MONOGRAFIA DE PROJETO FINAL DE GRADUAÇÃO

APLICAÇÃO DA TECNOLOGIA NEAR FIELD COMMUNICATION NO ATENDIMENTO DE EMERGÊNCIA DIRECIONADO A PROFISSIONAIS DE SAÚDE E SEGURANÇA

Leonardo de Oliveira Almeida

Rafael dos Santos Pereira

Curso Superior de Engenharia de Redes de Comunicação

DEPARTAMENTO DE ENGENHARIA ELÉTRICA
FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

MONOGRAFIA DE PROJETO FINAL DE GRADUAÇÃO

APLICAÇÃO DA TECNOLOGIA NEAR FIELD COMMUNICATION NO ATENDIMENTO DE EMERGÊNCIA DIRECIONADO A PROFISSIONAIS DE SAÚDE E SEGURANÇA

Leonardo de Oliveira Almeida

Rafael dos Santos Pereira

Monografia de Projeto Final de Graduação submetida ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Bacharel em Engenharia de Redes de Comunicação

Banca Examinadora

Dr. Georges Daniel Amvame Nze, EnE/UnB
Orientador ______________________________

Dr. Fábio Lúcio Lopes de Mendonça, EnE/UnB
Examinador Interno ______________________________

Esp. Diego Martins de Oliveira, IFB/Brasília
Examinador Externo ______________________________

FICHA CATALOGRÁFICA

ALMEIDA, LEONARDO DE OLIVEIRA; PEREIRA, RAFAEL DOS SANTOS
APLICAÇÃO DA TECNOLOGIA NEAR FIELD COMMUNICATION NO ATENDIMENTO DE EMERGÊNCIA DIRECIONADO A PROFISSIONAIS DE SAÚDE E SEGURANÇA [Distrito Federal] 2023.
xvi, 50 p., 210 x 297 mm (ENE/FT/UnB, Bacharel, Engenharia de Redes de Comunicação, 2023).
Monografia de Projeto Final de Graduação - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Militar	2. API Restful
3. Banco de Dados	4. Aplicação Móvel
5. NFC	6. Emergência

REFERÊNCIA BIBLIOGRÁFICA

ALMEIDA, L.O.; PEREIRA, R.S. (2023). *APLICAÇÃO DA TECNOLOGIA NEAR FIELD COMMUNICATION NO ATENDIMENTO DE EMERGÊNCIA DIRECIONADO A PROFISSIONAIS DE SAÚDE E SEGURANÇA*. Monografia de Projeto Final de Graduação, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 50 p.

CESSÃO DE DIREITOS

AUTORES: Leonardo de Oliveira Almeida; Rafael dos Santos Pereira
TÍTULO: APLICAÇÃO DA TECNOLOGIA NEAR FIELD COMMUNICATION NO ATENDIMENTO DE EMERGÊNCIA DIRECIONADO A PROFISSIONAIS DE SAÚDE E SEGURANÇA.
GRAU: Bacharel em Engenharia de Redes de Comunicação ANO: 2023

É concedida à Universidade de Brasília permissão para reproduzir cópias desta Monografia de Graduação e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. Do mesmo modo, a Universidade de Brasília tem permissão para divulgar este documento em biblioteca virtual, em formato que permita o acesso via redes de comunicação e a reprodução de cópias, desde que protegida a integridade do conteúdo dessas cópias e proibido o acesso a partes isoladas desse conteúdo. O autor reserva outros direitos de publicação e nenhuma parte deste documento pode ser reproduzida sem a autorização por escrito do autor.

Leonardo de Oliveira Almeida
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP: 70919-970 - Brasília-DF - Brasil

Rafael dos Santos Pereira
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP: 70919-970 - Brasília-DF - Brasil

"A ship in harbour is safe, but that is not what ships are built for"

John A. Shedd

AGRADECIMENTOS

Em primeiro lugar, a Deus, que sabe de todas as coisas, sempre me acompanhou e foi fundamental para conseguir atingir meus objetivos.

Aos meus pais Marcelo e Ana Paula, que me ensinaram o que é ser uma família, me incentivaram nos momentos difíceis, me ajudaram quando eu precisava, sempre me proporcionaram educação, carinho e amor.

A meus irmãos Lucas e Luanna, que me ajudaram com as tarefas de casa, quando eu precisava fazer as atividades da faculdade e por toda a parceria.

A meus avós Terezinha, Glaci e João Alberto, que sempre intercederam por mim e torceram pelo meu sucesso.

A meus amigos de curso, por tornar essa caminhada mais fácil, por todos os momentos desafiadores e por estarem sempre dispostos a ajudar.

A minha namorada Nátaly, por toda a paciência nessa fase, parceria e amor.

A minha cachorrinha Princesa, que estava sempre fazendo companhia.

Aos amigos, que sempre estiveram ao meu lado, pela amizade incondicional e pelo apoio demonstrado ao longo de todo esse período.

Ao nosso orientador Prof. Dr. Georges, que sempre se mostrou disposto a ajudar, acreditou no projeto e confiou em nós.

"A caminhada é longa, mas o resultado faz cada passo valer a pena "

Leonardo de Oliveira Almeida

A Deus e a Nossa Senhora pelo dom da vida e proteção durante toda a minha jornada, abrindo as portas certas e me ajudando a trilhar o melhor caminho.

A meus pais Fábio e Sandra Cristina por me criarem da melhor forma possível e me apoiarem na conquista de todos os meus objetivos, especialmente na minha escolha por Engenharia de Redes de Comunicação.

A meus avós Fernando e Glória, que me protegem lá de cima, e Job e Teresa, por todo o amor, carinho e histórias valiosas que levo comigo.

A meus companheiros de curso, por toda a irmandade e camaradagem, vencendo todos os desafios juntos como verdadeiros irmãos.

A minha namorada Daniela, pelo amor, companheirismo e carinho nos momentos felizes e turbulentos.

Ao meu cãozinho Rocky, por estar ao meu lado até o fim.

Ao nosso orientador Prof. Dr. Georges, por acreditar em nosso potencial e nos guiar nesse projeto.

"– Quem estará nas trincheiras ao teu lado?
– E isso importa?
– Mais do que a própria guerra."

Rafael dos Santos Pereira

RESUMO

O uso de tecnologias da informação, aplicadas em um cenário de atendimento de emergência, pode agregar na segurança do paciente, diminuindo riscos de erros médicos e agilizando o atendimento. Assim, o trabalho propõe o desenvolvimento e a implementação de um sistema de consulta a dados de pacientes, inseridas em um contexto de organização militar. A arquitetura prevê o uso de tecnologias como API *RESTful*, banco de dados relacional, aplicação móvel e NFC. Os testes, realizados em ambiente local, e seus respectivos resultados possibilitaram a verificação do funcionamento da arquitetura, que operou conforme esperado, cumprindo todos os objetivos propostos. Também foi possível observar limitações na arquitetura, passíveis de correção em trabalhos futuros.

Palavras-chave: API *RESTful*, Banco de Dados, Militar, Emergência, Aplicação Móvel, NFC.

ABSTRACT

The use of information technologies, applied in an emergency care scenario, can contribute to patient safety by reducing the risks of medical errors and speeding up the care process. Thus, the work proposes the development and implementation of a patient data query system, integrated within a military organization context. The architecture envisages the use of technologies such as RESTful APIs, relational databases, mobile applications, and NFC. The tests conducted in a local environment, along with their respective results, allowed for the verification of the architecture's functionality, which operated as expected, fulfilling all proposed objectives. It was also possible to identify limitations in the architecture that could be addressed in future works.

Keywords: RESTful API, Relational Database, Military, Emergency, Mobile Application, NFC.

SUMÁRIO

1 INTRODUÇÃO **1**
1.1 Motivação 1
1.2 Objetivos 1
1.2.1 Objetivo Geral 1
1.2.2 Objetivos Específicos 1
1.3 Estrutura Documental 2

2 REVISÃO BIBLIOGRÁFICA **3**

3 FUNDAMENTAÇÃO TEÓRICA **4**
3.1 Modelagem de Dados 4
3.2 SQL 5
3.3 HTTP 6
3.3.1 *Status* da Mensagem 7
3.3.2 Métodos 7
3.4 API RESTful 7
3.4.1 API 8
3.4.2 REST 8
3.5 CID 9
3.6 NFC 9
3.7 Aplicações Móveis 10
3.8 Open Source 11
3.9 Erwin Data Modeller 11
3.10 PostgreSQL 11
3.11 ChatGPT 12
3.12 SwaggerHub 12
3.13 WSO2 Micro Integrator 13
3.14 AWS 13
3.15 Postman 14
3.16 React Native 14
3.17 JavaScript 14
3.18 Android Studio 15
3.19 Firebase 15

4 ARQUITETURA PROPOSTA **16**
4.1 Metodologia 16
4.2 Configuração e Desenvolvimento da Arquitetura 16
4.2.1 Definição do DER 17

4.2.2 Criação do Banco de Dados ... 19
4.2.3 Definição da API ... 20
4.2.4 Desenvolvimento e Implementação da API ... 22
4.2.5 Transposição do Ambiente Local para a Nuvem ... 26
4.2.6 Aplicativo Móvel ... 28

5 TESTES E RESULTADOS ... 34
5.1 Metodologia ... 34
5.2 Resultados Obtidos ... 37
5.2.1 Recurso Militar ... 37
5.2.2 Recurso Medicamento ... 40
5.2.3 Recurso Comorbidade ... 43
5.2.4 Análise dos Resultados Obtidos ... 45

6 CONCLUSÃO ... 47

REFERÊNCIAS BIBLIOGRÁFICAS ... 49

LISTA DE FIGURAS

4.1 Arquitetura final proposta. Fonte: autores ... 17
4.2 Diagrama Entidade Relacionamento da arquitetura proposta. Fonte: autores ... 19
4.3 *Script* SQL no *PgAdmin* 4. Fonte: autores ... 20
4.4 Criação de um novo *Integration Project*. Fonte: autores ... 22
4.5 *Script* da regra de negócio. Fonte: autores ... 23
4.6 Configuração das credenciais de acesso ao banco de dados. Fonte: autores ... 24
4.7 Configuração do recurso /Militar *POST*, disponibilizados pela API. Fonte: autores ... 24
4.8 Configuração das *querys* do recurso /Militar *POST*. Fonte: autores ... 25
4.9 Configuração do *endpoint*. Fonte: autores ... 25
4.10 Telas de login, cadastro e registro de usuário. Fonte: autores ... 28
4.11 Layout da tela inicial da aplicação. Fonte: autores ... 29
4.12 Telas da aplicação de cadastro de medicamento e cadastro de comorbidade. Fonte: autores ... 30
4.13 Telas de leitura da etiqueta NFC. Fonte: autores ... 31
4.14 Exemplo de etiquetas NFC. Fonte: autores ... 31
4.15 Sutache adotado pelo Corpo de Bombeiros Militar do Distrito Federal. Fonte: Regulamento de Uniformes CBMDF ... 32
4.16 Telas contendo os campos de perfil do usuário e dados do paciente. Fonte: autores ... 32
4.17 Diagrama de Caso de Uso do Aplicativo. Fonte: autores ... 33

5.1 Execução da API no *WSO2 Integration Studio*. Fonte: autores ... 34
5.2 Console de execução da API no *WSO2 Micro Integrator*. Fonte: autores ... 35
5.3 Coleção *Postman* para testes da API. Fonte: autores ... 36
5.4 Resultado do teste do método *GET* /Militar/{cpf}. Fonte: autores ... 37
5.5 Resultado do teste do método *POST* /Militar. Fonte: autores ... 38
5.6 Resultado do teste do método *PUT* /Militar. Fonte: autores ... 39
5.7 Resultado do teste do método *DELETE* /Militar/{cpf}. Fonte: autores ... 39
5.8 Resultado do teste do método *GET* /Medicamento/{cpf}. Fonte: autores ... 40
5.9 Resultado do teste do método *POST* /Medicamento. Fonte: autores ... 41
5.10 Resultado do teste do método *PUT* /Medicamento. Fonte: autores ... 41
5.11 Resultado do teste do método *DELETE* /Medicamento/{cpf}/{gtin}. Fonte: autores ... 42
5.12 Resultado do teste do método *GET* /Medicamento/Registro/{gtin}. Fonte: autores ... 42
5.13 Resultado do teste do método *GET* /Comorbidade/{cpf}. Fonte: autores ... 43
5.14 Resultado do teste do método *POST* /Comorbidade. Fonte: autores ... 44
5.15 Resultado do teste do método *PUT* /Comorbidade. Fonte: autores ... 44
5.16 Resultado do teste do método *DELETE* /Comorbidade/{cpf}/{cid}. Fonte: autores ... 45
5.17 Resultado do teste do método *GET* /Comorbidade/Registro/{cid}. Fonte: autores ... 45

LISTA DE ABREVIATURAS E SÍMBOLOS

Siglas

ANSI	*American National Standards Institute*
API	*Application Programming Interface*
ASCII	*American Standard Code for Information Interchange*
AWS	*Amazon Web Services*
BMW	*Bayerische Motoren Werke*
CID	Classificação Internacional de Doenças
CPF	Cadastro de Pessoas Físicas
CPU	*Central Processing Unit*
CRUD	*Create, Read, Update and Delete*
CSV	*Comma-separated Values*
DDL	*Data Definition Language*
DER	Diagrama Entidade Relacionamento
DML	*Data Manipulation Language*
DQL	*Data Query Language*
EC2	*Elastic Compute Cloud*
GTIN	*Global Trade Item Number*
HATEOAS	*Hypermedia As The Engine Of Application State*
HTML	*HyperText Markup Language*
HTTP	*HyperText transfer protocol*
HTTPS	*HyperText Transfer Protocol Secure*
IBM	*International Business Machines*
IDE	*Integrated Development Environment*
IEC	*International Electrotechnical Commission*
IP	*Internet Protocol*
ISO	*International Organization for Standardization*
JDBC	*Java Database Connectivity*
JSON	*JavaScript Object Notation*
MI	*Micro Integrator*
NFC	*Near Field Communication*
NoSQL	*Not Only SQL*
OMS	Organização Mundial da Saúde
RAM	*Random Access Memory*
REST	*Representational State Transfer*
RFC	*Request for Comments*
RLHF	*Reinforcement Learning from Human Feedback*
RPC	*Remote Procedure Call*

SCP	*Secure Copy*
SGBD	Sistema de Gerenciamento de Banco de Dados
SMTP	*Simple Mail Transfer Protocol*
SO	Sistema Operacional
SOAP	*Simple Object Access Protocol*
SQL	*Structured Query Language*
SSH	*Secure Shell*
STEM	*Science, Technology, Engineering and Mathematics*
TCP	*Transport Control Protocol*
URI	*Uniform Resource Identifier*
UDP	*User Datagram Protocol*
URL	*Uniform Resource Locator*
XML	*Extensible Markup Language*
YAML	*Yet Another Markup Language*

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

De acordo com [1], o uso de tecnologias da informação pode influenciar positivamente na segurança do paciente por meio da redução de erros médicos, além da redução de reações adversas a medicamentos. Em um contexto de atendimento de emergência, o impacto dessas tecnologias pode ser ainda maior, por se tratar de uma situação que envolve maiores riscos e variáveis, principalmente o curto período de reação necessário, por parte dos socorristas.

Nesse cenário, o NFC se mostra como uma valiosa ferramenta, devido ao seu uso extenso na indústria, principalmente em *smartphones*, o que garante não só crescimento, mas também longevidade à tecnologia, conforme mostrado por [2]. Além disso, se mostra mais segura que outras tecnologias de curto alcance como o RFID e o *Bluetooth*.

Portanto, o desenvolvimento de uma solução, utilizando o NFC, pode otimizar o atendimento de emergência, aumentando a segurança do paciente.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Este trabalho tem como objetivo geral a implementação de um sistema que permita a consulta de dados relacionados à saúde de uma pessoa, inserida em um contexto de organização militar.

1.2.2 Objetivos Específicos

Os objetivos específicos promovem sustentação ao objetivo geral anteriormente proposto. Os tópicos descritos a seguir tem como finalidade a implementação de uma arquitetura para consulta de dados de saúde de um indivíduo a partir da leitura de uma etiqueta NFC.

- Projetar um modelo de dados que supra a necessidade de armazenamento dos dados dos indivíduos militares, bem como sua implementação em um banco de dados relacional;
- Especificar e implementar uma *API Restful* que permita o acesso do usuário da aplicação ao banco de dados, possibilitando a consulta, o envio, a atualização e a exclusão dessas informações;
- Integrar os componentes citados nos tópicos anteriores em uma solução em nuvem;
- Desenvolver uma aplicação que proporcione o consumo da *API Restful* e a exibição dos dados de um indivíduo, por meio da leitura de uma etiqueta NFC.

1.3 ESTRUTURA DOCUMENTAL

Esta seção tem como finalidade especificar a estrutura desse documento, que é segmentada em 6 capítulos, como descrito a seguir:

- O primeiro capítulo se refere à introdução, formada pelo detalhamento das motivações e objetivos do trabalho;
- O segundo capítulo faz referência à revisão bibliográfica, destacando trabalhos relacionados à temática abordada neste projeto;
- O terceiro capítulo é dedicado à fundamentação teórica, no qual são referenciados os conceitos e tecnologias estudados e utilizados no desenvolvimento do projeto. Somado a isso, são apresentadas as ferramentas utilizadas, detalhando as suas respectivas aplicações no escopo de desenvolvimento do projeto;
- O quarto capítulo apresenta a arquitetura proposta, exibindo a metodologia utilizada, bem como o detalhamento de cada etapa de construção da arquitetura;
- O quinto capítulo aborda os testes realizados para averiguação do funcionamento da arquitetura, assim como os seus respectivos resultados e a discussão acerca deles;
- O sexto capítulo tem como objetivo apresentar um fechamento do trabalho, abordando o cumprimento dos objetivos propostos, os resultados obtidos, as limitações encontradas na arquitetura e as propostas para trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Tópicos como *Near Field Communication* (NFC) e Aplicativos Móveis de Saúde (mHealth) têm ganhado cada vez mais relevância no âmbito da pesquisa acadêmica, pois oferecem oportunidades promissoras para melhorar a prestação de serviços de saúde e o cuidado aos pacientes.

O trabalho de [3] trata de uma solução para a área da saúde, que utiliza tecnologia NFC para criar um dispositivo de baixo custo e vestível que mede os batimentos cardíacos e a temperatura utilizando sensor sem fio e sem bateria, além de um aplicativo para *smartphones*. Em contrapartida, o objetivo deste trabalho é desenvolver uma solução para atendimentos de emergência, possibilitando que, por meio de um dispositivo móvel, seja feita a leitura de uma tag NFC, que apresente informações do usuário fundamentais para que o atendimento possa ser realizado com os devidos cuidados.

Em [4] é desenvolvida uma pesquisa que realiza a análise de vulnerabilidades da tecnologia NFC em diversos cenários e aborda diferentes ataques de segurança. Além disso, são apresentadas maneiras de se defender desses ataques, promovendo segurança para as soluções. Este trabalho apresenta uma forma de acessar informações sensíveis de usuários por meio de NFC. Sendo assim, a segurança é fundamental para essa solução, no entanto ela está vinculada à autenticação utilizada no aplicativo e aos protocolos utilizados para realizar a comunicação entre o aplicativo e o banco de dados.

Outro artigo que apresenta soluções de Telemedicina envolvendo NFC é o [5]. Seu objetivo é utilizar essa tecnologia para ajudar a identificar pacientes em hospitais, evitando erros de diagnósticos. Assim, é feita uma solução de gerenciamento de cuidados com a saúde do paciente que une NFC, tecnologias *Web* e *smartphones*. O foco deste projeto é diferente do prosposto nesse documento, pois não é direcionado a atendimentos de emergência, apesar de ambos utilizarem tecnologias móveis e NFC, para identificação dos usuários e levantamento de suas informações médicas.

O [6] desenvolve um estudo com o intuito de avaliar a situação dos serviços de pagamentos eletrônicos usando NFC, além de realizar uma pesquisa com usuários para verificar a intenção de continuar utilizando esse método de pagamento. Esse trabalho apresentou uma opinião positiva dos participantes da pesquisa, apontando um intuito de continuarem com o mesma forma pagamento. Com isso, pode-se observar um posicionamento favorável à utilização de tecnologia NFC e aplicativos *mobile*. Assim, a arquitetura apresentada neste documento utiliza a mesma tecnologia, porém voltada ao campo da saúde, proporcionando para os socorristas informações importantes para a realização eficiente do atendimento.

Em [7] apresenta uma pesquisa sobre adoção de tecnologias da informação em serviços de atendimento móvel de emergência. Esse trabalho trata dos principais motivos que devem ser atendidos no desnvolvimento de soluções tecnológicas para área da saúde, como a praticidade, além de elencar a importância de agregar tecnologia na forma como as práticas assistenciais são conduzidas. No entanto, esse trabalho não desenvolve uma solução *mobile* envolvendo etiquetas NFC para identificação e leitura dos dados do usuário.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo objetiva apresentar uma fundamentação teórica das tecnologias empregadas no projeto, sendo abordados conceitos, termos e teorias adjacentes essenciais para a sequência do trabalho. Além disso, serão expostas as ferramentas utilizadas na construção da arquitetura proposta, com suas respectivas funcionalidades e aplicações.

3.1 MODELAGEM DE DADOS

De acordo com [8], um modelo de dados consiste em uma descrição formal da estrutura de um banco de dados, ou seja, das informações ali armazenadas. Nesse contexto, podem existir diferentes abordagens para a execução da modelagem, de acordo com o objetivo e o público para o qual o modelo está direcionado.

Dessa forma, pode-se abstrair um modelo de dados em 3 níveis:

- **Conceitual:** Constitui o nível mais alto de abstração, por apenas descrever o banco de dados, mas sem se aprofundar na forma na qual os dados são armazenados. Por esse motivo, independe do SGDB utilizado, o que permite uma certa versatilidade. A técnica mais utilizada nesse modelo é a de entidade-relacionamento, que será descrita ainda nessa seção.
- **Lógico:** Aproxima-se do modelo visto em um SGBD, portanto difere-se de acordo com o sistema utilizado. Nesse modelo, a representação dos componentes se torna mais objetiva, com as entidades se tornando tabelas e atributos se tornando colunas.
- **Físico:** Trata-se do modelo de implementação de dados, utilizando, portanto, uma linguagem específica. Geralmente a linguagem utilizada é a SQL, associada a algum SGBD.

Conforme descrito por [9], um modelo entidade-relacionamento é composto por entidades, suas repectivas relações entre si e os atributos, que identificam uma entidade como única. A sua representação gráfica, chamada de DER, possui formas convencionadas para a representação de cada um dos componentes:

- **Retângulo:** Responsável por representar as entidades.
- **Elipse:** Representa os atributos de uma entidade.
- **Losango:** Retrata os relacionamentos entre as entidades.
- **Linha:** Simboliza o vínculo entre os demais componentes.

Segundo [10], "Uma entidade é uma ’coisa’ que pode ser identificada distintivamente. Uma pessoa, companhia, ou evento é um exemplo de entidade". Portanto, entidade é algo de possui características

próprias que a difere das outras. Essas entidades podem ser divididas entre fortes, fracas e associativas. A primeira é a mais comum e independe de qualquer outra para existir; a fraca tem sua existência justificada por outra entidade forte, o que gera uma relação de dependência. Já a associativa representa a relação de duas ou mais entidades, dessa maneira, geralmente está associada a uma ação, conforme mostrado por [11].

O atributo de uma entidade é o que a caracteriza, ou seja, uma propriedade dela. Por exemplo, um cidadão brasileiro que possua um CPF é caracterizado por esse código, o que o identifica para o Governo Federal. Caso um atributo identifique uma instância como única perante as outras de uma mesma entidade, ele é chamado de atributo identificador, como o caso exemplificado acima.

No modelo relacional, o atributo identificador é chamado de chave, e é responsável pela conexão entre tabelas que se relacionam. Ainda, pode ser dividida em 3 tipos:

- **Primária:** É definida como a coluna que identifica um registro em uma tabela como único, ou seja, difere um registro dos demais e, portanto, não pode se repetir em uma mesma tabela. Também pode ser utilizada a combinação de duas ou mais colunas, sendo designada como uma chave composta.
- **Alternativa:** Caso haja mais de uma coluna candidata à chave primária em uma mesma tabela, as colunas não escolhidas se tornam chaves alternativas.
- **Estrangeira:** Assim como a chave primária, também pode ser criada com a combinação de uma ou mais colunas. Seus valores correspondem à chave primária de outra tabela, a qual esta está relacionada. Portanto é imprescindível para o estabelecimento das relações entre as tabelas.

Por fim, um relacionamento pode ser definido como a associação entre duas ou mais entidades, o que pode incluir ela mesma, caracterizando um auto-relacionamento, de acordo com [8]. A cardinalidade é uma propriedade importante de um relacionamento, pois define a quantidade de ocorrências, máxima e mínima, que uma entidade pode ter com outra. As cardinalidades podem ser dos tipos muitos-para-muitos (n:n), um-para-muitos (1:n) e um-para-um (1:1). Por exemplo, na relação entre pai e filho, um pai pode ter n filhos, mas um filho tem apenas um pai, portanto essa relação tem cardinalidade (1:n).

3.2 SQL

A SQL, ou *Structured Query Language*, é a linguagem padrão para a operação de banco de dados relacionais [12]. Inicialmente foi derivada do projeto R, promovido pela IBM no início da década de 1970, e implementada no protótipo *System R*, da mesma empresa. Passou por diversas mudanças e atualizações até ser oficializada em 1986 como um padrão ANSI e ISO, chamado de SQL-86. Atualmente, ainda sofre mudanças e atualizações, com a sua última versão, a ISO/IEC 9075-1:2023, publicada em junho de 2023.

Conforme exposto por [9], a SQL fornece dois tipos de linguagem básicas:

- **Linguagem de Definição de Dados - DDL:** Representa o conjunto de comandos utilizados na definição dos esquemas e estrutura dos dados, que realiza ações como excluir, criar e alterar esquemas.

Alguns dos comando utilizados são *CREATE*, *DROP* e *ALTER*.

- **Linguagem de Manipulação de Dados - DML:** Reúne os comandos que permitem a manipulação dos dados, realizando ações parecidas com o DDL, porém no escopo dos dados propriamente ditos, e não a estrutura em que se encontram inseridos. Além disso, os autores também adicionam a capacidade de consulta aos dados nessa categoria, o que adiciona o comando *SELECT* a ela. Outras literaturas preferem separá-la em um conjunto próprio, chamado de DQL (*Data Query Language*).

O livro também aborda os diferentes domínios de dados aceitos pela SQL. Alguns são básicos como *char*, *varchar* e *int*, enquanto outros dados internos também estão disponíveis como *date*, *time* e *timestamp*. Vale ressaltar que esses domínios se referem ao padrão SQL, com os diversos SGBDs do mercado oferecendo diversos outros domínios para necessidades específicas, como imagem, documento, monetário e coordenadas geográficas.

Ainda, a SQL possui uma estrutura básica para a consulta aos dados. Os comandos são executados pelo banco de dados conforma a sua sequência de envio. Algumas cláusulas são importantes para a execução da tarefa:

- ***SELECT*:** Seleciona o que será exposto na visão da consulta, podendo ser informações do sistema, como horário e data atuais; dados de tabelas e expressões artiméticas.
- ***FROM*:** Define a origem dos dados selecionados para consulta. Caso os dados estejam em tabelas diferentes, é possível usar a cláusula *JOIN*, que permite exibir esses dados de tabelas distintas em uma mesma visão.
- ***WHERE*:** Utilizada como filtro na consulta, selecionando apenas os dados que respeitem as condições estabelecidas por ela. Pode ser utilizada com conectivos como *and*, *or* e *not*, assim como operadores lógicos de menor, maior e igual.

A linguagem oferece muitos mais recursos que não serão expostos nesse documento por serem irrelevantes ao desenvolvimento do projeto.

3.3 HTTP

O *Hypertext Transfer Protocol* é defino por [13] como o protocolo utilizado na camada de aplicação da Web, que faz uso de uma estrutura cliente/servidor, ou seja, dois sistemas finais distintos se comunicando através de mensagens. Segundo os autores, o HTTP define o modo como o cliente e o servidor se comunicam, como as regras de requisições e respostas. Quando uma página *Web* é solicitada, a aplicação do cliente envia mensagens de requisição HTTP dos objetos e, então, o servidor recebe essas requisições e envia as respostas HTTP contendo os objetos solicitados.

Uma característica relevante do HTTP é a utilização do TCP como protocolo de transporte, ao invés do UDP. Ainda, vale ressaltar que se trata de um protocolo sem estado, ou seja, o servido servidor não retém nenhuma informação do cliente.

Seu formato segue a seguinte estrutura, conforme exposto em [14]:

- ***Start Line*:** Trata-se da primeira linha da mensagem, escrita em ASCII comum. É composta por 4 campos, sendo eles o método, a URL, a versão do protocolo e o *status* da mensagem. Em uma mensagem de requisição, tem a função de comunicar ao servidor o que está sendo solicitado, enquanto que, em uma mensagem de resposta, carrega informação sobre a operação realizada, utilizando o campo de *status*.
- **Cabeçalho:** Adiciona informações adicionais à mensagem, em formato de pares de campos e seus respectivos valores. São informações como data, comprimento e tipo do conteúdo, e os formatos aceitos pelo cliente. Os dados contidos no cabeçalho podem sofrer variações de acordo com o tipo de mensagem em questão. Detalhes podem ver vistos na RFC 2616 seção 4.2.
- **Corpo:** Compõe a parte da mensagem responsável pelo envio de dados no HTTP. Trata-se de uma parte opcional, visto que algumas mensagens não possuem um *payload* de dados a ser enviado. Pode conter diversos tipos de dados como imagens, vídeos, HTML e documentos em diversos formatos.

3.3.1 *Status* da Mensagem

Em uma requisição HTTP muitas questões podem ocorrer durante a comunicação, como falhas no servidor ou no próprio cliente. Para permitir que o cliente tenha consciência dessas ocorrências, existe o *status* da mensagem, que é um código numérico de 3 dígitos que indica o resultado da operação solicitada ao servidor. De acordo, com o intervalo do código, pode ser indicado sucesso, erro no servidor ou até mesmo erro no cliente. A lista de códigos é definida pela RFC 2616 seção 10.

3.3.2 Métodos

Os métodos informam ao servidor qual tarefa deve ser executada, com cada um definindo uma tarefa específica, conforme previsto pela RFC 2616 seção 9. Por exemplo, o método *GET* solicita dados, enquanto o *POST* envia dados para serem processados pelo servidor.

O aprofundamento de quaisquer conceitos abordados nessa seção pode ser realizado por meio da RFC 2616 que define o protocolo HTTP em sua versão 1.1.

3.4 API RESTFUL

Segundo [15], trata-se de uma API construída a partir do estilo de arquitetura REST, que será descrito em uma seção posterior. Ela faz uso de uma infraestrutura baseada em HTTP, protocolo detalhado na seção 3.3, utilizando seus métodos para operações CRUD, em uma interface de recursos uniforme. Também fazem uso de *Uniform Resource Identifiers* (URIs) para disponibilizar o endereço desses recursos [16].

Nas seções 3.4.1 e 3.4.2 serão aprofundados todos os conceitos pertinentes ao projeto proposto e à melhor compreensão das informações acima citadas.

3.4.1 API

Chamado de *Application Programming Interface*, [15] cita que "uma API oferece um simples caminho para se conectar, integrar com e extender um sistema de software". O autor também diz que, geralmente, não são visíveis ao usuário final, operando diretamente em comunicação entre máquinas, integrando dois ou mais sistemas de *software*. Dessa forma os desenvolvedores precisam levar em consideração a perspectiva de quem utilizará a API diretamente, que não é o usuário final.

No cenário atual, o mercado e a indústria operam diferentes *softwares* em seus sistemas. Tipicamente, esses programas trabalham de forma isolada, não permitindo seu acesso a partir de outros *softwares*. Nesse contexto, uma API pode retirar essa deficiência, possibilitando que esses sistemas únicos e isolados tornem-se acessíveis uns aos outros, tanto de forma interna, em distintos setores dentro do próprio negócio, como externalizado, entre diferentes empresas.

Ainda, uma API pode ser construída conforme quatro arquiteturas, como mostrado por [15], sendo elas a REST, a HATEOAS, a RPC e a SOAP. Como objeto de estudo será detalhada a arquitetura REST, utilizada neste projeto.

3.4.2 REST

A arquitetura REST (*Representational State Transfer*) foi proposta por Roy Fielding [17], utilizando conceitos de arquiteturas baseadas em rede aliados a uma inteface de conector uniforme. O autor define as restrições da arquitetura em questão:

- **Cliente-Servidor:** Utiliza o conceito da arquitetura cliente-servidor, separarando as questões relacionadas à interface de usuário e ao armazenamento de dados. Assim, promove a portabilidade e escalabilidade da solução, ao diminuir a complexidade no contexto do servidor. Dessa forma, cada componente da arquitetura consegue evoluir de forma independente.

- ***Stateless*:** A comunicação não deve possuir estado definido, com cada requisição feita a partir do cliente contendo toda a informação necessária para que o servidor execute a tarefa, com o estado da sessão sendo mantido exclusivamente pelo cliente. Essa característica promove a visibilidade, confiabilidade e escalabilidade da solução.

- ***Cache*:** Define que toda resposta a uma requisição indique ser *cacheable* ou *non-cacheble*. Dessa forma, o cliente pode usar ou não os dados de uma resposta em um requisição futura, eliminando a necessidade de alguma interações, o que melhora a performance da rede.

- **Interface Uniforme:** Trata-se da característica que a diferencia de outras arquiteturas, definindo uma interface uniforme entre seus componentes. As implementações são desacopladas de seus respectivos serviços, o que permite uma evolução independente de cada componente. Porém, isso prejudica a sua eficiência, pelo fato dos dados serem padronizados, ao invés de específicos para a necessidade de cada aplicação. Também são definidas 4 regras para a interface: identificação de recursos, manipulação de recursos por meio de representações, mensagens auto-descritíveis e *hypermedia* como o motor de estado da aplicação.

- **Sistema em camadas:** Permite que a arquitetura utilize camadas hierárquicas em sua composição, com cada componente "enxergando"apenas a camada imediata a ele. Dessa forma, é possível diminuir a complexidade do sistema, trazendo independência aos componentes e promovendo escalabilidade.
- **Código sob Demanda:** Tem o objetivo de simplificar o cliente, sendo a única condição opcional da arquitetura, ao permitir o *download* e execução de *applets* e *scripts*. Isso promove uma maior extensibilidade ao sistema, mas reduz a visibilidade.

Vale salientar que REST não se trata de um protocolo de comunicação, e sim de um estilo de arquitetura de software com restrições propriamente definidas. Quando aplicada à uma solução, como uma API, tem-se uma solução *RESTful*.

3.5 CID

Criado pela OMS, a Classificação Internacional de Doenças fornece conhecimento a cerca de doenças humanas, como suas respectivas causas e consequências [18]. Por meio de uma base de dados mundiais de saúde, oferece também dados estatísticos acerca de doenças em cuidados primários, secundários e terciários, assim como em caso de morte. Esses dados são utilizados por diversos serviços, como administração de qualidade e segurança e pesquisa na área da saúde.

Desde a sua criação, ainda no século XIX, a CID é atualizada e lançada em versões, com a última delas, a versão 11, disponibilizada em janeiro de 2022.

O CID-11 é formado por códigos alfanuméricos que vão de 1A00.00 até ZZ9Z.ZZ. Sendo um código STEM, sua estrutura é definida da seguinte forma:

- O primeiro caracter define o número do capítulo a qual a doença está inserida.
- O segundo caracter é sempre uma letra, diferenciando o CID-11 de sua versão anterior.
- As letras ’O’ e ’I’ são omitidas para não serem confundidos com os números ’0’ e ’1’.

De acordo a OMS, a estrutura de capítulos segue o modelo proposto por William Farr, contendo doenças epidêmicas, doenças gerais, doenças locais organizadas por local, doenças de desenvolvimento e lesões. Adicionado a isso existem grupos especiais, que aglutinam condições que, separadas, prejudicariam estudos epidemiológicos.

3.6 NFC

Segundo [19] o *Near Field Communication* (NFC) é uma tecnologia de transmissão de dados sem fio de curta distância. Esta tecnologia está muito difundida na Europa e é uma das tecnologias mais inovadoras

da atualidade. Visando aplicações que demandem troca de informações rápidas e imediata, o NFC pode encontrar uma maneira de ser empregado.

Outro ponto importante é que, para se utilizar a tecnologia NFC, é necessário que os dois dispositivos possuam essa tecnologia, sendo eles o emissor e o receptor. O emissor é o dispositivo que realiza a primeira comunicação e envia os dados, como um *smartphone*, *tablet* ou *smartwatch*. Esses equipamentos enviam dados por meio de ondas de curto alcance quando estão próximos de um receptor. Em contrapartida, o receptor é responsável por receber as informações enviadas do emissor. Por fim, ao receber as informações pode responder e executar diversas ações com base nos dados recebidos.

De acordo com [20], as grandes fabricantes de *smartphones*, como Apple, Samsung e Google, incorporaram NFC em seus telefones celulares com o intuito de eliminar a necessidade de cartões de pagamento. Essas tecnologias são apontadas como seguras, pois são de curto alcance e não necessitam de Wi-Fi ou redes móveis. O uso dessa tecnologia vem crescendo de maneira significativa e, com isso, preocupações são levantadas pelos especialistas em segurança, pois não exigem uma autenticação robusta por parte dos utilizadores.

Em [21] são apresentados os ataques mais conhecidos sobre transações bancarias NFC e, após esse passo, são discutidas as soluções mais utilizadas recentemente para proteger essas transações. Por pagamentos se tratarem de dados sensiveis, é importante que sejam tomadas medidas de segurança que preservem essas informações.

Outra área que vem ganhando diversas soluções envolvendo NFC é a da saúde, como pode ser visto em [5]. O artigo utiliza essa tecnologia para identificação de paciente em ambiente hospitalar, com o foco de não cometer erros ao fazer o diagnóstico do paciente e evitar falha dos funcionários.

Dessa forma, o NFC se mostra como a mais adequada ferramenta para utilização no projeto em comparação à outras tecnologias como o *Bluetooth* e o RFID, devido a se mostrar como mais segura e com perspectivas de crescimento e ploriferação do seu uso na indústria [2].

3.7 APLICAÇÕES MÓVEIS

Aplicações Móveis são *softwares* desenvolvidos para serem instalados e utilizados em dispositivos móveis. O intuito dessas soluções é disponibilizar funcionalidades de maneira intuitiva, prática e rápida, além de poder ser transportado com o usuário.

De acordo com [22], os aplicativos de saúde vêm aumentando os números de usuários e sendo cada vez mais procurados. Isso também se intensificou devido a pandemia de COVID-19, que fez com que as pessoas buscassem maneiras de ter acesso a saúde sem precisar sair de casa. Nesse cenário os apps *mHealth* têm grande vantagem em relação à conveniência e aos acessos em diferentes localidades de maneira remota, além de reduzir os custos e tornar a saúde mais acessível para todos.

Um exemplo de aplicativo é o do [23], trabalho no qual é desenvolvido um aplicativo *mobile* Android para diabetes *mellitus* tipo 2 que realizam integrações com *hardware* de suporte, os quais fazem a medição e monitoramento de variáveis relacionadas aos pacientes. Essa é uma aplicação que utiliza o celular para

receber as informações, tratar os dados e armazenar na nuvem.

3.8 OPEN SOURCE

Open Source ou *Software* de código aberto é um *software* de computador que tem o código fonte disponibilizado e licenciado. Eles são disponibilizados de maneira gratuita para qualquer pessoa acessar, copiar, modificar e redistribuir, permitindo uma abordagem colaborativa.

Em [24] é mostrado uma solução automatizada de administração de insulina de código aberto, que apresenta um trabalho criado pela comunidade *online* o qual foram afetados pelas diabetes. Um importante indicador do sucesso dessa pesquisa é a sua aceitação global que vem crecendo.

Ao se tratar dessas tecnologias, outro ponto muito importante é a redução de custos. Isso pode ser observado no trabalho de [25], que fala sobre a importancia dos *softwares* de código aberto e da redução de custo gerado por essas tecnologias nesse trabalho voltado para a área de impressão 3D, no campo das separações químicas.

Por fim, outro benefício dessas tecnologias é com relação à segurança. Devido a terem acesso ao código, é possivel que especialistas em segurança encontrem vulnerabilidades e que essas falhas sejam corrigidas pela comunidade.

3.9 ERWIN DATA MODELLER

O Erwin Data Modeller [26] é uma solução em *software* criada para atuar na modelagem de sistemas e dados. Muito utilizada no meio empresarial, permite visualizar, projetar e implementar dados de forma padronizada e concistente.

Em sua versão padrão, que foi utilizada no projeto, é possível utilizar modelos pré-definidos, padrões de dados e linguagens próprias de diversos SGBD diferentes, como MySQL e PostgreSQL.

Sua utilização nesse projeto está diretamente relacionada à criação do modelo de dados relacional necessário à arquitetura, que será exposta posteriormente nesse documento. Isso inclui a criação dos modelos lógico e físico, com suas entidades, relacionamentos e atributos; a definição dos tipos de dados utilizados; e a criação do DER, anteriormente definido na seção 4.2.1.

3.10 POSTGRESQL

Como visto na seção 3.2, apesar da linguagem SQL ser identificada por padrões ISO e ANSI, diversas empresas trabalharam no desenvolvimento de suas próprias versões, baseando-se nessas definições oficiais.

Nesse contexto, surgiram ferramentas como o MySQL e o PostgreSQL, de código aberto, e programas proprietários como o SQLServer e o Oracle Database. Apesar de todas utilizarem o SQL como linguagem

padrão, cada uma possui uma linguagem adicional que adiciona funcionalidades próprias, para facilitar a solução de problemas e questões que demandavam uma notória complexidade quando utilizava-se apenas a linguaguem SQL, como laços de repetição e criação de funções.

O PostgreSQL [27], derivado de um projeto da universidade de *Bankley* chamado POSTGRES, é um SGBD objeto-relacional de código aberto, ou seja, executa tanto as funções de um SGBD relacional padrão, como também adiciona características de orientação a objetos. Em suas versões mais recentes possui recursos como funções armazenadas em diversas linguagens de programação, conexão SSL e extenções para diversos tipos de arquivo, com XML e dados geoespaciais.

Foi a ferramenta selecionada por ser de código aberto e possuir um importante suporte tanto pela própria *PostgreSQL Global Development Group*, quanto pela comunidade, o que facilita o seu acesso e uso no desenvolvimento da arquitetura. No projeto, foi usada para a criação e administração do banco de dados que armazena as informações pertinentes à arquitetura, como mostrado na seção .

3.11 CHATGPT

Trata-se de um *chatbot* [28], desenvolvido pela *OpenAI*, treinado para seguir instruções e fornecer respostas detalhadas para o usuário. Seu treinamento é realizado por meio de RLHF, *Reinforcement Learning from Human Feedback*, que utiliza um modelo de recompensa calibrado por respostas humanas.

Foi utilizado na geração dos dados fictícios necessários para a realização dos testes da arquitetura, como detalhado na seção 4.2.2.3 desse documento.

3.12 SWAGGERHUB

Desenvolvido para auxiliar no desenvolvimento em conjunto de APIs, o *SwaggerHub* [29] reúne duas ferramentas importantes, o *Swagger Editor* e o *Swagger UI*. O primeiro permite desenvolver, descrever e documentar uma API em diversos formato, o que inclui o *OpenAPI 3*, que foi o formato escolhido para esse projeto. Algumas de suas características facilitam o processo de desenvolvimento:

- **Execução em diferentes ambientes:** Permite trabalhar tanto em ambiente local como em ambiente WEB.
- ***Feedback* inteligente:** Valida a sintaxe do código escrito em tempo real, permitindo rápidas correções.
- **Visualização instantânea:** Renderiza a especificação da API de forma visual, permitindo interações imediatas.
- **Preenchimento Automático:** Completa a sintaxe do código de forma automatiza, tornando a programação mais rápida e fluída.

O *Swagger UI*, possibilita a visualização da API, com todos os recursos e definições de forma prática e visual, sem a necessidade de acessar o código fonte. Isso facilita o desenvolvimento dos outros conjuntos da arquitetura simultâneamente, agilizando o avanço do projeto.

Na seção 4.2.3, é possível observar os detalhes do uso dessa ferramenta no projeto, que contempla a criação da API que permite o consumo dos dados dos usuários por parte da aplicação.

3.13 WSO2 MICRO INTEGRATOR

O *WSO2 Micro Integrator* [30] é um *software* que permite a integração de APIs, serviços e dados de diferentes sistemas. Possui suporte a diversos protocolos como HTTP, HTTPS, SOAP e SMTP, além da transformação de dados em diferentes formatos como XML, JSON e CSV.

Inclui ainda, o *WSO2 Integration Studio*, uma IDE que permite trabalhar na implementação dos sistemas de forma visual, com o auxílio de uma plataforma gráfica, ou diretamente em código fonte. Como funciona em conjunto com o WSO2 MI, é possível executar testes de forma simples conforme o desenvolvimento da solução, ajustando possíveis problemas de forma prática e rápida. Além disso, permite a exportação do projeto final de diversas formas, como imagem *Docker*, *Kubernets* e aplicação *Carbon*, para ser aplicado em outros sistemas.

Como pode ser visto no capítulo 4, essa ferramenta foi utilizada tanto na implementação da API e sua integração com o banco de dados PostgreSQL, como também na execução da aplicação final no ambiente em nuvem.

3.14 AWS

A AWS, ou ainda *Amazon Web Services* [31], é uma plataforma de serviços de computação em nuvem, que fornece soluções como processamento, armazenamento, banco de dados, *machine learning*, entre outros. Seus servidores estão distribuídos pelo mundo inteiro, o que inclui América do Norte, Ásia, Europa e o único servidor da América do Sul, localizado no estado brasileiro de São Paulo. Ainda, seus serviços são utilizados por diversas empresas multinacionais como BWM, Coca-Cola e Netflix.

A oferta de recursos de processamento é feita por meio de instâncias EC2, que se diferem de acordo com a capacidade de processamento, armazenamento e memória, com essa capacidade sendo redimensionável a qualquer momento. O plano gratuito para computação fornece uma instância t2.micro contendo uma vCPU, 1 GB de memória RAM e até 20 GB de armazenamento, sendo esse o escolhido para a execução do projeto.

A instância EC2 foi utilizada para disponibilizar, de forma descentralizada, os recursos necessários ao projeto. Como pode ser observado no capítulo 4, tanto o WSO2 MI quanto o banco de dados PostgreSQL foram executados no mesmo ambiente da AWS.

3.15 POSTMAN

O *Postman* [32] é uma plataforma que permite a realização de testes e o desenvolvimento de APIs. Com o seu uso é possível simular requisições HTTP de forma visual e intuitiva, permitindo a configuração de cabeçalhos e *payloads* de forma gráfica, o que a torna uma ferramenta amigável ao usuário. Além disso, executa uma análise das respostas, exibindo as informações mais relevantes ao usuário, como o código, o corpo da mensagem e o atraso da resposta, tornando suas análises uma tarefa mais rápida e eficiente.

Dentre suas ferramentas, é importante destacar as coleções. Nada mais são do que um conjunto de requisições HTTP definidas pelo usuário, sendo possível criá-las de forma automática por meio da importação de uma definição *Swagger*. Assim, o *software* gera requisições para cada respectivo recurso, incluindo o cabeçalho e o corpo da mensagem. Ainda, para uma navegação mais adequada, todas essas requisições produzidas são organizadas hierarquicamente, conforme o contexto e o método HTTP.

Dessa forma, o Postman se mostra como uma ferramenta adequada para a execução de teste e análise dos resultados, para averiguar se a API supre as demandas exigidas pela arquitetura. O seu uso específico no projeto, bem como a análise dos resultados obtidos, serão explorados no capítulo 5.

3.16 REACT NATIVE

Como visto em [33], o *React Native* é um *framework open source* de desenvolvimento de aplicativos móveis que permite criar aplicativos nativos para iOS e *Android* utilizando o *JavaScript* e a biblioteca *React*. Foi desenvolvido pelo *Facebook*, além de ser uma extensão do *framework React*, que é usado para criar interfaces de usuário em aplicações *Web*.

A principal vantagem da sua utilização é desenvolver aplicativos móveis usando uma única base de código em *JavaScript*. Isso possibilita criar versões conjuntas para iOS e *Android*, pois há a possibilidade de utilizar grande parte do código para as duas plataformas. Outro ponto importante desse *framework* é a utilização de componentes nativos do sistema operacional, oferecendo uma experiência de usuário (UX) e desempenho superior a aplicações de desenvolvimento híbrido.

Dessa maneira, o React Native mostra ser uma ferramenta fundamental para o desenvolvimento do projeto, pois possibilitou a criação de um aplicativo móvel, usando componentes nativos. Além disso, ofereceu uma abordagem eficiente e robusta para aplicações móveis multiplataforma.

3.17 JAVASCRIPT

Segundo [34], o *JavaScript* é uma linguagem de programação versátil e robusta, utilizada como geradora de interatividade para páginas *Web*, para desenvolvimento de aplicativos *Web* completos e em servidores por meio de ambientes, como o Node.js. É uma linguagem de programação fundamental, que possui uma quantidade significativa de bibliotecas e frameworks disponíveis para facilitar o desenvolvimento de soluções complexas.

Essa linguagem de programação é usada tanto para escrever códigos no *framework React Native*, quanto para desenvolver a lógica do aplicativo, manipular a interface do usuário e interagir com as APIs nativas dos aparelhos móveis. Ainda, é possivel manipular dados, executar operações assíncronas e utilizar bibliotecas.

Diante do apresentado, o *JavaScript* é a linguagem de programação utilizada para desenvolver o aplicativo, pois permite a utilização de recursos fundamentais dos dispositivos móveis. Isso possibilita o desenvolvimento de uma solução complexa, a utilização de recursos e bibliotecas importantes.

3.18 ANDROID STUDIO

O *Android Studio*, como visto em [35], é o principal ambiente de desenvolvimento integrado (IDE) para a criação de aplicativos *Android* de alta qualidade. Foi criado pela *Google* e oferece diversas ferramentas e recursos para auxiliar o desenvolvimento, os testes e a depuração de aplicativos. Entre suas ferramentas e recursos oferecidos estão: editor de código, emulador, *layout editor*, depurador e análise de desempenho.

Essa IDE é utilizada em projetos *React Native* e desempenha uma função importante no desenvolvimento de aplicativos que utilizam essa plataforma, principalmente ao se tratar de tarefas relacionadas à compilação, depuração, execução e otimização do código nativo. Logo, o *Android Studio* é uma ferramenta complementar valiosa para lidar com a parte nativa do aplicativo *Android*.

Essa ferramenta foi utilizada no projeto para a compilação, execução e otimização do código nativo desenvolvido na aplicação móvel, possibilitando uma aplicação *Android* de alta qualidade.

3.19 FIREBASE

O *Firebase*, para [36], é uma plataforma da *Google* de desenvolvimento de aplicativos móveis e aplicações *Web*. Essa ferramenta fornece diversos serviços, que auxiliam na criação, desenvolvimento e implementação de aplicativos de maneira rápida e eficiente.

Dentre os serviços oferecidos pela plataforma, o *Authentication* e o *Cloud Firestore* recebem destaque, pois foram utilizados no presente trabalho.

O *Firebase Authentication* é um serviço completo de autenticação de usuário. Ele fornece recursos de autenticação segura e confiável para os aplicativos, com suporte a diversos provedores de identidade, gerenciamento de usuário e integração com outras ferramentas da plataforma.

O *Cloud Firestore* é um banco de dados *NoSQL* (orientado a documentos) altamente escalável. Ele fornece um modelo de dados flexível, sincronização em tempo real, consultas robustas e recursos de segurança integrados, fazendo dele uma opção poderosa para armazenamento e sincronização de dados.

Dessa forma, os recursos citados anteriormente foram empregados no trabalho para realizar a autenticação e o armazenamento de informações importantes para essa etapa.

4 ARQUITETURA PROPOSTA

4.1 METODOLOGIA

Essa seção é destinada à exposição da metodologia utilizada no desenvolvimento do projeto. Para alcançar os objetivos previamente abordados nesse documento, optou-se por dividir a execução do trabalho em etapas, conforme descrito nos items a seguir:

- **Etapa 1 → Definição do DER:** Aqui foram definidas todas as entidades necessárias e seus respectivos relacionamentos, que posteriormente foram implementados no banco de dados.
- **Etapa 2 → Criação do banco de dados:** Nessa etapa o banco de dados relacional foi criado utilizando a linguagem SQL e o diagrama desenvolvido na etapa anterior.
- **Etapa 3 → Definição da API:** Utilizando o *Swaggerhub*, toda a definição da API foi criada, o que incluiu os recursos que foram posteriormente consumidos pela aplicação.
- **Etapa 4 → Desenvolvimento e implementação da API:** Utilizou-se o *WSO2 Integration Studio*, juntamente com a definição da API criada na etapa anterior, para integrar os recursos com o banco de dados.
- **Etapa 5 → Transposição do ambiente local para ambiente em nuvem:** Com os recursos operando de forma satisfatória, fez-se necessário oferecer esse serviço em um ambiente descentralizado, o que facilitou o seu consumo por parte da aplicação.
- **Etapa 6 → Desenvolvimento da aplicação móvel:** Por meio de ferramentas como o *React Native* e o *Android Studio*, foi projetada a aplicação móvel para o SO *Android*, com o objeto de realizar a leitura da tag NFC, o consumo da API e a exibição dos dados ao usuário.

4.2 CONFIGURAÇÃO E DESENVOLVIMENTO DA ARQUITETURA

Nessa seção será mostrada e detalhada a configuração da arquitetura, conforme exemplificada na figura 4.1.

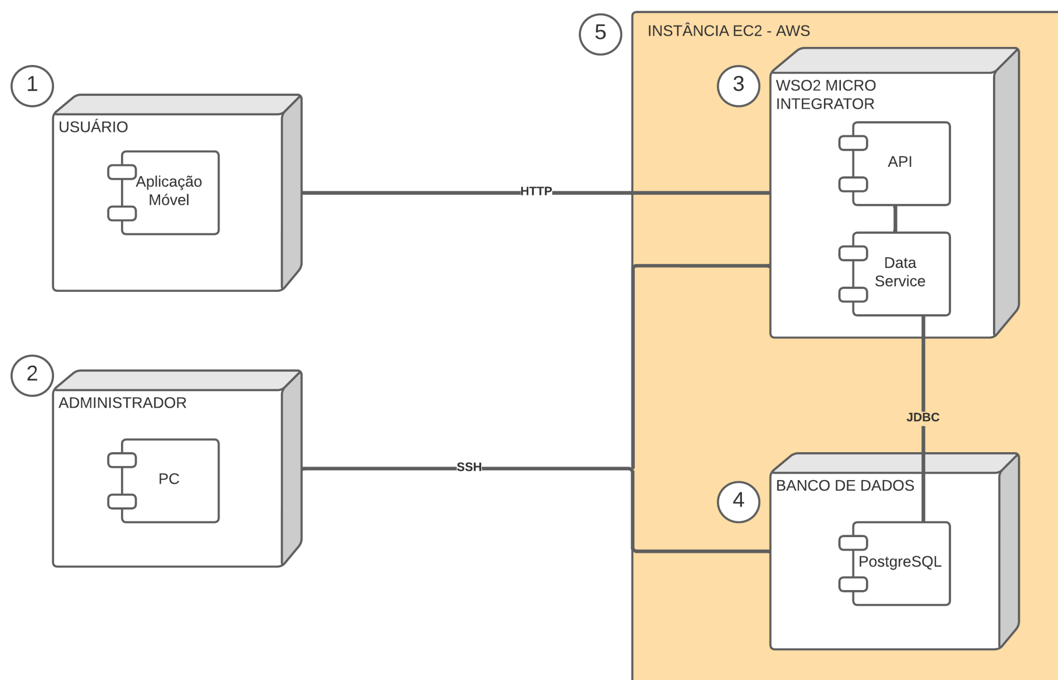

Figura 4.1: Arquitetura final proposta. Fonte: autores

Na arquitetura proposta, a API é executada pelo *WSO2 Micro Integrator* (3), que se comunica localmente com o banco de dados (4), por meio do JDBC. Todos se encontram hospedados em uma mesma instância EC2 (5), operando como uma aplicação monolítica. Então, os usuários (1) utilizam a aplicação móvel para consumir a API, com esse consumo sendo realizado por protocolo HTTP. Ainda, o adminstrador (2) tem acesso ao *back-end* via SSH, diretamente na instância.

Os detalhes de cada componente e seus respectivos relacionamentos serão explicados nas subseções deste capítulo.

4.2.1 Definição do DER

Para realizar a construção do DER foi utilizado o software *ERwin Data Modeler*, que possibilitou a criação de um modelo físico-lógico, o que facilitou a sua posterior implementação em um banco de dados.

Em uma primeira fase foram definidas todas as entidades do modelo, com base no objetivo do projeto:

- **Militar:** Trata-se da pessoa física pertencente a alguma instituição militar, a entidade central.
- **Comorbidade:** Refere-se à entidade relacionada às doenças e condições a qual um ser humano pode ter sua saúde submetida.
- **Medicamento:** É o produto utilizado para o tratamento de comorbidades, geralmente receitado por um médico.

O próximo passo foi determinar como essas entidades, uma vez definidas, se relacionam entre si. Ao observar a entidade "Militar", percebe-se um relacionamento de posse em relação às outras duas. Desse modo, um militar pode possuir nenhuma ou várias comorbidades e utilizar nenhum ou vários medicamentos. Da mesma forma, uma comorbidade pode ser possuída por nenhum ou vários militares, assim como um medicamento pode ser utilizado por nenhum ou vários militares.

Portanto, ambas as relações militar-comorbidade e militar-medicamento são de cardinalidade (n:n) ou "muitos para muitos". Porém, em um banco de dados relacional não é possível representar esse tipo de cardinalidade, sendo necessária a criação de entidades associativas que permitam relações (1:n) com as entidades fortes. Dessa forma, foram obtidas duas novas entidades, chamadas de militar-comorbidade e militar-medicamento.

Na etapa seguinte, determinou-se os atributos de cada uma das entidades, ou seja, as informações que caracterizam cada uma delas. Dentre esses atributos escolheu-se um deles para cada uma das entidades fortes. Ele deve ser único para cada registro inserido no banco de dados, pois exerce o papel de chave primária da sua respectiva tabela. A seguir são mostradas as chaves de cada uma das entidades fortes.

- **CPF:** Trata-se do cadastro de pessoas físicas, sendo um código único para cada cidadão brasileiro. Dessa forma, foi utilizado como chave primária da tabela "MILITAR".
- **CID:** Refere-se ao código internacional de doenças, criado pela OMS, para classificar as diversas comorbidades e condições de saúde reconhecidas pela organização. Cada doença possui um código próprio e único, sendo adequado para funcionar como chave primária da tabela "COMORBIDADE".
- **GTIN:** O Global Trade Item Number é um código utilizado mundialmente para identificar produtos comercializados, o mesmo que forma o código de barra das embalagens. Dentro desses produtos se encontram os medicamentos, portanto, o GTIN foi o ideal para realizar a função de chave primária da tabela "MEDICAMENTO".

Em relação às entidades associativas, utilizou-se ambas as chaves estrangeiras, das tabelas às quais cada uma delas se referem, como uma chave primária composta.

Por fim, determinou-se o domínio e extensão dos respectivos atributos de acordo com a necessidade específica de cada um deles. Por exemplo, o CPF, sendo um código de onze caracteres fixos, foi determinado como tipo 'CHAR(11)'.

Com a consolidação de todos os procedimentos descritos acima, chegou-se no DER exemplificado na figura 4.2

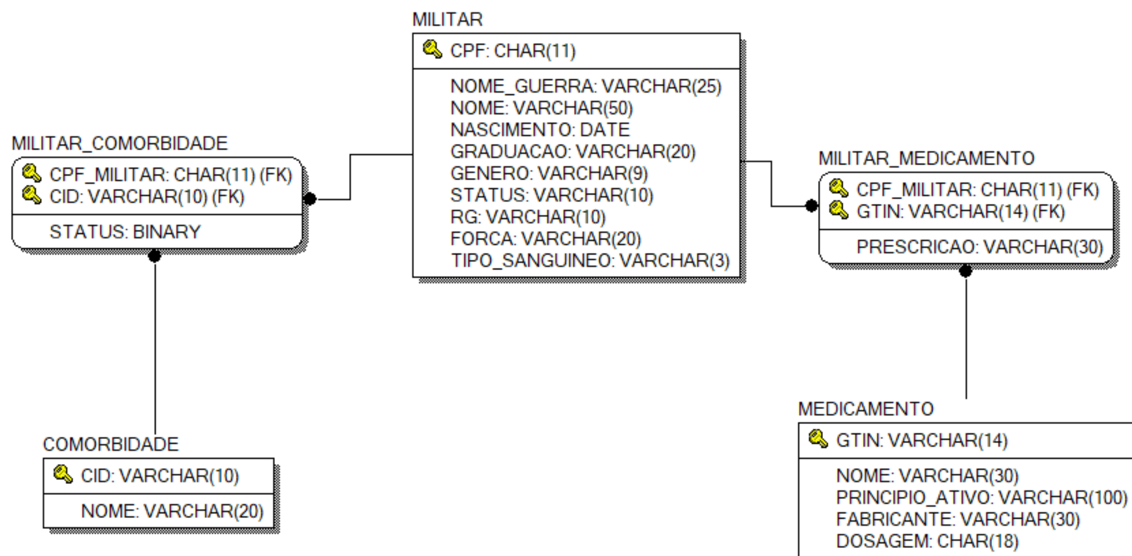

Figura 4.2: Diagrama Entidade Relacionamento da arquitetura proposta. Fonte: autores

4.2.2 Criação do Banco de Dados

Com o DER construído, foi realizada a criação do banco de dados *PostgreSQL*, onde foram gravadas e armazenadas as informações descritas na subseção anterior.

Para isso foi necessária a execução de algumas etapas: instalação do *PostgreSQL* em ambiente local, criação e execução dos *scripts* da estrutura do banco de dados, e criação e execução dos *scripts* de inserção de dados.

4.2.2.1 Instalação do PostgreSQL

Como visto na seção 3.10, o *PostgreSQL* é um *software* de banco de dados relacional de código aberto desenvolvido pela *PostgreSQL Global Development Group*. Em https://www.postgresql.org/download/ é possível ter acesso aos arquivos e tutoriais de instalação para diferentes SO. Nesse projeto foi utilizado o *Windows* 11 e o *PostgreSQL* 15, com a instalação sendo realizada com o instalador contendo o *PostgreSQL server* e o administrador *pgAdmin* 4.

4.2.2.2 Criação da Estrutura do Banco de Dados

Utilizando como referência o diagrama criado na subseção 4.2.1, foram escritos *scripts* SQL. Dessa forma, todas as tabelas, com seus respectivos atributos e constraints, foram criadas. A execução foi feita através do *pgAdmin* 4, que possibilita o acesso ao banco de dados por meio de uma interface gráfica. Um exemplo pode ser visto na figura 4.3.

DB_MILITAR/postgres@PostgreSQL 15

No limit

Query   Query History   Scratch Pad

```sql
create table "tb_militar"(
    cpf char(11) not null,
    nome_guerra varchar(25) not null,
    nome varchar(50) not null,
    nascimento date not null,
    graduacao varchar(20) not null,
    genero varchar(15) not null,
    status varchar(10) not null,
    rg varchar(10) not null,
    forca varchar(20) not null,
    tipo_sanguineo varchar(3) not null,
    primary key ("cpf")
);

create table "tb_comorbidade"(
    cid varchar(10) not null,
    nome varchar(50) not null,
    primary key ("cid")
);

create table "tb_medicamento"(
    gtin varchar(14) not null,
    nome varchar(50) not null,
    principio_ativo varchar(50) not null,
    fabricante varchar(30) not null,
    dosagem varchar(20) not null,
    primary key ("gtin")
);

create table "tb_militar_medicamento"(
    gtin_medicamento varchar(14) not null,
    cpf_militar char(11) not null,
    prescricao varchar(50) not null,
    primary key ("gtin_medicamento", "cpf_militar"),
    constraint "fk_militar" foreign key ("cpf_militar") references tb_militar("cpf"),
    constraint "fk_medicamento" foreign key ("gtin_medicamento") references tb_medicamento("gtin")
);

create table "tb_militar_comorbidade"(
    cid_comorbidade varchar(10) not null,
```

Figura 4.3: *Script* SQL no *PgAdmin* 4. Fonte: autores

4.2.2.3 Inserção de Dados

Com a estrutura do banco de dados criada, foi imprescindível a inserção de dados para posterior consumo pela API. Para objeto de estudo, todos os dados gerados são fictícios, apesar de possuírem o mesmo formato dos dados reais, como propostos anteriormente nesse documento. A única exceção foram as comorbidades, cujos dados foram retirados diretamente do CID 11, fornecido pela OMS.

A geração dos dados fictícios foi feita com o auxílio da ferramenta *ChatGPT* 3 que, como visto na seção 3.11, é um *chatbot online*, que utiliza inteligência artificial para executar tarefas. Para isso, foi fornecida à ferramenta toda a estrutura das tabelas e, então, um conjunto de dados foi gerado de acordo com essa definição, respeitando os respectivos tipos e tamanhos de cada entidade.

A partir dos dados gerados, foram criados *scripts* SQL para que toda essa informação fosse alocada corretamente no banco de dados anteriormente criado.

4.2.3 Definição da API

Toda a informação, agora consolidada no banco de dados, deve ser consumida pela aplicação final. Sendo assim, optou-se pelo uso de uma API *RESTful* para realizar a leitura, inscrição, atualização e remoção desses dados. Como detalhado na seção 3.4, uma API *RESTful* utiliza o protocolo HTTP e seus

respectivos métodos para fazer a manipulação dos dados, como descrito anteriormente neste parágrafo.

O primeiro passo para a criação da API foi escrever a sua definição utilizando o *Swaggerhub*, já descrito em detalhes na seção 3.12 deste documento.

Assim, cada recurso foi definido para uma função específica, de acordo com as necessidades do projeto. Os seus respectivos *paths*, estão detalhados a seguir:

- **/Militar [POST]:** Realiza a inserção de dados de um militar. Destinado ao cadastro de novos militares.
- **/Militar [PUT]:** Faz a atualização dos dados de um militar já cadastrado.
- **/Militar/{cpf} [GET]:** Realiza a consulta dos dados de um militar já cadastrado, por meio de seu CPF.
- **/Militar/{cpf} [DELETE]:** Utilizado para deletar todos os dados de um militar, através do seu CPF.
- **/Medicamento [POST]:** Insere informações sobre a medicação utilizada por um militar.
- **/Medicamento [PUT]:** Atualiza os dados sobre a administração de um medicamento utilizado por um militar.
- **/Medicamento/{cpf} [GET]:** Consulta as informações de toda a medicação consumida por um militar, utilizando o seu CPF.
- **/Medicamento/{cpf}/{gtin} [DELETE]:** Apaga a prescrição de um medicamento para um militar.
- **/Medicamento/Registro/{gtin} [GET]:** Consulta as informações de um medicamento cadastrado.
- **/Comorbidade [POST]:** Relaciona uma doença ao militar que a possui.
- **/Comorbidade [PUT]:** Atualiza o status de uma doença possuída por um militar.
- **/Comorbidade/{cpf} [GET]:** Consulta as informações de todas as doenças possuídas por um militar, utilizando o seu CPF.
- **/Comorbidade/{cpf}/{cid} [DELETE]:** Exclui o vínculo de uma doença ao militar que a possui.
- **/Comorbidade/Registro/{cid} [GET]:** Consulta as informações de uma doença cadastrada.

As informações sobre um medicamento ou uma comorbidade só podem ser inseridas, alteradas ou excluídas diretamente no banco de dados, por serem informações pouco voláteis e que não devem ser alteradas pelo usuário padrão. Apenas a consulta dessas informações foi permitida por meio da API.

Também foram definidas as estruturas de todos os *payloads*, sendo utilizado o formato JSON.

Ao final, é possível exportar a definição criada em formato YAML ou JSON, assim como sua documentação em HTML.

4.2.4 Desenvolvimento e Implementação da API

Como explicado na seção 3.13, a IDE *WSO2 Integration Studio* permitiu o desenvolvimento da API e sua implementação utilizando o WSO2 MI. O tutorial de instalação e configuração desse programa está disponível em https://ei.docs.wso2.com/en/latest/micro-integrator/develop/installing-WSO2-Integration-Studio/.

Primeiro, criou-se um novo *Integration Project*, com a criação de 3 módulos como exemplificado na figura 4.4. O próximo passo foi configurar 3 arquivos nesse projeto: A API REST, o *Endpoint* e o *Data Service*.

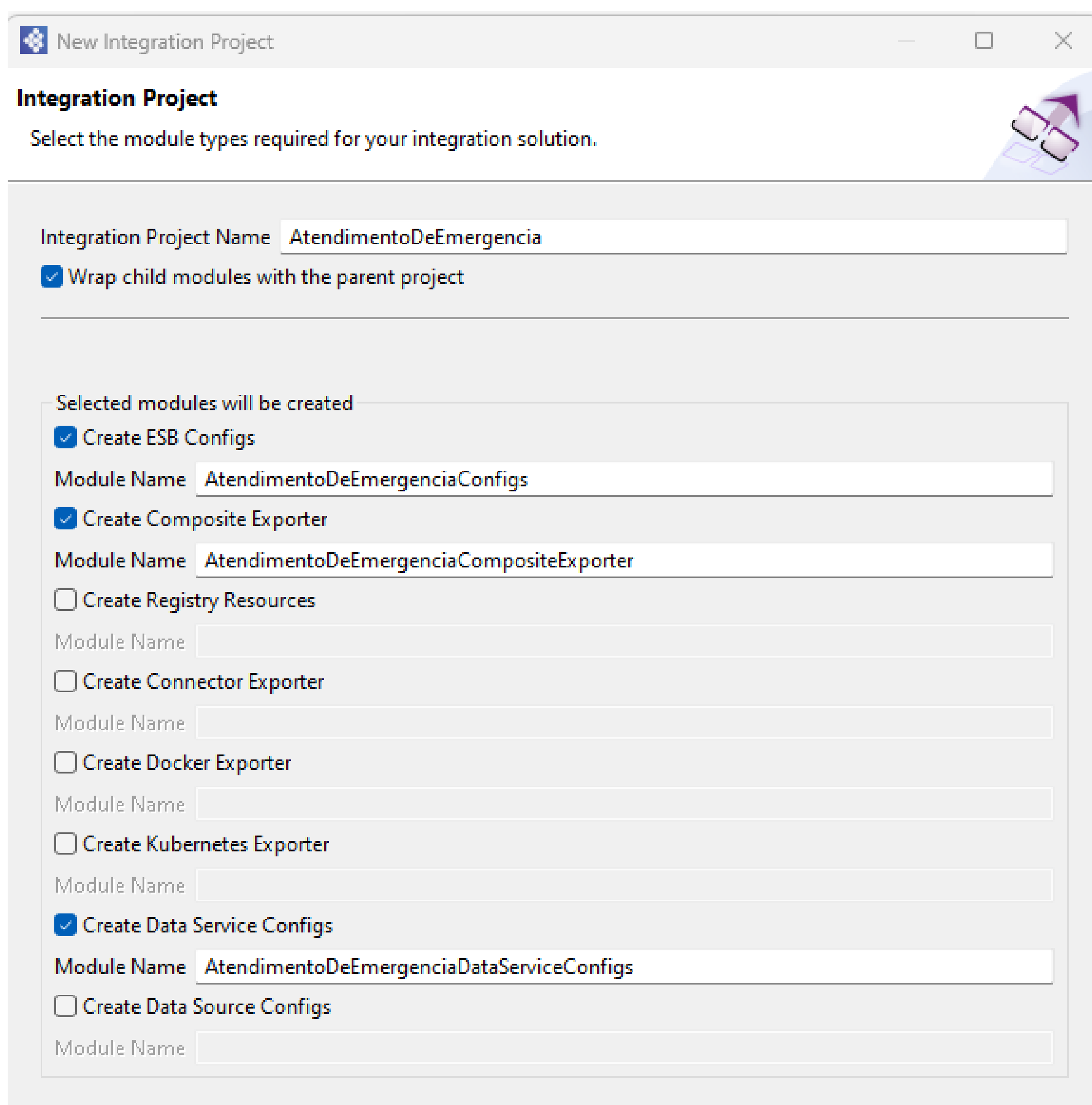

Figura 4.4: Criação de um novo *Integration Project*. Fonte: autores

Nas próximas subseções serão detalhados os processos de criação e programação desses três arquivos.

4.2.4.1 API REST

Uma nova API REST pode ser gerada por meio de uma definição *Swagger*, utilizando o arquivo YAML exportado do *Swaggerhub*. Dessa maneira, toda a estrutura da API foi construída de forma automatizada. Para isso, acessou-se *Configs* > *New* > *REST API* > *Generate API using Swagger Definition* e foi carregado o arquivo YAML.

O arquivo da API, em formato XML, se encontra em *Configs* > *src* > *main* > *synapse-config* > *api*. Abrindo-o tem-se acesso ao código fonte, onde foi possível editar o contexto geral e adicionar as regras de negócio para manipular o *payload* produzido pelo *Data Service*, que será visto posteriormente.

A regra de negócio foi feita em *javascript*, na qual se verifica o conteúdo do *payload* e são capturados possíveis erros de servidor. Caso o *payload* esteja vazio, o mesmo é zerado novamente e o *status* HTTP é definido para 404; caso contrário, o *payload* é mantido e o *status* se torna 200. Em ocorrência de erro o *payload* também é zerado e o *status* definido como 500. O trecho do código pode ser visto na figura 4.5

```
<outSequence>
    <script language="js"><![CDATA[var payload = mc.getPayloadJSON();
        try {
            if(payload.Militares.Militar == undefined) {
                mc.setProperty('status','404');
                payload = {};
            }
            else {
                mc.setProperty('status','200');
                payload = payload.Militares.Militar[0];
            }
        }
        catch(e) {
            mc.setProperty('error', e);
            mc.setProperty('status','500');
            payload = {};
        }
        mc.setPayloadJSON(payload);]]></script>
    <property expression="$ctx:status" name="HTTP_SC" scope="axis2" type="STRING"/>
    <respond/>
</outSequence>
```

Figura 4.5: *Script* da regra de negócio. Fonte: autores

Vale ressaltar que essa regra só foi definida para os métodos *GET*, pois são os únicos que possuem *payload* não nulo na resposta.

Terminando a configuração do arquivo da API REST, ainda dentro das regras de negócio, definiu-se o *endpoint* para qual os recursos são direcionados, que também será explorado em detalhes em uma próxima subseção desse documento.

4.2.4.2 Data Service

O *Data Service* é responsável por realizar a comunicação com banco de dados, recuperando ou gravando informações específicas. Quando um recurso da API é requisitado pela aplicação, o *endpoint* redire-

ciona para esse serviço, que faz a consulta necessária ao banco de dados para realizar a tarefa definida pelo respectivo recurso acionado: extrair, guardar, alterar ou deletar informações. Os dados são, então, tratados e encapsulados em um *payload* JSON, com estrutura anteriormente definida na subseção 4.2.3 e enviados como resposta ao *endpoint*.

A comunicação com o banco de dados foi feita por meio de *querys*, utilizando linguagem SQL. Para cada recurso tem-se um *script* definido, que executa uma tarefa específica. Por exemplo, para o método *GET* é feito um *SELECT* das informações necessárias, enquanto no *POST* realiza-se um *INSERT* dos dados enviados pelo usuário.

Para a criação do *Data Service* seguiu-se em *DataServiceConfigs* > *dataservice* > *new* > *Data Service*. Dentro da pasta *dataservice* se encontra um arquivo ".dbs", no qual foram programadas as suas funcionalidades.

O primeiro passo foi configurar as credenciais do banco de dados como a *url*, o *username* e o *password*, como detalhado na figura 4.6. Em *driverClassName*, foi colocado o JDBC específico para o banco de dados utilizado, no caso estudado sendo o *PostgreSQL*. Esse conector pode ser obtido no próprio site do desenvolvedor do *software*.

```
<config id="AtendimentoEmergenciaDS">
    <property name="driverClassName">org.postgresql.Driver</property>
    <property name="url">jdbc:postgresql://127.0.0.1:5432/DB_MILITAR</property>
    <property name="username">MILITAR</property>
    <property name="password">MILITAR</property>
</config>
```

Figura 4.6: Configuração das credenciais de acesso ao banco de dados. Fonte: autores

Em uma próxima etapa, foram definidos os recursos disponíveis na API. Aqui foram mapeados todos os parâmetros passados pelo usuário, presentes tanto no cabeçalho quanto no corpo da mensagem HTTP. Um exemplo pode ser visualizado na figura 4.7.

```
<resource path="/Militar" method="POST">
    <description />
    <call-query href="MilitarPOST">
        <with-param name="cpf" query-param="cpf" />
        <with-param name="nome" query-param="nome" />
        <with-param name="nome_guerra" query-param="nome_guerra" />
        <with-param name="nascimento" query-param="nascimento" />
        <with-param name="graduacao" query-param="graduacao" />
        <with-param name="genero" query-param="genero" />
        <with-param name="status" query-param="status" />
        <with-param name="rg" query-param="rg" />
        <with-param name="forca" query-param="forca" />
        <with-param name="tipo_sanguineo" query-param="tipo_sanguineo" />
    </call-query>
</resource>
```

Figura 4.7: Configuração do recurso /Militar *POST*, disponibilizados pela API. Fonte: autores

Finalmente, foram configuradas as *querys* que são empregadas por seus respectivos recursos, para acessar o banco de dados. Em *useConfig*, utilizou-se a configuração de acesso ao banco de dados que foi detalhada anteriormente nessa subseção; seguido pela própria *query* e o mapeamento dos parâmetros da mensagem HTTP e das variáveis do banco de dados. Na figura 4.8 é possível observar um exemplo.

```xml
<query id="MilitarPOST" useConfig="AtendimentoEmergenciaDS">
    <sql>
        insert into
            tb_militar(cpf, nome, nome_guerra, nascimento, graduacao, genero, status, rg, forca, tipo_sanguineo)
        values
            (:cpf, :nome, :nome_guerra, TO_DATE(:nascimento, 'DD/MM/YYYY'), :graduacao, :genero, :status, :rg, :forca, :tipo_sanguineo)
    </sql>
    <param name="cpf" sqlType="STRING" paramType="SCALAR" optional="false" />
    <param name="nome" sqlType="STRING" paramType="SCALAR" optional="false" />
    <param name="nome_guerra" sqlType="STRING" paramType="SCALAR" optional="false" />
    <param name="nascimento" sqlType="STRING" paramType="SCALAR" optional="false" />
    <param name="graduacao" sqlType="STRING" paramType="SCALAR" optional="false" />
    <param name="genero" sqlType="STRING" paramType="SCALAR" optional="false" />
    <param name="status" sqlType="STRING" paramType="SCALAR" optional="false" />
    <param name="rg" sqlType="STRING" paramType="SCALAR" optional="false" />
    <param name="forca" sqlType="STRING" paramType="SCALAR" optional="false" />
    <param name="tipo_sanguineo" sqlType="STRING" paramType="SCALAR" optional="false" />
</query>
```

Figura 4.8: Configuração das *querys* do recurso /Militar *POST*. Fonte: autores

Assim, toda a lógica de manipulação dos dados e mapeamento dos recursos foi completada.

4.2.4.3 Endpoint

Trata-se do componente mais simples dentre os três aqui detalhados. Sua função é conectar os contextos da API aos seus repectivos recursos no *Data Service*.

Para a criação de um *endpoint*, foi-se em*Configs* > *src* > *main* > *synapse-config* > *endpoints* > *new* > *Endpoint*. Agora, na pasta *endpoints*, foi gerado um arquivo XML, que foi utilizado para configurar o *endpoint*. No arquivo, foi colocado o URI como o endereço do arquivo de *Data Service*, como pode ser visto na figura 4.9.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<endpoint name="AtendimentoEmergenciaEP" xmlns="http://ws.apache.org/ns/synapse">
    <address uri="http://localhost:8290/services/AtendimentoEmergenciaDS">
        <suspendOnFailure>
            <initialDuration>-1</initialDuration>
            <progressionFactor>1.0</progressionFactor>
        </suspendOnFailure>
        <markForSuspension>
            <retriesBeforeSuspension>0</retriesBeforeSuspension>
        </markForSuspension>
    </address>
</endpoint>
```

Figura 4.9: Configuração do *endpoint*. Fonte: autores

Com essa configuração concluída, o *back-end* se tornou operacional em ambiente local.

4.2.5 Transposição do Ambiente Local para a Nuvem

A próxima etapa da arquitetura proposta, com o *back-end* operacional em ambiente local, foi transpor essa estrutura para uma aplicação em nuvem.

Como visto na seção 3.14, a AWS é uma plataforma que fornece serviços de computação em nuvem. Ela foi escolhida devido a sua ampla aceitação no mercado e pela opção de *tier* gratuito para testes. Dessa forma, para objetivos de demonstração em pequena escala, a solução atendente perfeitamente o objetivo proposto.

Antes de aprofundar na execução do projeto na plataforma *online*, foi necessário preparar todos os recursos previamente desenvolvidos para essa transferência.

4.2.5.1 Exportação do Projeto da API

O projeto da API precisou ser exportado em um arquivo CAR, que é executado pelo WSO2 MI, neste caso a versão 4.2.0. Para isso, seguiu-se em *CompositeExporter* > *Export Composite Application Project* e adicionou-se um destino para o arquivo. Em seguida, com o arquivo em posse, foi preciso movê-lo para o seguinte diretório, dentro do WSO2 MI: *repository* > *deployment* > *server* > *carbonapps*. Assim, quando o *Micro Integrator* for iniciado, o servidor executa o arquivo CAR gerado. Ainda, o JDBC do *PostgreSQL* precisou ser adicionado à pasta *lib*, também presente na pasta do WSO2 MI, para realizar a comunicação com o banco de dados.

4.2.5.2 *Backup* do Banco de Dados

Sequentemente, o *backup* do banco de dados precisou ser realizado. Com esse objetivo, utilizou-se o *pgAdmin 4*, que possui uma ferramenta própria para essa tarefa. Selecionou-se a ferramenta "*Backup*"no banco. Nas opções colocou-se no nome do arquivo e a escolha do formato *plain*. Assim, o arquivo de *backup* foi criado.

4.2.5.3 Configuração da Instância EC2

Agora com os recursos preparados, a instância EC2 foi criada e configurada para abrigá-los. O tutorial para a criação e acesso de uma instância pode ser obtido em https://docs.aws.amazon.com/pt_br/AWSEC2/latest/UserGuide/EC2_GetStarted.html?icmpid̄docs_ec2_console. Para o projeto foi utilizada uma instância t2.micro executando o SO *Ubuntu*.

Toda vez que uma instância é reiniciada, o seu IP público é alterado. Dessa maneira, fez-se necessário a atribuição de um IP elástico, para que a API fosse consumida de forma mais prática. Para isso, acessou-se *IPs elásticos* > *Alocar endereço IP elástico* > *Alocar*. Agora, com o IP alocado, foi selecionada a instância e, em *Ações* > *Redes* > *Associar endereço IP elástico*, foi atribuído o IP previamente alocado. Assim, o endereço público da instância se tornou fixo.

Para permitir que a API se comunique com a internet, foi preciso criar um *Security Group*. Para isso,

foi-se em *Security Group > Criar Grupo de Segurança*. Nessa página foi designado um nome ao grupo e sua descrição. Após, definiu-se uma regra de entrada autorizando o protocolo TCP na porta 8290, que é a porta padrão HTTP do WSO2 MI; repetiu-se o mesmo processo para a regra de saída.

Finalizada a configuração do *Security Group*, foi necessário adicionar a instância contendo a API a ele. Novamente, no painel EC2, selecionou-se a instância e, em *Ações > Segurança > Alterar Grupos de Segurança*, foi adicionado o *Security Group* criado. Assim, a comunicação TCP pela porta 8290 se tornou operacional

4.2.5.4 Execução da Arquitetura na EC2

A instância configurada permitiu que os componentes fossem transportados e executados no mesmo sistema. O transporte do WSO2 MI e do *backup* do banco de dados foi feito por SCP a partir do *Windows PowerShell* da máquina local. Então executou-se os seguintes comandos, com os arquivos e a chave da instância presentes em um mesmo diretório:

- scp -i "mykeys.pem".\wso2mi-4.2.0 ubuntu@54.207.101.216:/home/ubuntu
- scp -i "mykeys.pem".\backup.sql ubuntu@54.207.101.216:/home/ubuntu

Com a transferência concluída, o próximo passo foi instalar o banco de dados *PostgreSQL* na máquina *Ubuntu*. A comunicação com a máquina foi feita utilizando SSH, para isso realizou-se o seguinte comando, na máquina local *Windows*:

- ssh -i "mykeys.pem"ubuntu@ec2-54-207-101-216.sa-east-1.compute.amazonaws.com

Agora dentro da instância *Ubuntu*, prosseguiu-se com a instalação:

- sudo apt-get install postgresql

Em sequência configurou-se o ambiente, com a criação de um novo usuário e de um novo banco de dados, para receber o *backup* dos dados. O processo foi conduzido da seguinte forma:

- sudo -u postgres psql
- psql=# CREATE DATABASE ’DB_MILITAR’;
- psql=# CREATE USER ’MILITAR’ WITH ENCRYPTED PASSWORD ’MILITAR’;
- psql=# GRANT ALL PRIVILEGES ON DATABASE ’DB_MILITAR’ TO ’MILITAR’;

Dessa forma, o banco de dados ficou pronto para receber os dados por meio do arquivo de *backup*. O procedimento foi:

- sudo -u postgres psql ’DB_MILITAR’ < backup.sql

Após a execução do comando acima, o banco de dados se tornou operacional para a execução do projeto.

A execução da API não necessitou de instalação, apenas a execução do arquivo *micro-integrator.sh*, como superusuário, dentro de *wso2mi-4.2.0/bin*. Assim, o WSO2 MI foi iniciado, executando a API a partir do arquivo CAR gerado anteriormente.

Dessa forma, a API se tornou totalmente operacional e pronta para ser consumida pela aplicação.

4.2.6 Aplicativo Móvel

Nessa seção será tratada a etapa de desenvolvimento da aplicação móvel, conforme apresentado na figura 4.1 .

O aplicativo foi desenvolvido com o intuito de auxiliar os atendimentos de emergência realizados por profissionais da saúde, fornecendo informações do paciente por meio da leitura de uma tag NFC, o que permite que a assistência possa ser feita com o acesso a dados fundamentais em uma situação de urgência. Nesta seção serão apresentadas as telas do aplicativo e um diagrama mostrando o funcionamento dessa solução.

O primeiro passo da solução desenvolvida é a realização do cadastro ou *login*, pois são tratadas informações sensíveis dos usuários. A figura 4.10 mostra as telas de *login*, cadastro e registro.

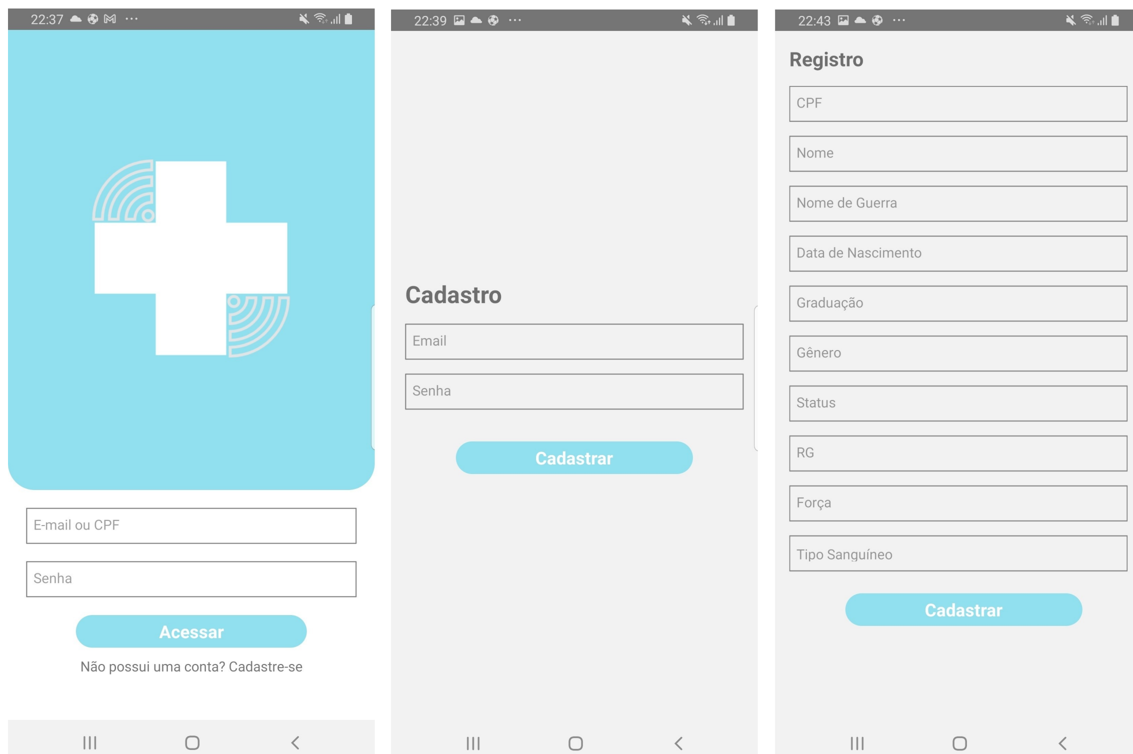

Figura 4.10: Telas de login, cadastro e registro de usuário. Fonte: autores

Na tela de *login* usou-se o *Firebase Authentication* que, após preencher o *email* e senha, realiza uma consulta na plataforma e, caso existam as credenciais, é possivel acessar a aplicação. Caso contrário, é possivel clicar no botão "Não possui uma conta? Cadastre-se"e iniciar o processo de cadastro. Além disso, a logo presente na tela é genérica, podendo ser alterada.

O primeiro passo do cadastro do usuário é a criação de um *email* e uma senha válidos, os quais são armazenados no *Firebase*. Ao realizar a etapa anterior, o usuário é redirecionado para a tela de *login* e, ao preencher as credenciais, é verificado se é seu primeiro acesso. Em caso positivo, o usuário é conduzido até a página de registro, na qual deve preencher todas as informações apresentadas, conforme a figura 4.10, para poder prosseguir para a tela inicial do aplicativo. Caso não seja o primeiro acesso, o usuário é "logado"no aplicativo.

A tela inicial, como mostrada na figura 4.11, conta com o nome do usuário, espaço para imagem de perfil, botões para cadastro de medicamentos e comorbidades, uma *Tab Bar* que contém as páginas do aplicativo, além de ser responsável pela navegação.

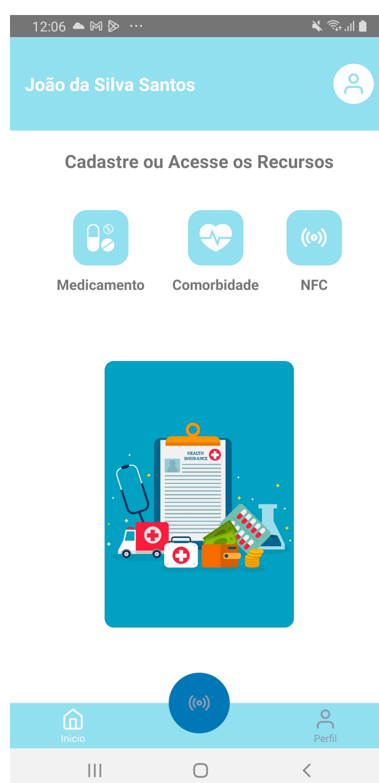

Figura 4.11: Layout da tela inicial da aplicação. Fonte: autores

Todos os medicamentos e comorbidades, relacionados ao usuário, devem ser cadastrados utilizando os botões presentes na tela inicial. O botão de medicamento solicita o informe do código de barras, para que o respectivo produto seja encontrado na base de dados. Em seguida, é solicitado sua prescrição ao usuário, para que essa medicação seja cadastrada. O botão de cadastro de comorbidade abre um *modal* que contém dois campos: o primeiro é uma lista contendo todas as doenças cadastradas na aplicação, enquanto o segundo informa se o usuário ainda possui a doença ou se já foi curado. Esses dois cadastros são muito importantes, pois permitem que os pronto socorristas verifiquem a presença dessas comorbidades e possam

agir de acordo com essas informações, facilitando o diagnóstico. As telas se encontram na figura 4.12.

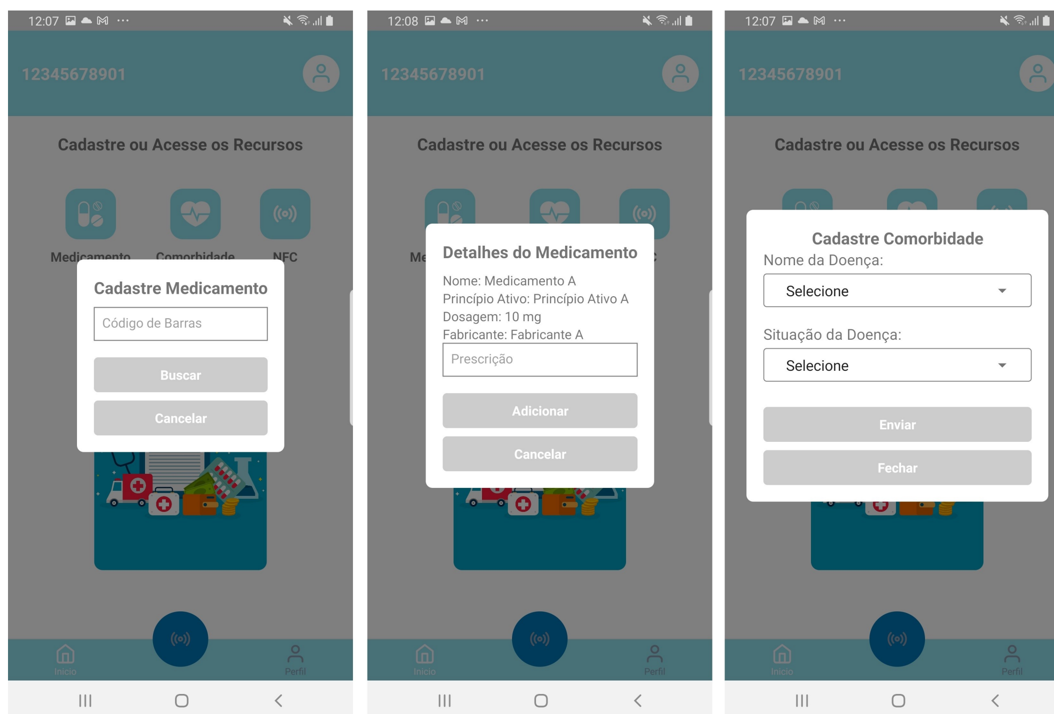

Figura 4.12: Telas da aplicação de cadastro de medicamento e cadastro de comorbidade. Fonte: autores

O acesso às informações do paciente é feito pela leitura do NFC. A tela possui um botão que, ao ser pressionado, abre um *modal* que, por sua vez, solicita a aproximação da *tag*, e permite também o cancelamento da leitura, como pode ser observado na figura 4.13.

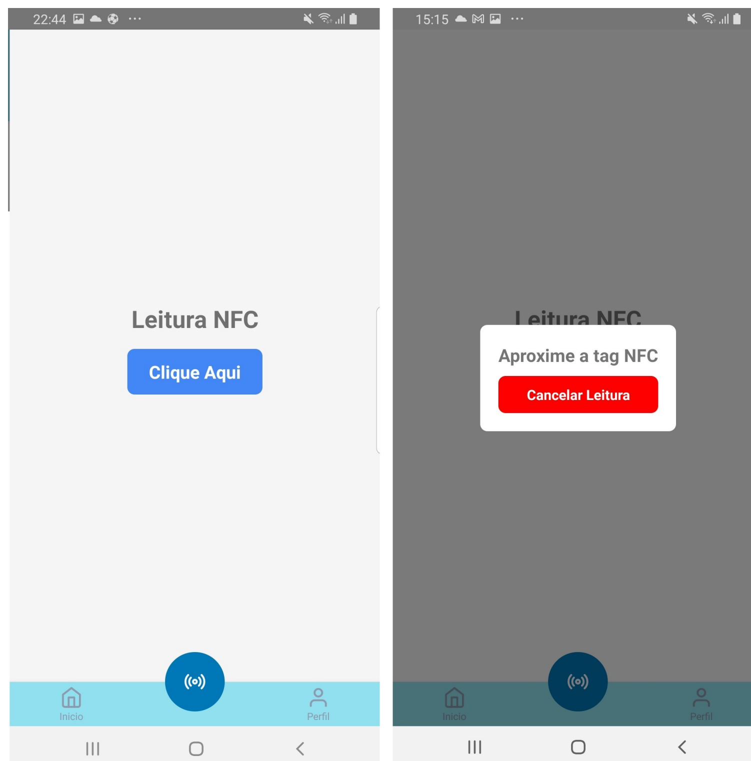

Figura 4.13: Telas de leitura da etiqueta NFC. Fonte: autores

Para realizar a comunicação NFC, foi utilizado um celular como receptor e um *chip NXP Ntag213* (com adesivo) que, além de transmitir a informação, também armazena os identificadores do paciente. Um exemplo pode ser visto na figura 4.14. O local mais adequado para a inserção do *chip* seria no interior dos sutaches dos militares (figura 4.15), pois pode ser definido como o local padrão de leitura da *tag* NFC, agilizando a sua localização pelos socorristas.

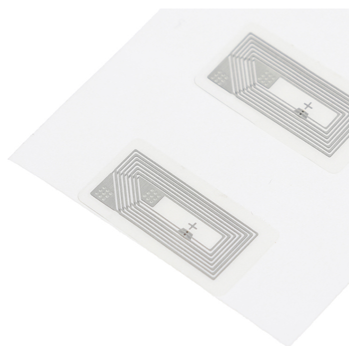

Figura 4.14: Exemplo de etiquetas NFC. Fonte: autores

Figura 4.15: Sutache adotado pelo Corpo de Bombeiros Militar do Distrito Federal. Fonte: Regulamento de Uniformes CBMDF

Ao realizar a leitura da etiqueta, o usuario é direcionado para a tela de informações do paciente, a qual contêm as informações de registro, das comorbidades (caso possua) e dos medicamentos (caso utilize). A tela de dados do paciente contém os mesmos campos dos dados de perfil do usuário, que podem ser observados na figura 4.16.

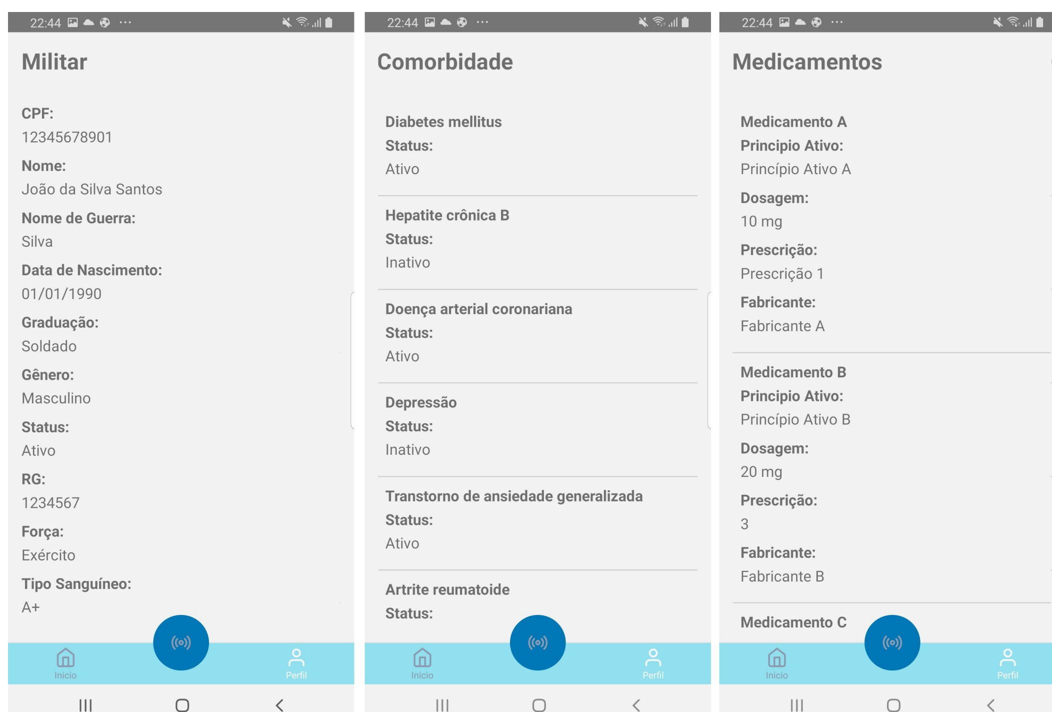

Figura 4.16: Telas contendo os campos de perfil do usuário e dados do paciente. Fonte: autores

Para o desenvolvimento do aplicativo foi utilizada a plataforma *React Native*, em conjunto com o *Android Studio*. Também fez-se uso de um celular *Samsung Galaxy S8+* com o SO *Android* 9, em modo de desenvolvedor, como dispositivo de testes e leitor de *tag* NFC.

Por fim, na figura 4.17 se encontra um diagrama de casos de uso da aplicação, apresentando as interações entre usuários e administradores com o aplicativo, fornecendo uma visão de alto nível do sistema.

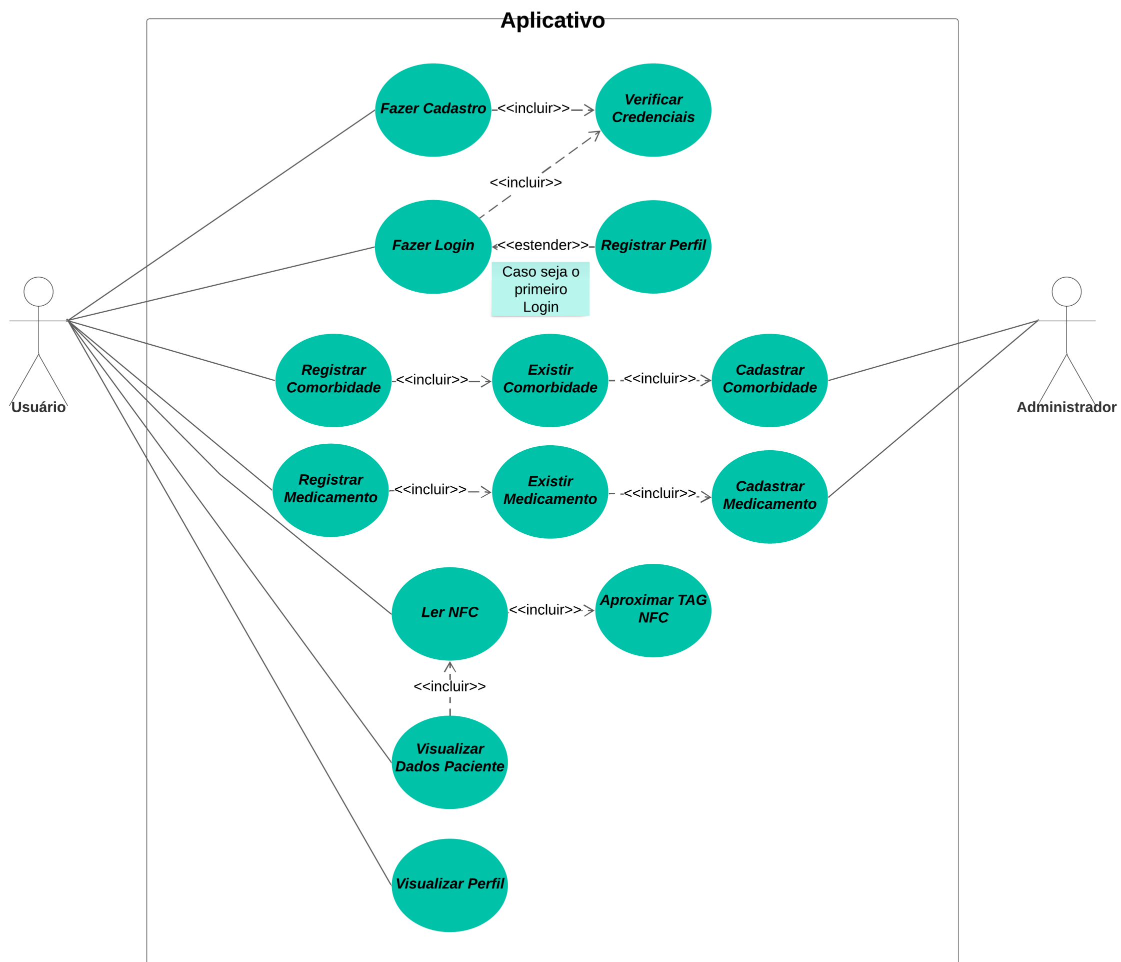

Figura 4.17: Diagrama de Caso de Uso do Aplicativo. Fonte: autores

5 TESTES E RESULTADOS

Nesse capítulo serão apresentados os testes executados para verificação do funcionamento da arquitetura desenvolvida nesse trabalho, bem como seus respectivos resultados.

5.1 METODOLOGIA

Os testes foram realizados em ambiente local, com todos os componentes sendo executados em uma mesma máquina. Assim, foi possível eliminar quaisquer interferências de fatores externos, como latência da conexão com os servidores da AWS.

A execução da API foi feita a partir da ferramenta *WSO2 Integration Studio*, que já possui o WSO2 MI integrado. A execução pode ser vista na figura 5.1.

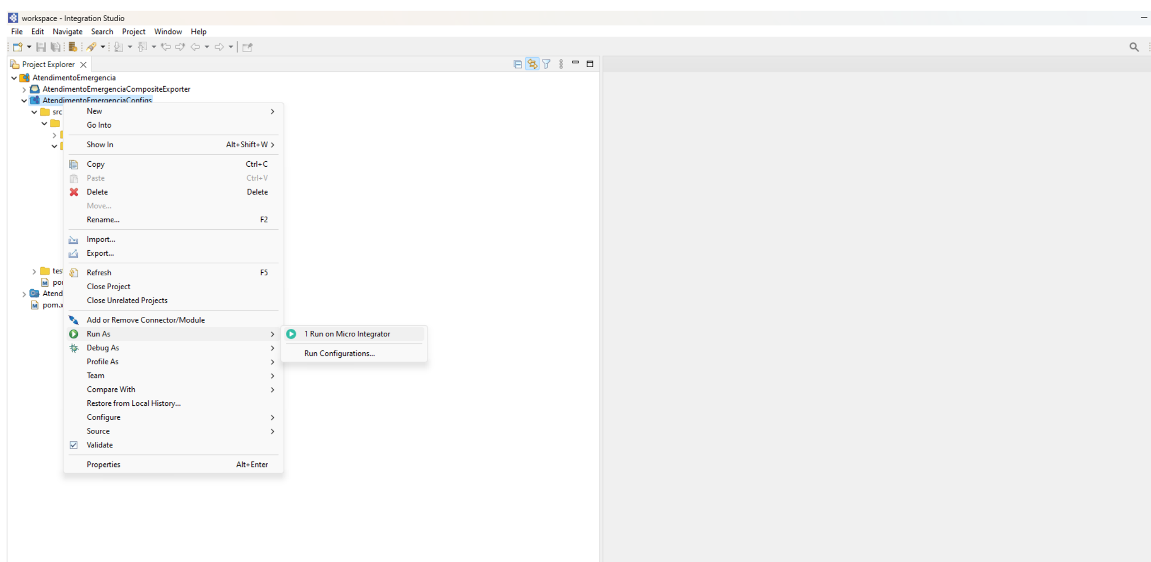

Figura 5.1: Execução da API no *WSO2 Integration Studio*. Fonte: autores

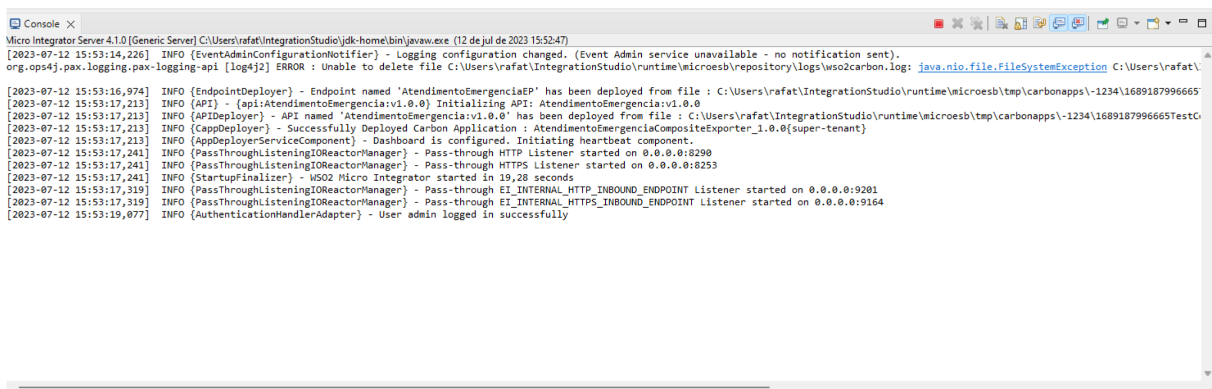

Figura 5.2: Console de execução da API no *WSO2 Micro Integrator*. Fonte: autores

Ainda, na figura 5.2, é possível observar o servidor preparado para receber conexões HTTP em endereço local (0.0.0.0) e porta 8290.

Para realizar o consumo da API, optou-se pelo uso da ferramenta *Postman*, que, como descrito na seção 3.15, permite o teste de APIs, realizando requisições HTTP. Uma coleção foi criada utilizando a definição *Swagger* anteriormente desenvolvida. Desse modo, todos os recursos são automaticamente criados e preparados para serem utilizados, conforme pode ser observado na figura 5.3.

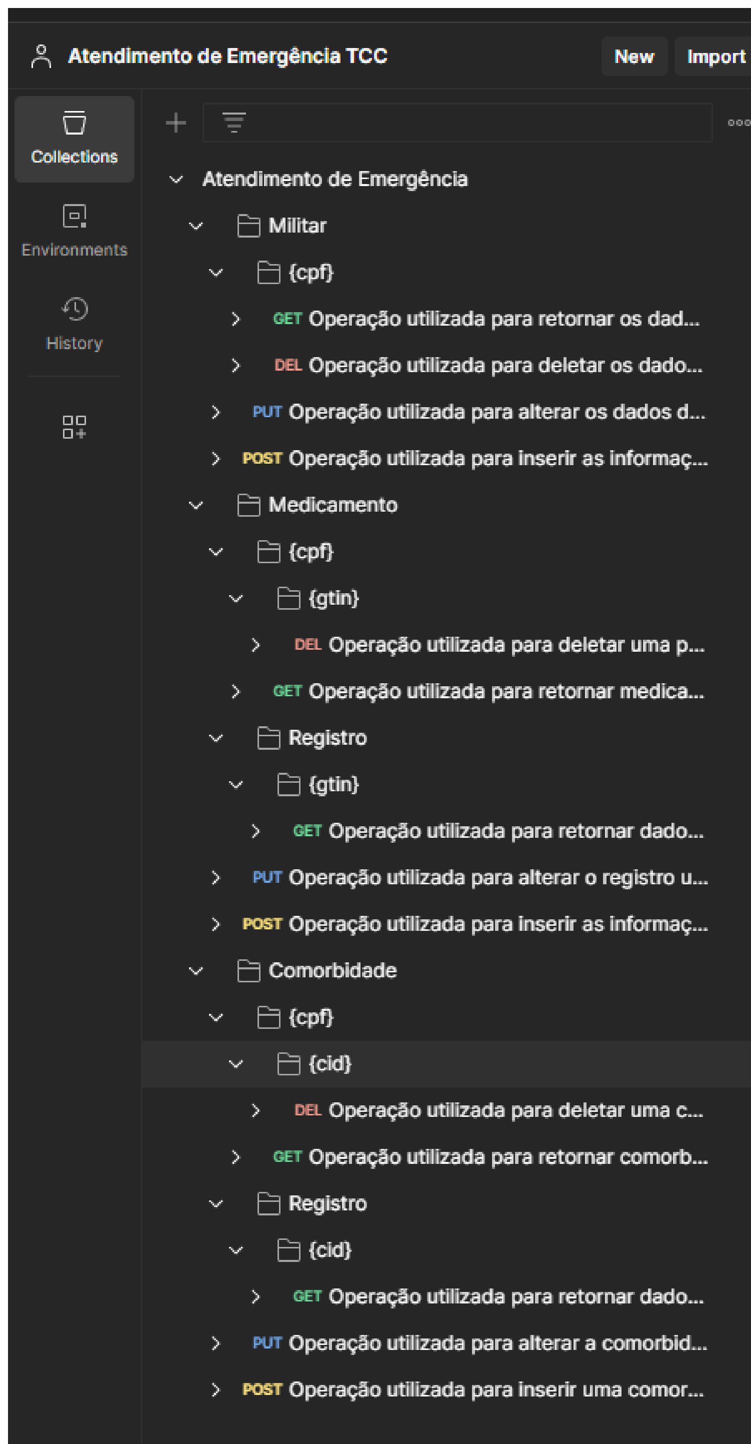

Figura 5.3: Coleção *Postman* para testes da API. Fonte: autores

Com todos os componentes preparados, foram realizadas as requisições para cada um dos recursos disponíveis. Todos eles, com seus respectivos métodos e funções, forma detalhados na seção 4.2.3 desse documento.

O objetivo foi verificar a consistência das respostas, comparando-as com as especificações definidas no capítulo 4. Além disso, também observou-se a latência de resposta entre o servidor e o cliente.

5.2 RESULTADOS OBTIDOS

Nessa seção serão expostos todos os resultados obtidos com a metodologia de testes descrita anteriormente neste capítulo.

5.2.1 Recurso Militar

O primeiro recurso testado foi o "militar", que é composto por quatro métodos. Os resultados para cada método podem ser vistos nas figuras 5.4, 5.5, 5.6 e 5.7.

Figura 5.4: Resultado do teste do método *GET* /Militar/{cpf}. Fonte: autores

Figura 5.5: Resultado do teste do método *POST* /Militar. Fonte: autores

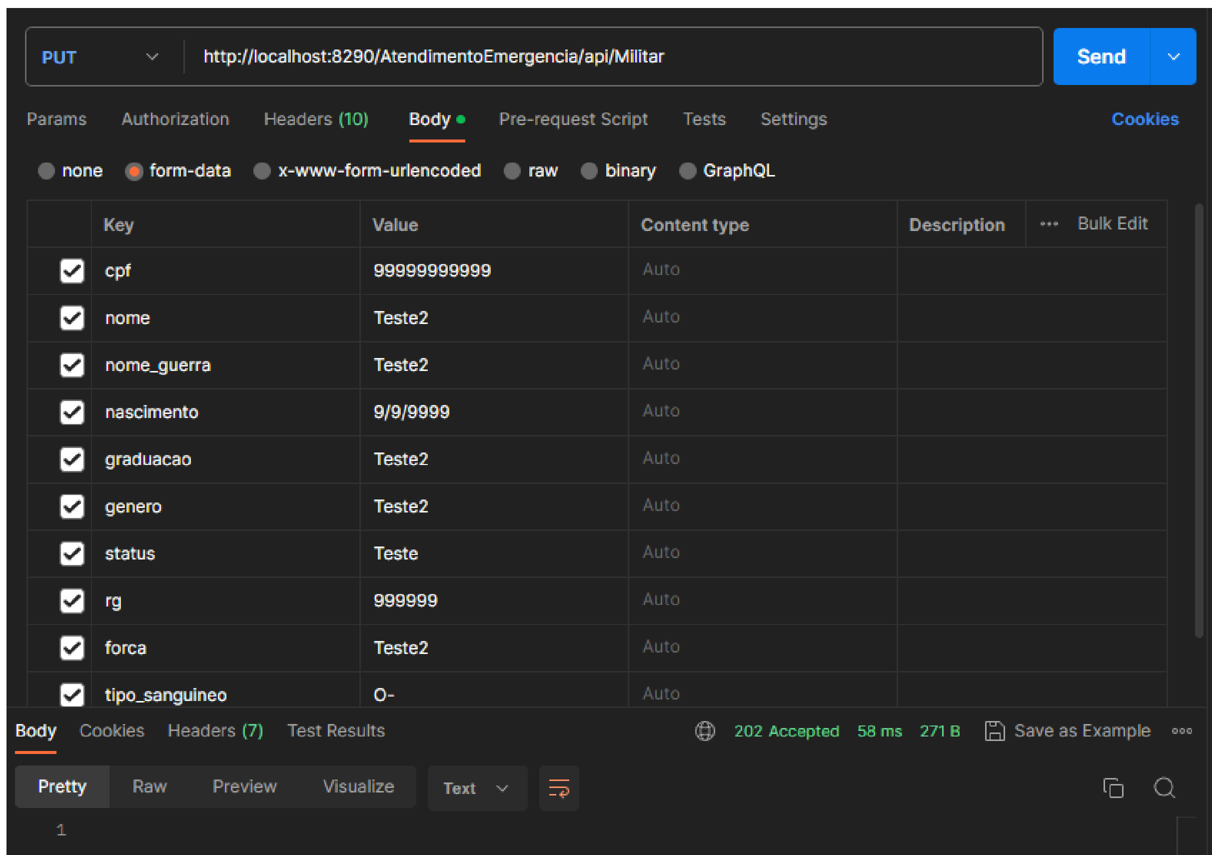

Figura 5.6: Resultado do teste do método *PUT* /Militar. Fonte: autores

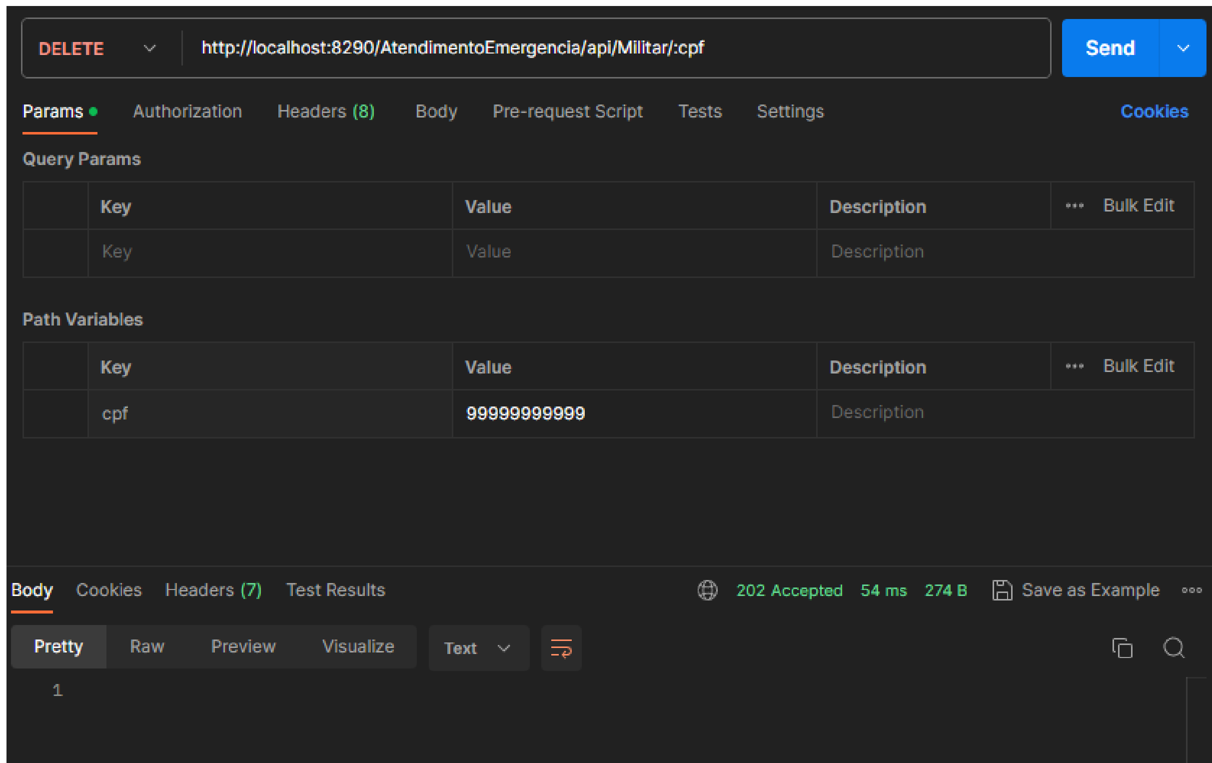

Figura 5.7: Resultado do teste do método *DELETE* /Militar/{cpf}. Fonte: autores

5.2.2 Recurso Medicamento

O recurso "medicamento"conta com cinco métodos distintos. As imagens 5.8, 5.9, 5.10, 5.11 e 5.12 contém os resultados alcançados.

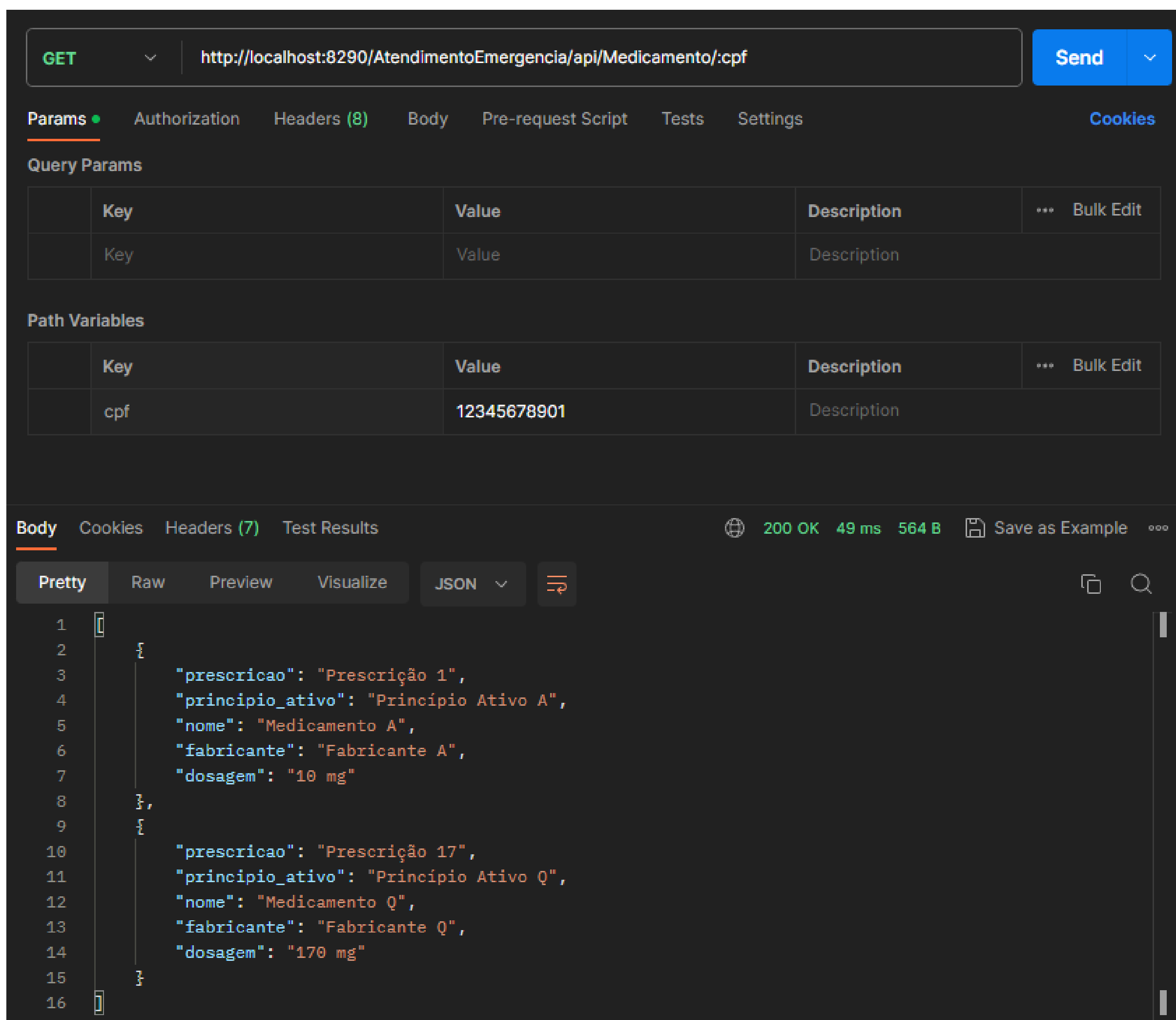

Figura 5.8: Resultado do teste do método *GET* /Medicamento/{cpf}. Fonte: autores

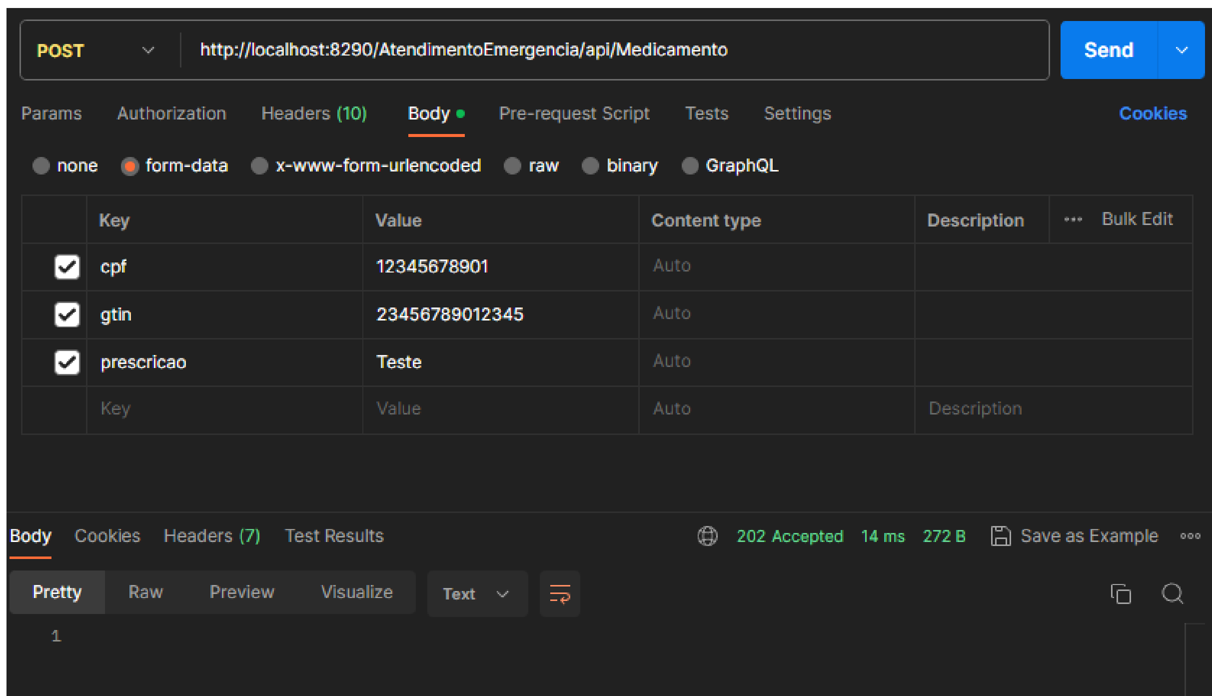

Figura 5.9: Resultado do teste do método *POST* /Medicamento. Fonte: autores

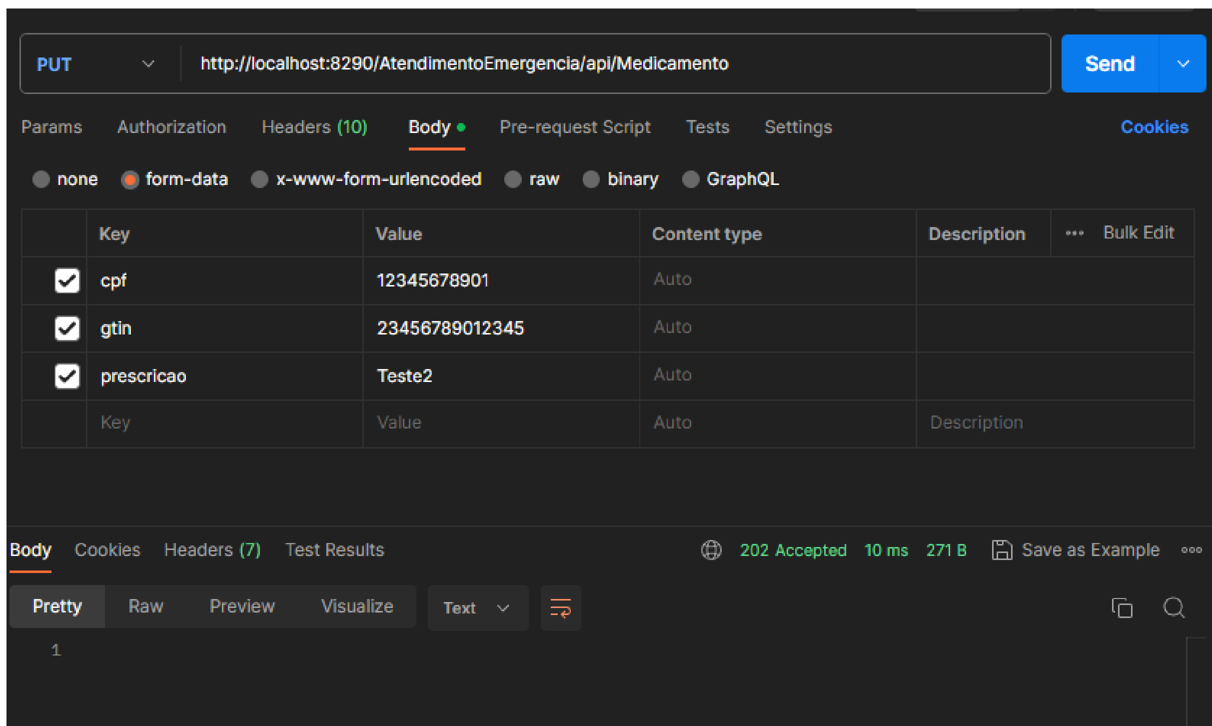

Figura 5.10: Resultado do teste do método *PUT* /Medicamento. Fonte: autores

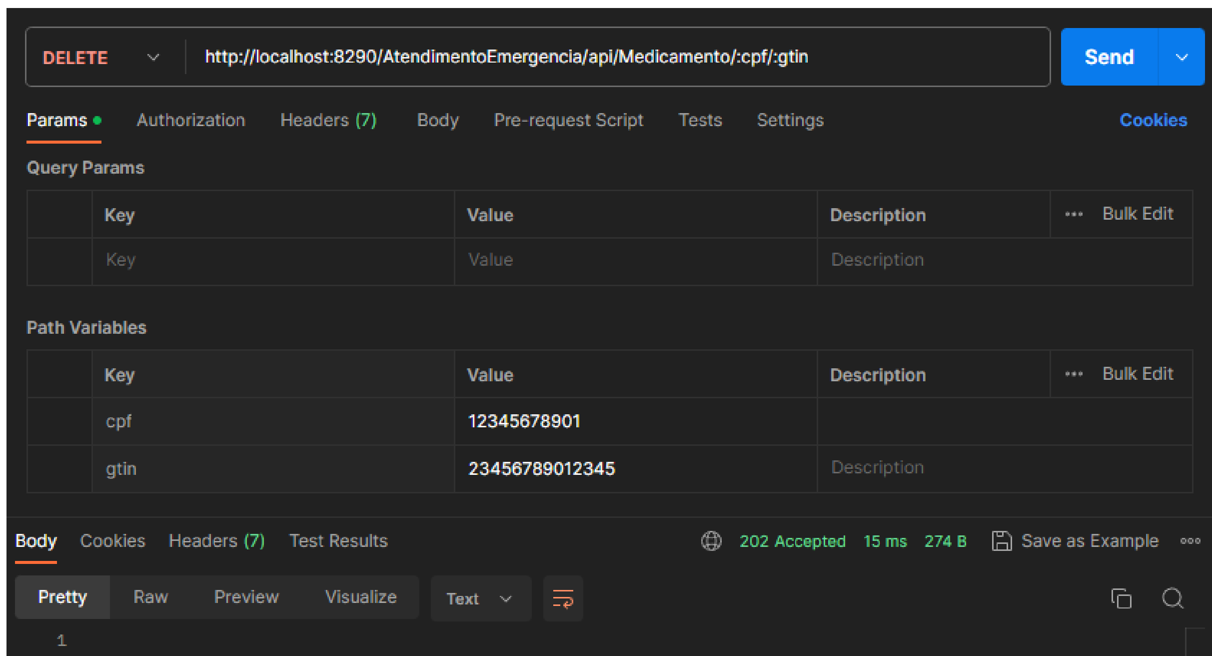

Figura 5.11: Resultado do teste do método *DELETE* /Medicamento/{cpf}/{gtin}. Fonte: autores

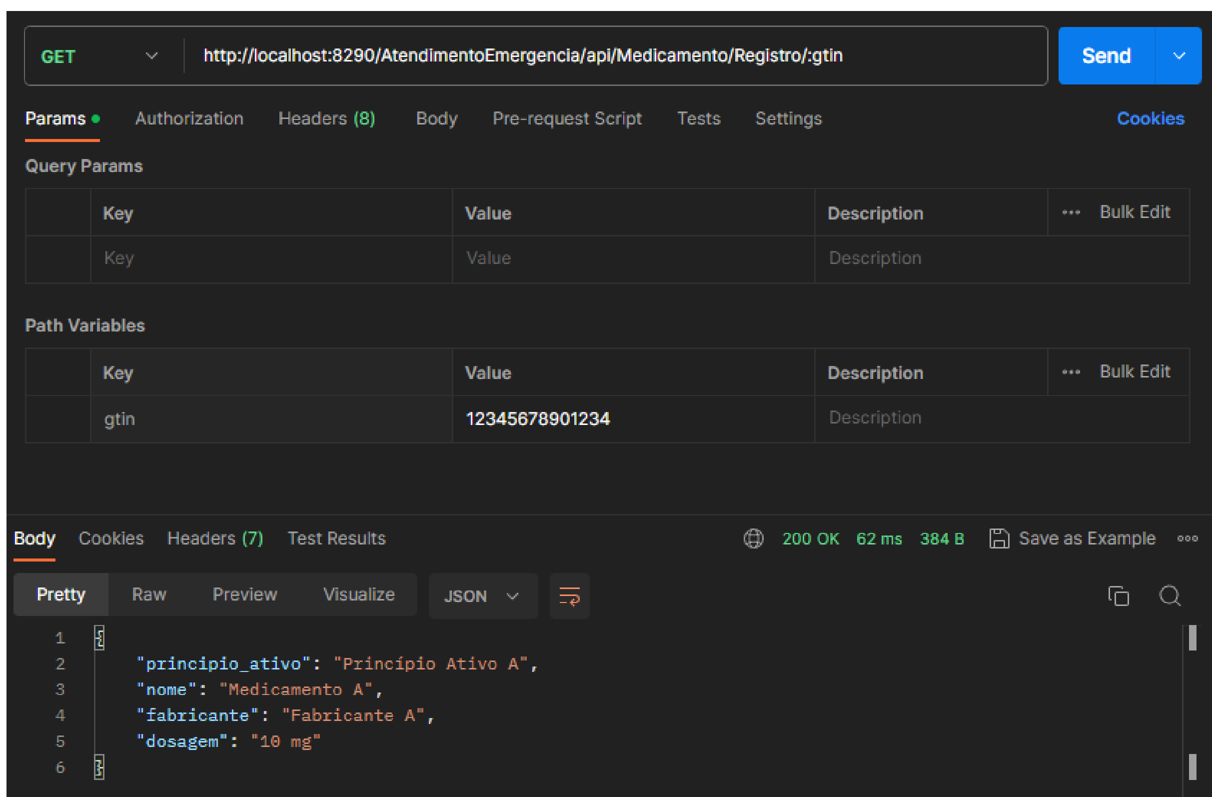

Figura 5.12: Resultado do teste do método *GET* /Medicamento/Registro/{gtin}. Fonte: autores

5.2.3 Recurso Comorbidade

Os resultados dos testes a respeito dos 5 métodos disponíveis para este recurso estão exibidos nas seguintes figuras: 5.13, 5.14, 5.15, 5.16 e 5.17.

Figura 5.13: Resultado do teste do método *GET* /Comorbidade/{cpf}. Fonte: autores

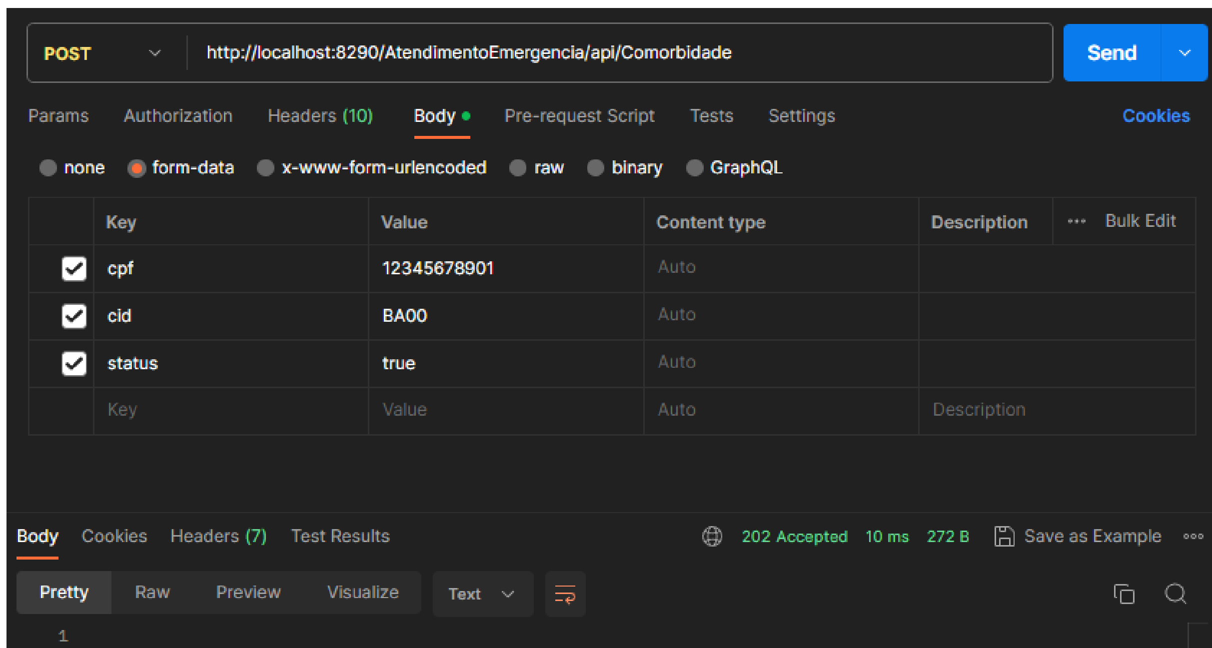

Figura 5.14: Resultado do teste do método *POST* /Comorbidade. Fonte: autores

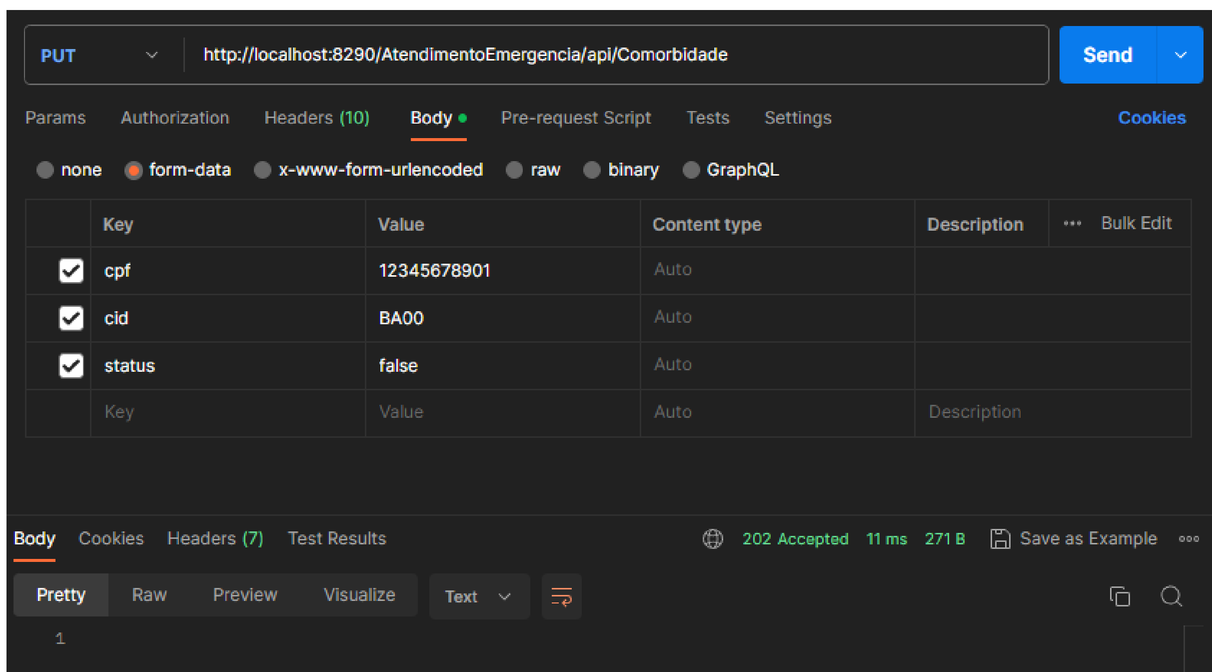

Figura 5.15: Resultado do teste do método *PUT* /Comorbidade. Fonte: autores

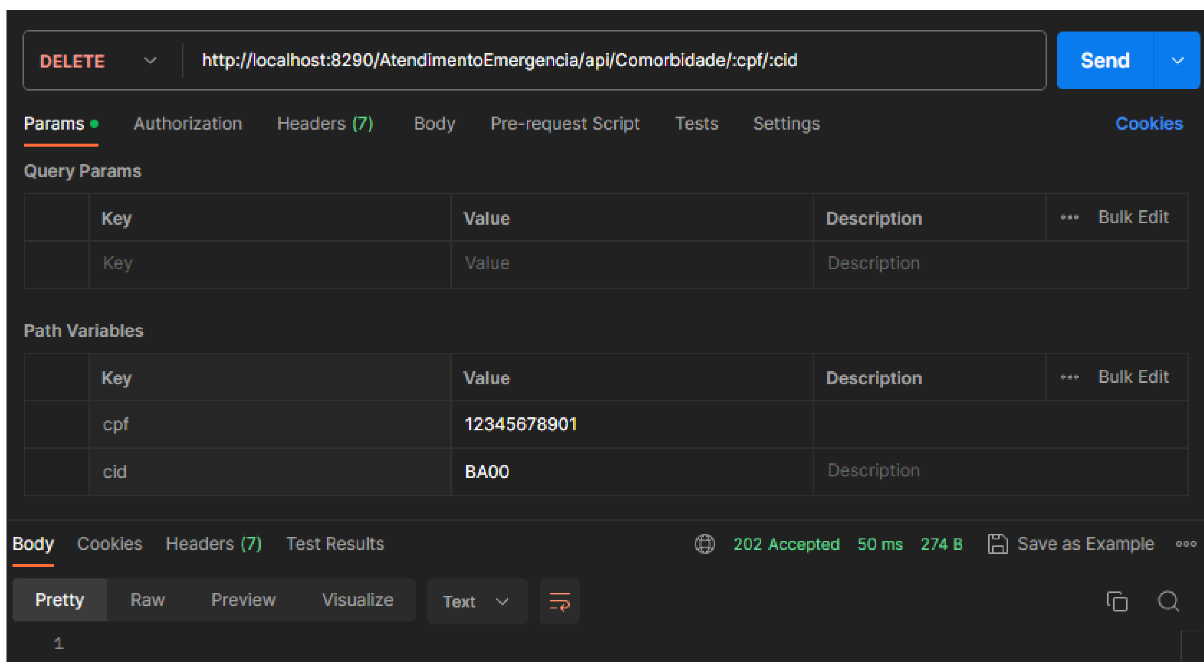

Figura 5.16: Resultado do teste do método *DELETE* /Comorbidade/{cpf}/{cid}. Fonte: autores

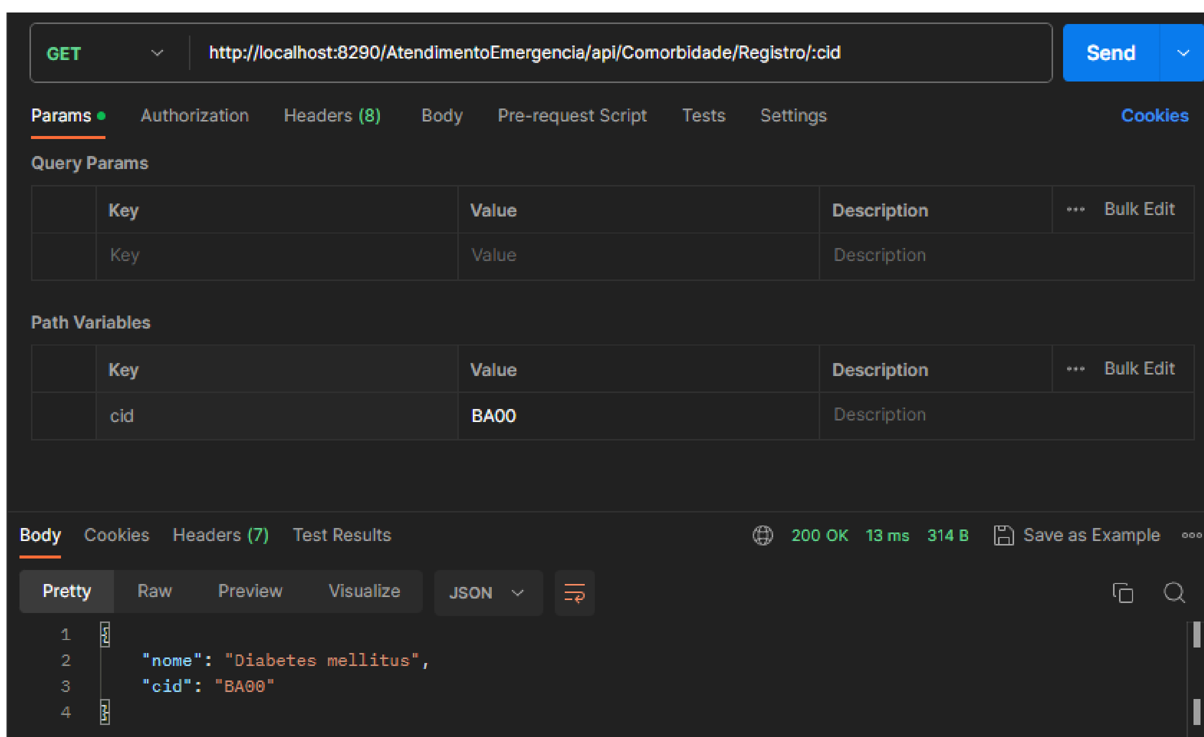

Figura 5.17: Resultado do teste do método *GET* /Comorbidade/Registro/{cid}. Fonte: autores

5.2.4 Análise dos Resultados Obtidos

O *Postman* disponibiliza os dados ao usuário de forma simples e visual, permitindo uma análise tanto das requisições enviadas quanto das respostas HTTP recebidas, como mostrado na seção 3.15 deste docu-

mento.

Dessa forma, a observação das figuras exibidas nas seções anteriores deste capítulo, permite inferir que os resultados se mostraram consistentes e adequados a proposta previamente estabelecida.

Todas as requisições foram efetuadas com sucesso, como visto pelo código "200 OK"no *status* das respostas HTTP. Também é relevante ressaltar que os *payloads* das respostas às requisições *GET*, presentes nas figuras 5.4, 5.8, 5.12, 5.13 e 5.17, seguem o mesmo padrão estabelecido pela definição *Swagger*, mostrada na seção 4.2.3. Finalmente, percebe-se que a latência de resposta do servidor se encontra em dezenas de milissegundos, considerando que a arquitetura foi executada em ambiente local, o que eliminou a interferência de *links* externos. Portanto, o tempo de resposta da API condiz com o objetivo proposto do trabalho.

Assim, a arquitetura conseguiu realizar com êxito todos os objetivos propostos.

6 CONCLUSÃO

O trabalho propôs a implementação de um sistema que conceda o acesso à dados de saúde de um indivíduo, de modo a auxiliar o atendimento médico de emergência. A arquitetura proposta consiste na visualização dos dados do paciente por meio de uma aplicação móvel, com essas informações sendo fornecidas por uma API *RESTful* e armazenadas em um banco de dados relacional.

Os resultados obtidos a partir da realização de teste, conforme mostrado no capítulo 5, validam a implementação da arquitetura, que atingiu todos os objetivos propostos. A API desenvolvida é capaz de fornecer todos os dados necessários, como definido inicialmente, tanto no que tange a integridade dos dados, quanto o tempo de resposta.

No entanto, durante os testes foram observadas algumas limitações da arquitetura:

- **Controle de acesso:** A arquitetura atual não possui qualquer espécie de controle de acesso para consumo da API. Dessa forma, não é possível gerenciar o acesso aos dados.
- **Comunicação criptografada:** O consumo da API é feito via protocolo HTTP, portanto, o conteúdo das mensagens fica exposto. Visto que essas informações são sensíveis, torna-se inviável o uso desse procotolo em aplicação real.

Ademais, os testes foram insuficientes para a averiguação da performance da ferramenta em cenários mais realistas.

Dessa forma propõem-se trabalhos futuros para aprimorar a operação da arquitetura e sua aplicação em contextos de utilização real:

- **Usabilidade da aplicação:** O estudo sobre o uso do aplicativo em ambientes reais, por meio da análise de *feedback* dos profissionais de saúde, pode auxiliar na otimização do aplicativo móvel, tornando-o mais eficiente na execução de sua proposta.
- **Volume de dados:** Como exposto na seção 4.2.5.3, este trabalho utilizou um *hardware* diminuto, com o objetivo de verificar o funcionamento da arquitetura em um volume baixo de dados. Assim, torna-se valiosa uma análise em relação à escalabilidade da solução, verificando a capacidade necessária em *hardware* para utilização do sistema em um ambiente verossímil.
- **Autenticação gov.br:** A autenticação do usuário no aplicativo é realizada pelo *Firebase Authentication*. Para aplicação da solução em ambientes governamentais, como as Forças Armadas, a autenticação *gov.br* traria uma maior credibilidade, aplicada nesse contexto.
- **Implementação do HTTPS:** A operação desse protocolo na arquitetura permite que os dados possam ser transportados pela rede de forma segura, com as suas integridade, autenticidade e confidencialidade garantidas.

- **Testes em ambiente em nuvem:** A realização de testes no ambiente AWS construído possiblita a comparação com os resultados obtidos em ambiente local, observando a performance da arquitetura nesse ambiente.

REFERÊNCIAS BIBLIOGRÁFICAS

1 ALOTAIBI, Y. K.; FEDERICO, F. The impact of health information technology on patient safety. *Saudi medical journal*, Saudi Medical Journal, v. 38, n. 12, p. 1173, 2017.

2 DU, H. Nfc technology: Today and tomorrow. *International Journal of Future Computer and Communication*, IACSIT Press, v. 2, n. 4, p. 351, 2013.

3 KANG, M. H.; LEE, G. J.; YUN, J. H.; SONG, Y. M. Nfc-based wearable optoelectronics working with smartphone application for untact healthcare. *Sensors*, MDPI, v. 21, n. 3, p. 878, 2021.

4 ALRAWAIS, A. Security issues in near field communications (nfc). *International Journal of Advanced Computer Science and Applications*, Science and Information (SAI) Organization Limited, v. 11, n. 11, 2020.

5 EBERE, O.; RAMSURRUN, V.; SEEAM, P.; KATSINA, P.; ANANTWAR, S.; SHARMA, M.; SEEAM, A. Nfc tag-based mhealth patient healthcare tracking system. In: IEEE. *2022 3rd International Conference on Next Generation Computing Applications (NextComp)*. [S.l.], 2022. p. 1–6.

6 LIÉBANA-CABANILLAS, F.; SINGH, N.; KALINIC, Z.; CARVAJAL-TRUJILLO, E. Examining the determinants of continuance intention to use and the moderating effect of the gender and age of users of nfc mobile payments: A multi-analytical approach. *Information Technology and Management*, Springer, v. 22, p. 133–161, 2021.

7 WENDLAND, J.; LUNARDI, G. L.; DOLCI, D. B. Adoption of health information technology in the mobile emergency care service. *RAUSP Management Journal*, SciELO Brasil, v. 54, p. 287–304, 2019.

8 HEUSER, C. A. *Projeto de banco de dados: Volume 4 da Série Livros didáticos informática UFRGS*. [S.l.]: Bookman Editora, 2009.

9 SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Sistema de Banco de Dados*. 5ª. ed. [S.l.]: Campus, 2006.

10 CHEN, P. P.-S. The entity-relationship model—toward a unified view of data. *ACM transactions on database systems (TODS)*, Acm New York, NY, USA, v. 1, n. 1, p. 9–36, 1976.

11 SORDI, J. O. D. *Modelagem de dados-estudos de casos abrangentes da concepção lógica à implementação*. [S.l.]: Saraiva Educação SA, 2019.

12 DATE, C. J. *Introdução a sistemas de bancos de dados*. [S.l.]: Elsevier Brasil, 2004.

13 KUROSE, J. F.; ROSS, K. W. *Redes de computadores e a internet: uma abordagem top-down*. 6ª. ed. [S.l.]: Pearson Education do Brasil, 2013.

14 GOURLEY, D.; TOTTY, B. *HTTP: the definitive guide*. [S.l.]: "O'Reilly Media, Inc.", 2002.

15 BIEHL, M. *API Architecture*. [S.l.]: API-University Press, 2015.

16 MASSE, M. *REST API design rulebook: designing consistent RESTful web service interfaces*. [S.l.]: "O'Reilly Media, Inc.", 2011.

17 FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. [S.l.]: University of California, Irvine, 2000.

18 OMS. *International Statistical Classification of Diseases and Related Health Problems (ICD)*. Disponível em: <https://www.who.int/standards/classifications/classification-of-diseases>.

19 KOLEV, S. Designing a nfc system. In: IEEE. *2021 56th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST)*. [S.l.], 2021. p. 111–113.

20 ALBATTAH, A.; ALGHOFAILI, Y.; ELKHEDIRI, S. Nfc technology: assessment effective of security towards protecting nfc devices & services. In: IEEE. *2020 International Conference on Computing and Information Technology (ICCIT-1441)*. [S.l.], 2020. p. 1–5.

21 CHABBI, S.; MADHOUN, N. E.; KHAMER, L. Security of nfc banking transactions: Overview on attacks and solutions. In: IEEE. *2022 6th Cyber Security in Networking Conference (CSNet)*. [S.l.], 2022. p. 1–5.

22 AMAGAI, S.; PILA, S.; KAAT, A. J.; NOWINSKI, C. J.; GERSHON, R. C. Challenges in participant engagement and retention using mobile health apps: literature review. *Journal of medical Internet research*, JMIR Publications Toronto, Canada, v. 24, n. 4, p. e35120, 2022.

23 JOSHUA, S. R.; ABBAS, W.; LEE, J.-H. M-healthcare model: An architecture for a type 2 diabetes mellitus mobile application. *Applied Sciences*, MDPI, v. 13, n. 1, p. 8, 2022.

24 BRAUNE, K.; LAL, R. A.; PETRUŽELKOVÁ, L.; SCHEINER, G.; WINTERDIJK, P.; SCHMIDT, S.; RAIMOND, L.; HOOD, K. K.; RIDDELL, M. C.; SKINNER, T. C. et al. Open-source automated insulin delivery: international consensus statement and practical guidance for health-care professionals. *The Lancet Diabetes & Endocrinology*, Elsevier, v. 10, n. 1, p. 58–74, 2022.

25 DAVIS, J. J.; FOSTER, S. W.; GRINIAS, J. P. Low-cost and open-source strategies for chemical separations. *Journal of Chromatography A*, Elsevier, v. 1638, p. 461820, 2021.

26 QUEST. *erwin Data Modeler|Industry-Leading Data Modeling Tool*. Disponível em: <https://www.erwin.com/products/erwin-data-modeler/>.

27 POSTGRESQL. *PostgreSQL: The world's most advanced open source database*. Disponível em: <https://www.postgresql.org/>.

28 OPENAI. *Introducing ChatGPT*. Disponível em: <https://openai.com/blog/chatgpt>.

29 SMARTBEAR. *SwaggerHub|API Design Documentation*. Disponível em: <https://swagger.io/tools/swaggerhub/>.

30 WSO2. *Micro Integrator*. Disponível em: <https://wso2.com/micro-integrator/>.

31 AWS. *O que é AWS? Como Funciona Amazon Web Services*. Disponível em: <https://aws.amazon.com/pt/what-is-aws/>.

32 POSTMAN. *What is Postman?* Disponível em: <https://www.postman.com/product/what-is-postman/>.

33 EISENMAN, B. *Learning react native: Building native mobile apps with JavaScript*. [S.l.]: "O'Reilly Media, Inc.", 2015.

34 PRESCOTT, P. *Programação em JavaScript*. [S.l.]: Babelcube Inc., 2016.

35 DEVELOPERS, A. *Conhecer o Android Studio*. Disponível em: <https://developer.android.com/studio/intro?hl=pt-br>.

36 KHAWAS, C.; SHAH, P. Application of firebase in android app development-a study. *International Journal of Computer Applications*, v. 179, n. 46, p. 49–53, 2018.