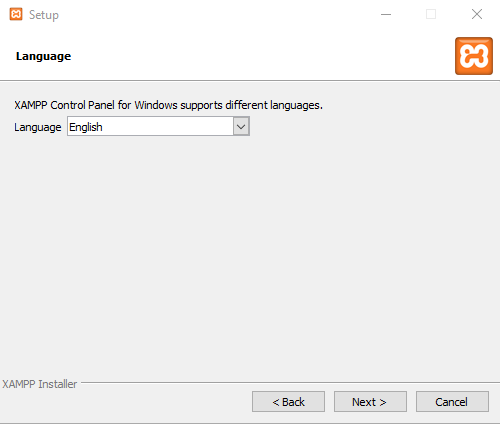


Figura A.5 – Linguagem de instalação do XAMPP

**Instalar o XAMPP**

Após definir as configurações de instalação o XAMPP estará pronto para ser instalado basta clicar em “Next” como mostra a Figura A.6 e aguardar.

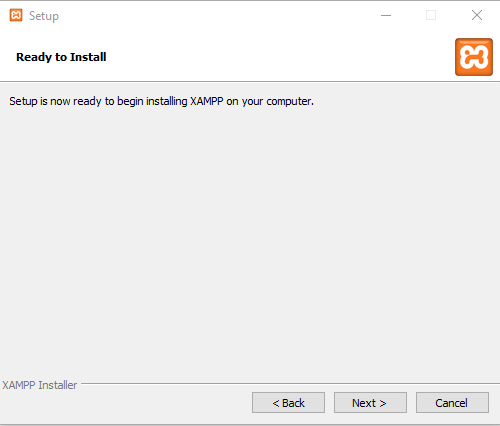


Figura A.5 – Instalação XAMPP

**Etapa 2 – Iniciar servidor Apache e MySQL**

**Abrindo XAMPP**

Após a instalação, basta abrir o XAMPP como na Figura A.6.

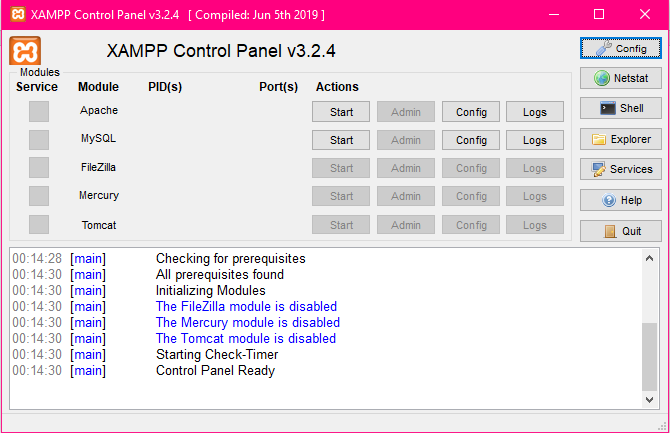


Figura A.6 – Tela inicial XAMPP

**Iniciando servidor Apache**

Basta clicar no botão “*Start*” do Apache e aguardar o início do servidor Apache. Caso funcione, irá aparecer o PID do processo e a porta que está sendo utilizada como na Figura A.7.

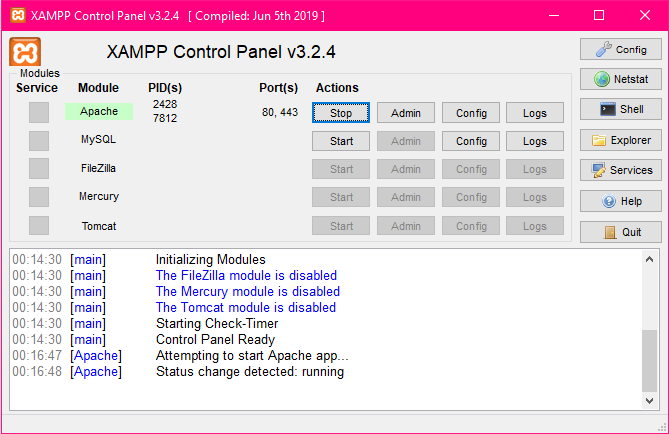


Figura A.7 – Servidor Apache Iniciado

**Iniciando Banco de Dados MySQL**

Basta clicar no botão “*Start*” do MySQL e aguardar o início do mesmo. Caso funcione, irá aparecer o PID do processo e a porta que está sendo utilizada como na Figura A.8.

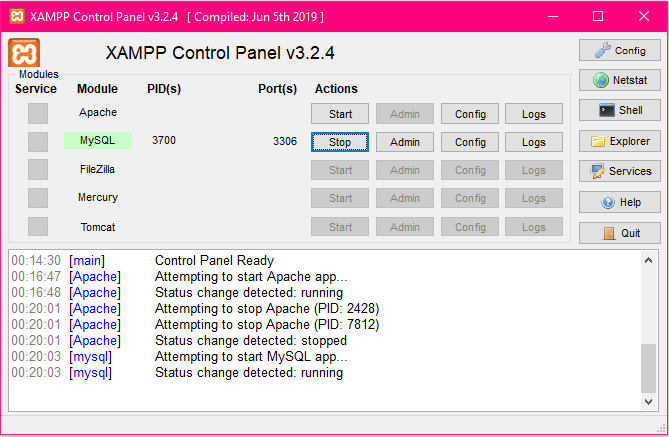


Figura A.8 – Banco de Dados MySQL operacional

**Abrindo configuração do Banco de Dados com phpMyAdmin**

Para acessar o gerenciador do Banco de Dados MySQL é preciso iniciar o servidor Apache e o MySQL como na Figura A.9.

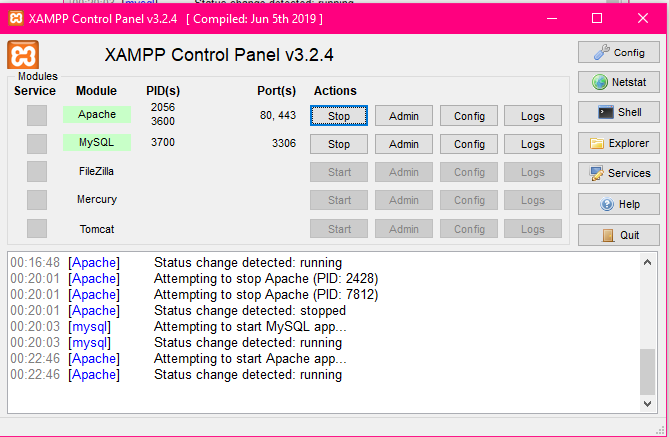


Figura A.9 – Banco de Dados MySQL e servidor Apache operacional

Após o inicio dos dois serviços basta clicar no botão “Admin” como mostrado na Figura A.10.

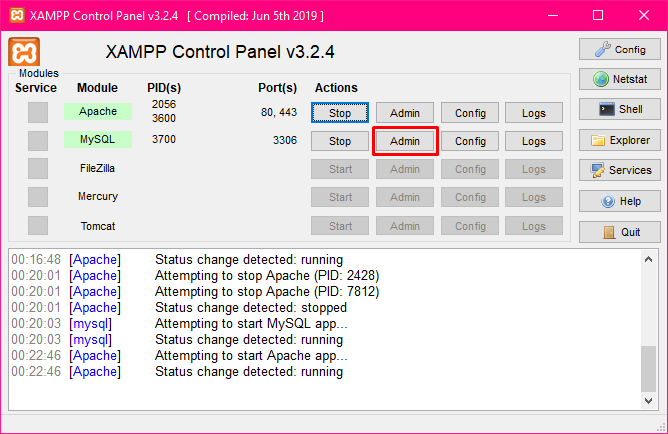


Figura A.10 – Acesso ao phpMyAdmin

Em seguida será aberto no navegador padrão da máquina, o gerenciador do MySQL como na Figura A.11.

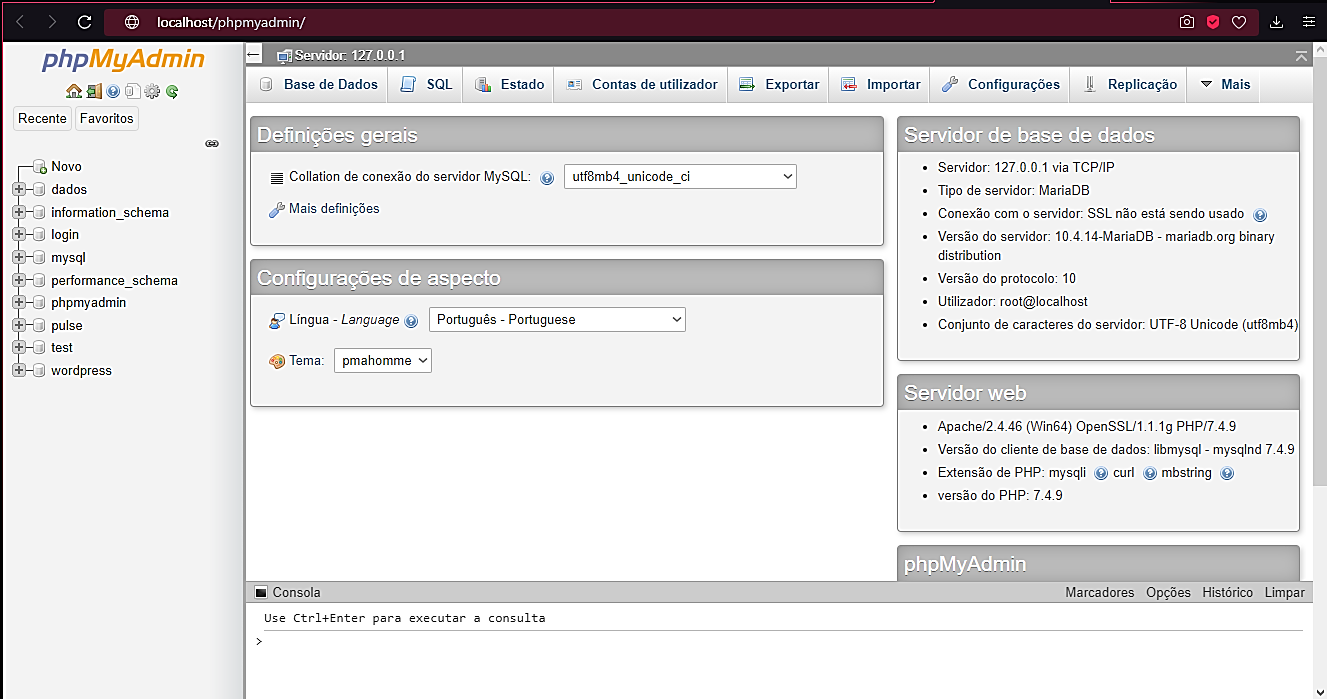


Figura A.11 – Gerenciador phpMyAdmin

# ANEXO B - INSTALAÇÃO VISUAL STUDIO CODE

**Sistema Operacional Usado:** Windows 10 Pro

**Etapa 1 – Instalação Visual Studio Code**

Acessando o site [**www.code.visualstudio.com**](http://www.code.visualstudio.com) apresentado na Figura B.1, a instalação é feita pelo próprio .exe do Visual Studio Code por ser mais fácil e rápida.

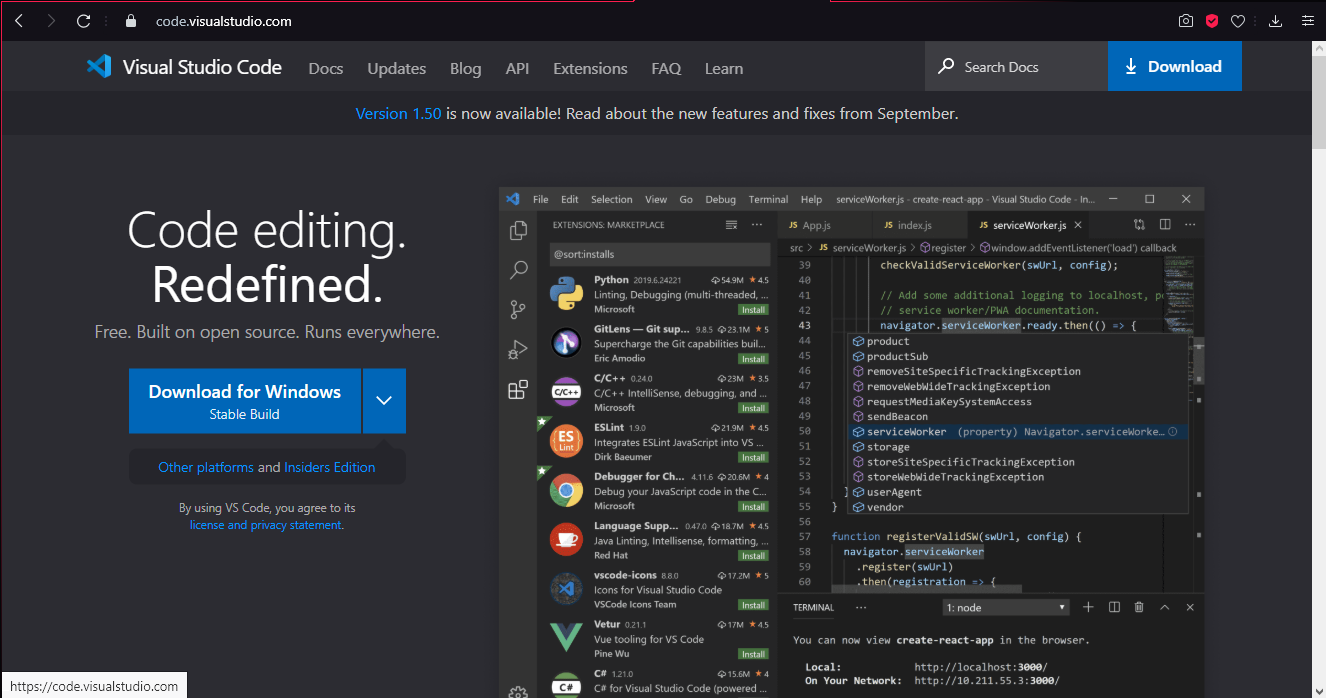


Figura B. - Página para *download* do Visual Studio Code

**Iniciando instalador Visual Studio Code**

Após abrir o executável do instalador, irá apresentar os termos de uso do *software*. Marcar a opção “Eu aceito o acordo” e clicar em “Próximo” como na Figura B.2.

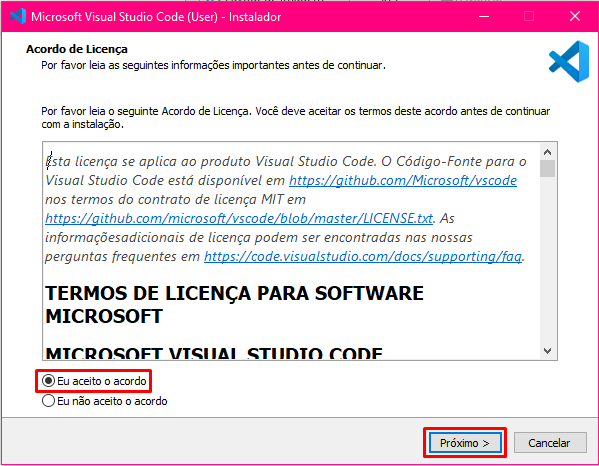


Figura B.2 – Tela inicial instalador Visual Studio Code

Após realizar essa etapa o instalador irá abrir outras opções para a instalação. Marcar as opções “Criar um atalho na área de trabalho” e “Adicione em PATH(Disponível após reiniciar)” e em seguida clique em “Próximo”. Como mostrado na Figura B.3.

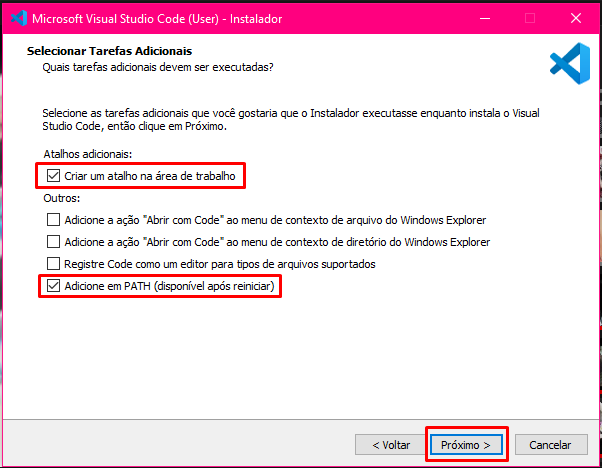


Figura B.3 – Opções de instalação Visual Studio Code

Assim que feitas as opções de configuração o instalador está pronto e é só clicar em “Instalar” e aguardar. Como mostrado na Figura B.4

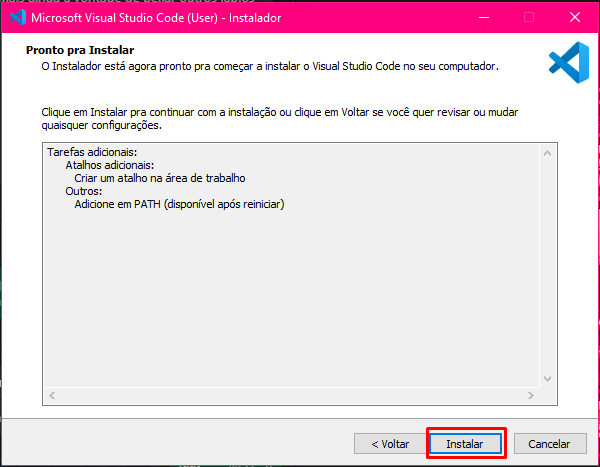


Figura B.4 – Instalação Visual Studio Code pronta

A seguir de todo esse processo o Visual Studio Code está pronto para uso como demonstrado na Figura B.5.

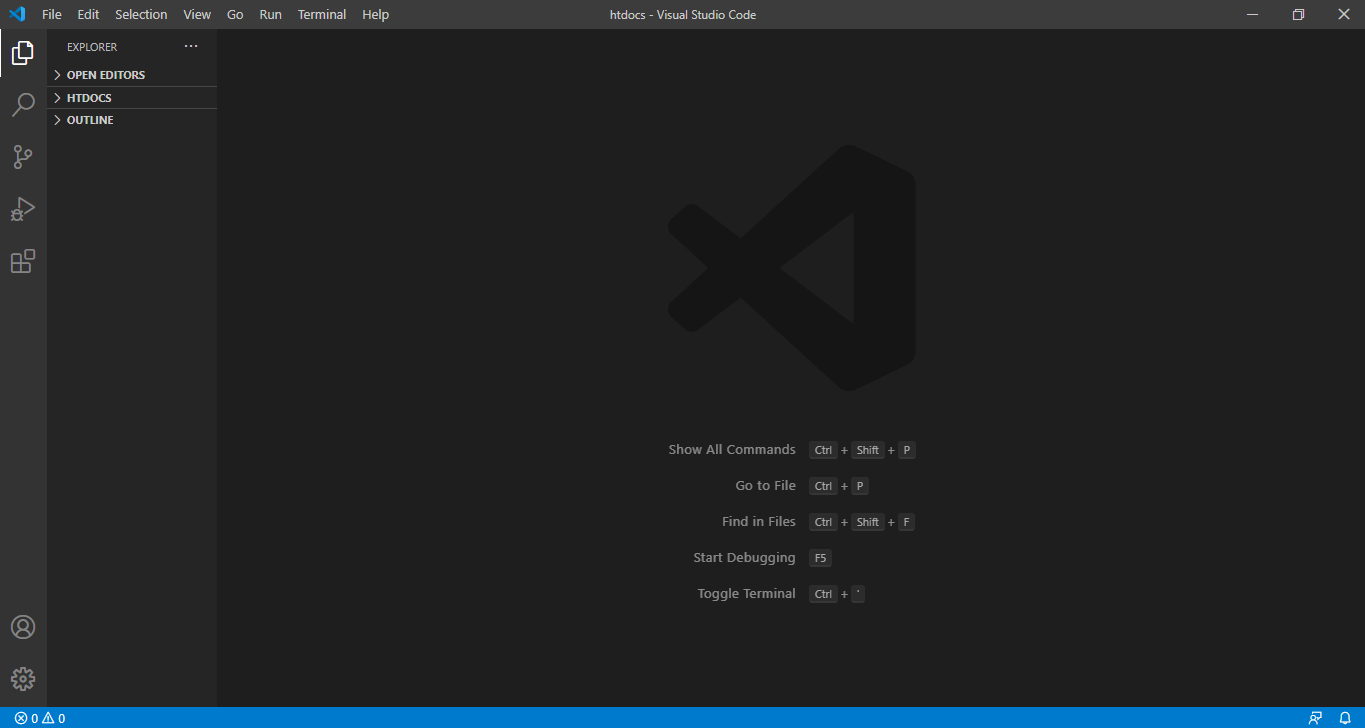


Figura B.5 – Visual Studio Code pronto para uso

# ANEXO C – CONFIGURAÇÃO ARDUINO IDE PARA O ESP32

**Sistema Operacional Usado:** Windows 10 Pro

**Etapa 1 – Instalação Arduino IDE**

Acessando o site [**www.arduino.cc/en/software**](http://www.arduino.cc/en/software) apresentado na Figura C.1, a instalação é feita pelo próprio .exe do Arduino IDE por ser mais fácil e rápida.

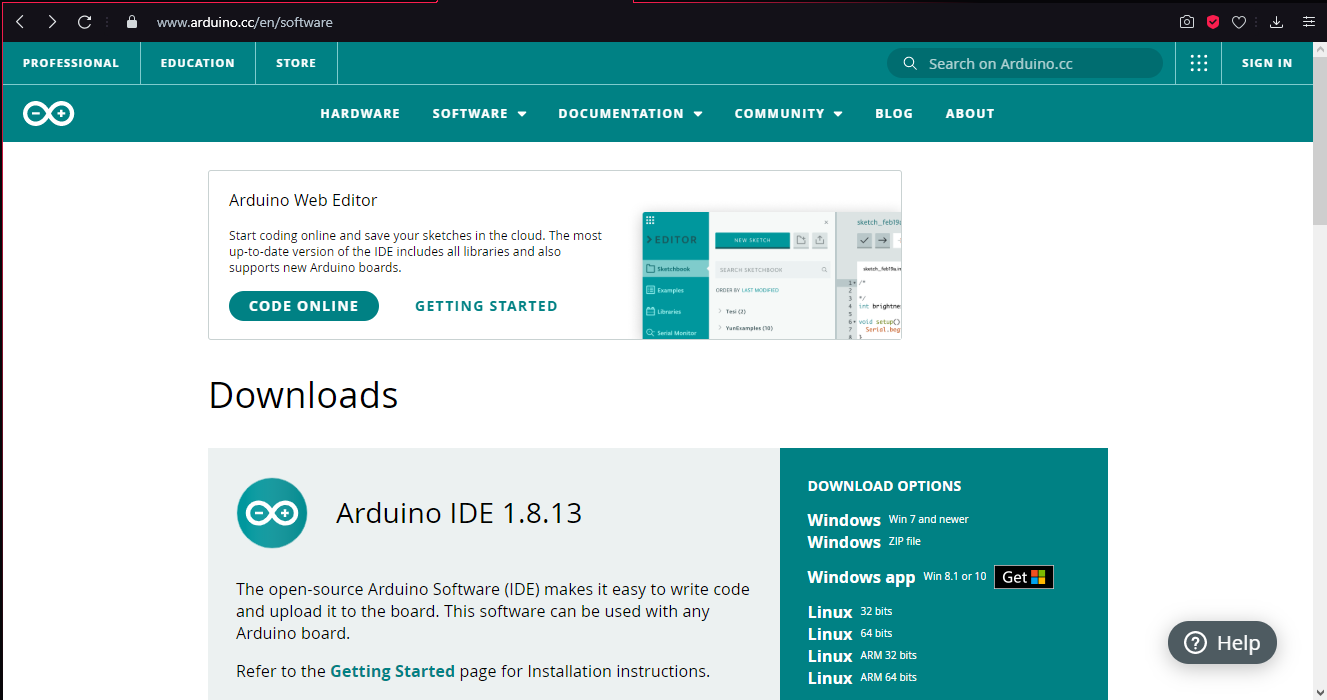


Figura C.1 - Página para *download* do Arduino ID

**Instalar o Arduino IDE**

Após abrir o instalador, aparecerá uma tela como mostrado na Figura C.2. Clicar em “I Agree”.

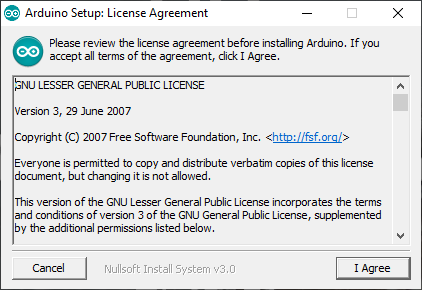


Figura C.2 Instalação Arduino IDE

Após essa etapa definir o local em que será feita a instalação do Arduino IDE, como demonstrado na Figura C.3 e em seguida clicar em “*Install*”.

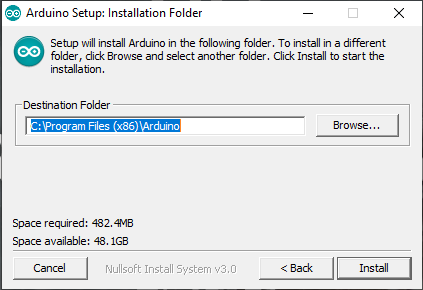


Figura C.3 – Local de instalação Arduino IDE

A seguir desse processo, apenas aguardar ser feito a instalação. Quando terminada aparecerá a seguinte tela como na Figura C.4. Basta clicar em “*Close*”.

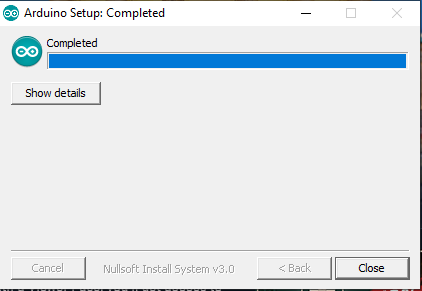


Figura C.4 – Instalação concluída Arduino IDE

**Etapa 2 – Adiciona a placa ESP32 no Arduino IDE**

**Adicionando reconhecimento da placa ESP32**

Abrir o Arduino IDE e ir em “Arquivos” e “Preferências” ou “Ctrl + ,” como demonstrado na Figura C.5.

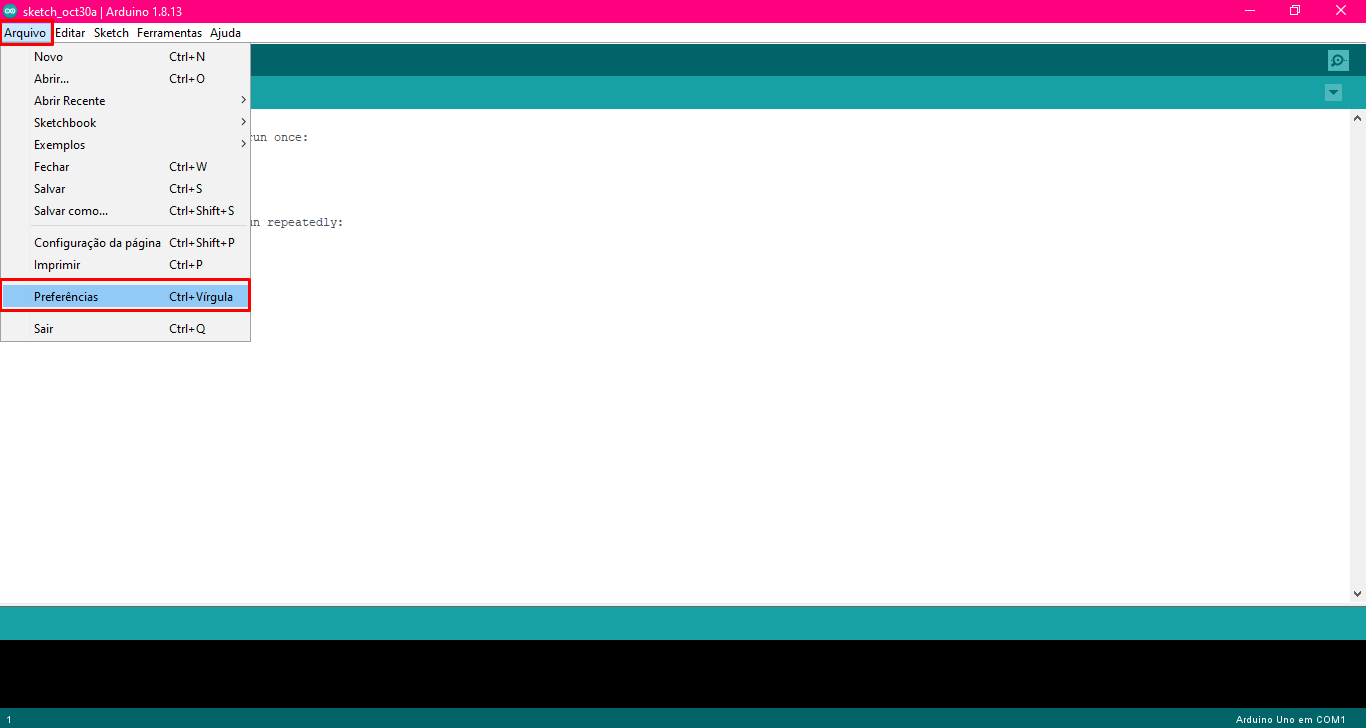


Figura C.5 – Arduino IDE

Abrirá então uma tela, deverá ser digitado: [**https://dl.espressif.com/dl/package\_esp32\_index.json**](https://dl.espressif.com/dl/package_esp32_index.json)em “URLs Adicionais para Gerenciadores de Placas” como demonstrado na Figura C.6.

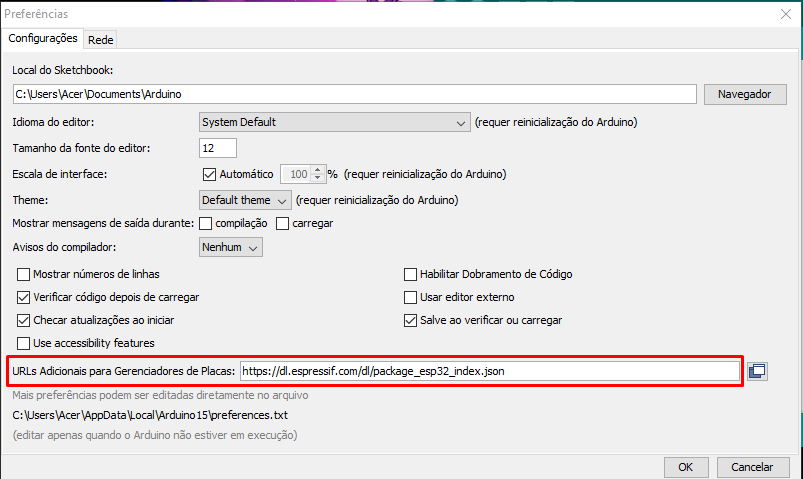


Figura C.6 – Adicionando placa ESP32 no Arduino IDE

Após esse processo a IDE estará pronta para uso com o ESP32. Basta selecionar o modelo da placa que deseja usar em “Ferramentas” e em seguida “Placa: “X” ”. Como demonstrado na Figura C.7.

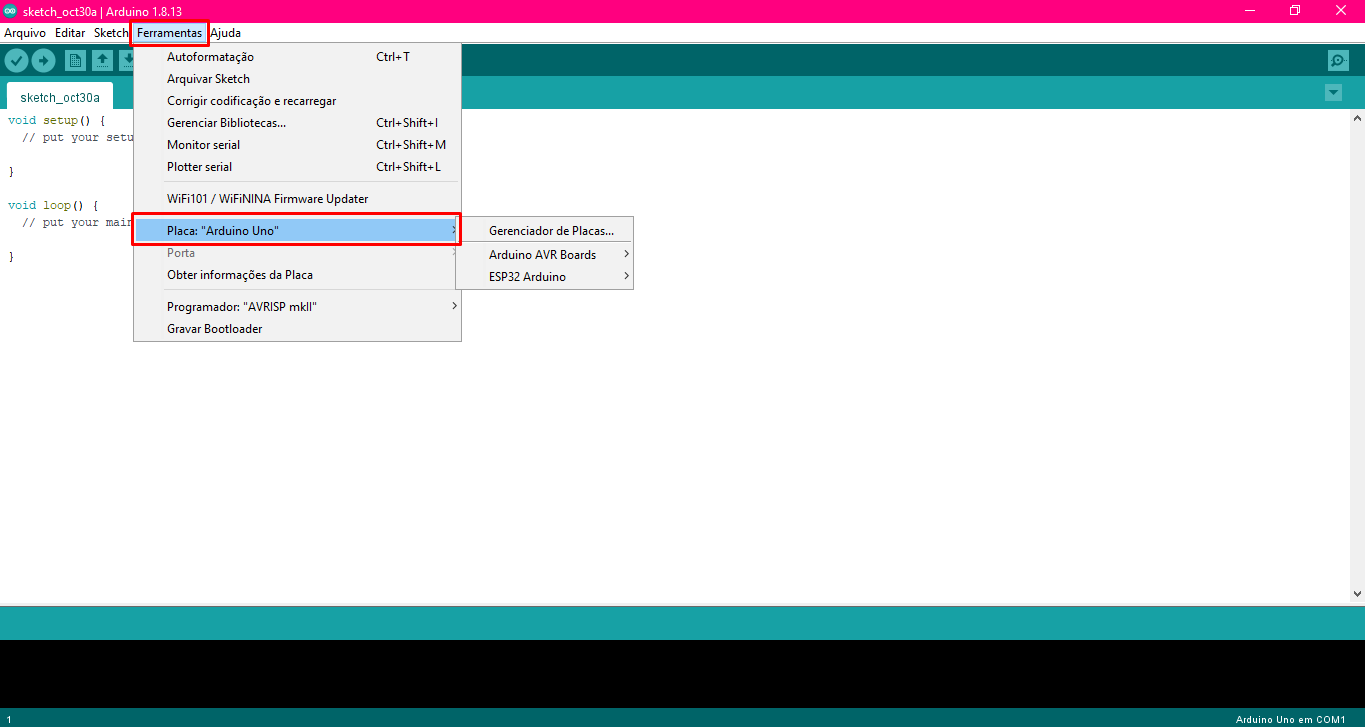


Figura C.7 – Definindo placa a ser utilizada