Biblioteca de autenticação por aproximação via Bluetooth

Authentication library by proximity via Bluetooth

José Claudio Pereira [*] Orientadora: Márcia Giraldi [†]
jpereira1330@gmail.com marcia.giraldi@animaeducacao.com.br

2022

Tubarão, Santa Catarina, Brasil

Resumo

Atualmente as linguagens de programação *C* e *C++*, que são essenciais no desenvolvimento para dispositivos embarcados, estão em baixa popularidade devido à complexidade e lentidão no desenvolvimento. E para isto, foi desenvolvida uma biblioteca que contempla as funcionalidades mais utilizadas por desenvolvedores de sistemas *IoT* tendo, como principal funcionalidade, a autenticação por aproximação via *Bluetooth*. A biblioteca e os testes da mesma foram realizados e o resultado foi satisfatório quanto as expectativas que projeto busca a resolver.

Palavras-chave: Autenticação; Bluetooth; Internet das coisas; ESP32

Abstract

Currently the *C* and *C++* languages programming, which are essential in development for embedded devices, are facing low popularity due to their complexity and slowness in development. To tackle this problem, the objective of this study was to develop a library which included the features which are mostly used by *IoT* systems developers having, as its main functionality, the authentication by approximation via Bluetooth. After the library was developed tests were performed, showing satisfactory results in terms of the objectives of this study.

Key Words: Authentication; Bluetooth; IoT; ESP32

[*] Acadêmico de Ciência da Computação da UNISUL
[†] Coordenadora do curso de graduação de Ciência da Computação da UNISUL

1 Introdução

O termo “internet das coisas”, mais conhecido pela sigla em inglês *IoT*, surgiu como resposta evolutiva de diversas subáreas do ramo tecnológico, tendo como principais ramos os microeletrônicos e comunicação. Conforme Santos et al. (2016), entendemos que *IoT* não se trata de um termo muito recente, pois foi inicialmente usado em 1999, contudo com nome diferente, sendo na época conhecido como *RSSF*, ou Rede de sensores Sem Fio, que utilizava como base os *RFID* (*Radio frequency identification* - Identificador de radio frequência); a sigla *RSSF* perdurou até 2005.

Hoje, o termo *RSSF* não é mais popular, pois o *IoT* acabou tomando seu posto. A ideia por trás desse termo remete ao seu sentido literal, "Internet das coisas", passando a noção de que qualquer coisa pode estar conectada à internet e com isso permitindo-lhe ter mais funcionalidades, como por exemplo, a geladeira criar uma lista de compras dos itens faltantes e enviar para o celular de seu proprietário como notificação.

Existem poucos projetos desenvolvidos em *C/C++* (DEITEL, 2011) como interface facilitadora para os programadores mais inexperientes na linguagem, e, os que existem, demandam a compreensão dos ponteiros de memória, uma característica básica da linguagem, porém de difícil compreensão para novos programadores.

Logo, se um programador de qualquer outra linguagem quiser desenvolver um projeto *IoT* no qual a tomada de decisão do dispositivo embarcado seja totalmente realizada na nuvem, usando *Python* por exemplo, isso é possível por meio da utilização dessas bibliotecas já existentes. Porém, a diferença é que, utilizando a biblioteca que será apresentada neste artigo, com poucas funções em *C++* e sem o uso explícito de ponteiros, o dispositivo embarcado já estará funcionando e pronto para receber comandos da nuvem.

2 Trabalhos correlatos

Foi realizada uma extensa pesquisa para encontrar um artigo científico como trabalho correlato em fontes como SciELO, RUNA ou Google Acadêmico e não foi encontrado nenhum outro artigo cientifico com um objetivo ou proposta próximo e/ou igual.

3 Problema e Solução

Conforme os dispositivos inteligentes vão se desenvolvendo e mais pessoas os vão utilizando, o mercado automaticamente vai pedir por cada vez mais programadores especializados no desenvolvimento de dispositivos *IoT*. Segundo o site *Devopedia*, uma pesquisa realizada em 2022 revelou que 56,4% dos dispositivos embarcados possui, como linguagem principal de desenvolvimento, o *C*, e em segundo lugar o *C++* com 38,3% (DEVOPEDIA, 2022).

Para termos uma métrica de quais linguagens de programação estão sendo mais utilizadas pelos desenvolvedores, utilizamos como base referencial o ranqueamento de popularidade emitido anualmente por alguns sites, que são frequentemente utilizados pelos programadores. Segundo o relatório publicado por Carbonnelle (2022) em abril de 2022 no PYPL (*PopularitY of Programming Language*) que baseia seus resultados nas tendências de pesquisa realizadas através do buscador Google, vimos que as linguagens *C* e *C++* que são as mais utilizadas no mercado para *IoT*, flutuam frequentemente entre a 5ª e a 6ª posição com apenas 7,00% das buscas, isso, levando em conta que esta porcentagem contempla a somatória de ambas as linguas. Para efeito de comparação, o primeiro lugar fica com o *Python*, possuindo 27,85% das pesquisas direcionadas.

O *Stack Overflow* utilizou uma estratégia diferente para levantar os dados para seu próprio relatório anual; sua estratégia foi aplicar um questionário entre os usuários da plataforma no qual o usuário responde apenas às perguntas que lhe interessarem. Dentre as perguntas formuladas pelo site, uma delas questiona se o usuário já possuiu algum contato com *C* ou *C++*, mesmo que seja de forma não profissional. Os dados indicaram que por volta de apenas 21 mil (35%) de aproximadamente 58 mil pessoas já tiveram algum tipo de contato com a linguagem. (STACKOVERFLOW, 2021b)

Continuando, no mesmo relatório do Stackoverflow (2021a), houve duas perguntas muito interessantes, que atingiram aproximadamente 83 mil usuários da plataforma e que são as seguintes: "Qual linguagem de programação você mais quer aprender?"e, respectivamente, "De qual linguagem você mais gosta e qual mais detesta?", que revelaram que apenas 3 mil (3,5%) do total de usuários querem aprender *C*, deixando-o na 15ª posição do ranking, ao passo que a segunda pergunta evidenciou que, dos 83 mil respondentes, apenas 7 mil (8,5%) usuários amavam a linguagem *C*, contra os 10 mil que marcaram como detestavam, o deixando na 31ª posição da mais amada. Os demais 66 mil usuários selecionaram outras linguagens demonstrando indiferença para o *C*.

Depois de analisarmos ambos relatórios, é perceptível que há um afastamento por parte de novos programadores do *C* e *C++*. Com esta informação, podemos deduzir que pode existir um déficit de programadores dessas linguagens no mercado, não acompanhando a demanda de novos programadores devido ao crescimento das novas tecnologias de *IoT*. Como solução deste problema, este projeto busca desenvolver uma biblioteca de desenvolvimento em *C++*, capaz de ajudar o programador a implementar algumas funcionalidades específicas sem precisar de muito conhecimento da linguagem.

Um outro problema, trata-se dos próprios desenvolvedores *C/C+* atualmente empregados, em que alguns destes desconhecem ou não se preocupam com a segurança dos dados do cliente os armazenando de qualquer forma em disco ou os mantendo em *RAM* sem uma necessidade.

Um erro muito comum dos programadores mais iniciante, trata-se de capturar e manipular valor sensíveis do usuário e não tratar este dado de forma correta, como

por exemplo uma senha ou um dado pessoal. imaginemos um cenário onde o usuário insere a senha dele, o programa lê esta senha e faz as validações necessárias, após isto, o desenvolvedor simplesmente esquece esta memoria alocada ou desaloca da memoria (caso tenha sido anteriormente alocada), o erro neste processo é que hoje existem alguns programas que varrem a memoria que foi utilizada pelo programa a procura de dados sensíveis esquecidos ou não encriptados.

Foi escolhido o *C++* como linguagem de desenvolvimento da biblioteca, pois a maioria dos dispositivos *IoT* a utilizam como linguagem de desenvolvimento principal, e além deste ponto, as bibliotecas *C* também podem ser utilizadas por programas escritos em *C++*. Assim, com apenas uma única biblioteca, conseguimos facilitar o desenvolvimento em duas linguagens de programação.

4 Solução proposta

Para resolver o problema apresentado, este trabalho traz como solução a implementação de uma biblioteca em *C++* que tem como objetivo, a implementação de funcionalidades especificas, para que desta forma, quem utiliza-la não precise entender muito a linguagem *C/C++*. Indiretamente, a biblioteca tambem facilita e agiliza o desenvolvimento de novas aplicações *IoT*, sem que o programador precise se preocupar com conexão *WiFi*, pareamento *Bluetooth*, Protocolo *HTTPs* e gerenciamento de *JSON*.

4.1 Escopo

A biblioteca de autenticação se limita ao *ESP32*, permitindo que o desenvolvedor possa escolher a plataforma e/ou a linguagem que alimentará a biblioteca em questão, desde de que respeite o protocolo estabelecido pelo *JSON*; por exemplo, neste trabalho utilizamos, como prova de conceito, um servidor *Lambda* hospedado na *AWS* e escrito em *Python*.

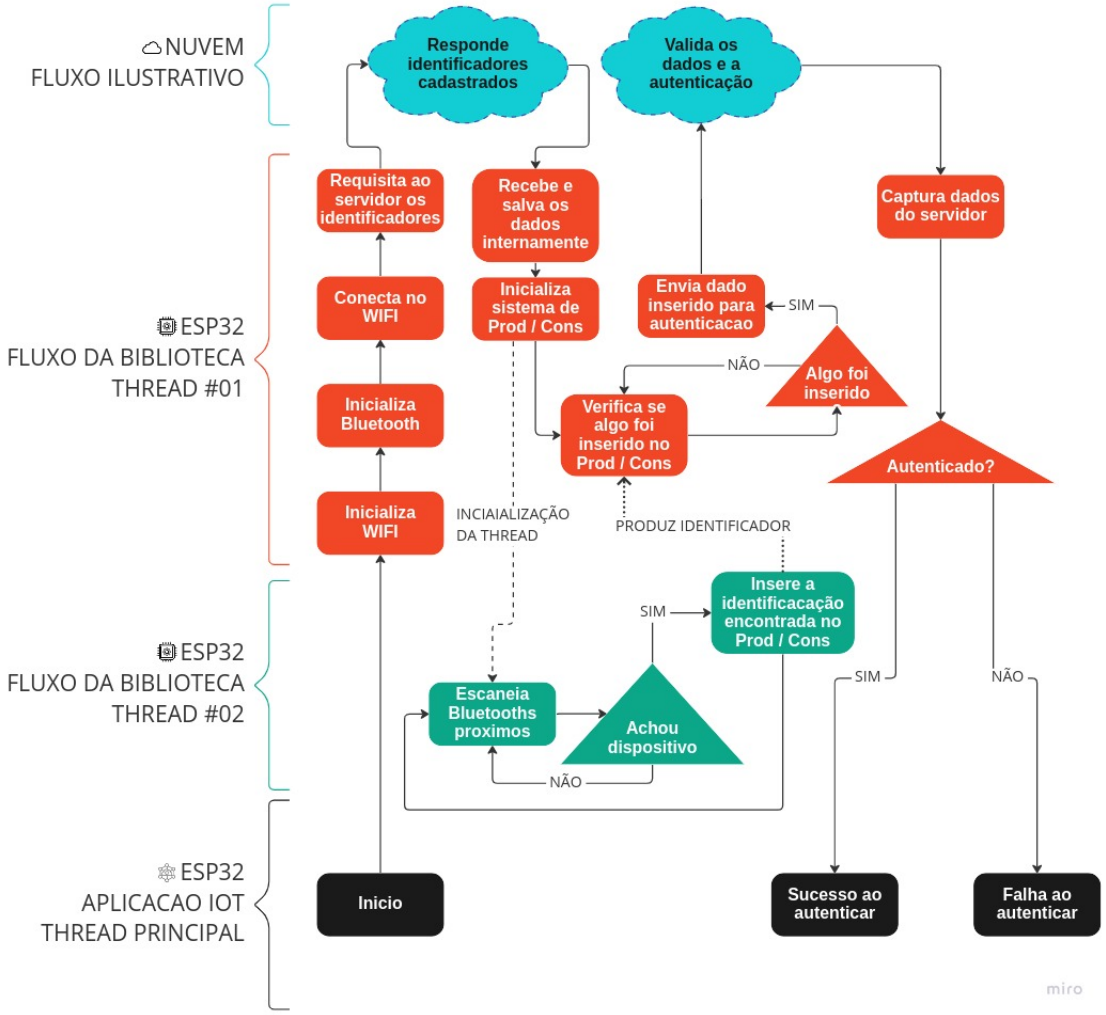
As principais funcionalidades da biblioteca, serão: Escanear e conectar a uma rede *WiFi*; Escanear as interfaces de *bluetooth* próximas; Manipular dados *JSON*; trocar dados com um servidor remoto.

A aplicação que utilizará a biblioteca e o servidor ao qual a biblioteca se conectará, estão fora do escopo do projeto, desta forma, permitindo que o desenvolvedor que utilizará a biblioteca fique livre para a escolha de suas tecnologias e como usará a biblioteca.

4.2 Funcionalidades implementadas na biblioteca

A biblioteca permitirá, conectar e desconectar do *WiFi*, trocar dados com um servidor remoto para validar quaisquer dados que o usuário necessitar, além de permitir o gerenciamento do módulo de *Bluetooth* do *ESP32*, tendo, como possibilidade, ligar e

Figura 1 - Fluxo simplificado de autenticação da biblioteca



Fonte: Autor (2022).

desligar o módulo, escanear os equipamentos de *Bluetooth* próximos e capturar informações deles.

4.3 Funcionamento

Para facilitar o entendimento do funcionamento da arquitetura macro da biblioteca, observe a **Figura 1 - Fluxo simplificado de autenticação da biblioteca**, onde o passo a passo das tomadas de decisão da aplicação através de um fluxograma é descrito de forma simplificada. Este fluxograma é separado em quatro etapas que estão descritas nos tópicos seguintes e em ordem de execução.

4.3.1 Aplicação IoT

Esta etapa se trata da própria aplicação do desenvolvedor, o que significa que é a primeira a ser executada, e é responsável pela inicialização, configuração, execução e consultas da biblioteca de autenticação. Esta etapa foge do escopo implementável do

projeto, pois fica a critério do utilizador da biblioteca decidir como deverá utilizá-la, e, por isso, é de suma importância incluir esta etapa amostra no fluxograma para uma melhor compreensão.

4.3.2 Fluxo da biblioteca - Thread 01

Já a segunda etapa da aplicação, a qual é inicializada pela *Aplicação IoT*, é a que mais possui responsabilidades, variando entre: inicialização e gestão das interfaces de comunicação (sendo elas *WiFi* e *Bluetooth*), comunicação com o servidor descrito posteriormente na etapa **Fluxo ilustrativo - Nuvem** a fim de atualizar o banco de dados local e validar autenticação do dispositivo, e, como última responsabilidade, gestão do produtor e consumidor da aplicação para que seja possível as *threads* se comunicarem entre elas de forma correta com o auxilio do semáforo, sem se preocupar com possíveis conflitos de memória.

4.3.3 Fluxo da biblioteca - Thread 02

Conforme pode ser observado na **Figura 1**, a etapa de *Fluxo da biblioteca - Thread 02* se trata de um processo independente em que seu único dever é verificar se existe um bluetooth próximo e, caso encontre, este processo deve informar o *Fluxo da biblioteca - Thread 01* através de um produtor consumidor.

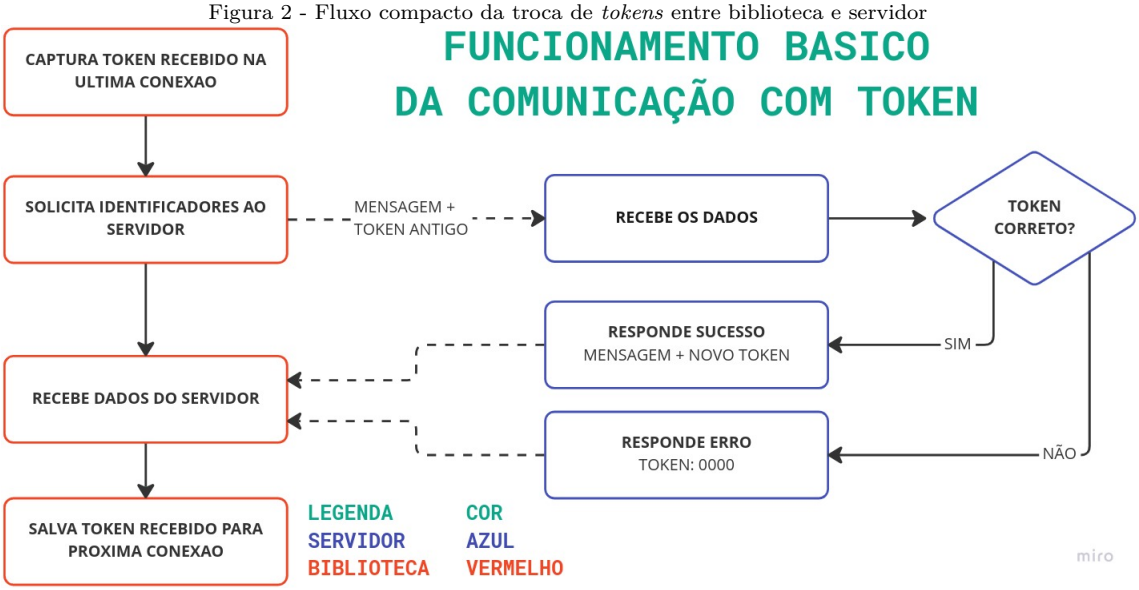
4.3.4 Fluxo ilustrativo - Nuvem

Esta etapa é a ultima das quatro e, assim como a primeira, e pelos mesmos motivos, foge do escopo de implementação do projeto. Esta decisão foi tomada, pois é indiferente a linguagem ou plataforma em que o servidor será hospedado, permitindo assim que o desenvolvedor possa escolher o que mais se adeque ao seu cenário.

4.4 Métodos de segurança

Devido ao modelo de atuação desta biblioteca de segurança ser restritamente embarcado, a substituição do próprio *hardware* para fazer um *bypass* no sistema é um problema de segurança real, assim, como solução, a biblioteca possui um campo a mais chamado de *token* que deve constar em toda troca de mensagem com o servidor, ou seja, quando o biblioteca entra em contato com o servidor, ela envia em conjunto da mensagem o *token* que foi recebido na última conexão e a resposta dessa conexão atual é finalizada com o servidor enviando o *token* que deverá ser enviado na próxima conexão; assim garantimos que o *Hardware* é o mesmo e não foi modificado. Pode-se entender melhor o fluxo acompanhando a **Figura 2 - Fluxo compacto da troca de tokens entre biblioteca e servidor**, onde toda a conexão segue este fluxograma.

Figura 2 - Fluxo compacto da troca de *tokens* entre biblioteca e servidor



Fonte: Autor (2022).

O *token*, como já mencionado, é um valor que deve variar toda vez que a biblioteca entra contato com o servidor, e esse campo deve possuir quatro caracteres alfanuméricos para que desta forma, existam 96^4 teste (noventa e seis elevado a quatro) combinações possíveis, deixando o algoritmo mais seguro pela baixa probabilidade de alguém adivinhar este código.

5 Tecnologias utilizadas

Após um estudo da arquitetura proposta como solução para o problema apresentado, foi percebido que para a implementação do projeto da biblioteca de autenticação, era necessário fazer o uso de algumas tecnologias específicas, entre elas o *Bluetooth*, *ESP32* e *WiFi*.

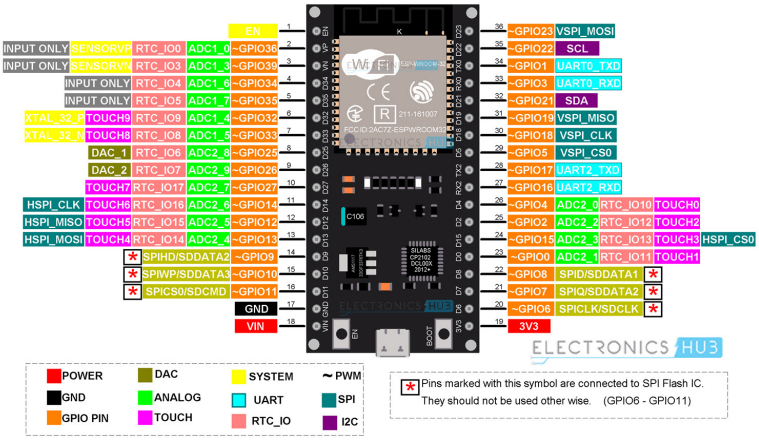
5.1 Bluetooth

Bluetooth é uma rede de tipo *PANs*(*Personal Area Networks*), desenvolvida para realizar troca de dados sem fio entre dispositivos próximos por meio de frequências de rádio de curto alcance, levando em conta a segurança e o baixo consumo de energia (SIQUEIRA, 2006).

5.2 WIFI

Atualmente, a tecnologia *WiFi* se tornou presente no dia-a-dia de muitas pessoas, sendo possível encontrar uma rede disponível para se conectar na maioria das casas e estabelecimentos, por se tratar de uma tecnologia que não exige cabos para permitir a conexão com a internet.

Figura 3 - ESP32 - Pin Out



Fonte: Teja, 17 de fevereiro. 2017. Disponivel em:
https://www.electronicshub.org/esp32-pinout/
Acesso em 01 de jun. 2022

WiFi, ou, como é chamado tecnicamente, *IEEE 802.11*, se trata de uma tecnologia que usa as ondas de rádio para trocar dados entre o cliente (Dispositivo conectado) a um aparelho *WiFi*. Esta troca de dados deve ser efetuada de forma bidirecional, ou seja, o aparelho *WiFi* deve tanto transmitir estas ondas de rádio como recebê las. (OLIVEIRA, 2020)

5.3 ESP32

O microcontrolador *ESP32* foi arquitetado e desenvolvido por uma empresa chinesa chamada *Espressif Systems*. Sua primeira versão foi lançada em 2016 com o objetivo de lançar no mercado uma solução mais completa contendo *WiFi* 802.11n (2.4GHz) e *Bluetooth* 4.2 integrado ao chip. O microcontrolador possui 520 kb de SRAM, 448 kb de ROM com 2 núcleos de 240 Mhz de processamento. (ESPRESSIF, 2021)

O *ESP32* foi escolhido para este projeto pois, devido a suas especificações de *Hardware* ele se revela uma placa barata e de fácil acesso a qualquer pessoa. Diferentemente do *Arduino*, que é mais conhecido e utilizado em Universidades e escolas técnicas, o *ESP32* consegue ser mais potente e barato.

A placa possui 36 *GPIO*, isto significa que existem 36 pinos de conexão para ligá-los em outros módulos e sensores a fim de ampliar a capacidade de atuação da placa; por exemplo: medidores de umidade, vento ou movimento.

6 Utilização

Neste tópico será abordado como o desenvolvedor poderá utilizar a biblioteca e detalhar as dependências para o correto funcionamento.

6.1 Preparação do ambiente

Para a correta utilização da biblioteca, é necessário primeiramente configurar o ambiente de desenvolvimento. A configuração pode ser realizada através do seguinte tutorial da própria fabricante Espressif (2022), que traz o passo a passo contendo a solução de alguns problemas que podem ocorrer durante as configurações do ambiente.

6.2 Uso da biblioteca

Neste tópico, será descrito quais funções foram implementadas para o uso da biblioteca.

```
1    bool bsa_init();
2    bool bsa_update(string url);
3    bool bsa_verify(string &listIdent);
4    bool bsa_authentication(string url, int indice);
5    bool bsa_start_scan();
6    bool bsa_stop_scan();
7    bool bsa_connect_wifi(string essid, string passwd);
```

- Função *bsa_init()*: Responsável por inicializar as *thread* e variáveis internas, permite que a biblioteca possa ser utilizada de forma correta. Em caso do utilizador da biblioteca não executar esta função, ou, executa-lá mais de uma vez, fará que a biblioteca fique com um comportamento imprevisto.

- Função *bsa_update(String)*: Possui o objetivo de atualizar o banco de dados interno da biblioteca, para que mesma saiba quais *MACs* possam ser enviados para uma possível autenticação. Esta função recebe como parâmetro uma variável do tipo *String* que deverá constar o *link* do servidor a ser questionado. A mensagem *JSON* que esta função enviara, esta descrita posteriormente no tópico **6.3.1 Requisição dos identificadores**.

- Função *bsa_verify(String)*: Esta função tem como objetivo capturar e retornar uma lista de *MACs* escaneados de ordem decrescente pelo nível de sinal. Esta lista sera separada pelo carácter ';' e só deverá constar os *MACs* que foram recebidos anteriormente pela função **bsa_update(String)**, logo, a execução da função **bsa_update(String url)** deverá ser anterior a esta descrita.

- Função *bsa_autentication(String, int)*: Realiza a solicitação da autenticação ao servidor que é informado no primeiro parâmetro da função, enquanto o segundo de tipo inteiro, serve para indicar qual *MAC* deve tentar autenticar pelo índice da fila.

- Função *bsa_start_scan()*: Executará a *Thread* que fará o escaneamento dos *Bluetooths* próximos do *ESP32*.

- Função *bsa_stop_scan()*: Para a *Thread* que fará o escaneamento dos *Bluetooths* próximos do *ESP32*.
- Função *bsa_connect_wifi(String, String)* : Tem como responsabilidade autenticar no *WiFi*. primeiro parâmetro se trata do nome da *WiFi* a ser conectada e o segundo a senha.

6.3 Campos do JSON

Neste tópico explicaremos a estrutura do *JSON* que será recebido e enviado pela etapa de *Fluxo ilustrativo - Nuvem* para o dispositivo embarcado.

6.3.1 Requisição dos identificadores

A biblioteca enviará este *JSON* para requisitar ao servidor a lista de identificadores candidatos para a autenticação.

```
{
   "token"       :"xxxxxx",
   "id"          :"xxxxxxxxxxxx",
   "request"     :"UPDATE"
}
```

- Campo *token*: A função e funcionamento deste campo já foram explicados anteriormente no tópico **Métodos de segurança**.
- Campo *id*: Deve conter 12 caracteres alfanuméricos que significa o código do dispositivo; este código, por obrigação, deve ser único.
- Campo *request*: Possui como objetivo informar o tipo da solicitação desejada, neste caso, deverá conter o valor *UPDATE*

Após a biblioteca enviar os dados acima, ela passa a esperar como resposta a estrutura *JSON* descrita abaixo; esta estrutura deve conter uma cadeia de objetos contendo nome e o identificador único da interface da placa de rede que, neste projeto utilizamos o *MAC*. Em conjunto destes dados já mencionados, deve ser enviado também o *token* como já previsto como regra no tópico **Métodos de segurança**.

```
{
   "token" : "xxxxxx",
   "identifiers" :[
        {
            "name"  : "Primeiro dispositivo",
            "mac"   : "00:00:5e:00:53:af"
```

```
        },
        {
            "name"  : "Segundo dispositivo",
            "mac"   : "00:00:5e:00:54:aa"
        },
    ]
}
```

- Campo *token*: A função e funcionamento deste campo já foram explicados anteriormente no tópico **Métodos de segurança**.
- Campo *identifiers*: Campo do *JSON* que representa uma cadeia de objetos, cada objeto dentro deste campo representa um dispositivo *Bluetooth* que pode se autenticar.
- Campo *name*: Este campo pertence ao *identifiers*, ele representa o nome do dispositivo para facilitar a identificação de qual *mac* pertence a qual dispositivo.
- Campo *mac*: Assim como *name*, este campo também pertence aos *identifiers*, e é responsável por armazenar o código da interface de comunicação que pode ser aceita.

6.3.2 Requisição da autenticação

Neste tópico será descrito como a autenticação será realizada. Por padrão, para o correto funcionamento desta etapa, é de obrigação realizar a etapa anterior **Requisição dos identificadores**

```
{
    "token"     : "xxxxxx",
    "id"        : "xxxxxxxxxxxx",
    "request"   : "AUTH"
    "mac"       : "ff:ff:ff:ff:ff:ff"
}
```

- Campo *token*: A função e funcionamento deste campo já foram explicados anteriormente no tópico **Métodos de segurança**.
- Campo *id*: Assim como já explicado anteriormente, deve conter 12 caracteres alfanuméricos que representa o dispositivo embarcado que está enviando esta informação.
- Campo *request*: Requisição que seré realizada ao servidor. Neste caso, deverá ser utilizado o valor **AUTH** para sinalizar autenticação.
- Campo *mac*: Identificador do dispositivo que deve ser conectado.

Como resposta da autorização, o servidor deve responder o seguinte *JSON* para a biblioteca, como resposta do *JSON* descrito acima.

```
{
    "token"      : "xxxxxx",
    "id"         : "xxxxxxxxxxxx",
    "status"     : "2XX" OU "4XX"
    "mac"        : "ff:ff:ff:ff:ff:ff"
}
```

- Campo *token*: A função e funcionamento deste campo já foram explicados anteriormente no tópico **Métodos de segurança**.
- Campo *id*: Assim como já explicado anteriormente, deve conter 12 caracteres alfanuméricos que representa o dispositivo embarcado que está enviando esta informação
- Campo *status*: Neste campo o servidor informa qual a decisão do mesmo, se a autenticação esta aprovada ou não, e a decisão deverá ser representada por uma variável de numero inteiro com três posições, sendo que caso a cadeia comece com o numero dois, significa que o sistema esta autorizado, caso não, o sistema retornará o numero quatro no inicio da cadeia, o motivo de ter mais numero, devesse a possibilidade de informar mais algum dado numérico que representa uma informação especifica.
- Campo *mac*: Identificador do dispositivo que deve ser conectado

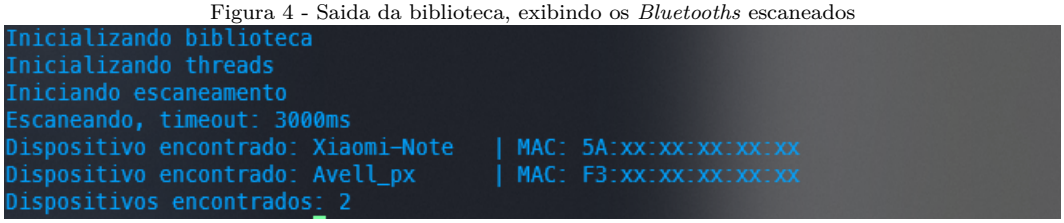
7 Testes

Como prova de conceito, foi de necessidade desenvolver um protótipo como teste para ter certeza de que a biblioteca realmente funciona como esperado. O protótipo realizado foi escrito pensando em testar a funcionalidade, segurança e usabilidade da biblioteca.

Como cenário do protótipo, foi escolhida a área de segurança automotiva. Nesse cenário quando o carro detectar uma partida, a biblioteca será chamada para varrer os dispositivos próximos usando *bluetooth* e validar se algum deles está cadastrado no sistema; caso sim, a partida do carro é liberada.

Os testes em específicos realizados em cima deste protótipo foram: Realizar uma tentativa de autenticação funcional com um dispositivo cadastrado; Testar a autenticação de um dispositivo não cadastrado; Realizar duas autenticações simultâneas, aproximando dois dispositivos cadastrados ao mesmo tempo. Todos os testes listados apresentaram os resultados esperados, autenticando quando necessário e ignorando a autenticação se o dispositivo não foi cadastrado anteriormente.

Figura 4 - Saida da biblioteca, exibindo os *Bluetooths* escaneados



Fonte: Autor (2022)

Como teste de usabilidade e intuitividade da biblioteca, foi de necessidade solicitar para terceiros tentarem entender a biblioteca e testa-la, sem qualquer ajuda do autor. Desta forma, conseguimos representar a realidade de um possível programador que utilizará a biblioteca. O resultado deste teste foi em parte satisfatório, pois, todos os programadores que analisaram a biblioteca, conseguiram entender de forma fácil o funcionamento da mesma, o problema em questão foi relacionado a desenvolvimento utilizando a biblioteca, onde tive dificuldade de encontrar alguém que possua o *ESP32* e que pudesse implementar, apenas uma pessoa realizou o teste de implementação em cima da biblioteca, porém, não conseguiu finalizar o desenvolvimento, relatando falta de tempo e não por dificuldades encontradas na biblioteca.

8 Análise dos resultados

Após desenvolver e reproduzir o cenário descrito no tópico **Testes**, ficou claro que o objetivo do projeto de implementar uma biblioteca capaz de facilitar as funções básicas da areá de *IoT* foi atingido, pois, um projeto como este, que foi descrito no tópico **Testes**, teria uma complexidade muito maior caso desenvolvido sem a biblioteca.

Uma importante constatação que deve ser levada em conta sobre o cenário desenvolvido como prova de conceito, trata-se da precisão das autenticações. Em todos os casos, quando algum dispositivo cadastrado chegou próximo do *ESP32*, rapidamente o sistema foi autenticado e liberado para o automóvel dar a partida.

Após colocar em prática o que foi descrito no tópico **6.4 Testes**, foi percebido um problema de segurança na biblioteca que, segundo nossa análise, não é grave para as demais aplicações da mesma. O problema resulta justamente da escolha de projeto de usar como chave de validação de proximidade o *MAC* do terceiro.

Para reproduzir a falha e burlar o sistema para ganhar acesso ao mesmo, é necessário descobrir um dos *MAC* cadastrados no *ESP32* e, com um terceiro dispositivo mutável, alterá-lo para que passe a transmitir o *MAC* cadastrado como interface *Bluetooth*. desta forma o *ESP32* vai encontrá-lo na proximidade e permitir seu acesso ao dispositivo que o usuário configurou.

Depois de analisar este cenário, foi reconhecido a falha de segurança e julgada como baixa periculosidade, por mais que seu impacto seja justamente a autenticação

indevida de um terceiro. Esta decisão de marcar como baixa periculosidade se deve à grande complexidade de realizar este feito.

A correção desse problema se torna inviável pela arquitetura do projeto, pois não está no escopo do mesmo o uso de um dispositivo terceiro que o usuário tenha que utilizar para que seja possível a biblioteca trocar dados e conferir se de fato o dispositivo é real. Isso implicaria em alterar um dos propósitos do trabalho que é permitir que a realização da autenticação seja possível através de qualquer dispositivo *Bluetooth*.

9 Conclusão

O objetivo geral deste projeto foi desenvolver uma biblioteca que permita facilitar o desenvolvimento de novas aplicações *IoT*, pois, conforme mencionado, a linguagem de programação recomendada para tal desenvolvimento *(C/C++)*, é considerada mais complexa do que as atualmente mais populares.

O protótipo realizado seguindo o cenário descrito no tópico **Testes** atendeu as expectativas, resolvendo o principal problema levantado por este projeto, ou seja, agilizou e facilitou a implementação de uma aplicação *IoT* que utilize este recurso.

Durante o desenvolvimento da biblioteca, houve uma certa dificuldade durante a implementação da biblioteca, pois em um determinado momento, a *IDE* que estava utilizando para desenvolvimento apontou que havia atingido o limite máximo de armazenamento do *ESP32*. Isso, entretanto, não era verdadeiro. Após uma longa pesquisa, foi descoberto que é necessário alterar uma configuração interna na IDE, pois, como padrão, a mesma aceita o envio de apenas *56KB*, em vez dos *4Mb* presentes.

Tendo em vista o problema detectado nesta pesquisa, apontado no tópico **Análise dos resultados**, sugerimos, como pesquisa futura, pesquisar e arquitetar uma forma de solucioná-lo. Segundo minhas pesquisas iniciais, possivelmente este problema poderá ser resolvido através de um pareamento entre o dispositivo e o dispositivo *ESP32* que possui a biblioteca.

Este projeto cumpre com o objetivo descrito neste artigo, facilitando a implementação da funcionalidade de autenticação por proximidade via *Bluetooth* para programadores de outras linguagens de mais alto nível, ou programadores novos na área de desenvolvimento *IoT*.

Referências

CARBONNELLE, P. *PYPL PopularitY of Programming Language.* 2022. Disponível em: <https://pypl.github.io/PYPL.html>. Acesso em: 10 de abril de 2022. Citado na página

3.

DEITEL, H. D. P. *C: Como Programar - 6ª edição*. [S.l.]: Pearson Universidades, 2011. ISBN 8576059347. Citado na página 2.

DEVOPEDIA. *Programming for IoT*. 2022. Disponível em: <https://devopedia.org/programming-for-iot>. Acesso em: 10 de abril de 2022. Citado na página 2.

ESPRESSIF. *Technical Reference Manual*. 2021. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf>. Acesso em: 06 de junho de 2022. Citado na página 8.

ESPRESSIF. *Getting Started with VS Code IDE*. 2022. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/vscode-setup.html>. Acesso em: 20 de abril de 2022. Citado na página 9.

OLIVEIRA, C. *Como funciona o WIFI*. 2020. Disponível em: <https://www.embarcados.com.br/como-funciona-o-wifi/>. Acesso em: 01 de junho de 2022. Citado na página 8.

SANTOS, B. P. et al. Internet das coisas: da teoriaa prática. *Minicursos SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos*, 2016. Citado na página 2.

SIQUEIRA, T. S. de. *Bluetooth–Características, protocolos e funcionamento*. 2006. Disponível em: <http://www.eletronica24h.net.br/files/Bluetooth_Caracteristicas_Protocolos_Funcionamento.pdf>. Acesso em: 10 de maio de 2022. Citado na página 7.

STACKOVERFLOW. *Most loved dreaded and wanted language love dread*. 2021. Disponível em: <https://insights.stackoverflow.com/survey/2021#most-loved-dreaded-and-wanted-language-love-dread>. Acesso em: 12 de abril de 2022. Citado na página 3.

STACKOVERFLOW. *Most popular technologies language prof*. 2021. Disponível em: <https://insights.stackoverflow.com/survey/2021#most-popular-technologies-language-prof>. Acesso em: 12 de abril de 2022. Citado na página 3.

Agradecimentos

Primeiramente, agradeço a meus familiares, principalmente aos meus pais, Cláudio Pereira e Sônia Souza Pereira, que me deram toda força e apoio emocional, estando ao meu lado nos momentos mais difíceis, assim, permitindo com que eu possa trilhar meu caminho sem tantos obstáculos.

Sou grato também, ao Marco Aurélio Tostes, um colega de trabalho que reconheço que fez diferença em minha vida acadêmica, pois, assim como meus professores, conseguiu me deixar mais empolgado com a areá da ciência da computação e suas sub-areas, fazendo com que eu tenha mais prazer em aprender.

Faço uma menção honrosa a minha vó, Alvina Faustina Pereira, ou, mais conhecida como "Didina", que desde minha infância me instigava para que eu entrasse numa faculdade,

que, infelizmente, ela não está mais presente para poder ver seus netos se formando como era um de seus sonhos.