UNIVERSIDADE FEDERAL DE PERNAMBUCO

Universidade Federal de Pernambuco
Centro de Tecnologia e Geociências
Departamento de Eletrônica e Sistemas

Graduação em Engenharia Eletrônica

Gabriel Firmo Soares de Queiroz

Sistema de controle de acesso por meio da detecção de pacotes Wi-Fi

Recife

2025

Gabriel Firmo Soares de Queiroz

Sistema de controle de acesso por meio da detecção de pacotes Wi-Fi

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

Orientador(a): Prof. José Sampaio de Lemos Neto, Ph.D.

Recife
2025

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

Queiroz, Gabriel Firmo Soares de.
Sistema de controle de acesso por meio da detecção de pacotes Wi-Fi / Gabriel Firmo Soares de Queiroz. - Recife, 2025.
84 : il., tab.

Orientador(a): José Sampaio de Lemos Neto
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia Eletrônica - Bacharelado, 2025.
Inclui referências, apêndices.

1. Internet das coisas. 2. Sistemas embarcados. 3. Redes de computadores. 4. Computação em nuvem. 5. Controle de acesso. 6. RSSI. I. Lemos Neto, José Sampaio de. (Orientação). II. Título.

620 CDD (22.ed.)

Gabriel Firmo Soares de Queiroz

Sistema de controle de acesso por meio da detecção de pacotes Wi-Fi

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

Aprovado em: 30/05/2025

Banca Examinadora

Prof. José Sampaio de Lemos Neto, Ph.D.
Universidade Federal de Pernambuco

Prof. João Marcelo Xavier Natário Teixeira, D.Sc.
Universidade Federal de Pernambuco

À memória de meus queridos

tios, George e Gilberto.

Levo comigo suas lembranças e

ensinamentos em cada passo

desta jornada.

Agradecimentos

Agradeço, primeiramente, a Deus, pela saúde, força e oportunidade de concluir mais essa etapa da minha vida.

Ao meu orientador, Prof. Dr. José Sampaio de Lemos Neto, pela paciência, pela orientação técnica precisa e pelo incentivo contínuo à excelência.

À Universidade Federal de Pernambuco e ao Departamento de Eletrônica e Sistemas, por me proporcionarem uma formação de qualidade e estrutura para a realização deste projeto.

À minha família, pelo amor incondicional, apoio constante e por acreditarem em mim mesmo quando eu duvidei.

Aos meus amigos e colegas de curso, por todas as trocas de conhecimento, pela parceria nas dificuldades e pelas boas risadas nos momentos de estresse.

Por fim, a todos que, direta ou indiretamente, contribuíram para a concretização deste trabalho — o meu mais sincero obrigado.

If I have seen further it is by standing on the shoulders of Giants.

Isaac Newton

Resumo do Trabalho de Conclusão de Curso apresentado ao Departamento de Eletrônica e Sistemas, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Eletrônica(Eng.)

Sistema de controle de acesso por meio da detecção de pacotes Wi-Fi

Gabriel Firmo Soares de Queiroz

O rápido avanço das tecnologias de comunicação sem fio impulsiona a Internet das Coisas (IOT), conectando objetos cotidianos e viabilizando novas aplicações em diversos setores. Neste contexto, este trabalho apresenta o desenvolvimento e a validação de um sistema de controle de acesso, idealizado para as bancadas do Laboratório Didático de Eletrônica (LDE), que utiliza a detecção de sinais Wi-Fi emitidos por *smartphones*. As atividades realizadas incluem: o desenvolvimento do sistema embarcado no microcontrolador ESP32, responsável por capturar pacotes Wi-Fi, extrair o endereço *Medium Access Control* (MAC) e a intensidade do sinal recebido RSSI, e aplicar a lógica de acesso baseada na verificação do MAC em lista de permissão e na superação de um limiar de RSSI experimentalmente determinado; a criação do *backend* na nuvem AWS, composto por uma *API REST Java/Spring Boot* e um banco de dados *PostgreSQL* para gerenciar cadastros e registros de acesso; o desenvolvimento de uma interface web *Angular* para interação do usuário; e a validação experimental do modelo de perda de caminho log-distância para sinais de radiofrequência no ambiente de teste.

Palavras-chave: Internet das coisas; controle de acesso; sistemas embarcados; redes de computadores; computação em nuvem; RSSI.

Abstract of Course Conclusion Work, presented to Departament of Eletronic and Systems, as a partial fulfillment of the requirements for the degree of Bachelor of Electronic Engineering(Eng.)

Access control system through Wi-Fi packet detection

Gabriel Firmo Soares de Queiroz

The rapid advancement of wireless communication technologies drives the IOT, connecting everyday objects and enabling new applications across various sectors. In this context, this work presents the development and validation of an access control system, conceived for the workbenches of the Laboratório Didático de Eletrônica (LDE), which uses the detection of Wi-Fi signals emitted by smartphones. The activities undertaken include: the development of the embedded system on the ESP32 microcontroller, responsible for capturing Wi-Fi packets, extracting the MAC address and the RSSI, and applying access logic based on checking the MAC against a whitelist and exceeding an experimentally determined RSSI threshold; the creation of the backend on the AWS cloud, comprising a *REST API Java/Spring Boot* and a *PostgreSQL* database to manage registrations and access logs; the development of a web interface *Angular* for user interaction; and the experimental validation of the log-distance path loss model for radio frequency signals in the test environment.

Keywords: Internet of things; access control; embedded systems; computer networks; cloud computing; RSSI.

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Acrônimos

ACID . . . . . Atômicas, Consistentes, Isoladas e Duráveis

AP . . . . . *Access Point*

API . . . . . *Application Programming Interface*

AWS . . . . . *Amazon Web Services*

BSS . . . . . *Basic Service Set*

CA . . . . . *Certificate Authority*

CI/CD . . . . . *Continuous Integration / Continuous Delivery*

CRC . . . . . *Cyclic Redundancy Check*

CSS . . . . . *Cascading Style Sheets*

ESP-IDF . . . . . *Espressif IoT Development Framework*

GCP . . . . . *Google Cloud Platform*

HTML . . . . . *Hypertext Markup Language*

HTTP . . . . . *Hypertext Transfer Protocol*

HTTPS . . . . . *Hypertext Transfer Protocol Secure*

I/O . . . . . Entrada/Saída

I2C . . . . . *Inter-Integrated Circuit*

IaaS . . . . . *Infrastructure as a Service*

IDE . . . . . *Integrated Development Environment*

IEEE . . . . . Instituto de Engenheiros Eletricistas e Eletrônicos

IOT . . . . . Internet das Coisas

IP . . . . . *Internet Protocol*

JSON . . . . . *JavaScript Object Notation*

LDE . . . . . Laboratório Didático de Eletrônica

lwIP . . . . . *Lightweight TCP/IP*

MAC . . . . . *Medium Access Control*

MIT . . . . . Massachusetts Institute of Technology

MQTT . . . . . *Message Queuing Telemetry Transport*

MVC . . . . . *Model-View-Controller*

OLED . . . . . Diodo orgânico emissor de luz

PaaS . . . . . *Platform as a Service*

QoS . . . . . *Quality of Service*

RDS . . . . . *Relational Database Service*

REST . . . . . *Representational State Transfer*

RMSE . . . . . *Raiz do Erro Médio Quadrático*

RSSI . . . . . Received Signal Strength Indicator

RTOS . . . . . Sistemas Operacionais de Tempo Real

SaaS . . . . . *Software as a Service*

SGBD . . . . . Sistema de Gerenciamento de Banco de Dados

SPA . . . . . *Single Page Application*

SQL . . . . . *Structured Query Language*

SSID . . . . . *Service Set Identifier*

SSL/TLS . . . . . *Secure Sockets Layer/Transport Layer Security*

TCP . . . . . *Transmission Control Protocol*

TI . . . . . Tecnologia da Informação

URL . . . . . *Uniform Resource Locator*

Wi-Fi . . . . . *Wireless Fidelity*

WPA2-PSK . . . . . *Wi-Fi Protected Access 2 - Pre-Shared Key*

WWW . . . . . *World Wide Web*

Sumário

Capítulo 1

Introdução

Nos últimos anos, o rápido avanço das tecnologias de comunicação sem fio, como *Wireless Fidelity* (Wi-Fi), *Bluetooth* e 5G, possibilitou a conexão de objetos cotidianos à internet, permitindo que dados sejam coletados e compartilhados de forma autônoma. Este fenômeno, conhecido como Internet das Coisas (IOT), tem se tornado um dos principais pilares da transformação digital em diversos setores, como saúde, segurança, indústria, transporte e automação residencial. Conforme ilustrado na Figura 1.1, a evolução da IOT desde seu surgimento em 1999 tem impactado significativamente diversas áreas do conhecimento (Ashton, 2009).

Neste contexto, a tecnologia Wi-Fi desempenha papel fundamental na IOT devido à sua ampla adoção, facilidade de integração e robustez nas conexões. A possibilidade de detectar dispositivos móveis por meio da análise dos pacotes Wi-Fi emitidos abre caminhos para aplicações práticas importantes, especialmente em sistemas de segurança e controle de acesso.

Este trabalho propõe a concepção de um sistema integrado utilizando o microcontrolador ESP32 para controle de acesso, inicialmente pensado para as bancadas do Laboratório Didático de Eletrônica (LDE) da Universidade Federal de Pernambuco, por meio da detecção dos sinais Wi-Fi emitidos por *smartphones*.

Figura 1.1: Linha do tempo da IOT (Fonte: [1] Adaptado pelo autor).

1.1 Justificativa

A busca por soluções eficientes e seguras para controle de acesso tem crescido devido às limitações e vulnerabilidades dos métodos tradicionais, como chaves físicas e sistemas baseados em senhas. Essas abordagens frequentemente resultam em problemas como perda, extravio e dificuldade de gerenciamento. Diante desse cenário, a utilização dos *smartphones*, dispositivos amplamente utilizados e presentes no cotidiano das pessoas, destaca-se como uma alternativa prática e eficiente. A detecção de sinais Wi-Fi emitidos por *smartphones* permite não apenas uma maior conveniência, segurança e gerenciamento, mas também possibilita o monitoramento em tempo real.

Este trabalho é relevante por explorar o potencial da tecnologia Wi-Fi associada à IOT, além de contribuir teoricamente com validações e ajustes do modelo log-distância aplicado à medição e análise de sinais Wi-Fi.

1.2 Objetivo Geral

O objetivo geral deste trabalho é desenvolver, implementar e validar um sistema integrado de controle de acesso utilizando a detecção e análise de pacotes

Wi-Fi emitidos por *smartphones*. Pretende-se configurar o microcontrolador ESP32 como componente central do sistema, integrado a uma infraestrutura de computação em nuvem para armazenamento, processamento e visualização dos dados coletados. Tornando-se uma alternativa aos métodos tradicionais de controle de acesso.

1.2.1 Objetivos específicos

Os objetivos específicos que deverão ser alcançados pelo trabalho são os seguintes:

- Configurar o módulo Wi-Fi do microcontrolador para funcionar como detector de pacotes;
- Implementar a máquina de estados responsável pela lógica da liberação de acesso;
- Desenvolver servidor *Web* para armazenar as informações dos usuários cadastrados;
- Desenvolver comunicação entre microcontrolador e servidor *Web* para acessar a lista de usuários cadastrados;
- Desenvolver interface gráfica (página *Web*) para realização de cadastro;
- Implementar as telas de status do sistema no *display* Diodo orgânico emissor de luz (OLED) presente na placa de desenvolvimento;
- Hospedar servidor e página *Web* num serviço de computação em nuvem;
- Verificar a validade do Modelo de perda de caminho log-distância;
- Definir o valor limiar do RSSI;

1.3 Organização do Trabalho

O conteúdo deste trabalho está dividido em seis capítulos e dois apêndices. As referências encontram-se logo após o último capítulo. A seguir, um resumo dos capítulos seguintes.

Capítulo 2. Apresenta a fundamentação teórica, abordando os conceitos e tecnologias essenciais ao desenvolvimento do projeto, como Wi-Fi, sistemas embarcados, desenvolvimento Web e computação em nuvem.

Capítulo 3. Descreve de forma detalhada a metodologia empregada, destacando as etapas e procedimentos de análise utilizados na validação do sistema proposto.

Capítulo 4. Apresenta a implementação prática do sistema, incluindo detalhes técnicos do desenvolvimento do *hardware* e *software* embarcados, servidor Web e interface gráfica.

Capítulo 5. Exibe os resultados obtidos, incluindo a validação do modelo log-distância, testes realizados para determinar o limiar do RSSI e a eficácia geral do sistema.

Capítulo 6. Encerra o trabalho com as considerações finais, apresentando as conclusões obtidas, dificuldades encontradas durante o desenvolvimento e sugestões para futuros estudos.

Apêndice A. Apresenta os passos necessários para configurar o serviço *Elastic Beanstalk* na *Amazon Web Services* (AWS).

Apêndice B. Apresenta os passos necessários para configurar o serviço RDS na AWS.

Capítulo 2

Fundamentação Teórica

Neste capítulo são apresentados os principais conceitos que dão suporte ao presente trabalho. Os tópicos abordados abrangem áreas-chave no campo da Tecnologia da Informação (TI), como comunicação sem fio, sistemas embarcados, desenvolvimento de aplicações Web e computação em nuvem.

Ao abordar o tema Wi-Fi, analisa-se seus protocolos, arquitetura e formato do quadro de dados, além do modelo de perda de caminho.

Em seguida, explora-se os sistemas embarcados, suas características e aplicações, mostrando exemplos de microcontroladores e apresentando os Sistemas Operacionais de Tempo Real (RTOS).

No contexto do desenvolvimento *Web*, examina-se os principais conceitos, técnicas e tecnologias envolvidas no processo de criação de sites e aplicações *Web*. Discutindo o papel das linguagens de programação e *frameworks* no desenvolvimento de soluções robustas e escaláveis.

Por fim, aborda-se a computação em nuvem, um paradigma que revolucionou o armazenamento e processamento de dados, permitindo o acesso a recursos computacionais de maneira flexível e sob demanda. Examinaremos os modelos de serviço, infraestrutura e arquiteturas envolvidos.

Padrão IEEE 802.11	Ano	Taxa de dados Máx.	Alcance	Frequência
802.11b	1999	11 Mbps	30 m	2.4 GHz
802.11g	2003	54 Mbps	30 m	2.4 GHz
802.11n (Wi-Fi 4)	2009	600 Mbps	70 m	2.4, 5 GHz
802.11ac (Wi-Fi 5)	2013	3.47 Gbps	70 m	5 GHz
802.11ax (Wi-Fi 6)	2020	14 Gbps	70 m	2.4, 5 GHz

Tabela 2.1: Resumo dos padrões IEEE 802.11 (Fonte: Kurose e Ross, 2021).

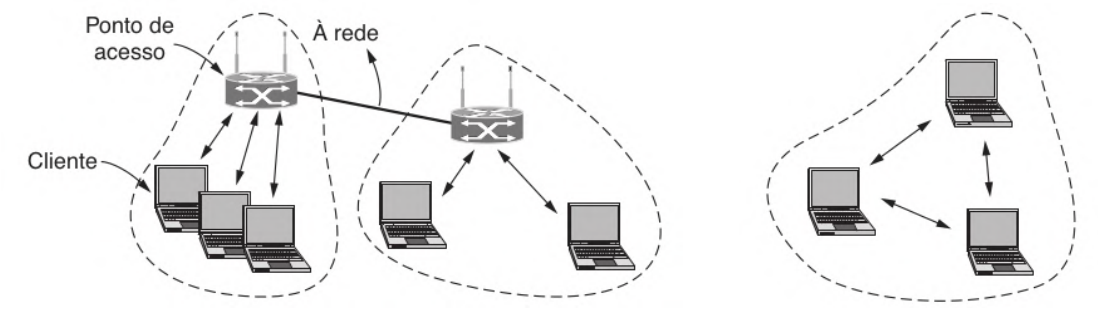
2.1 Wi-Fi: Padrão IEEE 802.11

O termo Wi-Fi trata-se de um apelido para o conjunto de padrões elaborado pelo Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) em meados da década de 1990, denominado IEEE 802.11, que define o funcionamento das redes sem fio, levando em consideração características como: modo e frequência de operação, taxa de transmissão, segurança, gerenciamento de energia e qualidade do serviço (QoS) [2]. Existem várias versões do padrão 802.11, tais quais: 802.11b, 802.11g, 802.11n, 802.11ac, 802.11ax, cujas características são mostradas na Tabela 2.1 [4]. O IEEE trabalha continuamente no desenvolvimento e aprimoramento do padrão 802.11, realizando atualizações periódicas para atender às demandas do mercado. Em 2024 foi lançada a emenda 802.11be, com o objetivo de melhorar a eficiência espectral e a capacidade de rede do Wi-Fi, especialmente em ambientes de alta densidade, possibilitando a utilização das faixas de frequência 6 e 7 GHz [13].

2.1.1 A arquitetura 802.11

As redes 802.11 podem ser usadas em dois modos. O modo mais popular é conectar clientes, como laptops e *smartphones*, a outra rede, como uma intranet da empresa ou a Internet. Esse modo aparece na Figura 2.1a e é denominado modo de infraestrutura, em que cada cliente está associado a um ponto de acesso (*Access Point* (AP)), que por sua vez, pode estar conectado a outra rede. Este conjunto de clientes e APs é conhecido também como *Basic Service Set* (BSS). O cliente transmite e recebe seus pacotes por meio do AP.

O outro modo, mostrado na Figura 2.1b, é uma rede *ad hoc*. Esse modo é uma



(a) Modo infraestrutura **(b)** Modo *ad hoc*

Figura 2.1: Arquitetura 802.11 (Fonte: Tanenbaum *et al.*, 2021).

coleção de computadores que estão associados de modo que possam enviar quadros diretamente uns aos outros, sem a necessidade de um AP [2].

No modo de infraestrutura, ao criar-se um AP, é necessário atribuir a ele um identificador alfanumérico, denominado *Service Set Identifier* (SSID), que será visível para os dispositivos clientes que busquem a conexão. O AP também deve especificar o número de canal que irá operar. Para a faixa de 2.4GHz existem 14 canais parcialmente sobrepostos, com largura de banda de 22 MHz cada um, como mostrado na Figura 2.2. O espectro disponível segue a regulamentação de cada país, no Brasil pode-se usar os canais de 1 a 13, correspondendo à faixa de frequência entre 2.400 e 2.483,5 MHz, porém em alguns países os canais 12 e 13 são restritos, como é o caso dos Estados Unidos [14].

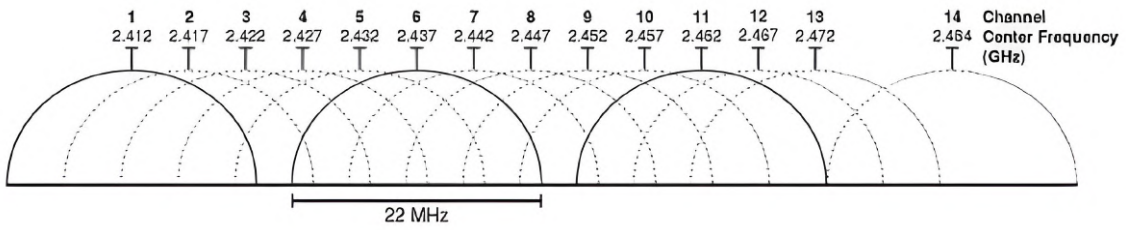


Figura 2.2: Espectro do 802.11 na faixa de 2.4GHz (Fonte: [3]).

Não há sobreposição entre quaisquer dois canais se, e somente se, eles estiverem separados por quatro ou mais canais. Em particular, nos países que só têm disponíveis os 11 primeiros canais, o conjunto 1, 6 e 11 é o único de três canais não sobrepostos, tornando-se uma escolha bastante popular quando há a necessidade de criação de 3 APs em localidades próximas [4].

2.1.2 O processo de autenticação e associação

No modo infraestrutura, um dispositivo conecta-se a uma rede Wi-Fi seguindo um processo chamado "autenticação e associação", que envolve a troca de quadros específicos entre o dispositivo cliente e o AP. Este processo segue os seguintes passos :

1. **Varredura (*Probing*)**: O dispositivo cliente inicia a busca por APs disponíveis, enviando um quadro de sonda de solicitação (*Probe request frame*). Este quadro contém informações sobre as capacidades do dispositivo, como frequência de rádio, taxa de dados e padrões Wi-Fi suportados.

2. **Resposta à sonda (*Probe response*)**: Os APs que receberem o quadro de sonda e que estão operando na mesma frequência de rádio do dispositivo, enviarão um quadro de resposta à sonda (*Probe response*) contendo informações sobre a rede, como o SSID, taxas de dados suportadas e informações de segurança.

3. **Autenticação**: O dispositivo cliente seleciona um AP e envia um quadro de autenticação. Se a rede Wi-Fi estiver protegida por uma senha, o cliente e o AP usarão um processo de troca de chaves, como o *Wi-Fi Protected Access 2 - Pre-Shared Key* (WPA2-PSK) ou outro semelhante, para verificar se a senha inserida pelo usuário está correta.

4. **Confirmação de autenticação**: Se o processo de autenticação for bem-sucedido, o AP enviará um quadro de confirmação de autenticação ao dispositivo cliente, confirmando que ele foi autenticado com sucesso.

5. **Associação**: Após a autenticação bem-sucedida, o dispositivo cliente enviará um quadro de solicitação de associação para o AP. Esse quadro contém informações adicionais sobre as capacidades do dispositivo e parâmetros de comunicação.

6. **Confirmação de associação**: Se o AP aceitar a solicitação de associação, ele enviará um quadro de confirmação ao dispositivo cliente, confirmando que ele está associado à rede Wi-Fi.

2.1.3 O quadro IEEE 802.11

O padrão 802.11 define três classes de quadros em trânsito: dados, controle e gerenciamento. Cada um deles possui cabeçalho com campos utilizados na subcamada MAC.

Os quadros de dados são usados para transmitir os dados do usuário na rede, como pacotes IP. Estes quadros são transmitidos pelos dispositivos finais na rede (por exemplo, *laptops* e *smartphones*), geralmente em resposta a um quadro de controle ou gerenciamento enviado pelo AP. Podem também ser utilizados para responder a solicitações de dados de outros dispositivos na rede.

Os quadros de controle são usados para controlar a transmissão de dados na rede e para gerenciar as conexões entre dispositivos. As funções deste tipo de quadro são: solicitar informações sobre o *status* da rede, ajustar a taxa de transmissão, informar outros dispositivos sobre o estado da conexão e sincronizar as transmissões.

Os quadros de gerenciamento são usados para controlar e gerenciar a configuração da rede, como a autenticação e associação de dispositivos, a descoberta de APs e a negociação de parâmetros de comunicação. Eles contêm informações importantes sobre a rede, como o SSID, o tipo de segurança, o canal de frequência, o *status* da conexão e outras informações de configuração [2].

O quadro 802.11 é dividido em nove campos, como mostrado na Figura 2.3. O campo *Controle de quadro* é responsável por indicar o tipo de quadro, versão do protocolo, modo de infraestrutura, entre outras configurações.

O campo *Duração* informa por quanto tempo o quadro e sua confirmação ocuparão o canal, medido em microssegundos. Este campo está presente em todos os tipos de quadros, incluindo os de controle [2].

Os três primeiros campos de endereços sempre são utilizados, independente do

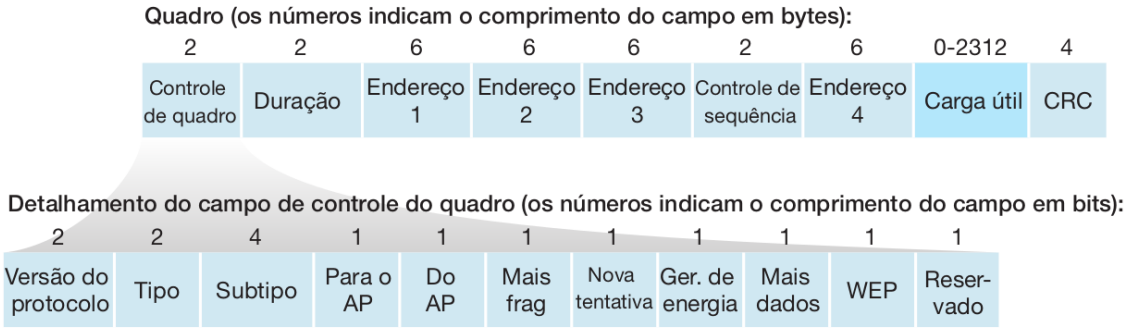


Figura 2.3: Formato do quadro de dados 802.11 (Fonte: Kurose e Ross, 2021).

tipo de quadro, já o *Endereço 4*, apenas é usado quando os dispositivos encaminham quadros uns aos outros em modo *ad hoc*.

O campo *Endereço 1* é preenchido com o MAC do receptor, o campo *Endereço 2* é destinado ao MAC do transmissor e o campo *Endereço 3* contém o MAC do dispositivo responsável pela interface entre o AP e a Internet ou outras sub-redes [4]. Para exemplificar a atribuição dos campos de endereço, considerando a estrutura mostrada na Figura 2.4, quando o roteador R1 recebe um pacote de dados para o *Internet Protocol* (IP) de H1, ele encapsula este pacote em um quadro 802.3 (*Ethernet*) com o MAC de H1 como destino. Ao AP receber este quadro, transforma-o em um quadro 802.11, preenchendo o *Endereço 1* com o MAC de H1, o *Endereço 2* com seu próprio MAC e o *Endereço 3* com o MAC de R1. Ao H1 receber o quadro, enviará um quadro de reconhecimento (*Acknowledgement*) preenchendo o *Endereço 1* com o MAC do AP, o *Endereço 2* com o seu próprio MAC e o *Endereço 3* com o MAC de R1.

O campo *Controle de sequência* é utilizado para indicar se o quadro está sendo transmitido pela primeira vez ou se é uma retransmissão, além da quantidade de retransmissões. O mesmo quadro será retransmitido até que o receptor envie o quadro de *Acknowledgement* ou até que o período de reserva do canal, indicado no campo *Duração*, expire.

O campo *Carga útil* consiste no pacote de dados a ser enviado. Está presente apenas nos quadros de dados e possui tamanho variável, podendo atingir até 2312

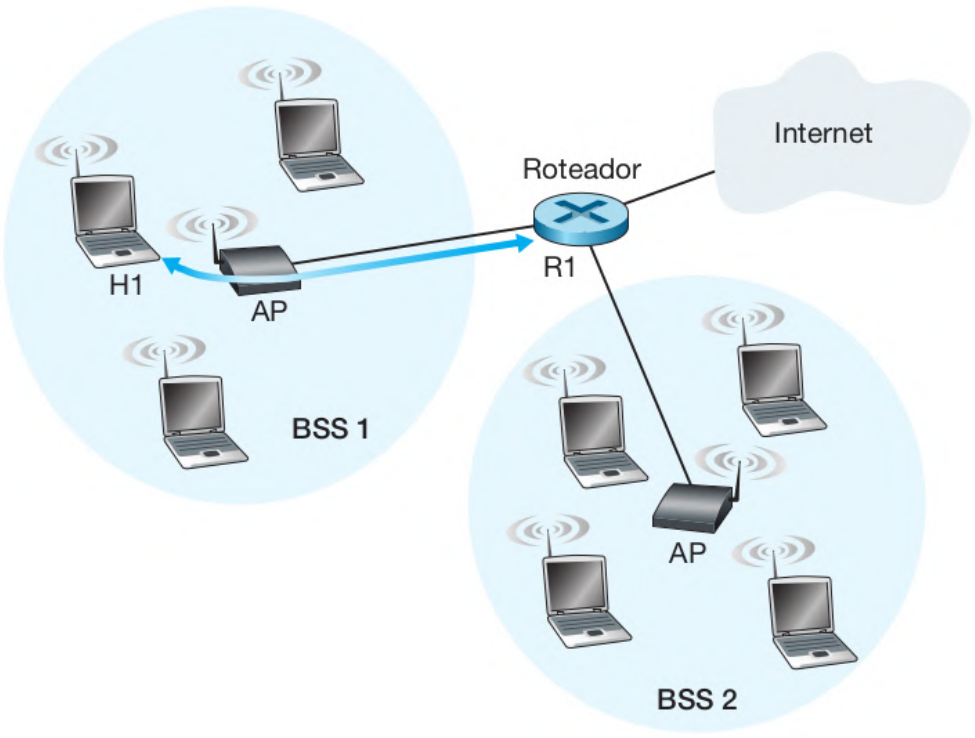


Figura 2.4: Utilização dos campos de endereço em quadros 802.11 (Fonte: Kurose e Ross, 2021).

bytes.

Por fim, o campo *Cyclic Redundancy Check (CRC)* é utilizado para detecção de erros.

2.1.4 RSSI e o Modelo de perda de caminho

O RSSI é o valor que representa a intensidade do sinal de rádio recebido por um dispositivo de comunicação sem fio, como um adaptador Wi-Fi. O RSSI é medido em dBm (decibéis relativos à potência de um milésimo de watt) e varia de acordo com a distância entre o dispositivo e o AP, bem como a presença de obstáculos ou interferência no ambiente. Quanto mais alto o valor do RSSI, maior será a intensidade do sinal recebido e vice-versa [11].

Para se estabelecer uma relação entre a potência do sinal recebido e a distância do receptor ao transmissor, é necessário modelar o canal de transmissão conforme os fenômenos por trás da propagação de ondas eletromagnéticas, em especial, os fenômenos de reflexão, difração e dispersão. Modelar estes canais de radiofrequência

é uma tarefa historicamente difícil, devido a sua natureza aleatória, e por isso geralmente é feita de maneira estatística, com base em medições [11].

Um dos modelos usados extensivamente é o Modelo de perda de caminho log-distância, que indica que a potência média do sinal recebido diminui logaritmicamente com a distância, tanto para ambientes internos ou externos. Considerando a potência recebida P_r, medida em dBm, a potência transmitida P_t, medida em dBm e a perda de caminho P_L, medida em dB, a relação entre elas é mostrada na Equação 2.1

$$P_r(d)[dBm] = P_t[dBm] - P_L(d)[dB]. \tag{2.1}$$

Segundo (Rappaport, 2008), a perda de caminho no modelo log-distância é calculada de acordo com a Equação 2.2

$$P_L(d)[dB] = P_L(d_0) + 10n \log\left(\frac{d}{d_0}\right) + X_\sigma, \tag{2.2}$$

em que d é a distância entre o receptor e o transmissor; d_0 é uma distância de referência já conhecida; n é o expoente de perda de caminho, que indica a velocidade com a qual essa perda aumenta, dependendo assim do ambiente de propagação específico; X_σ é uma variável aleatória com distribuição gaussiana de média zero com desvio padrão σ. Este parâmetro serve para descrever os efeitos aleatórios de sombreamento, que podem ocorrer em locais de mesma distância ao transmissor, porém com diferentes níveis de ruído no caminho de propagação.

A Tabela 2.2 mostra valores típicos para o expoente de perda de caminho de acordo com o ambiente. Na prática os valores de n e σ são calculados a partir dos dados medidos, usando regressão linear, de modo que a diferença entre as perdas de caminho medida e estimada são minimizadas para a média do erro quadrático em relação a uma grande faixa de distâncias medidas.

Ambiente	n
Espaço livre	2
Rádio-celular em área urbana	2,7 a 3,5
Rádio-celular urbano sombreado	3 a 5
Na linha de visão do prédio	1,6 a 1,8
Obstruído no prédio	4 a 6
Obstruído em fábricas	2 a 3

Tabela 2.2: Expoentes de perda de caminho para diferentes ambientes (Fonte: Rappaport, 2008).

2.2 Sistemas Embarcados

Um sistema embarcado é um sistema computacional composto por *hardware* e *software* projetado para executar uma tarefa específica em um sistema maior. Esses sistemas são integrados em outros produtos ou equipamentos com o objetivo de controlar ou monitorar uma determinada função ou processo. Eles são geralmente projetados para serem simples e de baixo custo, sendo compostos por um conjunto limitado de componentes, como microcontroladores, sensores e atuadores [15].

2.2.1 Microcontroladores

Os microcontroladores são componentes fundamentais nos sistemas embarcados, pois atuam como a "inteligência" por trás desses sistemas, permitindo o controle e a coordenação de suas funções. Um microcontrolador é um circuito integrado de único chip que contém um processador, memória e periféricos de Entrada/Saída (I/O) [16].

Existem várias famílias de microcontroladores disponíveis no mercado, cada uma com suas próprias características e aplicações. Algumas das famílias mais populares incluem: *Microchip PIC*, *Atmel AVR*, *ARM Cortex*, além da *Espressif ESP*, utilizada neste trabalho.

ESP32

O ESP32, mostrado na Figura 2.5 é um microcontrolador desenvolvido pala *Espressif Systems*, conhecido por sua conectividade sem fio integrada (Wi-Fi e *Blu-*

etooth), alto desempenho e baixo consumo de energia. Outro ponto importante deste microcontrolador é a possibilidade de utilizar *frameworks* de desenvolvimento como o do Arduino, *MicroPython* e o ESP-IDF, que fornecem ampla variedade de bibliotecas e facilitam a implementação de soluções [6]. Estas características são fundamentais em projetos de IOT, o que faz do ESP32 um dos mais populares nesse segmento.

Figura 2.5: Placa de desenvolvimento baseada no ESP32 (Fonte: [5]).

A Figura 2.6 mostra o diagrama em blocos do ESP32, em que é possível observar os componentes presentes no *chip*, incluindo a unidade de processamento; módulos de memória, Wi-Fi e *Bluetooth*; periféricos de I/O e interfaces de comunicação; aceleradores de *hardware* para criptografia; além de geradores de *clock*.

2.2.2 RTOS

RTOS são sistemas que respondem a estímulos externos em um intervalo de tempo especificado. Eles são caracterizados por terem o tempo como parâmetro-chave, com a precisão temporal sendo fundamental para garantir o correto funcionamento do sistema.

Esses sistemas são comumente encontrados em áreas como controle de processos industriais, aviônica, militar e outras aplicações semelhantes, em que é necessário

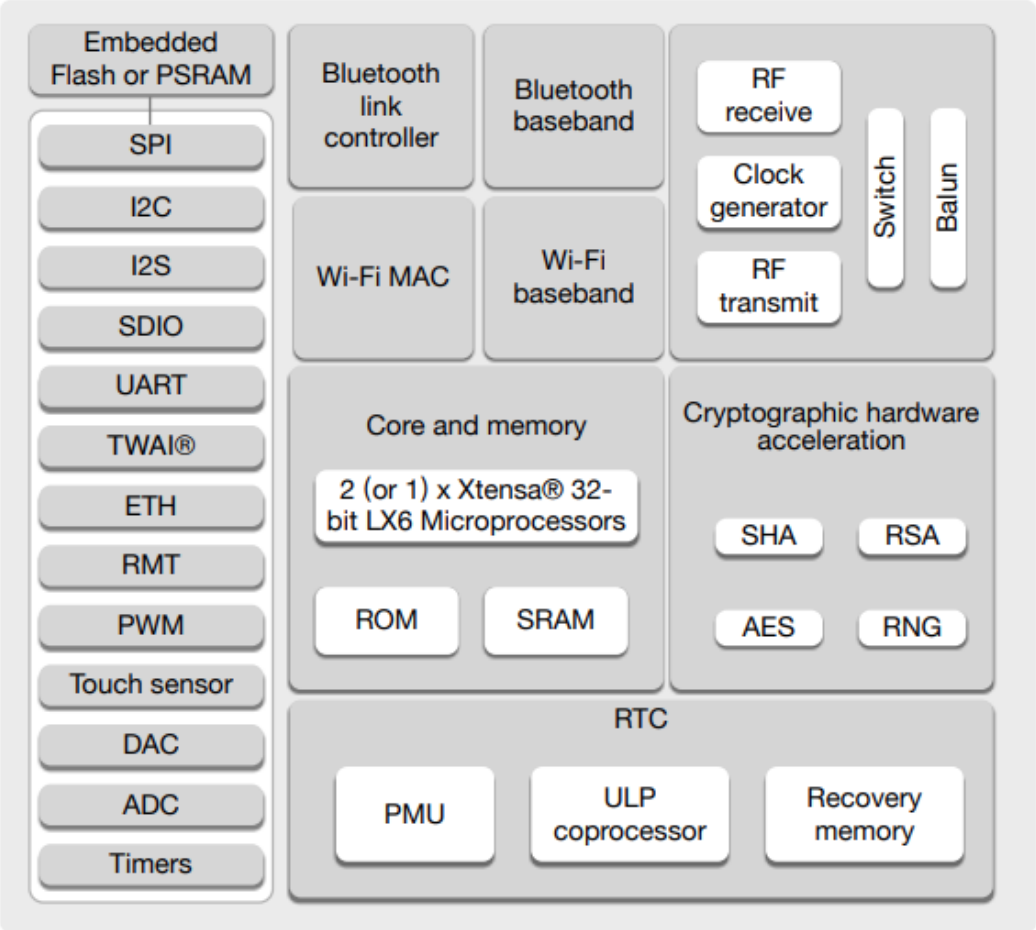


Figura 2.6: Diagrama em blocos do ESP32 (Fonte: [6]).

coletar e processar dados em tempo real para controlar o funcionamento de máquinas ou processos.

Os sistemas de tempo real são classificados em duas categorias: sistemas de tempo real críticos e sistemas de tempo real não críticos. Em sistemas de tempo real críticos, a precisão temporal é crucial e a perda de um prazo pode causar danos permanentes. Já em sistemas de tempo real não críticos, a perda ocasional de um prazo é aceitável e não causa danos permanentes [17].

FreeRTOS

Um exemplo de sistema operacional de tempo real voltado para sistemas embarcados é o *FreeRTOS*, distribuído sob a licença MIT de código aberto, que oferece

suporte para diversos processadores e arquiteturas.

O *FreeRTOS* é um *kernel* de baixo nível, que oferece um conjunto básico de serviços de gerenciamento de tarefas, escalonamento de processos, gerenciamento de memória e interrupções. Ele também inclui uma pilha TCP/IP, *drivers* de dispositivos, suporte a protocolos de comunicação e outras funcionalidades.

Uma das principais vantagens do *FreeRTOS* é a sua portabilidade, permitindo que o sistema operacional seja adaptado para diferentes plataformas e dispositivos. Além disso, ele é altamente configurável, permitindo que os desenvolvedores escolham os recursos que desejam utilizar e otimizem o sistema para as suas necessidades. Ele é considerado uma opção popular devido à sua facilidade de uso, desempenho e baixo custo, além de contar com uma comunidade ativa de desenvolvedores e usuários [18].

2.3 Desenvolvimento Web

O campo de Desenvolvimento Web é amplo e em constante evolução, abrangendo várias tecnologias e técnicas usadas para criar e manter *sites* e aplicações Web, podendo ser dividido em duas áreas principais: *frontend* e *backend*, em que cada uma contém suas próprias responsabilidades e conjunto de ferramentas [19].

2.3.1 Desenvolvimento *frontend*

O *frontend*, também conhecido como cliente, refere-se à parte visual e interativa de um *site* ou aplicação Web. É a interface com a qual os usuários interagem e experimentam os elementos visuais e de *design*. O desenvolvimento *frontend* é geralmente realizado utilizando-se as três principais tecnologias Web: HTML, CSS e *JavaScript* [19].

Hypertext Markup Language (HTML) é a linguagem padrão utilizada para estruturar o conteúdo de uma página Web. *Cascading Style Sheets* (CSS) é uma linguagem de estilo usada para descrever a aparência e formatação de um docu-

mento escrito em HTML. *JavaScript* é uma linguagem de programação que permite adicionar interatividade aos *sites*, permitindo a manipulação do HTML e CSS para criar animações, responder a eventos (como cliques do *mouse* ou pressionamento de teclas) e modificar o conteúdo da página em tempo real [19].

Frameworks JavaScript são bibliotecas ou conjuntos de ferramentas que ajudam os desenvolvedores a construir aplicações Web de maneira mais rápida e eficiente, fornecendo estruturas, componentes reutilizáveis e funcionalidades pré-definidas. Esses *frameworks* simplificam a complexidade inerente ao desenvolvimento Web e oferecem melhores práticas para criar aplicações escaláveis e de alto desempenho. Dentre alguns exemplos de *frameworks* para desenvolvimento *frontend* estão: *React*, *Vue.js*, *Ember.js*, além do *framework* utilizado neste trabalho, o *Angular* [20].

Angular

O *Angular* é um *framework JavaScript* de código aberto desenvolvido e mantido pelo *Google*, amplamente utilizado para criar aplicações Web de página única, ou *Single Page Application*s (SPAs). Ele destaca-se por sua abordagem modular e orientada a componentes, facilitando a organização, manutenção e escalabilidade do código. Uma característica notável do *Angular* é a utilização do *TypeScript* como linguagem de programação principal [21].

O *TypeScript*, desenvolvido e mantido pela *Microsoft*, é um superconjunto tipado de *JavaScript* que adiciona tipos estáticos e outros recursos, permitindo aos desenvolvedores escrever um código mais robusto e fácil de manter. Com o suporte do *TypeScript*, o *Angular* oferece uma experiência de desenvolvimento aprimorada, com recursos avançados de autocompletar, verificação de tipos e detecção de erros em tempo de compilação [22].

2.3.2 Desenvolvimento *backend*

O desenvolvimento *backend* é a parte que lida com a lógica de negócios, gerenciamento de dados e comunicação entre o servidor e o cliente (*frontend*). O

desenvolvimento *backend* concentra-se em garantir que os dados sejam armazenados, recuperados e processados de forma eficiente, bem como em criar *Application Programming Interface*s (APIs) e serviços para serem consumidos pelo *frontend*.

O desenvolvimento *backend* envolve o uso de diversas linguagens de programação, como *Python*, *Java*, *Ruby*, *C#* e *PHP*. Além de trabalharem com sistemas de gerenciamento de banco de dados, como *MySQL*, *PostgreSQL*, *MongoDB* e *Oracle* [23].

Assim como acontece no *frontend*, no desenvolvimento *backend* também são utilizados *frameworks* para facilitar a construção de aplicações Web e APIs. Esses *frameworks* fornecem abstrações, ferramentas e bibliotecas para lidar com tarefas comuns, como roteamento, autenticação e acesso a bancos de dados. Para cada uma das linguagens de programação citadas acima existem diversos *frameworks*, como por exemplo: *Django (Python)*, *Ruby on Rails (Ruby)*, *ASP .NET (C#)*, *Laravel (PHP)*, além do utilizado neste trabalho *Spring Boot (Java)* [24].

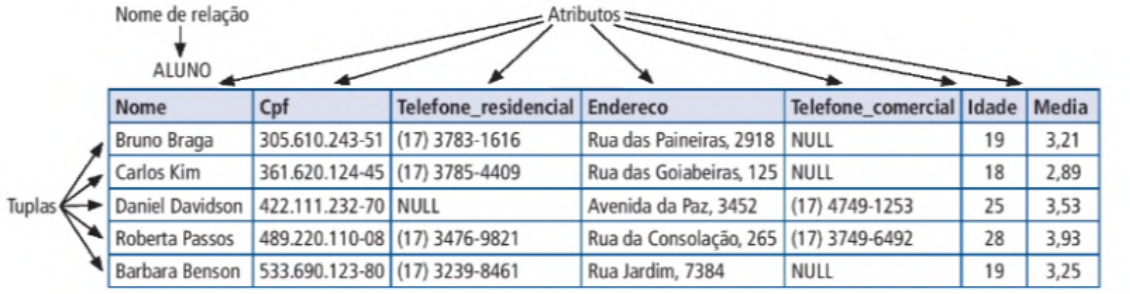
Spring Boot

O *Spring Boot* é um subprojeto do ecossistema *Spring*, mantido e desenvolvido pela *VMware*, projetado para otimizar e acelerar o processo de desenvolvimento de aplicações baseadas no *Spring framework*. Ele proporciona uma abordagem opinativa de configuração, minimizando a necessidade de configurações manuais e simplificando o gerenciamento de dependências compatíveis. Além disso, o *Spring Boot* oferece servidores Web ou de aplicação embarcados e mecanismos integrados de monitoramento e gerenciamento [25].

2.3.3 Banco de Dados Relacional

Banco de dados relacional é um Sistema de Gerenciamento de Banco de Dados (SGBD) baseado no modelo relacional, concebido por (Codd, 1970). Os bancos de dados relacionais organizam os dados em tabelas, também conhecidas como relações, compostas por linhas e colunas. Cada linha representa uma entidade ou objeto, enquanto cada coluna representa um atributo dessa entidade (Elmasri e Navathe,

2019). Na Figura 2.7 é mostrado um exemplo de modelo relacional de banco de dados para a relação *ALUNO*, em que cada tupla indica uma entidade da relação, ou seja, um aluno nesse caso.



Nome	Cpf	Telefone_residencial	Endereco	Telefone_comercial	Idade	Media
Bruno Braga	305.610.243-51	(17) 3783-1616	Rua das Paineiras, 2918	NULL	19	3,21
Carlos Kim	361.620.124-45	(17) 3785-4409	Rua das Goiabeiras, 125	NULL	18	2,89
Daniel Davidson	422.111.232-70	NULL	Avenida da Paz, 3452	(17) 4749-1253	25	3,53
Roberta Passos	489.220.110-08	(17) 3476-9821	Rua da Consolação, 265	(17) 3749-6492	28	3,93
Barbara Benson	533.690.123-80	(17) 3239-8461	Rua Jardim, 7384	NULL	19	3,25

Figura 2.7: Exemplo de Banco de dados relacional (Fonte: Elmasri e Navathe, 2019).

Os bancos de dados relacionais oferecem diversas vantagens como: estruturação dos dados, consultas e manipulação de dados por meio da linguagem SQL, integridade referencial, normalização de dados, além das operações de leitura escrita obedecerem às propriedades Atômicas, Consistentes, Isoladas e Duráveis (ACID) (Elmasri e Navathe, 2019).

SQL

SQL é uma linguagem de programação de domínio específico, amplamente utilizada para gerenciar, consultar e manipular dados em SGBDs. Desenvolvida na década de 1970 por Chamberlin D. e Boyce R. nos laboratórios de pesquisa da IBM, a SQL é uma linguagem declarativa, o que significa que os usuários especificam o resultado desejado de uma consulta, em vez de detalhar o processo passo a passo para chegar ao resultado. Essa abordagem permite que os sistemas de bancos de dados otimizem a execução das consultas, melhorando o desempenho e a eficiência na recuperação de informações.

Uma consulta (*Query*) SQL é mostrada no Código 1. Neste exemplo observam-se as cláusulas *SELECT*, que especifica as colunas a serem recuperadas, neste caso Nome e Cpf; *FROM*, que indica a tabela de onde os dados devem ser recuperados, neste exemplo ALUNO; *WHERE*, usada para filtrar os resultados com base em

uma condição específica, neste caso que a Media seja superior a $3,5$; *ORDER BY* utilizada para ordenar os resultados da consulta com base em uma ou mais colunas, neste exemplo a coluna Nome; Além da palavra chave *ASC* que indica a ordem alfabética crescente.

```
SELECT Nome, Cpf
FROM ALUNO
WHERE Media > 3.5
ORDER BY Nome ASC;
```

Código 1: *Query* SQL para listar alunos com base na média (Fonte: O autor, adaptado de Elmasri e Navathe, 2019)

2.3.4 O Protocolo HTTP e APIs

O *Hypertext Transfer Protocol* (HTTP) é um protocolo da camada de aplicação usado na Internet para transmitir informações entre clientes e servidores. Ele atua como base para a troca de dados e recursos na *World Wide Web* (WWW), seguindo um modelo de requisição-resposta, em que os clientes (geralmente navegadores Web) enviam requisições a servidores e recebem a resposta respectiva. Este modelo é ilustrado na Figura 2.8 [27].

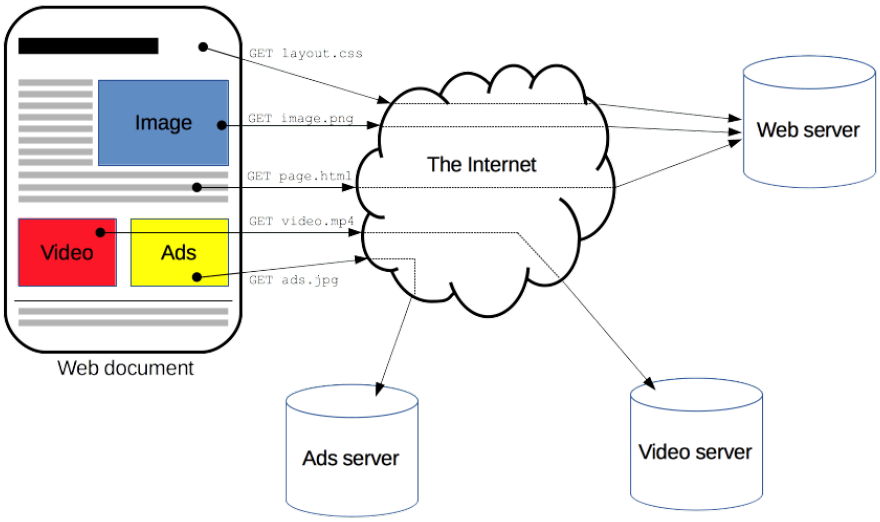


Figura 2.8: Modelo requisição-resposta do HTTP(Fonte: [8]).

Uma requisição HTTP é composta de três partes principais: a linha de requisição,

os cabeçalhos e o corpo da mensagem (opcional). A linha de requisição inclui o método HTTP, o *Uniform Resource Locator* (URL) do recurso desejado e a versão do protocolo. Os cabeçalhos são campos de metadados que fornecem informações adicionais sobre a requisição, como tipo de conteúdo, autenticação, entre outros. O corpo da mensagem contém os dados que estão sendo enviados ao servidor, geralmente usado em métodos *POST* e *PUT*. O Código 2 é um exemplo de requisição HTTP.

```
GET /index.html HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

Código 2: Exemplo de requisição HTTP (Fonte: [8])

Uma resposta HTTP também é composta de três partes principais: a linha de *status*, os cabeçalhos e o corpo da mensagem. A linha de status contém a versão do protocolo, o código de *status* e uma frase explicativa. Os cabeçalhos são campos de metadados que fornecem informações adicionais sobre a resposta, como tipo ou comprimento de conteúdo, entre outros. O corpo da mensagem contém os dados solicitados pelo cliente, como um arquivo HTML, dados em formato *JavaScript Object Notation* (JSON), ou outros formatos [27]. O Código 3 mostra a linha de *status* e os metadados da resposta à requisição feita no Código 2.

```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
Etag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
```

Código 3: Exemplo de resposta HTTP (Fonte: [8])

Os códigos de *status* indicam o resultado de uma requisição HTTP. Eles são divididos em cinco classes, com base no primeiro dígito do código [27].

- **1xx** (Informativo): A requisição foi recebida e o processamento está sendo

feito por parte do servidor.

- **2xx** (Bem-sucedido): A requisição foi processada com sucesso e aceita.
- **3xx** (Redirecionamento): A requisição precisa de mais ações para ser concluída.
- **4xx** (Erro do cliente): A requisição contém sintaxe incorreta ou não pode ser atendida pelo servidor.
- **5xx** (Erro do servidor): O servidor falhou em atender a uma requisição válida.

Os métodos HTTP, também conhecidos como verbos, são usados para indicar a ação que um cliente deseja executar num recurso específico do servidor Web. Os métodos mais comuns são: *GET*, *POST*, *PUT* e *DELETE*, cada um com seu propósito específico.

O método *GET* é usado para solicitar informações de um recurso específico. Este método é considerado seguro e idempotente, ou seja, várias requisições idênticas devem ter o mesmo efeito que uma única requisição. Não deve ser utilizado para enviar informações confidenciais, pois os dados são enviados como parte do URL.

O método *POST* é usado para enviar dados a um recurso específico para processamento. Os dados são enviados no corpo da requisição, o que permite o envio de informações confidenciais e grandes volumes de dados. Este método não é idempotente, o que significa que várias requisições idênticas podem ter efeitos diferentes, como a criação de múltiplos recursos.

O método *PUT* é usado para atualizar um recurso existente com novos dados. Assim como o *POST*, os dados são enviados no corpo da requisição, entretanto ele é considerado idempotente. Em geral, o método *PUT* requer que todas as informações sejam fornecidas pela requisição, mesmo que apenas parte dela seja atualizada.

O método *DELETE* é utilizado para excluir um recurso específico. Ele é considerado idempotente e não requer um corpo na requisição, pois todas as informações necessárias são fornecidas no URL [27].

APIs REST

Representational State Transfer (REST) é um estilo arquitetural, aplicado em APIs, que promove a criação de serviços Web escaláveis e de fácil manutenção. Ele baseia-se nos princípios do protocolo HTTP e utiliza seus métodos para realizar operações em recursos de servidores Web [27].

Segundo [27, Fielding], a arquitetura REST é baseada num conjunto de princípios fundamentais:

- ***Stateless***: Cada requisição feita à API deve conter todas as informações necessárias para o servidor processá-la. O servidor não deve armazenar informações sobre o estado do cliente entre as requisições.
- ***Cacheable***: As respostas da API podem ser armazenadas em cache no cliente, melhorando o desempenho da aplicação. O servidor deve indicar se a resposta é armazenável ou não.
- ***Client-Server***: A arquitetura REST segue o modelo cliente-servidor, em que cada cliente é responsável pela interface do usuário, e o servidor é responsável pelo processamento e gerenciamento dos recursos.
- ***Layered System***: O sistema REST é organizado em camadas, o que permite abstrair a complexidade de cada camada e promover a separação de responsabilidades.
- ***Code on Demand***: O servidor pode fornecer código executável para ser usado pelo cliente, para estender sua funcionalidade.
- ***Uniform Interface***: A API REST deve seguir uma interface uniforme para facilitar a comunicação e o entendimento entre clientes e servidores.

Arquitetura MVC

O *Model-View-Controller* (MVC) é um padrão de arquitetura de *software* utilizado para desenvolver interfaces para o usuário de forma organizada e dividida em

três componentes principais: Modelo (*Model*), Visão (*View*) e Controlador (*Controller*). Este padrão é amplamente adotado no desenvolvimento de aplicações Web devido à sua eficácia em reduzir o acoplamento de código, facilitando sua manutenção e reutilização [28].

- ***Model***: O *Model* representa a parte do sistema que gerencia os dados, a lógica e as regras do negócio, é responsável por acessar o banco de dados e definir a estrutura dos dados que a aplicação utiliza. O *Model* é completamente independente da interface do usuário e interage com o *Controller* para enviar e receber dados.

- ***View***: A *View* é responsável pela apresentação dos dados recebidos pelo *Model* ao usuário.

- ***Controller***: O *Controller* atua como intermediário entre o *Model* e a *View*, recebendo as entradas do usuário e convertendo em instruções para os outros dois componentes.

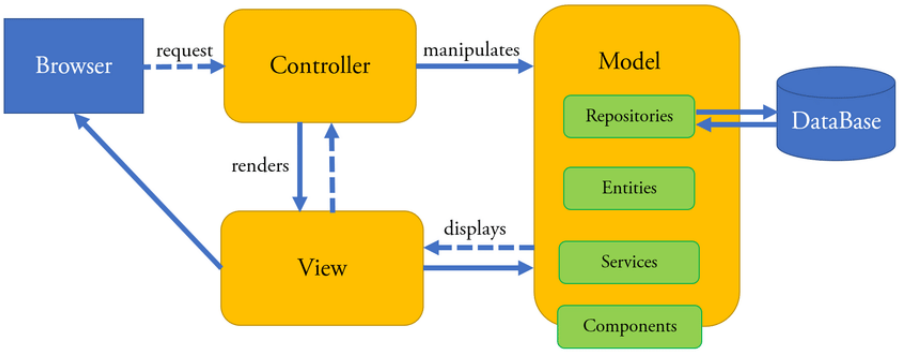


Figura 2.9: Ilustração da arquitetura MVC (Fonte: [9]).

2.3.5 O HTTPS

O *Hypertext Transfer Protocol Secure* (HTTPS) é uma versão aprimorada e mais segura do protocolo HTTP original, fornecendo criptografia e autenticação para proteger a privacidade e integridade dos dados trocados entre as partes envolvidas.

A segurança no HTTPS é garantida por meio de um protocolo adicional, chamado *Secure Sockets Layer/Transport Layer Security* (SSL/TLS) e segundo (Rescorla, 2000) o processo para estabelecer uma conexão HTTPS segue as seguintes etapas:

- ***Handshake***: O cliente envia a solicitação ao servidor para iniciar uma conexão segura. Essa solicitação inclui informações sobre as versões do SSL/TLS e os algoritmos de criptografia que o cliente suporta.
- **Certificado do Servidor**: O servidor responde com seu certificado digital, que inclui a chave pública do servidor e informalções sobre a entidade emissora do certificado *Certificate Authority* (CA).
- **Verificação do Certificado**: O cliente verifica a validade do certificado do servidor, garantindo que ele foi emitido por uma CA confiável e que o certificado não expirou ou foi revogado.
- **Geração da Chave de Sessão**: O cliente cria uma chave de sessão criptografada usando a chave pública do servidor e a envia de volta.
- **Comunicação Segura**: Com a chave de sessão em mãos, o cliente e o servidor podem trocar informações de forma segura. Todos os dados transmitidos entre eles são cifrados e só podem ser decifrados com a chave de sessão.

2.4 Computação em Nuvem

A computação em nuvem é um modelo que permite aos usuários acessar recursos de TI, como servidores, armazenamento, bancos de dados, redes e *software*, por meio da Internet, em vez de hospedá-los localmente em suas próprias infraestruturas.

Essa abordagem oferece uma série de benefícios, incluindo escalabilidade, flexibilidade, eficiência de custos e acesso a serviços e tecnologias inovadoras. A computação em nuvem facilita a implantação rápida de aplicações e serviços, permitindo que as empresas se adaptem rapidamente às mudanças nas demandas do mercado e aproveitem as oportunidades emergentes [30].

Existem três modelos de serviço principais na computação em nuvem: *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) e *Software as a Service* (SaaS).

No modelo IaaS os provedores de nuvem oferecem recursos de infraestrutura, como servidores virtuais, armazenamento e redes, que os usuários podem gerenciar e configurar de acordo com as suas necessidades. Exemplos de provedores de IaaS incluem AWS, *Microsoft Azure* e *Google Cloud Platform* (GCP).

No modelo PaaS os provedores de nuvem oferecem um ambiente de desenvolvimento e implantação de aplicações que inclui recursos de infraestrutura, bem como serviços e ferramentas para o desenvolvimento, teste e gerenciamento de aplicações. Exemplos de provedores de PaaS incluem *Heroku*, *IBM Cloud*, *Google App Engine* e AWS *Elastic Beanstalk*.

No modelo SaaS os provedores de nuvem oferecem aplicações prontas para uso, que podem ser acessadas por meio de um navegador Web ou cliente de *software*. Os usuários não precisam se preocupar com a infraestrutura ou manutenção do *software*, pois tudo é gerenciado pelo provedor de nuvem. Exemplos de aplicações SaaS incluem *Salesforce*, *Google Workspace* e *Microsoft Office 365* [31].

Além desses modelos de serviço, a computação em nuvem pode ser classificada em diferentes tipos de implantação [31]:

- Nuvem pública: Os recursos são fornecidos e gerenciados por provedores de nuvem de terceiros e compartilhados entre vários usuários. As nuvens públicas são geralmente mais econômicas do que outras opções de implantação.

- Nuvem privada: Os recursos são dedicados a uma única organização e podem ser hospedados no local ou por um provedor de nuvem externo. As nuvens privadas oferecem maior controle e segurança, mas podem ser mais caras e complexas de gerenciar.

- Nuvem híbrida: Combina recursos de nuvens públicas e privadas, permitindo que as organizações aproveitem os benefícios de ambos os modelos. As nuvens

híbridas oferecem maior flexibilidade e podem ser utilizadas para equilibrar custos, segurança e desempenho.

No campo da IOT, a computação em nuvem desempenha um papel fundamental no gerenciamento e processamento de dados gerados por dispositivos conectados. Ela também possibilita o desenvolvimento de soluções IOT mais complexas e integradas.

Já no desenvolvimento de *software*, a computação em nuvem permite a implementação de *pipelines* de *Continuous Integration / Continuous Delivery* (CI/CD), melhorando a eficiência do desenvolvimento e reduzindo o tempo de lançamento de novos recursos e atualizações, além de facilitar o uso de microsserviços e arquiteturas orientadas a eventos.

Capítulo 3

Metodologia

Neste capítulo são apresentados os procedimentos adotados para alcançar os objetivos do trabalho citados no Capítulo 1. A metodologia divide-se em três etapas principais: validar o modelo de perda de caminho proposto em (Rappaport, 2008), definição do limiar de RSSI para considerar que o dispositivo está próximo e construção do protótipo de sistema para controle de acesso. Os dois primeiros tópicos são abordados neste capítulo, já o terceiro é abordado no Capítulo 4.

3.1 Validação do modelo de perda de caminho

A fim de verificar a validade do modelo proposto em (Rappaport, 2008), o arranjo utilizado neste trabalho é composto por um transmissor e um receptor. São transmitidos pacotes do protocolo IEEE 802.11 na faixa de frequência de 2.4 GHz, realizada a leitura dos valores de RSSI para diferentes distâncias e a partir dos dados coletados, feita a estimativa do expoente de perda de caminho, apresentado na Equação 2.2. A abordagem definida para execução do experimento segue a realizada em (Miranda *et al.*, 2013).

3.1.1 Configuração do experimento

Para a execução do experimento, os dispositivos utilizados como transmissor e receptor, são o *Apple Iphone 16 Pro Max* e *WIFI Kit 32*, respectivamente. Os

dispositivos foram posicionados a 70 cm de altura em relação ao piso, num mesmo ambiente interno e sem obstáculos entre eles. Foram escolhidos para a transmissão os canais 1, 6 e 11, com frequências centrais $2,412\,GHz$, $2,437\,GHz$ e $2,462\,GHz$, respectivamente, visto que são os canais sem sobreposição de espectro. Os valores de RSSI são medidos para dez diferentes distâncias entre 50 cm até 5 m, variando com passo de 50 cm. Para cada distância são coletados vinte valores e considerada a média deles.

3.1.2 Análise do ajuste ao modelo

A partir da Equação 2.2, é possível estimar o parâmetro n (expoente de perda de caminho) por meio de regressão linear simples entre $P_L(d_i)$ e $\log_{10}(d_i)$, dessa forma, n é obtido a partir da divisão do coeficiente angular por 10.

A qualidade do ajuste é verificada por meio de métricas como o *Coeficiente de Determinação* (R^2) e *Raiz do Erro Médio Quadrático* (RMSE), que indicam o quão bem o modelo explica a variância dos dados. Outra maneira de validar o modelo é comparar o valor de n obtido experimentalmente com o valor correspondente ao cenário *Na linha de visão do prédio* na Tabela 2.2.

3.2 Definição do limiar de RSSI

O limiar de RSSI é o valor mínimo em que o dispositivo transmissor é considerado próximo ao receptor, estando apto a ter o acesso liberado pelo sistema. Este valor varia de acordo com as características específicas de cada transmissor, portanto deve ser considerada uma amostra com alguns dispositivos diferentes, calculando a partir dela uma estimativa confiável para o limiar que se estenda para novos dispositivos não presentes na amostra.

Neste trabalho foi considerada uma amostra com tamanho $N = 5$ e para cada dispositivo, consolidado o valor médio de vinte medidas dos valores de RSSI aferidos na distância limite, definida como 50 cm. Os dispositivos são: *Apple Iphone 11*,

Apple Iphone 16 Pro Max, *Motorola Moto E7*, *Samsung Galaxy J7* e *Samsung Galaxy S21*.

Para definir o limiar é realizado um teste com as seguintes hipóteses:

- **H_0**: O dispositivo pertence à mesma população de referência. $(\mu \leq \mu_0)$
- **H_1**: O dispositivo tem RSSI superior, logo está mais próximo. $(\mu > \mu_0)$

Assim, definindo um intervalo de confiança unilateral de 95 %, ou 5 % de significância, encontra-se o valor crítico utilizando a distribuição *t de Student*, com grau de liberdade 4, de modo que ao medir valores de RSSI acima do valor crítico, rejeita-se a hipótese nula (H_0) e considera que a medida não pertence a população de referência (medidas coletadas à distância de $0,5\,m$), concluindo que o dispositivo está mais próximo ao sistema, e portanto deve ter o acesso liberado.

A distribuição *t de Student* é utilizada devido à variância da população ser desconhecida, além do tamanho da amostra ser pequeno. A expressão da estatística t é dada pela Equação 3.1, em que R é o valor de RSSI a ser comparado, $\bar{R}$ é a média amostral dos valores de RSSI, s é a variância amostral e N é o tamanho da amostra

$$t = \frac{R - \bar{R}}{s/\sqrt{N}}. \tag{3.1}$$

O limiar é encontrado ao fazer a estatística t ser igual ao valor crítico definido a partir do intervalo de confiança, chamado t_{crit}, portanto ao rearranjar a Equação 3.1, obtém-se a expressão do limiar L, mostrada na Equação 3.2

$$L = \bar{R} + \frac{t_{crit}\, s}{\sqrt{N}}. \tag{3.2}$$

Capítulo 4

Desenvolvimento

Neste capítulo, são descritas as etapas do desenvolvimento do projeto, incluindo o sistema proposto, as ferramentas e tecnologias utilizadas, além das definições necessárias para atender os objetivos citados no capítulo 1.

O sistema proposto é formado por quatro módulos, conforme mostrado na Figura 4.1. Toda a lógica de detecção do sinal Wi-Fi e liberação de acesso está contemplada no módulo Microcontrolador, desenvolvido no ESP32. No módulo Página Web, hospedado no *Github Pages* e desenvolvido em *Angular*, está a interface de usuário, que permite cadastrar os endereços MAC e visualizar os registros de acesso por meio de uma página Web. Já os módulos API e Banco de Dados, são hospedados na AWS e desenvolvidos em *Java* e *PostgreSQL*, respectivamente. São responsáveis por realizar a comunicação entre a interface de usuário e a base de dados dos cadastros.

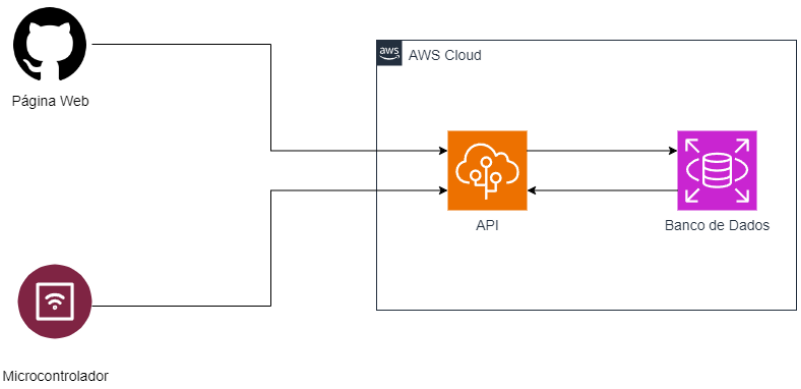


Figura 4.1: Esquema de módulos que compõem o sistema (Fonte: O autor).

Nas seções seguintes deste capítulo serão abordados os detalhes de implementação para cada módulo.

4.1 Microcontrolador

O microcontrolador é o núcleo central do sistema, nele são desenvolvidas as etapas necessárias para atingir os seguintes objetivos específicos citados no capítulo 1:

- Configurar o módulo Wi-Fi do microcontrolador para funcionar como detector de pacotes;
- Implementar a máquina de estados responsável pela lógica da liberação de acesso;
- Desenvolver comunicação entre microcontrolador e servidor *Web* para acessar a lista de usuários cadastrados;
- Implementar as telas de status do sistema no *display* OLED presente na placa de desenvolvimento;

Para este trabalho, foi escolhido o *WIFI Kit 32*, fabricado pela *DIY MORE*, exibido na Figura 4.2. Esta placa de desenvolvimento é baseada em ESP32, portanto como visto na seção 2.2, ela já possui módulo Wi-Fi integrado, além de incluir um *display* OLED, dispensando a necessidade de componentes adicionais. As especificações completas podem ser consultadas no manual fornecido pelo fabricante [33].

A programação do microcontrolador foi realizada por meio do *PlatformIO IDE*, extensão construída para o editor de texto *Microsoft Visual Studio Code*. Essa *Integrated Development Environment* (IDE) facilita o desenvolvimento em linguagem C/C++, permitindo que os detalhes específicos de *hardware* sejam abstraídos por um simples arquivo de configuração, além de possibilitar a utilização do *framework* ESP-IDF e automatizar o processo de *build* [34].

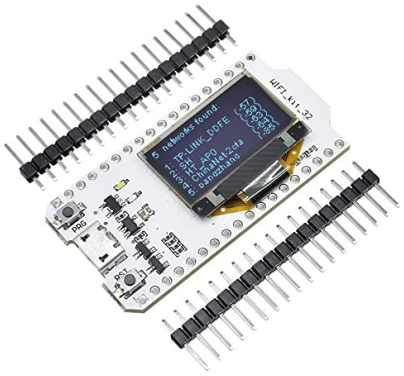


Figura 4.2: Placa de desenvolvimento *WIFI Kit 32 DIY MORE* (Fonte: [10]).

A estrutura geral do sistema é ilustrada pelo diagrama de fluxo exibido na Figura 4.3. Esta arquitetura consiste numa máquina com quatro estados, o estado inicial é responsável por realizar a conexão do microcontrolador com o ponto de acesso, já o estado seguinte faz a requisição para a API utilizando o protocolo HTTP, com a finalidade de receber e armazenar os dados referentes aos endereços MAC cadastrados no banco de dados. O terceiro estado é responsável por colocar o módulo de Wi-Fi no modo promíscuo, desempenhando o papel de analisador de pacotes e consequentemente liberando ou não o acesso pela quantidade de tempo determinada. Por fim, no último estado é enviada a requisição HTTP para gravar no banco de dados os registros de acesso.

Todo o código referente ao microcontrolador encontra-se no repositório *sniffer*, disponibilizado no *GitHub* [35] e a parte específica da máquina de estados está contida no arquivo *"main.c"*.

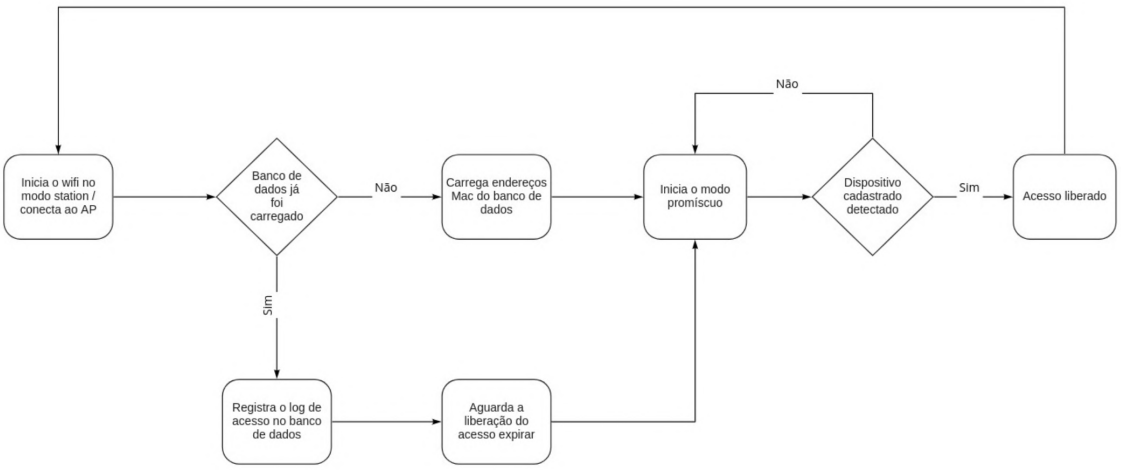


Figura 4.3: Diagrama de fluxo do sistema (Fonte: O autor).

4.1.1 Analisador de pacotes

Para desempenhar o papel de analisador de pacotes, é necessário configurar o módulo Wi-Fi para a operação no modo promíscuo, que permite capturar diferentes tipos de pacotes transmitidos no ambiente, independente de serem destinados ao próprio dispositivo. O *framework* ESP-IDF disponibiliza funções para utilizar o módulo Wi-Fi por meio da biblioteca *"esp_wifi.h"* [36].

Após a inicialização do módulo Wi-Fi por meio das funções *esp_wifi_init*, *esp_wifi_set_mode* e *esp_wifi_start*, para habilitar o modo promíscuo é preciso chamar a função *esp_wifi_set_promiscuous* com o argumento *true*, além de configurar o *callback* *wifi_sniffer_packet_handler* para realizar o processamento dos dados sempre que um pacote for capturado.

O formato dos pacotes é o exposto na Figura 2.3, mas o único campo utilizado é o Endereço 2, referente ao MAC do dispositivo transmissor. Além disso, é possível obter acesso ao valor de RSSI por meio de um *buffer* implementado pelo ESP-IDF.

Para a liberação de acesso, primeiro é verificado se o RSSI é superior ao *threshold*, em caso positivo, busca-se o conteúdo do campo Endereço 2 na lista com os endereços carregados do banco de dados, liberando o acesso, caso o endereço esteja presente na lista. O código referente a esta parte do sistema está contido no arquivo *"wifi_lib.c"*.

***Branch* de análise de resultados**

Além do sistema, foi desenvolvida uma *branch* separada apenas com a funcionalidade de analisador de pacotes, a fim de apoio para a análise de resultados realizada no capítulo 5. Na *branch analytics*, o módulo Wi-Fi é inicializado no modo promíscuo e filtra pelos pacotes de um dispositivo especificado, exibindo o valor de RSSI para cada pacote por meio do monitor serial do computador.

4.1.2 Protocolo HTTP

Para realizar a comunicação com a API por meio do protocolo HTTP, o ESP-IDF utiliza o *Lightweight TCP/IP* (lwIP), uma pilha de protocolos *Transmission Control Protocol* (TCP)/IP leve e eficiente, projetada para sistemas embarcados com recursos limitados lwIP [37].

Neste trabalho são implementadas duas tarefas, *http_get_task* e *http_post_task*, a primeira, responsável por realizar a requisição do tipo GET para a API que retorna todos os endereços MAC, interpretando os dados disponibilizados em formato JSON e armazenando os endereços MAC na memória. Já a outra tarefa é responsável por enviar a requisição do tipo POST para a API que insere os registros de acesso no banco de dados. Toda a implementação relativa às requisições HTTP está no arquivo *"http_lib.c"*.

4.1.3 *Display* OLED

O *WIFI Kit 32* é equipado com um *display* OLED de $0,96$ polegadas, controlado pelo *driver SSD1306*, que utiliza como interface de comunicação o protocolo *Inter-Integrated Circuit* (I2C), sendo necessário configurar os pinos *SDA* e *SCL* conforme a especificação do *driver* [38]. No arquivo *display_lib.c"* estão as configurações dos caracteres, assim como as funções de escrita e deleção de texto.

As funções implementadas servem como base para reproduzir as telas observadas na Figura 4.4. No instante inicial, em que o sistema está carregando a lista de endereços MAC do servidor, é apresentada a tela vista na Figura 4.4a. Em seguida,

(a) Tela de carregamento

(b) Tela do modo promíscuo

(c) Tela de acesso liberado

Figura 4.4: Telas implementadas no microcontrolador (Fonte: O autor).

o sistema entra no modo promíscuo para capturar os pacotes e a tela exibida neste momento é observada na Figura 4.4b. Por fim, após um dispositivo cadastrado liberar o acesso, a tela exibida é mostrada na Figura 4.4c, indicando que o acesso foi liberado, assim como o tempo restante e o nome cadastrado pelo usuário.

4.2 Servidor *Web*

Nesta seção são desenvolvidas as etapas necessárias para atingir o seguinte objetivo específico citado no capítulo 1: "Desenvolver servidor *Web* para armazenar as informações dos usuários cadastrados".

Como visto na seção 2.3, um servidor *Web* é composto principalmente de APIs e bancos de dados. Para este trabalho, foi escolhido construir a API utilizando o *framework Spring Boot*, da linguagem *Java*, já para o banco de dados, foi definido o SGBD *PostgreSQL*.

Todo o código referente ao servidor *Web* está no repositório *sniffer-api*, disponibilizado no *GitHub* [39].

4.2.1 API

Para desenvolver a API foi utilizada a ferramenta *spring initializr*, disponibilizada pelo *framework Spring Boot* conforme mostrado na Figura 4.5. Esta ferramenta cria uma estrutura inicial de arquivos, já incluindo as dependências necessárias, tanto para o desenvolvimento da API quanto para a integração com o banco de dados [40].

Foram desenvolvidos dois controladores, um responsável pelos endereços MAC e outro pelos *logs* de acesso. Os *endpoints* implementados são exibidos na Figura 4.6. Em ambos os controladores, os métodos *GET* são utilizados para retornar a lista de endereços MAC ou *logs*, e no caso em que também é passado algum *id*, é retornado apenas o item relativo à este *id*.

Já os métodos *POST* são utilizados para inserir tanto os endereços MAC quanto os *logs* no banco de dados. Por fim, os métodos *DELETE* servem para remover

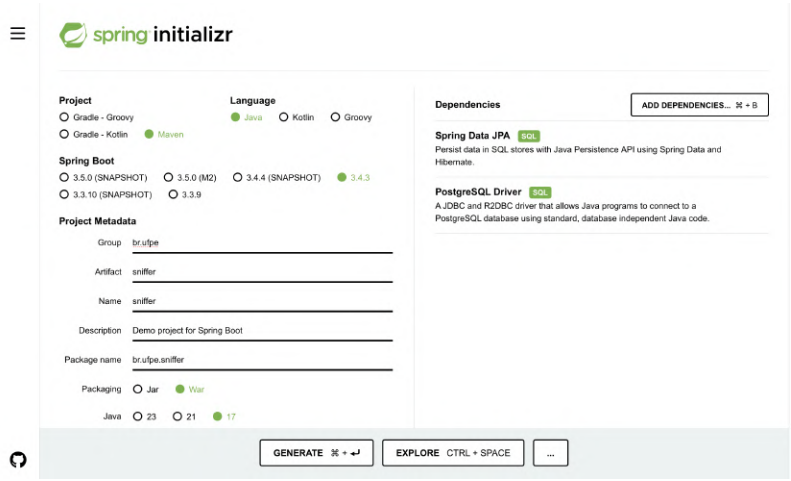


Figura 4.5: Tela de configuração do spring initializr (Fonte: O autor).

algum dos registros do banco de dados.

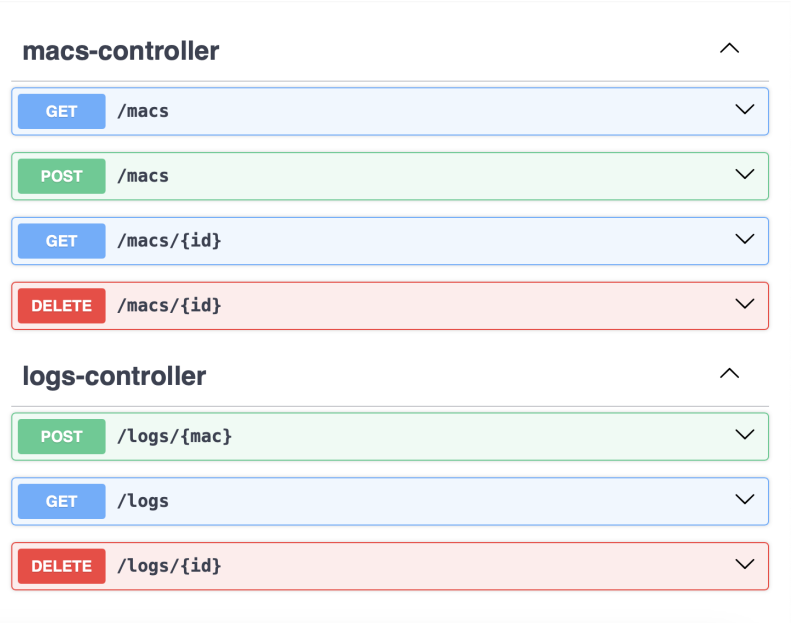


Figura 4.6: Lista com todos os *endpoints* da API (Fonte: O autor).

Uma vez desenvolvida a API, é necessário hospedá-la num serviço de nuvem, para que fique disponível à internet. O serviço escolhido foi o AWS, por meio do *Elastic Beanstalk*, uma solução PaaS que fornece todo o ambiente necessário para a implantação de uma API. Para isto, basta gerar o pacote *war*, por meio do comando *mvn clean package* e fazer o *upload* no *Elastic Beanstalk* como mostrado no Apêndice A.

4.2.2 Banco de dados

Para a criação do banco de dados foi utilizado o serviço AWS RDS, que fornece uma solução para gerenciamento do banco de dados de forma simplificada e exige poucas configurações, como pode ser visto no Apêndice B.

O banco de dados criado é composto por duas tabelas, *macs* e *logs*, conforme mostrado na Figura 4.7. A tabela *macs* é responsável por armazenar o cadastro dos dispositivos e possui os campos *mac_id*, *mac* e *nome*. O primeiro campo é um número inteiro que atua como chave primária da tabela, o segundo é uma cadeia de 12 caracteres que armazena o endereço MAC, já o terceiro é uma cadeia de até 255 caracteres que representa o nome do usuário que cadastrou o respectivo endereço MAC.

A tabela *logs* é responsável por armazenar os registros de acesso dos dispositivos cadastrados e possui os campos *log_id*, *hora* e *mac_id*. O primeiro campo é um número inteiro que atua como chave primária da tabela, o segundo é do tipo *timestamp*, para registrar o horário do acesso, já o terceiro é um número inteiro que atua como chave estrangeira para a tabela *macs*, referenciando o dispositivo associado ao registro de acesso.A *query* utilizada para a criação das tabelas é mostrada no Código 4.

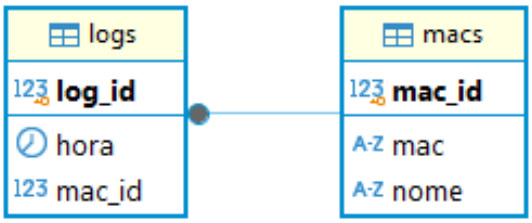


Figura 4.7: Diagrama relacional do banco de dados (Fonte: O autor).

4.3 Interface de Usuário

Para a interface de usuário, foi criada uma página *Web* utilizando o *framework Angular*. Esta página tem o objetivo de facilitar a experiência do usuário tanto ao

```sql
CREATE TABLE macs (
mac_id  SERIAL PRIMARY KEY,
mac VARCHAR(12) UNIQUE NOT NULL,
nome VARCHAR(255) NOT NULL
);

CREATE TABLE logs (
log_id SERIAL PRIMARY KEY,
hora TIMESTAMP NOT NULL,
mac_id SERIAL NOT NULL,
CONSTRAINT fk_logs_macs FOREIGN KEY (mac_id)
REFERENCES macs(mac_id)
);
```

Código 4: *Query* SQL para criar as tabelas do banco de dados (Fonte: O autor).

cadastrar o endereço MAC de seu dispositivo quanto ao acompanhar os registros de acesso, sem a necessidade de utilizar diretamente as APIs. Como visto na seção 2.3.1, o projeto é construído como uma SPA que possui 3 componentes, *lista-cadastros*, *lista-logs* e *novo-cadastro*.

O primeiro componente, observado na Figura 4.8, é responsável por exibir uma tabela com todos os endereços MAC cadastrados, assim como o nome associado a eles. Já o segundo componente, mostrado na Figura 4.10, mostra numa tabela todos os registros de acesso, com hora, nome e endereço MAC. Por fim, o último componente, exibido na Figura 4.9, permite que o usuário realize um novo cadastro, fornecendo o nome e endereço MAC.

Figura 4.8: Tela que apresenta a lista de cadastros (Fonte: O autor).

Figura 4.9: Tela para inserir novos cadastros (Fonte: O autor).

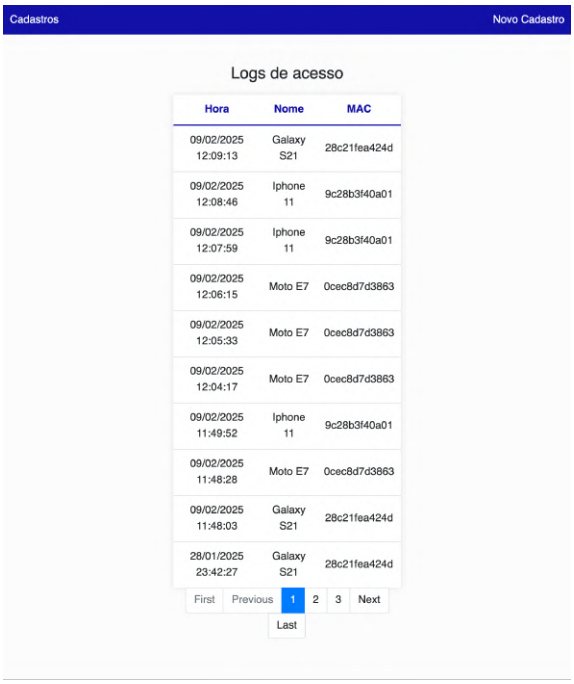


Figura 4.10: Tela que apresenta a lista de registros de acesso (Fonte: O autor).

Os componente são desenvolvidos basicamente por funções nativas do *framework Angular*, HTML e CSS, com exceção do elemento de paginação, este é construído com a utilização da biblioteca externa *jw-paginate*. Todo o código está disponibilizado no repositório *sniffer-front* [41]. Para hospedar a página foi utilizado o *GitHub Pages*, que permite a hospedagem sem necessidade de comprar domínio.

Capítulo 5

Resultados

Neste capítulo apresenta-se a análise dos resultados obtidos no estudo, com o intuito de interpretá-los à luz dos objetivos estabelecidos no Capítulo 1. Para isso, verifica-se a validade do modelo de perda de caminho, além da definição do valor limiar de RSSI conforme explicitado no Capítulo 3. O código em *Python* utilizado como auxílio para realizar as análises e construir os gráficos está no *Notebook "analysis.ipynb"*, localizado na *branch analytics* do repositório *Sniffer* [35].

5.1 Validação do modelo de perda de caminho

Para validar experimentalmente o modelo de perda de caminho apresentado na Seção 2.1.4, é seguido o procedimento descrito na Seção 3.1. As médias das medidas coletadas para as respectivas distâncias e canais são exibidas na Tabela 5.1.

Com o auxílio das bibliotecas *matplotlib* e *numpy* de *Python*, constroem-se, a partir das medidas coletadas, os gráfico mostrados nas Figuras 5.1, 5.2 e 5.3, relativas aos canais 1, 6 e 11, respectivamente. São explicitadas as barras de erro com o desvio padrão para cada distância, além da curva ajustada, obtida por meio da regressão linear.

A partir da regressão linear, observam-se os valores do expoente de perda de caminho (n), coeficiente de determinação (R^2) e RMSE de acordo com a Tabela 5.2.

O ajuste do modelo apresentou valores de R^2 acima de 0,92 para todos os canais,

Distância (m)	RSSI ch 1 (dBm)	RSSI ch 6 (dBm)	RSSI ch 11 (dBm)
0,5	-39,50	-39,60	-40,05
1,0	-45,10	-44,65	-45,85
1,5	-51,65	-51,00	-52,15
2,0	-52,95	-52,75	-54,00
2,5	-52,85	-52,35	-52,60
3,0	-53,05	-52,15	-52,90
3,5	-54,95	-53,90	-54,75
4,0	-56,10	-55,30	-56,30
4,5	-56,35	-56,00	-56,25
5,0	-58,00	-57,55	-58,25

Tabela 5.1: Médias de RSSI para as respectivas distâncias e canais (Fonte: O autor).

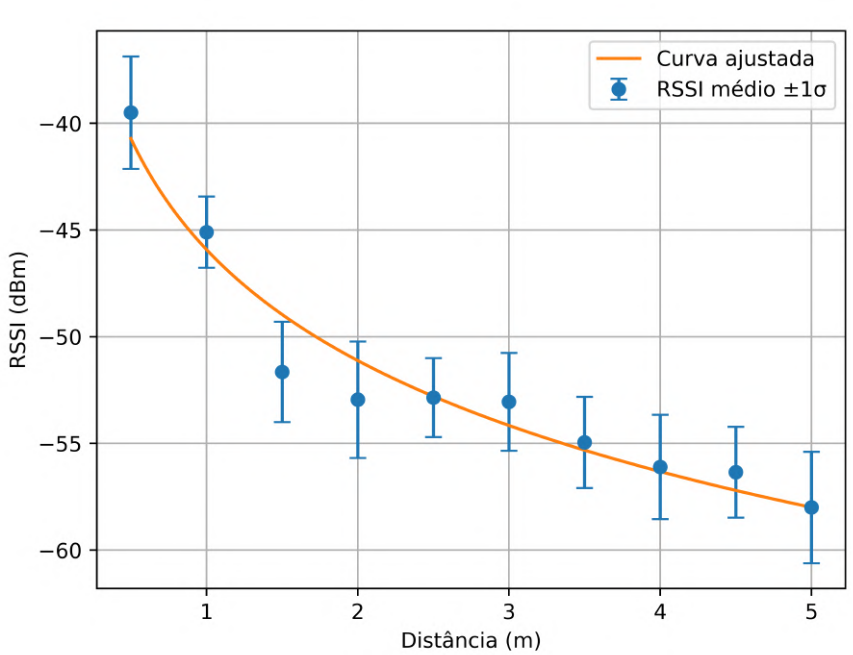


Figura 5.1: Gráfico de RSSI versus distância para o canal 1 (Fonte: O autor).

indicando que um alto percentual da variância observada nas médias de RSSI é explicada pela regressão linear em função de $\log_{10}(d)$. Esses valores, consideravelmente elevados, demonstram que a dependência entre potência recebida e distância está sendo descrita de forma bastante fiel pelo modelo proposto. Observa-se também que os valores de RMSE obtidos foram inferiores aos desvios-padrão das medições individuais, que variam entre 1,61 e 2,82 dB. Além disso, os valores encontrados para o expoente de perda de caminho estão dentro do intervalo teórico mostrado na

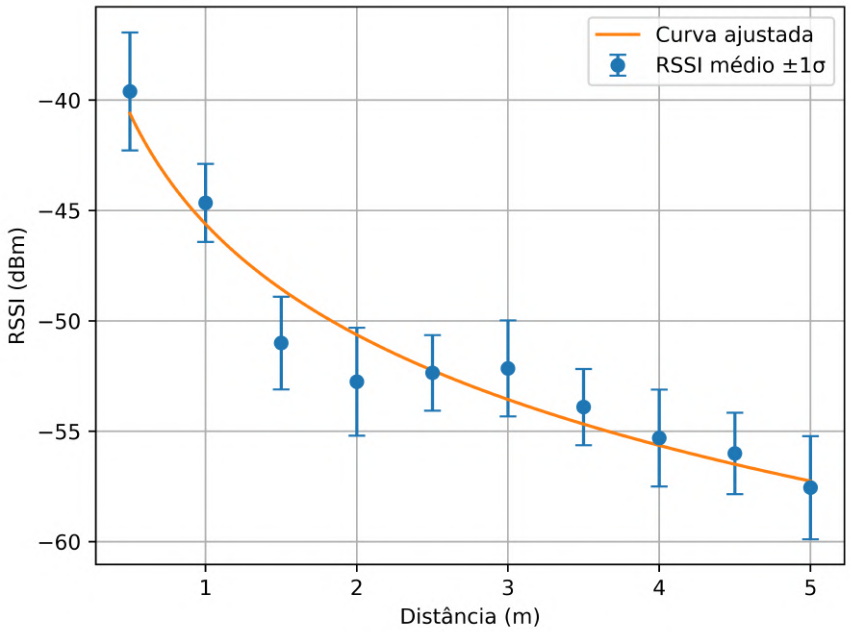


Figura 5.2: Gráfico de RSSI versus distância para o canal 6 (Fonte: O autor).

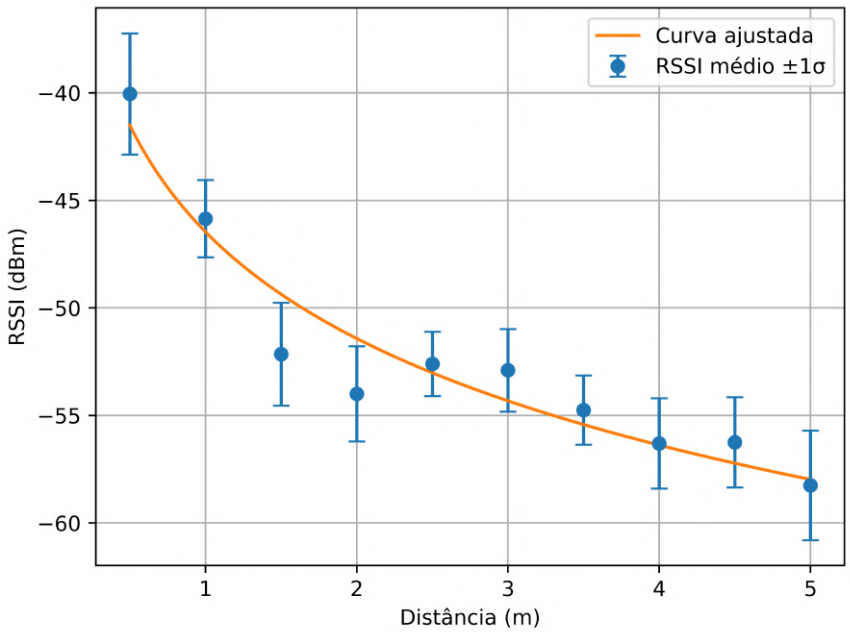


Figura 5.3: Gráfico de RSSI versus distância para o canal 11 (Fonte: O autor).

Tabela 2.2, entre 1,6 e 1,8.

	Canal 1	Canal 6	Canal 11
n	1,728	1,666	1,647
R^2	0,948	0,942	0,923
RMSE	1,220 dB	1,245 dB	1,435 dB

Tabela 5.2: Métricas obtidas a partir da regressão linear (Fonte: O autor).

5.2 Definição do limiar de RSSI

O limiar de RSSI é definido experimentalmente seguindo o procedimento apresentado na Seção 3.2. As médias dos valores coletados para cada dispositivo à distância de $0,5\,m$ são mostrados na Tabela 5.3. A partir destas médias, foi calculada a média geral para todos os dispositivos da amostra, resultando em $-41,46\,\text{dB}$.

Dispositivo	RSSI (dBm)
Apple Iphone 16 Pro Max	-39,50
Apple Iphone 11	-42,35
Samsung Galaxy S21	-42,15
Samsung Galaxy J7	-41,25
Motorola Moto E7	-42,05

Tabela 5.3: Medidas de RSSI coletadas a $0,5\,m$ para diferentes dispositivos (Fonte: O autor).

Sabendo que o valor de t_{crit} é $2,132$ para o nível de confiança de $95\,\%$, considerando um teste unilateral, ao aplicar os dados obtidos à Equação 3.2, encontra-se o valor de $-40,34\,\text{dB}$ para o limiar de RSSI. Dado o nível de significância de $5\,\%$, caso uma nova medida apresente valor maior que o limiar, existe $5\,\%$ de probabilidade de que o dispositivo não esteja dentro do limite de $0,5\,m$, como ilustrado na região sombreada da Figura 5.4, conclue-se dessa forma, com $95\,\%$ de confiança, que o dispositivo está estatisticamente mais próximo do que $0,5\,m$.

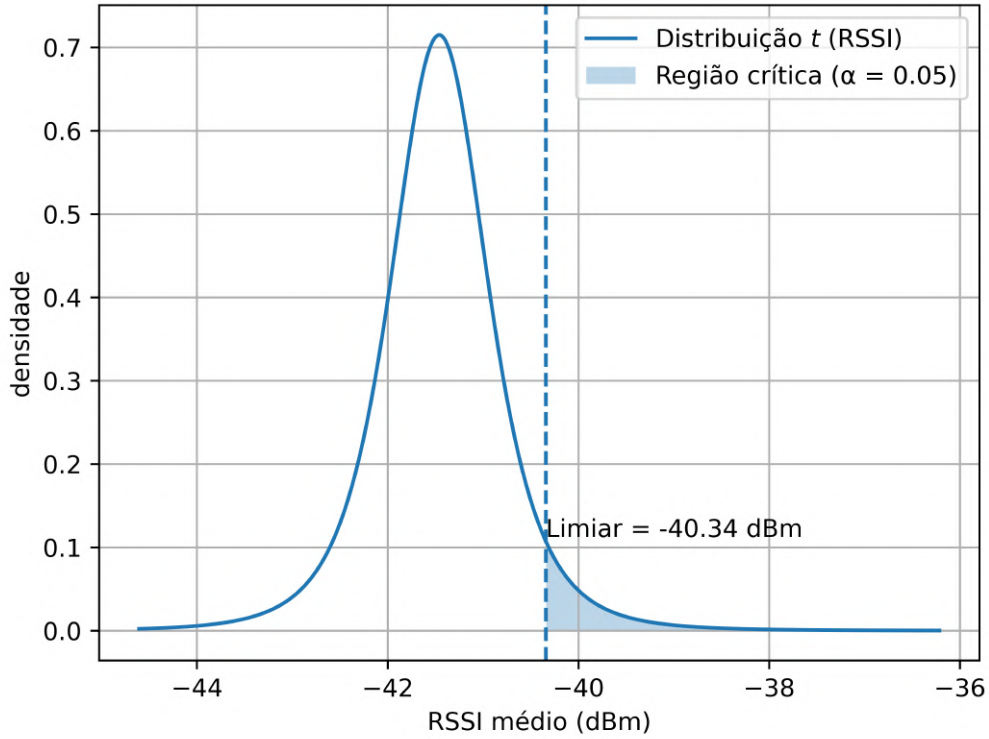


Figura 5.4: Distribuição de probabilidade para os valores de RSSI coletados à $0,5\,m$ (Fonte: O autor).

Capítulo 6

Considerações Finais

Este capítulo final consolida os resultados e aprendizados obtidos ao longo do desenvolvimento deste Trabalho de Conclusão de Curso. Apresenta-se a conclusão geral do projeto, destacando o alcance dos objetivos propostos, as principais dificuldades enfrentadas durante a execução e sugestões para trabalhos futuros que possam dar continuidade ou expandir a pesquisa realizada.

6.1 Conclusão

O objetivo central deste trabalho foi desenvolver, implementar e validar um sistema integrado de controle de acesso baseado na detecção de pacotes Wi-Fi emitidos por *smartphones*, utilizando o microcontrolador ESP32 como unidade de processamento embarcada e integrando-o a uma infraestrutura de computação em nuvem para gerenciamento de dados e interface com o usuário.

Ao longo do desenvolvimento, os objetivos específicos definidos foram alcançados com sucesso. O microcontrolador ESP32, especificamente a placa *WIFI Kit 32*, foi configurado para operar em modo promíscuo, permitindo a captura e análise de pacotes Wi-Fi no ambiente. Foi implementada uma máquina de estados que gerencia a lógica do sistema, desde a conexão inicial e obtenção da lista de dispositivos autorizados via HTTP até a análise dos pacotes capturados e a subsequente liberação (ou não) do acesso. A comunicação entre o microcontrolador e o servidor Web foi

estabelecida com sucesso, permitindo a consulta à lista de MACs cadastrados e o envio de registros de acesso para o banco de dados.

Um aspecto crucial do trabalho foi a validação experimental do modelo de perda de caminho log-distância para o ambiente interno específico, utilizando os canais 1, 6 e 11 da faixa de 2,4 GHz. Os resultados obtidos no Capítulo 5 demonstraram uma boa aderência do modelo aos dados coletados, com coeficientes de determinação (R^2) superiores a $0,92$ e expoentes de perda de caminho (n) entre $1,647$ e $1,728$, valores consistentes com a teoria para ambientes internos sem obstáculos.

O limiar de RSSI, essencial para determinar a proximidade do dispositivo, foi definido experimentalmente em $-40,34\,\text{dBm}$, com base em medições realizadas a $0,5$ metros com uma amostra de cinco *smartphones* distintos e utilizando um teste de hipótese com $95\,\%$ de confiança. Este limiar permite ao sistema inferir, com significância estatística, se um dispositivo detectado está suficientemente próximo para ter o acesso concedido.

A infraestrutura de *backend* foi desenvolvida utilizando *Java* com *Spring Boot* para a API REST e *PostgreSQL* como SGBD, ambos hospedados na plataforma AWS *Elastic Beanstalk* e RDS, respectivamente. Essa arquitetura demonstrou ser robusta para gerenciar os cadastros de dispositivos, bem como os *logs* de acesso. De modo complementar, uma interface de usuário Web foi desenvolvida com *Angular* e hospedada no *GitHub Pages*, oferecendo uma forma intuitiva para os usuários cadastrarem seus dispositivos e visualizarem o histórico de acessos. O sistema também fornece *feedback* visual ao usuário por meio do *display* OLED integrado à placa de desenvolvimento.

Conclui-se, portanto, que o projeto atingiu seus objetivos, demonstrando a viabilidade técnica da utilização da detecção de pacotes Wi-Fi como método de controle de acesso. O sistema integra conceitos de IOT, sistemas embarcados, desenvolvimento Web e computação em nuvem, resultando em um protótipo funcional que pode servir como uma alternativa conveniente e moderna aos métodos tradicionais de controle de acesso, como chaves ou cartões, especialmente em ambientes contro-

lados como laboratórios didáticos.

6.2 Dificuldades Encontradas

Durante o desenvolvimento do projeto, as seguintes dificuldades foram identificadas:

- **Variabilidade do RSSI:** O RSSI mostrou-se inerentemente variável, mesmo para distâncias fixas. Fatores como orientação do dispositivo, reflexões de sinal, interferência de outras redes Wi-Fi ou fontes de ruído eletromagnético, e até mesmo pequenas movimentações no ambiente podem influenciar as medidas. Isso tornou a definição do limiar um desafio, justificando a abordagem estatística utilizada (teste t de Student) baseada em uma amostra de dispositivos.

- **Diferenças entre Dispositivos:** Constatou-se que diferentes modelos e marcas de *smartphones* possuem características de transmissão Wi-Fi distintas, resultando em variações nos valores de RSSI medidos sob as mesmas condições. Isso reforçou a importância de utilizar uma amostra diversificada para calcular o limiar de RSSI, visando abranger uma gama maior de dispositivos.

- **Limitações na Captura de Pacotes pelo ESP32:** Observou-se que, ao operar em modo promíscuo para capturar os pacotes Wi-Fi, o ESP32 pode apresentar limitações em sua capacidade de processamento, como por exemplo, a possibilidade de receber pacotes apenas de um canal por vez, resultando na perda de alguns deles. Essa perda pode ser mais acentuada em ambientes com alta densidade de tráfego de rede sem fio. Embora o sistema tenha mostrado-se funcional para o propósito desejado, essa limitação pode, ocasionalmente, atrasar a detecção de um dispositivo autorizado que esteja próximo, impactando a responsividade imediata do sistema.

- **Randomização de Endereço MAC:** O crescente uso da randomização de

endereços MAC por dispositivos móveis modernos representa um desafio a sistemas baseados na identificação por endereço MAC. O sistema funciona apenas se a randomização estiver desativada, ou se o dispositivo estiver conectado a um AP com algum MAC fixo.

6.3 Trabalhos Futuros

Com base no trabalho realizado e nas dificuldades encontradas, diversas oportunidades para aprimoramento e expansão do sistema podem ser exploradas em trabalhos futuros:

- **Aprimorar a segurança:** Implementar camadas adicionais de segurança, como criptografia na comunicação entre o ESP32 e a API, garantindo o uso de HTTPS, além de explorar mais mecanismos de autenticação.
- **Acionamento físico:** Integrar o sistema a atuadores físicos, como relés, travas elétricas ou solenoides) para controlar o acesso físico a portas, armários ou equipamentos nas bancadas do LDE, tornando a aplicação mais prática e completa no contexto originalmente pensado para o laboratório.
- **Escalabilidade** Expandir a arquitetura para suportar múltiplos dispositivos ESP32 detectores distribuídos no ambiente, permitindo cobrir diferentes pontos de acesso.
- **Comunicação MQTT:** Utilizar o protocolo *Message Queuing Telemetry Transport* (MQTT) para a comunicação entre os múltiplos detectores e um servidor central, aproveitando sua leveza e adequação para comunicação em redes com múltiplos dispositivos IOT, substituindo as requisições HTTP individuais de cada ESP32.

Referências

[1] Lin, G. “Some Real-World Examples of Internet of Things (IoT) Applications in Renewable Energy Projects”. Bachelor’s thesis, Lappeenranta–Lahti University of Technology LUT, 2024.

[2] Tanenbaum, A. S., Feamster, N., Wetherall, D. J. *Redes de Computadores*. Bookman, 2021.

[3] “Redes WLAN: Canalização Disponível”. 2024. Disponível em: <https://www.teleco.com.br/tutoriais/tutorial802-11ac/pagina_3.asp>.

[4] Kurose, J. F., Ross, K. W. *Redes de computadores e a Internet: uma abordagem top-down*. Bookman, 2021.

[5] “ESP32 Modulo WIFI e Bluetooth ESP-WROOM-32”. 2024. Disponível em: <https://www.rsrobotica.com.br/esp32-modulo-wifi-e-bluetooth-esp-wroom-32>.

[6] “ESP32 Series Datasheet”. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>.

[7] Elmasri, R., Navathe, S. B. *Sistemas De Banco De Dados*. Pearson, 2019.

[8] “Uma visão geral do HTTP”. 2024. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview>.

[9] “The Spring MVC architecture”. 2024. Disponível em: <https://www.researchgate.net/figure/The-Spring-MVC-architecture-as-depicted-in-16_fig5_349049076>.

[10] “WIFI Kit 32 ESP32 WIFI wireless with 0.96 inch OLED Display”. 2025. Disponível em: <https://www.diymore.cc/products/diymore-wifi-kit-32-esp32-wifi-wireless-with-0-96-inch-oled-display-cp2102-develo

[11] Rappaport, T. S. *Comunicações Sem Fio: Princípios e Práticas*. Pearson, 2008.

[12] Ashton, K. “That ’Internet of Things’ Thing”. 2009. Disponível em: <https://www.rfidjournal.com/expert-views/that-internet-of-things-thing/73881/>.

[13] IEEE 802.11 Working Group. *IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 7: Enhancements for Extremely High Throughput (EHT).* Standard IEEE Std 802.11be-2024, IEEE, jan. 2024.

[14] Agência Nacional de Telecomunicações (ANATEL). *Resolução nº 506, de 21 de novembro de 2012: Regulamenta o uso do espectro de radiofrequências na faixa de 2,4 GHz e 5 GHz para sistemas de WLAN.* Relatório técnico.

[15] Lee, E. A., Seshia, S. A. *Introduction to Embedded Systems: A Cyber–Physical Systems Approach.* MIT Press, 2016.

[16] Wilmshurst, T. *Designing embedded systems with PIC microcontrollers: Principles and applications.* Elsevier, 2011.

[17] Tanenbaum, A. S., Bos, H. *Sistemas Operacionais Modernos.* Pearson, 2015.

[18] Li, Q., Yao, C. *Real-Time Concepts for Embedded Systems.* CMP Books, 2003.

[19] Flanagan, D. *JavaScript: O Guia Definitivo.* O’Reilly, 2012.

[20] “10 frameworks e bibliotecas JavaScript mais importantes do mercado”. 2024. Disponível em: <https://blog.geekhunter.com.br/frameworks-javascript-e-bibliotecas-java/>.

[21] “Angular”. 2024. Disponível em: <https://angular.io/>.

[22] “TypeScript”. 2024. Disponível em: <https://www.typescriptlang.org/>.

[23] “A melhor maneira de aprender desenvolvimento em back-end para a web”. 2024. Disponível em: <https://www.freecodecamp.org/portuguese/news/a-melhor-maneira-de-aprender-desenvolvimento-em-backend-para-a-web/>.

[24] “Top 10 frameworks para desenvolver seu backend”. 2024. Disponível em: <https://blog.back4app.com/pt/melhores-frameworks-para-desenvolver-backend/>.

[25] “Spring Boot.” 2024. Disponível em: <https://spring.io/projects/spring-boot>.

[26] Codd, E. “A Relational Model of Data for Large Shared Data Banks”, *Communications of the ACM*, v. 13, pp. 377–387, 1970.

[27] Fielding, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Tese de Doutorado, University of California, Irvine, 2000.

[28] Krasner, G. E., Pope, S. T. “A Description of the Model–View–Controller User Interface Paradigm in the Smalltalk–80 System”, *Journal of Object Oriented Programming*, v. 1, n. 3, pp. 26–49, 1988.

[29] Rescorla, E. “HTTP Over TLS”. RFC 2818, 2000. Disponível em: <https://www.rfc-editor.org/info/rfc2818>.

[30] Erl, T., Mahmood, R., Puttini, Z. *Cloud Computing: Concepts, Technology & Architecture*. Prentice Hall, 2013.

[31] Mell, P., Grance, T. *The NIST Definition of Cloud Computing*. Special publication 800-145, National Institute of Standards and Technology (NIST), set. 2011.

[32] Miranda, J., Abrishambaf, R., Gomes, T., et al. “Path loss exponent analysis in Wireless Sensor Networks: Experimental evaluation”. In: *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, pp. 54–58, 2013. doi: 10.1109/INDIN.2013.6622857.

[33] “WiFi Kit 32 Development Kit”. 2025. Disponível em: <https://resource.heltec.cn/download/WiFi_Kit_32/WiFi%20Kit32.pdf/>.

[34] “PlatformIO IDE”. 2025. Disponível em: <https://platformio.org/platformio-ide>.

[35] “Repositório Sniffer”. 2025. Disponível em: <https://github.com/gabrielfirmo/sniffer>.

[36] “Documentação do módulo Wi-Fi no framework ESP-IDF”. 2025. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/network/esp_wifi.html>.

[37] “lwIP - A Lightweight TCP/IP stack”. 2025. Disponível em: <https://savannah.nongnu.org/projects/lwip/>.

[38] “Interfacing the Arduino with an SSD1306 driven OLED Display”. 2025. Disponível em: <https://robotcantalk.blogspot.com/2015/03/interfacing-arduino-with-ssd1306-driven.html>.

[39] “Repositório Sniffer API”. 2025. Disponível em: <https://github.com/gabrielfirmo/sniffer-api>.

[40] “spring initializr”. 2025. Disponível em: <https://start.spring.io>.

[41] “Repositório Sniffer Frontend”. 2025. Disponível em: <https://github.com/gabrielfirmo/sniffer-front>.

Apêndice A

Configuração do *Elastic Beanstalk*

Neste apêndice são mostrados os passos necessários para configurar a hospedagem da API no AWS *Elastic Beanstalk*. O processo é intuitivo, o início dá-se ao pressionar *Create application*, bastando em seguida preencher os campos e pressionar *next* até o final do fluxo, conforme exibido nas Figuras A.1, A.2, A.3, A.4, A.5 e A.6.

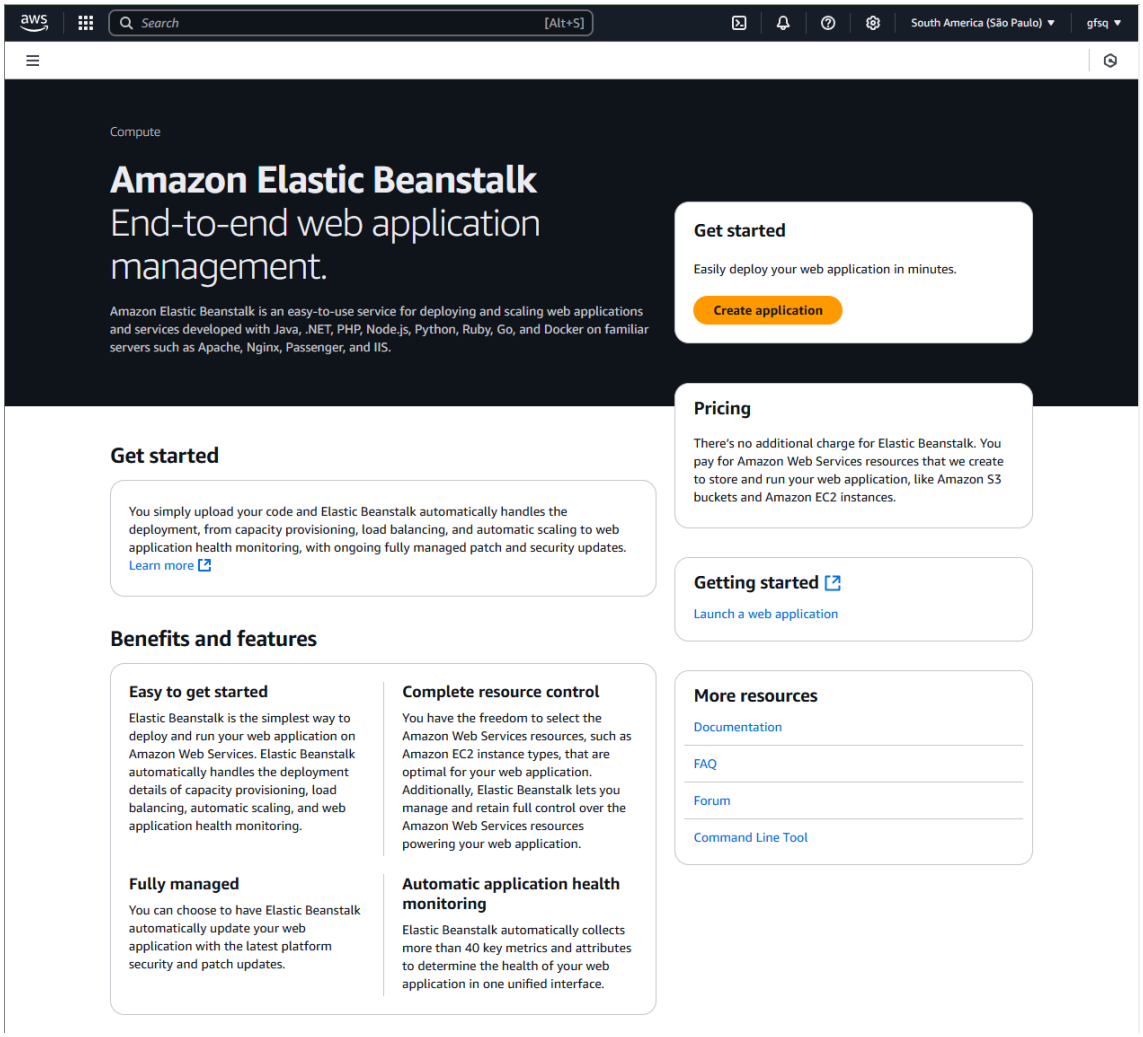


Figura A.1: Configuração do *Elastic Beanstalk* - Parte 1 (Fonte: O autor.).

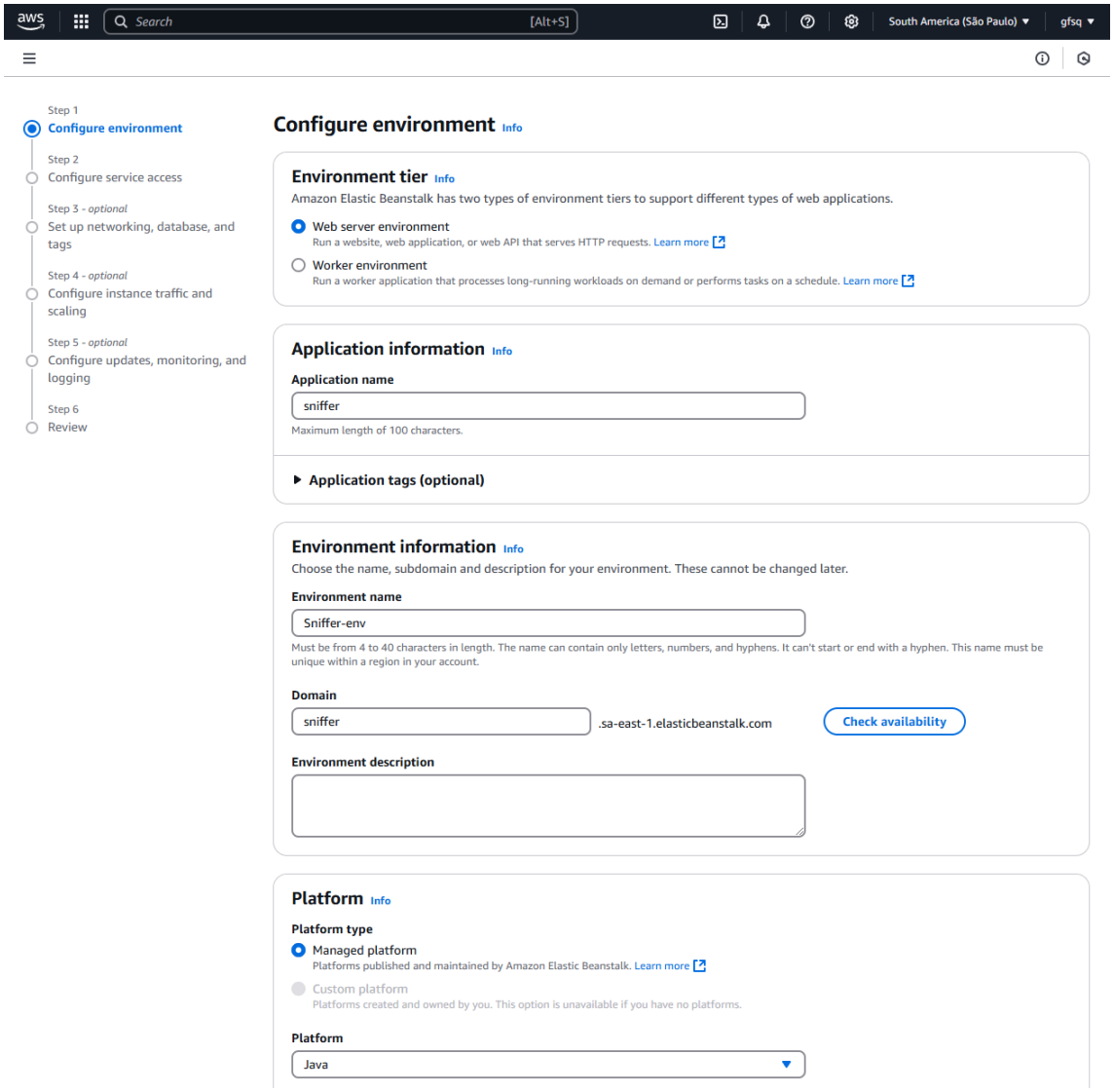


Figura A.2: Configuração do *Elastic Beanstalk* - Parte 2 (Fonte: O autor.).

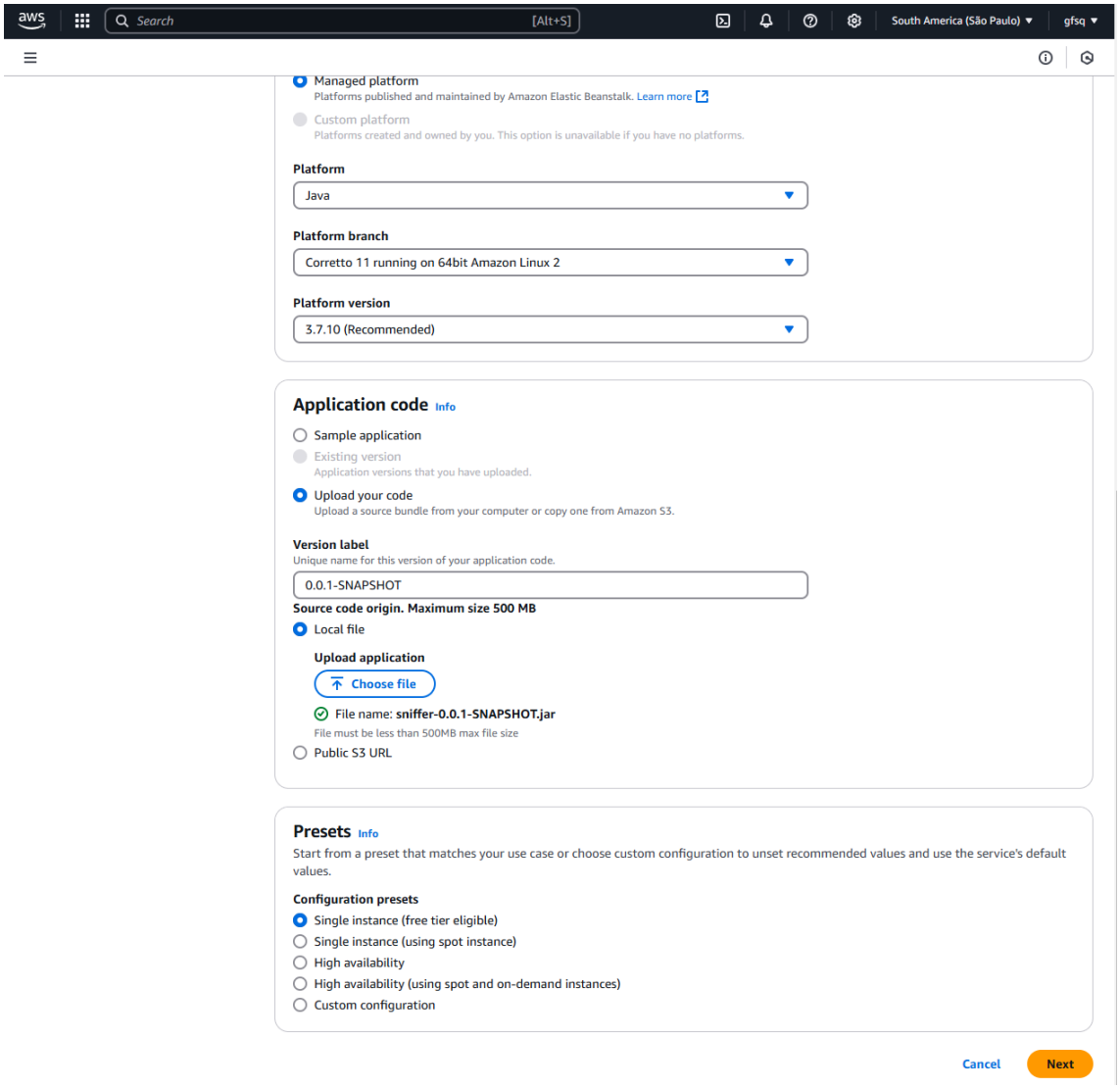


Figura A.3: Configuração do *Elastic Beanstalk* - Parte 3 (Fonte: O autor.).

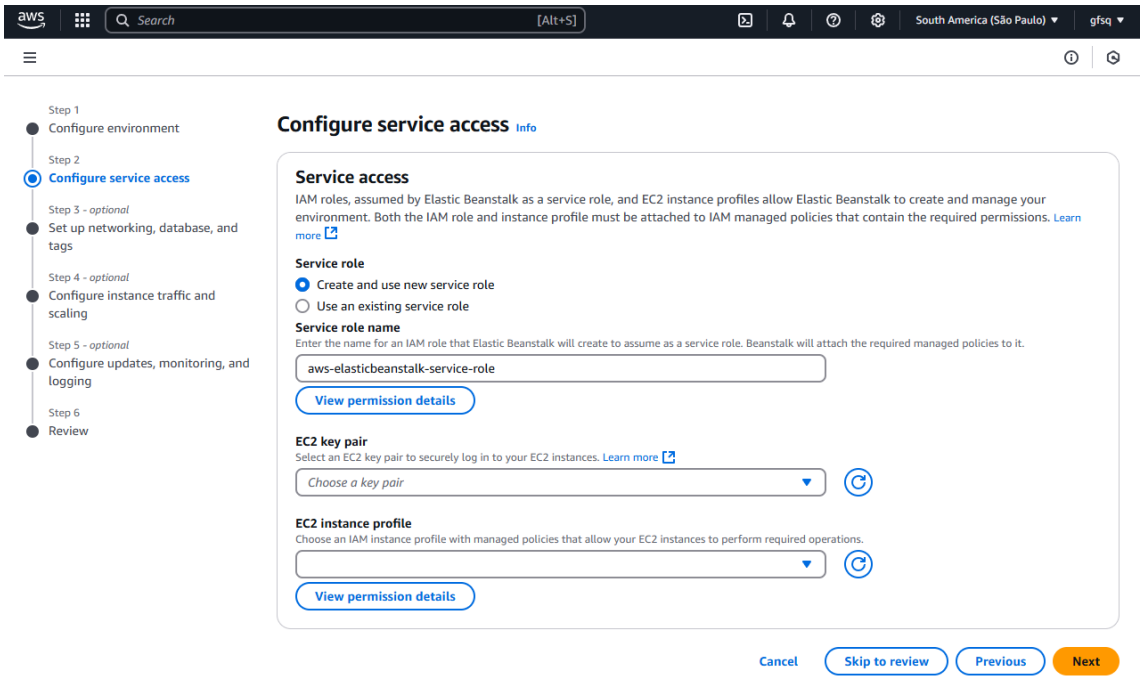


Figura A.4: Configuração do *Elastic Beanstalk* - Parte 4 (Fonte: O autor.).

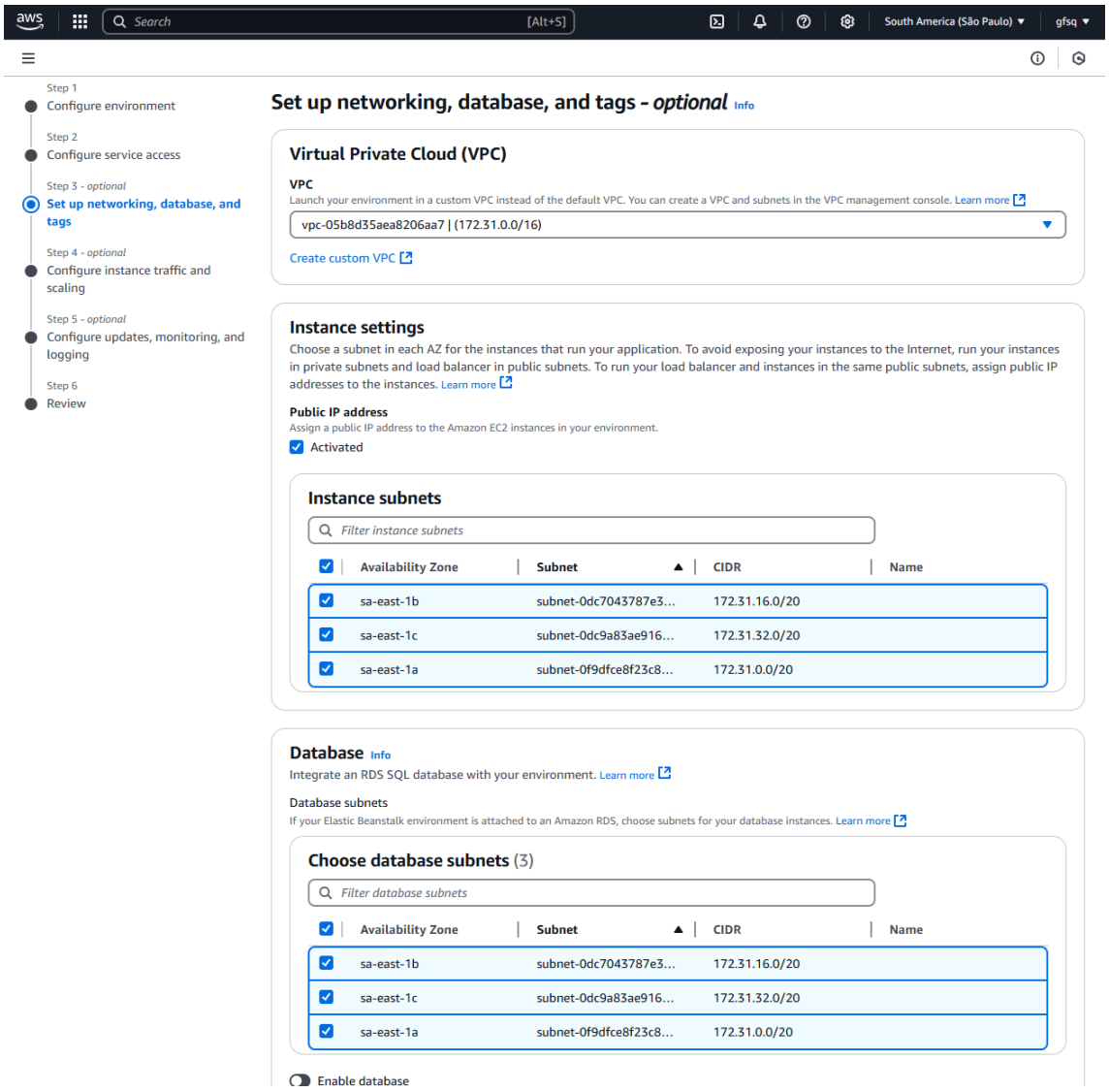


Figura A.5: Configuração do *Elastic Beanstalk* - Parte 5 (Fonte: O autor.).

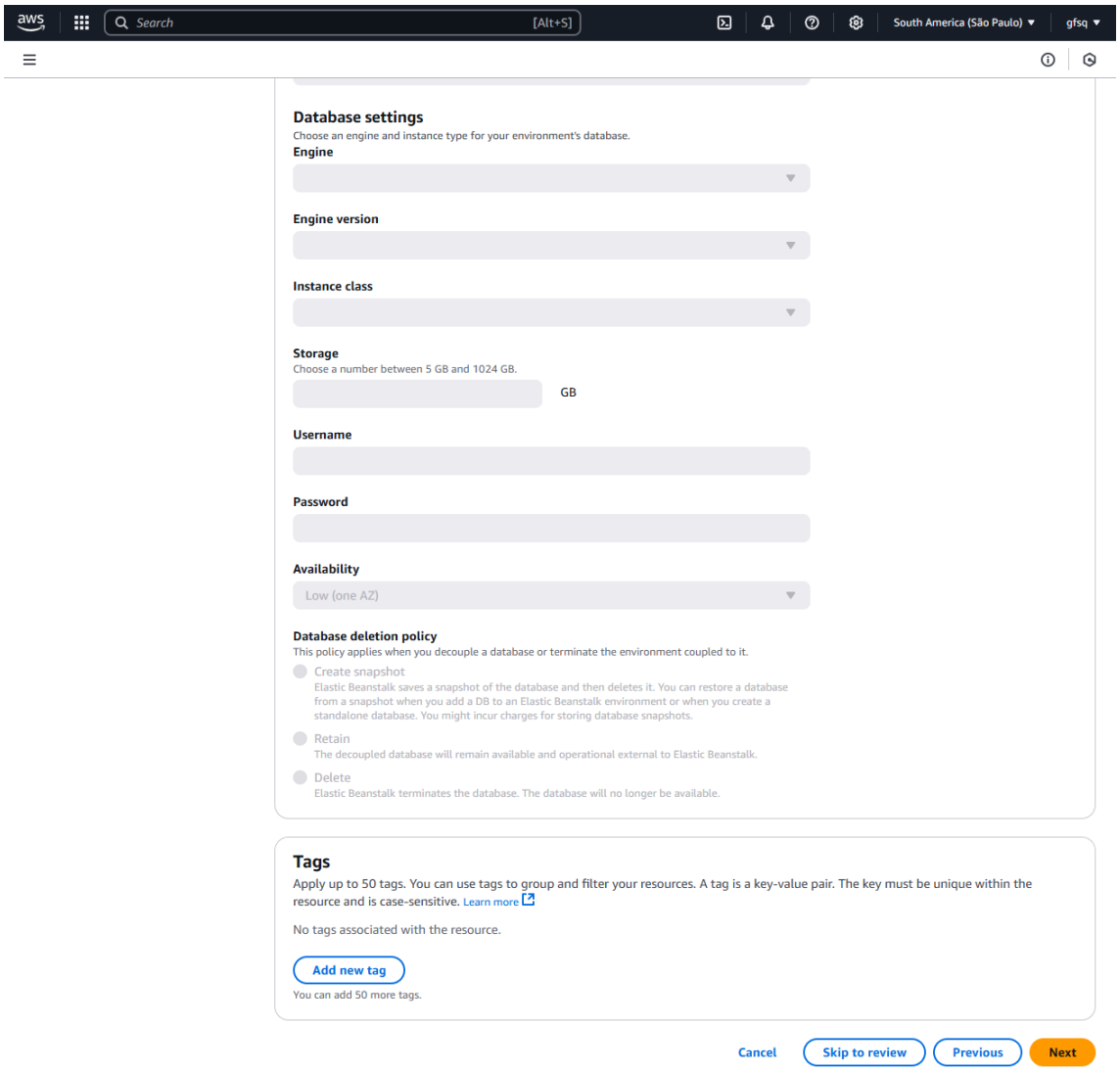


Figura A.6: Configuração do *Elastic Beanstalk* - Parte 6 (Fonte: O autor.).

Apêndice B

Configuração do RDS

Neste apêndice são mostrados os passos necessários para configurar o banco de dados utilizando o AWS RDS. O processo é intuitivo, o início dá-se ao pressionar *Create database*, bastando em seguida escolher o SGBD, preencher os campos e pressionar *next* até o final do fluxo, conforme exibido nas Figuras B.1, B.2, B.3, B.4.

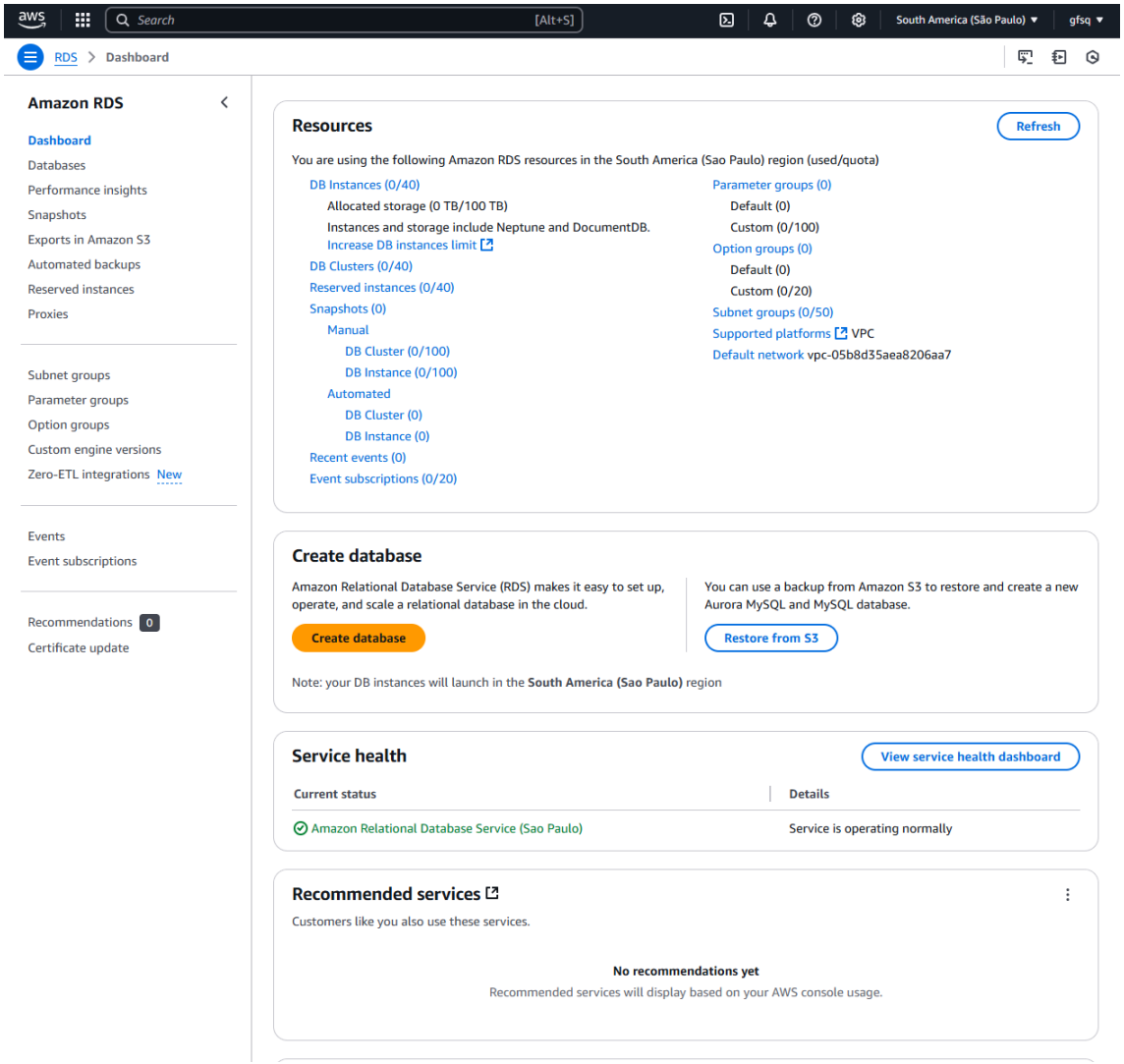


Figura B.1: Configuração do RDS - Parte 1 (Fonte: O autor.).

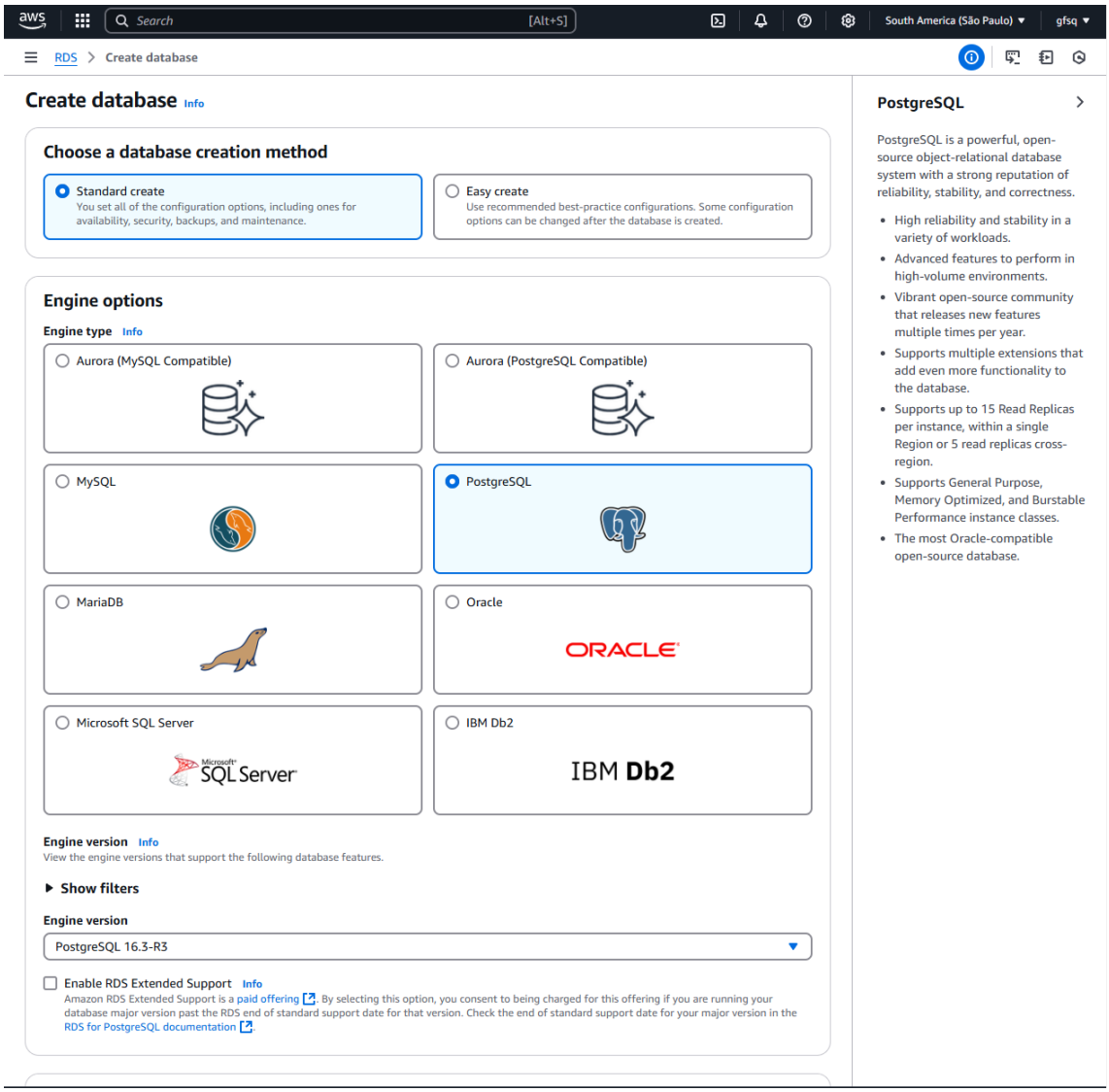


Figura B.2: Configuração do RDS - Parte 2 (Fonte: O autor.).

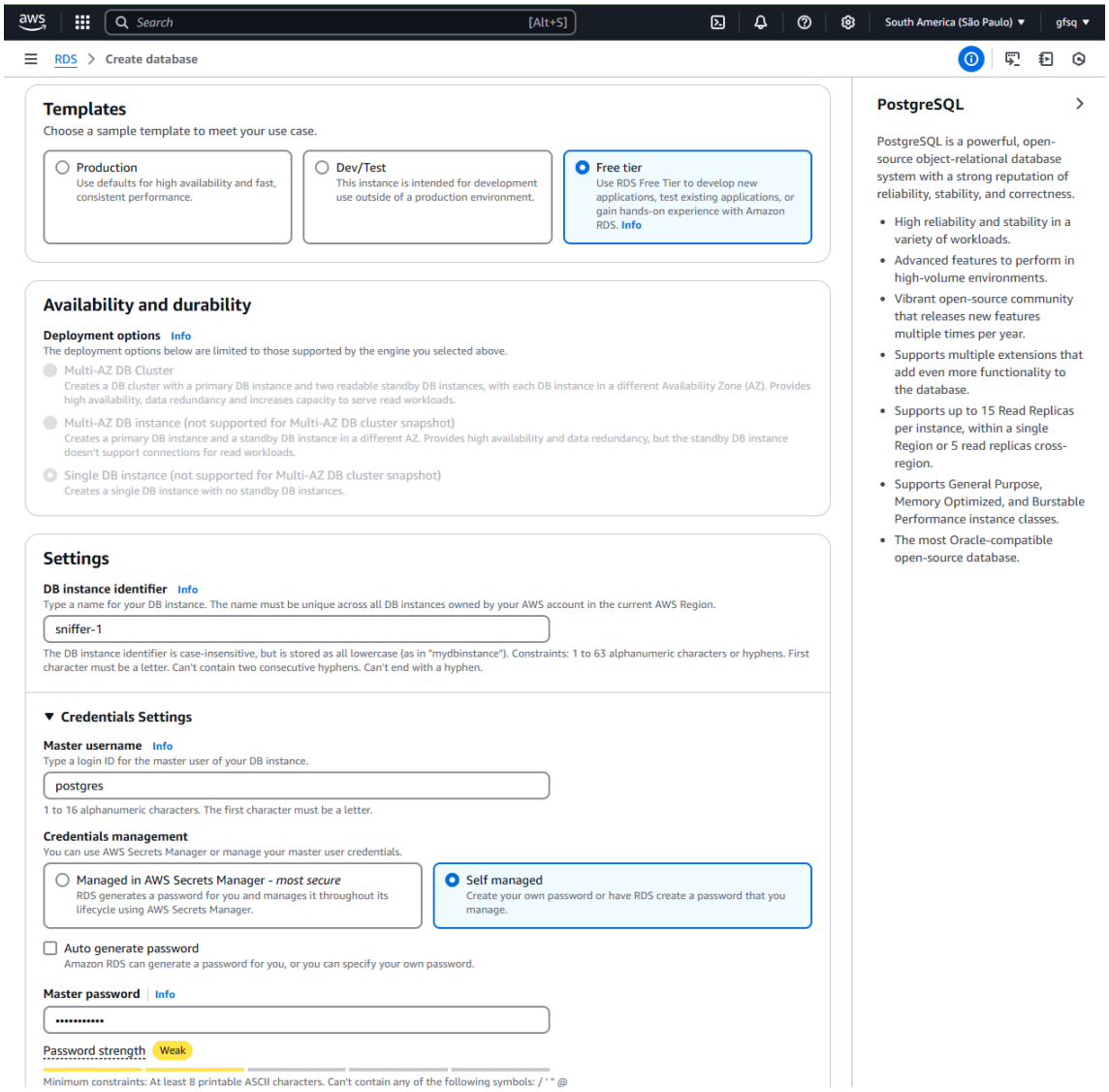


Figura B.3: Configuração do RDS - Parte 3 (Fonte: O autor.).

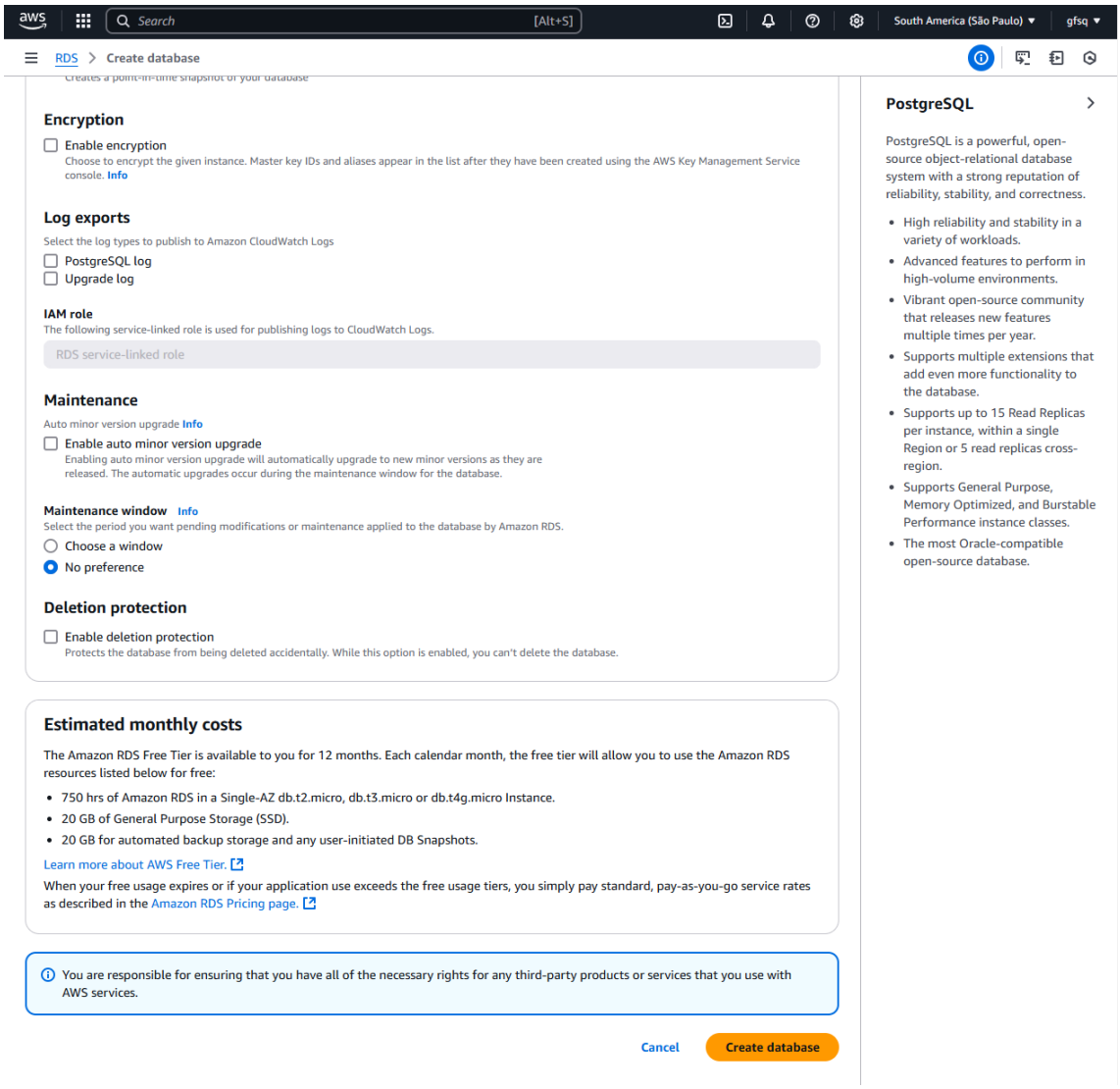


Figura B.4: Configuração do RDS - Parte 4 (Fonte: O autor.).