



UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO  
PRÓ-REITORIA DE GRADUAÇÃO  
DEPARTAMENTO DE CIÊNCIAS EXATAS E TECNOLOGIA DA INFORMAÇÃO  
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

JÚLIO GOMES DA SILVA NETO

**API PARA AUXILIAR OS CLIENTES LOJISTAS DE EMPRESAS DE  
E-COMMERCE NA SELEÇÃO DE ENDEREÇOS DE ENTREGA DE PRODUTOS**

ANGICOS

2023

JÚLIO GOMES DA SILVA NETO

**API PARA AUXILIAR OS CLIENTES LOJISTAS DE EMPRESAS DE  
E-COMMERCE NA SELEÇÃO DE ENDEREÇOS DE ENTREGA DE PRODUTOS**

Trabalho de conclusão de curso  
apresentado à Universidade Federal Rural  
do Semi-Árido como requisito para  
obtenção do título de Bacharel em  
Sistemas de Informação.

Orientador:  
Prof. Dr. Araken de Medeiros Santos

Coorientador:  
Prof. Me. Serafim do Nascimento Junior

ANGICOS

2023

© Todos os direitos estão reservados a Universidade Federal Rural do Semi-Árido. O conteúdo desta obra é de inteira responsabilidade do (a) autor (a), sendo o mesmo, passível de sanções administrativas ou penais, caso sejam infringidas as leis que regulamentam a Propriedade Intelectual, respectivamente, Patentes: Lei nº 9.279/1996 e Direitos Autorais: Lei nº 9.610/1998. O conteúdo desta obra tomar-se-á de domínio público após a data de defesa e homologação da sua respectiva ata. A mesma poderá servir de base literária para novas pesquisas, desde que a obra e seu (a) respectivo (a) autor (a) sejam devidamente citados e mencionados os seus créditos bibliográficos.

SS586 Silva Neto, Júlio Gomes da.  
a API para Auxiliar os Clientes Lojistas de  
Empresas de E-commerce na Seleção de Endereços de  
Entrega de Produtos / Júlio Gomes da Silva Neto. -  
2023.  
78 f. : il.

Orientador: Araken de Medeiros Santos.  
Coorientador: Serafim do Nascimento Junior.  
Monografia (graduação) - Universidade Federal  
Rural do Semi-árido, Curso de Sistemas de  
Informação, 2023.

1. E-commerce. 2. Loja virtual. 3. Delivery.  
4. Lojistas. 5. Software. I. de Medeiros Santos,  
Araken, orient. II. do Nascimento Junior,  
Serafim, co-orient. III. Título.

Ficha catalográfica elaborada por sistema gerador automático em conformidade  
com AACR2 e os dados fornecidos pelo(a) autor(a).

Biblioteca Campus Angicos  
Bibliotecário: Helder Romero Maia Duarte  
CRB: 15/673

O serviço de Geração Automática de Ficha Catalográfica para Trabalhos de Conclusão de Curso (TCC's) foi desenvolvido pelo Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (USP) e gentilmente cedido para o Sistema de Bibliotecas da Universidade Federal Rural do Semi-Árido (SISBI-UFERSA), sendo customizado pela Superintendência de Tecnologia da Informação e Comunicação (SUTIC) sob orientação dos bibliotecários da instituição para ser adaptado às necessidades dos alunos dos Cursos de Graduação e Programas de Pós-Graduação da Universidade.

JÚLIO GOMES DA SILVA NETO

**API PARA AUXILIAR OS CLIENTES LOJISTAS DE EMPRESAS DE  
E-COMMERCE NA SELEÇÃO DE ENDEREÇOS DE ENTREGA DE PRODUTOS**

Trabalho de conclusão de curso  
apresentado à Universidade Federal Rural  
do Semi-Árido como requisito para  
obtenção do título de Bacharel em  
Sistemas de Informação.

Defendida e aprovada em **17 / 05 / 2023**.

**BANCA EXAMINADORA**

Documento assinado digitalmente



ARAKEN DE MEDEIROS SANTOS

Data: 19/05/2023 09:02:18-0300

Verifique em <https://validar.itl.gov.br>

---

Prof. Araken de Medeiros Santos (UFERSA)  
Doutor em Ciência da Computação  
Presidente e Orientador

Documento assinado digitalmente



SERAFIM DO NASCIMENTO JUNIOR

Data: 18/05/2023 01:56:57-0300

Verifique em <https://validar.itl.gov.br>

---

Prof. Serafim do Nascimento Junior (IMD-UFRN)  
Mestre em Ciência da Computação  
Coorientador

Documento assinado digitalmente



RAFAEL JULLIAN OLIVEIRA DO NASCIMENTO

Data: 18/05/2023 20:22:52-0300

Verifique em <https://validar.itl.gov.br>

---

Prof. Rafael Jullian Oliveira do Nascimento (UNIPÊ)  
Mestre em Ciência da Computação  
Membro Examinador Externo

Documento assinado digitalmente



JAIME BRUNO CIRNE DE OLIVEIRA

Data: 18/05/2023 20:43:05-0300

Verifique em <https://validar.itl.gov.br>

---

Coordenador de TI Jaime Bruno Cirne de Oliveira (UFRN)  
Mestre em Tecnologia da Informação  
Membro Examinador Externo

*Dedico este trabalho a meu pai, José do Patrocínio Gomes (in memoriam). Obrigado por seus ensinamentos, conselhos e princípios, os quais permanecem sendo os pilares da minha conduta. Além do incentivo aos estudos, sua sabedoria me orientou em meio às dificuldades e incertezas da vida.*

## **AGRADECIMENTOS**

Agradeço primeiramente a Deus e ao conjunto de forças inexplicáveis que sempre me ajudaram nos momentos mais atribulados.

Agradeço aos meus pais, José do Patrocínio Gomes e Raimunda Neta da Silva Gomes, por me educarem com carinho e dedicação. Também pelo apoio durante toda a minha jornada acadêmica, desde a minha alfabetização até o presente grau almejado.

Agradeço ao meu orientador de Projeto de TCC e coorientador no TCC Serafim do Nascimento Junior e ao meu orientador de TCC Araken de Medeiros Santos pelo acompanhamento e tempo dedicado na orientação rumo a conclusão deste de TCC.

Agradeço ao meu colega de firma e amigo Isaac Franco pelo direcionamento durante o desenvolvimento deste trabalho, contribuindo tanto na concepção de ideias, quanto na orientação da parte mais técnica.

Agradeço às pessoas que contribuíram com esta pesquisa, seja através de questionários ou de conversas informais que me guiaram durante o processo.

Agradeço à instituição UFERSA e aos prezados professores do curso de Bacharelado em Sistemas de Informação pela oportunidade de desenvolvimento acadêmico e profissional.

*"O tempo é o melhor autor, sempre  
encontra o final perfeito"*

(Charles Chaplin)

## RESUMO

O comércio eletrônico, ou *e-commerce*, é uma atividade econômica que está em crescimento no Brasil e no mundo. Nesse contexto, uma *startup* do estado brasileiro do Rio do Grande do Norte está desenvolvendo uma plataforma Web com o intuito de facilitar a criação e a configuração de lojas virtuais, contribuindo para viabilizar a entrada ou a adaptação de negócios no *e-commerce*. No entanto, os usuários lojistas do sistema reclamaram da dificuldade ao realizar o cadastro das regiões de entrega de encomendas via motoboy ou *delivery*. Esse obstáculo na seleção dos locais de entrega impossibilita que, no momento da compra, os clientes escolham onde receber seus pedidos, que é uma das principais vantagens ao se adquirir bens ou serviços pela *internet*. Portanto, o presente trabalho objetivou projetar e desenvolver uma aplicação que auxiliasse os lojistas, usuários de plataformas de *e-commerce*, no cadastro e seleção de regiões de entrega para seus produtos. Para isso, a metodologia utilizada foi composta por pesquisas bibliográficas e de campo, além da coleta de dados e de métodos da Engenharia de *Software*. Ao final do trabalho, foi possível projetar uma base de dados com cidades, bairros, logradouros e CEPs de todo o Brasil, além de implementar uma API *REST* documentada que pode ser integrada por plataformas de *e-commerce* para auxiliar a seleção e o cadastro dos locais de entrega.

**Palavras-chave:** *E-commerce*. Loja virtual. *Delivery*. Lojistas. *Software*.

## ABSTRACT

Electronic commerce, or e-commerce, is an economic activity that is growing in Brazil and in the world. In this context, a startup from the Brazilian state of Rio Grande do Norte is developing a Web platform that has the intention of facilitating the creation and configuration of virtual stores, contributing to enable the entrance or adaptation in e-commerce businesses. However, the shopkeepers using the system complained about the difficulty in registering order delivery regions. This obstacle in selecting delivery locations makes it impossible for, in checkout, the clients choose where to receive their orders, which is one of the main advantages of buying goods and services through the internet. Therefore, this present work aimed to project and to develop a standalone application to help the shopkeepers, e-commerce platform users, in registration and selection of order delivery regions for their products. For that, the methodology employed in this study consisted of bibliographic and field research, as well as data collection and Software Engineering methods. As a result of this work, it was possible to design a database with cities, neighborhoods, streets, and zip codes throughout Brazil. Additionally, a documented REST API was implemented, which can be integrated by e-commerce platforms to assist in the selection and registration of delivery locations.

**Keywords:** E-commerce. Virtual store. Delivery. Shopkeepers. API *REST*.

## LISTA DE TABELAS

|   |   |                                                                      |    |
|---|---|----------------------------------------------------------------------|----|
| 1 | – | Descrição da tarefa criada no <i>backlog</i> da <i>startup</i> ..... | 30 |
| 2 | – | Requisitos funcionais .....                                          | 47 |
| 3 | – | Requisitos não funcionais .....                                      | 47 |
| 4 | – | Rotas especiais de ceps da API .....                                 | 57 |
| 5 | – | Rotas da API .....                                                   | 58 |

## LISTA DE FIGURAS

|    |                                                                                             |    |
|----|---------------------------------------------------------------------------------------------|----|
| 1  | – Cadastro de região de entrega na plataforma da <i>startup</i> potiguar .....              | 16 |
| 2  | – Exemplo API <i>REST</i> em ação .....                                                     | 26 |
| 3  | – Formulário para cadastro de regiões de entrega Loja Integrada .....                       | 33 |
| 4  | – Formulário de cadastro de zonas de entrega .....                                          | 33 |
| 5  | – Cadastro e exibição de faixas de CEP para entrega .....                                   | 34 |
| 6  | – Resultado da busca por Faixa de CEP ou UF/Localidade - Correios .....                     | 35 |
| 7  | – Diagrama Modelo ER conceitual do banco de dados .....                                     | 49 |
| 8  | – Modelo lógico do banco de dados .....                                                     | 51 |
| 9  | – SQL para a criação do banco de dados "ceps" .....                                         | 52 |
| 10 | – SQL para a criação da tabela "cidades" .....                                              | 53 |
| 11 | – SQL para a criação da tabela "bairros" .....                                              | 53 |
| 12 | – SQL para a criação da tabela "logradouros" .....                                          | 54 |
| 13 | – SQL para a criação da tabela "plataformas" .....                                          | 54 |
| 14 | – Exemplo do formato <i>JSON</i> utilizado na API .....                                     | 59 |
| 15 | – Código-fonte para a consulta de uma cidade por CEP .....                                  | 60 |
| 16 | – Código-fonte para a consulta de bairros por CEP no <i>Insomnia</i> .....                  | 61 |
| 17 | – Código-fonte para a consulta de logradouros de um bairro por CEP no <i>Insomnia</i> ..... | 62 |
| 18 | – Teste de cadastro de plataforma usando o <i>Insomnia</i> .....                            | 63 |
| 19 | – Erro ao tentar cadastrar uma nova plataforma com o mesmo e-mail .....                     | 64 |
| 20 | – Obter dados de uma plataforma por e-mail e senha no <i>Insomnia</i> .....                 | 64 |
| 21 | – Teste consultando cidade por CEP de um logradouro no <i>Insomnia</i> .....                | 65 |
| 22 | – Consulta de todos os bairros de uma cidade por CEP .....                                  | 65 |
| 23 | – Consulta de todos os logradouros de um bairro por CEP no <i>Insomnia</i> .....            | 66 |
| 24 | – Documentação da API com <i>Swagger</i> .....                                              | 68 |
| 25 | – Documentação da criação de nova plataforma .....                                          | 68 |
| 26 | – Documentação das respostas da API para criar nova plataforma .....                        | 69 |
| 27 | – Exemplo de utilização da API para seleção de regiões de entrega .....                     | 70 |

## LISTA DE ABREVIATURAS E SIGLAS

|      |                                          |
|------|------------------------------------------|
| API  | <i>Application Programming Interface</i> |
| B2B  | <i>Business to Business</i>              |
| B2C  | <i>Business to Consumer</i>              |
| C2C  | <i>Consumer to Consumer</i>              |
| CEP  | Código de Endereçamento Postal           |
| HTTP | <i>Hipertext Transfer Protocol</i>       |
| REST | <i>Representational State Transfer</i>   |
| SDK  | <i>Software Development Kit</i>          |
| SQL  | <i>Structured Query Language</i>         |

## SUMÁRIO

|                                                    |           |
|----------------------------------------------------|-----------|
| <b>1 INTRODUÇÃO.....</b>                           | <b>14</b> |
| 1.1 Contexto.....                                  | 14        |
| 1.2 Motivação.....                                 | 16        |
| 1.3 Objetivos.....                                 | 17        |
| 1.4 Metodologia.....                               | 18        |
| 1.5 Organização do Trabalho.....                   | 18        |
| <b>2 REFERENCIAL TEÓRICO.....</b>                  | <b>19</b> |
| 2.1 E-commerce.....                                | 19        |
| 2.1.1 Origem do e-commerce.....                    | 19        |
| 2.1.2 Tipos de e-commerce.....                     | 20        |
| 2.2 Lojas virtuais e marketplaces.....             | 21        |
| 2.3 Código de Endereçamento Postal (CEP).....      | 22        |
| 2.4 Logradouros e endereços.....                   | 23        |
| 2.5 Delivery e entrega de encomendas.....          | 23        |
| 2.6 Application Programming Interface (API).....   | 24        |
| 2.7 API REST.....                                  | 25        |
| <b>3 METODOLOGIA.....</b>                          | <b>27</b> |
| 3.1 Pesquisa Bibliográfica.....                    | 27        |
| 3.2 Pesquisa de Campo.....                         | 28        |
| 3.3 Coleta de Dados.....                           | 30        |
| 3.3.1 Observação assistemática.....                | 31        |
| 3.3.2 Faixas de CEP e regiões de entrega.....      | 32        |
| 3.4 Desenvolvimento da aplicação.....              | 36        |
| 3.4.1 Engenharia de Software.....                  | 36        |
| 3.4.1.1 Análise e especificação de requisitos..... | 37        |
| 3.4.1.2 Projeto e implementação da aplicação.....  | 38        |
| 3.4.1.3 Validação da aplicação.....                | 39        |
| 3.4.1.4 Evolução da aplicação.....                 | 39        |

|                                            |           |
|--------------------------------------------|-----------|
| 3.4.2 Desenvolvimento ágil.....            | 40        |
| 3.4.3 Tecnologias.....                     | 41        |
| 3.4.3.1 Node.js.....                       | 42        |
| 3.4.3.2 Express.js.....                    | 42        |
| 3.4.3.3 MySQL.....                         | 43        |
| 3.4.3.4 MySQL Workbench.....               | 43        |
| 3.4.3.5 Knex.js.....                       | 44        |
| 3.4.3.6 Swagger.....                       | 44        |
| 3.4.3.7 Insomnia.....                      | 45        |
| 3.4.3.8 GIT e versionamento de código..... | 45        |
| <b>4 DESENVOLVIMENTO DA API.....</b>       | <b>46</b> |
| 4.1 Requisitos funcionais.....             | 46        |
| 4.2 Requisitos não funcionais.....         | 47        |
| 4.3 Modelagem de dados.....                | 48        |
| 4.3.1 O modelo conceitual.....             | 49        |
| 4.3.2 O modelo lógico.....                 | 50        |
| 4.3.2 O modelo físico.....                 | 52        |
| 4.3.3 Construção da base de dados.....     | 55        |
| 4.4 Implementação da API.....              | 56        |
| 4.4.1 Rotas.....                           | 56        |
| 4.4.2 Formato de resposta.....             | 58        |
| 4.4.3 Codificação da API.....              | 59        |
| 4.5 Testes.....                            | 62        |
| 4.6 Documentação do software.....          | 66        |
| 4.7 Validação de uso da API.....           | 69        |
| <b>5 CONSIDERAÇÕES FINAIS.....</b>         | <b>71</b> |
| 5.1 Conclusões.....                        | 71        |
| 5.2 Trabalhos futuros.....                 | 72        |
| <b>REFERÊNCIAS.....</b>                    | <b>73</b> |

|                                                                                                                         |           |
|-------------------------------------------------------------------------------------------------------------------------|-----------|
| <b>APÊNDICE - Script na linguagem Python para transformar o arquivo do Anexo A em uma base de dados relacional.....</b> | <b>76</b> |
| <b>ANEXO - Estrutura do arquivo de texto utilizado como base de dados.....</b>                                          | <b>78</b> |

## 1 INTRODUÇÃO

Neste capítulo introdutório encontra-se uma breve apresentação deste TCC, dividindo as seções do trabalho da seguinte maneira: a Seção 1.1 apresenta o contexto do trabalho; a Seção 1.2 evidencia a motivação e a proposta que será abordada; a Seção 1.3 descreve os objetivos pretendidos; a Seção 1.4 expressa, de forma resumida, a metodologia que será aplicada no decorrer do projeto; finalmente, a Seção 1.5 traz a estrutura de organização do trabalho.

### 1.1 Contexto

O comércio eletrônico, ou *e-commerce*, transformou o modo como se adquire produtos e serviços, seja através de dispositivos eletrônicos ou de plataformas virtuais. O processo de compra vem se tornando mais prático e cômodo para o consumidor, enquanto os lojistas têm a oportunidade de expandir o alcance de seus negócios. Segundo Fagundes (2009), “o comércio eletrônico através da internet é o ramo de atividade econômica que mais cresce no mundo”.

Esse crescimento é evidenciado por meio dos dados da *Neotrust* (2022), uma empresa que monitora cerca de 85% do *e-commerce* no Brasil, os quais ressaltam que o comércio virtual cresceu 21% em janeiro de 2022 comparando-se ao mesmo período do ano anterior.

Todavia, entrar no mercado de vendas *online* nem sempre foi algo viável, especialmente para pequenas empresas ou vendedores iniciantes. Segundo relatos da pesquisa de campo, seja por falta de recursos financeiros para implementar a infraestrutura necessária ou pela dificuldade de utilizar as tecnologias, muitos desistiram dessa possibilidade. Nesse contexto, surgiram diversas *startups* com o intuito de viabilizar a criação e o gerenciamento de lojas virtuais de maneira simplificada e relativamente barata para o usuário lojista.

Em Natal, no Rio Grande do Norte, surgiu uma *startup* formada por desenvolvedores de *software* e especialistas em *e-commerce* que decidiram lançar uma plataforma virtual neste segmento, mais precisamente um *marketplace*. Um sistema como esse tem o propósito de oportunizar aos lojistas de todo o Brasil construir sua loja virtual, cadastrar seus produtos, configurar meios de pagamento e entrega, além de acompanhar suas vendas.

Ao criar sua loja em um *marketplace* ou plataforma similar, o usuário lojista deve configurar, dentre outros, os métodos de entrega, que serão exibidos para os clientes escolherem no momento da compra. Os Correios e as transportadoras são exemplos de serviços amplamente utilizados.

Porém, uma alternativa mais barata e simples para vendedores é utilizar a entrega via motoboy, ou *delivery*, como também é chamado. Trata-se de um serviço de entrega mais regional, para atender somente determinados bairros ou ruas dentro de um raio de alcance.

Para pequenos e médios negócios que atendem a uma faixa restrita de localidades, utilizar motoboys ou táxis para realizar entregas pode ser mais viável do que outros serviços. Assim sendo, o vendedor precisa informar no sistema quais regiões, ou faixas de CEP, ele irá atender, quanto irá custar, qual o limite de peso e o prazo estimado. Dessa forma, na finalização da compra, o comprador poderá escolher a opção de entrega que mais lhe agrada.

No entanto, durante o desenvolvimento desse produto de *software* e, tendo como base os testes com os primeiros clientes, alguns problemas foram identificados justamente na configuração do método de entrega *delivery*. Ao passo em que uns reportaram a insuficiência de clareza no momento de inserir as informações, outros afirmaram desconhecer as localizações de entrega.

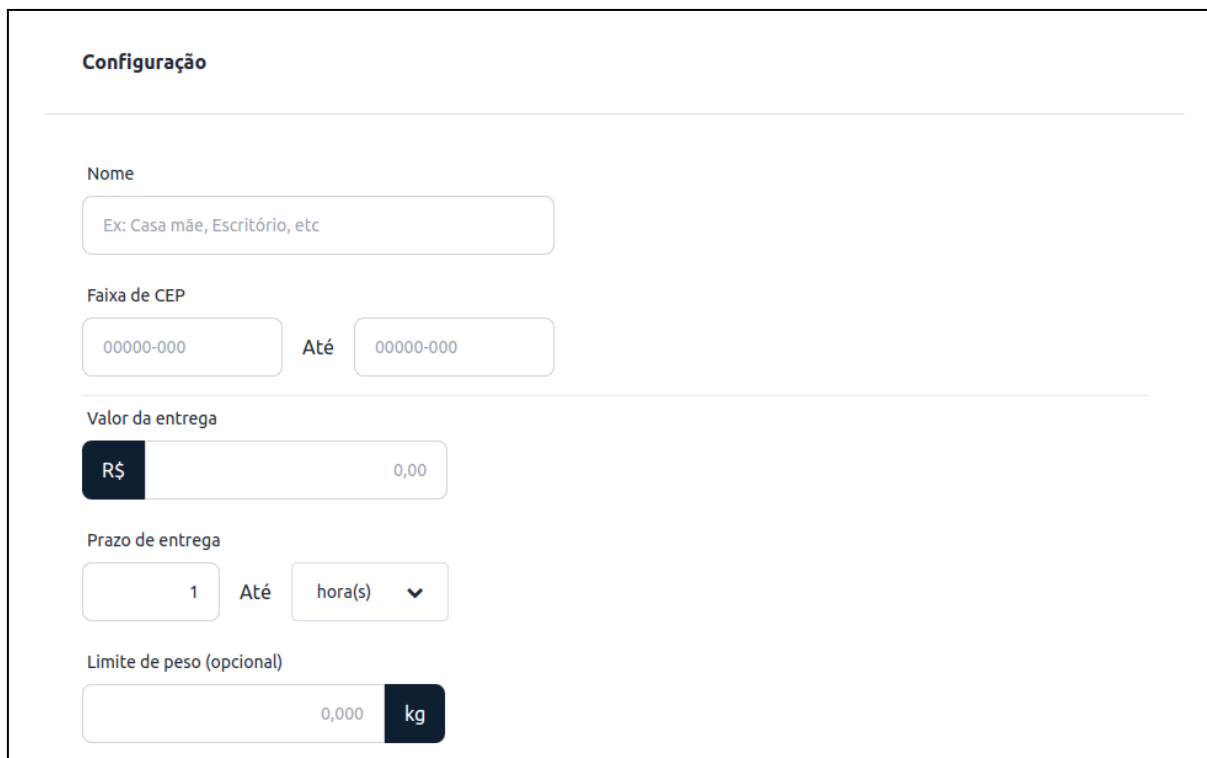
É possível observar na Figura 1, que demonstra o formulário de cadastro de regiões de entrega na *startup* norte-rio-grandense, as dificuldades reportadas pelos testadores. A principal dificuldade está em compreender, pesquisar e identificar as faixas de CEP para a região desejada. Muitas dúvidas surgiram e esse incômodo travou o fluxo de configuração.

Vale ressaltar que, enquanto para o consumidor uma das principais vantagens de comprar através da internet é a comodidade de receber seu pedido em casa, para o lojista isso pode se tornar um desafio, tanto de planejamento quanto de logística. Por exemplo, conhecer faixas de cep, zonas regionais, realizar cálculos de prazo e custo, não são tarefas simples e se tornam um obstáculo para a realização da entrega no destino correto.

Como consequência, se uma área de entrega não for configurada corretamente, seja pela dificuldade imposta pelo sistema ou falta de compreensão do usuário, situações inconvenientes e reclamações serão geradas com frequência.

Esse problema pode afetar negativamente a visão que os clientes possuem da plataforma, assim como também a credibilidade do vendedor.

**Figura 1** - Formulário de cadastro de região de entrega na plataforma da *startup* potiguar.



**Configuração**

---

Nome

Ex: Casa mãe, Escritório, etc

Faixa de CEP

00000-000 Até 00000-000

---

Valor da entrega

R\$ 0,00

Prazo de entrega

1 Até hora(s) ▼

Limite de peso (opcional)

0,000 kg

**Fonte:** *Startup* de Natal/RN estudada neste trabalho (2022).

Ao buscar soluções para resolver o empecilho de inserir essas informações no sistema, foi verificado pelos desenvolvedores da *startup* citada que as alternativas existentes, inclusive as presentes em outras empresas, são pouco intuitivas, geram muitas dúvidas e dificultam o cadastro dos locais de entrega via *delivery*. Por conseguinte, surgiu a necessidade de desenvolver e implementar uma nova solução que fosse mais eficaz, fácil de usar e capaz de ser integrada a essa e a outras plataformas.

## 1.2 Motivação

A partir da problemática apresentada na seção anterior, esse trabalho foi motivado pela dificuldade que os usuários lojistas enfrentam ao cadastrar regiões de

entrega em plataformas de *e-commerce*. Ao criar suas lojas virtuais através desses sistemas, um dos aspectos mais importantes é a entrega dos produtos. Essa função precisa estar bem configurada com as localidades corretas, caso contrário, irá impossibilitar a realização de vendas e impactar negativamente a empresa.

Nesse sentido, a responsabilidade desse trabalho foi pesquisar, avaliar, desenvolver e implementar um serviço que possa ser incorporado em outros sistemas e que disponibilize dados de regiões para as plataformas com o intuito de descomplicar o processo de inserção e configuração dos possíveis locais de entrega para as encomendas.

### 1.3 Objetivos

Abaixo estão listados os objetivos pretendidos por este trabalho, tanto a ideia central, quanto o passo a passo para alcançá-la.

- **Objetivo geral**

- Desenvolver uma API para auxiliar os clientes lojistas de empresas de e-commerce na seleção de endereços de entrega de produtos.

- **Objetivos específicos**

- Analisar a viabilidade da implementação da aplicação sistemas multiplataforma;
- Identificar as tecnologias necessárias para a implementação desse serviço;
- Construir uma base de dados com os ceps, cidades, bairros, logradouros e dados necessários para o desenvolvimento da API;
- Mapear as possíveis regiões de entrega dos produtos a partir de um endereço de origem;
- Desenvolver a aplicação utilizando as tecnologias definidas;
- Desenvolver uma documentação de *software* para auxiliar as plataformas na integração; e
- Validar a implementação da API nos sistemas onde foi implantada.

## 1.4 Metodologia

Como forma de alcançar os objetivos estabelecidos neste trabalho, os procedimentos metodológicos utilizados foram organizados em duas fases, sendo a primeira mais focada na pesquisa e investigação, enquanto a segunda se concentra na implementação e codificação da aplicação.

Inicialmente, a pesquisa bibliográfica foi aplicada para embasar o trabalho e garantir uma compreensão melhor da problemática e seu contexto, fazendo uso de artigos científicos, de livros, de sistemas *Web* e sites da *internet*.

Ademais, a pesquisa de campo realizada na *startup* norte-rio-grandense, citada anteriormente, foi fundamental para levantar dados relevantes e justificar o desenvolvimento deste trabalho. Por meio de testes e análise das soluções existentes, foi possível projetar melhor a aplicação que irá auxiliar as plataformas de lojas virtuais no processo de cadastro de locais para entrega de encomendas.

A segunda fase foi pautada pelo uso de métodos da Engenharia de *Software*, tais como a análise e especificação de requisitos, a elaboração do projeto de *software*, a codificação e, por fim, a manutenção ou evolução do sistema. O uso dessas técnicas ajudou como um guia para o desenvolvimento da aplicação final, colaborando para os prazos de entrega e a qualidade do projeto.

## 1.5 Organização do Trabalho

Este projeto está organizado de tal maneira que: o Capítulo 1 expressou a introdução do trabalho apresentando o contexto, a motivação, os objetivos e uma breve metodologia; o Capítulo 2 apresenta a fundamentação teórica, buscando definir os principais conceitos que foram relevantes para o contexto deste trabalho; o Capítulo 3 apresenta a metodologia de maneira mais detalhada; o Capítulo 4 mostra todo o processo de desenvolvimento da solução; e, por fim, o Capítulo 5 expõe as considerações finais e sugestões de trabalhos futuros.

## 2 REFERENCIAL TEÓRICO

Este capítulo é composto por seções que visam fundamentar esta pesquisa. A seção 2.1 apresentará o *e-commerce*, sua origem e características. Em seguida, a seção 2.2 trata da influência das lojas virtuais e *marketplaces* no desenvolvimento do comércio eletrônico. A seção 2.3 aborda o significado de CEP. A seção 2.4 trata do conceito de logradouro e endereçamento. A seção 2.5 fala a respeito do *delivery* e da entrega de encomendas. A seção 2.6 exhibe os conceitos de uma API. Por fim, a última seção, 2.7, caracteriza o padrão arquitetural *REST*.

### 2.1 E-commerce

O *e-commerce*, ou comércio eletrônico, é "o negócio de compra e venda de bens e serviços na *internet*", de acordo com o dicionário *online* Cambridge (2022). No entanto, antes do surgimento da rede mundial de computadores, essa relação entre comprador e vendedor ocorria da forma tradicional, exigindo que o consumidor se deslocasse até a loja física para adquirir um produto de seu interesse.

O *e-commerce* permite com que os consumidores façam transações por meios eletrônicos a qualquer momento, independente do dia, horário e lugar. Outra vantagem incrível desse modelo é tornar possível que pequenos comerciantes atendam a uma quantidade significativa de consumidores perto ou longe de sua localidade. (DE MENDONÇA, p 241, 2016).

Portanto, o comércio eletrônico transformou a maneira como a sociedade se comporta por intermédio das facilidades que essa modalidade oferece, dando mais oportunidades a pequenos comerciantes e expandindo seu alcance de venda.

#### 2.1.1 Origem do e-commerce

De acordo com De Mendonça (2016), até a década de 70, as práticas comuns de venda eram realizadas presencialmente, porta a porta e telefone. Até então, essas eram as principais formas pelas quais os lojistas entravam em contato com seus consumidores. Vale mencionar também os anúncios nos jornais e revistas, que eram comuns nessa época.

A partir do nascimento da *internet*, na década de 1960, o cenário começou a se transformar e novas oportunidades se originaram através da rede. Todavia, essa inovação tecnológica não foi desenvolvida especificamente para o comércio. Por isso que, somente em março de 1991, segundo Ramos (2015), se deu o marco inicial da *internet* comercial, quando as leis norte-americanas passaram a permitir o tráfego comercial na *internet*.

Existem divergências sobre qual foi o primeiro produto comercializado pela *internet*. De acordo com dados levantados por De Mendonça (2016), a primeira venda foi registrada por volta de 1994 pela Pizza Hut, empresa norte-americana. Mas segundo o Exame (2015), o consenso é de que o primeiro produto a ser vendido foi um CD do cantor Sting, em 11 de agosto de 1994 através do site *NetMarket*.

Ainda no início do processo de desenvolvimento dessa atividade mercantil, havia muita propaganda e campanhas publicitárias com o intuito de impulsionar a adesão de empresas na *internet*. Esse ritmo acelerado culminou no surgimento de muitos negócios *online* (RAMOS, 2015).

Nesse período, começaram a surgir ferramentas e plataformas que se tornaram importantes para a popularização do comércio eletrônico, como a hospedagem de *sites*. Esses instrumentos, aliados ao desenvolvimento da indústria de *software* e dos meios de comunicação, ajudaram a democratizar a entrada de novas lojas no comércio eletrônico, apesar das barreiras impostas.

Embora o início do comércio eletrônico não tenha sido fácil, seja por falta de recursos ou conhecimento sobre as novas tecnologias ou ainda pela dificuldade de se estabelecer confiança e credibilidade em algo novo, com o passar do tempo, muitos negócios foram surgindo e se adaptando à nova realidade econômica.

### **2.1.2 Tipos de e-commerce**

Em artigo do Sebrae (2016), o *e-commerce* pode ser dividido de acordo com as entidades envolvidas durante o processo de compra e venda de bens ou serviços. Por exemplo, a categoria B2B (Business to Business) é o nome dado quando uma empresa comercializa com outra através de um sistema. Além dessa, outras relações comuns no comércio eletrônico são:

- *Business to Consumer* - B2C - (empresa - consumidor);

- *Business to Government* - B2G - (empresa - governo);
- *Consumer to Government* - C2G - (consumidor - governo);

Outra categoria emergente no *e-commerce* é conhecida como C2C (Consumer to Consumer), ou ainda, de consumidor para consumidor.

**O comércio eletrônico consumer-to-consumer (C2C) tem provado ser uma modalidade de comércio eletrônico capaz de equiparar ou, mesmo, superar o sucesso dos principais modelos, o Business-to-Consumer (B2C) e o Business-to-Business (B2B) (PINHO, p. 3, 2015).**

Essa modalidade de comércio está presente no Brasil por meio de plataformas como a Olx e o Mercado Livre. Essas empresas permitem que pessoas físicas vendam produtos novos ou usados para outras pessoas.

## **2.2 Lojas virtuais e *marketplaces***

Para entrar no *e-commerce* e vender através da *internet*, as empresas interessadas costumam buscar por uma loja virtual própria, com a identidade visual desejada e a presença *online* garantida para começar a vender ou adaptar um negócio já existente à essa tecnologia e expandir seu alcance.

Contudo, os gastos com infraestrutura de *software* e *hardware*, além da contratação de desenvolvedores e o pagamento por todos os serviços adjacentes necessários, afasta muitos lojistas de possuírem uma loja virtual.

Nesse contexto, os *marketplaces* são plataformas virtuais na *Web* que estão se tornando cada vez mais comuns no cenário do *e-commerce*.

**Esse modelo funciona como uma vitrine virtual, onde várias lojas expõem os produtos e serviços. O vendedor pode se concentrar mais nos aspectos de venda e atendimento, como preço, promoções e campanhas, sem se preocupar com a estrutura e condições da plataforma (MATA, 2021, p. 15).**

O sucesso desse modelo, entre outras coisas, se deve ao fato de viabilizar a entrada no comércio eletrônico de forma mais barata e sem muitas complicações para o comerciante, iniciante ou não.

De acordo com Rosa (2019), atualmente há pelo menos três modelos de *marketplace* no mercado. O primeiro é chamado de "puro", e serve basicamente como uma vitrine virtual, onde somente os vendedores vendem seus produtos e serviços, como *eBay* ou *Mercado Livre*.

O segundo tipo é conhecido como modelo híbrido e possui como exemplo empresas como *Amazon* e *Americanas.com*, que são uma combinação de *e-commerce* tradicional com o *marketplace*. Por fim, o terceiro modelo é o híbrido com a adição de possuir loja física (ROSA, 2019).

A Loja Integrada é um exemplo de *marketplace* puro que se tornou muito popular no Brasil. Uma das missões da empresa, segundo seu próprio *site* oficial, é democratizar as vendas *online* e simplificar o acesso a tecnologia e informação. De acordo com dados de 2023, a plataforma possui mais de 2,5 milhões de lojas *online*. A Loja Integrada também segue um modelo de negócio semelhante a *startup* de Natal/RN e, por esse motivo, se tornou relevante para a pesquisa de campo e coleta de dados durante o decorrer do trabalho.

### 2.3 Código de Endereçamento Postal (CEP)

Segundo artigo do *site* oficial dos Correios (2023), uma das principais fontes de informação sobre entrega de encomendas no território nacional, o CEP é um conjunto numérico formado por oito dígitos que facilita o encaminhamento, o tratamento e a distribuição de objetos de correspondência pela atribuição a localidades, logradouros, unidades dos Correios, serviços, órgãos públicos, empresas e edifícios.

Para os Correios (2023), todas as pessoas devem utilizar o CEP para endereçar suas correspondências. Nesse sentido, esse número se torna essencial para a entrega de encomendas. Geralmente, é a maneira pela qual os usuários de lojas *online* identificam sua localização para verificar se a loja possui entregas para aquela região, bem como o valor do frete cobrado e a estimativa de prazo. Logo, seja por transportadora ou via *delivery*, o CEP é um dado relevante para realizar entregas nos locais corretos.

Os algarismos de um CEP representam, nessa ordem, a região, sub-região, setor, subsetor, divisor de subsetor e identificadores de distribuição. Os cinco primeiros algarismos podem identificar uma cidade ou distrito, enquanto os últimos

três algarismos destinam-se à localidades, logradouros, códigos especiais e Unidades dos Correios.

Essa divisão que constitui o CEP foi baseada em estudos de desenvolvimento socioeconômico e demográfico das regiões brasileiras, considerando a capacidade de crescimento de cada uma como um dos fatores para determinar a distribuição de CEPs pelo Brasil (CORREIOS, 2023). Todavia, essa disposição de endereços não é simples e podem ocorrer atualizações ao longo do tempo.

## **2.4 Logradouros e endereços**

Para entender melhor o domínio do problema das entregas enfrentado pelos lojistas, é necessário conhecer o conceito e as problemáticas envolvendo a palavra “logradouro”, termo muito utilizado durante a pesquisa para designar locais de entrega. De acordo com o dicionário *online* Michaelis, logradouros são lugares livres, destinados à circulação de pedestres e veículos, tal como ruas, avenidas, praças, viadutos, dentre outros.

Ainda segundo o *site* Cep Mais (2023), que permite a busca de logradouros por CEP, existem mais denominações para um logradouro além daquelas citadas, como aeroporto, alameda, área, campo, chácara, colônia, condomínio, conjunto, distrito, esplanada, estação etc. Segundo essa fonte, a cidade de São Paulo/SP possui mais de 65000 (sessenta e cinco mil) logradouros contabilizados oficialmente. No entanto, considerando os que ainda não foram oficializados pelo Diário Oficial do Município, esse número pode passar de 100000 (cem mil).

O logradouro é, portanto, parte essencial que constitui um endereço de entrega, juntamente com o bairro, a cidade e estado, o CEP, o número e algum complemento (opcional). Com essas informações é possível realizar entregas através das transportadoras ou motoboys.

Por essa razão, o CEP é muito utilizado para consultar regiões de entrega e métodos de envio, pois muitas fontes, como o Cep Mais, possuem mapeamentos entre CEPs e logradouros, tornando possível a obtenção das demais informações do endereçamento.

## **2.5 Delivery e entrega de encomendas**

As lojas virtuais oferecem diferentes métodos de entrega de produtos. Entre eles, os Correios, que é uma empresa pública federal, cuja responsabilidade é o envio e entrega de correspondências no Brasil (Portal PPI GOV, 2022). Além dos Correios, é comum a utilização de transportadoras particulares que oferecem serviços de entrega por todo o território nacional.

No entanto, plataformas como a Loja Integrada, oferecem a possibilidade do vendedor cadastrar regiões de entrega por motoboy, pois é um modelo mais simples e prático, especialmente para pequenos lojistas que não entregam para todo o Brasil, mas somente em regiões da sua própria cidade. Esse serviço é comumente chamado de *delivery*.

Segundo artigo da CNN Brasil (2022), a pandemia provocada pela Covid-19 impulsionou o serviço de *delivery* no Brasil, que era uma tendência observada no mercado. Segundo dados apresentados pelo Instituto Food Service Brasil, os gastos com esse método de entrega em 2021 representou 24% a mais que no ano anterior.

Esse aumento da utilização do serviço de *delivery* como uma solução para a entrega de encomendas é também explicado por Ferreira (2021):

Tendo em vista a necessidade de ficarem isoladas em casa e também devido ao fechamento do comércio para conter a propagação do coronavírus, as pessoas passaram a solicitar ainda mais esse tipo de serviço para atender às necessidades da vida doméstica (entrega de refeições, de compras de supermercado, de remédios etc.) e, por sua vez, as próprias empresas passaram a investir no serviço de entregas a domicílio (FERREIRA, p. 358 , 2021).

Logo, a entrega por meio de motoboys têm se tornado uma prática comum, e isso explica porque algumas plataformas de *e-commerce* possibilitam a configuração desse método por parte dos lojistas. Dessa forma, eles vão poder atender locais mais próximos com velocidade e sem precisar contratar serviços mais robustos, como os Correios. Todavia, as dimensões físicas e o peso do pedido fazem a diferença no momento de escolher qual serviço de entrega utilizar.

## 2.6 Application Programming Interface (API)

Segundo a IBM (2023), uma API é um conjunto de regras que definem como aplicações ou dispositivos devem se comunicar e há diversos padrões a serem seguidos para alcançar esse objetivo. Essas regras constituem um contrato de como

as duas ou mais partes devem estabelecer a conexão e proceder com a troca de informações.

Em um sentido mais amplo, API é um conceito que não envolve, necessariamente, a *internet* ou o padrão *REST*, que será abordado mais adiante. Um sistema operacional, por exemplo, contém APIs para acessar e controlar determinados recursos, como *drivers* e outros programas. Logo, uma API também pode ser definida como um serviço encapsulado que expõe recursos através de uma interface programada com o intuito de atender a uma demanda específica.

Neste trabalho, é implementada uma API que utiliza o padrão *REST* para retornar informações importantes relativas às possíveis regiões de entrega de um lojista em uma plataforma de *e-commerce*. Os detalhes de implementação não precisam ser explícitos, uma vez que é esperado que o cliente acesse uma *URL* com os parâmetros corretos e obtenha uma resposta de acordo com a consulta.

## 2.7 API REST

Representational State Transfer (*REST*) é um estilo de design de arquitetura de APIs muito utilizado que faz uso do protocolo HTTP e dos seus métodos para criar interfaces eficazes, consistentes e padronizadas para o desenvolvimento *Web*. Segundo seu criador, Fielding (2000), a motivação para o desenvolvimento do padrão REST foi criar um modelo de arquitetura para definir como a *Web* deveria funcionar, sendo um guia para os padrões de protocolo da *Web*.

A arquitetura *REST* possui alguns princípios basilares para seu pleno funcionamento. De acordo com Fielding (2000), são eles:

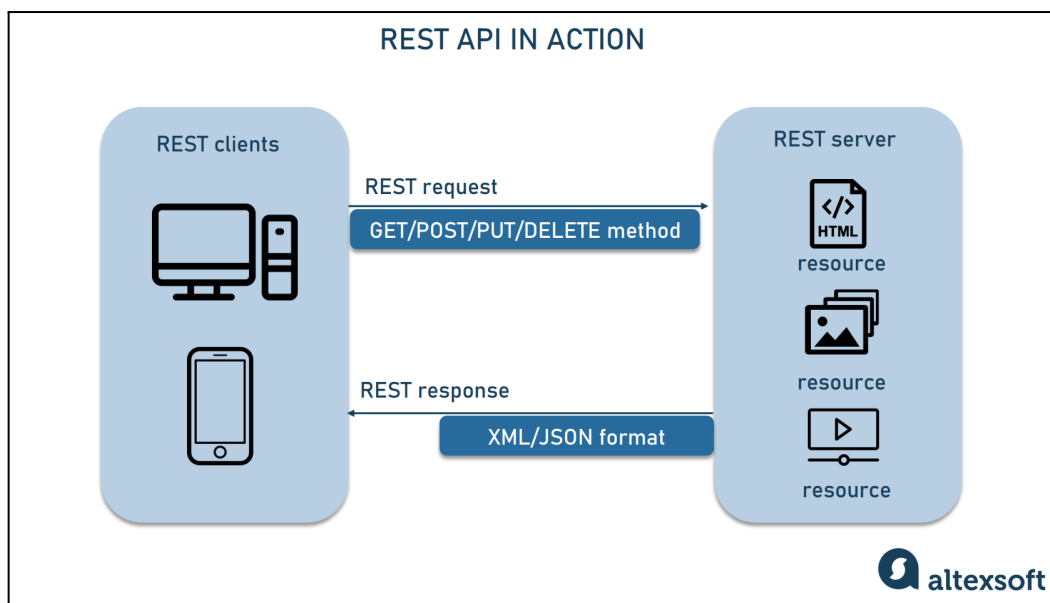
- *Client-Server*: Separar as responsabilidades na arquitetura Cliente-Servidor, em que cada qual possui as suas atribuições, permitindo que os dois se desenvolvam independentemente.
- *Stateless*: A comunicação não deve manter um estado entre as requisições. Logo, toda requisição do cliente para o servidor deve ser completa para ser respondida corretamente. Esse princípio objetiva melhorar a visibilidade, a confiabilidade e a escalabilidade.
- *Cache*: Essa característica do *REST* visa melhorar a eficiência, ao permitir que os dados contidos na resposta possam ser "cacheados", ou seja,

armazenados para futuras consultas, diminuindo o número de requisições e melhorando a performance.

Esse *design* arquitetural se tornou muito popular nos últimos anos devido a crescente necessidade de integrar diversos dispositivos (celulares, televisões, *desktops* e outros) a uma única aplicação através de uma API. Nesse contexto o *REST* oferece uma interface de fácil compreensão, baseada em um protocolo já conhecido da *Web*, o HTTP.

A Figura 2 ilustra o funcionamento de uma API *REST* onde, ao lado esquerdo, os dispositivos fazem requisições ao servidor, ao lado direito, utilizando métodos HTTP, como GET, POST, PUT e DELETE. O servidor *REST* é responsável por receber os pedidos por meio de uma URL e retornar o recurso requerido, que pode ser uma página HTML, uma imagem, um vídeo ou simplesmente um texto simples.

**Figura 2** - Exemplo da API *REST* em ação.



**Fonte:** Altexsoft (2023).

### 3 METODOLOGIA

Este capítulo descreve os procedimentos metodológicos utilizados no desenvolvimento desta pesquisa. Está organizado da seguinte maneira: a Seção 3.1 relata a pesquisa bibliográfica; a Seção 3.2 expõe a pesquisa de campo e como foi desenvolvida; a Seção 3.3 apresenta a etapa de coleta de dados; a Seção 3.4 explana o desenvolvimento do *software*.

#### 3.1 Pesquisa Bibliográfica

Como de praxe, o trabalho iniciou-se com uma revisão bibliográfica acerca do tema que é abordado durante o projeto. Por meio da consulta a livros didáticos, artigos científicos e determinados *sites* oficiais, foi possível alicerçar ainda mais o intuito da presente pesquisa. Ademais, vale ressaltar a importância dessa fase inicial para o desenvolvimento dos demais capítulos do trabalho, uma vez que:

**Nessa etapa, como o próprio nome indica, analisamos as mais recentes obras científicas disponíveis que tratem do assunto ou que deem embasamento teórico e metodológico para o desenvolvimento do projeto de pesquisa. É aqui também que são explicitados os principais conceitos e termos técnicos a serem utilizados na pesquisa (PRODANOV; FREITAS, 2013, p. 131).**

Ainda segundo Prodanov e Freitas (2013), o intuito da pesquisa científica não é apenas servir como um relatório ou uma "coletânea de resumos", mas também desenvolver algum caráter interpretativo sobre o que foi coletado. Logo, o levantamento bibliográfico também exercita a capacidade de discutir e estruturar, de forma sistemática, as opiniões e o conhecimento sobre determinado assunto.

Para além de materiais físicos, fez-se necessário o acesso a sítios na *internet* com dados e informações necessárias. Para Marconi e Lakatos (2017), "hoje, com a *Internet*, o levantamento de obras (livros e artigos) que tratam de um tema de pesquisa pode ser feito de casa, sem necessidade de deslocamento". Nesse sentido, os recursos que serviram de base para a fundamentação teórica foram consultados, principalmente, através de plataformas como o Google Acadêmico e do acervo da própria Universidade Federal Rural do Semi-árido (UFERSA).

Mediante a realização da pesquisa bibliográfica foi possível apresentar definições e discussões relevantes para o trabalho. Ao pesquisar sobre o *e-commerce* desde sua origem, compreendeu-se melhor o seu desenvolvimento até os dias atuais e porque tem alcançado números expressivos. Inclusive, a classificação das plataformas virtuais em B2B, B2C e C2C foi útil para entender as relações entre os vendedores e consumidores nesses *sites*.

Durante a pesquisa, encontrou-se detalhes e exemplos de lojas virtuais e *marketplaces*, conceitos que indicam uma tendência de mercado do *e-commerce*, pois muitas empresas como Amazon, Magazine Luiza, Casas Bahia, dentre outras passaram a oferecer esses serviços.

Outro ponto investigado foi a condição do *delivery* e entrega de encomendas. Foi analisado que a demanda por envios via motoboy ou motoristas de aplicativo tem aumentado, especialmente após a pandemia que se iniciou no final de 2019. Essa informação é relevante para apoiar a justificativa do trabalho.

Ademais, a busca por definições sobre usabilidade, prototipagem, engenharia de *software* e tecnologias para o desenvolvimento da aplicação foram fundamentais para pensar numa metodologia adequada para a realização dessa pesquisa e amadurecer os procedimentos que são aplicados até o fim.

Em suma, os materiais consultados forneceram o básico para aprofundar o entendimento do tema, tais como livros didáticos conhecidos da área tanto de pesquisa científica quanto de engenharia de *software*, além de artigos científicos buscados através de agregadores como o Google Acadêmico. Por se tratar de *e-commerce* e de tecnologia, assuntos que mudam constantemente, foi importante consultar *sites* mais atualizados sobre o tema.

### 3.2 Pesquisa de Campo

A problemática que originou este trabalho surgiu de forma espontânea, a partir de relatos e reclamações de usuários e desenvolvedores de uma plataforma virtual desenvolvida por uma *startup* de Natal, no Rio Grande do Norte. Com a intenção de formalizar e especificar melhor os problemas informados, foram utilizadas a pesquisa de campo e a coleta de dados.

De acordo com Prodanov e Freitas (2013, p. 59), uma vantagem da pesquisa de campo sobre os levantamentos ou *survey* é que: "[...] o planejamento do estudo

de campo apresenta muito mais flexibilidade, podendo ocorrer mesmo que seus objetivos sejam reformulados ao longo do processo de pesquisa". Tal flexibilidade permite uma melhor observação do fenômeno estudado e auxilia na coleta de dados.

Marconi e Lakatos (2017) listam uma das vantagens desse método, que é o "acúmulo de informações sobre determinado fenômeno, que também podem ser analisadas por outros pesquisadores, com objetivos diferentes". Essa característica faz parte do perfil exploratório das pesquisas de campo, que busca observar um evento em um determinado ambiente e tecer sobre ele descrições qualitativas e quantitativas.

Durante a pesquisa de campo, a equipe responsável por lidar com os clientes que estavam realizando testes na plataforma, disponibilizou um contato através do *whatsapp*, por onde os usuários poderiam relatar suas dificuldades e possíveis sugestões. Dessa maneira, novas funcionalidades ou pedidos de resolução de *bugs* seriam adicionados no quadro de tarefas para serem feitas pelos desenvolvedores.

Ao longo dos testes, a problemática do cadastro das regiões de entrega no módulo de *delivery* do sistema foi se tornando mais evidente e motivando a necessidade de uma solução para os obstáculos enfrentados pelos usuários. Os mesmos relataram dificuldades no preenchimento dos dados, o que bloqueava o processo de configuração de suas lojas virtuais na plataforma.

A pesquisa de campo foi essencial para encontrar a problemática que justifica esta pesquisa, através da qual pretende-se implementar uma aplicação que seja capaz de solucionar os obstáculos enfrentados. Como resultado, obteve-se uma elaboração mais específica do problema, por meio de diálogos informais e registros formais provenientes da *startup* de Natal/RN, onde o estudo se iniciou.

Na Tabela 1 é possível observar a transcrição de uma tarefa criada no *backlog* da *startup* supracitada, que é a lista de tarefas que são encaminhadas para desenvolvimento de acordo com a metodologia ágil adotada na empresa. Essa tarefa demonstra um problema que está impactando os usuários lojistas, identificados como *testers*, seguido de uma possível solução dada pelos analistas responsáveis.

Durante a pesquisa de campo, esse problema havia sido discutido de maneira informal, através apenas da observação, sem seguir métodos específicos. No entanto, o problema foi registrado como uma nova funcionalidade a ser

desenvolvida, o que implica na sua relevância tanto para essa, quanto para outras plataformas.

**Tabela 1** - Descrição da tarefa criada no *backlog* da *startup*.

| <b>Título da tarefa</b> | <b>Implementar novo fluxo de cadastro de faixa de entrega no aplicativo motoboy (<i>delivery</i>)</b>                                                                                                                |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Prioridade              | Alta                                                                                                                                                                                                                 |
| Tipo de tarefa          | <i>Feature</i>                                                                                                                                                                                                       |
| Descrição (resumo)      | Nossos <i>testers</i> estão tendo dificuldades no cadastro das faixas de CEP para entrega dos seus produtos. A nossa sugestão é que a gente utilize a API de CEPs para poder já trazer o nome das cidades e bairros. |

**Fonte:** *Startup* de Natal/RN (2022).

### 3.3 Coleta de Dados

Os dados são úteis em todo o processo de pesquisa pois, sem eles, não há pesquisa, mas apenas especulação. Esses dados podem ser primários ou secundários, dependendo do processo de pesquisa. Os dados extraídos da realidade pelo pesquisador são considerados primários, enquanto os dados que se encontram em bibliotecas, livros, artigos e acervos online são chamados de secundários (PRODANOV; FREITAS, 2002).

Os dados secundários desta pesquisa, portanto, foram os utilizados para construção da fundamentação teórica através da revisão bibliográfica realizada por meios de recursos que já existiam na literatura.

Os artigos e livros lidos, por exemplo, ajudaram a entender o processo de pesquisa e a como encontrar citações, referências e interpretações sobre o tema abordado, em diferentes contextos. Além disso, esse tipo de dado também será

consultado ao acessar sistemas similares, bem como na coleta de faixas de CEP e regiões, assuntos que são abordados nas subseções 3.3.2 e 3.3.3, respectivamente.

Enquanto isso, os dados primários que são utilizados nesta pesquisa foram captados por intermédio do estudo de campo e da interação formal e informal com os usuários da startup, com o auxílio de técnicas de pesquisa, tais como a observação e o questionário. Este último é relevante para entender melhor o que foi observado na pesquisa de campo e gerar insumos para a análise posterior.

Portanto, a coleta de dados seguiu-se interligada à pesquisa de campo. Durante essa etapa, foram levantados dados importantes tanto para a fundamentação teórica, quanto para a metodologia do trabalho, bem como para a implementação da aplicação. Tecnologias, métodos, informações gerais sobre *e-commerce* e sobre as plataformas existentes, são exemplos de dados coletados através da pesquisa de campo e da pesquisa bibliográfica.

### **3.3.1 Observação assistemática**

Dentre os métodos de coleta de dados primários, Prodanov e Freitas (2002) destacam a técnica de observação. Segundo os autores, existem diferentes maneiras de aplicar esse método. Um deles é a observação assistemática, que é uma técnica de observação não estruturada ou espontânea, que pode ser utilizada de modo informal e livre, sem planejamento prévio. Mas Marconi e Lakatos (2017) alertam para um dos perigos dessa técnica, que é quando o observador se deixa envolver emocionalmente ou pensa que sabe muito sobre o tema.

Como dito anteriormente, a problemática discutida neste projeto de pesquisa surgiu de maneira informal, através de reclamações de pessoas que testaram uma funcionalidade de um sistema e encontram nela problemas. Para não cair na armadilha desse tipo de pesquisa observatória, é preciso formalizar esses relatos, tanto para embasar melhor o trabalho, quanto para futura análise e comparação. Caso contrário, serão apenas impressões parciais do pesquisador quando foi exposto ao problema.

Nesse sentido, a coleta de dados buscou formalizar, quando possível, o máximo de informações extraídas durante as observações realizadas na *startup* norte-rio-grandense. Para isso, as pesquisas de campo e bibliográfica foram

importantes para sustentar os relatos concedidos pelos desenvolvedores e usuários da empresa estudada nesse processo.

### 3.3.2 Faixas de CEP e regiões de entrega

De acordo com o tutorial oficial da Loja Integrada, uma das plataformas de lojas virtuais mais utilizadas no Brasil, para utilizar serviços de entrega como motoboy, transportadoras ou fretes personalizados, é necessário entender o conceito de faixas de CEP.

**Faixa de CEP é o intervalo de CEP's de uma região, por exemplo, a faixa de CEP do estado de São Paulo é: 01000-000 a 19999-999, ou seja, todos os CEP's dentro desse intervalo são de bairros e cidades dentro do estado de SP (LOJA INTEGRADA, 2022).**

No entanto, encontrar esses números para cada região que pretende-se entregar encomendas pode gerar empecilhos para os usuários, como a dificuldade de pesquisar por esses dados e o esforço extra para cadastrar cada uma dessas faixas, exigindo muita paciência e concentração.

Com o intuito de desenvolver uma aplicação que solucione esses problemas, é preciso compreender como esse cadastro é realizado atualmente nas plataformas e onde é possível encontrar essas faixas de CEP. Após identificá-las, foi necessário mapear essas faixas de acordo com suas respectivas regiões.

De início, procurou-se identificar as soluções e serviços que já estavam em uso nessas plataformas para compreender melhor o cenário atual e propor uma alternativa mais assertiva. De modo geral, as plataformas delegam aos lojistas a responsabilidade de buscar as faixas de CEP para as localidades que pretendem inserir. Por existir muitas fontes distintas e o processo de busca ser muito manual, é comum que o usuário se perca ou demore bastante para encontrar todos os CEPs que procura.

Em seguida são exibidos os formulários que algumas plataformas utilizam para inserção dessas regiões de entrega. O exemplo na Figura 3 é o da Loja Integrada, uma das plataformas pioneiras em facilitar a criação de lojas virtuais para que vendedores consigam entrar no *e-commerce*.

**Figura 3** - Formulário para cadastro de regiões de entrega Loja Integrada.

**Faixas de CEP atendidas**

CEP inicial      CEP final      Prazo de entrega

           0 dias ▼

[Adicionar faixa de CEP](#)

Ao colocar CEP inicial 01000-000 e CEP final 09999-999 você está definindo que vai atender o estado de São Paulo.  
 Não sabe a faixa de CEP? [clique aqui.](#)

| 1 faixa de CEP cadastrada |           |                  |
|---------------------------|-----------|------------------|
| CEP inicial               | CEP final | Prazo de entrega |
| 00000-000                 | 99999-999 | 1 dia            |

**Fonte:** Loja Integrada (2022).

A Figura 4 exibe o formulário de cadastro de regiões de entrega da *Nuvem Shop*, outra plataforma que permite a criação de lojas virtuais, semelhante à Loja Integrada e também a *startup* norte-rio-grandense abordada neste trabalho. Nesse caso, a diferença é que a solução apresentada abaixo é um pouco mais sofisticada do ponto de vista da usabilidade, pois possui um *autocomplete* e, à priori, esconde as faixas de CEP.

**Figura 4** - Formulário de cadastro de zonas de entrega.

**Zonas de entrega**

Defina para quais estados você deseja oferecer esta forma de entrega:

são I

SP - São Paulo

☐ Mostrar como ponto de retirada

Seus clientes poderão retirar seus pedidos nesse ponto. Fica visível separadamente dos envios a domicílio.

**Fonte:** Nuvem Shop (2022).

A Figura 5 demonstra mais uma plataforma, com propósito similar, chamada *Tray Commerce*, onde pode-se observar outras dificuldades para inserir as

informações de entrega. No topo, o formulário para cadastro de faixas de CEP e abaixo a listagem com as faixas cadastradas. A maior dificuldade apresentada é conseguir cadastrar, uma por uma, todas as faixas possíveis de entrega. Dependendo do alcance do lojista, essa pode se tornar uma tarefa exaustiva.

**Figura 5 - Cadastro e exibição de faixas de CEP para entrega.**

| Faixa Inicial | Faixa Final | Adicional Fixo | Adicional Produto | Excedente | Faixa de Frete            | Prazo de Entrega | Valores         | Ações |
|---------------|-------------|----------------|-------------------|-----------|---------------------------|------------------|-----------------|-------|
| 01000000      | 03999999    | R\$ 0.00       | 0.00 %            | -         | -                         | de 2 a 4 dias    | Incluir Valores |       |
| 01100000      | 99999999    | R\$ 0.00       | 0.00 %            | -         | R\$ 200.00 a R\$ 10000.00 | de 1 a 100 dias  | Incluir Valores |       |
| 04000000      | 05999999    | R\$ 0.00       | 0.00 %            | -         | -                         | de 2 a 4 dias    | Incluir Valores |       |
| 06000000      | 06999999    | R\$ 0.00       | 0.00 %            | -         | -                         | de 2 a 4 dias    | Incluir Valores |       |
| 07000000      | 08999999    | R\$ 0.00       | 0.00 %            | -         | -                         | de 2 a 4 dias    | Incluir Valores |       |
| 09000000      | 09499999    | R\$ 0.00       | 0.00 %            | -         | -                         | de 3 a 7 dias    | Incluir Valores |       |
| 09500000      | 09999999    | R\$ 0.00       | 0.00 %            | -         | -                         | de 3 a 7 dias    | Incluir Valores |       |

**Fonte:** Tray Commerce (2022).

Como foi possível notar, cada plataforma implementa suas telas de uma forma diferente, mas o lojista precisa preencher uma quantidade significativa de dados para configurar os métodos de entrega via *delivery*, resultando nas dificuldades apresentadas até então.

Sendo assim, um dos objetivos específicos do trabalho foi encontrar informações sobre as faixas de CEP para construir uma base de dados central capaz de tornar todo esse processo de seleção mais transparente e fácil para o lojista através de uma API que seria integrada pela plataforma.

Todavia, foi muito difícil encontrar conjuntos de dados confiáveis, consistentes e, principalmente, atualizados. Durante a pesquisa, foram consultadas diversas fontes de dados, como o Instituto Brasileiro de Geografia e Estatística (IBGE), a API do *ViaCep*, o sistema CEPS Brasil, o serviço *Ceps.io* e o próprio *site* oficial dos Correios, os quais disponibilizam todos os CEPs do Brasil.

As alternativas mencionadas acima já são indicadas pelas próprias plataformas de *e-commerce* como locais de consultas, onde os usuários podem obter os CEPs das regiões de entrega desejadas. No entanto, dessas, apenas o *ViaCep* possui uma API pública para consulta por código postal, que é utilizada mais

por desenvolvedores do que por clientes ou vendedores. Por isso, os lojistas findam pesquisando de forma manual, CEP a CEP, até encontrar todos que deseja.

Na Figura 6, por exemplo, observa-se a realização de uma busca por faixa de CEP ou UF/localidade no *site* oficial dos Correios. Essa ferramenta permite encontrar diversas faixas de CEP utilizando os parâmetros de busca e é uma das opções mais sugeridas para os lojistas.

**Figura 6** - Resultado da busca por Faixa de CEP ou UF/Localidade - Correios.

Resultado da Busca por Faixa de CEP ou UF/Localidade

| UF | Faixa de CEP          |
|----|-----------------------|
| SP | 01000-000 a 19999-999 |

Próximo 1 a 50 de 842

| Localidade        | Faixa de CEP          | Situação                       | Tipo de Faixa            |
|-------------------|-----------------------|--------------------------------|--------------------------|
| Adamantina        | 17800-000 a 17809-999 | Não codificada por logradouros | Total do município       |
| Adolfo            | 15230-000 a 15239-999 | Não codificada por logradouros | Total do município       |
| Aguaí             | 13860-001 a 13869-999 | Codificado por logradouros     | Total do município       |
| Aguaí             | 13860-001 a 13869-999 | Codificada por logradouros     | Exclusiva da sede urbana |
| Agudos            | 17120-001 a 17149-999 | Codificado por logradouros     | Total do município       |
| Agudos            | 17120-001 a 17139-999 | Codificada por logradouros     | Exclusiva da sede urbana |
| Alambari          | 18220-000 a 18224-999 | Não codificada por logradouros | Total do município       |
| Alfredo Marcondes | 19180-000 a 19189-999 | Não codificada por logradouros | Total do município       |
| Altair            | 15430-000 a 15439-999 | Não codificada por logradouros | Total do município       |
| Altinópolis       | 14350-000 a 14389-999 | Não codificada por logradouros | Total do município       |

**Fonte:** Site oficial dos Correios (2023).

Com base nas dificuldades encontradas para formar a base de dados da API proposta pelo trabalho, foi utilizado um agrupamento de dados composto por mais de 750000 (setecentas e cinquenta mil) linhas, onde cada uma continha um CEP de um logradouro, seu bairro e sua cidade. O arquivo de texto disponibilizado pela plataforma *ceps.la* contém dados de 2018 extraídos dos Correios.

A partir desses dados, é possível pensar em maneiras de mapear essas informações de acordo com um CEP de origem e retornar essa informação para as plataformas apresentarem para o lojista as regiões de entrega para que sejam, de alguma forma, selecionadas. Dessa maneira, descartando um esforço maior por parte do vendedor de acessar outro sistema para obter essas faixas.

Portanto, através da coleta de dados, obteve-se diferentes soluções implementadas por plataformas distintas. Porém todas apresentam problemas similares. Com esse levantamento, já foi possível enxergar o amplo contexto em que este trabalho está inserido. Embora a problemática seja específica, há pelo menos quatro plataformas, incluindo a *startup* estudada, que podem aprimorar a experiência de seus usuários através de uma solução independente, que possa ser integrada mais facilmente.

### 3.4 Desenvolvimento da aplicação

Após a realização das pesquisas bibliográficas e de campo, além da coleta dos dados necessários, iniciou-se a fase de desenvolvimento propriamente dita. O produto final desse projeto visa ser uma aplicação independente de plataforma que possa ser integrada através de alguma biblioteca de software ou API para solucionar os problemas durante o cadastro de regiões de entrega nos sistemas de *e-commerce*.

Com tudo isso, esta seção descreve os processos e atividades que foram tomados com o objetivo de analisar, desenvolver e codificar a aplicação. De modo geral, são utilizadas definições de Engenharia de *Software*, desde a metodologia de desenvolvimento utilizada até as tecnologias escolhidas para a implementação do *software*.

#### 3.4.1 Engenharia de *Software*

O presente trabalho utilizará conceitos da Engenharia de *Software* para guiar o desenvolvimento da aplicação. Para Sommerville (2011),

**Engenharia de software é uma disciplina de engenharia cujo foco está em todos os aspectos da produção de software, desde os estágios iniciais da especificação do sistema até sua manutenção, quando o sistema já está sendo usado" (SOMMERVILLE, p. 5, 2011).**

Desse modo, as etapas de especificação, desenvolvimento, validação e evolução de *software* são as atividades responsáveis pela produção profissional de *software*.

### 3.4.1.1 Análise e especificação de requisitos

Inicialmente, para começar o desenvolvimento da aplicação, é preciso conhecer quais são os seus requisitos, ou seja, o que se propõe a fazer e como. Para isso, uma sub-área da Engenharia de *Software*, conhecida como Engenharia de Requisitos, aplica algumas etapas para identificar e especificar as exigências de funcionamento da aplicação.

De acordo com Pressman (2011), a Engenharia de Requisitos possui um total de 7 tarefas distintas tais como concepção, levantamento, elaboração, negociação, especificação, validação e gestão. Essas tarefas auxiliam com ações e estratégias sistemáticas para detectar os requisitos do sistema, de modo que serão brevemente descritas a seguir, com base em Pressman (2011):

A concepção é, como o nome sugere, a fase inicial de viabilidade e de necessidade do negócio ou do *software*. Pode se dar por meio de conversas informais ou de uma análise sobre uma determinada problemática. A partir disso, é feito o levantamento, que é uma tarefa voltada para perguntas e questionamentos sobre os objetivos do *software* e como ele agrega valor.

Com o que foi levantado, é feito a elaboração, que contém um refinamento do que foi coletado anteriormente, descrevendo as relações entre os atores (usuários ou sistemas) e entre o próprio sistema. Nessa etapa, as entidades que faziam parte do sistema começaram a aparecer no cenário, como plataformas, cidades, bairros e logradouros.

A negociação é a conciliação entre o que foi pedido pelos clientes e o que é viável, aliando as expectativas durante o processo. Essa fase também ocorreu de maneira informal, entre conversas com os usuários do sistema, com os desenvolvedores e gestores de produto e de projeto, para conciliar o que era possível fazer dentro das necessidades da empresa.

A etapa de especificação é a atividade responsável por detalhar os requisitos coletados até então através de cenários de uso, por exemplo. A especificação ocorreu por meio de linguagem natural, tentando criar sentenças claras e sem ambiguidades para evitar problemas de implementação e validação.

A validação, por sua vez, avalia e revisa os artefatos gerados pelas tarefas da Engenharia de Requisitos, a fim de garantir que os requisitos foram especificados corretamente, evitando problemas como a ambiguidade. Por último, a sétima tarefa

é a gestão de requisitos, que visa a preparação para possíveis mudanças que possam ocorrer durante o processo ou no futuro.

Essas atividades continuaram sendo realizadas em conjunto com a *startup* norte-rio-grandense, onde o problema foi identificado. Nesse caso, essa empresa seria um dos *stakeholders*, ou seja, faria parte do grupo de pessoas ou entidades interessadas nesse trabalho. Ademais, os dados coletados de outras empresas também foram importantes para formar a lista de requisitos funcionais e não funcionais.

#### 3.4.1.2 Projeto e implementação da aplicação

Após a análise e especificação dos requisitos, iniciará o desenvolvimento da aplicação. Nesta fase, as funcionalidades já identificadas são transformadas em *software*.

**Um projeto de software é uma descrição da estrutura do software a ser implementado, dos modelos e estruturas de dados usados pelo sistema, das interfaces entre os componentes do sistema e, às vezes, dos algoritmos usados (SOMMERVILLE, p. 25, 2011).**

De acordo com o objetivo deste projeto, a aplicação desenvolvida será independente e poderá ser integrada às plataformas interessadas. Para isso, nessa etapa, é projetada a arquitetura do sistema, seus componentes e a interface de comunicação entre eles, além do banco de dados onde serão armazenadas as informações necessárias.

Primeiramente, a aplicação foi implementada a partir dos dados coletados e requisitos do sistema, tais como consultar cidades, bairros e logradouros com base em um CEP de origem, concedido pelo lojista durante a configuração do método de entrega.

Em seguida, foram realizados os devidos testes na API para corrigir possíveis erros e garantir a qualidade do código. Após isso, foi preciso validar a API através do uso em situações reais, implementando uma ideia de interface visual, onde os dados disponíveis na API poderiam ser utilizados com o intuito de verificar o comportamento da aplicação de uma ponta a outra.

Por fim, disponibilizar essa API para que outras plataformas possam realizar a integração dessa aplicação nos seus módulos de entrega, permitindo que seus clientes possam utilizá-la da melhor forma possível. Para isso, a documentação do *software* se faz necessária, para guiar os desenvolvedores durante o processo de integração.

#### **3.4.1.3 Validação da aplicação**

A validação de *software* tem o objetivo de verificar se as especificações de um *software* estão em conformidade com o que foi estabelecido junto ao cliente. Essa atividade pode conter revisões e inspeções em cada estágio do desenvolvimento, em um processo iterativo (SOMMERVILLE, 2011).

Durante a validação de um *software*, diferentes testes são realizados para garantir a qualidade quanto ao cumprimento dos requisitos. Sommerville (2011), aponta pelo menos três testes essenciais: testes de desenvolvimento; testes de sistema; testes de aceitação.

Cada teste tem função específica e podem ser aplicados em diferentes estágios de desenvolvimento, um por vez ou em conjunto. Os três tipos de testes são abordados nesta pesquisa. Os testes de desenvolvimento são aplicados para validar funções e componentes específicos. Os testes de sistema servirão para verificar a integração entre os componentes e identificar possíveis erros. Por último, os testes de aceitação visam comprovar o funcionamento da aplicação com dados reais de usuários.

Pressman (2011), demonstra uma preocupação, por exemplo, com a interação entre usuário e tela: "[...] se a interface for mal concebida, o usuário se move aos trancos e barrancos, e o resultado será frustração e baixa eficiência no trabalho". Logo, é preciso validar se a solução desenvolvida conseguiu melhorar a usabilidade.

#### **3.4.1.4 Evolução da aplicação**

Um *software* possui um ciclo de vida que se estende para além do seu desenvolvimento. Ele tende a continuar funcionando por muito tempo, mas é comum que novas alterações surjam, como novas funcionalidades ou correção de

problemas. Para prevenir custos maiores e uma maior dificuldade, ao longo da implementação, decisões que visam uma boa manutenção no futuro devem ser tomadas. Como, por exemplo, boas práticas, utilização de versionamento de código, *backup* de dados, arquiteturas escaláveis, documentação atualizada, etc.

Segundo Sommerville (2011), "poucos sistemas de software são completamente novos, e faz muito mais sentido ver o desenvolvimento e a manutenção como processos contínuos". Com base nesse pensamento, surgiram as práticas de desenvolvimento ágil.

### 3.4.2 Desenvolvimento ágil

Nas seções anteriores, foi descrito as etapas da Engenharia de *Software* que foram seguidas por este trabalho. No entanto, a maneira como são aplicadas depende do modelo de processo que é usado. Sommerville (2011) apresenta um dos modelos mais tradicionais, que é conhecido como cascata, em que uma etapa sucede a outra, somente quando a anterior termina por completo.

Existem alguns problemas na abordagem tradicional, principalmente relacionados às mudanças que impactam o andamento do projeto. Como a aplicação que será desenvolvida se encontra em um contexto mais empresarial, nesse caso, a utilização de metodologias ágeis se encaixou melhor.

De acordo com o site *oficial* do manifesto ágil, um dos doze princípios desse movimento diz que: "Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado" (AGILE MANIFESTO, 2001).

Com base nisso, as metodologias ágeis foram aplicadas sempre que possível, iterando sobre as atividades de Engenharia de *Software*, providenciando entregas frequentes para que os usuários e os *stakeholders* possam testar e validar, assim mantendo um alinhamento e controle de qualidade.

Um dos métodos que fazem parte do ciclo ágil de desenvolvimento é o *Kanban*, que é um técnica para dividir as tarefas em diferentes estados e atribuições. Durante o desenvolvimento da API, foi elaborado um quadro com pequenas tarefas que precisavam ser feitas. À medida que uma tarefa era completada, ela era movida para a seção de tarefas feitas, mantendo um histórico das atividades realizadas.

Embora seja difícil manter essa organização de forma consistente, devido ao número imprevisível de obstáculos, *bugs* e mudanças de direção, é importante manter os objetivos alinhados com os demais interessados.

A prática de testes, mesmo que manuais, foi parte da rotina de desenvolvimento para garantir o andamento da solução desenvolvida. Quando uma nova mudança ocorria na API, era preciso repensar e refazer os testes para manter a mesma consistência de antes da mudança. Essa prática é aconselhável pela metodologia ágil, pois visa corrigir os erros o mais rápido possível e não apenas no final do projeto.

Ademais, outra prática comum, realizada quando possível, durante o processo dessa pesquisa, foram as reuniões de alinhamento e acompanhamento do trabalho, o que permitiram a correção de partes específicas e evitaram problemas futuros em relação aos requisitos do sistema e os objetivos do projeto final.

### 3.4.3 Tecnologias

As tecnologias identificadas e escolhidas para o desenvolvimento desta aplicação foram selecionadas de acordo com a popularidade atual, a curva de aprendizagem e o propósito de cada uma. Há tecnologias que são mais utilizadas em determinados contextos, e quanto maior o seu uso, mais documentação e material de apoio são gerados, o que facilita a aprendizagem e o uso dessas ferramentas.

Considerando o ambiente da *Web*, pode-se dividir as tecnologias em dois lados distintos, mas que se comunicam entre si através de protocolos bem definidos. O primeiro lado é formado pelo *front-end* da aplicação, ou seja, a parte visual e interativa que será acessada através do navegador do usuário. O outro lado é o *back-end*, também conhecido como lado servidor, que é responsável pelo processamento dos dados e pelas consultas no banco de dados.

O *back-end* da API foi desenvolvido utilizando o *Node.js*, uma tecnologia moderna e amplamente utilizada em diversos cenários, especialmente para o desenvolvimento de serviços como *Web Services* e APIs. Para o controle e tratamento de rotas e *endpoints* foi utilizado o *Express.js*, que faz parte do ecossistema do *Node.js*. Por fim, o MySQL foi a opção determinada para ser o gerenciador de banco de dados do projeto.

### 3.4.3.1 *Node.js*

Como um ambiente de execução *JavaScript* assíncrono orientado a eventos, o *Node.js* é projetado para desenvolvimento de aplicações escaláveis de rede (NODEJS, 2022). Atualmente na sua versão LTS 18.12.1, o *Node.js* é praticamente imprescindível para criar aplicações em *JavaScript* que funcionem no lado servidor.

A arquitetura do *Node.js* é baseada em eventos e no *Event Loop*, que cria uma fila de execução de rotinas assíncronas, capaz de atender um número alto de requisições, sem bloquear a aplicação. Esse e outros conceitos tornam esse ambiente de execução muito versátil, podendo ser escolhido para diversas aplicações.

O fato de utilizar a linguagem *JavaScript*, que é uma das mais utilizadas atualmente, atraiu o público de desenvolvedores, que costumavam apenas a trabalhar com *front-end* no navegador, para implementar APIs e outros serviços utilizando *Node.js*, o que culminou em um ecossistema repleto de ferramentas e muito conteúdo produzido sobre.

Embora possua algumas desvantagens como as dificuldades ao lidar com o processamento de arquivos volumosos de dados por padrão, essa ferramenta é importante para criar a aplicação, processar os dados necessários e se comunicar com o banco de dados. Além disso, o *Node.js* também é utilizado como um ambiente de desenvolvimento, capaz de fornecer outras ferramentas para automatização de testes, criação de novas bibliotecas, *build* de aplicações, dentre outras funções.

### 3.4.3.2 *Express.js*

Segundo sua própria documentação oficial, o *Express* se apresenta como um *framework* pequeno e flexível para ser utilizado juntamente com *Node.js* para a criação de APIs e serviços escaláveis. Essa ferramenta é uma das alternativas mais utilizadas nesse contexto, devido sua simplicidade e poucas dependências iniciais, o que torna a curva de aprendizado menor em relação a *frameworks* mais complexos (EXPRESSJS, 2023).

Um dos principais recursos do *Express* é o roteamento, que é a capacidade de responder às requisições dos usuários de acordo com o *endpoint* ou o caminho

acessado. Cada rota no *Express* permite a utilização de métodos para receber a requisição através do protocolo *HTTP*, processar os dados necessários e responder o resultado para o cliente que buscou o recurso.

Também é possível criar *middlewares*, que são funções que podem ser executadas em momentos estratégicos para transformar os dados da requisição antes de retornar para o usuário. Esse recurso é muito utilizado para criar mecanismos de autenticação e autorização, bem como de validação de dados.

### 3.4.3.3 MySQL

O MySQL é o sistema de gerenciamento de banco de dados mais popular para persistir seus dados numa ferramenta robusta e multi-plataforma (MYSQL, 2022). A linguagem utilizada no MySQL, assim como na maioria dos SGBDs, é a chamada Linguagem Padrão de Consultas (SQL), por meio da qual se pode definir tabelas do banco de dados e realizar consultas ou *queries*.

O MySQL é um banco de dados relacional, ou seja, é baseado em relacionamentos entre seus registros, que são intuitivamente dispostos em tabelas. Essa tecnologia é fundamental para, por exemplo, armazenar as faixas de CEP e as regiões: cidades, bairros e logradouros. Utilizar um banco relacional tem a vantagem de manter uma relação mais forte entre as entidades que fazem parte do problema, o que pode facilitar a consulta dos dados.

### 3.4.3.4 MySQL Workbench

O *MySQL Workbench* é um *software* que auxilia arquitetos, desenvolvedores e administradores de banco de dados na realização de operações de consulta, de modelagem e de manipulação de dados através de uma interface gráfica com diversas funcionalidades (MYSQL WORKBENCH, 2023).

O *MySQL Workbench* foi utilizado durante o desenvolvimento do trabalho para visualizar, construir e modelar o banco de dados utilizado para o consumo da API. Ele permite inserir diretamente o SQL para criar e manipular as tabelas e os dados através da conexão estabelecida com o MySQL.

Além disso, ele é capaz de transformar as tabelas geradas em diagramas do modelo ER lógico, o que facilita a modelagem dos dados. Também é possível fazer o

caminho inverso e criar tabelas por meio de diagramas. Essas e outras funcionalidades foram importantes para criar e gerenciar a base de dados do sistema.

#### 3.4.3.5 *Knex.js*

O *Knex.js* é um popular *query builder*, programa utilizado para construir consultas em SQL mais simples, sem perder o poder e a robustez do SQL puro. Segundo sua própria documentação oficial, o *Knex.js* foi desenvolvido para o ambiente *Node.js* e pode ser integrado em diversos projetos com SGBDs diferentes, como *Postgresql*, *sqlite3*, *Oracle db* e o *MySQL*.

Essa ferramenta possui diversas funcionalidades, como a possibilidade de versionar seu banco de dados através da definição de *migrations*. Também é possível utilizar *transactions* e outros conceitos mais avançados do universo SQL. Sua utilização é recomendada para criar consultas mais seguras e complexas, escrevendo um código mais conciso.

#### 3.4.3.6 *Swagger*

*Swagger* é uma ferramenta que auxilia na documentação de APIs *REST* por meio de diversos recursos como o *design*, o desenvolvimento e os testes da API. Entre os serviços ofertados pelo *Swagger*, estão o *Swagger UI*, o *Swagger Hub* e o *Swagger Editor*. O primeiro é utilizado neste trabalho para visualizar a documentação de uma API com base no formato definido pela especificação *OpenAPI*.

O *OpenAPI* é uma especificação para documentação e descrição de uma API *REST*, sendo definido o formato, as características e todas as propriedades que podem ser utilizadas. Além disso, ela expõe como o arquivo deve ser estruturado, as restrições de tipos e parâmetros a serem implementados. Essa especificação é utilizada em sua versão 3.0.0 neste trabalho para criar o arquivo lido pelo *Swagger*.

O *Swagger* também possui um editor capaz de definir e elaborar uma API, desde o seu *design* até o desenvolvimento e testes. Esses projetos desenvolvidos podem ser compartilhados e editados de forma colaborativa utilizando o serviço

*Swagger Hub*. Alguns desses serviços são pagos, mas os métodos utilizados durante a presente pesquisa são totalmente gratuitos e *open source*.

#### 3.4.3.7 *Insomnia*

O *Insomnia* é um *software* de código aberto para *design*, depuração e testes de APIs. Ele possui uma interface fácil de se utilizar, com funcionalidades avançadas para autenticação, geração de código e variáveis de ambiente (INSOMNIA, 2023). O *Insomnia* é utilizado também como um cliente HTTP que possibilita testar uma API através de requisições aos seus *endpoints* usando os métodos GET, POST, PUT, DELETE e outros.

Essa ferramenta foi aplicada para realizar os testes na API. Com ela, é possível criar novas requisições utilizando algum dos métodos HTTP, além de organizá-las em pastas e *workspaces* para manter os ambientes isolados. Também é capaz de alterar os *headers*, os parâmetros e o corpo da requisição, simulando um navegador ou uma outra aplicação que vai consumir os dados da API. Logo, o *Insomnia* desempenha o papel de validar a utilização das rotas da API, sem a necessidade de desenvolver uma interface *Web* para isso.

#### 3.4.3.8 GIT e versionamento de código

De acordo com seu site oficial, *git-scm.com*, o *Git* é um sistema de controle de versão distribuído criado para controlar as alterações de pequenos e grandes projetos. Segundo Sommerville (2011),

Sistemas de gerenciamento de versões incluem recursos para coordenar o desenvolvimento de diversos programadores. Eles bloqueiam um desenvolvedor que esteja escrevendo sobre um código submetido ao sistema por outra pessoa (SOMMERVILLE, p. 137, 2011).

O *Git* é primordial para manter a consistência e estabilidade do código-fonte durante o processo de desenvolvimento e as sucessivas mudanças que podem acontecer. Seu uso se torna cada vez mais importante, especialmente para times com vários desenvolvedores e muitas alterações ocorrendo em paralelo na mesma base de código.

## 4 DESENVOLVIMENTO DA API

Este capítulo refere-se ao desenvolvimento da API proposta no trabalho e inicia apresentando os requisitos funcionais da aplicação na seção 4.1. A seção 4.2 trata dos requisitos não funcionais. Na sequência, a seção 4.3 exhibe a estrutura e o processo de construção da base de dados da aplicação. A seção 4.4 descreve, passo a passo, a implementação do serviço propriamente dito. A seção 4.5 demonstra o processo de testes da aplicação. A seção 4.6 mostra a documentação da API e as instruções para realizar a integração. Por último, a validação da utilização da API ocorre na seção 4.7.

### 4.1 Requisitos funcionais

Os requisitos funcionais foram obtidos durante as fases de pesquisa de campo e coleta de dados, tomando como base os formulários das plataformas de *e-commerce* e as tarefas previamente criadas pela *startup* estudada. De acordo com o autor Sommerville (2011), os requisitos funcionais são declarações dos serviços que o sistema deve fornecer ou como alguns processamentos devem ser realizados. Nesse caso, cada requisito funcional foi mapeado para um recurso da API, que indica qual resultado é esperado após o processamento dos dados.

Há mais de uma forma de expressar os requisitos funcionais de um sistema, seja através de linguagem natural estruturada, casos de usos, cenários ou diagramas. Outro ponto relevante diz respeito ao nível de detalhamento da especificação dos requisitos, que é diferente dependendo do público-alvo. Por exemplo, engenheiros de *software* precisam de detalhes mais técnicos de implementação, enquanto os usuários e clientes lidam com um nível de abstração maior e, portanto, menos detalhado.

Abaixo segue-se a tabela com os códigos e as descrições dos requisitos funcionais que serviram de base para o desenvolvimento dos recursos da API, incluindo sua base de dados relacional, da qual os dados necessários são extraídos e processados. Cada requisito possui um código “RF\_” junto à uma numeração para servir como referência para aquele requisito. Além disso, As descrições são afirmativas do que deve ser feito.

**Tabela 2** - Requisitos funcionais

| <b>Código</b> | <b>Descrição</b>                                       |
|---------------|--------------------------------------------------------|
| RF_01         | A API deve permitir a consulta de cidades por CEP      |
| RF_02         | A API deve permitir a consulta de bairros por CEP      |
| RF_03         | A API deve permitir a consulta de logradouros por CEP  |
| RF_04         | A API deve retornar todos os bairros de uma cidade     |
| RF_05         | A API deve retornar todos os logradouros de um bairro  |
| RF_06         | A API deve permitir a integração com novas plataformas |

**Fonte:** Autoria própria.

## 4.2 Requisitos não funcionais

Os requisitos não funcionais, também chamados de requisitos de qualidade, são definidos como sendo características do sistema, como confiabilidade, segurança, tempo de resposta, usabilidade, organização, eficiência, implementação, portabilidade, para citar algumas (SOMMERVILLE, 2011). Embora possa ser difícil diferenciá-los dos requisitos funcionais em determinados momentos, os requisitos não funcionais não correspondem às funcionalidades do sistema, mas sim às restrições que o envolvem.

**Tabela 3** - Requisitos não funcionais.

| <b>Código</b> | <b>Descrição</b>                                         |
|---------------|----------------------------------------------------------|
| RNF_01        | A API deve implementar o padrão estrutural <i>REST</i>   |
| RNF_02        | A API não deve permitir acesso não autorizado            |
| RNF_03        | A API precisa de documentação para facilitar integrações |
| RNF_04        | A API deve ser testada para garantir qualidade de código |

**Fonte:** Autoria própria.

Muitos requisitos não funcionais já são esperados pelos usuários, mesmo que de forma implícita, como disponibilidade, usabilidade, segurança, questões éticas e de desempenho. Há também requisitos que visam facilitar o lado dos

desenvolvedores, como requisitos de manutenibilidade, de testes, de gerenciamento de versões, de padrões técnicos e de desenvolvimento no geral. Logo, apesar de não serem funcionalidades, são aspectos importantes para a vida útil do *software*.

A Tabela 3 conta com os principais requisitos não funcionais (RNF) da API desenvolvida, que cobrem não só o padrão utilizado na implementação, mas também a segurança, a documentação e os testes. Essas restrições ajudam a garantir a qualidade geral da aplicação e são prioridades de acordo com a visão dos *stakeholders*.

### 4.3 Modelagem de dados

A modelagem de dados é o processo de estudar as informações existentes em um determinado contexto para construir uma representação estruturada desse contexto. A forma como esses dados são agrupados, denominada de estrutura, é o conjunto dos relacionamentos e restrições existentes entre eles e que compõem o cenário estudado (MACHADO, 2014).

Ainda segundo Machado, o conceito de abstração, fundamental para a modelagem de dados, permite analisar somente os elementos que são relevantes para o contexto estudado, excluindo momentaneamente toda a complexidade desnecessária. Tais elementos podem se tornar entidades no Modelo Entidade-Relacionamento (ER), que é bastante utilizado para construção de bancos de dados, uma vez que é capaz de representar, de maneira abstrata, os relacionamentos e características de objetos do mundo real.

Esse modelo foi o escolhido pois os dados sobre as regiões e CEPS disponíveis atualmente não possuem uma relação forte para realizar consultas mais práticas como, por exemplo, retornar todos os bairros para uma determinada cidade, ou retornar os logradouros do mesmo bairro ao qual o lojista pertence. Há muitas fontes na *internet* que retornam essas informações, mas de forma desestruturada ou não relacional.

Heuser (1998) diz que um banco de dados pode ser modelado de várias maneiras, dependendo dos níveis de abstração. Por exemplo, uma visualização do modelo mais simplificado, contendo apenas conceitos, é ideal para os usuários conhecerem a modelagem e as entidades que fazem parte do contexto do problema.

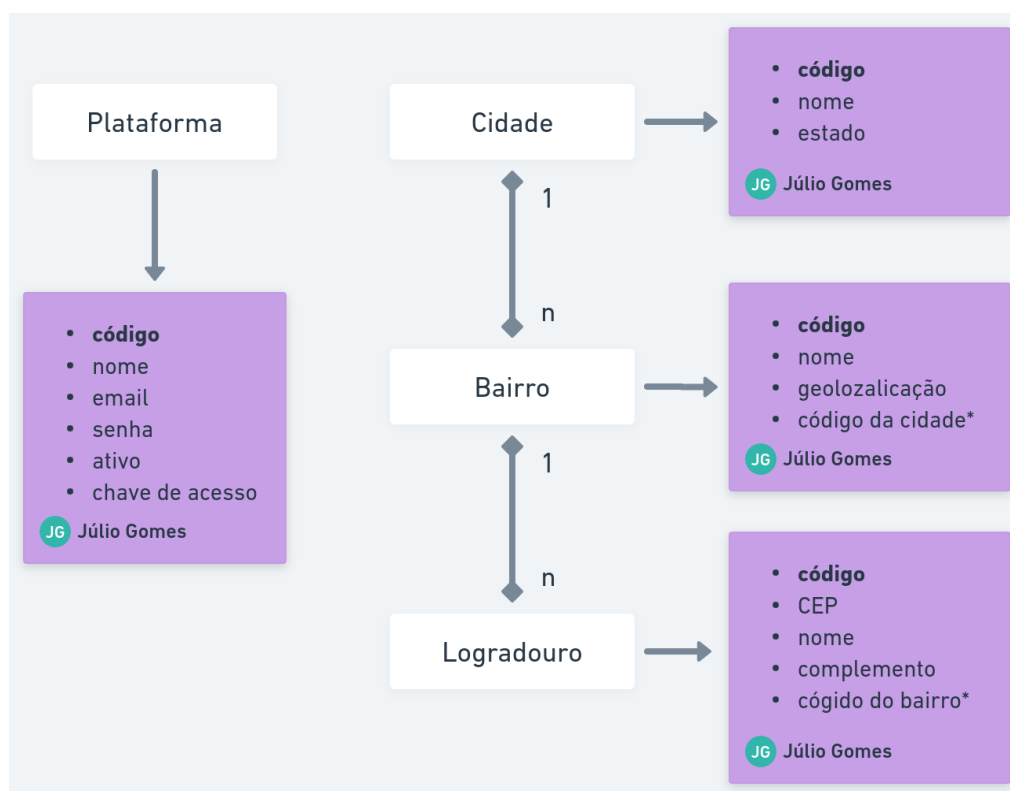
Enquanto isso, para um desenvolvedor ou um técnico especializado em banco de dados, essa abstração não seria suficiente, pois seria preciso obter mais detalhes.

#### 4.3.1 O modelo conceitual

O modelo conceitual é uma descrição mais abstrata do banco de dados, que é independente do Sistema Gerenciador de Banco de Dados (SGBD) utilizado. Ele apresenta as entidades que fazem parte do escopo, suas propriedades e os relacionamentos entre elas.

Na Figura 7, as entidades Plataforma, Cidade, Bairro e Logradouro fazem parte do contexto do sistema, cada qual com suas propriedades, como nome, código, CEP e assim por diante. Além disso, as relações são representadas por meio das setas e o número em cada ponta representa a cardinalidade, que diz respeito às ocorrências daquela relação (HEUSER, 1998).

**Figura 7** - Diagrama Modelo ER conceitual do banco de dados



**Fonte:** Autoria própria.

No diagrama da Figura 7, há a descrição das principais entidades do sistema. A propriedade "código" que aparece em todas, por exemplo, poderia ser implementada como sendo o campo "id" no *MySQL* ou outro SGBD, mas não é a preocupação principal desse nível de abstração. Da mesma forma, as propriedades que utilizam o \* (asterisco) na frente, seriam implementadas como chaves estrangeiras e criariam a relação entre as tabelas.

De acordo com a interpretação das cardinalidades, para cada ocorrência da entidade Cidade, podem existir muitas ocorrências da entidade Bairro. A mesma relação pode ser lida ao contrário: um bairro estará sempre relacionado a uma cidade. Seguindo essa lógica, o mesmo pode ser estendido para a relação entre bairros e logradouros.

#### 4.3.2 O modelo lógico

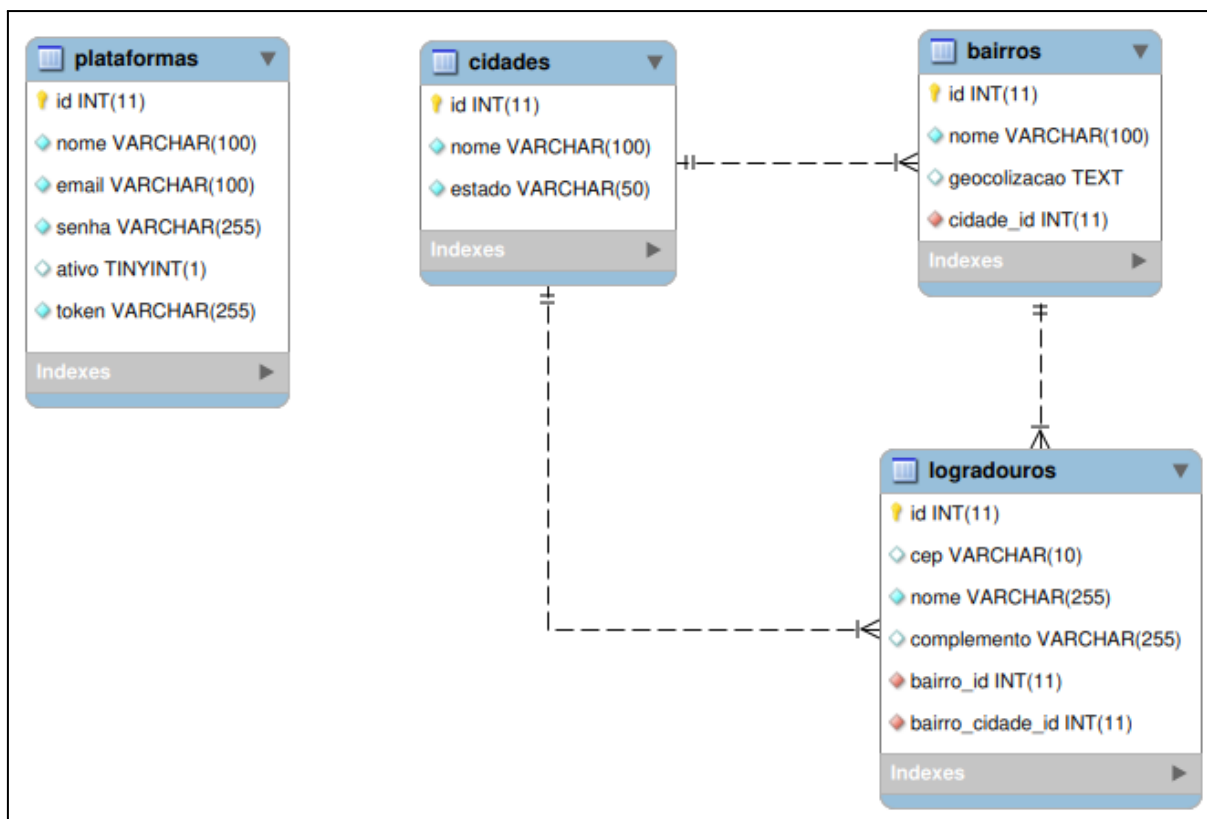
Um modelo lógico é uma descrição de um banco de dados que depende do SGBD utilizado e possui um nível de abstração menor em relação ao modelo conceitual (HEUSER. 1998). Nesse tipo de modelo, é comum ver novos campos e informações acrescentadas pelo SGBD utilizado, como o tipo da coluna e algumas restrições de tamanho, de unicidade, dentre outras.

No mais, alguns atributos como "código" passam a se chamar "id" ou outro padrão previamente definido, além de que não deve haver espaços ou caracteres especiais nas definições de nomes das colunas do banco. Geralmente o padrão "nome\_da\_coluna" é muito utilizado nessas situações.

Outra característica acrescentada nesse nível de abstração é a presença de campos especiais como chave primária e chave estrangeira. Essas duas colunas são importantes para estabelecer a relação entre as tabelas. Esse é um ponto importante no modelo ER, pois diz respeito às relações existentes e que foram observadas na fase anterior de conceituação e identificação de entidades.

Por exemplo, a tabela cidades possui uma chave primária chamada "id" (ou código) que é um identificador único e incremental do tipo inteiro. Enquanto isso, a tabela de bairros possui uma coluna chamada cidade\_id (ou código da cidade) e, como o nome sugere, é o campo que guarda a referência para a coluna "id" da tabela cidades, provocando uma relação entre os dois objetos ou registros do banco de dados.

**Figura 8** - Modelo lógico do banco de dados.



**Fonte:** Autoria própria.

Na Figura 8 acima, o diagrama exibe dados mais técnicos e mais próximos da implementação do banco de dados de acordo com o MySQL, que foi o SGBD escolhido. As entidades foram transformadas em blocos que representam as tabelas e as relações continuam sendo representadas pelas setas e cardinalidades em forma de "pés de galinha", como entre cidades e bairros ou entre bairros e logradouros.

Um adendo especial sobre esse projeto de banco de dados se encontra na relação entre logradouros e cidades, estabelecida através da chave estrangeira "bairro\_cidade\_id" contida na tabela de logradouros. Essa relação não é obrigatória e pode não fazer sentido de acordo com as regras de normalização, uma vez que já existe uma relação entre cidades e bairros, onde os logradouros residem.

No entanto, para facilitar a consulta de logradouros por cidade, se for necessária, os passos para se alcançar esse objetivo diminuem com essa implementação extra.

### 4.3.2 O modelo físico

O modelo físico é o que mais se aproxima da implementação propriamente dita, o que, assim como o modelo lógico, depende do SGBG utilizado. A linguagem utilizada para descrever os comandos que serão executados é o SQL. Tanto para a definição de tabelas e dados do banco, quanto para consultas e modificações, o SQL é a linguagem mais utilizada. Dependendo da *engine* implementada pelo SGBD, as instruções podem sofrer alterações.

Logo em seguida, as figuras abaixo, geradas por meio do *software MySQL Workbench*, apresentam os comandos utilizados para definir a estrutura ou o *schema* do banco de dados, como também as tabelas das entidades do modelo conceitual.

Inicialmente foi preciso criar um banco de dados utilizando o SQL da Figura 9 logo a seguir. O nome "ceps" é arbitrário e referencia o principal propósito da base de dados, que é armazenar os CEPs das regiões de entrega e as relações entre elas.

**Figura 9** - SQL para a criação do banco de dados "ceps".

```
-- Schema ceps
--
CREATE SCHEMA IF NOT EXISTS `ceps` DEFAULT CHARACTER SET utf8mb4 ;
USE `ceps` ;
```

**Fonte:** Autoria própria.

A tabela de cidades descrita na Figura 10 possui um índice, ou *index* na linguagem SQL, chamado `idx_nome`. Os índices são interessantes, entre outras coisas, para melhorar a performance durante as consultas, pois permite a indexação pelo próprio mecanismo do banco de dados, diminuindo a complexidade da operação de seleção. Como a base de dados possui inúmeros logradouros, bairros e cidades, é importante otimizar o máximo possível as consultas.

As tabelas de bairros e logradouros também possuem índices pelo mesmo motivo da tabela de cidades. Além disso, o *schema* de logradouros possui duas chaves estrangeiras, contendo referência para a tabela de cidades e de bairros, respectivamente. Como mencionado anteriormente, essa relação direta com cidades

poderia ser descartada, pois já existe, de forma implícita, a mesma relação através da tabela de bairros que, por sua vez, possui relacionamento com cidades.

**Figura 10** - SQL para a criação da tabela "cidades".

```
-- Table `ceps`.`cidades`
-----
CREATE TABLE IF NOT EXISTS `ceps`.`cidades` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(100) NOT NULL,
  `estado` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `idx_nome` (`nome` ASC),
ENGINE = InnoDB
AUTO_INCREMENT = 10444
DEFAULT CHARACTER SET = utf8mb4;
```

**Fonte:** Autoria própria.

**Figura 11** - SQL para a criação da tabela "bairros".

```
-- Table `ceps`.`bairros`
-----
CREATE TABLE IF NOT EXISTS `ceps`.`bairros` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(100) NOT NULL,
  `geocolizacao` TEXT NULL DEFAULT NULL,
  `cidade_id` INT(11) NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `cidade_id` (`cidade_id` ASC),
  INDEX `idx_nome` (`nome` ASC),
  CONSTRAINT `bairros_ibfk_1`
    FOREIGN KEY (`cidade_id`)
    REFERENCES `ceps`.`cidades` (`id`))
ENGINE = InnoDB
AUTO_INCREMENT = 45991
DEFAULT CHARACTER SET = utf8mb4;
```

**Fonte:** Autoria própria.

A tabela de plataformas não possui uma relação com as outras, o que é um caso a ser analisado. Todavia, nessa versão mais simplificada, somente as

informações como email, senha e *token* já são suficientes para manter o registro das plataformas que eventualmente farão a integração com a API.

**Figura 12** - SQL para a criação da tabela "logradouros".

```
-- Table `ceps`.`logradouros`
-----
CREATE TABLE IF NOT EXISTS `ceps`.`logradouros` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `cep` VARCHAR(10) NULL DEFAULT NULL,
  `nome` VARCHAR(255) NOT NULL,
  `complemento` VARCHAR(255) NULL DEFAULT NULL,
  `bairro_id` INT(11) NOT NULL,
  `bairro_cidade_id` INT(11) NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `bairro_id` (`bairro_id` ASC),
  INDEX `bairro_cidade_id` (`bairro_cidade_id` ASC),
  CONSTRAINT `logradouros_ibfk_1`
    FOREIGN KEY (`bairro_id`)
    REFERENCES `ceps`.`bairros` (`id`),
  CONSTRAINT `logradouros_ibfk_2`
    FOREIGN KEY (`bairro_cidade_id`)
    REFERENCES `ceps`.`cidades` (`id`))
ENGINE = InnoDB
AUTO_INCREMENT = 732764
DEFAULT CHARACTER SET = utf8mb4;
```

**Fonte:** Autoria própria.

**Figura 13** - SQL para a criação da tabela "plataformas".

```
-- Table `ceps`.`plataformas`
-----
CREATE TABLE IF NOT EXISTS `ceps`.`plataformas` (
  `id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(100) NOT NULL,
  `email` VARCHAR(100) NOT NULL,
  `senha` VARCHAR(255) NOT NULL,
  `ativo` TINYINT(1) NULL DEFAULT 1,
  `token` VARCHAR(255) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4;
```

**Fonte:** Autoria própria.

No contexto desse trabalho, cada tabela foi pensada para ser o mais simples possível para facilitar as consultas ao banco de dados e retornar as informações necessárias. Se novas informações forem necessárias, podem ser inseridas em forma de novas colunas nas tabelas já existentes. No entanto, a estrutura atual é suficiente para atender aos requisitos funcionais da API.

#### 4.3.3 Construção da base de dados

Após a definição das tabelas, foi preciso popular a base de dados com as informações de cidades, bairros e logradouros. Encontrar esses dados de maneira estruturada é bastante difícil, tanto pela escassez de projetos que tentam resolver esse problema, quanto pela ausência de serviços que disponibilizam bancos de dados relacionais com essas informações.

O *site* oficial dos Correios possui algumas páginas de busca, em que o usuário pode realizar pesquisas a partir do nome da cidade, do bairro ou do logradouro. Um dos problemas nesse tipo de busca é que o texto não pode conter erros de digitação, o que é difícil quando há irregularidades nos nomes de muitos locais do Brasil. Por exemplo, há cidades cujo os nomes possuem duas formas de escrita. Logo, não é um método muito confiável de consulta.

Outros sistemas similares ao dos Correios não possuíam uma forma fácil de relacionar um bairro a uma cidade, por exemplo, ou de retornar logradouros de um mesmo bairro. Essas buscas deveriam ser feitas de forma separada, uma por vez, e a relação entre os resultados da pesquisa ficava por responsabilidade do usuário.

Portanto, criar uma base de dados relacional facilitaria as consultas e permitiria expandir os tipos de rotas existentes, como as que serão apresentadas em maior detalhe nas próximas seções. No entanto, ainda havia a dificuldade de juntar esses dados e criar as relações.

Para a construção dessa aplicação foi utilizada uma base de dados pronta, extraída do *site* dos Correios, contendo um único arquivo de texto com quase 1 milhão de linhas de CEPs, seus respectivos logradouros, bairros e cidades. A base era do ano de 2018 e foi encontrada através do *site cep.la*, que também permite realizar consultas por meio de uma caixa de texto.

Contudo, os dados não estavam prontos ainda para serem inseridos no banco criado anteriormente. Foi preciso adaptar as linhas do arquivo com extensão *.txt*

para criar *queries* do tipo *insert*, para criar os registros nas tabelas. Com esse intuito, foi feito um *script* na linguagem *Python*, que percorria todas as linhas do arquivo e, para cada uma, realizava as seguintes operações:

1. Verificar se a cidade já existe no banco de dados
  - a. Se não existir, deve criar e retornar o identificador (ID).
  - b. Se existir, apenas retornar o identificador (ID).
2. Verificar se o bairro já existe no banco de dados
  - a. Se não existir, deve criar e retornar o identificador (ID).
  - b. Se existir, apenas retornar o identificador (ID).
3. Com os IDs da cidade e do bairro, deve-se inserir o logradouro junto ao seu CEP no banco, contendo as referências ou chaves estrangeiras da cidade e do bairro ao qual ele pertence.

A partir desse algoritmo, foi possível obter um conjunto de dados mais coeso e com os relacionamentos criados entre eles, de forma a facilitar as consultas da API ao banco de dados. O arquivo de texto utilizado como base está disposto nos anexos desse trabalho, bem como o *script* é apresentado nos apêndices.

## 4.4 Implementação da API

A API desenvolvida utiliza o padrão *REST* e, portanto, implementa as práticas dessa arquitetura, como a utilização dos métodos HTTP na comunicação entre cliente e servidor. Por se tratar de um serviço de consulta de regiões por CEP, o método de requisição mais utilizado foi o *GET*, que semanticamente significa obter um recurso da API através de um rota.

### 4.4.1 Rotas

As rotas da API são fundamentais para permitir o acesso a determinados recursos. De acordo com o padrão *REST* mencionado, cada rota pode ser acessada por todos os verbos *HTTP*, cabendo ao *framework Express.js* a responsabilidade de tratar cada requisição individualmente, acionar o processamento dos dados e responder com o *status* correto.

A tabela abaixo descreve as rotas criadas para os recursos cidades, bairros e logradouros, como sendo uma referência de consulta para o desenvolvimento da API. A documentação contendo as requisições e respostas se encontram melhor descritos na seção 4.7.

**Tabela 4** - Rotas especiais de CEPs da API.

| <b>Método<br/>HTTP</b> | <b>URI</b>                      | <b>Descrição</b>                                                 |
|------------------------|---------------------------------|------------------------------------------------------------------|
| GET                    | /ceps/cidades/{cep}             | Retorna uma única cidade para o CEP informado                    |
| GET                    | /ceps/cidades/{cep}/bairros     | Retorna todos os bairros de uma cidade com base no CEP informado |
| GET                    | /ceps/bairros/{cep}             | Retorna um único bairro para o CEP informado                     |
| GET                    | /ceps/bairros/{cep}/logradouros | Retorna os logradouros de um bairro para o CEP informado         |
| GET                    | /ceps/logradouros/{cep}         | Retorna um único logradouro para o CEP informado                 |

**Fonte:** Autoria própria.

As rotas listadas na tabela anterior utilizam o CEP como informação essencial para obter as informações buscadas, o que torna possível trazer as regiões de entrega com base na localidade em que se encontra o lojista. Se esse vendedor reside na cidade de Natal/RN, em um logradouro qualquer, ao utilizar seu CEP de origem, a plataforma conseguirá obter qual a cidade, o bairro e o logradouro em que ele se encontra por meio dos respectivos *endpoints*.

A rota */cidades/{cep}/bairros* permite o retorno dos bairros adjacentes, incluindo aquele em que atende o lojista. Dessa forma, ele terá como opções de entrega todos os bairros da cidade em que ele vive ou deseja realizar as entregas. De forma análoga, a rota */bairros/{cep}/logradouros* traz todos os logradouros os quais pertencem ao bairro de origem do lojista, caso ele deseje selecionar locais mais específicos de atendimento.

Por outro lado, a tabela abaixo segue o padrão de rotas mais tradicionais da API, dando oportunidade para que consultas mais diretas sejam realizadas pelas plataformas, utilizando o identificador único de cada região na base de dados. Dessa

maneira, o cliente da API poderá ter mais opções de consulta, dependendo do cenário. Pode haver situações em que a consulta pelo CEP faça mais sentido, outras em que o identificador (ou ID) é suficiente.

**Tabela 5** - Rotas da API.

| <b>Método<br/>HTTP</b> | <b>URI</b>                | <b>Descrição</b>                                                |
|------------------------|---------------------------|-----------------------------------------------------------------|
| POST                   | /plataformas              | Cria uma plataforma de integração e retorna um código de acesso |
| GET                    | /plataformas/{id}         | Retorna informações de uma plataforma                           |
| GET                    | /cidades/{id}             | Retorna uma cidade com base no seu ID informado                 |
| GET                    | /cidades/{id}/bairros     | Retorna todos os bairros de uma cidade com base no ID informado |
| GET                    | /bairros/{id}             | Retorna um único bairro para o ID informado                     |
| GET                    | /bairros/{id}/logradouros | Retorna os logradouros de um bairro para o ID informado         |
| GET                    | /logradouros/{id}         | Retorna um único logradouro para o ID informado                 |

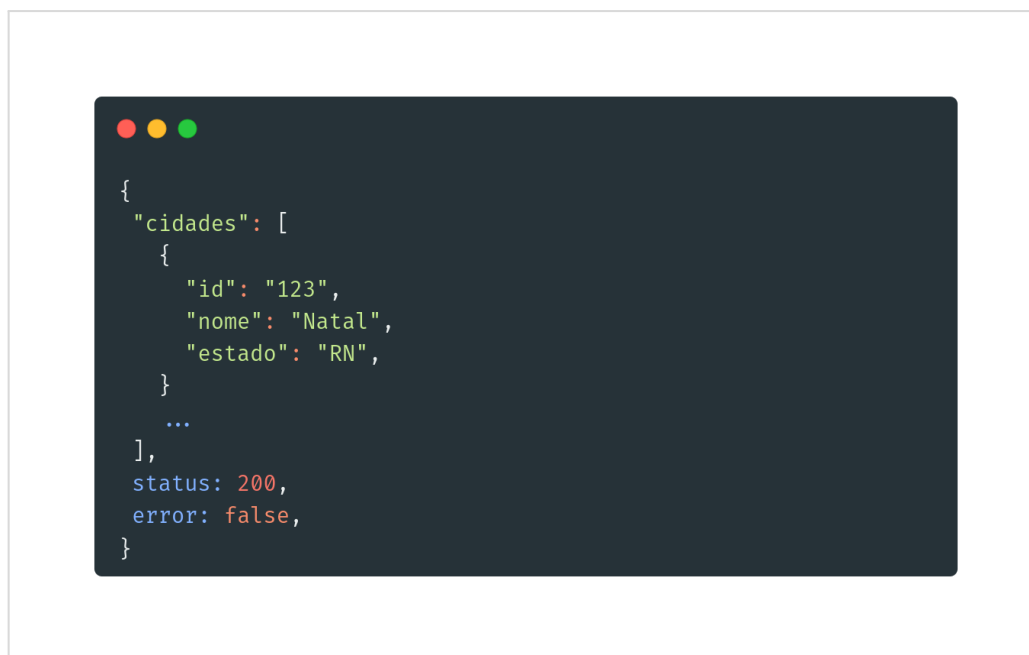
**Fonte:** Autoria própria.

#### 4.4.2 Formato de resposta

Cada rota será responsável por retornar os resultados pedidos na requisição. Há muitos formatos utilizados em *Web Services* e *APIs* para o retorno, como *XML* e *JSON*. Esse último é o mais utilizado atualmente, pois possui uma legibilidade maior em relação a outras alternativas e o seu uso se popularizou junto ao padrão *REST* e ao ecossistema *JavaScript*.

Abaixo segue um exemplo de retorno em *JSON* do *endpoint* /cidades/{cep}. Na imagem, está sendo retornado uma lista ou *array* de cidades. É comum adicionais ao retorno a informação de *status* e de *error*, para que o cliente possa tratar em caso de erro ou estado inesperado.

**Figura 14** - Exemplo do formato *JSON* utilizado na API.



**Fonte:** Autoria própria.

#### 4.4.3 Codificação da API

Seguindo com a implementação da aplicação, esta subseção irá apresentar os procedimentos para a codificação da API. Nesse processo, não foi utilizado um *Object Relational Mapper* (ORM), mas sim um *query builder* chamado *Knex.js*, que entrega uma interface mais amigável para executar consultas no banco de dados. Além disso, a arquitetura de *software* utilizada é a mais simples possível, organizando as rotas de acordo com seus recursos.

As figuras abaixo apresentam o código-fonte de algumas das rotas criadas na API (as que consultam por CEP) a título de compreensão e demonstração da arquitetura *REST* utilizada e das consultas realizadas no banco de dados através do *Knex.js*. O objetivo é comentar brevemente sobre os passos tomados para implementar os principais *endpoints* da API e ressaltar a importância da modelagem de dados.

##### **GET /ceps/cidades/{cep}**

A Figura 15 exibe o código-fonte utilizado para realizar a consulta de uma cidade por CEP. O dado é recebido através do parâmetro */:cep* na *URL*, que é uma

das maneiras pelas quais o *Express.js* é capaz de receber e interpretar parâmetros passados na requisição.

**Figura 15** - Código-fonte para a consulta de uma cidade por CEP.

```

1 // /ceps/cidades/{cep}
2 app.get('/ceps/cidades/:cep', async (req, res) => {
3   const { cep } = req.params;
4   if(!cep) return res.status(404).json({ error: "CEP inválido ou não encontrado!" });
5
6   const cepFormatted = cep.replace('-', '');
7   try {
8     const cidade = await knex('logradouros as l')
9       .join('cidades as c', 'c.id', '=', 'l.bairro_cidade_id')
10      .where('cep', `${cepFormatted}`)
11      .first("c.id", "c.nome", "c.estado")
12
13     if(!cidade) {
14       return res.status(404).json({
15         error: "Cidade não encontrada! Verifique o CEP e tente novamente."
16       });
17     }
18
19     return res.status(200).json({ cidade, error: false });
20   } catch(err) {
21     console.log("err", err);
22     return res.status(500).json({ error: "Erro inesperado ocorreu!" });
23   }
24 });

```

**Fonte:** Autoria própria.

Logo em seguida é verificado se o valor do CEP existe e, em caso afirmativo, é feita uma simples conversão para normalizar o seu formato, removendo possíveis hífens ou separadores. O bloco *Try-Catch* é necessário para tratar exceções que possam ocorrer, impedindo que o serviço pare de funcionar em caso de erros inesperados. Como está se realizando uma consulta no banco de dados, eventuais erros de conexão ou execução de comandos podem acontecer e precisam ser percebidos e capturados imediatamente.

Os códigos entre as linhas 8 e 11 da Figura 15 são responsáveis por realizar a consulta, utilizando funções do *query builder* como *join*, *where* e *first*. O método *join* é equivalente à instrução SQL *join*. O mesmo é válido para os demais métodos utilizados, com exceção do *first*, que retorna o primeiro resultado da consulta.

A modelagem de dados foi importante para estabelecer os relacionamentos entre as entidades e possibilitar consultas como essa, na qual a cidade é retornada após uma consulta por CEP, que é realizada juntamente à tabela *logradouros*, que

possui esse dado. Um logradouro pertence à uma cidade e essa relação é estabelecida pela referência (chave estrangeira) existente entre as duas tabelas, definida pelos campos `bairro_cidade_id` e `id` (`idades.id`).

No final, caso não haja erros, é retornado um objeto contendo a cidade encontrada após a operação de *join* entre as tabelas `idades` e `logradouros`. Uma cidade também pode ser consultada através do seu identificador único. Porém, nesse caso, para facilitar as consultas das plataformas, o CEP também se torna uma opção mais viável nessas situações.

### GET /ceps/cidades/{cep}/bairros

O código-fonte a seguir retorna uma lista de bairros da mesma cidade com base em um CEP recebido por parâmetro. Da linha 8 à linha 10, é buscado o logradouro, a entidade que detém a informação de CEP. Caso for encontrado algum, são trazidos os bairros que pertencem à mesma cidade. O primordial para realizar essa consulta é a coluna `bairro_cidade_id` na tabela de logradouros, que referencia a tabela de cidades.

**Figura 16** - Código-fonte para a consulta de bairros por CEP no *Insomnia*.

```

1 // /ceps/cidades/{cep}/bairros
2 app.get('/ceps/cidades/:cep/bairros', async (req, res) => {
3   const { cep } = req.params;
4   if(!cep) return res.status(404).json({ error: "CEP inválido ou não encontrado!" });
5
6   const cepFormatted = cep.replace('-', '');
7   try {
8     const logradouro = await knex('logradouros')
9       .where('cep', cepFormatted)
10      .first();
11
12     if(!logradouro) {
13       return res.status(404).json({
14         error: "CEP não encontrado! Verifique o CEP e tente novamente"
15       });
16     }
17
18     const bairros = await knex('bairros')
19       .select('*')
20       .where('cidade_id', logradouro.bairro_cidade_id)
21
22     return res.status(200).json({ bairros, error: false });
23   } catch(err) {
24     console.log("err", err);
25     return res.status(500).json({ error: "Erro inesperado ocorreu!" });
26   }
27 });

```

**Fonte:** Autoria própria.

## GET /ceps/cidades/{cep}/bairros

De acordo com a Figura 17, o código-fonte busca encontrar todos os logradouros para um determinado bairro com base no CEP informado na requisição. Primeiro se procura um logradouro na respectiva tabela cujo CEP é igual ao informado, após ser formatado corretamente. Na linha 18, a lista de logradouros é buscada com base no campo `bairro_id` do objeto logradouro que foi retornado na linha 8.

**Figura 17** - Código-fonte para a consulta de logradouros de um bairro por CEP no *Insomnia*.

```

1 // /ceps/bairros/{cep}/logradouros
2 app.get('/ceps/bairros/:cep/logradouros', async (req, res) => {
3   const { cep } = req.params;
4   if(!cep) return res.status(404).json({ error: "CEP inválido ou não encontrado!" });
5
6   const cepFormatted = cep.replace('-', '');
7   try {
8     const logradouro = await knex('logradouros')
9       .where('cep', cepFormatted)
10      .first();
11
12     if(!logradouro) {
13       return res.status(404).json({
14         error: "CEP não encontrado! Verifique o CEP e tente novamente"
15       });
16     }
17
18     const logradouros = await knex('logradouros')
19       .select('*')
20       .where('bairro_id', logradouro.bairro_id);
21
22     return res.status(200).json({ logradouros, error: false });
23   } catch(err) {
24     console.log("err", err);
25     return res.status(500).json({ error: "Erro inesperado ocorreu!" });
26   }
27 });

```

**Fonte:** Autoria própria.

## 4.5 Testes

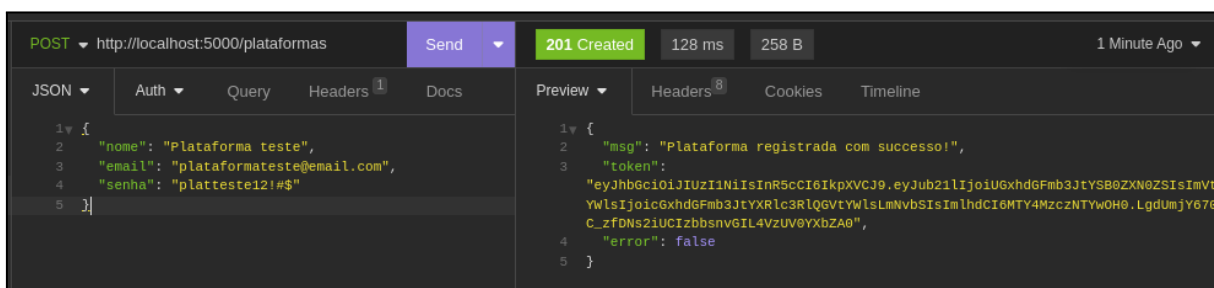
Os testes realizados utilizaram o *software Insomnia*, que é um cliente HTTP *open source* com vários recursos para construir requisições no padrão *REST* utilizando os métodos conhecidos. Esses testes são essenciais para garantir o funcionamento da API em diferentes cenários.

Abaixo segue exemplos de testes em algumas das principais rotas do sistema. Com exceção da rota para criação de uma nova plataforma, todas as outras precisam utilizar o *token* associado à sua plataforma para ter acesso aos recursos da API. O *token* ou código de acesso deve ser passado, quando necessário, por meio do *header* de *authorization*, que é um cabeçalho comumente utilizado para a autenticação nos sistemas.

## POST /plataformas

O primeiro teste realizado foi o do cadastro de uma nova plataforma, através da rota */plataformas* com o método *POST*. O corpo da requisição continha nome, email e senha de testes. O retorno da requisição é um *JSON* contendo uma mensagem de sucesso, seguido do *token JWT* gerado durante o processo de inserção da nova plataforma. Esse *token* deve ser armazenado pela plataforma como forma de autenticação para futuras consultas. Por fim, uma propriedade *error* com o valor *false*, acusando que tudo ocorreu bem.

**Figura 18** - Teste de cadastro de plataforma usando o *Insomnia*.



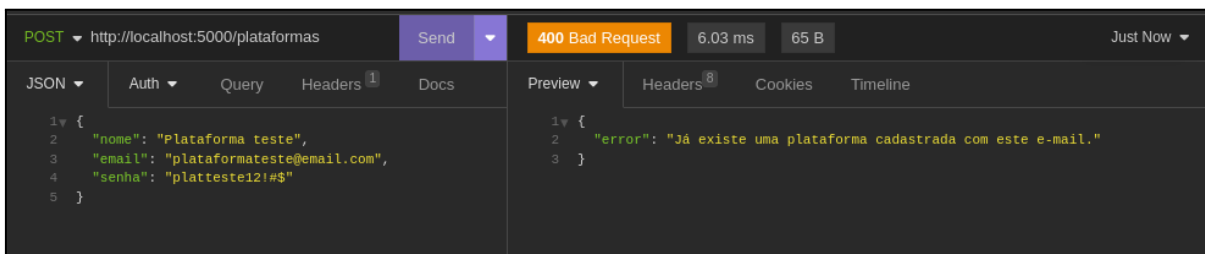
**Fonte:** Autoria própria.

No caso de uma nova tentativa utilizando os mesmos dados, inclusive o mesmo e-mail, um erro é retornado, avisando que já existe uma plataforma cadastrada com essa informação e, portanto, não é possível cadastrar novamente. Esse tratamento é importante para evitar duplicatas na base de dados e manter a consistência dos registros.

Após o cadastro da plataforma, há outra rota relevante nesse fluxo, que serve para retornar informações de uma plataforma cadastrada, caso exista, como o código de acesso (*token*) gerado no momento do cadastro. Para isso, é preciso

utilizar e-mail e senha como forma de obter acesso a esses dados. Essas credenciais são validadas e retornam positivamente caso estejam corretas.

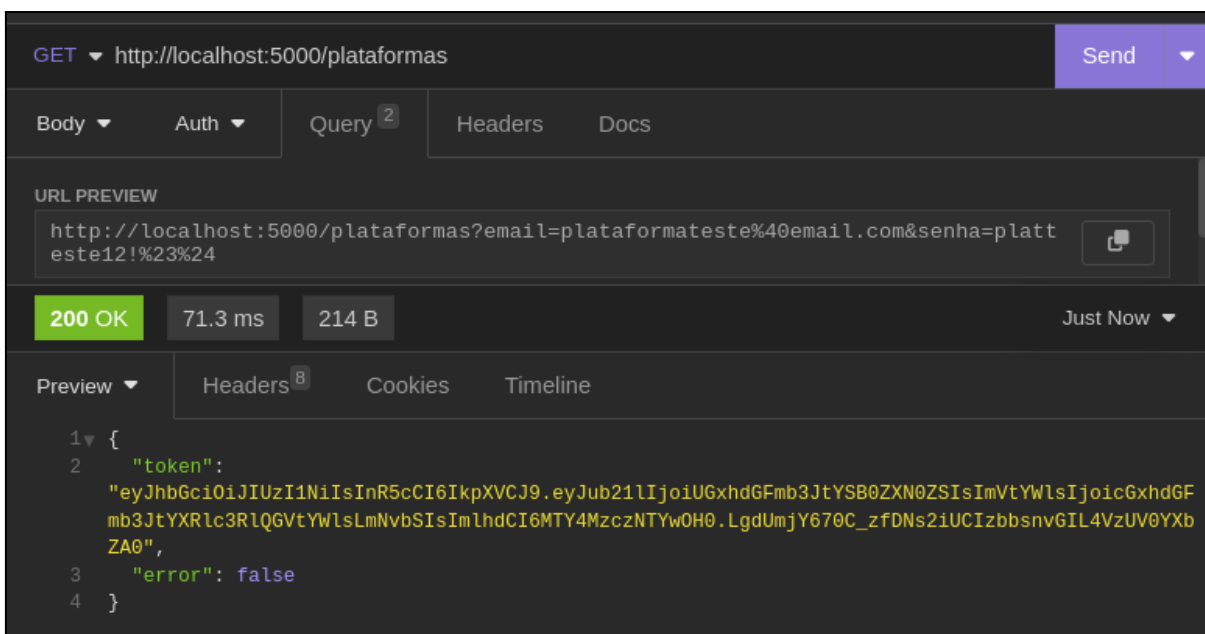
**Figura 19** - Erro ao tentar cadastrar uma nova plataforma com o mesmo e-mail.



Fonte: Autoria própria.

## GET /plataformas

**Figura 20** - Obter dados de uma plataforma por e-mail e senha no *Insomnia*.

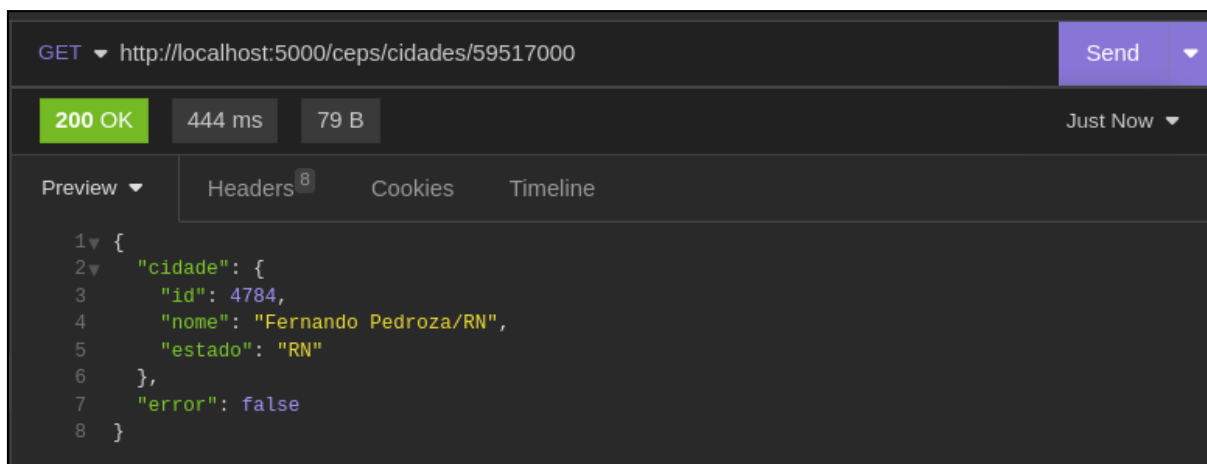


Fonte: Autoria própria.

## GET /ceps/cidades/{cep}

O teste seguinte foi realizado em uma das rotas principais da aplicação: `/ceps/cidades/{cep}`, que retorna uma cidade com base no CEP passado como parâmetro na requisição. Nesse caso, o valor do CEP foi 59517000, que é referente a um logradouro da cidade de Fernando Pedroza no interior do Rio Grande do Norte.

**Figura 21** - Teste consultando cidade por CEP de um logradouro no *Insomnia*.

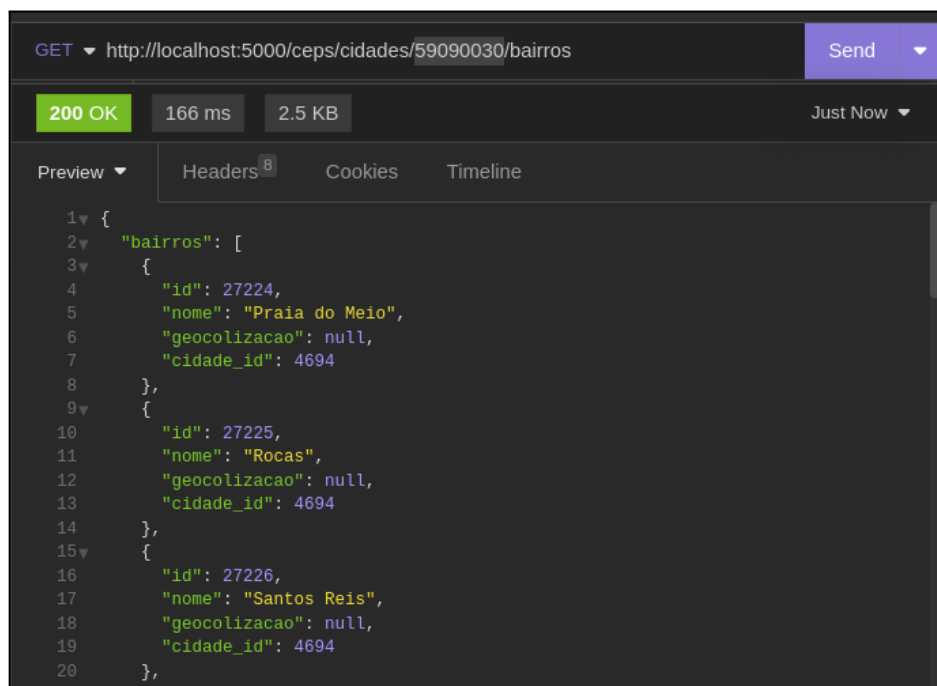


Fonte: Autoria própria.

### GET /ceps/cidades/{cep}/bairros

O teste realizado nessa rota deve retornar uma lista de bairros da mesma cidade ao qual pertence o CEP passado como parâmetro. Nesse caso, o CEP é de uma rua da cidade de Natal/RN. Como resposta, a requisição obteve uma lista de bairros da mesma cidade para que a plataforma apresente para o lojista selecionar os bairros como regiões de entrega dos produtos.

**Figura 22** - Consulta de todos os bairros de uma cidade por CEP.

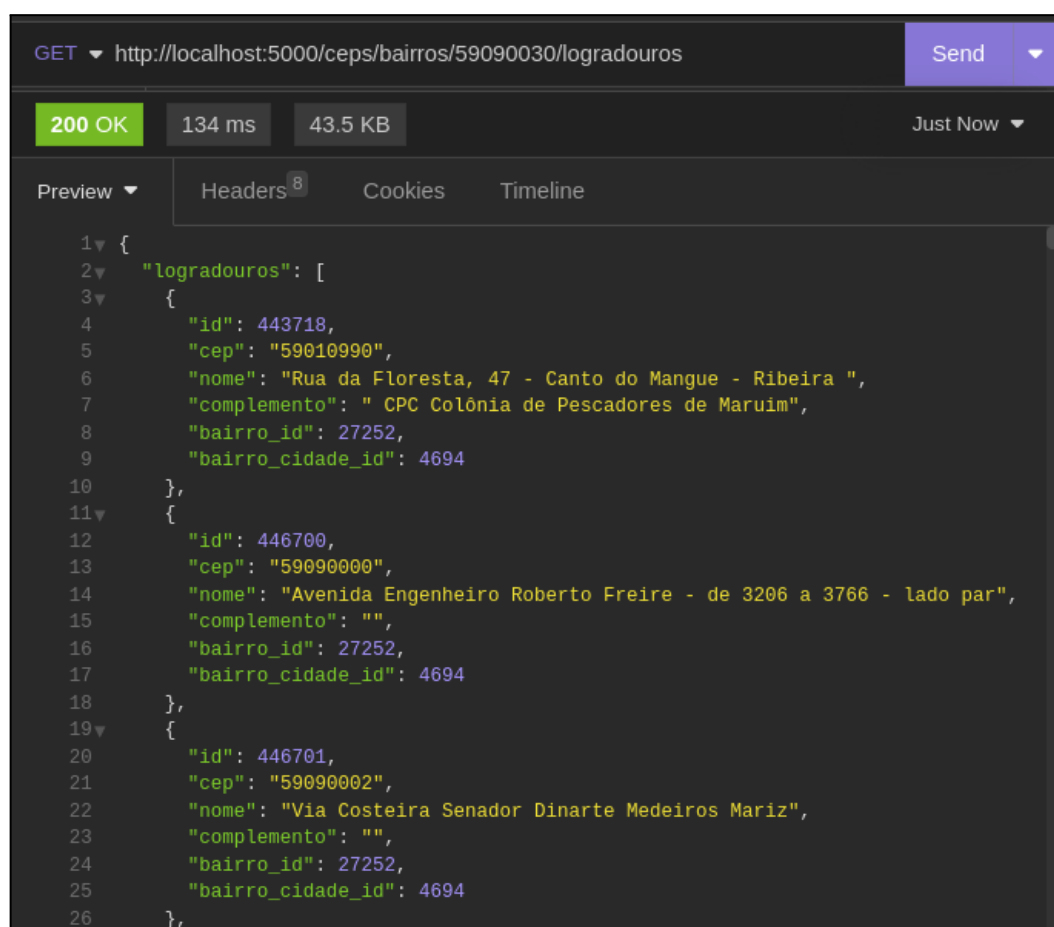


Fonte: Autoria própria.

### /ceps/bairros/{cep}/logradouros

Essa rota deve retornar todos os logradouros de um bairro dado um CEP de origem. No caso do teste abaixo, o CEP pertence ao bairro de Ponta Negra em Natal/RN. Como esperado, é retornada uma lista com todos os logradouros e CEPs pertencentes a esse bairro. Esse *endpoint* visa ajudar os lojistas que costumam entregar somente em logradouros conhecidos.

**Figura 23** - Consulta de todos os logradouros de um bairro por CEP no *Insomnia*.



**Fonte:** Autoria própria.

## 4.6 Documentação do software

Segundo Sommerville (2011), os agilitas discutem a relevância e o foco na documentação de um *software* que está em constante evolução, uma vez que esses documentos podem não acompanhar as frequentes alterações de código. Não que a documentação seja complementarmente dispensável, mas o foco deve ser nas

entregas. No entanto, o mesmo autor afirma que *software* também é a documentação e todos os componentes associados.

Em uma API *REST*, a documentação é essencial para que os interessados consigam fazer a integração da melhor maneira possível e sanar eventuais dúvidas no percurso. Há diversas ferramentas e modos de se documentar um *software* desse tipo, mas o mais importante, entretanto, é a clareza nas descrições do que cada rota faz e retorna. O cliente da API não precisa saber da implementação, mas deve estar ciente das interações e erros que podem acontecer.

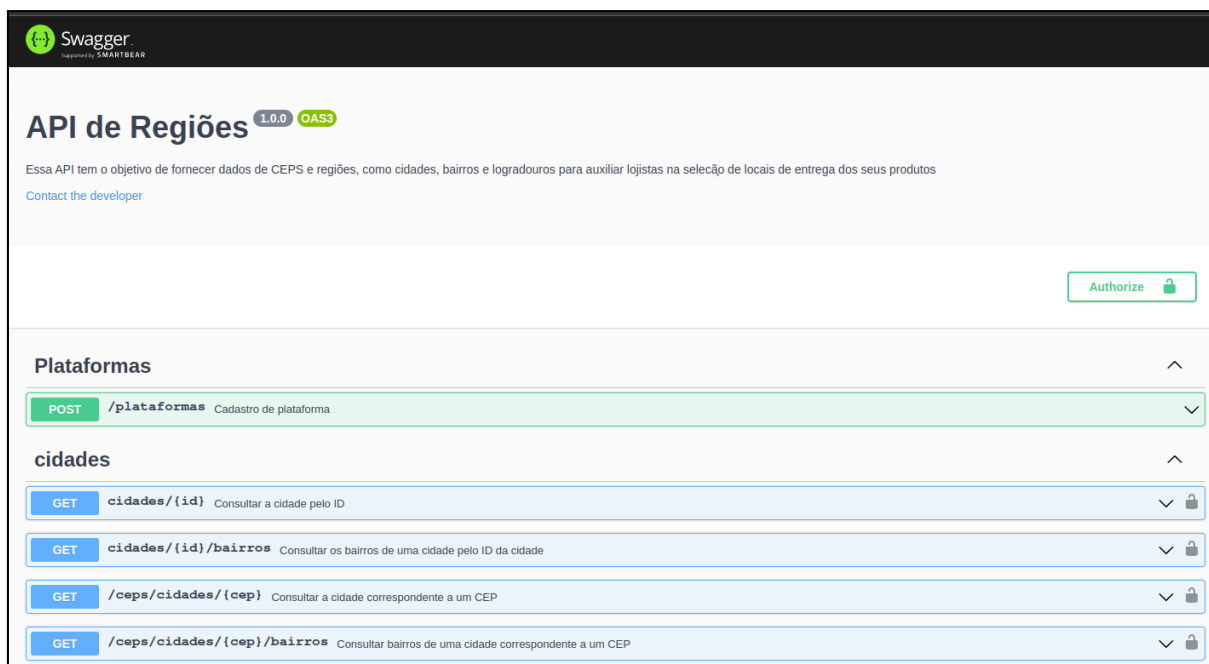
O *Swagger* é uma dessas ferramentas que facilitam a criação e manutenção de uma API bem documentada. Ele permite que sejam definidas todas as rotas da API, o que é esperado em cada requisição e quais são as possíveis respostas do *endpoint*. Além disso, é capaz de definir ambientes diferentes para testes, cada qual com uma *URL* específica. No mais, o *Swagger* também pode ser utilizado como cliente para testes assim como o *Insomnia*, mas seu principal objetivo é auxiliar na elaboração das descrições da API.

O *Express.js* possui uma biblioteca chamada "*swagger-ui-express*" que pode ser adicionada ao projeto para auxiliar no desenvolvimento da documentação com o *Swagger*. É preciso configurar um arquivo de nome "*swagger.json*" com todas as informações requeridas para que o *Swagger* consiga interpretar e exibir corretamente. Também foi necessário criar uma rota especial chamada *"/api-docs"* para expor a documentação criada pelo *Swagger*.

A Figura 24 mostra a parte inicial da documentação da aplicação com *Swagger* com base no arquivo *JSON* definido, utilizando o padrão *OpenAPI* na versão 3.0.0. As rotas são agrupadas por "tags" como plataformas e cidades, onde ficam as respectivas descrições de cada uma. Antes da tabela de rotas e ao final de cada linha há um ícone de cadeado, que significa um pedido de autorização para acessar o *endpoint* correto. Nesse momento, o cliente pode fornecer o *token* gerado ao criar uma nova plataforma para que todas as requisições sejam autenticadas.

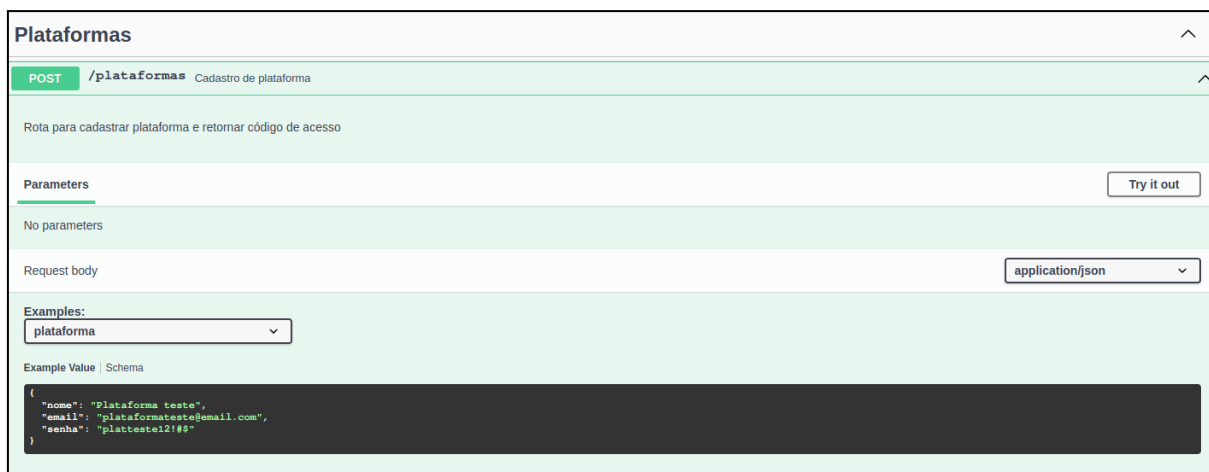
A Figura 25 representa a rota de plataformas, onde é possível cadastrar uma nova passando alguns parâmetros como e-mail e senha. Como essa rota é responsável por gerar o *token*, ela não precisa de autenticação prévia. Também é possível observar que é dado um exemplo com dados fictícios de como criar uma plataforma e quais campos devem ser passados no corpo da requisição.

**Figura 24** - Documentação da API com Swagger.



**Fonte:** Autoria própria.

**Figura 25** - Documentação da criação de nova plataforma.



**Fonte:** Autoria própria.

A Figura 26 demonstra as possíveis respostas do servidor, com a descrição de cada *status*. Logo, o desenvolvedor que for realizar a integração possui uma visão clara do que deve tratar em cada caso de erro ou de sucesso, o que orienta também a criação e manipulação das interfaces gráficas para aderir ao tipo de resposta da API.

**Figura 26** - Documentação das respostas da API para criar nova plataforma.

| Responses |                                                                                                                                                                                                                 |          |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| Code      | Description                                                                                                                                                                                                     | Links    |
| 201       | <p>Plataforma registrada com sucesso!</p> <p>Media type<br/> <input type="text" value="application/json"/></p> <p>Controls Accept: header.</p> <p>Example Value   Schema</p> <pre>{   "token": "string" }</pre> | No links |
| 400       | Já existe uma plataforma cadastrada com este e-mail.                                                                                                                                                            | No links |
| 500       | Não foi possível criar a plataforma.                                                                                                                                                                            | No links |

Fonte: Autoria própria.

## 4.7 Validação de uso da API

Nesta seção é apresentado um exemplo de uso da API desenvolvida para fins de mostrar como esse serviço pode ser integrado por alguma plataforma, cabendo ao próprio cliente a responsabilidade pela interface do usuário e experiência de uso. É assim pois cada plataforma possui sua identidade visual, ideia e valores, assim com guias de estilos e de usabilidade.

Há várias maneiras de utilizar os dados da API para facilitar o uso da seleção de endereços de entrega. A Figura 27 apresenta uma interface simples, intuitiva e direta para cumprir com essa finalidade. Ela foi desenvolvida utilizando tecnologias da *Web* e fazendo uso da *interface fetch* do *JavaScript* para realizar as consultas na API.

O objetivo é possibilitar que, a partir de um CEP de origem, o usuário lojista consiga visualizar as alternativas de regiões de entrega, sem precisar cadastrar manualmente faixas de CEP. Esse processo, que antes era exaustivo e demandava muito tempo, se torna transparente ao lojista e cabe à plataforma que está integrando realizar as devidas consultas na API desenvolvida para obter os CEPs das regiões selecionadas.

Na Figura 27, há um campo para inserção do CEP de origem do lojista, ou da unidade em que ele realiza as entregas. Ao clicar no botão "OK", é realizada a consulta na API, que retorna todos os bairros da cidade ao qual pertence o CEP. No caso do exemplo, o CEP 01153000 pertence a algum logradouro da cidade de São

Paulo/SP. Os bairros são exibidos logo abaixo em caixas para serem selecionadas pelo usuário.

**Figura 27** - Exemplo de utilização da API para seleção de regiões de entrega.

**Selecionar regiões de entrega**  
Marque as regiões de entrega de suas encomendas

01153000

Bairros da cidade de São Paulo/SP

|                 |                                    |                                  |                            |
|-----------------|------------------------------------|----------------------------------|----------------------------|
| Sé<br>R\$ 12,00 | Centro<br>R\$ 15,50                | Brás                             | República                  |
| Luz             | Ponte Pequena                      | Bom Retiro                       | Canindé<br>R\$ 5,00        |
| Barra Funda     | Várzea da Barra Funda<br>R\$ 25,00 | Parque Industrial<br>Tomas Edson | Santa Cecília<br>R\$ 56,00 |
| Perdizes        | Campos Elíseos                     | Santa Efigênia                   | Vila Buarque               |

**Fonte:** Autoria própria.

Ao realizar a seleção, é exibido uma caixa de diálogo (ou *modal*) para inserir o preço que será cobrado em entregas naquela região selecionada. As regiões marcadas em azul foram as selecionadas nesse exemplo. Ao clicar no botão "Continuar" logo mais abaixo, a plataforma pode seguir com outros passos, caso existam.

A API, como mostrado durante o desenvolvimento, também disponibiliza os logradouros de um bairro ou de toda a cidade, dando à plataforma integrada a oportunidade de apresentar essas opções de escolha da maneira que achar mais adequada e na ordem que preferir.

## 5 CONSIDERAÇÕES FINAIS

Esta seção apresenta as conclusões do trabalho, passando brevemente pelos objetivos da pesquisa, os problemas enfrentados e os resultados alcançados ao final do projeto. Além disso, para que o trabalho possa ser continuado no futuro e incrementado cada vez mais, a seção 5.2 sugere possibilidades de adições ao projeto para torná-lo ainda mais útil ao que se propõe.

### 5.1 Conclusões

O objetivo deste trabalho foi implementar uma aplicação que solucionasse uma problemática presente em algumas plataformas de *e-commerce*, que prejudica o fluxo de configuração do módulo de entrega via motoboy ou *delivery* de lojas virtuais. Nessa linha, a pesquisa:

- Apresentou o processo de identificação do problema na *startup* potiguar;
- Buscou construir um referencial teórico basilar com conceitos e definições úteis para o trabalho;
- Realizou um estudo de campo junto à coleta de dados com o intuito de compreender o cenário onde a problemática estava inserida e solidificar a proposta da aplicação desenvolvida ao final;
- Construiu uma base de dados relacional a partir de dados não relacionados, possibilitando consultas mais simples e diretas e;
- Implementou uma API *REST* capaz de retornar dados de regiões e CEPs para que as plataformas consigam melhorar a experiência dos usuários em relação à seleção e ao cadastro de novas regiões de entrega.

No entanto, algumas dificuldades se apresentaram durante o trajeto e devem ser ressaltadas. Primeiro, obter dados confiáveis, completos e atualizados de regiões e CEPs no Brasil é uma tarefa bastante complicada, tendo em vista as alterações constantes como o surgimento de novos bairros e logradouros. Soma-se a isso, a própria delimitação e nomenclatura dessas localidades, que pode ser complexa e conter várias divergências.

Segundo, estabelecer relacionamentos entre essas regiões, muito por causa do problema citado anteriormente, se tornou mais complicado do que aparentava ser

a princípio. Há, por exemplo, cidades do interior que não possuem bairros muito bem delimitados ou tem apenas um endereço de CEP, o que resultava em dados incompletos ou inconsistentes durante a fase de modelagem de dados. Em alguns casos, foi percebido que havia endereços postais sem associação a um logradouro ou a um bairro.

Apesar desses empecilhos, o resultado final conseguiu atender às expectativas quanto ao retorno das regiões de entrega, auxiliando as plataformas de *e-commerce* na solução dessa problemática. Em suma, o trabalho atingiu seu principal objetivo, percorrendo todas as etapas previstas desde o início, que foram demonstradas neste presente documento.

## 5.2 Trabalhos futuros

Por se tratar de um problema amplo, há algumas sugestões do que pode ser abordado em futuros trabalhos, incluindo novas implementações tais quais:

1. Permitir a atualização de regiões de entrega e CEPs para sempre manter a base de dados atual e consistente. Os usuários poderiam enviar pedidos ou solicitações para criar, mudar ou excluir alguma região de entrega.
2. Disponibilizar mais informações sobre cidades, bairros e logradouros, como complementos, observações e distâncias em relação a algum determinado ponto central para fins de calcular prazos estimados de entrega.
3. Adicionar dados geográficos para possibilitar a visualização das regiões de entrega em mapas simplificados. Essa é uma alternativa excelente e que torna a aplicação mais rica em detalhes, pois não há muitas soluções completas nesse sentido devido a dificuldade de obter esses dados.
  - a. Uma sugestão de base para obter dados de geolocalização é o *OpenStreetMaps* que possui uma enorme quantidade de informações, incluindo pontos delimitados para construção de polígonos em mapas.
  - b. O *Nominatim* é uma aplicação que utiliza o *OpenStreetMaps* para realizar consultas por países, estados, cidades, bairros, ruas e outras delimitações políticas e geográficas. Ele trabalha com diversos dados que podem ser úteis para complementar ainda mais a aplicação construída neste trabalho.

## REFERÊNCIAS

AGILE MANIFESTO. Agile Manifesto For Software Development. Disponível em <<https://agilemanifesto.org/>>. Acesso em: 02 de out. 2022.

CAMBRIDGE Dictionaries Online. Disponível em: <<http://dictionary.cambridge.org/>>. Acessado em outubro de 2022. Acesso em: 01 de out. 2022.

CORREIOS. **Tudo sobre CEP**. Disponível em: <<https://www.correios.com.br/enviar/precisa-de-ajuda/tudo-sobre-cep>>. Acesso em: 12 de mai. 2023.

CEPMAIS. **O que significa a palavra logradouro?** Disponível em: <<https://www.cepmais.com.br/o-que-e-logradouro.html>>. Acesso em 13 de mai. 2023.

CNN Brasil. **Gasto com delivery sobre em 2021 e tendências de consumo pós pandemia**. Sofia Kercher, 2022. Disponível em: <<https://www.cnnbrasil.com.br/business/gasto-com-delivery-sobe-24-em-2021-veja-tendencias-de-consumo-do-pos-pandemia>>. Acesso em: 05 de out. 2022.

DE MENDONÇA, Herbert Garcia. **E-commerce**. Revista Inovação, Projetos e Tecnologias, v. 4, n. 2, p. 240-251, 2016. ed. São Paulo: Atlas, 2017. Acesso em: 02 de out. 2022.

EXAME. **Eis a primeira coisa a ser vendida pela internet**. Disponível em: <<https://exame.com/marketing/eis-a-primeira-coisa-a-ser-vendida-pela-internet-na-historia>>. Acesso em: 02 de out. 2022.

EXPRESSJS. **Framework web rápido, flexível e minimalista para Node.js**. Disponível em: <https://expressjs.com/>. Acesso em: 07 de abr. 2023.

FAGUNDES. **O que é e-commerce?** Artigo do Blog Fagundes. Disponível em: <<https://efagundes.com/artigos/o-que-e-e-commerce/>>. Acesso em: 05 de out. 2022.

FERREIRA, Crismanda Maria. **O trabalho via aplicativos: Exponenciação do valor na era flexível-digital**. Temporalis, v. 21, n. 41, p. 353-366, 2021.

FIELDING, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures**. University of California, Irvine, 2000.

GIT. **Sistema de controle de versão distribuído**. Disponível em <<https://git-scm.com/>>. Acesso em: 09 de out. 2022.

HEUSER, Carlos Alberto. **Projeto de Banco de Dados**. Instituto de Informática da UFRGS. Editora Sagra. 4 edi, 1998.

INSOMNIA. **Cliente HTTP para construir, depurar e testar APIs**. Disponível em: <<https://insomnia.rest/>>. Acesso em 12 de mai. 2023.

IMB. **What is a REST API?**. Disponível em: <<https://www.ibm.com/topics/rest-apis>>. Acesso em: 01 de abr. 2023.

KNEX.JS. **Query builder para Node.js**. Disponível em: <<https://knexjs.org/>>. Acesso em 19 de abr. 2023.

LOJA INTEGRADA. **O que é uma faixa de CEP**. Disponível em: <<https://ajuda.lojaintegrada.com.br/pt-BR/articles/939050-o-que-e-uma-faixa-de-cep>> Acesso em: 08 de out. 2022.

MACHADO, Felipe Nery Rodrigues. **Projeto e implementação de banco de dados**. 3. ed. São Paulo; Érica, 2014.

MARCONI, M. D. A.; LAKATOS, E. M. **Fundamentos de Metodologia Científica**. 8.

MATA, Késley Brenner da Costa. **E-commerce: análise de dados sobre o comércio eletrônico no Brasil**. 2021.

MDN Web Docs. **Mozilla Developer Network Web Documentation**. Disponível em: <<https://developer.mozilla.org/>>. Acesso em: 09 de out. 2022.

MICHAELIS. **Dicionário Brasileiro da Língua Portuguesa**. Disponível em: <<https://michaelis.uol.com.br/>>. Acesso em: 13 de mai. 2023.

MySQL. **Site oficial do MySQL**. Disponível em: <<https://www.mysql.com/>>. Acesso em: 10 de out. 2022.

MYSQL WORKBENCH. **Site oficial do MySQL Workbench**. Disponível em: <<https://www.mysql.com/products/workbench/>>. Acesso em: 22 de mai. 2023.

NEOTRUST. **E-commerce brasileiro tem o melhor faturamento dos últimos anos em janeiro**. Disponível em: <<https://neotrust.com.br/2022/04/08/e-commerce-brasileiro-tem-melhor-faturamento-dos-ultimos-anos-em-janeiro>>. Acesso em: 16 de out. 2022.

NIELSEN, Jakob. **Usabilidade na Web**. Elsevier Brasil, 2007.

Node Js. **Site oficial e documentação**. Disponível em: <<https://nodejs.org>>. Acesso em: 09 de out. 2022.

PINHO, Cátia Alexandra Soares. **Determinantes da intenção de adesão às plataformas online C2C**. 2015. Tese de Doutorado.

Portal PPI GOV. **Estudos relativos aos correios**. Disponível em: <<https://portal.ppi.gov.br/estudos-relativos-aos-correios>>. Acesso em: 15 de out. 2022.

PRESSMAN, Roger S. **Engenharia de Software: uma abordagem profissional**. 7. ed. Porto Alegre. AMGH Editora Ltda, 2011.

PRODANOV, Cleber Cristiano; DE FREITAS, Ernani Cesar. **Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico-2ª Edição**. Editora Feevale, 2013.

RAMOS, Eduardo. **E-commerce**. Editora FGV, 2015.

ROSA, João Roberto Conceição. Marketplace no Brasil: desafios, vantagens e tendências deste modelo de negócio para empresas varejistas. **São Paulo: Faculdade FIA de Administração e Negócios**, 2019.

SEBRAE. **Uma breve definição sobre o comércio online**. Disponível em: <<https://www.sebrae.com.br/sites/PortalSebrae/artigos/uma-breve-definicao-sobre-o-comercio-online,08cfa5d3902e2410VgnVCM100000b272010aRCRD>>. Acesso em: out. 2022.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

SWAGGER. **API Documentantion e Design Tools for Teams**. Disponível em: <<https://swagger.io/>>. Acesso em 20 de abr. 2023.

Tray Commerce. **Base de conhecimento: Frete por faixa de CEP**. Disponível em: <<https://basedeconhecimento.tray.com.br/hc/pt-br/articles/6735972422939-Frete-e-Envio-Tabela-por-Faixa-de-CEP>>. Acesso em: 09 de out. 2022.

WINCKLER, Marco; PIMENTA, Marcelo Soares. **Avaliação de usabilidade de sites Web**. Escola de Informática da SBC Sul (ERI 2002). Porto Alegre, v. 1, p. 85-137, 200.

## APÊNDICE - *Script* na linguagem *Python* para transformar o arquivo do Anexo A em uma base de dados relacional

Python

```
import mysql.connector

# Conectar ao banco de dados
cnx = mysql.connector.connect(
    user="root", password="", host="localhost", database="ceps"
)
cursor = cnx.cursor(buffered=True)

# Iterar em cada linha do arquivo lido 'ceps.txt'
with open("ceps.txt", "r") as f:
    i = 0
    for line in f:
        print(i)
        fields = line.strip().split("\t")
        length = len(fields)

        cep = ""
        cidade = ""
        bairro = ""
        logradouro = ""
        complemento = ""

        cep = fields[0]
        cidade = fields[1]
        if length > 2:
            bairro = fields[2]
        if length > 3:
            logradouro = fields[3]
        if length > 4:
            complemento = fields[4]

        # Checar se existe um registro com o mesmo nome da
        cidade na table cidades
        query = "SELECT id FROM cidades WHERE nome = %s limit
1"
        cursor.execute(query, (cidade,))
```

```

        cidade_result = cursor.fetchone()

        cidade_id = cidade_result[0]

        # Checar se o campo neighborhood_field já existe na
        # tabela bairros
        query = "SELECT id, cidade_id FROM bairros WHERE nome =
        %s and cidade_id = %s limit 1"
        cursor.execute(query, (bairro, cidade_id))
        bairro_result = cursor.fetchone()

        # Se não existir ou tiver um cidade_id diferente,
        # insira e retorne o seu id
        if not bairro_result:
            query = "INSERT INTO bairros (nome, cidade_id)
VALUES (%s, %s)"
            cursor.execute(query, (bairro, cidade_id))
            bairro_id = cursor.lastrowid
        else:
            bairro_id = bairro_result[0]

        # Insert into logradouros
        query = "INSERT INTO logradouros (cep, nome,
complemento, bairro_cidade_id, bairro_id) VALUES (%s, %s, %s,
%s, %s)"
        cursor.execute(query, (cep, logradouro, complemento,
cidade_id, bairro_id))
        i += 1

# Commitar as mudanças e encerrar/terminar a conexão
cnx.commit()
cursor.close()
cnx.close()
cnx.close()

```

## ANEXO - Estrutura do arquivo de texto utilizado como base de dados

---

|                         |              |        |                                 |                                |
|-------------------------|--------------|--------|---------------------------------|--------------------------------|
| 01001000                | São Paulo/SP | Sé     | Praça da Sé - lado ímpar        |                                |
| 01001001                | São Paulo/SP | Sé     | Praça da Sé - lado par          |                                |
| 01001010                | São Paulo/SP | Sé     | Rua Filipe de Oliveira          |                                |
| 01001900                | São Paulo/SP | Sé     | Praça da Sé, 108                | UNESP - Universidade Estadual  |
| Júlio de Mesquita Filho |              |        |                                 |                                |
| 01001901                | São Paulo/SP | Sé     | Praça da Sé, 371                | Edifício Santa Lúcia           |
| 01001902                | São Paulo/SP | Sé     | Praça da Sé, 385                | OAB - Ordem dos Advogados do   |
| Brasil.                 |              |        |                                 |                                |
| 01002000                | São Paulo/SP | Sé     | Rua Direita - lado par          |                                |
| 01002001                | São Paulo/SP | Sé     | Rua Direita - lado ímpar        |                                |
| 01002010                | São Paulo/SP | Sé     | Praça do Patriarca              |                                |
| 01002020                | São Paulo/SP | Centro | Viaduto do Chá                  |                                |
| 01002900                | São Paulo/SP | Centro | Viaduto do Chá, 15              | Prefeitura do Município de São |
| Paulo                   |              |        |                                 |                                |
| 01002901                | São Paulo/SP | Sé     | Rua Direita, 32                 | Edifício Piratininga           |
| 01002902                | São Paulo/SP | Sé     | Rua Direita, 191                | Edifício Zogbi                 |
| 01002903                | São Paulo/SP | Sé     | Rua Direita, 250                | Edifício Barão de Iguape       |
| 01003000                | São Paulo/SP | Sé     | Rua José Bonifácio - lado par   |                                |
| 01003001                | São Paulo/SP | Sé     | Rua José Bonifácio - lado ímpar |                                |
| 01003010                | São Paulo/SP | Sé     | Praça Ouvidor Pacheco e Silva   |                                |
| 01003900                | São Paulo/SP | Sé     | Rua José Bonifácio, 24          | Edifício Triângulo             |
| 01003901                | São Paulo/SP | Sé     | Rua José Bonifácio, 93          | Edifício Santa Cruz            |
| 01003902                | São Paulo/SP | Sé     | Rua José Bonifácio, 209         | Edifício Brasília              |
| 01003903                | São Paulo/SP | Sé     | Rua José Bonifácio, 250         | Edifício Ouvidor               |
| 01003904                | São Paulo/SP | Sé     | Rua José Bonifácio, 278         | Edifício Barão de Piracicaba   |
| 01003905                | São Paulo/SP | Sé     | Rua José Bonifácio, 367         | Edifício Serasa                |
| 01004000                | São Paulo/SP | Sé     | Rua Barão de Paranapiacaba      |                                |
| 01004010                | São Paulo/SP | Sé     | Rua Quintino Bocaiúva           |                                |
| 01004900                | São Paulo/SP | Sé     | Rua Quintino Bocaiúva, 71       | Edifício Major Porkis          |
| 01004901                | São Paulo/SP | Sé     | Rua Quintino Bocaiúva, 176      | Edifício Casa das Arcadas      |
| 01004902                | São Paulo/SP | Sé     | Rua Quintino Bocaiúva, 191      | Edifício Santa Rita            |
| 01004903                | São Paulo/SP | Sé     | Rua Quintino Bocaiúva, 231      | Edifício Itacolomi             |
| ... mais de 750.000     |              |        |                                 |                                |

---