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

MARIA HELENA ROCHA DE ALENCAR BEZERRA

CONTROLE DE ACESSO A AMBIENTES FÍSICOS UTILIZANDO FERRAMENTAS DE IOT E COMPUTAÇÃO NA BORDA

Recife 2024

MARIA HELENA ROCHA DE ALENCAR BEZERRA

CONTROLE DE ACESSO A AMBIENTES FÍSICOS UTILIZANDO FERRAMENTAS DE IOT E COMPUTAÇÃO NA BORDA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Bacharel em Engenharia de Controle e Automação.

Orientador(a): Prof. Dr. Geraldo Leite Maia Junior

Recife 2024

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

Bezerra, Maria Helena Rocha de Alencar.
Controle de acesso a ambientes físicos utilizando ferramentas de IoT e computação na borda / Maria Helena Rocha de Alencar Bezerra. - Recife, 2024.
85 p. : il.

Orientador(a): Geraldo Leite Maia Junior
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia de Controle e Automação - Bacharelado, 2024.
Inclui referências, apêndices.

1. controle de acesso. 2. Raspberry Pi. 3. Supabase. 4. Expo. 5. Internet of Things (IoT). I. Maia Junior, Geraldo Leite . (Orientação). II. Título.

620 CDD (22.ed.)

MARIA HELENA ROCHA DE ALENCAR BEZERRA

CONTROLE DE ACESSO A AMBIENTES FÍSICOS UTILIZANDO FERRAMENTAS DE IOT E COMPUTAÇÃO NA BORDA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Bacharel em Engenharia de Controle e Automação.

Aprovado em: 22/03/2024.

BANCA EXAMINADORA

_____

Prof. Dr. Geraldo Leite Maia Junior
Universidade Federal de Pernambuco

_____

M.Sc Eduardo Augusto Oliveira Barbosa
Universidade Federal de Pernambuco

_____

Prof. Dr. Marcio Rodrigo Santos Carvalho
Universidade Federal de Pernambuco

Este trabalho é dedicado a todos que de alguma forma contribuíram para esta construção. À minha família, pelo apoio incondicional, compreensão e amor durante toda a jornada acadêmica. Aos meus amigos, pela paciência e incentivo nos momentos de desafio. Aos professores, pela orientação, conhecimento compartilhado e inspiração para buscar sempre mais. Aos colegas de curso, pelas trocas de experiências e aprendizados. E a todas as pessoas que, de alguma maneira, colaboraram para a realização deste trabalho, o meu mais sincero agradecimento.

AGRADECIMENTOS

Primeiramente, gostaria de expressar minha profunda gratidão à Thaís Rodrigues, que contribuiu ativamente na pesquisa desta aplicação e esteve sempre presente nos momentos bons e nos difíceis. Aos meus pais, Marta e Luiz, que sempre apoiaram meus estudos e foram fontes inesgotáveis de amor e incentivo. A minha irmã Ana Rosa e Nathália que sempre acreditaram em mim.

Gostaria de estender meus agradecimentos a Vitor Dantas e Beatriz Lopes, que deram início à concepção deste trabalho. Agradeço também a Luis Felipe Alves, meu amigo e monitor de redes, que contribuiu para o aumento do meu interesse no assunto e sempre esteve disponível para compartilhar conhecimento. À Thaís Rodrigues novamente, bem como a Rodrigo Bezerra, Ingrid Carolinne, Ana Carolinna Tavares, Caio Lins, André Luiz, Maiara Nunes, Vitor Dantas e tantos outros amigos que contribuíram durante toda a jornada da minha graduação, seja com apoio moral, ajuda em projetos ou simplesmente com companhia nos momentos de estudo. Aos professores que tiveram a paciência e o interesse de ensinar e tirar minhas dúvidas, meu sincero agradecimento. Em especial, destaco o apoio e a orientação dedicados do professor Geraldo Maia, cujo conhecimento e orientação foram fundamentais para o desenvolvimento deste trabalho.

Agradeço também a Filipe, meu melhor amigo, por sempre se preocupar com meu bem-estar durante a graduação e por ser um apoio constante. Ao Maracatronics e toda a equipe, que contribuíram para o meu crescimento profissional, acadêmico e pessoal. Ao Diretório Acadêmico, que me mostrou a importância e os desafios de liderar, e me fez crescer nesse aspecto. Gostaria de estender meus agradecimentos a todos os meus familiares que sempre me apoiaram Por fim, mas não menos importante, agradeço aos meus avós, Luiz, Rosa e Laura, que sempre apoiaram todos os passos da minha jornada desde antes de meu nascimento, sendo exemplos de amor e dedicação. Se por acaso esqueci de mencionar alguém, peço sinceras desculpas, pois cada contribuição foi fundamental para minha trajetória acadêmica e pessoal.

O treinamento de usuários consiste em parte do processo de educação, em base repetitiva, compreende ações e/ou estratégias para desenvolver determinadas habilidades ou habilidades específicas do usuário por desconhecer situações específicas de uso da biblioteca e seus recursos informacionais, que envolvem o conjunto de meios necessários para tal (DIAS; PIRES, 2004, p. 36).

RESUMO

A necessidade de enfrentar desafios no Departamento de Engenharia Elétrica (DEE) da UFPE, incluindo o acesso indiscriminado às suas instalações, a ausência de controle sobre as pessoas que entram nas salas e laboratórios, bem como o registro de furtos de equipamentos. Além disso, o aumento significativo no número de dispositivos conectados, projetado para superar os 75 bilhões até 2025, (ZUBAIR SHARIF, 2022) impulsionou a demanda por uma gestão mais eficiente e segura dos espaços físicos. Este trabalho se concentra no desenvolvimento de um aplicativo de controle de acesso de baixo custo, operando de forma independente da conexão à internet e abrangendo a criação da interface do aplicativo, o desenvolvimento do banco de dados e a implementação de funcionalidades para acesso em ambientes físicos. Para alcançar esses propósitos, foram utilizadas plataformas como Expo para o desenvolvimento do aplicativo mobile, o Supabase como serviço de back-end, e os protocolos de comunicação HyperText Transfer Protocol (HTTP), Message Queuing Telemetry Transport (MQTT) e *WebSocket* responsáveis pelas requisições necessárias para acesso aos ambientes. A *Raspberry Pi* foi escolhida como *hardware* de baixo custo para implementar a aplicação. Durante os testes, uma rede local foi estabelecida integrando diversos componentes, como *Raspberry Pi*, *Expo*, *Supabase*, *Expo Go*, *Node-RED* e *broker MQTT*. Algumas incompatibilidades foram identificadas, sendo necessário adicionar um intermediário para garantir a transferência adequada de dados. O resultado final demonstrou efetividade na garantia do controle de acesso proposto para automação na borda utilizando ferramentas de internet das coisas (do inglês, Internet of Things - (IoT)).

Palavras-chave: controle de acesso; *Raspberry Pi*; *Supabase*; *Expo*; *Internet of Things* (IoT).

ABSTRACT

The need to address challenges in the Electrical Engineering Department (DEE) at UFPE, including indiscriminate access to its facilities, lack of control over individuals entering rooms and laboratories, as well as equipment theft records. Additionally, the significant increase in the number of connected devices, projected to surpass 75 billion by 2025 (ZUBAIR SHARIF, 2022), has driven demand for more efficient and secure management of physical spaces. This work focuses on developing a low-cost access control application, operating independently of internet connection and encompassing the creation of the application interface, database development, and implementation of functionalities for physical access. To achieve these purposes, platforms such as Expo for mobile app development, Supabase as a back-end service, and communication protocols such as HyperText Transfer Protocol (HTTP), Message Queuing Telemetry Transport (MQTT) and WebSocket responsible for the necessary requests for access to environments were utilized. Raspberry Pi was chosen as the low-cost hardware to implement the application. During testing, a local network was established integrating various components like Raspberry Pi, Expo, Supabase, Expo Go, Node-RED, and MQTT broker. Some incompatibilities were identified, necessitating the addition of an intermediary to ensure proper data transfer. The final outcome demonstrated effectiveness in ensuring the proposed access control for edge automation using Internet of Things (IoT) tools.

Keywords: *access control; Raspberry Pi; Supabase; Expo; Internet of Things (IoT).*

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
API	*Application Programming Interface*
APK	*Android Package Kit*
ARM	*Advanced RISC Machine*
AWS	*Amazon Web Services*
BaaS	*Backend-as-a-Service*
FAT32	*File Allocation Table 32*
FTP	*File Transfer Protocol*
GB	*GigaBytes*
GPIO	*General Purpose Input/Output*
HD	*Hard Disk*
HDMI	*High-Definition Multimedia Interface*
HTTP	*HyperText Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
IBM	*International Business Machines Corporation*
ID	*Identity*
IDFV	*Identifier For Vendors*
iOS	*iPhone Operating System*
IoT	*Internet of Things*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
JVM	*Java Virtual Machine*
JWT	*JSON Web Tokens*
LAN	*Local Area Network*
LED	*Light-Emiting Diode*
LTS	*Long-term support*
MicroSD	*Micro Secure Digital*
MQTT	*Message Queuing Telemetry Transport*
NFC	*Near Field Communication*
NIST	*National Institute of Standards and Technology*
NoSQL	*Not Only Structured Query Language*

NPM	*Node Package Manager*
NPX	*Node Package Executor*
QR	*Quick Response*
RAM	*Random Access Memory*
RBAC	*Role-Based Access Control*
RISC	*Reduced Instruction Set Computing*
SDK	*Software Development Kit*
SGBD	Sistemas de Gerenciamento de Banco de Dados
SMTP	*Simple Mail Transfer Protocol*
SQL	*Structured Query Language*
SSD	*Solid State Drive*
SSL	*Secure Sockets Layer*
TCP	*Transmission Control Protocol*
TLS	*Transport Layer Security*
UDP	*User Datagram Protocol*
USB	*Universal Serial Bus*
UUID	*Universally Unique Identifier*
WAN	*Wide Area Network*

SUMÁRIO

1 INTRODUÇÃO

Na atualidade, se convive constantemente com sistemas de controle de acesso. Nas residências, por exemplo, há o controle físico realizado por chaves e fechaduras, ou mesmo por fechaduras digitais. Nos transportes públicos, se encontram catracas e cartões de proximidade. Ao utilizar aplicativos bancários, a autenticação ocorre por meio de senha e tokens de segurança. Com o *Role Based Access Control* (RBAC), é possível atribuir permissões com base nas funções dos usuários. Esses exemplos fazem parte do controle de acesso físico e lógico.

Com as evoluções tecnológicas, novos sistemas surgem, possibilitando a integração com vigilância por vídeo, IoT, análise de dados e aprendizado de máquina, soluções hospedadas na nuvem, aplicações móveis, reconhecimento de voz, registro de acesso, entre outros.

Com inúmeras tecnologias disponíveis, a escolha do sistema de controle de acesso dependerá de requisitos específicos do ambiente em que será utilizado, do tipo de usuário que utilizará a plataforma, do nível de segurança desejado, entre outras particularidades.

O controle de acesso é definido como um conjunto de tecnologias e práticas implementadas para restringir a entrada ou uso de locais ou sistemas. A relevância desse conceito tornou-se evidente durante a Segunda Guerra Mundial em instalações militares críticas (MICHAEL E. WHITMAN, 2014). Como resultado, tornou-se crucial assegurar que apenas entidades autorizadas tenham acesso a áreas específicas.

Existem dois tipos principais de controle de acesso: o físico e o lógico. O controle físico está relacionado à regulação da entrada em espaços físicos, utilizando métodos como chaves e fechaduras, cartões de acesso, biometria física, catracas, entre outros. Por outro lado, o controle lógico abrange recursos digitais, sistemas de informação ou dados, utilizando métodos como a autenticação de senha, *tokens* de segurança, certificado digital, biometria lógica e Controle de Acesso Baseado em Função (do inglês, *Role-Based Access Control* (RBAC)).

Os métodos tradicionais de controle de acesso envolvem abordagens convencionais, como chaves físicas, cartões magnéticos e identificação por crachá.

Essas estratégias são suscetíveis a perdas, falsificações, cópias não autorizadas e dificuldades de gestão. Em contraste, as abordagens modernas incorporam tecnologias avançadas e inovações. Exemplos dessas abordagens incluem autenticação biométrica, sistemas de controle de acesso eletrônico, aplicações de controle de acesso na nuvem e uso de dispositivos móveis. Essas abordagens modernas são menos propensas a falsificações, proporcionam gestão eficiente, permitem monitoramento em tempo real, oferecem flexibilidade, mobilidade, entre outros benefícios.

Este trabalho apresenta uma análise minuciosa sobre o controle de acesso em ambientes físicos, abrangendo desde conceitos fundamentais até tecnologias emergentes. É destacado a integração de sistemas contemporâneos de controle de acesso com tecnologias presentes em *smartphones* e internet das coisas, o que deve impulsionar uma maior eficiência operacional.

Além disso, será explorada a inclusão de dispositivos móveis. A incorporação desses dispositivos deverá oferecer um potencial para melhora do controle de acesso, mas também apresenta desafios que precisarão ser cuidadosamente gerenciados.

A análise se estenderá à arquitetura de redes, onde será abordado o hardware, software e paradigmas cliente-servidor, enfatizando a importância da escolha da arquitetura na eficiência operacional. A seleção da arquitetura de redes exemplificada poderá influenciar diretamente na eficácia deste trabalho.

Os protocolos de comunicação, *HyperText Transfer Protocol* (HTTP) e *Message Queuing Telemetry Transport* (MQTT), serão discutidos e a escolha e implementação segura dos mesmos deverão ser vitais para garantir a integridade e a segurança das informações durante o controle de acesso.

Na análise abrangente dos bancos de dados, será destacado o *PostgreSQL* como escolha robusta para ambientes relacionais, enquanto o banco de dado *Not Only Structured Query Language* (NoSQL) será abordado como alternativa flexível. A seleção do banco de dados mencionado deverá ser o mais adequado e poderá influenciar diretamente no sucesso do controle de acesso.

Outras seções, como plataformas de desenvolvimento de aplicativos *mobile*, linguagens de programação, serviços de back-end e segurança da informação, serão

exploradas com a hipótese de que cada uma dessas áreas desempenha um papel crucial e contribui para uma visão abrangente e integrada do controle de acesso em ambientes físicos e tecnológicos, desde os fundamentos até as inovações mais recentes.

Este trabalho deve optar pela abordagem do controle de acesso em ambientes físicos através do uso de dispositivos móveis.

1.1 Objetivos

1.1.1 Geral

Aprimorar o controle de acesso em ambientes por meio da implementação de automação na borda de forma economicamente acessível, utilizando ferramentas de IoT para acionar a abertura de ambientes físicos, combinando conceitos consagrados no desenvolvimento de aplicativos e integrando-os eficientemente em um sistema de baixo custo e fácil aquisição, destinado à aplicação prática em universidades.

1.1.2 Específicos

1.1.2.1 Identificar e selecionar conceitos avançados na produção de aplicativos e IoT que possam ser utilizados no contexto de controle de acesso em ambientes físicos.

1.1.2.2 Escolher e justificar a utilização de tecnologias de automação na borda, considerando a viabilidade econômica e a facilidade de aquisição, com ênfase na integração de ferramentas de IoT.

1.1.2.3 Realizar a instalação e configuração adequada do sistema operacional e dos softwares no hardware selecionado (Raspberry Pi) para suportar a automação na borda, incluindo as ferramentas de IoT necessárias.

1.1.2.4 Integrar de forma eficaz os softwares Expo, Supabase e Node-RED, juntamente com as ferramentas de IoT, para criar uma solução coesa e funcional.

1.1.2.5 Definir e implementar protocolos de comunicação (MQTT, HTTP) para garantir a efetiva troca de informações entre os dispositivos no controle de acesso, considerando as peculiaridades das ferramentas de IoT.

1.1.2.6 Desenvolver um aplicativo mobile que permita o controle de acesso aos ambientes de maneira intuitiva e eficiente, integrando-se de forma sinérgica às ferramentas de IoT.

1.1.2.7 Estabelecer uma rede local para condução de testes que simulem condições práticas, verificando a robustez e a confiabilidade do sistema em diferentes cenários, com ênfase na funcionalidade de acionamento via IoT.

1.1.2.8 Avaliar a eficácia da solução proposta, considerando critérios de desempenho, segurança, usabilidade e a integração bem-sucedida das ferramentas de IoT.

1.1.2.9 Propor eventuais melhorias e ajustes na implementação com base nos resultados obtidos durante os testes, visando otimizar a utilização das ferramentas de IoT.

1.1.2.10 Documentar e apresentar de maneira clara e abrangente os resultados e conclusões alcançados no desenvolvimento do projeto de TCC, destacando o papel fundamental das ferramentas de IoT na solução proposta.

1.2 Organização do Trabalho

Esta dissertação está estruturada em duas principais seções: "Fundamentação Teórica (Metodologia)" e "Desenvolvimento do Trabalho". A seção de “Controle de acesso” introduz os conceitos fundamentais relacionados ao controle de acesso, estabelecendo a base para discussões posteriores. Os “Sistemas de controle de

acesso” explora os diferentes tipos de sistemas de controle de acesso, destacando suas características individuais e aplicações práticas.

O tópico “tecnologias para controle de acesso em dispositivos móveis” foca nas tecnologias específicas destinadas ao controle de acesso em celulares, considerando suas implicações e eficácia. Os tópicos subsequentes abordam questões como arquitetura de rede, protocolos de comunicação, modelos de banco de dados, servidores, computação em nuvem, computação na borda, IoT, plataformas de desenvolvimento mobile, linguagens de programação, serviços de back-end, segurança da informação, Raspberry Pi e *Node-RED*.

O “Desenvolvimento do trabalho” detalha o processo prático de desenvolvimento do sistema, abordando aspectos desde a arquitetura do sistema até a implementação de segurança no back-end. No “Desenvolvimento do aplicativo” se dá início ao desenvolvimento prático com a criação do aplicativo, destacando aspectos relacionados à interface do usuário. Os capítulos subsequentes abordam temas como a arquitetura do sistema, configuração do *Raspberry Pi* e ambiente de desenvolvimento, desenvolvimento do front-end, integração do protocolo MQTT e *Node-RED* para controle de acesso, desenvolvimento do banco de dados com *Supabase*, desenvolvimento do back-end com *Supabase* e a implementação de medidas de segurança no back-end. Em “Conclusões e propostas de continuidade” estão as conclusões derivadas do trabalho e sugere propostas para continuidade, incluindo testes e resultados obtidos.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Tecnologias para Controle de Acesso em Dispositivos Móveis

Ao escolher integrar smartphones nos sistemas de controle de acesso, atribui-se a esses dispositivos a responsabilidade de autenticar e autorizar usuários para acessar locais e sistemas específicos. Esses dispositivos podem substituir as credenciais tradicionais, permitindo a adoção de tecnologias acessíveis, versáteis e seguras, como leitura de impressão digital, reconhecimento facial ou outras formas modernas de autenticação.

Dentre as tecnologias presentes nos dispositivos móveis, destacam-se NFC, Bluetooth e Códigos QR. O NFC é uma tecnologia de comunicação a curta distância, segura e simples (CNN, 2023). O Bluetooth, que possui características semelhante ao NFC, permite distâncias um pouco maiores. Já os códigos QR são ubíquos, encontrados em *websites*, anúncios e dispositivos móveis. Criados pela *Denso Wave* em 1994 para rastrear veículos na fabricação, esses códigos retêm dados e oferecem acesso imediato (Um guia sobre códigos QR e como fazer sua leitura, 2023).

Diferentemente dos códigos de barras convencionais, os QR podem ser lidos em duas direções, permitindo armazenar mais informações. Sua fácil leitura e *design* característico contribuíram para rápida adoção na indústria automotiva. A *Denso Wave* tornou o código QR de código aberto, impulsionando sua inserção em vários setores. Em 2002, telefones com leitores de QR foram lançados, marcando uma nova fase de uso generalizado. Atualmente, os códigos QR continuam a evoluir, sendo amplamente empregados em estratégias de marketing, rastreamento e até mesmo em transações financeiras. Seu funcionamento envolve padrões binários interpretados por leitores que analisam e decifram as informações contidas.

A utilização de *smartphones* em sistemas de controle de acesso enfrenta desafios, sendo a segurança uma preocupação significativa. A diversidade de sistemas operacionais e hardwares pode complicar a integração dessas tecnologias. Contudo, as características inerentes de flexibilidade e mobilidade dos dispositivos

móveis surgem como uma oportunidade para aprimorar a eficiência operacional e promover maior acessibilidade.

2.2 ARQUITETURA E PARADIGMAS DE REDE

A arquitetura de rede é um conjunto abrangente de diretrizes e estruturas que define a organização, o design e a implementação de uma rede de computadores. Ela unifica conceitos físicos, como dispositivos e cabos, com elementos conceituais, como protocolos e padrões, para criar uma infraestrutura de comunicação eficaz e segura (TANENBAUM, 2003). Abrange diversos elementos essenciais que colaboram para o funcionamento eficiente de uma rede de computadores. Isso inclui *hardware*, *software*, protocolos e a definição de como os dispositivos são interconectados. No contexto da arquitetura de rede, é crucial considerar tanto os aspectos físicos quanto os conceituais.

No âmbito físico, a arquitetura de rede especifica os dispositivos que compõem a infraestrutura, tais como roteadores, *hubs*, *switches*, cabos e servidores. Esses componentes físicos desempenham papéis distintos, como rotear dados, concentrar conexões, e são capazes de fornecer recursos de processamento, armazenamento e serviços.

Além disso, a arquitetura de rede aborda questões relacionadas à transmissão de dados, determinando como os dados serão transmitidos entre os dispositivos conectados. Isso envolve a escolha de protocolos de comunicação que define, como os dados são fragmentados, transmitidos e reagrupados. Aqui, estão alguns exemplos de arquiteturas de rede:

Ethernet: Amplamente empregado em redes locais (LANs, abreviação de Local Area Network). Fisicamente, sua implementação pode ocorrer através de cabo par trançado, fibra óptica ou cabo coaxial.

Token Ring: Os dispositivos são conectados fisicamente em um anel. A comunicação se estabelece por meio de unidades de dados, conhecidos como "tokens", que são compartilhados entre os dispositivos.

Transmission Control Protocol/Internet Protocol (TCP/IP): O Protocolo de Controle de Transmissão/Protocolo da Internet desempenha um papel central na infraestrutura da Internet e em diversas redes da atualidade. Ela incorpora protocolos essenciais, como IP, TCP e *User Datagram Protocol* (UDP), apresentando quatro camadas do modelo OSI: aplicação, transporte, internet e enlace.

Para uma compreensão abrangente de redes de computadores, é igualmente crucial ter conhecimento sobre os paradigmas de rede. Conceitualmente, esses paradigmas representam formas distintas de organizar a comunicação e a colaboração entre os elementos em uma rede. Através deles, é possível estruturar e coordenar as interações entre os componentes da rede. Alguns exemplos de paradigmas de rede são:

Cliente-servidor: os clientes requisitam recursos ou serviços ao servidor. É comumente utilizado em redes onde os servidores centralizam os dados ou funcionalidades.

Produtor-consumidor: os produtores geram os dados enquanto os consumidores os utilizam. Isso ocorre através de uma fila ou *buffer* onde os dados são depositados com a finalidade de serem retirados. Assim, é possível coordenar os dados de maneira eficiente, evitando a superprodução ou subconsumo.

Mestre-escravo: o mestre, entidade central, envia a requisição e os escravos obedecem. Comumente empregado em sistemas distribuídos, a comunicação é, em sua maioria das vezes, unidirecional.

Publicador-assinante: os publicadores enviam mensagens para um intermediário (broker ou servidor de mensagens). As mensagens são publicadas em tópicos. Os assinantes recebem mensagens de um ou mais tópicos específicos. Quando uma nova mensagem é publicada em um tópico, o broker automaticamente a encaminha para todos os assinantes daquele tópico.

Para definir qual arquitetura e paradigma devem ser utilizados, é necessário avaliar quais os requisitos da aplicação e observar os prós e contras de cada opção. Neste trabalho o paradigma de rede escolhido foi o Cliente-Servidor” utilizando o padrão Ethernet.

2.3 PROTOCOLO DE COMUNICAÇÃO

Antes de prosseguir, é importante definir a palavra protocolo, que nada mais é que um conjunto de regras e padrões. Existe protocolo para tudo, numa conversa entre duas pessoas, por exemplo, é necessário que uma fale e a outra escute e que essa troca ocorra ao longo do ato. Um protocolo de comunicação não está muito longe disso, podemos enxergar como a forma padrão de transmitir, formatar e receber dados entre dois ou mais dispositivos.

É importante que um protocolo de comunicação proporcione confiabilidade e segurança nas trocas de dados, isso é feito através das regras definidas para cada protocolo. É muito importante garantir a correção de erros e integridade dos dados. Existem diferentes tipos de protocolos, alguns exemplos muito conhecidos são o TCP/IP, HTTP, Protocolo de Transferência de Correio Simples, do inglês *Simple Mail Transfer Protocol* (SMTP), Protocolo de Transferência de Arquivos do inglês *File Transfer Protocol* (FTP), MQTT, entre outros.

Protocolos de comunicação são padrões que os dispositivos e sistemas utilizam para se comunicarem, são essenciais para permitir a conexão dos dispositivos em rede. A seguir, são apresentados os protocolos que foram utilizados neste trabalho:

2.3.1 HTTP

O HTTP é um protocolo utilizado para a comunicação entre navegadores *web* e servidores. Ele oferece uma estrutura bem definida para a transferência de dados, especialmente projetada para a navegação na web. O HTTP opera no modelo cliente-servidor, onde o navegador atua como cliente, fazendo solicitações, e o servidor responde com os dados ou recursos solicitados (Protocolo MQTT: O Que é, Como Funciona e Vantagens, 15/03/2023).

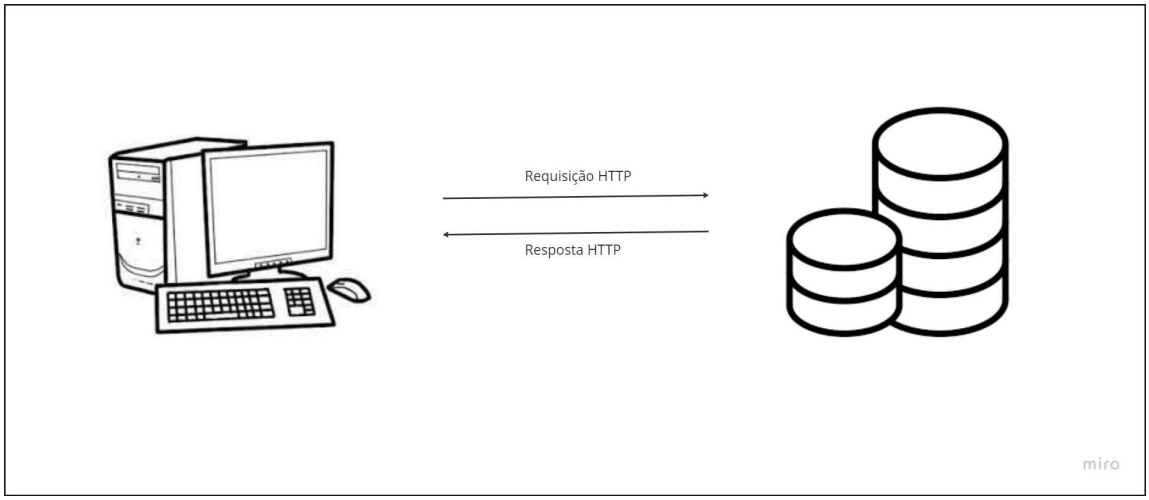
Apesar de proporcionar uma comunicação estruturada, o HTTP apresenta algumas limitações, principalmente no que diz respeito à segurança. As informações transferidas entre o cliente e o servidor, como dados de formulários ou credenciais, são enviadas em texto simples, o que pode tornar vulnerável a interceptação por parte

de terceiros mal-intencionados. Isso é particularmente preocupante em ambientes onde a privacidade e a segurança dos dados são essenciais.

Adicionalmente, a complexidade da comunicação no HTTP pode ser considerada um desafio em certos contextos. A comunicação baseada em texto e a necessidade de várias requisições para carregar uma única página *web* podem resultar em um desempenho moderado, especialmente em ambientes com largura de banda limitada.

Para abordar essas limitações, surgiu o *Hypertext Transfer Protocol Secure* (HTTPS), uma extensão do HTTP que utiliza uma camada adicional de segurança através do protocolo *Secure Sockets Layer*/*Transport Layer Security* (SSL/TLS). O HTTPS criptografa os dados transmitidos, proporcionando maior privacidade e proteção contra ataques de interceptação. A adoção crescente do HTTPS reflete a importância dada à segurança na comunicação *web*.

Figura 1 - Arquitetura de publicação/assinatura do HTTP.



Fonte: O'Autor.

2.3.2 MQTT

Outro protocolo que surgiu mediante a necessidade de uma comunicação mais simples e segura foi o *Message Queuing Telemetry Transport* (MQTT). Na década de

90, a *International Business Machines Corporation* (IBM) junto com a Eurotech, criam o MQTT (em português, Transporte de Filas de Mensagem de Telemetria).

Ele é comumente utilizado em IoT. Como o próprio nome sugere, é um protocolo de mensagens, leve e projetado para dispositivos restritos e redes com largura de banda mínima, alta latência ou não confiáveis. Esse protocolo é utilizado em dispositivos pequenos que possuem recursos limitados e necessitam se comunicar uns com os outros ou com servidores de back-end (Protocolo MQTT: O Que é, Como Funciona e Vantagens, 15/03/2023).

O MQTT é um protocolo projetado para funcionar como um transporte de mensagens de publicação/assinatura, portanto, os dispositivos ou aplicativos podem publicar mensagens para um intermediário central, broker, que entrega essas mensagens a todos os clientes inscritos, como mostrado na Figura 2. Apresenta, também, uma confiabilidade na entrega das mensagens que é importante para usos em IoT, e 3 níveis de qualidade de serviço definidos: no máximo uma vez, pelo menos uma vez e exatamente uma vez. Ele é baseado em TCP/IP e outros protocolos, utiliza um formato binário simples para codificação de mensagens, o que o torna eficiente em termos de largura de banda e poder de processamento.

Os principais usos do MQTT são:

Comunicação IoT: é frequentemente utilizado em aplicativos IoT, pois possui uma maneira leve e eficiente comunicação entre dispositivos ou servidores back-end. Pode ser utilizado em sistemas domésticos ou prediais inteligentes, para permitir que sensores e atuadores se comuniquem com serviços de nuvem ou um hub central.

Automação industrial: fornece uma maneira eficiente e confiável para a troca de dados de dispositivos e sistemas. Um exemplo do seu uso é em um sistema de automação de fábrica para controlar e monitorar máquinas.

Mensagens em tempo real: o MQTT é capaz de enviar mensagens em tempo real, em chats ou serviços de mensagens instantâneas. Portanto, pode ser utilizado para distribuir mensagens de forma escalável e eficiente entre clientes.

É usado também no setor automotivo, de logística, manufatura, lar inteligente, produtos de consumo e transporte. Alguns exemplos de implementações MQTT incluem:

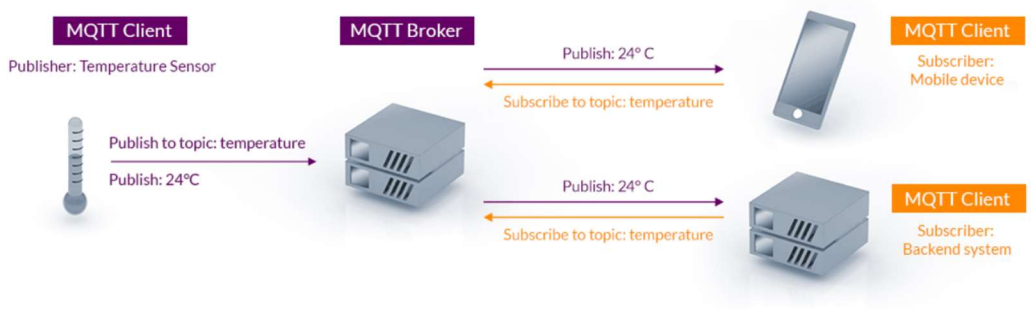
Eclipse Mosquitto: um *broker* MQTT de software livre utilizado na construção de aplicativos baseados em MQTT.

Amazon Web Services IoT Core: serviço baseado em nuvem que fornece comunicação baseada em MQTT para dispositivos e aplicativos IoT.

HiveMQ: um *broker* MQTT comercial que fornece recursos avançados, como *clustering* e alta disponibilidade.

Home Assistant: uma plataforma de automação residencial de código aberto que oferece suporte à integração MQTT para conectar dispositivos e sistemas.

Figura 2 – Arquitetura de publicação/assinatura do MQTT.



Fonte: (MQTT, 2022).

É crucial abordar a relação entre cliente e broker no contexto do MQTT. Neste cenário, o broker atua como um servidor centralizado, responsável pela gestão e distribuição de mensagens entre os dispositivos clientes. Estes, por sua vez, podem assumir o papel de publicadores, enviando mensagens associadas a um tópico específico, ou de assinantes, aguardando por mensagens de tópicos de seu interesse. Ao receber uma mensagem de um cliente publicador, o broker a encaminha para todos os clientes assinantes daquele tópico específico. Tal mecanismo promove uma notável escalabilidade e flexibilidade nas aplicações, permitindo, adicionalmente, a implementação de diversos níveis de qualidade de serviço. Essa estrutura assegura que até mesmo dispositivos com recursos limitados sejam capazes de estabelecer uma conexão confiável com o broker, garantindo assim a eficácia da comunicação dentro de redes potencialmente instáveis ou de variada capacidade.

2.4 Banco de Dados

Na década de 60, o conceito de banco de dados emergiu devido à percepção das empresas sobre os altos custos de empregar um grande número de pessoas para armazenar e organizar dados. Por isso, grandes organizações como a IBM, não mediram esforços e investimentos para criar e produzir uma plataforma inovadora e eficiente capaz de estruturar e organizar os dados, (A História dos Banco de Dados). Inicialmente os modelos apresentavam relações de muitos para muitos, suas representações eram feitas em caixas e linhas, Figura 3 e seus relacionamentos eram feitos por *links*. Outra forma de organização era através do modelo hierárquico, sua representação se assemelhava a uma árvore, Figura 4.

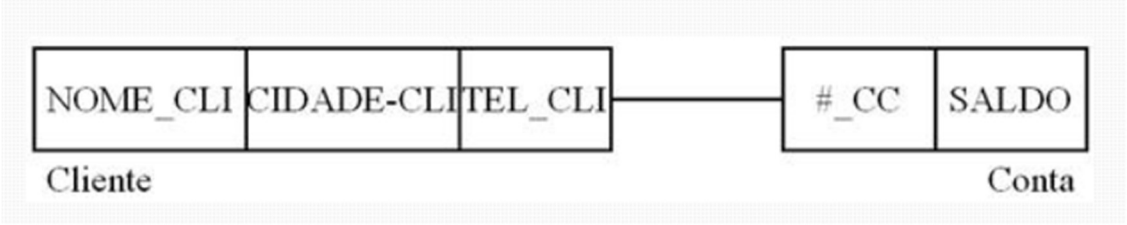
2.4.1 Modelo Relacional

Na década de 70, foi proposto o conhecido Modelo Relacional, também foi criado o modelo de Entidade Relacionamento, e, finalmente, na metade dos anos 80, a Linguagem estruturada de consulta, conhecida como *Structured Query Language* (SQL) se torna padrão mundial.

Um banco de dados nada mais é do que uma coleção de dados organizados de forma estruturada para que possam ser acessados, gerenciados e atualizados de acordo com cada necessidade, (ELMASRI, 2019). Isso permite uma confiabilidade e precisão dos dados. Uma das principais razões de utilizar banco de dados é reduzir as redundâncias.

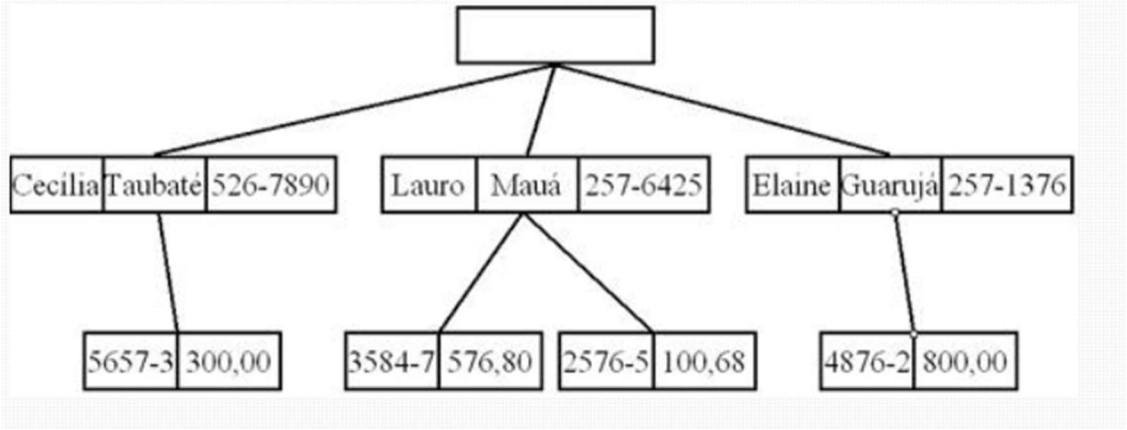
Juntamente com banco de dados, normalmente são utilizados os SGBDs (Sistemas de Gerenciamento de Banco de Dados). Esses sistemas ajudam a criar, manter e gerenciar os bancos de dados.

Figura 3 - Representação de um modelo de dados em rede.



Fonte: adaptado de (A História dos Banco de Dados).

Figura 4 - Representação de um modelo de dados hierárquico.



Fonte: adaptado de (A História dos Banco de Dados).

Bancos de dados relacionais desempenham um papel vital na gestão de dados estruturados, oferecendo consistência e integridade. Dentre os exemplos notáveis, destacam-se o *MySQL*, conhecido por sua confiabilidade, *Oracle Database*, pioneiro em escalabilidade, e o *Microsoft SQL Server*, integrado ao ecossistema *Microsoft*.

No cenário de código aberto, o *PostgreSQL* se destaca como uma opção notável. Reconhecido por sua flexibilidade, confiabilidade e extensibilidade, ele oferece suporte a diversos tipos de dados e recursos avançados, tornando-se uma escolha popular em várias indústrias, (GRUPO DE DESENVOLVIMENTO GLOBAL POSTGRESQL, 2024).

O *PostgreSQL* é conhecido por suportar transações ACID (atomicidade, consistência, isolamento e durabilidade), garantindo a consistência e integridade dos

dados. Sua arquitetura extensível permite a criação de tipos de dados e funções personalizadas, proporcionando flexibilidade para atender a diversas necessidades.

Além de ser compatível com SQL padrão, o *PostgreSQL* se integra facilmente com tecnologias modernas. A capacidade nativa de suportar dados de Notação para Objetos em *Javascript*, do inglês *JavaScript Object Notation* (JSON), oferece flexibilidade para aplicações que lidam com dados não estruturados, agregando versatilidade ao sistema.

Apesar de sua robustez, o *PostgreSQL* pode demandar um conhecimento técnico mais aprofundado em comparação com sistemas mais simples. Em cenários de alta concorrência, otimizações podem ser necessárias para assegurar um desempenho ideal.

Grandes empresas como *Apple*, *Cisco* e *Fujitsu* confiam no *PostgreSQL* para atender as suas necessidades de banco de dados. Sua comunidade ativa e engajada contribui para seu desenvolvimento contínuo, solidificando-o como uma escolha confiável para uma ampla gama de projetos.

2.4.2 Modelo não relacional

Além dos tradicionais bancos de dados relacionais, existe uma categoria de sistemas conhecida como bancos de dados não relacionais ou NoSQL. Os bancos de dados NoSQL foram desenvolvidos para atender a requisitos específicos que muitas vezes não são totalmente abordados pelos modelos relacionais, (SADALAGE e FOWLER, 2012).

Ao contrário dos modelos relacionais, os bancos de dados NoSQL não seguem um esquema fixo e não requerem uma estrutura de tabelas predefinida. Isso proporciona flexibilidade significativa para lidar com dados variados e em constante evolução, como é comum em ambientes modernos de desenvolvimento de software. Alguns exemplos de bancos não relacionais são: *MongoDB*, *CouchDB*, *Redis*, *DynamoDB*, *Cassandra*, *HBase*, entre outros.

Neste projeto, foi decidido utilizar um banco de dados relacional, e entre as opções consideradas, o *PostgreSQL* destacou-se como a melhor alternativa. Essa

escolha também foi motivada pela familiaridade com o mesmo, o que facilita o desenvolvimento e a manutenção do sistema.

2.5 Servidores - Arquitetura, *Hardware*, *Software* e Características

Os servidores representam elementos cruciais na infraestrutura de redes, sendo computadores dedicados com recursos robustos e funcionalidades específicas. Compostos por processadores, bancos de memória, portas de comunicação, softwares e sistemas de armazenamento de dados, como *Solid State Drives* (SSDs) ou *Hard Disks* (HDs), esses sistemas desempenham um papel central na execução de serviços em redes LAN ou redes de longa distância, em inglês *Wide Area Networ* (WAN), (O que é um servidor em computação?, 2024).

Na arquitetura cliente-servidor, os servidores atuam como pontos centrais, respondendo às solicitações dos dispositivos clientes. Essa dinâmica permite a execução centralizada de programas, bem como o armazenamento e compartilhamento eficiente de arquivos em redes locais e remotas. Os servidores locais podem ser implementados em diferentes formatos, como *rack*, *blade* ou torre, e são reconhecidos como servidores dedicados.

No que diz respeito ao sistema operacional, servidores frequentemente utilizam distribuições *Linux*, como *Red Hat*, ou diversas versões do *Windows Server* da *Microsoft*. Esses sistemas operacionais são escolhidos de acordo com as necessidades específicas do ambiente e das aplicações a serem executadas.

As funções dos servidores abrangem uma ampla gama de atividades, desde a execução de aplicativos corporativos, como bancos de dados e virtualização, até o controle preciso do acesso às informações. Por meio da criação de contas e senhas, os servidores desempenham um papel crucial na segurança e na administração efetiva de recursos, contribuindo significativamente para o funcionamento eficiente de redes empresariais.

Além dos aspectos técnicos, é fundamental considerar os custos associados à implementação e manutenção de servidores. O investimento em servidores envolve não apenas a aquisição de *hardware*, como processadores, memória e

armazenamento, mas também licenças de *software*, custos operacionais e despesas relacionadas à infraestrutura física.

A seleção do sistema operacional também pode influenciar os custos, uma vez que algumas distribuições *Linux* são de código aberto, enquanto sistemas *Windows* podem envolver custos adicionais de licenciamento.

2.6 Computação em nuvem

A computação em nuvem introduziu uma transformação significativa na prestação de serviços de tecnologia da informação, fundamentada em princípios que reformularam a abordagem das organizações em relação à infraestrutura e administração de recursos computacionais. Inicialmente impulsionada por pioneiros como *Salesforce.com* e Amazon Web Services (AWS), a computação em nuvem é caracterizada pela disponibilidade sob demanda de recursos escaláveis, abrangendo armazenamento, capacidade de processamento e funcionalidades empresariais. O modelo, formalizado pelo *National Institute of Standards and Technology* (NIST), descreve a computação em nuvem como um paradigma que oferece acesso à rede, agrupamento de recursos configuráveis e provisionamento dinâmico. Essa abordagem, apoiada por avanços tecnológicos como virtualização e clusters, é impelida por fatores como planejamento de capacidade, redução de custos e agilidade organizacional, conferindo eficiência, flexibilidade e capacidade de adaptação às flutuações nas demandas empresariais, (THOMAS ERL, 2013).

Como dito anteriormente, a implementação de controle de acesso proporciona significativas vantagens, como o aprimoramento da segurança, a flexibilidade para adaptação às necessidades em evolução e a automatização de processos, otimizando a eficiência operacional. No entanto, nesse processo não está isento de desafios, incluindo custos iniciais substanciais, complexidade na integração com sistemas existentes e a necessidade contínua de manutenção e atualizações.

Dentre os serviços relevantes para uma implementação bem-sucedida, destacam-se a consultoria em segurança para avaliação de riscos, o desenvolvimento de software personalizado adaptado às exigências específicas, a integração eficaz de

sistemas, programas de treinamento e conscientização para usuários, e monitoramento contínuo para identificação proativa de ameaças à segurança. Esses elementos combinados buscam equilibrar as vantagens e desvantagens, proporcionando uma abordagem abrangente para fortalecer o controle de acesso em ambientes corporativos.

2.7 Computação na Borda e Acesso Remoto em Aplicações

Neste contexto, a computação na borda engloba dispositivos periféricos estrategicamente posicionados nas extremidades da rede, como sensores e câmeras. Sua principal função é coletar dados relevantes do ambiente circundante e transmiti-los para serem processados de maneira descentralizada, diretamente na borda da rede. Essa abordagem descentralizada proporciona vantagens significativas, como a redução da latência na tomada de decisões, otimizando a eficiência do processamento local.

A computação na borda surge como uma resposta às necessidades corporativas de otimizar a proteção dos colaboradores, proporcionando um tempo de resposta aprimorado. Essa abordagem viabiliza a computação em tempo real, controlando o volume de dados gerados pelos dispositivos atuais pela rede, evitando custos elevados, (MICROSOFT, 2024).

Nesse contexto, dispositivos têm a capacidade de acessar dados na borda da rede por meio da comunicação com o servidor local. Os benefícios da computação na borda incluem operações mais eficientes, tempos de resposta mais rápidos, maior produtividade dos colaboradores, segurança no local de trabalho, funcionalidade em locais distantes, segurança reforçada, sincronia de dados e redução de custos de TI.

Os componentes de *hardware* na computação na borda compreendem dispositivos de borda, processadores, *clusters*/servidores, *gateways*, roteadores, comutadores e nós. Esses *hardwares* apresentam características como sistemas com e sem ventilação, resistência a temperatura e movimentos bruscos, amplo armazenamento, compatibilidade diversa, conectividade variada e suporte a vários tipos de entrada de energia. Além disso, são projetados para oferecer proteção eficaz

contra ataques cibernéticos, reforçando a segurança da infraestrutura. A seguir serão definidos e abordados alguns dos componentes.

Processadores: Unidades de processamento responsáveis por executar operações e processar dados. Na computação na borda, processadores eficientes são essenciais para o rápido processamento local.

Clusters/Servidores: Agrupamento de servidores ou um servidor dedicado que executa tarefas de processamento e armazenamento local de dados, oferecendo suporte a aplicações na borda da rede.

Gateways: Dispositivos que facilitam a comunicação entre dispositivos na borda e servidores centrais, gerenciando o tráfego de dados bidirecional.

Roteadores: Responsáveis pela transmissão de dados entre diferentes redes, conectando dispositivos na borda à infraestrutura de rede mais ampla.

Comutadores: Equipamentos que conectam dispositivos em uma rede local, permitindo a troca eficiente de dados entre eles.

Nós: Dispositivos individuais, como computadores ou sensores, que compõem uma rede distribuída e contribuem para a coleta e processamento descentralizado de dados.

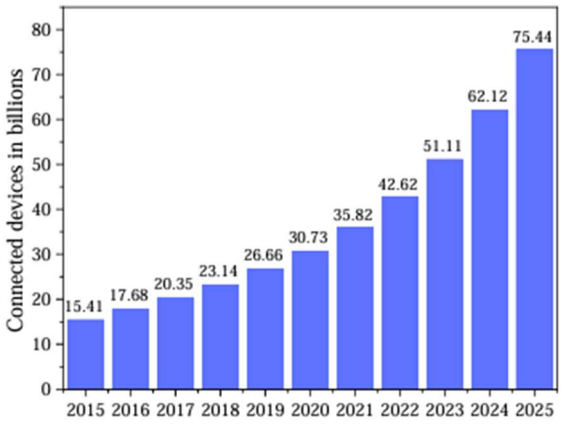
2.8 IoT e Automação

A evolução do controle de acesso é impulsionada pela integração de tecnologias inovadoras, especialmente a IoT. Nesse contexto, métodos avançados de autenticação e autorização, utilizando dispositivos inteligentes e sensores conectados, destacam-se pela eficiência e segurança. Em 2022, foram registrados 42 milhões de dispositivos conectados, projetando mais de 75 milhões até 2025, como mostrado na Figura 5. A aplicação prática da IoT na automação do controle de acesso permite uma abordagem dinâmica, adaptável e eficaz, lidando com cenários complexos e fortalecendo a segurança.

A integração de dispositivos automatizados, como parte da IoT, destaca-se nesse avanço. A presença de sensores fornece dados precisos para validar a

identidade dos usuários, formando uma rede inteligente capaz de responder dinamicamente a demandas variáveis de autenticação. Essa abordagem dinâmica é essencial para lidar com cenários complexos, promovendo eficiência e segurança. O uso crescente dessas tecnologias reflete a necessidade de soluções avançadas no controle de acesso, onde a IoT se destaca como componente crucial para garantir segurança e otimizar processos em sistemas modernos.

Figura 5 - O número atual e esperado de dispositivos conectados.



Fonte: (ZUBAIR SHARIF, 2022)

2.9 Plataformas de Desenvolvimento de Aplicativos *Mobile*

As plataformas de desenvolvimento são peças-chave na criação de aplicativos e sistemas inovadores, variando em características desde linguagens suportadas até conjuntos de desenvolvimento e integração com serviços em nuvem. A escolha depende dos requisitos de projeto, podendo envolver ambientes visuais intuitivos ou ser orientada a programadores mais experientes (BITTENCOURT, 2021).

A integração com tecnologias IoT e computação na borda tornou-se uma prioridade para as plataformas modernas, oferecendo Interfaces de Programação de Aplicação, do inglês *Application Programming Interface* (API), e ferramentas para facilitar o desenvolvimento de aplicativos que interagem eficientemente com dispositivos IoT. No entanto, essa evolução não está isenta de desafios, como

problemas de compatibilidade e complexidades ao integrar dispositivos IoT, exigindo garantia de interoperabilidade e segurança.

Diversas plataformas exemplares se destacam, como o *Android Studio* para desenvolvimento *Android*, o *Xcode* da *Apple* para aplicativos *iPhone Operating System* (iOS), e o *Microsoft Visual Studio*, versátil no suporte a várias linguagens. A escolha entre elas dependerá das características específicas do projeto e das metas estabelecidas pela equipe de desenvolvimento.

O *Expo* é uma plataforma que simplifica a criação de aplicativos móveis, utilizando *JavaScript* e *React Native*. Sua abordagem visual intuitiva torna o desenvolvimento acessível, facilitando a escrita de código em *JavaScript* e permitindo a criação de aplicativos para *Android* e iOS com uma base de código compartilhada (Expo, 2024).

A integração do *Expo* com tecnologias como IoT e computação na borda se destaca, possibilitando a criação de aplicativos eficientes para dispositivos conectados. Apesar de sua eficiência, o *Expo* apresenta desafios e limitações em personalização avançada e integração com funcionalidades específicas de dispositivos, sendo mais adequado para projetos menos complexos.

O *Expo* é exemplificado pelo aplicativo “*Expo Go*”, que atua como ambiente de teste durante o ciclo de desenvolvimento. Essa demonstração prática revela a eficácia do *Expo* em oferecer uma solução completa, produzindo resultados integrados para ambas as plataformas, *Android* e iOS.

2.10 Explorando Linguagens de Programação para Desenvolvimento de Aplicativos *Mobile*

No processo de desenvolvimento de aplicativos *mobile*, é crucial selecionar as linguagens de programação adequadas para construir tanto a parte frontal (front-end) quanto a traseira (back-end) da aplicação. A seguir, serão apresentados alguns exemplos dessas linguagens.

Swift é uma linguagem de programação utilizada no desenvolvimento de aplicativos nativos para dispositivos *Apple*, como *iPhones*, *iPads* e *Macs*.

Caracterizada por uma sintaxe limpa e moderna, a *Swift* oferece recursos notáveis de segurança e desempenho, além de possuir código aberto, (APPLE INC, 2024).

Kotlin, desenvolvida pela *JetBrains*, destaca-se no cenário de desenvolvimento *Android*, surgindo como uma alternativa ao *Java*. Sua interoperabilidade com *Java* contribui para sua popularidade e aceitação na comunidade de desenvolvimento móvel, (JETBRAINS).

Java, amplamente empregada para o desenvolvimento *Android*, é reconhecida por sua fama, orientação a objetos e portabilidade. É uma linguagem de programação consolidada e fundamental no ecossistema *Android*, (ORACLE, 2024).

C#, desenvolvida pela *Microsoft*, é utilizada tanto para criar aplicativos para *Windows* quanto para o desenvolvimento *mobile*, empregando estruturas como o *Xamarin*. Sua orientação a objetos a torna uma escolha robusta para diversos cenários de desenvolvimento, (MICROSOFT, 2023).

JavaScript, conhecida por seu amplo uso no desenvolvimento *web*, também é empregada na criação de aplicativos móveis híbridos. *frameworks* como o *React Native* permitem o desenvolvimento eficiente para *Android* e iOS usando *JavaScript*, promovendo a reutilização de código em diversas plataformas, (MOZILLA CORPORATION, 2024).

Flutter, um SDK de código aberto desenvolvido pelo *Google*, destaca-se por possibilitar o desenvolvimento de aplicativos de alta qualidade para várias plataformas a partir de um único código-fonte, (GOOGLE, 2024).

O *framework* simplifica o processo de desenvolvimento, permitindo que os desenvolvedores foquem no projeto em si, sem se preocuparem com detalhes de configurações repetitivas. Analogamente, um *framework* pode ser comparado à estrutura básica de uma casa, aguardando o desenvolvimento das características específicas, como escolha de acabamentos e componentes. Essa abordagem visa evitar tarefas repetitivas, automatizando parte do trabalho e proporcionando maior eficiência no desenvolvimento de projetos, (AOVS SISTEMAS DE INFORMÁTICA S.A, 2021).

2.11 Serviços de Back-end

Descrever o back-end, conhecido como "os bastidores" ou do “lado do servidor", (PAIXÃO, 2020) constitui uma das camadas cruciais entre o *hardware* e o usuário final. O desenvolvimento de back-end representa o processo de concepção da lógica do lado do servidor, que realiza operações em segundo plano para sites e aplicativos. Abrange todos os trechos de código necessários para criar e manter o banco de dados, o servidor e a aplicação, (SINGH, 2009).

Apesar de não ser perceptível ao usuário, o que é desenvolvido do “lado do servidor”, desempenha um papel essencial no funcionamento do sistema. Nesta camada, ocorre a manipulação de dados e a interação com bancos de dados, sejam eles relacionais ou não-relacionais. Além disso, é responsável pela segurança das informações presentes na aplicação.

Alguns exemplos de linguagens utilizadas no back-end podem ser observadas a seguir:

C++: Combina todos os recursos de *C* com ferramentas de programação orientada a objetos, como classes. É de baixo nível. Ideal para aplicativos sensíveis ao desempenho no nível do sistema, como videogames e grandes aplicativos da *web*.

C#: Preferencial para servidores e ambientes *Windows*, especialmente quando equipes de desenvolvimento utilizam tecnologias *Microsoft*. Nesse contexto, *C#* é frequentemente a linguagem mais eficaz.

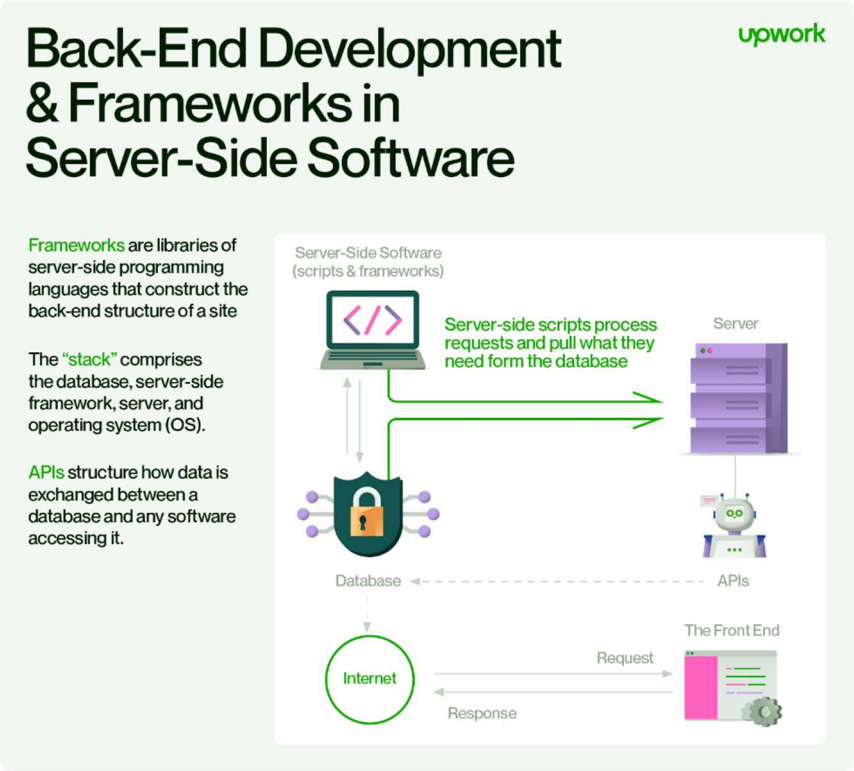
Java: É orientada a objetos de uso geral, projetada com interoperabilidade entre plataformas. Os seus programas são executáveis em qualquer computador que possua a *Java Virtual Machine* (JVM) instalada.

JavaScript (Node.js): Inicialmente *JavaScript* era apenas utilizada “do lado do cliente”, no front-end, mas o *Node.js* consolidou essa linguagem de programação orientada a objetos no “lado do servidor”. Com essa versatilidade se torna uma linguagem muito procurada para desenvolvimento de aplicativos tanto mobile quanto *web*.

Os servidores, bancos de dados, *middleware* e APIs são alguns dos componentes essenciais encontrados no back-end. A Figura 6 oferece uma

representação visual de como esses serviços interagem entre si. Essa arquitetura mantém uma consistência fundamental, ao mesmo tempo em que se adapta às diferentes formas de implementação do back-end. Seja por meio de servidores e armazéns de dados em nuvem, adoção de contêineres com *Docker*, provedores de back-end como serviço, do inglês *Backend-as-a-Service* (BaaS), ou integração de APIs para simplificar processos mais complexos.

Figura 6 – Arquitetura do Back-end.



Fonte: (SINGH, 2009).

Servidores são componentes fundamentais na infraestrutura de redes, desempenhando um papel crucial entre as quatro partes da pilha de back-end, quer estejam localizados fisicamente ou na nuvem. Estes computadores robustos desempenham a função vital de disponibilizar recursos compartilhados essenciais para o funcionamento das redes. Isso inclui, mas não se limita a, armazenamento de arquivos, implementação de medidas de segurança e criptografia, gerenciamento de bancos de dados, prestação de serviços de e-mail e fornecimento de serviços da *web*.

Em suma, os servidores desempenham um papel central ao prover os recursos necessários para sustentar o funcionamento eficiente das redes.

No cenário de um aplicativo, os bancos de dados desempenham um papel central, servindo como a espinha dorsal que confere dinamismo à plataforma. Ao utilizar um aplicativo de gerenciamento financeiro, por exemplo, quando um usuário registra uma despesa, o banco de dados assume o papel de aceitar essa solicitação, armazenar os detalhes da transação e posteriormente recuperar essas informações quando necessário. Da mesma forma, se os usuários interagem com o aplicativo de planejamento de viagens, adicionando novos destinos ou modificando suas preferências de itinerário, essas alterações são refletidas e registradas no banco de dados.

Independentemente do contexto do aplicativo, seja para anotações, compartilhamento de mídia ou planejamento de eventos, a interação contínua dos usuários com o banco de dados possibilita a personalização e atualização constante do conteúdo. Isso proporciona uma experiência mais adaptada às necessidades específicas de cada usuário, tornando o aplicativo mais eficiente e intuitivo.

Em seu cerne, o *middleware* refere-se a qualquer *software* no lado do servidor que estabelece a ponte entre a interface do usuário e a lógica operacional de um aplicativo. Visualizando o *middleware* como a infraestrutura subjacente do seu aplicativo - ele gerencia solicitações e respostas, direcionando dados do aplicativo para o servidor ou banco de dados. Assim como a infraestrutura oculta de uma casa, o *middleware* opera de maneira invisível, mas é indispensável, demandando confiabilidade e consistência em suas funções.

Hoje em dia, torna-se inevitável abordar o componente do lado do servidor de um aplicativo sem destacar as APIs e as integrações fluidas que estabelecem entre *software*, aplicativos, bancos de dados e serviços. A grande maioria dos esquemas de design de *software* no lado do servidor é desenvolvida utilizando APIs, frequentemente substituindo abordagens mais intricadas para viabilizar a comunicação entre o software e a transferência de dados.

Alguns exemplos de serviços de back-end podem ser observados a seguir:

Firebase (*Google*): Possui autenticação, armazenamento em nuvem, banco de dados em tempo real e funções *serverless*.

AWS Amplify: Fornece recursos para criação rápida de aplicativos, incluindo autenticação, armazenamento, banco de dados, funções *serverless* e integração contínua.

Supabase: Uma plataforma *open source* de back-end que inclui um banco de dados *PostgreSQL* escalável, autenticação, armazenamento e recursos de desenvolvimento rápido.

Microsoft Azure: Apresenta bancos de dados, armazenamento, funções *serverless* e soluções de inteligência artificial.

O Termo *serverless* significa “computação sem servidor”, é um modelo de computação em nuvem que permite aos desenvolvedores executarem código sem a necessidade de gerenciar diretamente os servidores.

2.12 Segurança da Informação

No âmbito da segurança da informação, a aplicação de conceitos fundamentais, como criptografia, medidas de prevenção contra acessos não autorizados e salvaguardas destinadas a manter a integridade dos dados, é aprofundada. Esses elementos constituem a base essencial para proteger o sistema contra diversas ameaças de segurança, garantindo a confidencialidade e a integridade das informações armazenadas, (NIELS FERGUSON, 2010).

Dentro do escopo da segurança da informação, é possível contextualizar os desafios e estratégias específicas relacionadas a ambientes de IoT. Discutir medidas de criptografia, proteção contra ameaças cibernéticas direcionadas a dispositivos IoT e garantias de integridade dos dados em um contexto de controle de acesso.

2.13 Raspberry Pi

Raspberry Pi, lançada pela Fundação *Raspberry Pi* em 2012, é um exemplo de mini computador de baixo custo e dimensões reduzidas. Além da versão original, vários outros modelos foram desenvolvidos, cada uma com melhorias em termos de desempenho e recursos, (FUNDAÇÃO RASPBERRY PI, 2024).

Raspberry Pi Zero: Uma versão extremamente compacta e acessível, projetada para aplicações simples e de baixo consumo de energia.

Raspberry Pi 1: A versão inicial que estabeleceu a base para o desenvolvimento subsequente. Possui especificações básicas.

Raspberry Pi 2: Introduzida com melhorias significativas de desempenho em comparação com a versão anterior, permitindo uma gama mais ampla de aplicações.

Raspberry Pi 3B: Incorporando recursos como conectividade Wi-Fi e Bluetooth integrados, ampliando ainda mais as possibilidades de conectividade sem fio.

Raspberry Pi 4B: Equipada com 8 *GigaBytes* (GB) de *Random Access Memory* (RAM), oferece um aumento significativo no poder de processamento e capacidade de multitarefa.

A *Raspberry Pi* é compatível com uma variedade de sistemas operacionais, permitindo aos usuários escolherem o mais adequado para suas necessidades. Alguns exemplos incluem:

Raspbian: Um sistema operacional otimizado para *Raspberry*, baseado no *Debian*, projetado para oferecer desempenho máximo.

Ubuntu Mate: Uma versão do *Ubuntu* projetada para ambientes de *desktop*, adaptada para esta placa.

Windows 10 IoT *Core*: Não é uma versão disponibilizada pela *Microsoft,* foi desenvolvida para ser similar ao *Windows* e projetada para aplicações IoT.

Arch Linux ARM: Uma distribuição *Linux* mais avançada e personalizável, adaptada para a arquitetura *Advanced RISC Machine* (ARM) da *Raspberry*.

Estas opções destacam a versatilidade dessa plataforma de computação embarcada, adequada para uma ampla variedade de projetos, desde servidores simples até aplicações de IoT mais complexas.

O Raspberry Pi adota a arquitetura ARM, conhecida por ser uma implementação de processador baseada em Reduced Instruction Set Computing (RISC), que oferece eficiência energética e desempenho satisfatório para uma variedade de aplicações. Essa arquitetura é comumente empregada em dispositivos móveis e sistemas embarcados devido a sua capacidade de fornecer um equilíbrio entre consumo de energia e potência de processamento.

Os processadores ARM presentes nas diferentes versões da *Raspberry Pi* fornecem um ambiente de execução eficiente, permitindo que a placa atenda às demandas de diversos projetos. Essa arquitetura também facilita a portabilidade de software, já que muitos sistemas operacionais e aplicativos são desenvolvidos com suporte nativo para a arquitetura ARM.

Quanto aos recursos de conectividade, é notável por oferecer diversas opções para interação e comunicação com outros dispositivos e redes. Alguns dos recursos de conectividade presentes incluem:

Wi-Fi e Ethernet: A presença de uma placa Wi-Fi e porta Ethernet permite a conexão sem fio ou com fio à rede.

Bluetooth: A capacidade de suportar Bluetooth amplia as opções de conectividade para periféricos e dispositivos compatíveis.

Portas *Universal Serial Bus* (USB): As portas USB na *Raspberry Pi* permitem a conexão de uma variedade de dispositivos externos, como teclados, mouses, câmeras e unidades de armazenamento.

High-Definition Multimedia Interface (HDMI): A presença de uma porta HDMI permite a conexão direta a monitores e televisores, facilitando a exibição de saídas visuais.

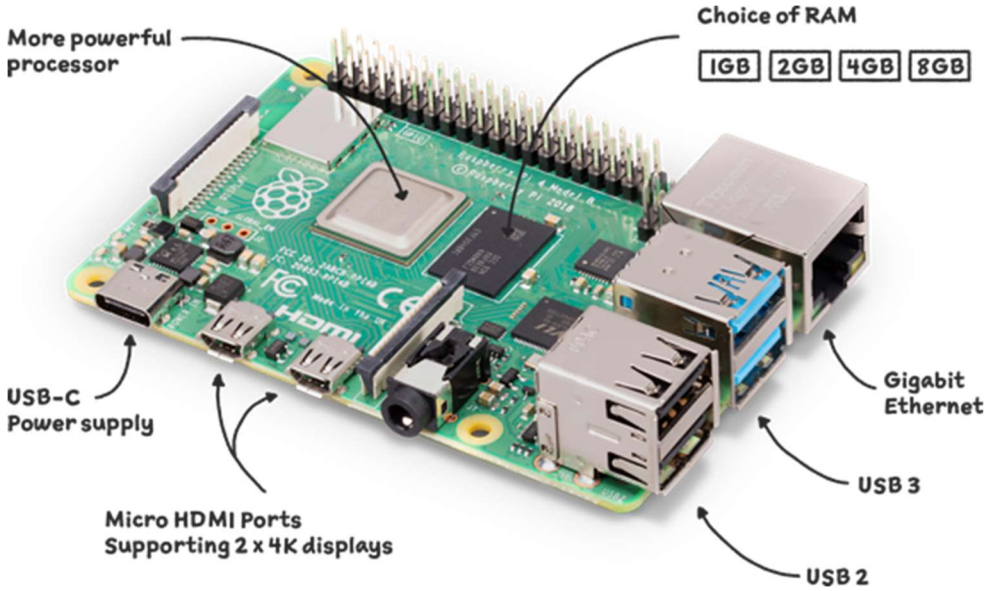
General Purpose Input/Output (GPIO): Os pinos GPIO fornecem uma interface para a interação com dispositivos externos, permitindo a expansão das capacidades

do *Raspberry Pi* através da conexão de sensores, *Light-Emiting Diode* (LEDs) e outros componentes.

Esses recursos de conectividade tornam o *Raspberry Pi* uma escolha versátil para uma variedade de projetos, desde simples projetos de automação residencial até aplicações mais complexas de IoT e servidores. A capacidade de se conectar a diferentes dispositivos e redes é fundamental para a adaptabilidade do *Raspberry Pi* em uma ampla gama de cenários de uso.

O *Raspberry Pi* suporta cartões *Micro Secure Digital* (microSD) para armazenar o sistema operacional e dados. Recomenda-se um cartão de 8 GB a 32 GB, dependendo das necessidades. A compatibilidade e a velocidade do cartão devem ser consideradas para otimizar o desempenho. Neste trabalho, utilizaremos uma *Raspberry Pi* 4, Figura 7, com 8 GB de RAM, e o sistema operacional escolhido para instalação é o *Raspbian*.

Figura 7 – *Raspberry Pi* 4 e seus componentes.



Fonte: (FUNDAÇÃO RASPBERRY PI, 2024).

No projeto em questão, a *Raspberry Pi* foi escolhida como a plataforma de hardware. Embora outras opções, como o *Banana Pi*, *Orange Pi* ou outros mini computadores, pudessem ter sido consideradas, a *Raspberry Pi* foi preferida devido à

sua ampla consolidação no mercado. Além disso, a disponibilidade dessa placa no Departamento de Engenharia Elétrica e a familiaridade da desenvolvedora com o hardware e o sistema operacional *Raspbian*, que é totalmente compatível com a *Raspberry Pi*, também influenciaram na escolha. Esses fatores garantem uma integração mais tranquila e eficiente do hardware e do software no desenvolvimento e manutenção do projeto.

2.14 *Node-RED* - Automação Visual para Desenvolvimento Rápido de Fluxos IoT

Em 2013, *Nick O'Leary* e *Dave Conway-Jones* do grupo *Emerging Technology Services* da IBM deram início ao projeto do *Node-RED*. Sua criação se deu como uma prova de conceito de manipulação e visualização entre tópicos MQTT (Node-RED, 2024).

O *Node-RED* é fundamentado em programação baseada em fluxo, o comportamento das aplicações desse gênero funciona como uma rede de caixas pretas. Esse programa é uma ferramenta de programação visual que desde o princípio foi produzida e desenvolvida em código aberto. Essa plataforma é usada para conectar hardwares, APIs e serviços locais ou online de maneira inovadora. Em questões de usabilidade o editor de código é baseado em navegador, o que facilita a criação de fluxos, ele também utiliza uma interface de arrastar e soltar as “caixas pretas” que agora serão chamadas de nós, como pode ser observado na Figura 8.

É válido salientar que o *Node-RED* é construído sobre o *Node.js* e sua linguagem de programação é o *JavaScript*. A plataforma apresenta uma biblioteca de nós pré-construídos que podem ser utilizados para conectar aplicações internas ou externas, o programa também abre espaço para a construção de nós personalizados, o que dá além da modularidade, uma liberdade para a criatividade e inovação nos projetos.

Algumas aplicações em que o *Node-RED* é usado são:

Aplicativos IoT: conexão e controle de dispositivos.

Processamento de dados: filtragem transformação e análise de dados.

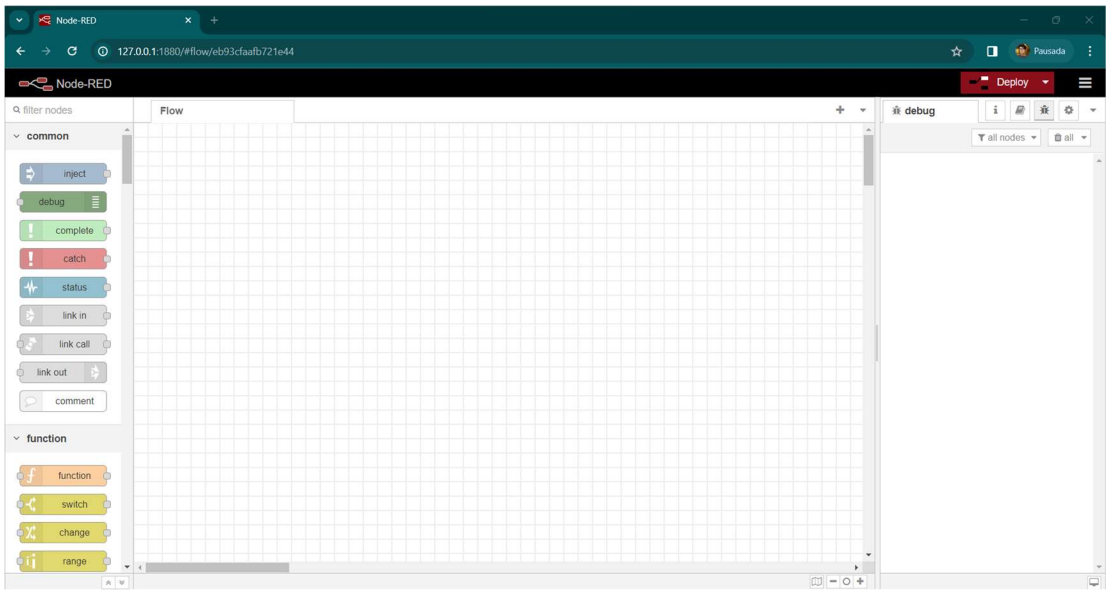
Automação do fluxo de trabalho: através da conexão de diferentes serviços e aplicativos.

Dashboads: construções de interfaces para o usuário.

Coleta de dados

Um dos casos de uso mais comuns do *Node-RED* é integrá-lo ao MQTT para comunicação de dispositivo para dispositivo, ele já fornece um suporte integrado para MQTT e permite que você se conecte facilmente a um *broker* MQTT e envie mensagens para dispositivos.

Figura 8 – Interface do *Node-RED*.



Fonte: O'Autor.

No desenvolvimento inicial deste projeto, o Node-RED foi empregado como uma ferramenta de teste da aplicação. Posteriormente, serão explorados outros casos de uso para o Node-RED dentro do escopo deste trabalho.

3 DESENVOLVIMENTO DO TRABALHO

3.1 Desenvolvimento do Aplicativo

Esta seção detalha as etapas cruciais do desenvolvimento do aplicativo de controle de acesso a ambientes físicos, abordando a criação da interface do usuário, a implementação do banco de dados local e a incorporação de funcionalidades de controle de acesso. Inclui também a instalação do sistema operacional na *Raspberry Pi*, juntamente com a instalação do *Expo Go* e do *Supabase*. O processo abrange a implementação das aberturas das portas utilizando o protocolo MQTT e *WebSocket*, bem como a integração do *Supabase*. O enfoque primordial é assegurar a eficiente usabilidade do aplicativo, garantindo a coesa integração de elementos essenciais para a gestão de acesso a ambientes físicos.

3.2 Arquitetura do Sistema

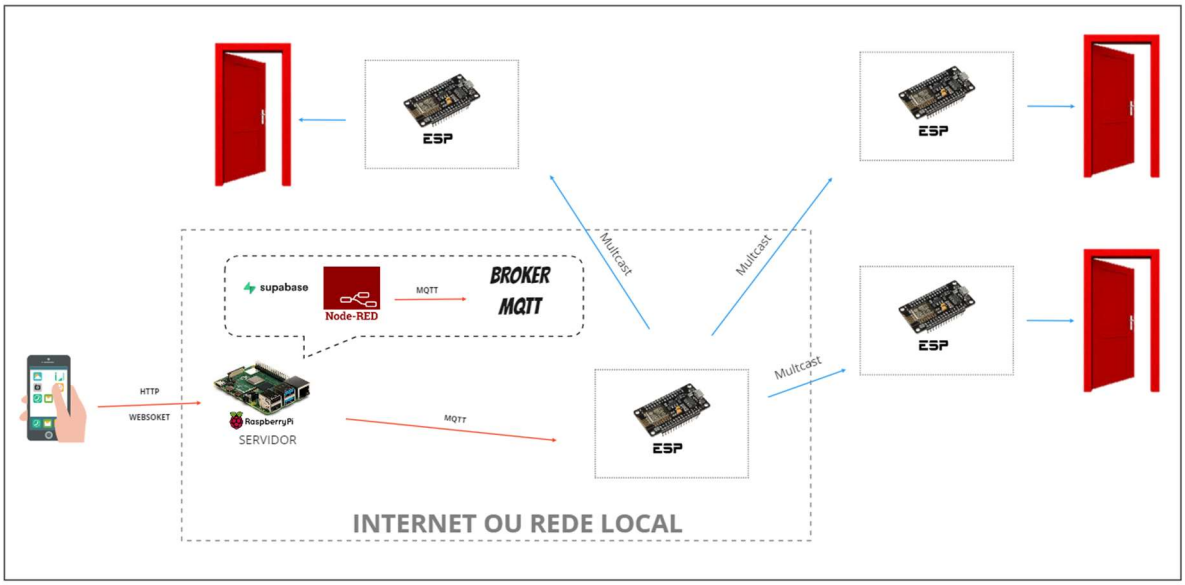
Cada componente da arquitetura desempenha um papel crucial no funcionamento integrado do sistema, conforme apresentado na Figura 9. O *Raspberry Pi*, central na infraestrutura, hospeda o *Expo CLI*, o *Supabase CLI* e o *Node-RED*, permitindo a utilização do ambiente de desenvolvimento para testes iniciais. Destaca-se que a implementação inicial é conduzida através do aplicativo *Expo Go*, sem a necessidade de publicação nas lojas da *Apple* ou *Android*. Essa abordagem visa validar o ambiente de desenvolvimento e a eficácia do aplicativo antes de proceder a etapa de publicação, assegurando um processo produtivo e preciso durante a fase inicial do projeto.

O *Expo Go*, um componente vital, é utilizado para inicializar o aplicativo móvel, proporcionando aos usuários a capacidade de acessar os ambientes solicitados e autorizados. Ao efetuar login no aplicativo e clicar no botão de liberação específico, os usuários podem solicitar a abertura da porta. Cada botão de liberação disponível para o usuário logado é previamente autorizado pelo responsável do respectivo ambiente.

O protocolo MQTT é implementado através do *broker*, que atua como um canal eficiente de comunicação entre o aplicativo móvel e o sistema de controle de acesso. Ao clicar no botão de liberação desejado, uma mensagem é enviada para o *broker* com intermédio do *Node-RED*, contendo informações associadas ao tópico MQTT correspondente armazenado no *Supabase*. Essa ação aciona a abertura da porta desejada.

O *Supabase*, componente-chave do sistema, desempenha um papel crucial no armazenamento e gestão de informações críticas. Ele abrange uma variedade de dados, como tópicos MQTT associados aos botões de liberação autorizados, informações de usuários, atributos dos ambientes, configurações do *broker* e solicitações de acesso. Esses exemplos representam apenas uma parte dos diversos dados gerenciados, todos essenciais para o funcionamento adequado do sistema. Cada componente é integrado de maneira sinérgica para criar uma arquitetura coesa e eficaz para o sistema de controle de acesso proposto.

Figura 9 – Arquitetura geral da aplicação de controle de acesso.



Fonte: O'Autor.

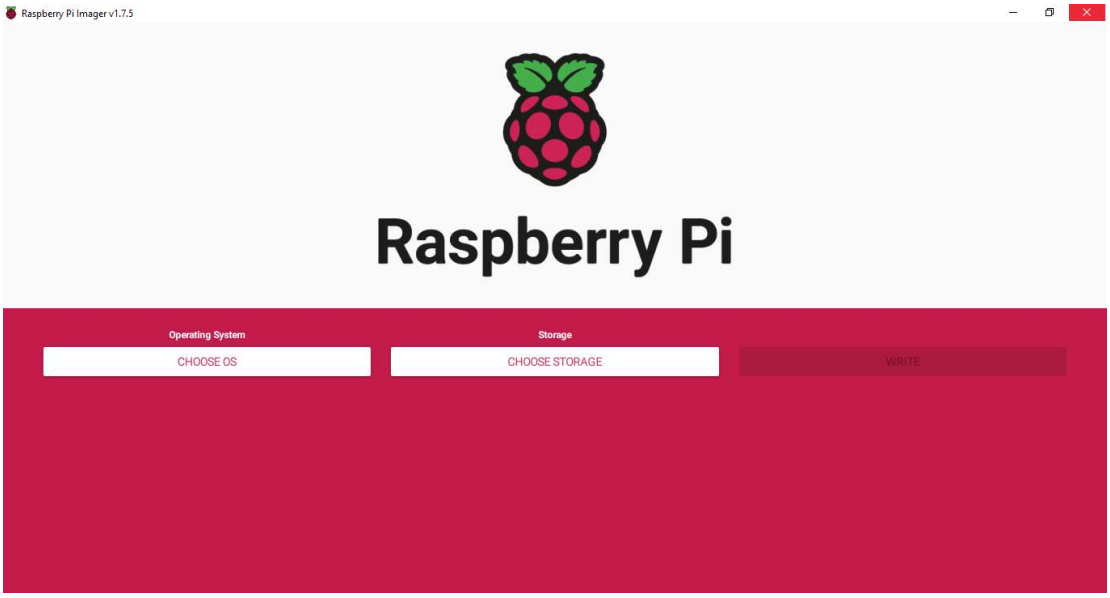
3.3 Configuração da *Raspberry Pi* e Ambiente de Desenvolvimento

Com o objetivo de oferecer uma solução econômica e acessível para a Universidade Federal de Pernambuco, foi decidido utilizar hardware *Raspberry Pi*. A configuração escolhida foi a de ponta: *Raspberry Pi 4* com 8GB de RAM e um cartão de memória de 32GB. No entanto, após a instalação dos *softwares* necessários para desenvolver a aplicação, tornou-se evidente que o espaço disponível era insuficiente, visto que mais de 30GB já estavam sendo. Após a realização de testes, foi definido que a opção ideal seria um cartão de 128GB. Vale ressaltar que, para utilizar cartões de memória com capacidade superior a 32GB, é necessário formatá-los como tabela de alocação de arquivos, do inglês *File Allocation Table* 32 (FAT32), e lidar com suas limitações.

Dentre as limitações desse sistema de arquivos gerenciador e organizador de dados está a impossibilidade de armazenar arquivos individuais maiores que 4GB e a fragmentação do sistema que pode impactar o sistema, principalmente em grandes volumes.

Quanto ao sistema operacional, foi escolhido o *Raspbian* devido ao suporte oficial da *Raspberry Pi*, sua gratuidade, base no *Debian* e otimização para este tipo de *hardware*, (Bem vindo ao Raspbian). Para a instalação do sistema operacional, foi utilizado o "*Raspberry Pi Imager*", conforme demonstrado na Figura 10. Basta instalá-lo em um computador com leitor de cartão de memória, escolher o sistema operacional e o local de armazenamento (cartão de memória) e clicar em "Escrever". Após a conclusão deste processo, o cartão de memória já pode ser inserido no *Raspberry Pi*. Agora, é possível conecta-oa à energia, e adicionar teclado, mouse e monitor para utilizá-lo como um computador convencional.

Figura 10 – Tela inicial do *Raspberry Pi imager*.

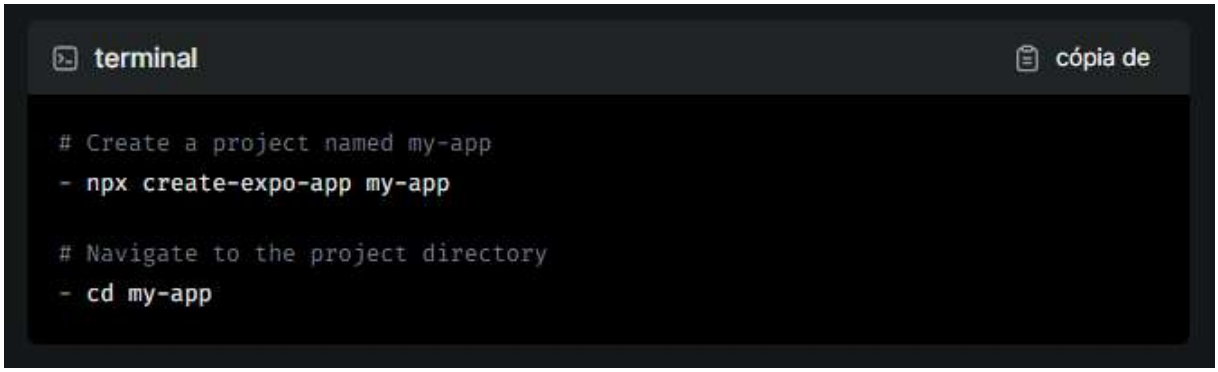


Fonte: Adaptado de (FOUNDATION).

Com o hardware e o sistema operacional configurados, foi possível proceder com a instalação do ambiente de desenvolvimento. Inicialmente, foi instalado o *Expo CLI*. Os requisitos para instalação do *Expo* incluem o *Node.js* na versão suporte de longo prazo, do inglês *Long-term support* (LTS), *Git* e o *Watchman* necessário em sistemas Linux, (Expo, 2024). Com esses programas instalados, nas versões 20.9.0, 2.37.1 e 4.9.0, foi possível abrir o terminal, entrar na pasta correspondente e criar o aplicativo, conforme demonstrado na Figura 11. Para iniciar o servidor de desenvolvimento, utiliza-se o comando apresentado na Figura 12. Em seguida, foi instalado o aplicativo *Expo Go* no celular e lido o código QR gerado no terminal ao iniciar o serviço. A partir desse momento, o aplicativo já pode ser aberto no dispositivo móvel. Para iniciar as modificações no front-end da aplicação, foi instalado o *Visual Studio Code*, recomendado pela *Expo*. Utilizou-se também o *VS Code Expo Extension* para facilitar a identificação e correção de erros, além do *Yarn* para gerenciar as dependências de forma mais rápida e conveniente em comparação com o *Node Package Manager* (NPM) e *Node Package Executor* (NPX).

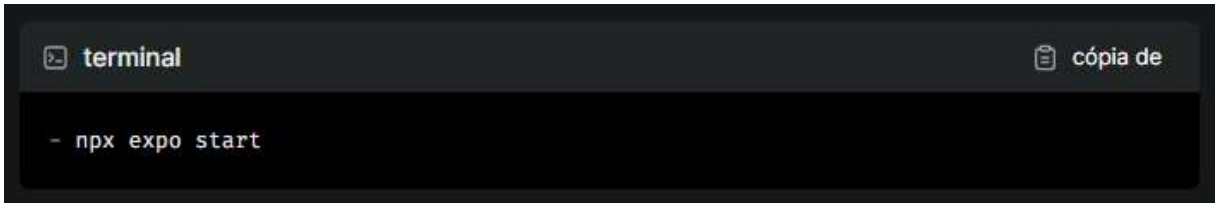
No Apêndice B é possível observar configurações adicionais para garantir que o serviço *Expo* seja sempre reinicializado de acordo com a inicialização do servidor.

Figura 11 – Comandos que devem ser executados no terminal para criação do aplicativo.



Fonte: (Supabase, 2024).

Figura 12 – Comando para levantar o servidor de desenvolvimento.



Fonte: (Supabase, 2024).

Após essa etapa, a seleção do serviço de back-end foi feita entre as opções recomendadas pelo *Expo*, com a escolha direcionada para o *Supabase*. Visando garantir a automação na borda, o *Supabase* foi integrado à máquina local. Entre os benefícios notáveis do desenvolvimento local, destacam-se a rapidez na produção, graças à ausência de latência na rede e interrupções de internet. Além disso, essa abordagem proporciona maior colaboração em projetos conjuntos, economia de recursos ao permitir a criação ilimitada de projetos locais e a capacidade de trabalhar *offline*, entre outras vantagens.

Existem duas opções para utilizar o *Supabase* em uma máquina de desenvolvimento local: por meio do *Supabase Docker* ou do *Supabase CLI*. Devido a considerações de compatibilidade com o *hardware* ARM 64 do *Raspberry Pi*, a escolha recaiu sobre o *Supabase CLI*, visto que a maioria dos *Dockers* é desenvolvida com arquitetura de 32 *bits*. Os testes realizados com a opção *Docker* no *Raspberry Pi* resultaram em problemas de compatibilidade como esperado. Para garantir o pleno

acesso a todos os serviços do *Supabase*, recomenda-se, no mínimo, 7GB de memória RAM, garantindo a harmonização com o *hardware* utilizado, (Supabase, 2024).

Até o momento, não foi realizada uma análise completa de todos os serviços do *Supabase*, o que pode exigir a exclusão eventual da verificação de saúde dos containers, caso esteja causando obstáculos à inicialização, devido à proximidade do limite de utilização de memória RAM do *hardware*, (Supabase, 2024). Para iniciar um novo projeto, basta acessar o diretório desejado e inserir o comando exibido na Figura 13. Para iniciar o *Supabase*, é necessário entrar na pasta criada e utilizar o comando indicado na Figura 14.

Figura 13 – Comando para criar um projeto do *Supabase*.

Fonte: (Supabase, 2024).

Figura 14 – Comando para iniciar o *Supabase*.

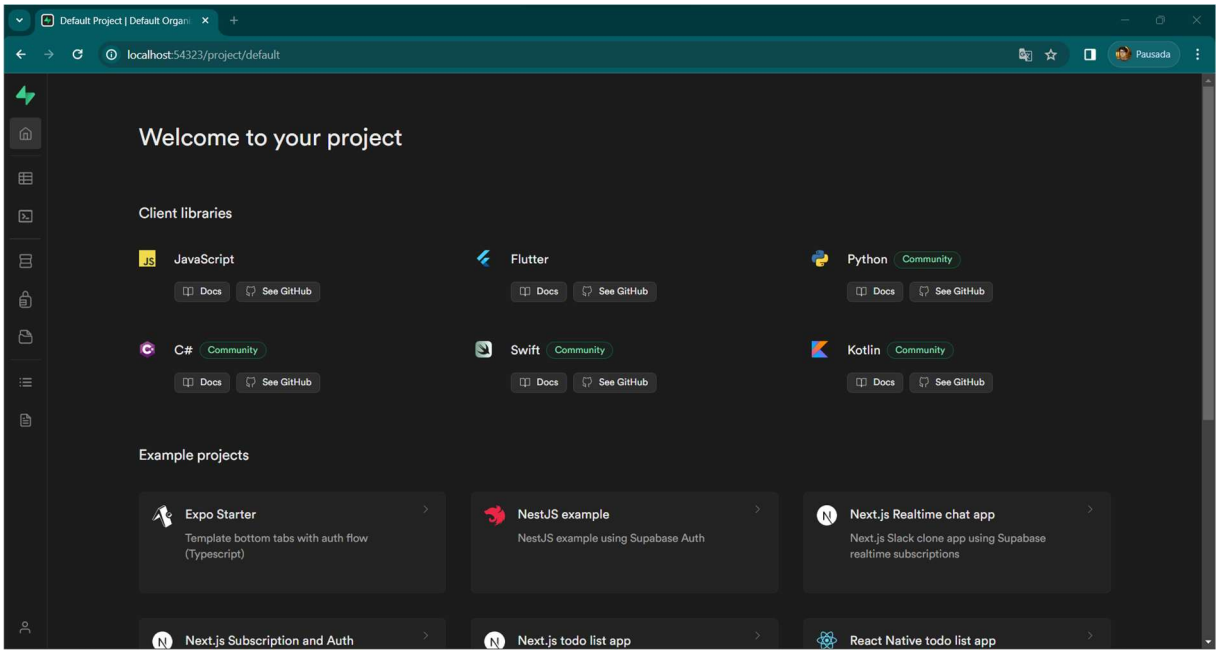
Fonte: (Supabase, 2024).

O *Supabase CLI* inclui uma ferramenta de *dashboard* para o desenvolvimento da aplicação, acessível através do endereço “*localhost*:54323”. Além disso, é possível utilizar outros SGBDs, como o *pgAdmin* ou *DBeaver*. Na configuração específica da *Raspberry Pi*, que possui recursos limitados, a escolha pelo *DBeaver* se justifica devido a sua eficiência superior no consumo de recursos do sistema. Essa eficiência resulta em um desempenho mais responsivo e adequado às restrições de *hardware* da *Raspberry Pi* em comparação com outras opções estudadas.

Neste trabalho a execução da programação foi feito num computador convencional e os testes foram realizados do ambiente configurado com as

especificações ditas, portanto foi utilizada a própria dashboard do *Supabase CLI* devido a sua ótima usabilidade, esta interface pode ser observada na Figura 15.

Figura 15 – Interface da dashboard do *Supabase*.



Fonte: O'Autor.

3.4 Desenvolvimento do Front-End

Primeiramente, a tela de abertura foi configurada, empregando uma imagem com a logo exclusiva da aplicação, como apresentado na Figura 16. Em seguida, a página de entrada foi desenvolvida, incorporando dois campos destinados à visualização do usuário e senha, sendo esses campos importados da biblioteca "*react-native*". A classe utilizada para isso foi a "*TextInput*". Esses campos serão automaticamente preenchidos com o auxílio da programação do lado do servidor. Além disso, a página de entrada inclui um botão "Entrar" para acessar a interface inicial do aplicativo, um botão dedicado a exibir a identidade (ID) única do dispositivo e outro botão específico para solicitar acesso aos ambientes, proporcionando uma experiência intuitiva ao usuário. Todos esses elementos podem ser visualizados na Figura 17.

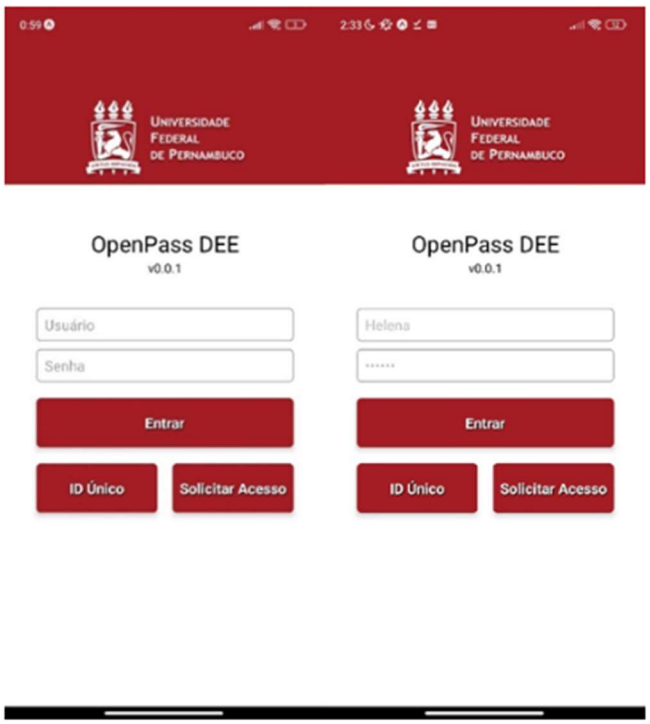
Figura 16 – Tela de abertura do aplicativo de controle de acesso.

Fonte: O’Autor.

Para a construção deste botão, foi utilizada a mesma biblioteca mencionada anteriormente, sendo a classe empregada o “*TouchableOpacity*”.

Ao acionar o botão “ID Único”, uma tela modal é aberta. E ela foi feita utilizando o método “modal” do “*react-native*”, também conhecida como pop-up. O termo “*pop-up*” deriva da ideia de algo que salta, assemelhando-se a uma página que “salta” na tela do usuário. Nessa tela, é exibido o ID único do dispositivo, coletado por meio da biblioteca “*expo-application*”.

Figura 17 – Página de entrada do aplicativo de controle de acesso sem e com usuário cadastrados, respectivamente.



Fonte: O'Autor.

Além disso, a tela modal dispõe de um botão de retorno, permitindo ao usuário regressar à tela de entrada do aplicativo. Na Figura 18, é possível visualizar essa descrição.

Ao acionar o botão "Solicitar Acesso", a tela para solicitação de acesso aos ambientes é aberta, conforme apresentado na Figura 19. Nessa tela, é disponibilizado um formulário para o preenchimento dos dados do usuário. Caso o usuário já esteja cadastrado, os campos serão automaticamente preenchidos com o auxílio de ferramentas do lado do servidor. Ao pressionar o botão de enviar, o formulário será submetido, e um alerta será exibido indicando que a solicitação foi realizada com sucesso, como mostrado na Figura 20. No entanto, se houver campos não preenchidos, um aviso será apresentado, orientando o preenchimento correto, como evidenciado na Figura 21. Em caso de cancelamento, ao pressionar o botão correspondente, o usuário retorna à página de entrada.

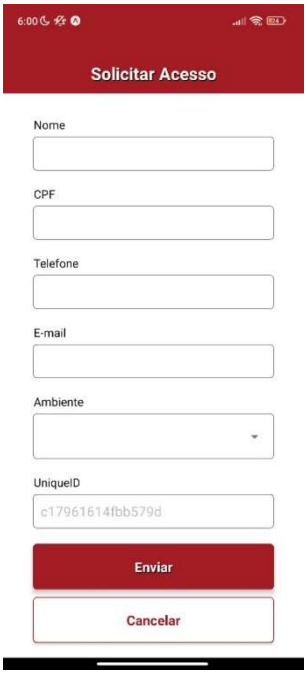
Figura 18 – Tela modal para observar o ID único do aplicativo de controle de acesso.

Fonte: O'Autor.

É importante destacar que para a exibição dessas mensagens de alerta e aviso, foi utilizada a classe "*Alert*" da mesma biblioteca empregada para a construção da interface do usuário. Essa abordagem visa manter uma aparência padronizada nas telas, garantindo uma experiência coesa ao usuário.

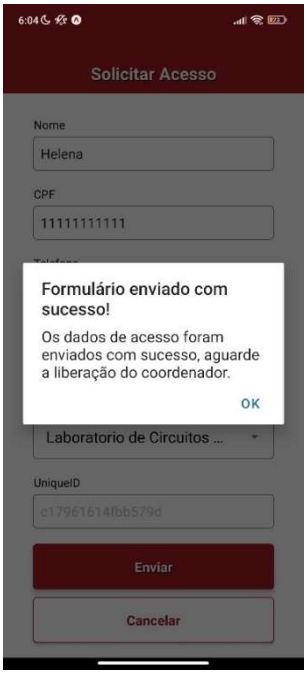
Um dos campos de preenchimento representa um menu suspenso. Nesse menu, o usuário pode escolher qualquer ambiente cadastrado na aplicação para solicitar acesso. A biblioteca utilizada para criar esse menu é o "*react-native-select-dropdown*", por meio da classe "*SelectDropdown*", conforme apresentado na Figura 22.

Figura 19 – Página para solicitar acesso aos ambientes do aplicativo de controle de acesso.



Fonte: O’Autor.

Figura 20 – Alerta indicando que a solicitação foi realizada com sucesso.

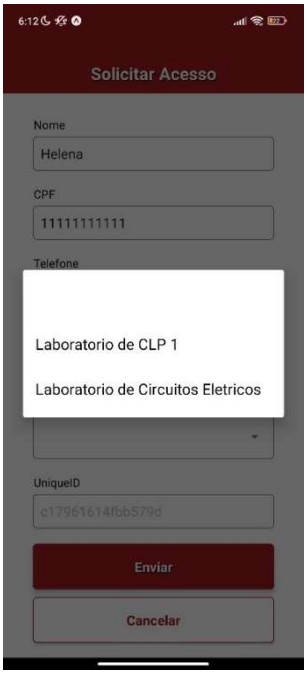


Fonte: O’Autor.

Figura 21 – Aviso orientando o preenchimento correto do formulário de solicitação de acesso.

Fonte: O’Autor.

Figura 22 – Menu suspenso com todos os ambientes cadastrados.



Fonte: O’Autor.

Com o cadastro validado, ao clicar no botão “Entrar” na tela apresentada na Figura 17 , a página mostrada na Figura 23 será aberta. Nessa tela, é possível visualizar na barra superior o usuário logado, o botão de sair e os ambientes aos quais esse usuário tem acesso. Se o acesso a um determinado ambiente for do tipo “coordenador”, um botão com uma engrenagem aparecerá no lado direito, permitindo a gestão de acesso à sala ou ao laboratório. A logo do Departamento de Engenharia Elétrica (DEE) foi adicionada à barra superior para indicar que esses ambientes fazem parte do DEE.

Para adicionar os ícones de “sair” e “configuração” foi utilizada a biblioteca “*react-native-vector-icons/FontAwesome*” e a classe “*Icon*”.

Figura 23 – Página inicial com os ambientes habilitados para o usuário.

Fonte: O’Autor.

Ao acionar o botão de configuração, o usuário com perfil de “*coordenador*” terá acesso à tela de gestão de permissões específica daquele ambiente. Ao clicar no botão do tipo “*switch*”, como ilustrado na Figura 24, o coordenador pode liberar o acesso solicitado para o ambiente escolhido pelo usuário. O

botão “*switch*” utilizado nesta implementação faz parte da biblioteca “*react-native*” e é representado pela classe denominada “*Switch*”. Além disso, há um botão de retorno que leva de volta à página dos ambientes, configurado da mesma maneira que na tela modal.

Figura 24 – Página de gestão de permissões do ambiente.

Fonte: O’Autor.

Outra funcionalidade presente que, quando o usuário clica nos nomes das listas de usuários com acesso e solicitações pendentes, a tela de gestão do usuário é acessada, conforme mostrado na Figura 25, o coordenador tem visibilidade de todos os dados cadastrados pelo usuário. Nessa tela, há a opção de definir se o tipo de acesso é limitado ou ilimitado, utilizando um menu *dropdown* semelhante ao presente na tela de solicitar acesso, como mostrado na Figura 19. Se a opção selecionada for limitada, serão apresentados dois campos para que o coordenador defina o horário de utilização do ambiente, até o momento não foi possível habilitar a funcionalidade de definição de horário de uso dos locais.

Figura 25 – Página de gestão do usuário.

Fonte: O'Autor.

Na Figura 44 presente no Apêndice C, é possível ter uma compreensão visual da navegação das telas do aplicativo.

3.5 Integração do Protocolo MQTT e *Node-RED* para Controle de Acesso

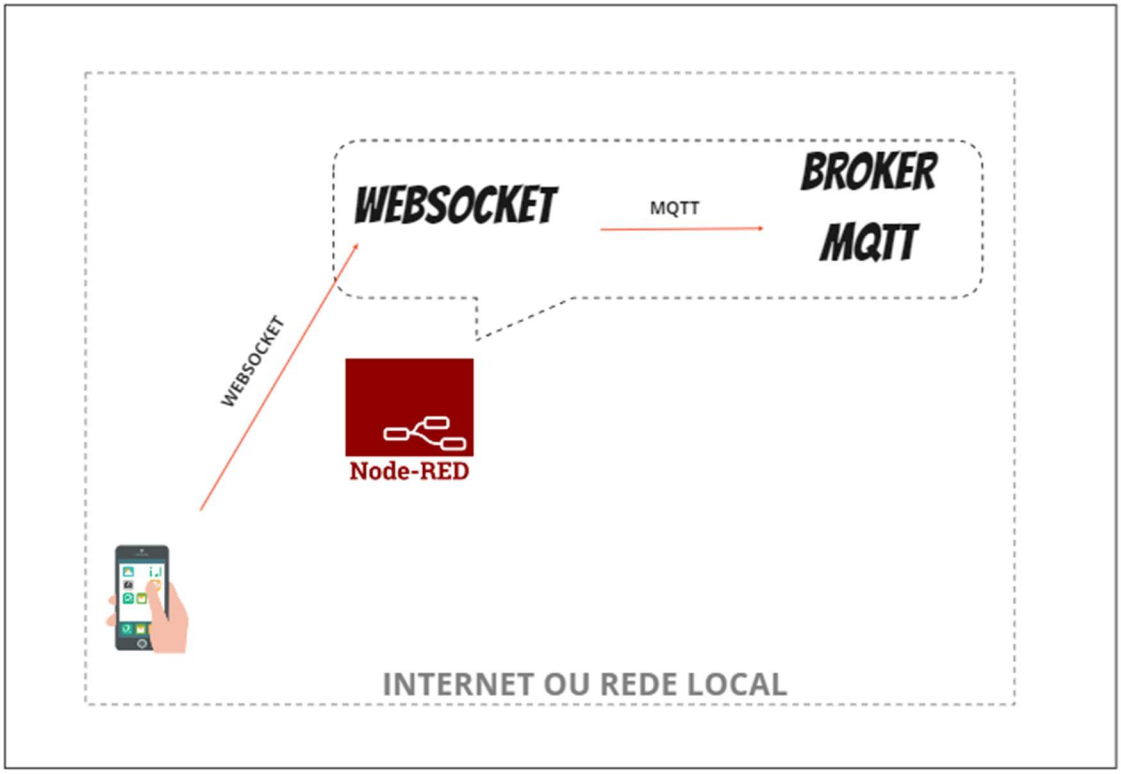
Para controlar o acesso aos ambientes seria necessário clicar em um dos botões, como exemplificado na Figura 23, e o aplicativo enviaria um tópico MQTT com uma mensagem ao *broker*. Entretanto, foram feitos diversos testes com as bibliotecas "*paho-mqtt*", "*paho-client*", "*mqtt*", "*mqtt-reat-native*" e muitas outras, e foi observado que o real protocolo utilizado, era *WebSocket*.

Devido ao prazo final de entrega deste trabalho, a solução encontrada foi fazer uma ponte com o *Node-RED*, em que este recebe a mensagem e o tópico via *WebSocket* e encaminha ao *broker* via MQTT, exemplificado na Figura 26.

A implementação do *WebSoket* para controlar a abertura das portas por meio do aplicativo móvel foi realizada utilizando a biblioteca "*paho-mqtt*" no *Expo*. Para organizar o código e utilizar as classes necessárias, o objeto *JavaScript global* "*Paho"*

foi empregado. Os dados de configuração do *WebSocket*, essenciais para o acesso, são armazenados no *Supabase*. O aplicativo móvel, por meio do serviço do servidor, solicita e recupera essas informações do banco de dados para acessá-lo.

Figura 26 – Arquitetura de comunicação entre o aplicativo, o *WebSocket* e o *broker* MQTT.



Fonte: O'Autor.

WebSocket é uma tecnologia que permite a comunicação bidirecional em tempo real entre um navegador (ou qualquer cliente) e um servidor *web*. Ao contrário do HTTP tradicional, que é unidirecional e baseado em requisições do cliente e respostas do servidor, o *WebSocket* estabelece uma conexão persistente que permite tanto ao cliente quanto ao servidor enviarem dados a qualquer momento, (WebSockets).

Assim, o servidor pode enviar dados para o cliente sem que o cliente precise solicitar. Isso é ideal para aplicativos que exigem atualizações em tempo real, como salas de *chat*, jogos *multiplayer online*, *feeds* de notícias ao vivo e muito mais.

Além disso, o *WebSocket* é mais eficiente do que alternativas como *polling* (fazer repetidas requisições ao servidor para buscar atualizações) e *long polling* (manter uma requisição aberta até que o servidor tenha dados para enviar), pois reduz a sobrecarga de requisições HTTP desnecessárias.

Algumas ferramentas populares para trabalhar com *WebSocket* incluem:

Socket.IO: Uma biblioteca *JavaScript* para uso em aplicativos *web* e servidores *Node.js* que simplifica a comunicação em tempo real.

WebSocket API nativa: A maioria dos navegadores modernos suporta a API *WebSocket* nativa, que permite criar facilmente uma conexão sem depender de bibliotecas externas.

SignalR: Uma biblioteca da *Microsoft* para aplicações .NET que simplifica a criação de aplicativos em tempo real.

O *Node Aedes* é um servidor de *broker* MQTT escrito em *JavaScript* para o ambiente *Node.js*. Embora esse nó seja principalmente focado em suportar o protocolo MQTT, ele pode ser estendido para oferecer suporte a outros protocolos de comunicação, como o *WebSocket*. A integração dele permite que clientes se conectem diretamente ao servidor *Aedes* usando o protocolo *WebSocket* para comunicação em tempo real, (Node-RED, 2024).

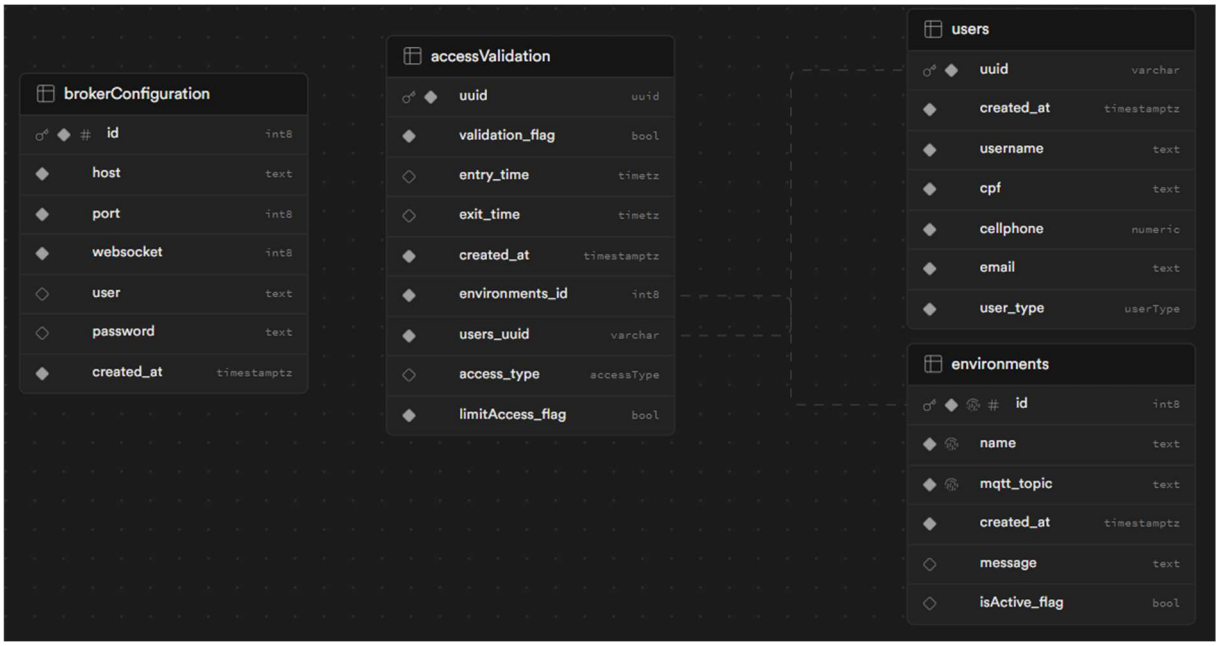
Neste trabalho, ao empregar o nó *Aedes*, foi essencial ter precaução no gerenciamento da abertura e fechamento da seção do protocolo *WebSocket*, visando evitar sobrecargas na comunicação. Os métodos utilizados no código da biblioteca "*paho-mqtt*" já mencionada foram "*connect*" e "*disconnect*". Na Figura 37 observada no Apêndice A é possível observar o fluxo construído no *Node-RED*.

3.6 Desenvolvimento do Banco de dados com *Supabase*

O banco de dados é composto por quatro tabelas principais: "*accessValidation"*, "*brokerConfiguration*", "*environments*" e "*users*". Cada tabela tem características específicas e está interligada por chaves estrangeiras para estabelecer relacionamentos entre elas. Essas entidades podem ser observadas na Figura 27.

A tabela “*accessValidation*” armazena informações sobre validações de acesso, como identificador único universal (UUID) “*uuid*”, estado de validação “*validation_flag*”, horários de entrada e saída “*entry_time*” e “*exit_time*”, data de criação do registro “*created_at*”, tipo de acesso “*access_type”*, e uma flag indicando limitação de acesso “*limitAccess_flag*”. Esta tabela se relaciona com as tabelas “*users*” e “*environments*” por meio de chaves estrangeiras “*users_uuid*” e “*environments_id*”.

Figura 27 – Esquema do modelo entidade relacionamento do banco de dados.



Fonte: O’Autor.

A tabela “*brokerConfiguration*” armazena configurações do broker, como host, porta, *WebSocket*, usuário e senha. Possui um identificador único “id” e um campo para registrar a data de criação do registro “*created_at*”.

A tabela “*environments*” contém informações sobre ambientes, como identificador único “id”, nome “*name*”, tópico MQTT “*mqtt_topic*”, data de criação do registro “*created_at*”, uma mensagem padrão “*message*”, e uma flag indicando se o ambiente está ativo “*isActive_flag*”. Esta tabela é referenciada pela tabela “*accessValidation*” por meio da chave estrangeira “*environments_id*”.

A tabela "*users*" armazena dados sobre os usuários, como identificador único “*uuid*”, data de criação do registro “*created_at*”, nome de usuário “*username*”, CPF “cpf”, número de celular “*cellphone*”, e-mail “*email*”, e tipo de usuário “*user_type*”. A tabela “*accessValidation*” se relaciona com esta tabela por meio da chave estrangeira “*users_uuid*”.

Dessa forma, as tabelas estão interconectadas por meio de chaves estrangeiras, permitindo que informações em uma tabela se relacionem e dependam de registros em outras tabelas, estabelecendo uma estrutura coesa para representar e manter dados no banco. Essa abordagem de design de banco de dados facilita a integridade referencial e proporciona uma maneira eficaz de organizar e recuperar informações relacionadas de maneira consistente e eficiente.

3.7 Desenvolvimento do Back-End com *Supabase*

Na página de entrada mencionada anteriormente, retratada na Figura 17, são importadas diversas bibliotecas, incluindo “*react*”, “*react-native*”, “*@react-navigation/native*”, e “*expo-application*”, além das funções dos diretórios responsáveis pelo *Supabase* e pelo identificador exclusivo do dispositivo.

Após a importação, as variáveis são declaradas, e uma função “*useEffect*” é empregada para obter esse identificador único assim que a página é carregada. A função “*getAndroidId()*” é utilizada, proporcionando uma *string* hexadecimal única para cada combinação de chave de assinatura de aplicativo, usuário e dispositivo no ambiente Android. Uma *string* hexadecimal é uma representação de números em base 16, utilizando os caracteres de 0 a 9 e de A a F. É importante destacar que esse valor pode ser alterado caso ocorra uma redefinição de fábrica no dispositivo ou se houver uma modificação na chave de assinatura do *Android Package Kit* APK, (Expo, 2024).

Para dispositivos iOS, o método “*getIosIdForVendorAsync()*” é empregado, capturando o identificador do fornecedor (IDFV) do iOS. Esse identificador único é uma *string* que identifica exclusivamente um dispositivo para o fornecedor do aplicativo.

Posteriormente, ao capturar o identificador do dispositivo, é enviado uma requisição ao banco de dados para descobrir se o usuário já é cadastrado. Se a verificação for positiva, o usuário terá acesso à tela de ambientes; caso contrário, uma mensagem será exibida indicando que o usuário não está cadastrado, conforme ilustrado na Figura 28.

Figura 28 – Mensagem indicando que o usuário não está cadastrado.

Fonte: O'Autor.

Na tela "Solicitar Acesso", realiza-se uma consulta SQL "*SELECT"* com uma cláusula "*WHERE*" para comparar o ID do dispositivo e verificar a existência do usuário no banco de dados, permitindo assim a recuperação dos dados associados a esse usuário, de maneira similar a tela anterior.

Na consulta realizada ao *Supabase* utilizando o comando "*SELECT"*, todos os dados da tabela de ambientes são recuperados. No entanto, para otimizar o desempenho e a interface do usuário, apenas os nomes dos ambientes são utilizados no menu suspenso. Isso é feito para evitar o excesso de informações e garantir uma

experiência mais eficiente e intuitiva ao usuário, apresentando apenas as informações relevantes para a seleção do ambiente desejado.

Com o ambiente escolhido e todos os campos devidamente preenchidos, ao clicar no botão “Enviar”, uma operação de inserção será realizada nas entidades de usuários e validação de acesso, registrando assim o a solicitação de acesso e o cadastro de usuário no banco de dados.

Ao acessar a tela de ambientes, conforme ilustrado na Figura 23, são incorporadas as mesmas bibliotecas previamente mencionadas, além da função relacionada ao *banner*, desenvolvida em um arquivo específico no mesmo diretório. Esse arquivo abriga o desenvolvimento da integração da comunicação com WebSocket ao aplicativo, conforme detalhado no tópico 3.5. Os *banners* são, na verdade, botões que, ao serem acionados, estabelecem comunicação com o broker utilizando o método “*connect*” da biblioteca “*Paho*”. Adicionalmente, lidam com situações de conexão perdida e utilizam o método “*send*” para transmitir a mensagem correspondente.

A implementação inclui o uso de um comando "*SELECT*" com dois critérios "*WHERE*" para verificar o identificador único do usuário e garantir que a *flag* de validação esteja definida como verdadeiro, tornando o *banner* visível e operável para o usuário. No caso de um coordenador, o botão de gestão de acesso será renderizado. Para verificar se um usuário é coordenador, é analisado o tipo de acesso associado ao ambiente na entidade de validação de acesso. Na barra superior, é exibido o e-mail do usuário logado para facilitar a identificação e a associação dos ambientes correspondentes.

Na tela de gestão de acesso, é realizada a configuração da barra superior, semelhante à página de ambientes anteriormente mencionada na Figura 23. No código dessa página, a função “*fetchUsersWithAccess*” é responsável por buscar e recuperar dados dos usuários associados aos pedidos de acesso a um ambiente específico.

Dentro do corpo da tela, a função utiliza o *Supabase* para consultar a entidade “*accessValidation*”, selecionando os “*uuid*”s dos usuários associados ao ambiente identificado por “*environment.id*”. Posteriormente, é aplicado o método “*map*” para

extrair a lista de “*uuid*”s desses usuários. Essa lista é, então, utilizada para buscar os dados detalhados dos usuários na tabela “*users*” por meio do método “*in*” no *Supabase*.

Os dados resultantes, contendo informações como “*uuid*”, nome de usuário e e-mail, são armazenados na variável “*userData*”. Em caso de erro durante o processo, mensagens de erro são registradas como alertas. Os dados dos usuários com acesso ao ambiente são, por fim, atualizados no estado da aplicação por meio da função “*setUsersWithAccess*”.

O “*switch*” indica se o usuário está validado para acessar este ambiente, através de uma consulta SQL na entidade “*accessValidation*”.

A tela “Gestão de Usuário”, Figura 25. Apresenta um back-end similar a tela “Solicitar acesso”, Figura 19. A diferença do formulário é o atributo “*acces_type*”, selecionado no menu *dropdown* que vem da entidade “*accesValidation*”, para definir o tipo de acesso que o usuário possuirá, limitado ou ilimitado. No Apêndice C é possível ter uma visão geral do funcionamento do aplicativo presente na Figura 45.

3.8 Segurança no Back-End com *Supabase*

No contexto da segurança no back-end com *Supabase*, a comunicação entre o banco de dados e o aplicativo é estabelecida por meio do endereço do *Supabase* e da chave anônima fornecida durante a instalação. Essa chave, uma *string* criptografada de 32 caracteres, deve ser alterada utilizando o gerador de chaves disponível no site oficial do *Supabase*.

É destacado que a chave anônima é gerada a partir de uma *string* específica, a qual é submetida a um processo de criptografia. Para assegurar a segurança, é crucial alterar essa chave periodicamente.

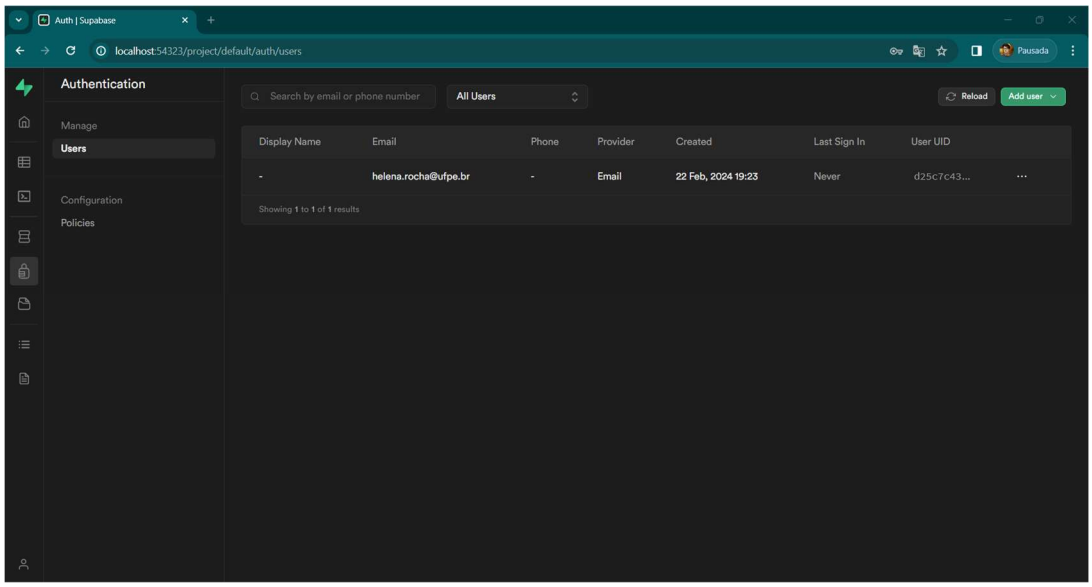
Os *JSON Web Tokens* (JWTs), utilizados no *Supabase*, são objetos JSON codificados, assinados e transmitidos como *strings*. Esses *tokens* são distribuídos aos usuários como uma forma de comprovar seu direito de acesso a determinado conteúdo no serviço ou site. É ressaltado que qualquer pessoa que possua a *jwt_secret* pode criar novos *tokens* e verificar a validade dos existentes.

Feito isso, foram criadas usuários, Figura 29 e políticas de autenticação,

Figura 30 utilizando a dashboard do *Supabase*. Essas políticas podem ser definidas individualmente para cada entidade e definir quais comandos e usuários serão permitidos nas ações definidas. Neste trabalho, a maioria das entidades até o momento permanece com autenticação de chave anônima e usuário "*anon*" sem a utilização de JWTs específicos que podem ser definidos como *tokens*.

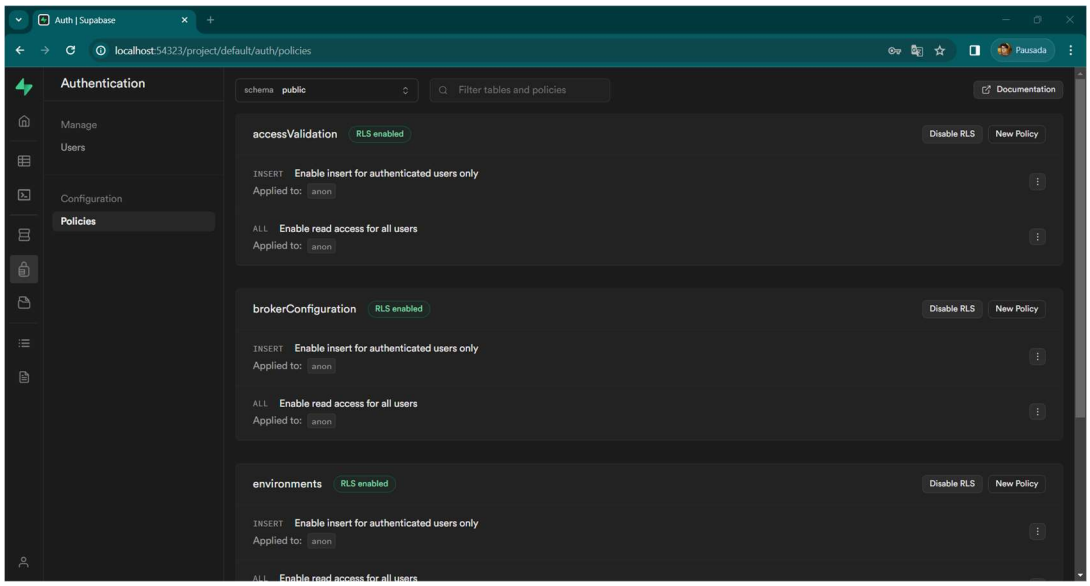
O *anon key*, destina-se a ser enviado pelos desenvolvedores junto com suas solicitações de API sempre que desejarem interagir com o banco de dados *Supabase*, permitindo uma comunicação eficiente e segura entre o aplicativo e o back-end, (Supabase, 2024).

Figura 29 – Interface para a criação de usuários que gerenciam o banco de dados.



Fonte: O'Autor.

Figura 30 – Interface em que se configuram as políticas das entidades.



Fonte: O'Autor.

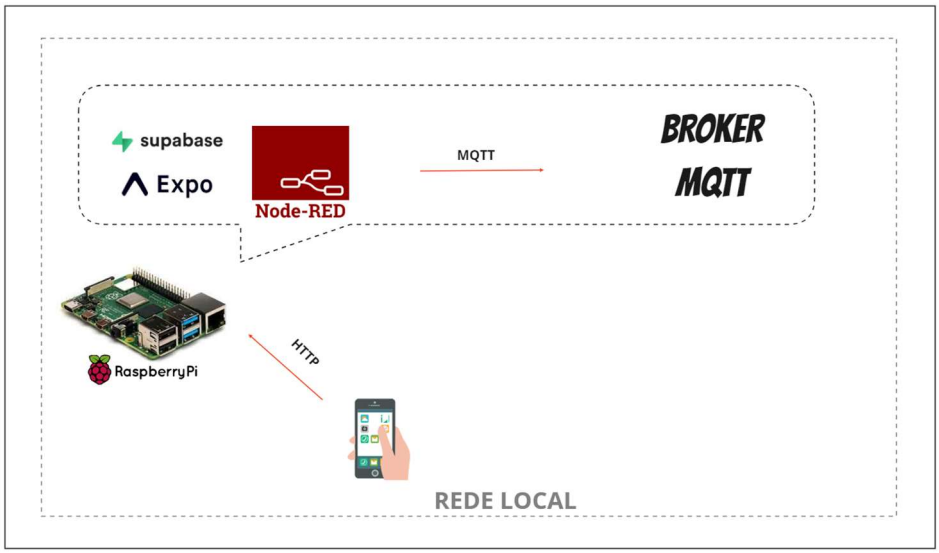
4 RESULTADOS

Durante os testes, uma rede local foi estabelecida para integrar a *Raspberry Pi* contendo o *Expo* e o *Supabase*, o aplicativo móvel contendo o *Expo Go* e o *Node-RED* com *WebSocket* e um *broker* MQTT, essa arquitetura pode ser observada na Figura 31. Nesse teste foram observadas as questões de incompatibilidade das bibliotecas com o protocolo MQTT, por isso, para a mensagem ser lida pelo *broker*, foi necessária a adição da ponte para ocorrer a transferência do dado.

Com a finalidade de depurar toda a comunicação entre o aplicativo e o *Node-RED*, foi utilizado o *software Wireshark* aplicado nas portas do *WebSocket* e do MQTT, como pode ser visto na Figura 32. Assim, observo-se a abertura de comunicação com o "*handshake* triplo" do TCP depois é aberta a comunicação HTTP e por fim aparece o protocolo *WebSocket*. Outra informação importante é que o MQTT enviado aparece

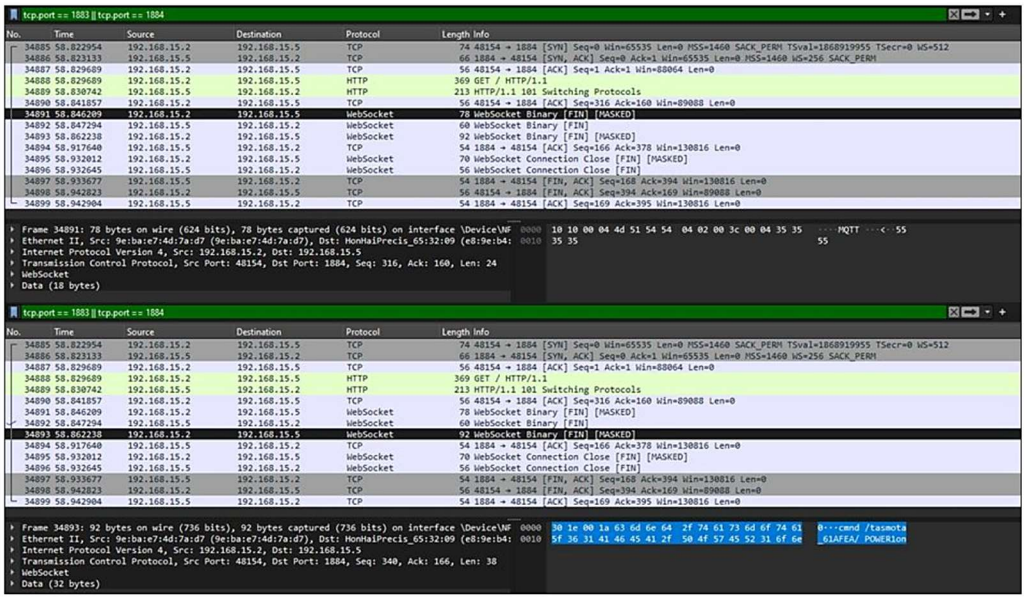
em forma de texto e o tópico e a mensagem, enviados ao apertar o botão, são recebidos como dado.

Figura 31 – Arquitetura de rede local com todas as aplicações embarcadas na *Raspberry*.



Fonte: O'Autor.

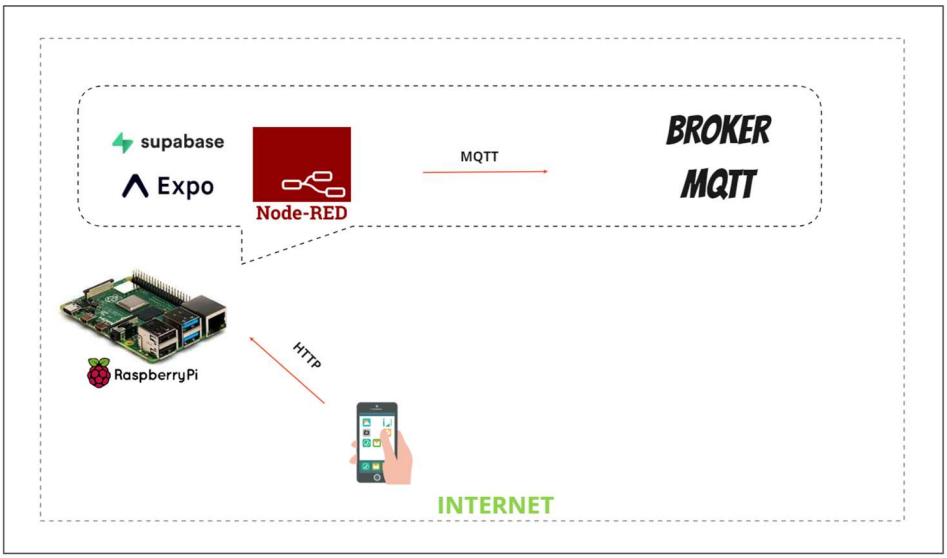
Figura 32 – Teste realizado no *Wireshark*.



Fonte: O'Autor.

Depois de testado em rede local, tanto o *Raspberry* quanto o celular foram colocados na internet e com os ajustes feitos no primeiro teste a transferência de dados foi realizada. A ilustração desse procedimento pode ser observada na Figura 33.

Figura 33 - Arquitetura com as aplicações conectadas a internet.



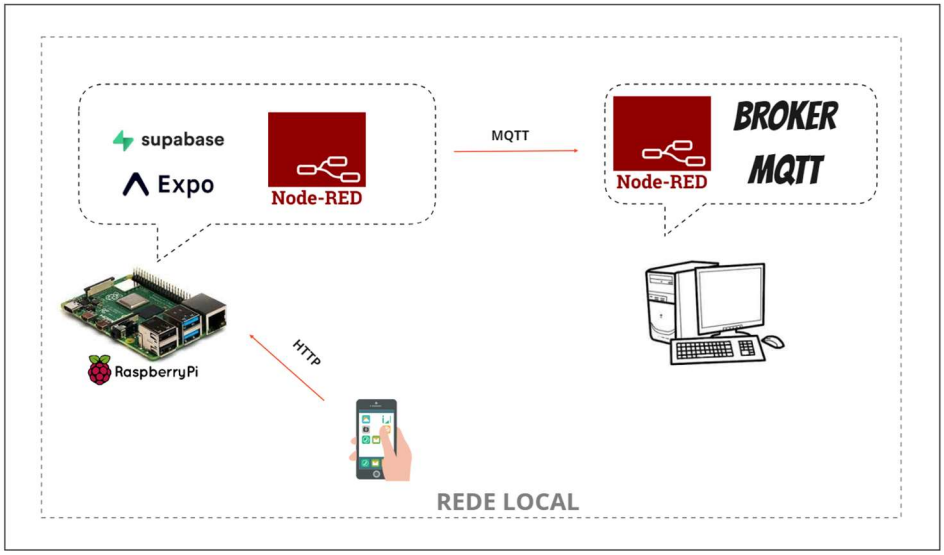
Fonte: O'Autor.

Com a finalidade de comprovar ainda mais que a aplicação produzida estava funcionando como esperado, o *broker* de teste foi colocado adicionado a outra máquina, como mostrado na Figura 34 e novamente a transferência de dados ocorreu com sucesso.

Para reproduzir o ambiente real foi utilizado o *broker* mosquitto disponibilizado em (Mosquitto, 2024) ao invés de um *broker* local, e a mensagem MQTT foi novamente recebida pelo *broker*. Na Figura 35 pode ser observado a arquitetura do teste.

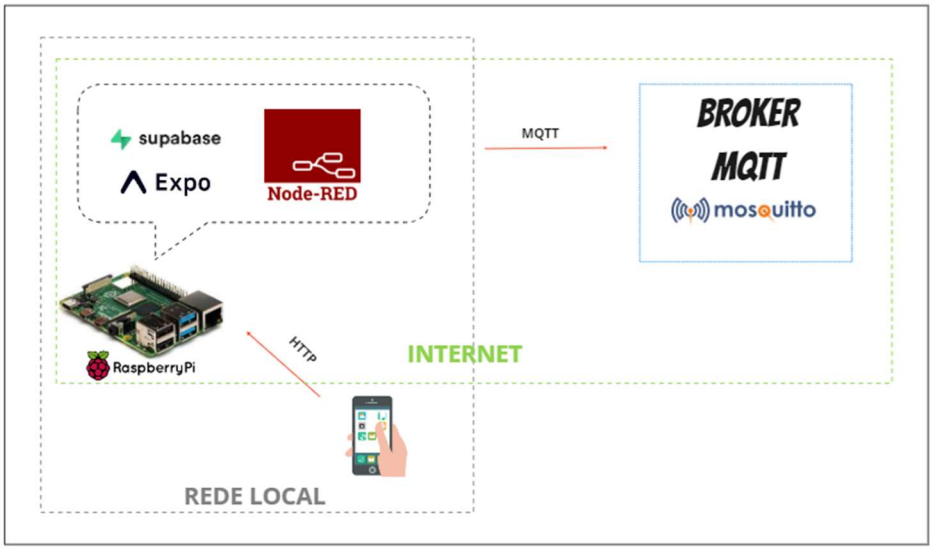
Finalmente, a aplicação foi levada ao ambiente real e ao apertar o botão produzido no aplicativo da *Expo Go*, a porta física de um dos laboratórios foi aberta, validando assim a aplicação desenvolvida.

Figura 34 - Arquitetura de rede local com o *broker* numa máquina diferente.



Fonte: O'Autor.

Figura 35 – Arquitetura de rede utilizando *broker* mosquitto.



Fonte: O'Autor.

5 CONCLUSÕES E PROPOSTAS DE CONTINUIDADE

O desenvolvimento do aplicativo de controle de acesso apresentado neste trabalho representa um passo significativo na busca por soluções inovadoras e eficientes para a gestão de ambientes físicos. Ao longo das seções detalhadas, foi possível compreender o processo de criação do aplicativo, desde a configuração da *Raspberry Pi* até a integração de protocolos como MQTT e *WebSocket*.

A arquitetura do sistema, centrada no *Raspberry Pi*, *Expo CLI* e *Supabase*, proporciona uma base sólida para a implementação das funcionalidades necessárias. Destaca-se a escolha da *Raspberry Pi* 4 com 8GB de RAM, evidenciando o comprometimento em oferecer uma solução econômica e acessível para a universidade. A configuração do sistema operacional *Raspbian* e a instalação do ambiente de desenvolvimento *Expo CLI* foram etapas cruciais para o prosseguimento do projeto.

A integração do *Supabase* desempenha um papel central no armazenamento e gestão de informações críticas. As tabelas do banco de dados, como "*accessValidation*", "*brokerConfiguration*", "*environments*" e "*users*" foram projetadas para garantir uma estrutura coesa e eficaz. A comunicação segura entre o aplicativo e o *Supabase* foi estabelecida por meio da chave anônima.

O desenvolvimento do front-end do aplicativo, destacado na seção 3.4, evidencia a preocupação com a experiência do usuário. Desde a tela de abertura até a gestão de permissões, cada elemento foi cuidadosamente projetado para proporcionar uma interação intuitiva e eficaz. A utilização de bibliotecas como "*react-native*" e a integração de recursos visuais, como a logo do Departamento de Engenharia Elétrica, contribuem para a identidade visual do aplicativo.

A implementação do controle de acesso, utilizando protocolos como MQTT e *WebSocket*, enfrentou desafios superados com soluções engenhosas. A escolha de fazer uma ponte com o *Node-RED* para lidar com a questão do protocolo *WebSocket* demonstra flexibilidade e adaptabilidade diante de obstáculos técnicos.

O aplicativo de controle de acesso desenvolvido neste trabalho não apenas representa uma solução prática para a gestão de ambientes físicos, mas também

serve como um exemplo de integração de tecnologias e boas práticas de desenvolvimento.

O trabalho realizado até o momento proporciona uma base sólida para futuras melhorias e expansões do aplicativo, consolidando-se como um marco no contexto de automação na borda de controle de acesso.

As propostas de continuidade do trabalho incluem a implementação de medidas de segurança adicionais, como a alteração periódica da chave e a utilização de *JSON Web Tokens*. Além disso, está prevista a introdução de uma dupla validação de usuário, utilizando dados dos estudantes fornecidos pelas coordenações dos cursos de Engenharia Elétrica e Engenharia de Controle e Automação.

Para reforçar a segurança, deverá ser incorporada a capacidade de configurar dias e horários específicos para a permissão de acesso aos ambientes para cada usuário. Um gerenciador das configurações do *broker* poderá ser implementado, permitindo ajustes eficientes nos parâmetros do *broker*, como *host*, porta, usuário e senha.

Outra proposta envolve o desenvolvimento de uma biblioteca personalizada que se comunique diretamente com o *broker* através do protocolo MQTT, otimizando a eficiência da comunicação. Além disso, a integração da tecnologia NFC deverá ser realizada para permitir o acesso independente de redes locais.

Por fim, poderá ser desenvolvida uma versão APK independente da *Expo*, visando reduzir a dependência do servidor local e proporcionar maior flexibilidade aos usuários, permitindo a instalação do aplicativo sem a necessidade de acesso à infraestrutura *Expo*. Essas propostas buscam fortalecer a segurança, expandir funcionalidades e aprimorar a usabilidade do sistema de controle de acesso em desenvolvimento.

REFERÊNCIAS

A História dos Banco de Dados. **DEVMEDIA**. Disponível em: <https://www.devmedia.com.br/a-historia-dos-banco-de-dados/1678>. Acesso em: 22 abr. 2023.

ABNT - ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. Medição de resistência de aterramento e de potenciais na superfície do solo em sistemas de aterramento. **ABNT NBR 15749**, p. 49, set. 2009.

AOVS SISTEMAS DE INFORMÁTICA S.A. Framework: o que é e pra que serve essa ferramenta? **Alura**, 2021. Disponível em: <https://www.alura.com.br/artigos/framework-o-que-e-pra-que-serve-essa-ferramenta?utm_term=&utm_campaign=&utm_source=adwords&utm_medium=ppc&hsa_acc=7964138385&hsa_cam=20946398532&hsa_grp=153091871930&hsa_ad=688089973825&hsa_src=g&hsa_tgt=dsa-2258482181923&>. Acesso em: 05 fev. 2024.

APPLE INC. Swift. Uma linguagem aberta e poderosa para todo mundo criar apps incríveis. **Apple**, 2024. Disponível em: <https://www.apple.com/br/swift/#:~:text=Swift%20%C3%A9%20uma%20linguagem%20de,mais%20liberdade%20para%20os%20desenvolvedores.>. Acesso em: 02 fev. 2024.

BEM vindo ao Raspbian. **Raspbian**. Disponível em: <https://www.raspbian.org/>. Acesso em: 05 jan. 2024.

BITTENCOURT, Jeniffer. Desenvolvimento de Apps Mobile: por onde começar? **alura**, 2021. Disponível em: <https://www.alura.com.br/artigos/desenvolvimento-apps-mobile-por-onde-comecar>. Acesso em: 06 jan. 2024.

CNN. NFC no celular: o que é, como funciona e para que serve a tecnologia. **cnnbrasil**, 2023. Disponível em: <https://www.cnnbrasil.com.br/tecnologia/nfc/#:~:text=NFC%20%C3%A9%20uma%20tecnologia%20que,e%20seguran%C3%A7a%20para%20os%20usu%C3%A1rios.>. Acesso em: 23 dez. 2023.

ELMASRI, Ramez. **Sistemas de banco de dados**. 1. ed.

EXPO. **Expo**, 2024. Disponível em: <https://expo.dev/>. Acesso em: 07 jan. 2024.

FOUNDATION, Raspberry P. Sistema operacional Raspberry Pi. **Raspberry Pi**. Disponível em: <https://www.raspberrypi.com/>. Acesso em: 05 jan. 2024.

FUNDAÇÃO RASPBERRY PI. Raspberry Pi. **Raspberry Pi**, 2024. Disponível em: <https://www.raspberrypi.com/>. Acesso em: 17 jan. 2024.

GOOGLE. Flutter. **Flutter**, 2024. Disponível em: <https://flutter.dev/?gad_source=1&gclid=EAIaIQobChMI5biixMe8hAMVlkVIAB3lCg9bEAAYASAAEgIfevD_BwE&gclsrc=aw.ds>. Acesso em: 03 fev. 2024.

GREGOR HOHPE, Bobby W. **Enterprise Integration Patterns**.

GRUPO DE DESENVOLVIMENTO GLOBAL POSTGRESQL. Sobre. **PostgreSQL**, 2024. Disponível em: <https://www.postgresql.org/about/>. Acesso em: 07 jan. 2024.

JETBRAINS. Kotlin. **Kotlin**. Disponível em: <https://kotlinlang.org/>. Acesso em: 03 fev. 2024.

JR, Elemar. **Fundamentos para arquitetura de sistemas escaláveis**. 3. ed.

MICHAEL E. WHITMAN, Herbert J. M. **Principles of Information Security**. 4ª. ed.

MICROSOFT. Um tour pela linguagem C#. **Microsoft**, 2023. Disponível em: <https://learn.microsoft.com/pt-br/dotnet/csharp/tour-of-csharp/>. Acesso em: 03 fev. 2024.

MICROSOFT. O que é a computação de borda? **Azure**, 2024. Disponível em: <https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-edge-computing>. Acesso em: 16 jan. 2024.

MOSQUITTO. **Mosquitto**, 2024. Disponível em: <https://test.mosquitto.org/>. Acesso em: 10 fev. 2024.

MOZILLA CORPORATION. O que é JavaScript? **mdn_**, 2024. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/What_is_JavaScript>. Acesso em: 03 fev. 2024.

MQTT. **MQTT**, 2022. Disponível em: <https://mqtt.org/>.

NIELS FERGUSON, Bruce S. T. K. **Cryptography Engineering:** Design Principles and Practical Applications.

NODE-RED. **Node-RED:** Low-code programming for event-driven applications., 2024. Disponível em: <https://nodered.org/>.

O que é IaaS (infraestrutura como serviço)? **aws**, 2024. Disponível em: <https://aws.amazon.com/pt/what-is/iaas/>. Acesso em: 03 jan. 2024.

O que é PaaS? **azure**, 2022. Disponível em: <https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-paas#:~:text=O%20PaaS%20(plataforma%20como%20servi%C3%A7o,empresariais%20habilitados%20para%20a%20nuvem>. Acesso em: 03x jan. 2023.

O que é um servidor em computação? **Controle Net**, 2024. Disponível em: <https://www.controle.net/faq/o-que-sao-servidores>. Acesso em: 17 fev. 2024.

ORACLE. Java. **Java**, 2024. Disponível em: <https://www.java.com/pt-BR/>. Acesso em: 03 fev. 2024.

PAIXÃO, João R. D. Desenvolvimento Backend: Os bastidores de um software. **Medium**, 2020. Disponível em: <https://medium.com/fretebras-tech/desenvolvimento-backend-os-bastidores-de-um-software-f5e558f3f61c>. Acesso em: 06 fev. 2024.

PROTOCOLO MQTT: O Que é, Como Funciona e Vantagens. **Automação Industrial**, 15/03/2023. Disponível em: <https://www.automacaoindustrial.info/mqtt/>. Acesso em: 20 abr. 2023.

SADALAGE, Pramod J.; FOWLER, Martin. **NoSQL Distilled:** A Brief Guide to the Emerging World of Polyglot Persistence. 1ª. ed.

SINGH, Nachiketa. Um guia para iniciantes em desenvolvimento back-end. **DZone**, 2009. Disponível em: <https://dzone.com/articles/a-beginners-guide-to-back-end-development>. Acesso em: 15 dez. 2023.

SUPABASE. **Supabase**, 2024. Disponível em: <https://supabase.com/>. Acesso em: 10 jan. 2024.

TANENBAUM, Andrew S. **Computer Networks**. 4. ed.

THOMAS ERL, Zaigham M. E. R. P. **Cloud Computing:** Concepts, Technology & Architecture. 1ª. ed.

UM guia sobre códigos QR e como fazer sua leitura. **kaspersky**, 2023. Disponível em: <https://www.kaspersky.com.br/resource-center/definitions/what-is-a-qr-code-how-to-scan>. Acesso em: 27 dez. 2023.

WEBSOCKETS. **mdn web docs**. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/API/WebSockets_API>. Acesso em: 15 fev. 2024.

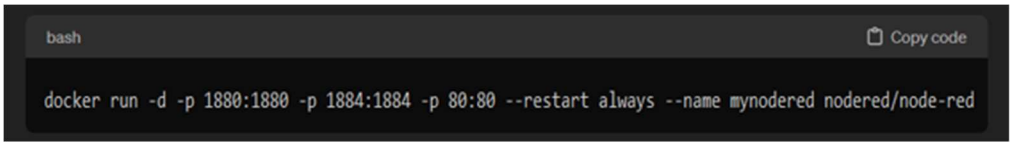
WELCOME to Raspbian. **Raspbian**. Disponível em: <https://www.raspbian.org/>. Acesso em: 05 jan. 2024.

ZUBAIR SHARIF, Low T. J. M. A. M. Y. D. K. Smart Home Automation by Internet-of-Things Edge. **(IJACSA) International Journal of Advanced Computer Science and Applications**, v. XIII, n. 4, 2022.

APÊNDICE A – CONFIGURAÇÃO DO NODE-RED NA RASPBERRY PI: PASSO A PASSO

1. Com o *Docker* já instalado na *Raspberry Pi*, inicie o *container* do *Node-RED*: execute o comando da Figura 36.

Figura 36 - Comando de inicialização de *container Node-RED*.



Fonte: O'Autor.

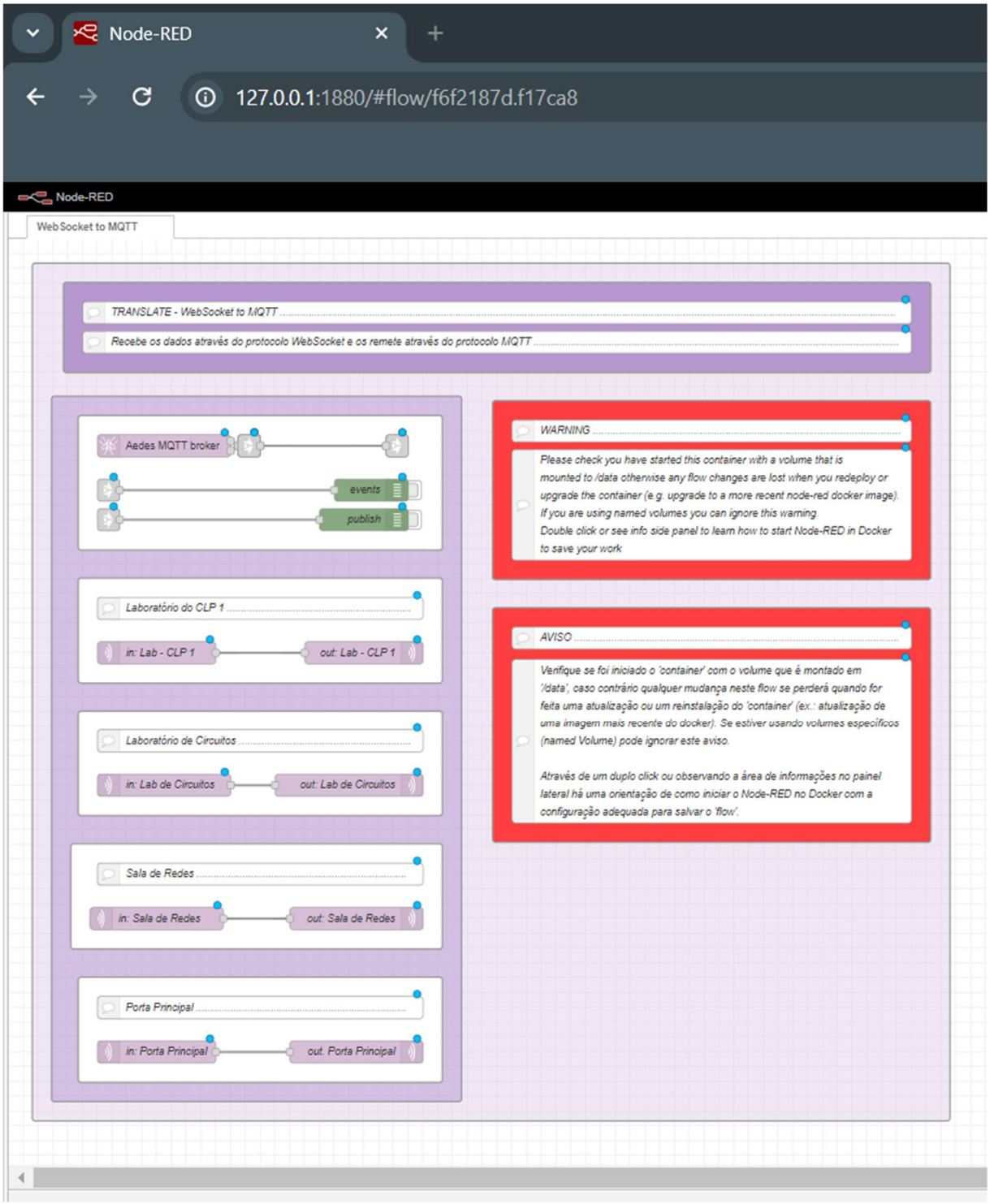
O comando da Figura 36 cria e inicia um *container* do *Node-RED*, é importante utilizar o parâmetro "-- *restart always*" para que o *container* seja reiniciado automaticamente.

A porta 1880 é aberta para ser possível acessar o *Node-RED* fora do *container*, a porta 1884 é aberta pois é utilizada na comunicação via *websocket* e a porta 80 é aberta para comunicação MQTT.

2. Abra o navegador e digite o endereço 127.0.0.1:1880 para acessar o *Node-RED*.
3. Adicione o fluxo criado neste trabalho no *Node-RED* para que a ponte com o *broker* que se comunica com as portas físicas seja encontrado. Na Figura 37 é possível observar uma imagem do fluxo criado.

O fluxo se inicia com o *broker* para comunicação *websocket* através dos blocos de entrada com seus respectivos tópicos MQTT e finaliza com os blocos de saída que enviam via MQTT para o *broker* que possui acesso as portas. Foi adicionado também blocos de *debug* para depurar o código. Dentre as configurações dos blocos estão os tópicos, endereços de ip, portas e qualidade de serviço. Todos esses parâmetros foram definidos de acordo com os hardwares que não fizeram parte desta pesquisa. É importante salientar que a qualidade de serviço utilizada, foi a do tipo 2.

Figura 37 – Fluxo criado no Node-RED para funcionar como intermediário.

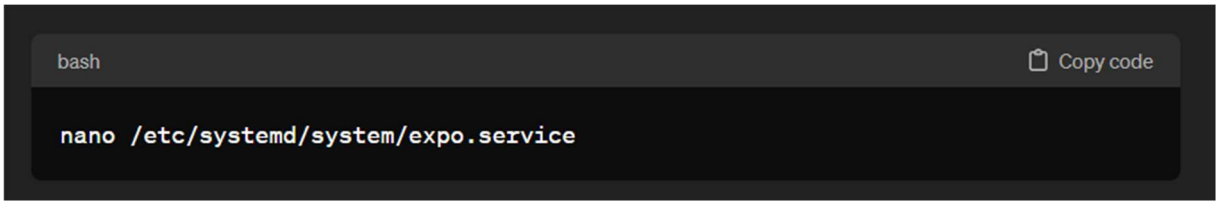


Fonte:O'Autor.

APÊNDICE B – CONFIGURAÇÃO DO SERVIÇO *EXPO* NA *RASPBERRY PI*: PASSO A PASSO

1. Criar o arquivo de serviço: abra um terminal no Raspberry Pi e execute o comando da Figura 38 para criar o arquivo de serviço do Expo:

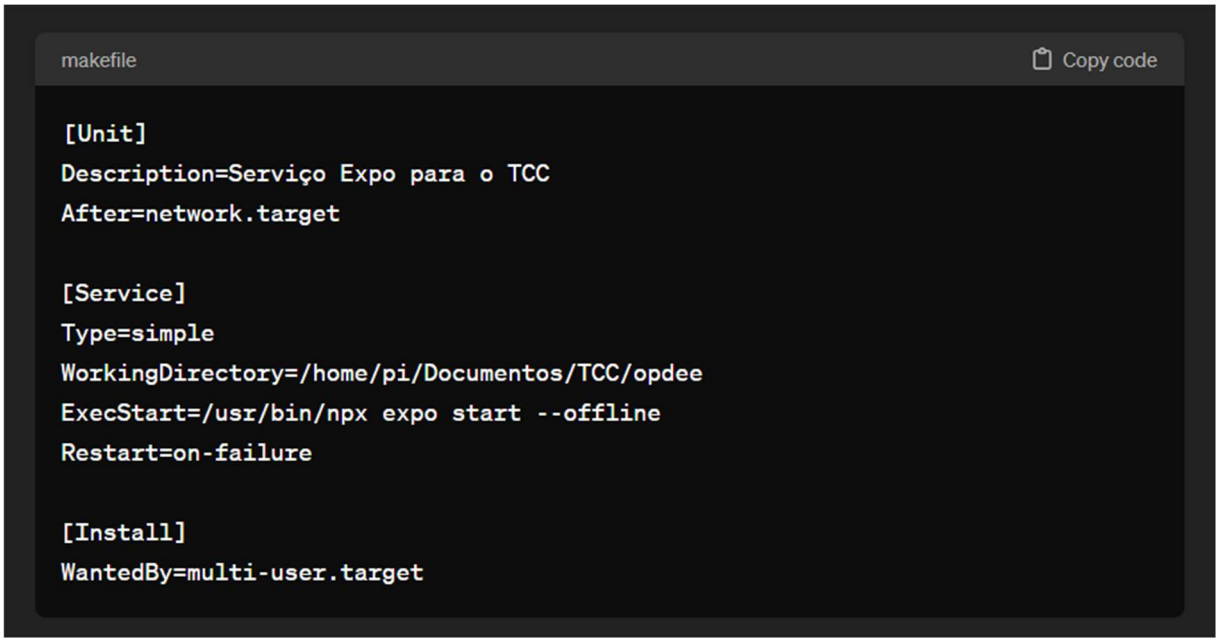
Figura 38 – Comando para criar arquivo de serviço.



Fonte: O'Autor.

2. Adicionar as configurações ao arquivo: Dentro do arquivo `expo.service`, escreva o código observado na Figura 39.

Figura 39 – Código para inicializar o serviço expo do projeto desenvolvido.



Fonte:O'Autor.'

3. Salvar e sair do editor: pressione “Ctrl + O” para salvar o arquivo e “Ctrl + X” para sair do editor Nano.
4. Recarregar o *systemd*: atualize o *systemd* para reconhecer as alterações feitas no arquivo de serviço, utilizando o código da Figura 40 no terminal.

Figura 40 – Comando para atualizar o systemd.

Fonte: O’Autor.

5. Iniciar o serviço Expo: escreva o comando da Figura 41 no terminal.

Figura 41 – Comando para iniciar o serviço Expo criado.

Fonte: O’Autor.

6. Verificar o status do serviço: verifique se o serviço Expo está em execução corretamente utilizando o comando da

Figura 42 – Comando que verifica o status do serviço Expo.

Fonte: O’Autor.

Este comando fornecerá informações sobre o estado atual do serviço *Expo*, incluindo se está em execução ou se ocorreu algum erro.

7. Habilitar o serviço na inicialização: para garantir que o serviço *Expo* seja iniciado automaticamente sempre que a *Raspberry Pi* for inicializada, utilize o comando da Figura 43.

Figura 43 – Habilitar o serviço *Expo* criado na inicialização.

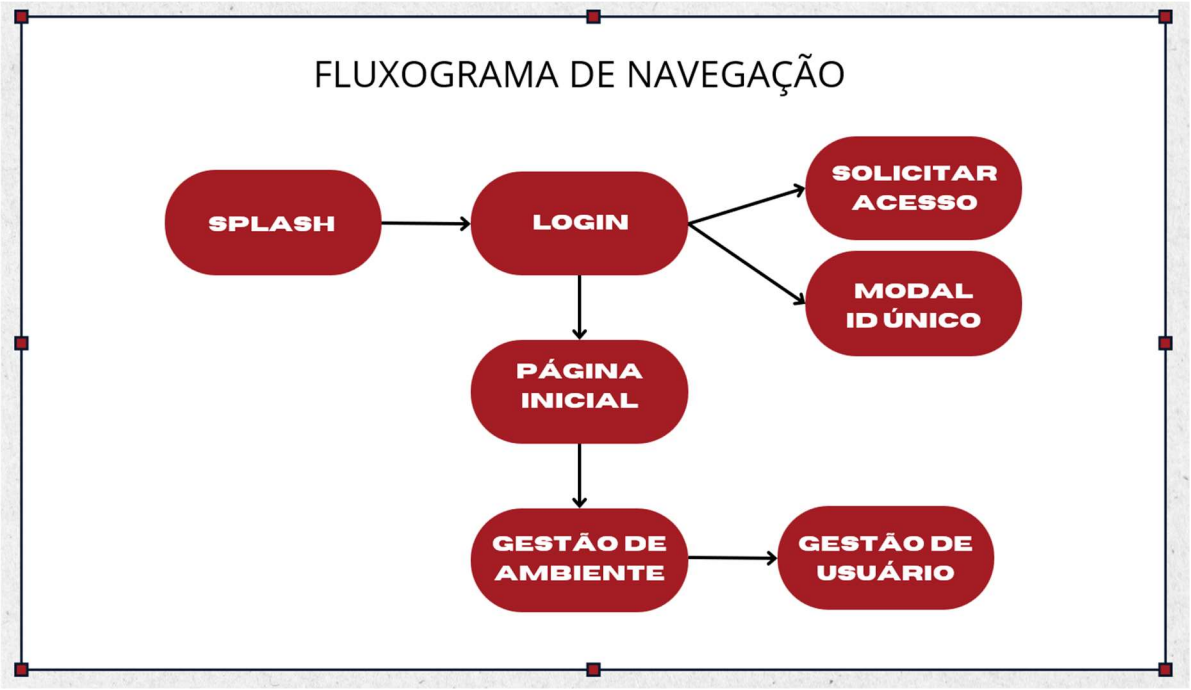
Fonte: O'Autor.

Após a execução desses passos, o serviço Expo será iniciado automaticamente na inicialização da *Raspberry Pi*. É importante se certificar do caminho do diretório do projeto e do comando "npx" estejam corretos no arquivo de serviço. Outro fator importante é utilizar o parâmetro "*--offline"* para que nenhuma porta adicional do roteador seja aberta.

APÊNDICE C – FLUXOGRAMA DO APLICATIVO DE CONTROLE DE ACESSO A AMBIENTES FÍSICOS

Na Figura 44, é possível observar a navegação por todas as telas do aplicativo. Ao entrar na aplicação, a tela splash é exibida, seguida pela tela de login. Nesta última, o usuário pode observar o ID único do dispositivo ou acessar a tela de solicitação de acesso ou prosseguir para a página inicial para acessar os ambientes, caso já esteja cadastrado. Se o usuário for do tipo coordenador, ele terá acesso à página de gestão de ambiente e, consequentemente, à página de gestão de usuários.

Figura 44– Fluxograma de navegação do aplicativo.



Fonte: O'Autor.

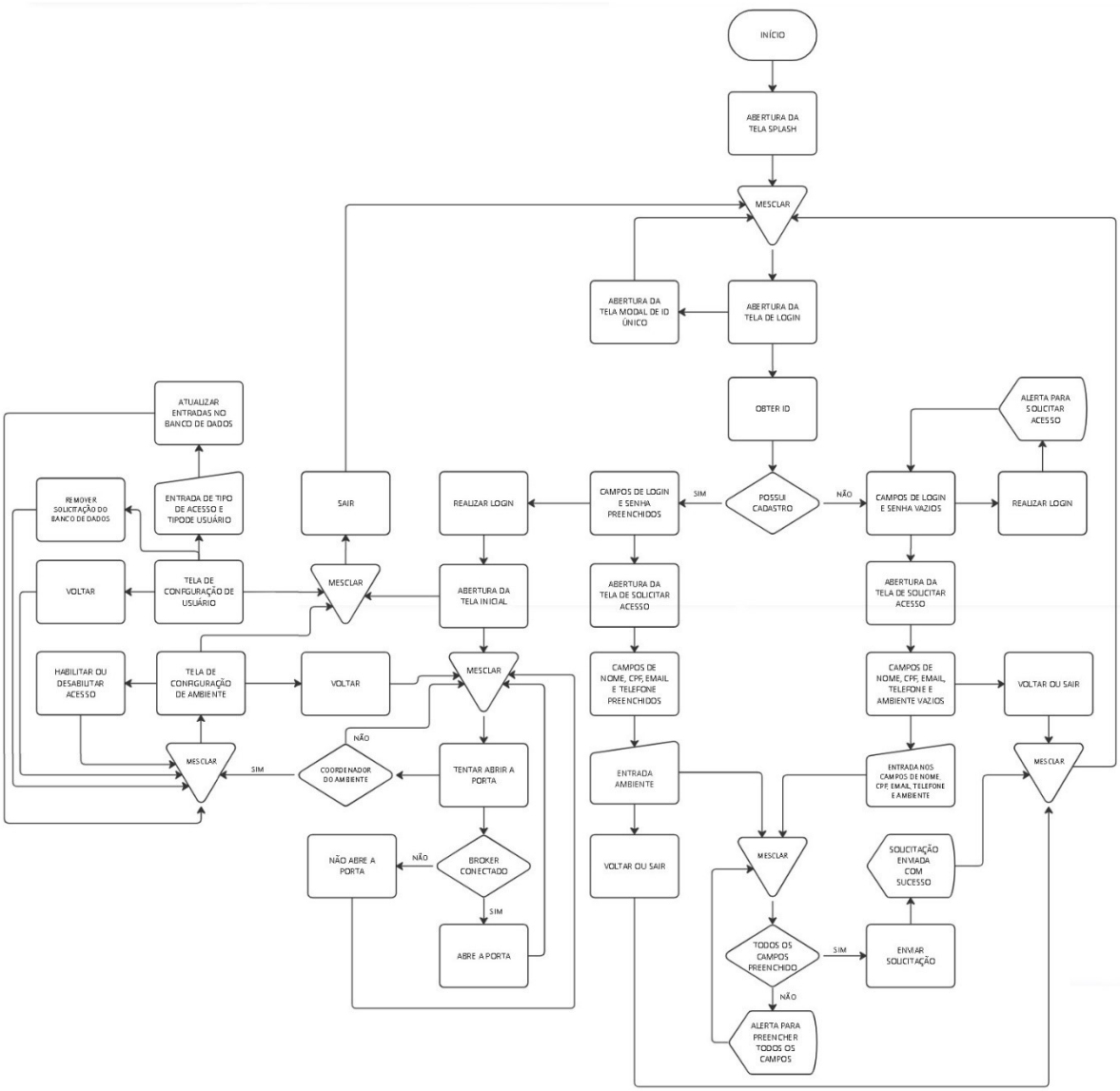
Na Figura 45, é possível observar a parte do servidor. Ao adentrar no aplicativo, é gerado o ID único do dispositivo e verificado se esse usuário existe no banco de dados. Se sim, os campos de usuário e senha na página de *login* são preenchidos, e o usuário pode acessar a página inicial. Se não, ao tentar acessar a página inicial, será emitido um alerta informando que o usuário não está cadastrado.

Ao pressionar o botão "ID único", é exibida uma tela modal com o ID único do dispositivo gerado anteriormente. Ao pressionar o botão "Solicitar acesso", outra tela é aberta com um formulário. Se o usuário já estiver cadastrado, a maioria dos campos é preenchida com informações obtidas do banco de dados. Caso contrário, os campos permanecem em branco, aguardando preenchimento. Se todos os campos forem preenchidos corretamente, ao pressionar o botão "Enviar", o formulário será enviado com sucesso e será exibido um alerta confirmando. Caso contrário, será exibido um alerta informando que os campos não foram preenchidos corretamente. Se o botão "Cancelar" for pressionado, a página de *login* será reaberta.

Ao acessar a página inicial, o usuário terá acesso aos ambientes disponíveis para ele. O usuário pode ser do tipo "usuario" (com acesso apenas à abertura de portas) ou "coordenador" (com acesso à abertura de portas, gestão de ambientes e gestão de usuários). Ao clicar no botão da porta, uma requisição é enviada ao broker e a porta é aberta. Ao clicar no botão de configurações, a tela de gestão de ambientes é aberta. Ao clicar no *switch*, o valor do botão é invertido e o usuário perde/ganha acesso ao ambiente. Ao clicar no nome do usuário, a tela de gestão de usuário é aberta, onde é possível definir o tipo de usuário (“usuário” ou “coordenador”) e o tipo de acesso (limitado ou ilimitado). Ao pressionar o botão "Enviar", uma requisição de atualização é feita ao banco. Ao pressionar "Remover", uma requisição de deleção é feita no banco. Ao pressionar "Cancelar", a tela de gestão de ambientes é reaberta.

Ao pressionar "Sair" em qualquer uma das telas, a tela de *login* é reaberta.

Figura 45 – Fluxograma geral do aplicativo.



Fonte: O’Autor.