UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIAS
CURSO DE ENGENHARIA DA COMPUTAÇÃO

Luziana de Fátima de Oliveira Assunção

Aplicativo de Caronas entre Amigos com Intersecção de Rotas via API Rest

São Luís
2024

Luziana de Fátima de Oliveira Assunção

Aplicativo de Caronas entre Amigos com Intersecção de Rotas via API Rest

Trabalho de Conclusão de Curso II apresentado ao curso de Engenharia da Computação da Universidade Federal do Maranhão como parte dos requisitos necessários para obtenção do grau de Engenheiro(a) da Computação.

Orientador: Prof. Dr. Paulo Rogério de Almeida Ribeiro

São Luís

2024

Luziana de Fátima de Oliveira Assunção

Aplicativo de Caronas entre Amigos com Intersecção de Rotas via API Rest

Trabalho de Conclusão de Curso II apresentado ao curso de Engenharia da Computação da Universidade Federal do Maranhão como parte dos requisitos necessários para obtenção do grau de Engenheiro(a) da Computação.

Prof. Dr. Paulo Rogério de Almeida Ribeiro
Orientador

Prof. Dr. Davi Viana dos Santos
Examinador

Prof. Dr. Sérgio Souza Costa
Examinador

São Luís
2024

Agradecimentos

Primeiramente, agradeço a Deus e a Jesus por sempre colocarem as pessoas certas em minha vida, pelo sustento, amor e sabedoria.

Agradeço à minha família, especialmente ao meu pai, Márcio Hugo, às minhas mães, Luciana e Ana Luiza, aos meus irmãos, Davi, Letícia, Luís e Cosme, e ao meu tio Paulo, por todo o apoio e incentivo.

Também sou grata a todos os meus amigos e colegas que me ajudaram, direta ou indiretamente, ao longo dessa jornada.

Aos professores do curso de Engenharia da Computação, que sempre oferecem suporte e orientação aos alunos, deixo meu sincero agradecimento.

Por fim, um agradecimento especial à meu orientador, professor Paulo Rogério de Almeida Ribeiro, por todo o auxílio prestado durante este trabalho, sempre com paciência e compreensão.

"Porque o Senhor dá a sabedoria;
da sua boca é que vem
o conhecimento e o entendimento."

Provérbios 2:6 *"Bíblia"*

Resumo

Este trabalho apresenta um aplicativo de caronas integrado a uma rede social, que visa reduzir problemas de mobilidade urbana, como congestionamentos e poluição, incentivando o uso eficiente de veículos. O diferencial está na rede social, permitindo que os usuários compartilhem viagens com amigos de confiança, sem alteração significativa das rotas dos motoristas.

O aplicativo oferece funcionalidades como registro, login, adição de amigos e criação de rotas, utilizando geolocalização para otimizar caronas. A interseção das rotas é feita por consultas a um banco de dados espacial com PostGIS. O sistema inclui uma API REST, desenvolvida em Java com Spring Boot, e uma interface em React Native, com rotas geradas pelo OSRM.

O sistema foi validado através de estudos de caso, que confirmaram a correta aplicação das regras de segurança e interseção de rotas. Nos testes realizados: (1) A, amigo de B, teve sua carona exibida para B, pois suas rotas se intersectaram; (2) C, também amigo de B, não teve a carona exibida devido à ausência de interseção de rotas; e (3) A e C, apesar de suas rotas se cruzarem, não visualizaram as caronas um do outro por não serem amigos, de acordo com as regras de segurança do sistema.

Com base nos resultados, o aplicativo atingiu seus objetivos, contribuindo para a redução de problemas relacionados à mobilidade urbana, como congestionamentos e emissões de poluentes, ao incentivar o compartilhamento de caronas entre amigos.

Palavras-chave: Compartilhamento, Aplicativo, Carona, Segurança.

Abstract

This work presents a carpooling app integrated with a social network, aiming to reduce urban mobility issues like traffic congestion and air pollution by encouraging more efficient vehicle use. The key feature is the social network, allowing users to share rides with trusted friends, without significantly altering drivers' usual routes.

The app offers functionalities such as registration, login, adding friends, and creating routes, using geolocation to optimize carpools. Route intersections are determined through spatial database queries using PostGIS. The system includes a REST API, developed in Java with Spring Boot, and a React Native interface, with routes generated by OSRM.

Tests confirmed the system's effectiveness and security rules, allowing ride visibility only between friends with compatible routes. The app successfully achieved its goals of improving urban mobility and reducing emissions.

Keywords: Carpooling, App, Ridesharing, Security.

Lista de ilustrações

Figura 1 – Municípios com a maior quantidade de automóveis 11
Figura 2 – Mortes Violentas Brasil 2011 - 2020 11
Figura 3 – Blablacar - Caronas e Ônibus 12
Figura 4 – Cabify 13
Figura 5 – Uber 13
Figura 6 – Arquitetura REST 18
Figura 7 – Pictogramas para as formas e relações espaciais 20
Figura 8 – Fonte: Costa, 2018 20
Figura 9 – Exemplo de latitude e longitude 22
Figura 10 – Diagrama Entidade Relacionamento 26
Figura 11 – Diagrama de Sequência: Cadastrar Carona 28
Figura 12 – Diagrama de Sequência: Buscar Carona 28
Figura 13 – Diagrama de Sequência: Página Minhas Caronas 29
Figura 14 – Diagrama de contexto 30
Figura 15 – Swagger - Api Carona Friend 31
Figura 16 – Lista de amizades de cada usuário envolvido no processo de carona. . . 33
Figura 17 – Dados da Carona ofertada por A. 34
Figura 18 – Dados da Busca por carona de B. 35
Figura 19 – Detalhes da carona de A. 36
Figura 20 – Detalhes da carona de B. 36
Figura 21 – Dados da carona aceita por B e informações, para A, sobre os amigos que estão em sua carona. Como A e B são amigos, a carona foi exibida para B e aceita por ele, mostrando os dados de quem irá participar da carona. Para A, são exibidos os amigos que aceitaram sua carona. . . . 37
Figura 22 – Dados da carona ofertada por C. 38
Figura 23 – Dados da busca por carona de B. 39
Figura 24 – Dados da carona solicitada por B, que não possui interseção com nenhuma outra rota, e informações para C, indicando que nenhum amigo aceitou sua carona. 40
Figura 25 – Dados da Busca por carona de C. 41
Figura 26 – Dados da carona solicitada por C, que possui interseção com a carona de A, mas, por não serem amigos, não aparece na lista de interseções. Informações para A indicam que somente B irá participar de sua carona. 42

Lista de abreviaturas e siglas

API	*Application Programming Interface*
APK	*Android Package*
BSD	*Berkeley Software Distribution*
CO-OP	*Cooperative Routing Optimization Protocol*
DATASUS	*Departamento de Informática do Sistema Único de Saúde*
HTTP	*Hypertext Transfer Protocol*
IPEA	*Instituto de Pesquisa Econômica Aplicada*
OMG	*Object Management Group*
OSRM	*Open Source Routing Machine*
RA	*Área de Roteamento*
REST	*Representational State Transfer*
Senatran	*Secretaria Nacional de Trânsito*
Sinfo	*Superintendência de Informática*
SIG	*Sistemas de Informação Geográfica*
SGBD	*Sistema de Gerenciamento de Banco de Dados*
SRID	*Spatial Reference System Identifier*
SQL	*Structured Query Language*
TCC	*Trabalho de Conclusão de Curso*
UFMA	*Universidade Federal do Maranhão*
UFF	*Universidade Federal Fluminense*
UFRN	*Universidade Federal do Rio Grande do Norte*
UML	*Unified Modeling Language*

Sumário

1 INTRODUÇÃO 10
1.1 Objetivo Geral 15
1.2 Objetivo Específico 15

2 FUNDAMENTAÇÃO TEÓRICA 16
2.1 Protocolos 16
2.2 Diagramas 17
2.3 BackEnd 18
2.4 Banco de Dados 19
2.5 Modelos de Banco de Dados 19
2.6 FrontEnd 22

3 METODOLOGIA 24

4 RESULTADOS 31

5 CONCLUSÃO 43

REFERÊNCIAS 44

Anexo: Código SQL 46

1 Introdução

A Organização Mundial da Saúde (OMS) define a poluição do ar como a contaminação de ambientes internos e externos por agentes químicos, físicos ou biológicos que alteram as características naturais da atmosfera. Esse tipo de poluição é considerado um dos maiores desafios ambientais globais da atualidade (DIAS et al., 2022). Um dos primeiros grandes episódios desse tipo de poluição no mundo moderno aconteceu em Londres em 1952, resultando em sérios problemas de saúde pública e aumento na mortalidade. Desde então, a sociedade tem buscado formas de reduzir os poluentes, investindo em diversas áreas para enfrentar o problema (CASTRO; GOUVEIA; ESCAMILLA-CEJUDO, 2003).

Atualmente, a utilização intensiva do petróleo nas indústrias, termoelétricas e no transporte automotivo é uma das principais fontes de poluição do ar nas cidades ao redor do mundo, aumentando os riscos para a saúde pública e para o meio ambiente. Na América Latina, destacam-se as cidades do México, São Paulo, Rio de Janeiro, Porto Alegre, Belo Horizonte, Recife e Santiago do Chile, que enfrentam poluição significativa devido ao elevado número de veículos automotores nas ruas e avenidas (OLIVEIRA, 2013).

De acordo com a Secretaria Nacional de Trânsito (Senatran), o Brasil possui atualmente 61,2 milhões de carros registrados, o que representa um aumento de 14% em relação ao último estudo divulgado pela Confederação Nacional de Municípios em 2018. Comparando com o Censo Demográfico de 2022 do Instituto Brasileiro de Geografia e Estatística (IBGE), há um carro para cada 3,32 habitantes no país. Na Figura 1 mostra os dez municípios com o maior número de automóveis no Brasil, sendo que nove deles são capitais. A cidade de São Paulo concentra 10,10% da frota nacional, seguida pelo Rio de Janeiro, com 3,50%, e Belo Horizonte, com 2,79% (Confederação Nacional de Municípios, 2023). Já a cidade de São Luís, no estado do Maranhão, registrou um total de 222.900 automóveis que representavam aproximadamente 0,36% da frota nacional (IBGE, 2023).

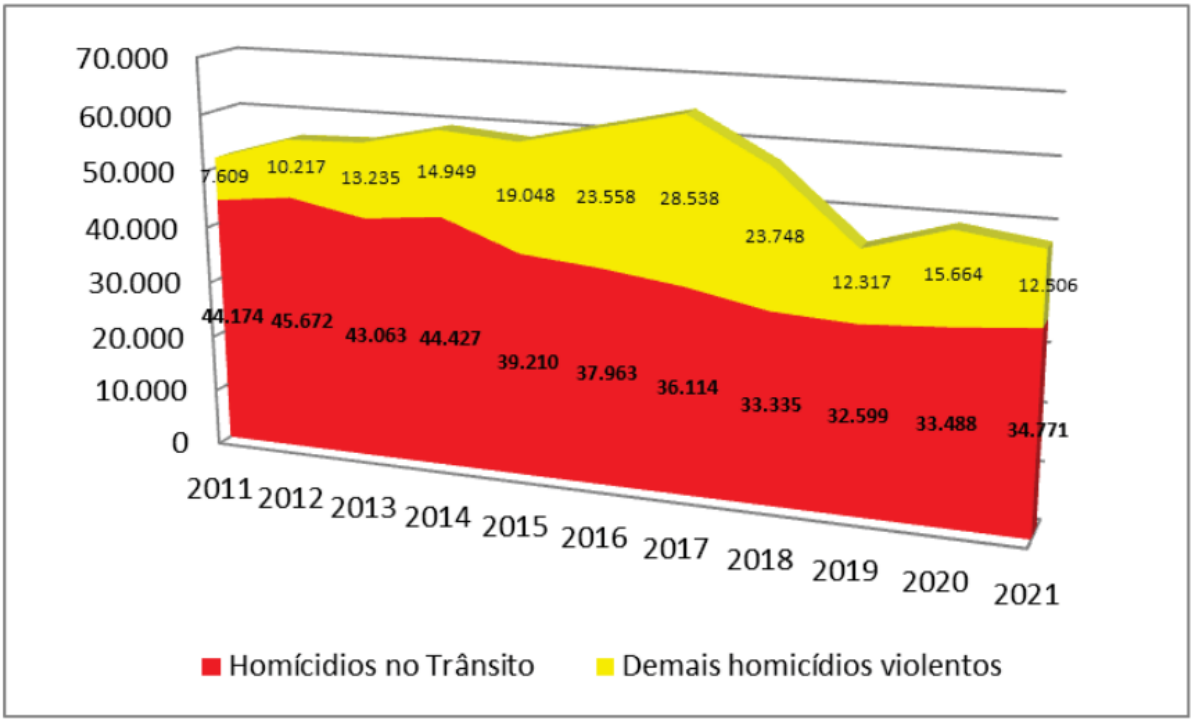
Além disso, entre 2011 e 2021, o Brasil registrou 424.816 mortes no trânsito, de acordo com dados disponíveis no Departamento de Informática do Sistema Único de Saúde (DATASUS) e no Instituto de Pesquisa Econômica Aplicada (IPEA). O país tem historicamente um alto número de mortes violentas anuais, que incluem não apenas acidentes de trânsito, mas também mortes por arma de fogo. No entanto, a maioria das mortes classificadas como violentas no Brasil é, na realidade, resultado de acidentes de trânsito, como mostrado na Figura 2 (GRISA; DOMINGUES, 2023).

Figura 1 Municípios com a maior quantidade de automóveis

Município	Total	%
São Paulo	6.176.807	10,10%
Rio de Janeiro	2.138.226	3,50%
Belo Horizonte	1.706.604	2,79%
Brasília	1.397.177	2,28%
Curitiba	1.141.356	1,87%
Goiânia	658.521	1,08%
Campinas	628.163	1,03%
Salvador	628.094	1,03%
Fortaleza	622.983	1,02%
Porto Alegre	605.657	0,99%

Fonte: Confederação Nacional de Municípios, 2023

Figura 2 Mortes Violentas Brasil 2011 - 2020



Fonte: GRISA; DOMINGUES, 2023

Diante dessa realidade, existem diversas estratégias para mitigar o impacto ambiental e físico decorrente do elevado número de veículos nas vias. Uma dessas estratégias é o Cooperative Routing Optimization Protocol (CO-OP), um sistema inteligente de transporte que utiliza roteamento cooperativo para gerenciar congestionamentos em áreas urbanas. O CO-OP identifica e classifica congestionamentos em uma área específica, chamada Área de Roteamento (RA), e prioriza o roteamento de veículos com maior

urgência, ou seja, aqueles mais próximos do congestionamento e com maior risco de ficarem presos. O objetivo é minimizar e controlar os congestionamentos de forma eficiente (SOUZA et al., 2015).

Assim, este trabalho investiga a implementação de sistemas de caronas como solução para os desafios do transporte de indivíduos. Com o surgimento de aplicativos especializados, que têm se mostrado eficazes ao facilitar o compartilhamento de caronas, essas plataformas se apresentam como alternativas mais sustentáveis e eficientes para lidar com os problemas decorrentes do aumento do uso de veículos. Exemplos desses aplicativos incluem os listados abaixo.

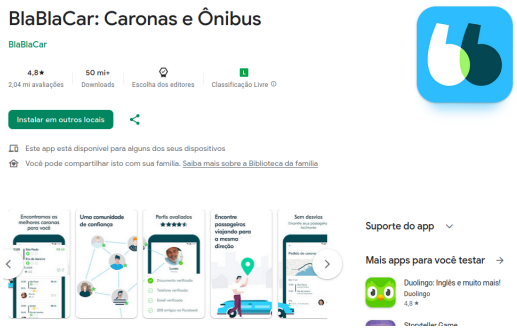
- **Blablacar - Caronas e Ônibus**

  Descrição: BlaBlaCar é a principal rede global de caronas, com mais de 90 milhões de membros em 22 países. A BlaBlaCar contribui para a redução de 1,6 milhões de toneladas de CO2 e promove 120 milhões de conexões humanas anualmente[1] (BlaBlaCar, 2024).

  Gratuidade: ausente.

  Disponibilidade: Android, iOS e web.

Figura 3 Blablacar - Caronas e Ônibus



Fonte: Play Store

- **Cabify**

  Descrição: Na Cabify, conectamos pessoas e empresas com os meios de transporte que melhor atendem suas necessidades, promovendo um modelo de negócios sustentável e ético. Nosso objetivo é melhorar as cidades, oferecendo um futuro com novas possibilidades em mais de 40 cidades, respeitando as pessoas e o meio ambiente[2] (CABIFY, 2024).

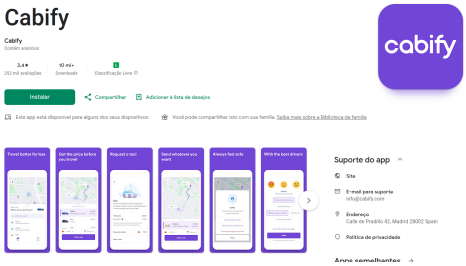
  Gratuidade: ausente.

[1] BlaBlaCar. Disponível em: <https://blog.blablacar.com.br/about-us?_gl=1*qelh4n*_gcl_au*MTU4MjU0MjA4Ni4xNzI0MjAxNTg4>. Acesso em: 20 agosto 2024.

[2] Cabify. Disponível em: <https://cabify.com/en/about-us>. 20 de agosto de 2024

Disponibilidade: Android e iOS.

Figura 4 Cabify



Fonte: Play Store

- **Uber**

  Descrição: Somos uma empresa de tecnologia que conecta o mundo físico ao digital, permitindo que tudo aconteça com um simples toque. Acreditamos que a mobilidade deve ser acessível, segura e sustentável, ajudando você a se mover e ganhar dinheiro sem prejudicar o meio ambiente[3] (Uber, 2024).

  Gratuidade: ausente.

  Disponibilidade: Android e iOS.

Figura 5 Uber

Fonte: Play Store

Além dos aplicativos de carona já disponíveis no mercado, algumas universidades têm se dedicado ao desenvolvimento de soluções próprias para facilitar a mobilidade de seus estudantes. Dois exemplos notáveis são o RideUFF e o Vemcar.

RideUFF é um aplicativo móvel criado para a Universidade Federal Fluminense (UFF), com o objetivo de promover caronas solidárias entre pessoas vinculadas à instituição. O RideUFF permite que os usuários ofereçam ou busquem caronas com datas, horários e destinos específicos. A principal meta do aplicativo é proporcionar conforto e segurança, além de reduzir o tempo de trajeto regular. Ao automatizar o processo de compartilhamento de caronas, o RideUFF se apresenta como uma solução prática para superar as deficiências dos transportes públicos urbanos, frequentemente ineficientes (RESENDE; CUZZUOL, 2019).

[3] Uber. Disponível em: <https://www.uber.com/br/pt-br/>. Acesso em: 20 agosto 2024.

VemCar, por sua vez, é uma iniciativa recente da Superintendência de Informática (Sinfo) da Universidade Federal do Rio Grande do Norte (UFRN). Lançado para dispositivos Android, o Vemcar já registrou mil downloads desde sua disponibilização. O aplicativo é exclusivo para servidores e alunos da UFRN e visa facilitar o compartilhamento de caronas dentro do campus universitário. Após o login e aceitação dos termos, os usuários têm acesso a um tutorial que orienta sobre como solicitar ou oferecer caronas. O Vemcar exibe um mapa com as localizações dos passageiros, permitindo uma coordenação mais eficiente dos trajetos (GADELHA, 2017).

Também podemos citar o SISTEMA DE CARONA BASEADO EM REDE SOCIAL, desenvolvido por Luziana Assunção como trabalho de conclusão de curso na Universidade Federal do Maranhão, como um exemplo de sistema que busca solucionar os problemas de mobilidade urbana através do compartilhamento de caronas. Diferente de outros sistemas que se concentram apenas na conexão entre motoristas e passageiros com base em suas localizações, este sistema valoriza a relação social entre os usuários, exigindo que eles sejam amigos em uma rede social específica para compartilhar uma carona. Essa abordagem, combinada com a utilização de coordenadas geográficas, contribui para a criação de um ambiente mais seguro e confiável para os usuários. No entanto, a plataforma web, apesar de sua acessibilidade, pode apresentar limitações em relação a aplicativos móveis, como a falta de notificações em tempo real e a menor precisão na localização (ASSUNÇÃO, 2019).

Ao analisarmos os aplicativos de carona mencionados, destacam-se quatro aspectos principais:

1. **Interação entre desconhecidos:** As viagens geralmente envolvem indivíduos que não se conhecem previamente.
2. **Custo do serviço:** É comum a presença de uma taxa a ser paga pelo serviço de carona.
3. **Público-alvo específico:** Os serviços são frequentemente destinados a grupos restritos, como apenas universitários.
4. **Ausência de aplicativo:** Alguns serviços de carona não possuem um aplicativo dedicado para facilitar o pedido de carona.

Com o objetivo de oferecer caronas seguras e sem custos para o passageiro (ou com custos previamente acordados de forma voluntária), este trabalho propõe o desenvolvimento de um aplicativo. Este funcionará como uma rede social onde as caronas oferecidas por um motorista estarão disponíveis apenas para passageiros dentro de seu círculo de amizades, ou seja, apenas para aqueles aceitos como amigos na plataforma e que compartilham pontos de partida e destino próximos. É importante notar que o propósito do aplicativo não é

gerar lucro para o motorista, mas sim facilitar a conexão entre motoristas e passageiros com rotas que se cruzam.

Como este trabalho assume a integração com uma rede social e se concentrará em trajetos compartilhados entre passageiros e motoristas, focando na interseção das rotas com os pontos de partida e destino dos usuários que buscam a carona. Assim, o banco de dados não apenas armazenará informações relacionais, como idade e nome, mas também dados espaciais, utilizando a abordagem de Sistema de Informação Geográfica (SIG).

Para o gerenciamento e armazenamento de dados conceituais e espaciais, será utilizado o *PostgreSQL* com a extensão *PostGIS*. A aplicação incorporará a biblioteca de mapas *OpenLayers*, que facilita a inclusão de um mapa dinâmico em qualquer página da web.

A implementação do trabalho será dividida em duas fases: *FrontEnd* e *BackEnd*. O *FrontEnd* será desenvolvido com *React Native*, enquanto o *BackEnd* será construído em *Java* usando a arquitetura *Representational State Transfer (REST)* com o *framework SpringBoot*.

1.1 Objetivo Geral

O objetivo principal deste trabalho é desenvolver um aplicativo que facilite o compartilhamento de caronas entre pessoas que estão conectadas na rede social integrada ao próprio aplicativo.

1.2 Objetivo Específico

- Desenvolver um aplicativo dentro de uma rede social dedicada para motoristas e passageiros.
- Os motoristas podem cadastrar caronas, fornecendo informações sobre a rota, pontos de partida e destino, e disponibilidade.
- Os passageiros podem buscar caronas disponíveis informando apenas o ponto de partida e o destino desejado.
- O sistema verifica se os pontos de partida e destino dos passageiros estão próximos à rota dos motoristas, permitindo a oferta de carona quando apropriado.
- É possível adicionar e gerenciar amigos dentro da rede social do aplicativo.

2 Fundamentação Teórica

Neste capítulo, são abordados os principais conceitos envolvidos no desenvolvimento do trabalho. Esta seção será dividida em cinco partes principais:

- **Protocolos**: Esta parte discutirá os diferentes protocolos que são relevantes para o trabalho. Inclui a descrição dos protocolos de comunicação e interação que serão utilizados no aplicativo, bem como a importância de cada um no contexto do projeto.
- **Diagramas**: Nesta seção, apresentaremos os diagramas que serão usados para representar os principais aspectos da arquitetura do sistema. Serão incluídos diagramas de sequência e diagramas de contexto. Estes diagramas fornecerão uma visualização clara e concisa do fluxo de dados, das interações entre os componentes do sistema e do comportamento geral do aplicativo.
- **BackEnd**: Nesta seção, exploraremos os conceitos associados ao BackEnd do aplicativo, abordando a arquitetura, a linguagem de programação e as ferramentas empregadas no desenvolvimento. Forneceremos uma visão geral dos componentes do servidor e de como eles se comunicam com o FrontEnd. Além disso, discutiremos como os dados serão estruturados e armazenados no banco de dados.
- **Banco de Dados** Nesta seção, discutiremos a modelagem de dados relacionais e espaciais. Focaremos na estrutura das tabelas e tipos de dados, além de explorar o uso de Sistemas de Informação Geográfica (SIG) para dados espaciais. Abordaremos também as ferramentas e tecnologias escolhidas para gerenciar e analisar esses dados, incluindo Sistemas de Gerenciamento de Banco de Dados (SGBDs) especializados em dados geográficos e conceitos de latitude e longitude.
- **FrontEnd**: Esta seção abordará as tecnologias empregadas na criação da interface do usuário do aplicativo. Serão discutidos os frameworks e bibliotecas utilizadas para garantir que a interface seja intuitiva, responsiva e funcional, proporcionando uma boa experiência para o usuário.

2.1 Protocolos

Os protocolos de comunicação são conjuntos de regras que definem como os dados são transmitidos e recebidos entre dispositivos em uma rede. Segundo Tanenbaum e Wetherall, *"um protocolo é um conjunto de regras que governa a comunicação entre entidades que participam de uma rede de computadores"* (TANENBAUM; WETHERALL, 2011).

Neste trabalho, são utilizados os protocolos *Hypertext Transfer Protocol* (HTTP) e *Hypertext Transfer Protocol Secure* (HTTPS). O *HTTP*, de acordo com Fielding e Reschke, é *"um protocolo genérico e sem estado utilizado por sistemas de informação distribuídos, colaborativos e baseados em hipermídia"* (FIELDING; RESCHKE, 2014). O *HTTPS*, por sua vez, adiciona uma camada de criptografia *SSL/TLS* ao *HTTP*, garantindo que os dados transmitidos sejam protegidos contra interceptações e ataques *man-in-the-middle*, o que é essencial para a segurança de informações sensíveis.

A arquitetura do projeto utiliza *REST* para o backend, permitindo que o aplicativo interaja com a *API* para obter e salvar informações. O *REST*, que opera sobre *HTTP/HTTPS*, oferece operações como *GET*, *POST*, *PUT* e *DELETE* para interagir com os recursos do servidor. Detalharemos mais sobre esses conceitos na seção abaixo.

A *API* foi publicada no *Heroku*, uma plataforma de computação em nuvem criada em 2007, que oferece suporte a diversas linguagens de programação. O *Heroku* permite que os usuários criem e implantem suas aplicações de maneira simplificada, utilizando contêineres Unix isolados chamados *dynos*, que fornecem o ambiente de execução necessário. O código pode ser lançado na plataforma via *Git*, com opções de planos gratuitos e pagos, sendo o plano gratuito limitado a um *dyno*, que é interrompido após trinta minutos de inatividade (SOLÓRZANO; CHARAO, 2017).

Essa combinação de tecnologias e protocolos permitiu estabelecer uma comunicação eficiente e segura entre o aplicativo e o backend, garantindo que as funcionalidades da aplicação sejam realizadas de maneira confiável e protegida.

2.2 Diagramas

A Unified Modeling Language (UML) é uma linguagem padrão para a estruturação e desenvolvimento de projetos de software. Ela é amplamente utilizada para criar, especificar e documentar sistemas orientados a objetos (SANTOS, 2008). Atualmente, a UML é mantida pela Object Management Group (OMG)(Object Management Group, 2005).

Os diagramas da UML utilizados neste estudo de caso são:

Os **Diagramas de Sequência** são um tipo específico de diagrama UML que permitem modelar as trocas de mensagens entre objetos de acordo com uma ordem cronológica. Diferente de outros diagramas de interação, os diagramas de sequência mostram a sequência exata das mensagens trocadas entre os objetos, facilitando a compreensão da dinâmica dos processos do sistema. A compreensão dos elementos que compõem um diagrama de sequência é essencial para o entendimento dos comportamentos descritos (JÚNIOR, 2012).

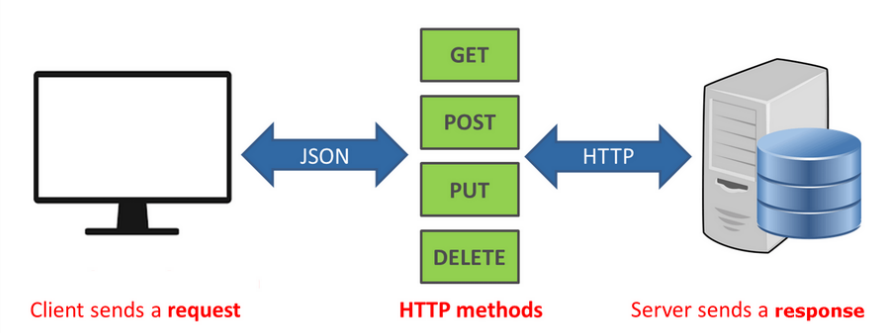
Os **Diagramas de Contexto**, por outro lado, é uma representação visual que mostra um sistema de software e suas interações com o ambiente externo. Ele inclui o sistema central, os atores externos (como usuários e outros sistemas) e as interações entre eles. Esse diagrama é essencial para entender como o sistema se encaixa no ecossistema mais amplo, destacando as trocas de informações e comandos entre o sistema e seus atores externos (BROWN, 2018).

2.3 BackEnd

A arquitetura REST foi utilizada para o desenvolvimento do backend. REST opera sobre o protocolo *HTTP* e oferece um estilo arquitetônico que simplifica a interação com recursos por meio de operações padronizadas como *GET*, *POST*, *PUT* e *DELETE* (FIELDING; RESCHKE, 2014), Figura 6. Segundo Fielding, REST "utiliza os princípios e conceitos do protocolo HTTP para fornecer um conjunto de operações padrão e uma abordagem uniforme para a comunicação entre clientes e servidores"(FIELDING; RESCHKE, 2014). Cada um dos métodos desempenha um papel específico:

- *GET*: Utilizado para recuperar informações do servidor, como listas de itens ou detalhes de usuários.
- *POST*: Usado para enviar dados ao servidor para criar ou modificar recursos, como novos registros.
- *PUT*: Empregado para atualizar recursos existentes.
- *DELETE*: Utilizado para remover recursos do servidor.

Figura 6 Arquitetura REST



Fonte: <https://phpenthusiast.com/blog/what-is-rest-api>

Para a construção da API, foi utilizado o *Spring Boot*, um *framework* baseado em Java que facilita o desenvolvimento de aplicações robustas e escaláveis. De acordo com

a documentação do Spring Boot, *"é um projeto da Spring que ajuda a criar aplicações autônomas e de produção com o mínimo de configuração necessária"* (SOFTWARE, 2024). O Spring Boot oferece uma configuração mínima e rápida, permitindo a criação de APIs RESTful com suporte para validação, segurança e acesso a dados.

Para garantir a segurança e a proteção das APIs, foi implementado o *OAuth* 2.0, um protocolo de autorização que permite que aplicativos acessem recursos em nome de um usuário sem expor credenciais sensíveis. O *OAuth* 2.0 é amplamente utilizado para a autorização de APIs, proporcionando um método seguro para autenticação e autorização de usuários (OAuth, 2024).

Além disso, a documentação da API foi gerada utilizando Swagger, uma ferramenta de documentação para APIs RESTful. Swagger fornece uma interface interativa para explorar e testar as APIs, facilitando a compreensão das funcionalidades disponíveis e dos parâmetros necessários. De acordo com a documentação oficial do Swagger, *"o Swagger é uma ferramenta que ajuda a descrever, consumir e visualizar APIs RESTful"* (Swagger, 2024).

2.4 Banco de Dados

2.5 Modelos de Banco de Dados

Nesta seção, abordaremos a modelagem de dados relacionais e espaciais. Os dados relacionais são organizados em tabelas dentro de um sistema de gerenciamento de banco de dados relacional (SGBDR), permitindo consultas e manipulação complexas. *Codd definiu o modelo relacional como "uma organização de dados em tabelas que podem ser manipuladas através de operações como SELECT, INSERT, UPDATE e DELETE"* (CODD, 1970). Em contraste, dados espaciais referem-se a informações geográficas essenciais para aplicações que requerem análise espacial. Orenstein descreve dados espaciais como *"informações sobre a localização e a forma dos objetos, que podem ser analisadas para entender melhor o contexto geográfico e espacial"* (ORENSTEIN, 1999). Esses dados são cruciais para entender e interpretar a distribuição e a relação entre objetos em um espaço geográfico.

Os modelos de banco de dados são essenciais para a organização e gestão de informações em sistemas computacionais. Inicialmente, os SGBDs estruturavam dados de forma hierárquica ou em rede, mas o modelo relacional, introduzido por Ted Codd, se tornou o padrão predominante ao organizar dados em tabelas. A popularidade das linguagens orientadas a objetos levou à criação de bancos de dados orientados a objetos, que influenciaram os bancos de dados objeto-relacionais, como o PostgreSQL. Atualmente, bancos de dados não relacionais, ou NoSQL, estão sendo explorados, utilizando estruturas como documentos e grafos, refletindo as necessidades dinâmicas das aplicações modernas e

a importância de compreender suas características para desenvolver soluções eficazes e escaláveis (COSTA, 2021).

No modelo conceitual, Longley et al. (2015) enfatizam que a visão do usuário é crucial para identificar e organizar dados, utilizando entrevistas e documentos. O segundo passo envolve definir objetos e seus relacionamentos, onde diagramas simples, como os de entidade e relacionamento, são utilizados para compreender o domínio. Para a representação de dados geográficos, existem duas visões: objetos discretos (geo-objetos), que têm limites definidos, e campos contínuos (geo-campos), que representam variáveis em posições específicas. Pictogramas, como propostos por Shekhar e Chawla (2003), são essenciais para visualizar as formas e relações espaciais, sendo úteis em diagramas de classe. A Figura 8 destaca alguns desses pictogramas. Além disso, o modelo OMT-G, uma extensão do OMT, permite modelar dados espaciais e não espaciais, definindo classes e relacionamentos que facilitam a geoprocessamento no Brasil, promovendo um padrão para a estruturação de dados geoespaciais (COSTA, 2021).

Figura 7 Pictogramas para as formas e relações espaciais

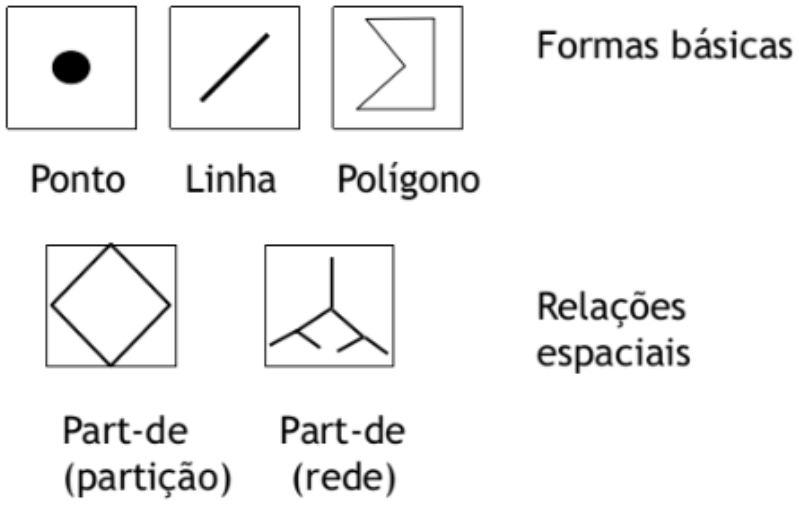


Figura 8 Fonte: Costa, 2018

O modelo de dados para representar a geometria dos geo-objetos é definido pelo Open Geospatial Consortium (OGC), utilizando a classe **Geometry** como base para uma hierarquia de tipos geométricos. Cada objeto geométrico deve estar associado a um sistema de referência, especificado por códigos EPSG, que identificam Sistemas de Referência de Coordenadas (SRC).

No contexto de bancos de dados geográficos como o PostGIS, a criação de tabelas para armazenar dados espaciais requer a definição de um tipo de geometria e um código EPSG, assegurando a correta interpretação das coordenadas.

A OpenGIS padroniza a representação de objetos espaciais em dois formatos principais: Well-Known Text (WKT) e Well-Known Binary (WKB). Esses formatos expressam o tipo do objeto e suas coordenadas, como ilustrado pelos seguintes exemplos:

- `POINT(0 0)`
- `LINESTRING(0 0,1 1,1 2)`
- `POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))`
- `MULTIPOINT((0 0),(1 2))`
- `MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))`
- `MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)), ((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))`
- `GEOMETRYCOLLECTION(POINT(2 3),LINESTRING(2 3,3 4))`

Esses formatos suportam apenas geometrias 2D sem incluir o SRID. O PostGIS, entretanto, estendeu esses formatos para incluir coordenadas 3D, 4D e a especificação do SRID, resultando nos formatos EWKB e EWKT, como mostrado a seguir:

- `POINT(0 0 0)` – XYZ
- `SRID=32632;POINT(0 0)` – XY com SRID
- `POINTM(0 0 0)` – XYM
- `POINT(0 0 0 0)` – XYZM
- `TRIANGLE((0 0, 0 9, 9 0, 0 0))`
- `TIN(((0 0 0, 0 0 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 0 0 0)))`

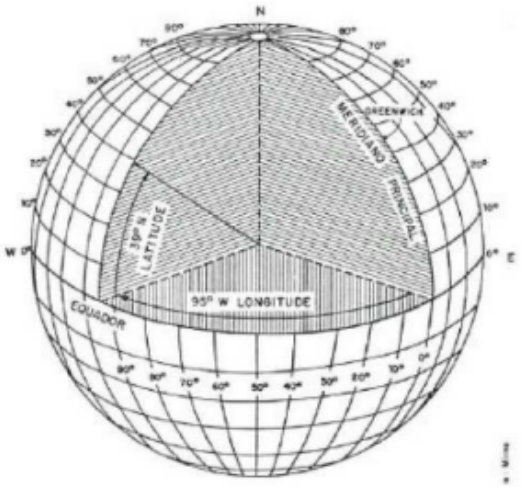
A adequada representação e manipulação de dados geoespaciais são essenciais para a eficácia dos Sistemas de Informação Geográfica, permitindo análises espaciais e decisões informadas em áreas como planejamento urbano e gerenciamento de recursos naturais.

Também é importante compreender os conceitos de latitude e longitude, uma vez que uma rota é composta por esse conjunto de pontos. Adicionalmente, a Figura 9 fornece um exemplo visual desses conceitos, considerando a Terra:

- **Latitude:** Representa a distância, medida em graus, minutos e segundos, ao Norte ou Sul em relação à linha do Equador, variando de 0 a 90°. Em outras palavras, a latitude é o ângulo entre a linha do Equador e o paralelo no qual o ponto está situado (GOMES, 2010).

- **Longitude:** É a distância, medida em graus, minutos e segundos, a Leste ou Oeste em relação ao Meridiano de Greenwich, variando de 0 a 180°. A longitude é o ângulo entre o Meridiano Principal (Greenwich) e o meridiano onde o ponto está localizado, considerando o centro da Terra (GOMES, 2010).

Figura 9 Exemplo de latitude e longitude



Fonte: GOMES, 2010

2.6 FrontEnd

O conceito de aplicativo, também conhecido como aplicação, refere-se a um software projetado para realizar uma função específica ou um conjunto de funções para o usuário. Segundo Sommerville (2011), *"um aplicativo é um programa de computador que fornece uma funcionalidade específica para o usuário final* (SOMMERVILLE, 2011).

No contexto do desenvolvimento móvel, o Android é uma das plataformas mais amplamente utilizadas. Desenvolvido pelo Google, o Android é um sistema operacional baseado em Linux projetado principalmente para dispositivos móveis, como smartphones e tablets. Ele oferece uma ampla gama de APIs e ferramentas para criar aplicativos ricos e interativos.

Para o desenvolvimento de aplicativos, diversas abordagens estão disponíveis, incluindo o uso do React Native, um framework criado pelo Facebook. O React Native permite a criação de aplicativos móveis para IOS e Android utilizando JavaScript e React, proporcionando uma experiência nativa ao usuário, enquanto aproveita a flexibilidade e a eficiência da web. De acordo com o Facebook (2020), *"React Native é um framework*

que permite o desenvolvimento de aplicativos móveis para iOS e Android usando a mesma base de código JavaScript" (Facebook, 2020).

Para aplicativos que necessitam de visualização de mapas, o OpenLayers se destaca como uma solução eficaz. Esta biblioteca JavaScript de código aberto facilita a visualização e a interação com dados geoespaciais na web. Conforme descrito na documentação oficial, *"OpenLayers é uma biblioteca que permite a construção de aplicações web de mapas com funcionalidades avançadas de interação e visualização"* (OPENLAYERS, 2023).

Além disso, para o gerenciamento de rotas, o uso integrado do Open Source Routing Machine (OSRM), uma ferramenta de roteamento de código aberto que utiliza algoritmos avançados para calcular rotas de maneira rápida e eficiente(OSRM, 2023).

O OSRM utiliza principalmente dois algoritmos para gerar rotas:

- **Algoritmo de Dijkstra:** Este algoritmo é utilizado para encontrar o caminho mais curto em um grafo ponderado. Conforme descrito por Montezi (2017), o Algoritmo de Dijkstra determina a menor rota a partir de um ponto de origem até todos os outros pontos em um grafo, considerando os pesos (custos) associados às arestas. É amplamente reconhecido por sua precisão e eficiência em roteamento (MONTEZI, 2017).

- **Algoritmo A*:** O A* é uma variação otimizada do algoritmo de Dijkstra que utiliza uma heurística para estimar a distância restante até o destino. Segundo Costa e Tonidandel (2018), o algoritmo A* combina a busca de menor custo com uma função heurística para estimar o custo total de chegar ao destino. Essa combinação permite ao A* focar a busca em áreas mais promissoras, acelerando o cálculo da rota e tornando-o mais eficiente, especialmente em grafos grandes e complexos (COSTA; TONIDANDEL, 2018).

3 Metodologia

O presente trabalho traz o desenvolvimento de uma aplicação para caronas que conectam motoristas e passageiros, permitindo que caronas sejam oferecidas apenas para amigos. Essa exigência de amizade visa garantir a segurança dos usuários, garantindo que as caronas sejam exclusivas para pessoas conhecidas e previamente cadastradas como amigos.

O desenvolvimento deste seguiu a abordagem detalhada abaixo:

1. **Revisão Bibliográfica:** Realizamos um estudo sobre as melhores práticas para o desenvolvimento de arquiteturas *RESTful*.
2. **Definição dos Tipos de Dados:** Investigamos os tipos de dados a serem utilizados no sistema e determinamos as melhores formas de obtê-los. Esse processo incluiu a identificação dos dados essenciais para a operação do aplicativo e a definição de como esses dados seriam adquiridos e manipulados.
3. **Modelagem de Dados:** Criamos o modelo de dados, que envolveu a definição das tabelas e a estruturação das relações entre elas.
4. **Criação da Base de Dados:** Implementamos o banco de dados, configurando as tabelas e estabelecendo os relacionamentos necessários. Adicionamos também extensões para lidar com *dados espaciais*, permitindo que o sistema gerenciasse e consultasse informações geográficas.
5. **Desenvolvimento da API:** Desenvolvemos a API, definindo todos os *endpoints* necessários para a operação do aplicativo. A implementação incluiu a definição de *endpoints RESTful* e a integração de medidas de segurança, como autenticação, para proteger os dados e as operações da API.
6. **Implantação da API:** A API foi implantada no *Heroku*, o que permitiu que fosse acessada de qualquer lugar. Esse passo foi essencial para garantir a disponibilidade da API.
7. **Desenvolvimento do Aplicativo:** Nesta fase, focamos na implementação do aplicativo em si. Geramos o *APK* para testes, permitindo a validação das funcionalidades do aplicativo.

Após a análise da arquitetura escolhida e a o entendimento dos dados a serem manipulados, iniciamos o desenvolvimento a fase de modelagem de dados. Nesta etapa,

definimos a estrutura das tabelas e como os dados serão armazenados. O diagrama entidade-relacionamento, apresentado na Figura 10, oferece uma visão abrangente e detalhada dos elementos envolvidos no sistema e das interações entre eles. Este diagrama é essencial para garantir a integridade e a organização dos dados, facilitando a implementação e o gerenciamento eficiente do banco de dados.

Neste diagrama, são representados os diferentes tipos de entidades envolvidas, como pessoa, rota, carona e amizade, e como elas se conectam umas às outras. Além disso, o diagrama define os atributos de cada entidade e as regras de relacionamento. Esses dados incluem informações essenciais para a operação do aplicativo, como as informações dos usuário e detalhes de caronas.

Na parte inferior da imagem, também é possível identificar as tabelas carregadas pela extensão POSTGIS, que possibilita o uso de funções para manipulação de dados espaciais.

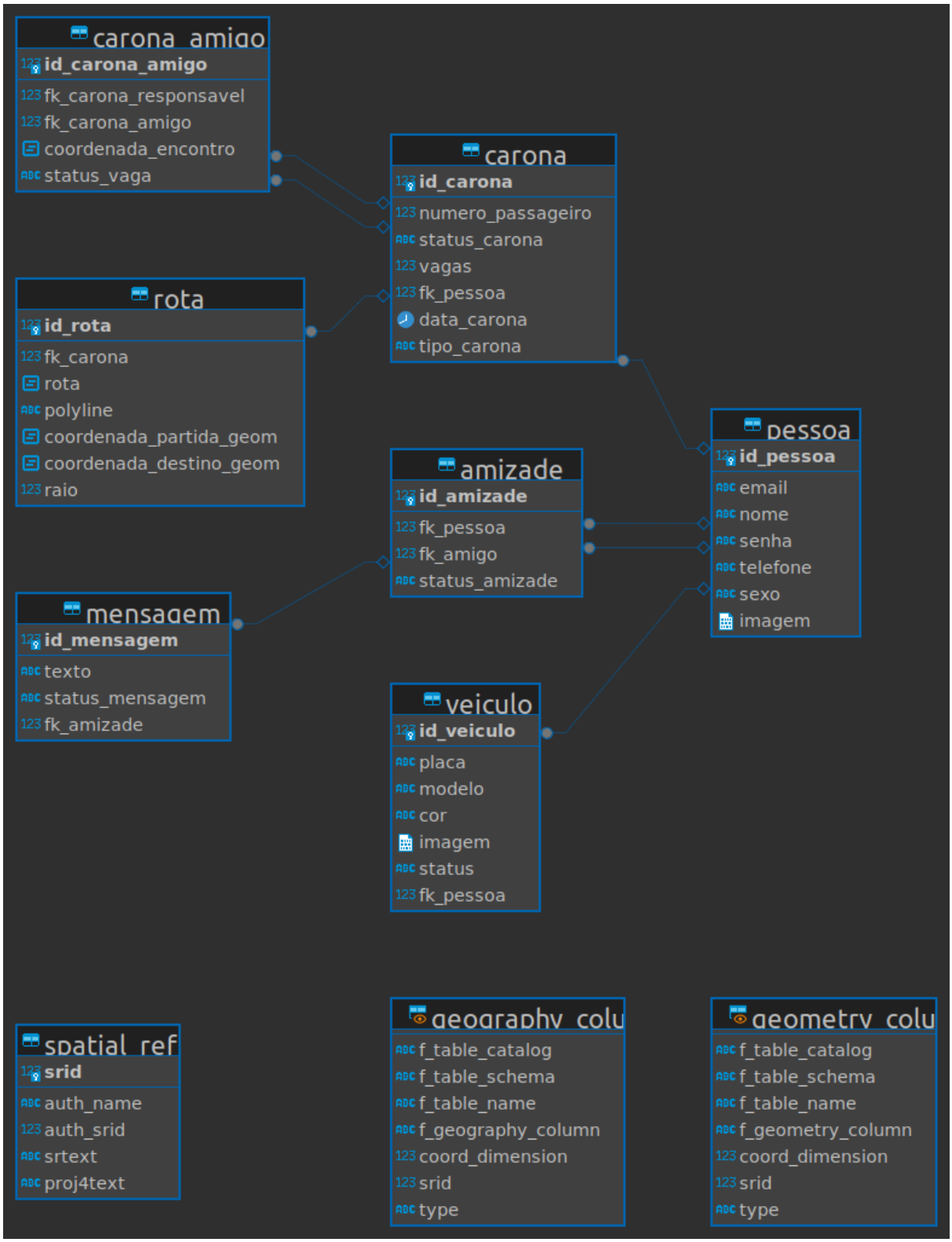
Na fase seguinte, focamos no desenvolvimento da API, onde definimos e implementamos todas as restrições e requisitos necessários para assegurar o funcionamento adequado do aplicativo. Especificamente, ao oferecer uma carona, ela será visível apenas para a rede de amigos previamente cadastrados, assegurando que apenas esses amigos possam ver e aceitar a carona. Além disso, para que uma solicitação de carona seja válida, tanto o ponto de partida quanto o destino do solicitante devem estar dentro de uma distância específica, medida em metros, em relação à rota do motorista. Essa distância pode ser definida pelo próprio motorista, ou, na ausência de uma definição, será assumida como 100 metros por padrão.

Para garantir a interseção das rotas, foi desenvolvida uma função no banco de dados que utiliza as funcionalidades do PostGIS, uma extensão espacial do PostgreSQL. Essa função, chamada `verifica_intersecao_rota`, verifica se a rota proposta por um usuário, com pontos de partida e destino, está próxima de uma rota existente, dentro de um raio específico. No Anexo 1, fornecemos o código SQL para a função.

A função converte as coordenadas geográficas para a projeção adequada (SRID 3857) para realizar os cálculos de distância. Em seguida, calcula a distância do ponto de partida e do destino em relação à rota existente. Se ambos os pontos estiverem dentro do raio especificado em metros e a ordem dos pontos ao longo da rota for lógica, ou seja, o ponto de partida e o destino seguem a mesma direção do trajeto do motorista, evitando uma interseção cruzada, a função considera a rota válida e retorna `true`; caso contrário, retorna `false`. A função do PostGIS usada para o cálculo da distância foi a `ST_DWithin`.

Após a implementação do *backend* e do banco de dados, publicamos a API no *Heroku*, uma plataforma de nuvem, para que a mesma ficasse disponível online. Em seguida, iniciamos o desenvolvimento do aplicativo. Como o aplicativo oferece serviços gratuitos,

Figura 10 Diagrama Entidade Relacionamento



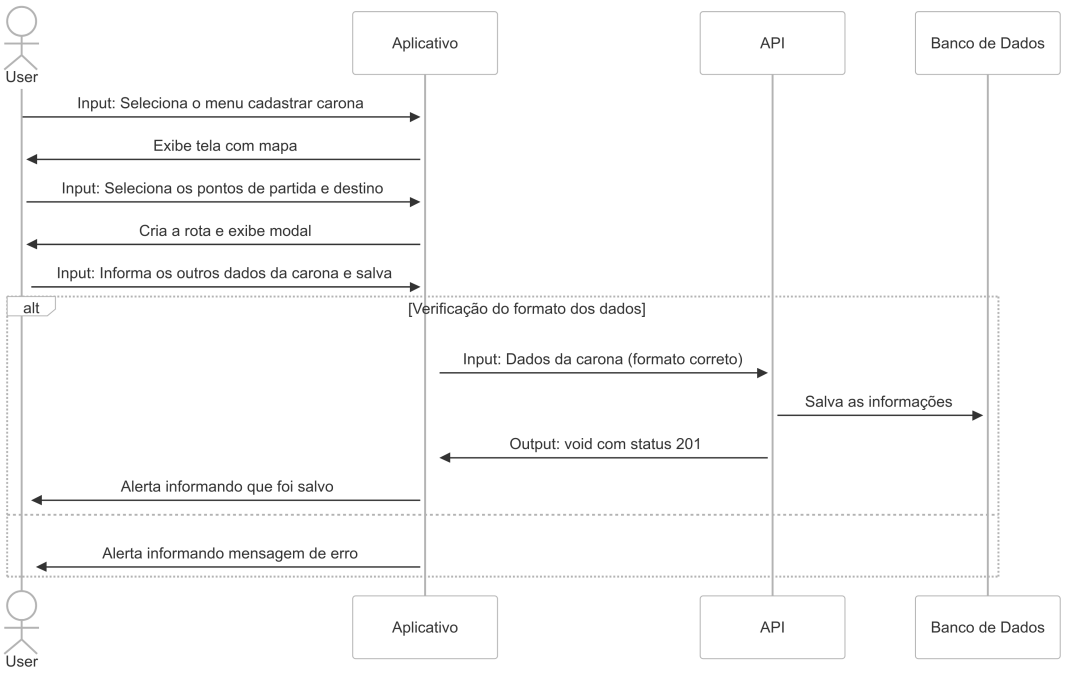
Fonte: Autor

todas as bibliotecas utilizadas precisaram ser de código aberto. Para a geração dos mapas, utilizamos a biblioteca *OpenLayers*, voltada para aplicações web, e para os serviços de roteamento, empregamos o *OSRM*, responsável por criar rotas entre dois pontos. Para incorporar essas funcionalidades no aplicativo, utilizamos a biblioteca `WebView` do *React Native*, que permite renderizar páginas web dentro do aplicativo, viabilizando assim o uso de serviços web diretamente na aplicação móvel.

Além disso, a seguir são apresentados os diagramas de sequência que ilustram as principais funcionalidades do sistema. A Figura 11 exibe o diagrama de sequência para o cadastro de caronas, destacando a interação do usuário com o aplicativo durante essa ação. O usuário deve fornecer os dados necessários para criar uma carona, como os pontos de partida e destino, a data e o número de vagas. Após preencher essas informações, o sistema as envia para a API, que as encaminha para o banco de dados. Já a Figura 12 detalha a funcionalidade de busca de caronas, evidenciando a principal diferença em relação ao cadastro de caronas: a ausência de uma rota específica. Este diagrama foca na correspondência entre o ponto de partida e o destino, além de considerar outros dados fornecidos pelo usuário no momento do cadastro.

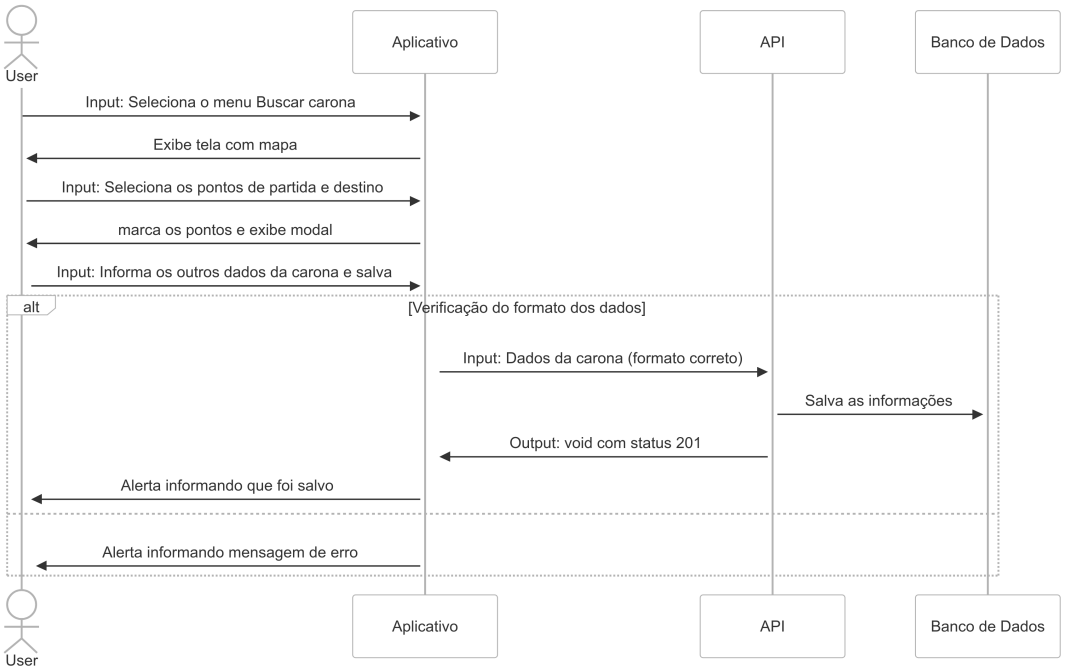
A Figura 13 apresenta o diagrama de sequência da página "Minhas Caronas", que classifica as caronas de acordo com seus status: ATIVA, EM ANDAMENTO, CANCELADA e CONCLUÍDA, ilustrando todo o fluxo da tela "Minhas Caronas". O diagrama foca especialmente nas caronas com o status ATIVA. A visualização dessas caronas varia conforme o papel do usuário. Para os usuários que estão buscando uma carona, a tela exibe um card com as informações da carona e oferece a opção "Ver Interseção", que lista todos os amigos que têm interseção com os pontos de partida e destino da carona. Em contraste, para os usuários que estão oferecendo uma carona, o diagrama mostra um botão "Ver Detalhes", que permite visualizar a lista de amigos que já aceitaram a carona, além de exibir as pessoas que estão participando.

Figura 11 Diagrama de Sequência: Cadastrar Carona



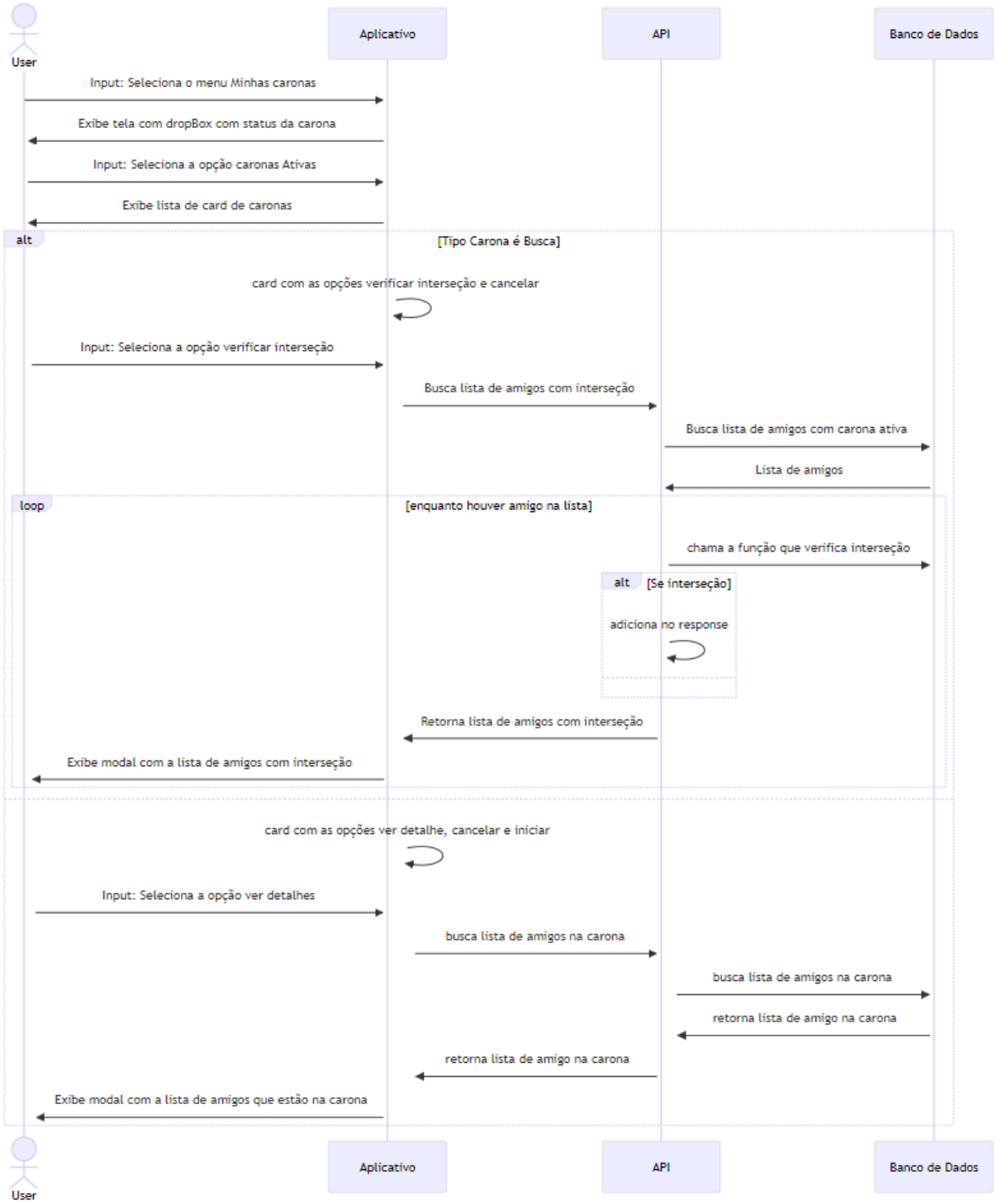
Fonte: Autor

Figura 12 Diagrama de Sequência: Buscar Carona



Fonte: Autor

Figura 13 Diagrama de Sequência: Página Minhas Caronas

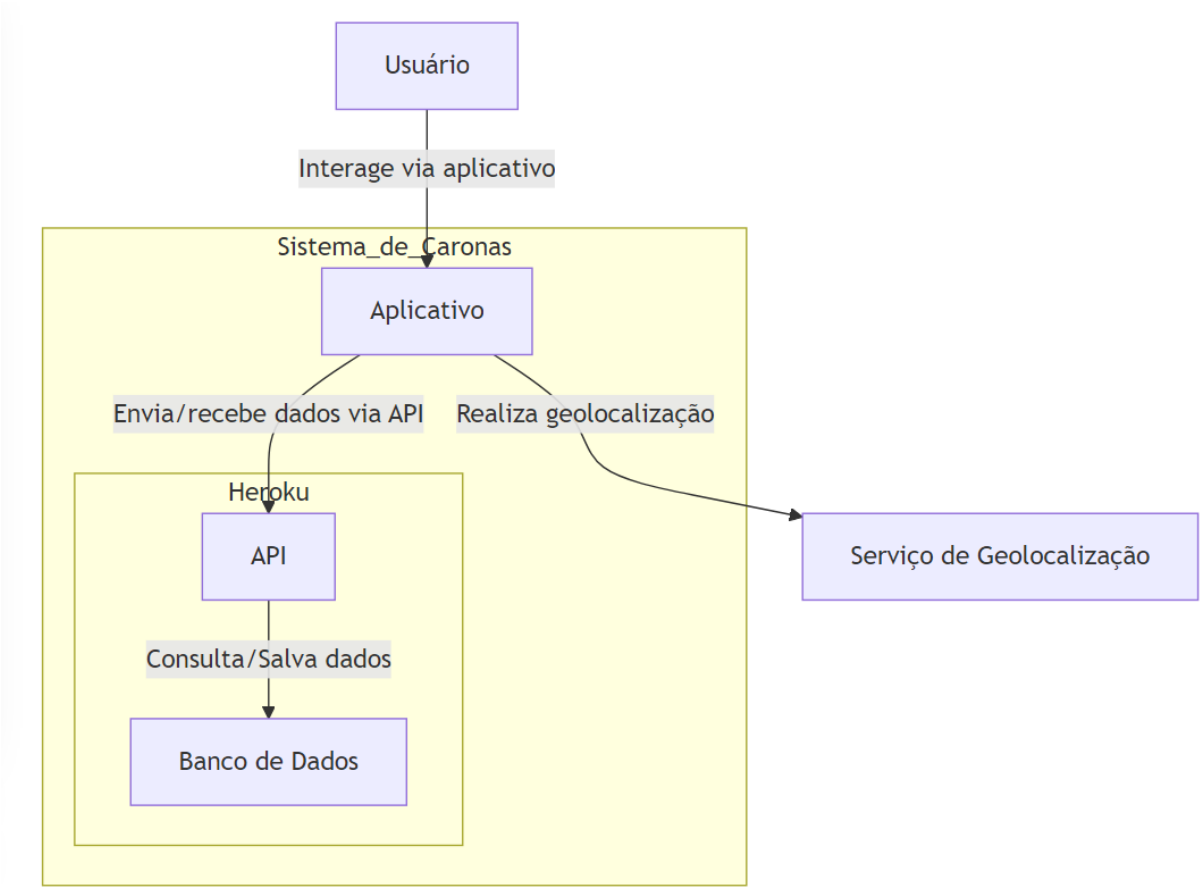


Fonte: Autor

Também destacamos o diagrama de contexto, apresentado na Figura 14, que ilustra claramente todas as partes envolvidas no sistema de caronas, incluindo os usuários. Este diagrama oferece uma visão geral das interações entre os diferentes componentes do sistema, demonstrando como os usuários, o aplicativo, os serviços de geolocalização, a API e o

banco de dados se conectam e trocam informações, garantindo a operação integrada do sistema.

Figura 14 Diagrama de contexto



Fonte: Autor

Após o detalhamento, na seção seguinte abordaremos os resultados, focando nos testes e validações realizados no aplicativo.

4 Resultados

Neste capítulo, é abordado o desenvolvimento do aplicativo de carona, começando pela análise da API que sustenta o sistema e prosseguindo para o desenvolvimento do aplicativo que consome os endpoints dessa API.

A API foi projetada utilizando a arquitetura *REST*, o que garante uma comunicação eficiente e escalável entre os diversos componentes do sistema. Com a implementação do modelo de autenticação *OAuth 2*, foi possível fortalecer a segurança, assegurando que as rotas sejam protegidas e as senhas sejam criptografadas, o que contribui para a integridade e confidencialidade dos dados.

O desenvolvimento do aplicativo de carona se beneficia diretamente da API, que fornece os endpoints necessários para a interação com os dados do sistema. Cada funcionalidade do aplicativo, desde a criação e gerenciamento de caronas até a autenticação dos usuários, é realizada por meio das operações disponibilizadas pela API.

Para obter uma visão detalhada dos endpoints e das operações da API, consulte a documentação interativa fornecida pelo *Swagger*, ilustrada na Figura 15. A documentação está acessível através do seguinte link: <https://carona-friend-ce3958279de8.herokuapp.com/swagger-ui/index.html#/>.

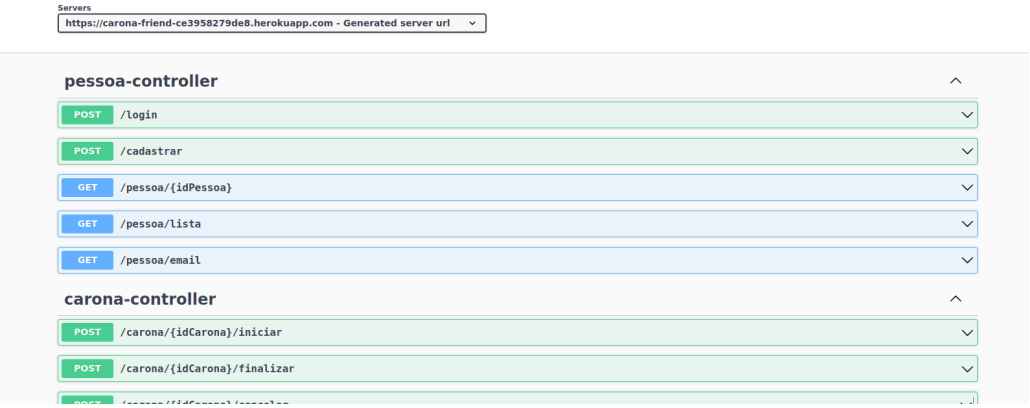


Figura 15 Swagger - Api Carona Friend

Fonte: Autor

Após a API estar online, sua integração foi essencial para o desenvolvimento do aplicativo de carona. Com a API disponível, foi possível construir e testar as telas e funcionalidades do aplicativo, que utilizam os endpoints da API para operações como criação e gerenciamento de caronas, e autenticação de usuários. Cada funcionalidade do aplicativo foi implementada para refletir os recursos oferecidos pela API, garantindo que a interface do usuário e as operações de backend estejam alinhadas. A interação

contínua com a API permitiu ajustes e otimizações no aplicativo, assegurando que todas as funcionalidades operem conforme o esperado e atendam aos requisitos estabelecidos.

Para validar a funcionalidade de oferta de carona restrita a amigos com rotas intersectantes, foram elaborados três casos de teste distintos. Esses testes visam verificar se as ofertas de carona são exibidas corretamente com base na interseção das rotas e no status de amizade entre os usuários, garantindo que as regras de visibilidade e segurança da aplicação sejam adequadamente aplicadas:

1. **A é amigo de B**: Neste cenário, o usuário A oferece uma carona cuja rota tem interseção com os pontos do usuário B, que busca carona. Como A e B são amigos, a carona será exibida para B, permitindo que ele visualize e aceite a oferta.

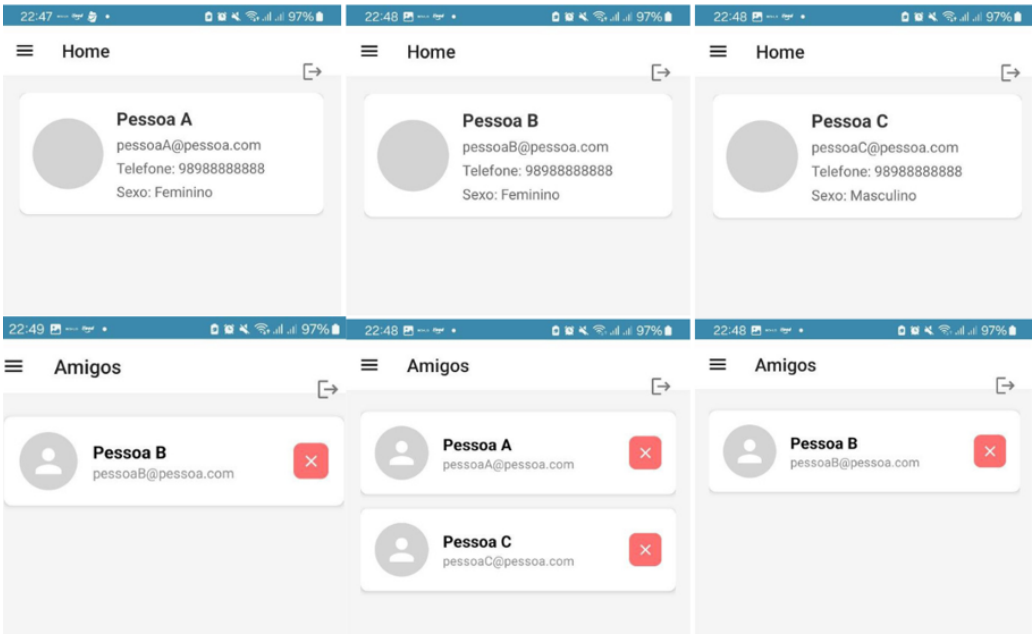
2. **C é amigo de B**: Aqui, o usuário C oferece uma carona, mas sua rota não tem interseção com a rota de B. Mesmo sendo amigos, a carona não aparecerá para B, pois não há correspondência entre as rotas de ambos.

3. **A não é amigo de C**: Neste caso, A oferece uma carona e C realiza uma busca, ambos com rotas que se intersectam. No entanto, como A e C não são amigos, a carona não será exibida para C, mesmo havendo uma interseção de rotas, cumprindo a regra de segurança do sistema de que apenas amigos podem visualizar as caronas oferecidas.

Primeiramente, apresentaremos a lista de amizades dos três usuários envolvidos no processo de carona. Na Figura 16, é possível observar, na parte superior, as informações detalhadas dos usuários A, B e C. Para cada usuário, a lista exibe seus detalhes de cadastro, incluindo informações como nome e e-mail. Abaixo dessas informações, são listados os amigos de cada usuário.

A análise da lista revela que o usuário A tem apenas um amigo, que é o usuário B. Da mesma forma, o usuário C também é amigo somente de B. Assim, o usuário B mantém amizade tanto com A quanto com C. Esta configuração é fundamental para validar os cenários de teste, uma vez que as interações entre esses usuários são essenciais para verificar a funcionalidade da oferta de carona restrita a amigos com rotas intersectantes.

Figura 16 Lista de amizades de cada usuário envolvido no processo de carona.



Fonte: Autor

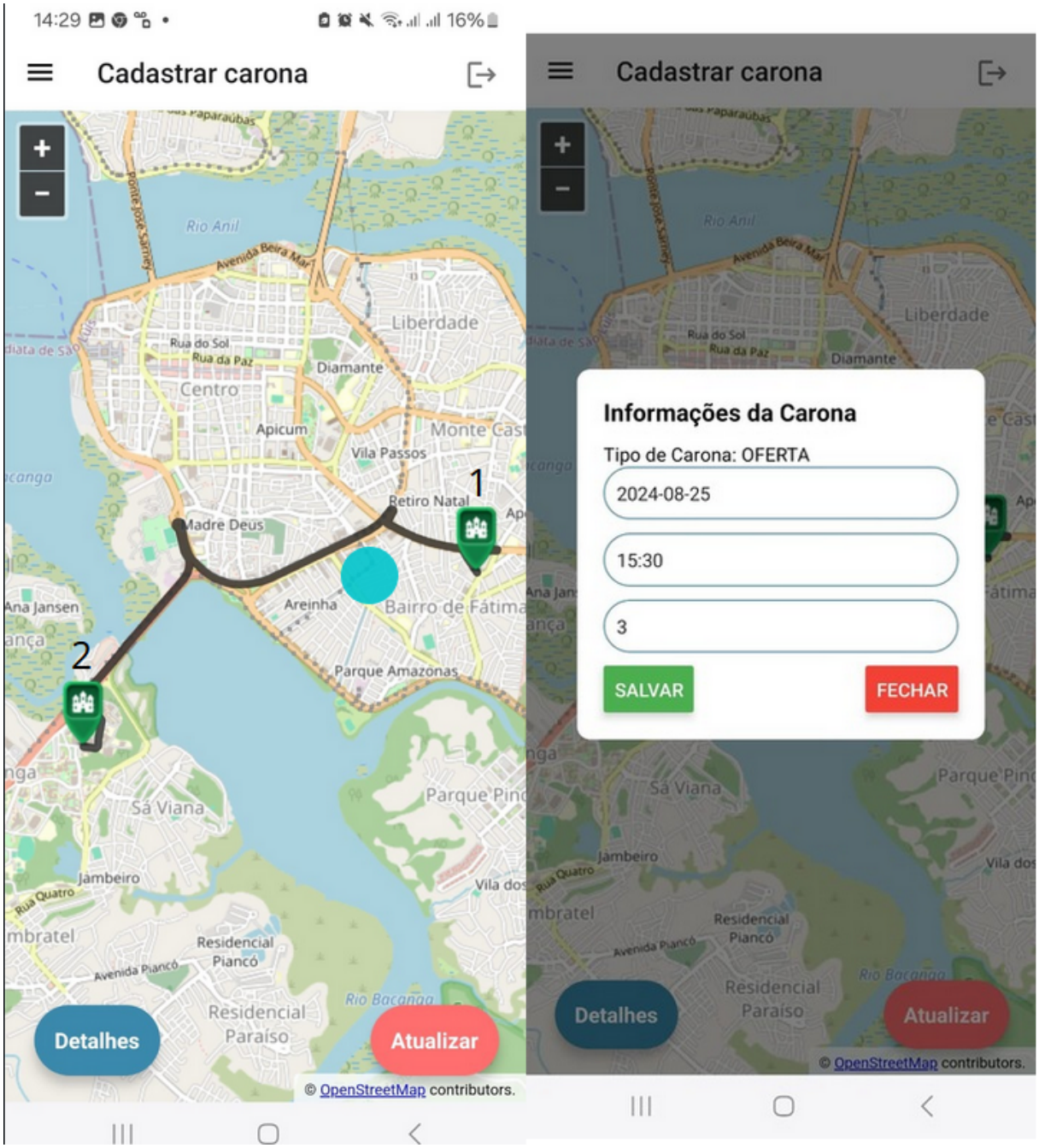
A seguir, detalham-se os casos de teste:

Caso de teste 1 - A é amigo de B

Este teste verifica se a funcionalidade de oferta de carona opera conforme o esperado quando A e B são amigos. O objetivo é confirmar que, quando o usuário A oferece uma carona cuja rota intercepta ou está próxima dos pontos de partida e destino do usuário B, a oferta de carona seja visível para B. Assim, a carona ofertada por A deve ser corretamente exibida para B, permitindo que ele visualize e aceite ou recuse a oferta.

Na Figura 17, estão apresentados os detalhes da carona oferecida por A. No lado esquerdo da imagem, observa-se um mapa com a rota gerada. Ao selecionar o ponto de partida (ponto 1 na imagem), marcado com um ícone verde, e depois o ponto de destino (ponto 2), a rota é automaticamente traçada com uma linha preta. Após o traçado da rota, um modal é exibido, mostrado no lado direito da Figura, permitindo que o usuário insira informações adicionais, como a data, a hora, e o número de vagas disponíveis. Estas informações são apresentadas nos campos do modal quando não estão preenchidas. O usuário pode fechar o modal para revisar os pontos selecionados; se clicar novamente no mapa, a rota será removida, permitindo a definição de novos pontos. Os dados da carona também desaparecerão se o usuário clicar no botão de atualizar. Caso o modal não apareça automaticamente após marcar todos os pontos, o usuário pode reabri-lo clicando no botão azul "Detalhes". Após preencher todos os campos necessários, o usuário deve clicar no botão "Salvar"no modal para concluir o processo.

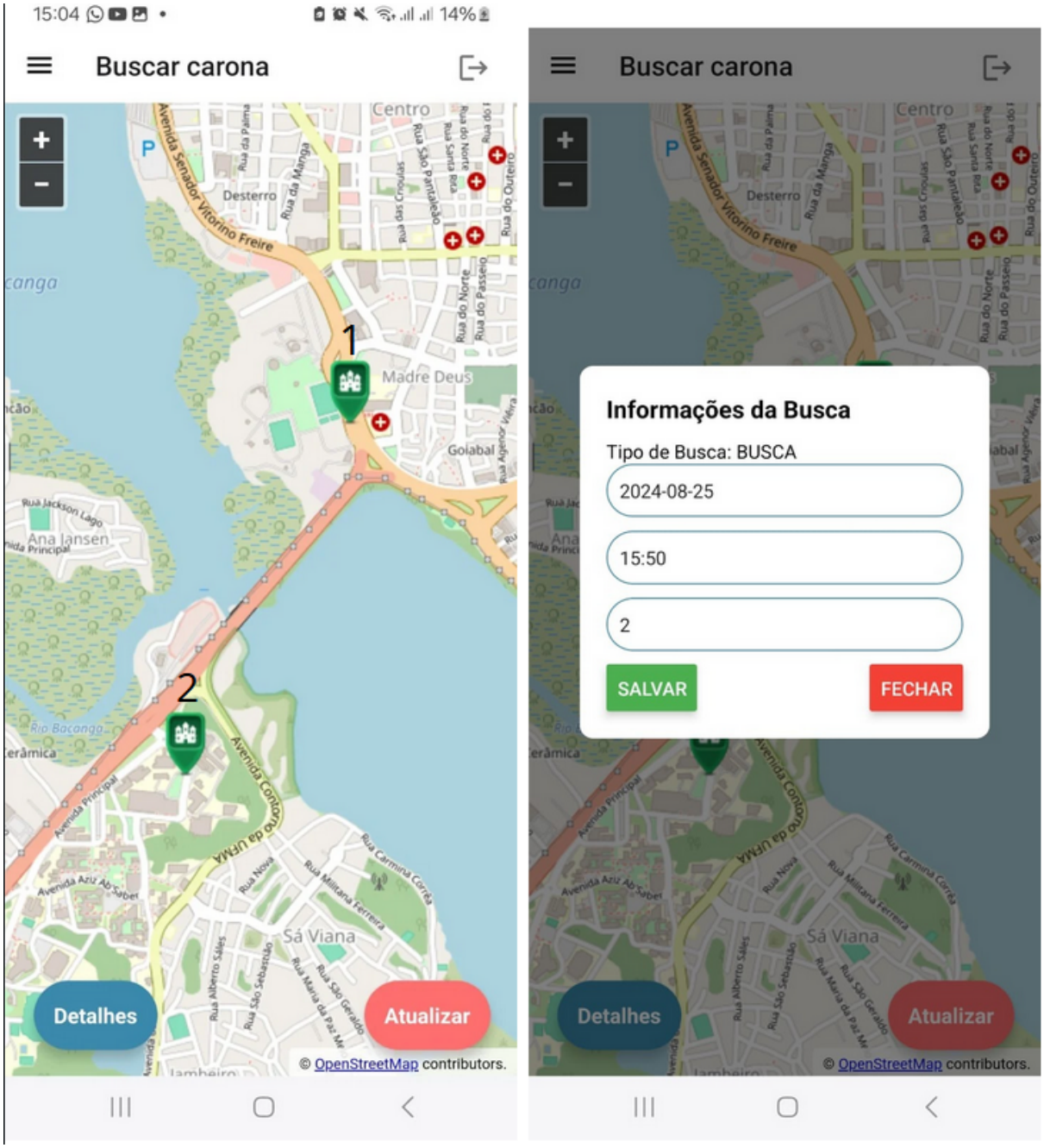
Figura 17 Dados da Carona ofertada por A.



Fonte: Autor

Na Figura 18, são apresentados os detalhes da carona oferecida por B. O ponto 1 indica o ponto de partida e o ponto 2 o destino. Ao contrário da Figura 17, que mostra uma rota traçada, na busca de carona de B, apenas os pontos de partida e destino, marcados pelo ícone verde, são destacados. As demais funcionalidades disponíveis na busca de carona são semelhantes às da Figura 17, com a principal diferença sendo a ausência da rota.

Figura 18 Dados da Busca por carona de B.



Fonte: Autor

Após o preenchimento dos dados da carona, tanto A quanto B podem acompanhar as informações de suas caronas na aba "Minhas Caronas". Nesta tela, todas as caronas são exibidas, incluindo tanto as ofertas quanto as buscas, organizadas por status.

Para as caronas do tipo "Oferta", os cards exibem informações como vagas disponíveis, vagas totais, e a data da carona. Além disso, são apresentados botões para iniciar ou cancelar a carona. O link "Ver Detalhes"permite visualizar todos os amigos que

possuem interseção com a rota e aceitaram a oferta, como mostrado na Figura 19, onde A está oferecendo uma carona.

Já para as caronas do tipo "Busca", o card apresenta informações semelhantes, mas sem o botão "Iniciar"e com o link "Ver Detalhes"substituído por "Verificar Interseção". Ao clicar em "Verificar Interseção", abre-se um modal que lista todos os amigos com interseções e exibe detalhes como nome, telefone para contato, horário e número de vagas. Também há um botão "Aceitar Carona", que, ao ser clicado, move a carona para o status "Em Andamento", indicando que a busca foi bem-sucedida e que apenas ajustes de ponto de encontro são necessários. A Figura 20 ilustra um exemplo da carona oferecida por B.

Figura 19 Detalhes da carona de A.

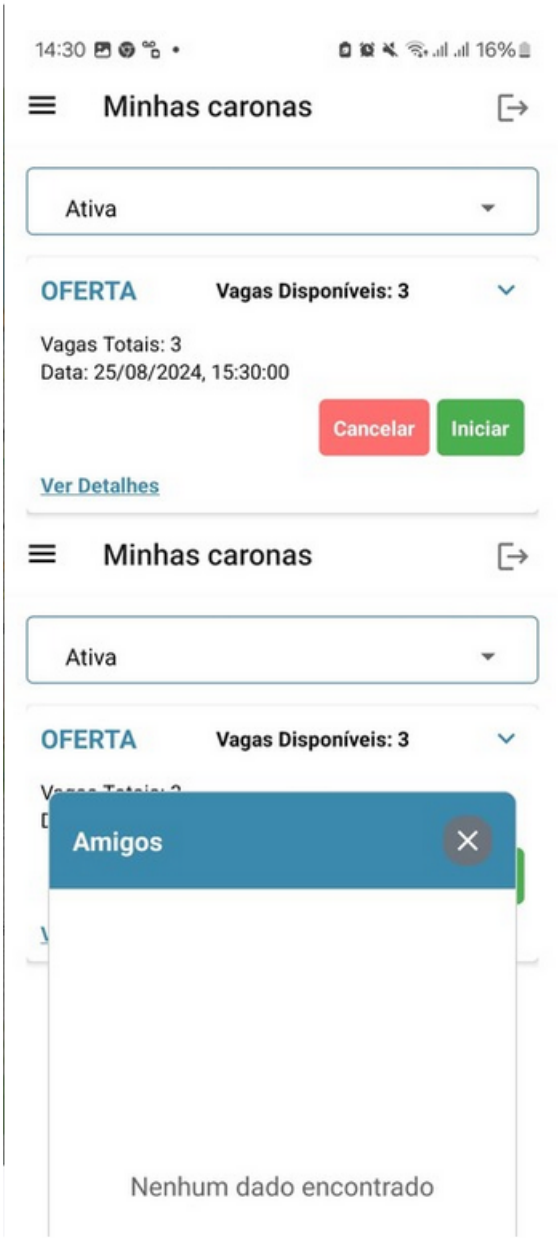
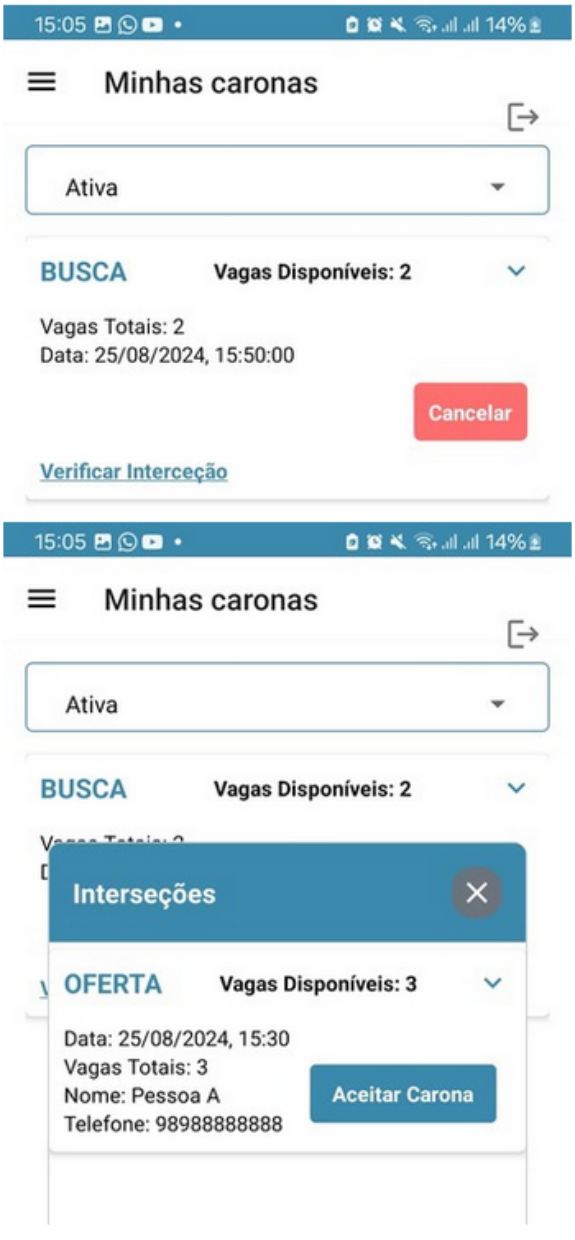


Figura 20 Detalhes da carona de B.

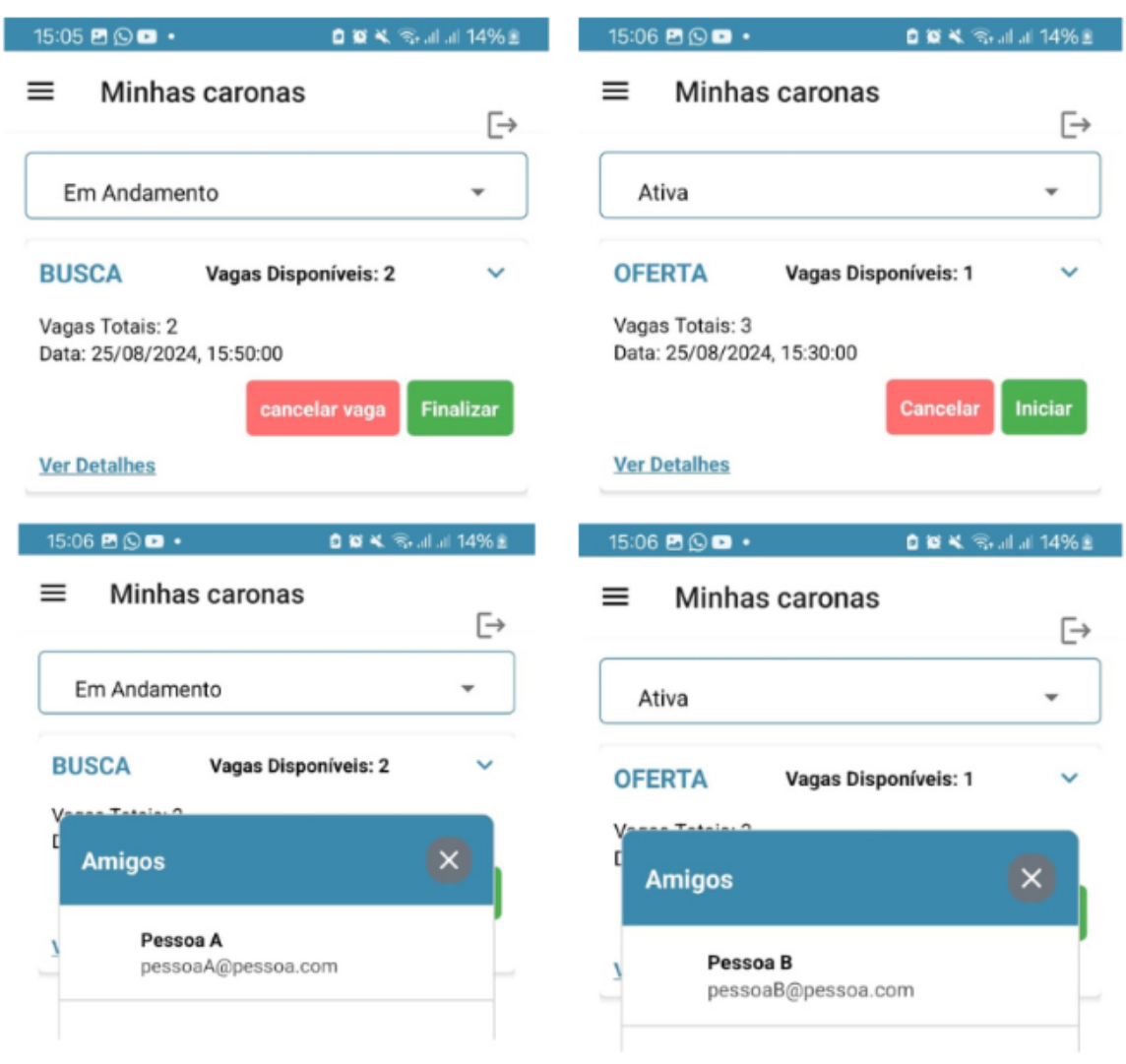


Fonte: Autor

Quando o usuário que busca uma carona encontra uma interseção e a aceita, o status de sua carona muda para "Em Andamento". Nesse estágio, quem solicitou a carona pode acompanhar os detalhes da pessoa que a está oferecendo. Da mesma forma, quem está ofertando a carona pode monitorar os amigos que já a aceitaram, como mencionado anteriormente.

Na Figura 21, temos exatamente essa ilustração. Na parte direita da imagem, é possível ver quem está oferecendo a carona para B, enquanto na parte esquerda, podemos visualizar a pessoa que já está na carona ofertada por A. Além disso, as vagas disponíveis na carona de A diminuem de 3 para 1, já que B ocupará duas vagas.

Figura 21 Dados da carona aceita por B e informações, para A, sobre os amigos que estão em sua carona. Como A e B são amigos, a carona foi exibida para B e aceita por ele, mostrando os dados de quem irá participar da carona. Para A, são exibidos os amigos que aceitaram sua carona.



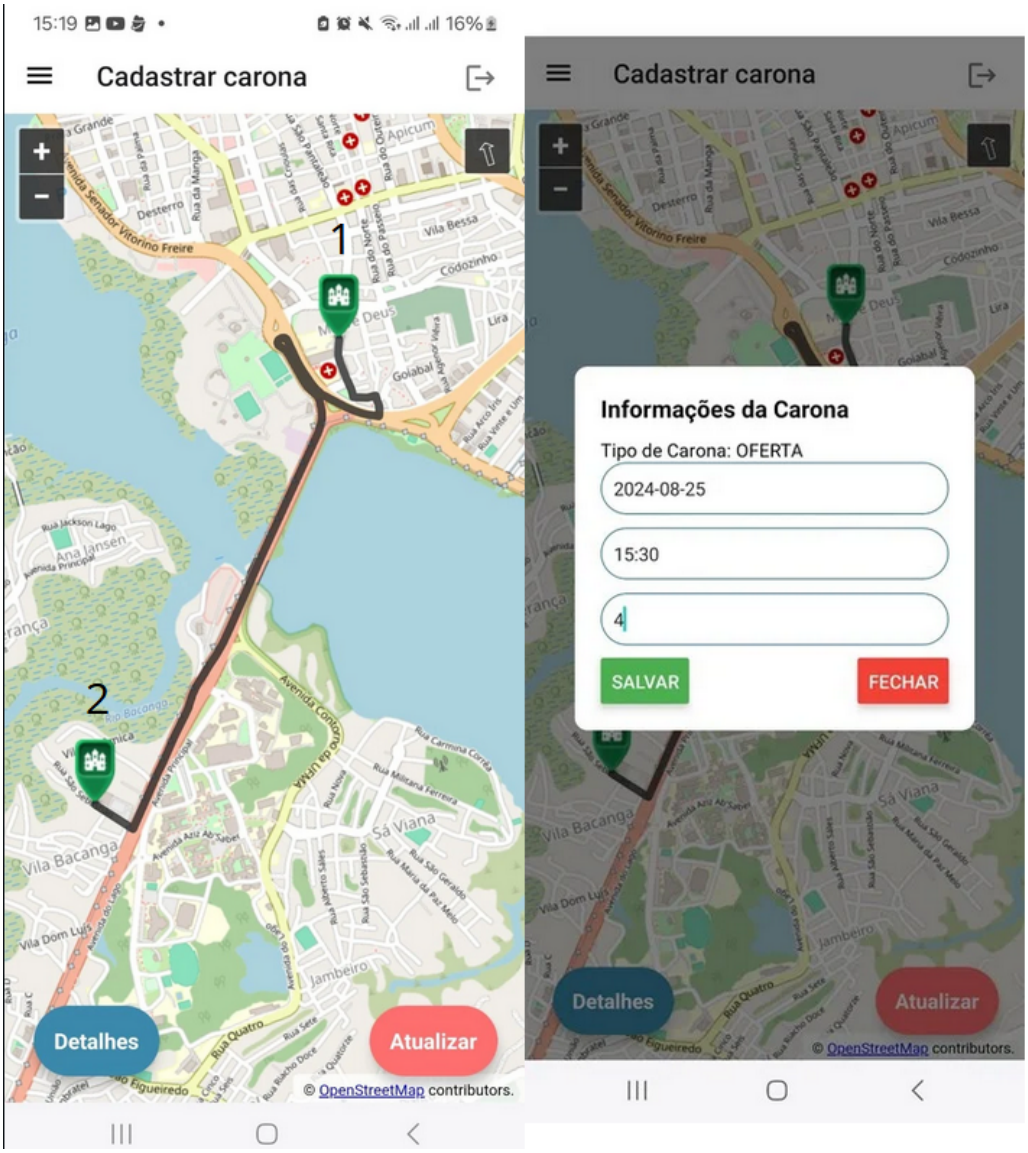
Fonte: Autor

Caso de teste 2 - C é amigo de B

Este teste verifica se a funcionalidade de oferta de carona opera conforme o esperado quando C e B são amigos, mas as rotas de suas caronas seguem direções opostas. O objetivo é garantir que, mesmo sendo amigos, C e B não terão uma interseção de rotas quando os pontos de partida ou destino não coincidem.

Na Figura 22, estão apresentados os detalhes da carona ofertada por C. O ponto 1 indica o local de partida e o ponto 2 o destino. Embora C e B sejam amigos, como suas rotas seguem direções opostas, não haverá uma interseção completa entre elas, e portanto, B não verá a oferta de carona de C. As funcionalidades da tela mostrada na figura, como o traçado da rota e a inserção de detalhes da carona, já foram explicadas nos casos anteriores.

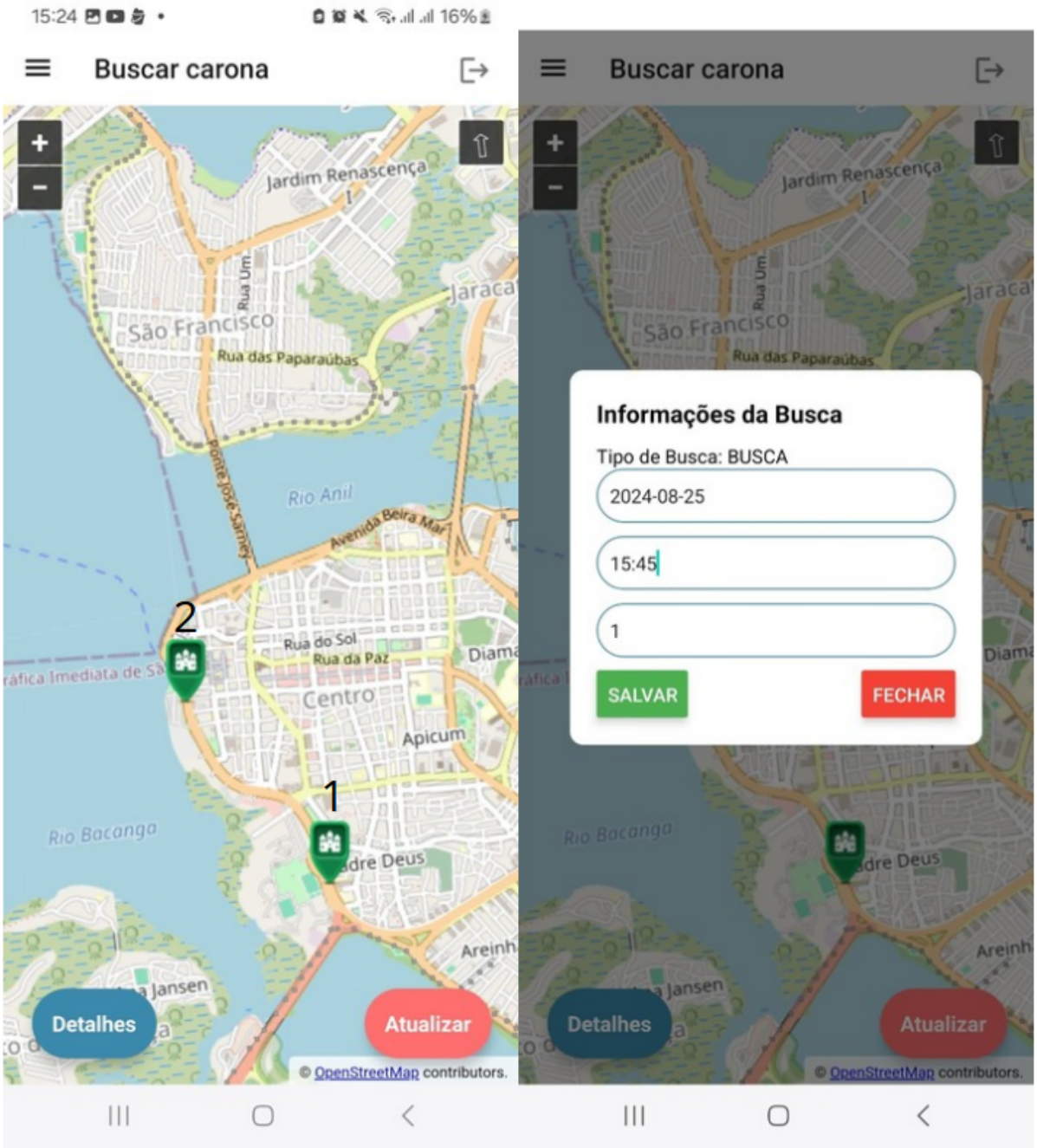
Figura 22 Dados da carona ofertada por C.



Fonte: Autor

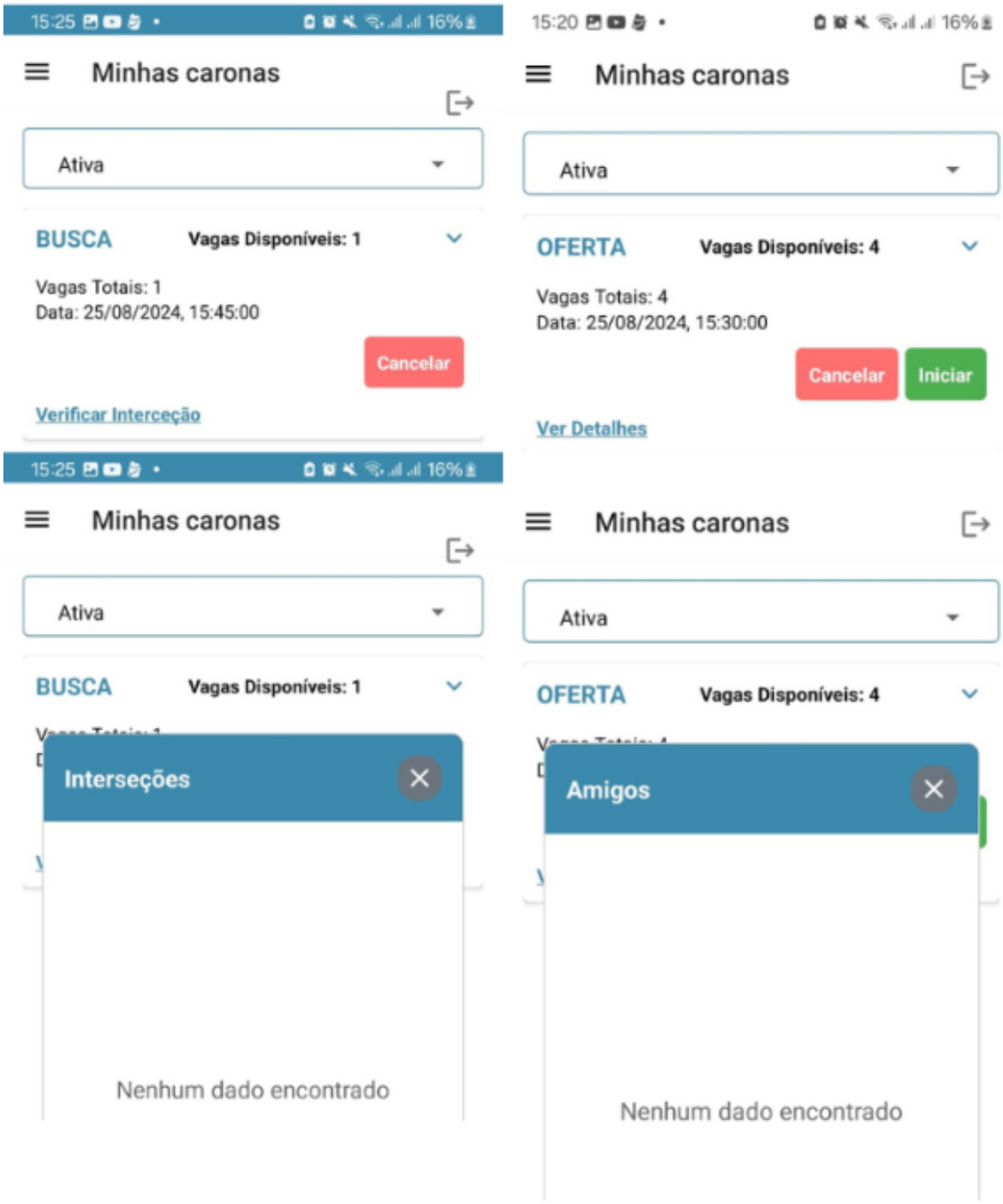
A Figura 23 apresenta os detalhes da busca por uma nova carona realizada por B, onde o ponto 1 indica o local de partida e o ponto 2 o destino. Nota-se que, apesar de a rota de C cruzar o ponto de partida de B, os destinos são opostos, impossibilitando uma interseção completa entre as caronas. Isso é ilustrado na Figura 24, onde, no lado esquerdo da imagem, B não possui nenhuma interseção de rota, enquanto no lado direito, C não tem nenhum amigo presente na rota, validando assim o Caso 2.

Figura 23 Dados da busca por carona de B.



Fonte: Autor

Figura 24 Dados da carona solicitada por B, que não possui interseção com nenhuma outra rota, e informações para C, indicando que nenhum amigo aceitou sua carona.



Fonte: Autor

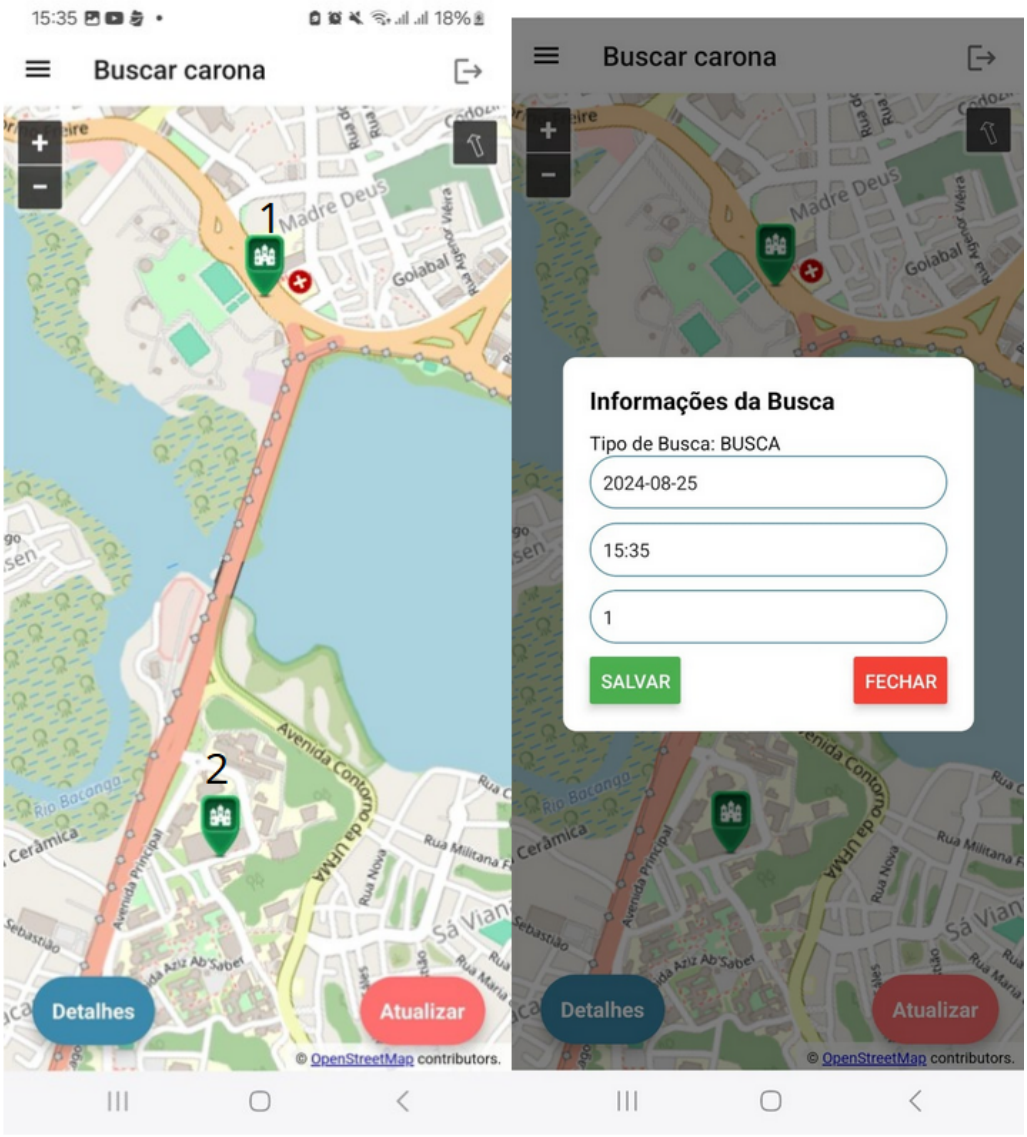
Caso de teste 3 - A não é amigo de C

Este teste verifica se a funcionalidade de oferta de carona opera conforme o esperado quando A e C não são amigos. O objetivo é confirmar que, mesmo que a rota de uma

carona oferecida por A intercepte ou esteja próxima dos pontos de partida e destino de C, essa oferta de carona não deve ser visível para C. Dessa forma, a carona ofertada por A não deve ser exibida para C, garantindo que a visibilidade de caronas seja limitada a amigos.

Na Figura 17, já detalhada no Caso de Teste 1, é apresentada a oferta de carona de A. A Figura 25 mostra C como o usuário que busca carona. Embora os pontos de partida e destino de C (representados pelos pontos 1 e 2, respectivamente) sejam os mesmos que os de B, o que, em teoria, cria uma interseção com a rota de A, a carona não é apresentada como uma opção para C, uma vez que A e C não são amigos.

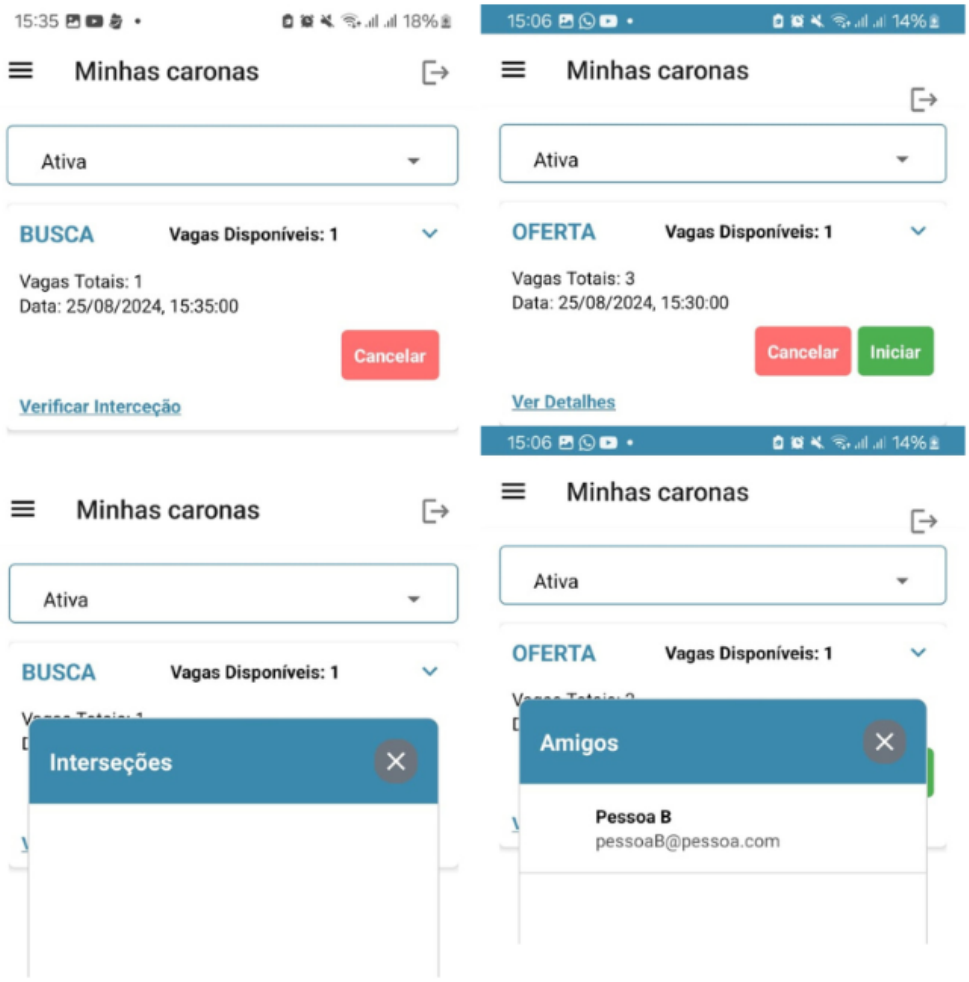
Figura 25 Dados da Busca por carona de C.



Fonte: Autor

Essa ausência de interseção visível é ilustrada na Figura 26. Mesmo que exista uma sobreposição nas rotas, a carona de A não é mostrada a C, validando assim o Caso de Teste 3, que assegura que apenas amigos podem visualizar ofertas de carona que compartilham rotas.

Figura 26 Dados da carona solicitada por C, que possui interseção com a carona de A, mas, por não serem amigos, não aparece na lista de interseções. Informações para A indicam que somente B irá participar de sua carona.



Fonte: Autor

A execução deste caso de teste concluiu o processo de implementação do aplicativo, validando com êxito a lógica de interseção de rotas e as restrições baseadas em amizade. Dessa forma, asseguramos que o sistema opera conforme o esperado, oferecendo caronas exclusivamente para amigos cujas rotas coincidam, o que reforça tanto a segurança quanto a confiabilidade da aplicação.

5 Conclusão

O projeto atingiu com sucesso seu objetivo principal: desenvolver um sistema seguro para cruzamento de rotas, ofertando caronas exclusivamente a pessoas conhecidas. Para isso, foi necessário um amplo conhecimento em áreas como SIG, banco de dados, e linguagens de programação, incluindo JavaScript, HTML, CSS, JAVA, SQL, Além de frameworks como Spring Boot e React Native. Esses recursos foram combinados para criar uma aplicação eficiente, voltada para a redução do transporte unitário — uma prática comum em que apenas uma pessoa utiliza o carro.

A aplicação busca contribuir para a solução de problemas de mobilidade urbana, reduzindo engarrafamentos e diminuindo a poluição do ar. Além disso, o sistema potencialmente melhora a segurança pública, uma vez que os usuários podem identificar previamente quem precisará de carona, diminuindo a exposição a riscos como assaltos em paradas de ônibus.

O aplicativo também otimiza a prática cotidiana de oferecer caronas entre amigos, evitando manobras perigosas como troca de faixa ou paradas bruscas. Com a programação antecipada das caronas, o risco de acidentes de trânsito é reduzido, aumentando a segurança tanto para motoristas quanto para passageiros.

A proposta do aplicativo visa, ainda, realçar a eficiência das caronas, permitindo que passageiros e motoristas alternem seus papéis conforme a necessidade. Isso contribui para a diminuição não apenas do transporte unitário, mas também do número total de carros em circulação, aliviando o trânsito e beneficiando a todos, sejam usuários do aplicativo ou não.

Como trabalhos futuros, planeja-se o aperfeiçoamento da aplicação com a adição de novas funcionalidades, como a janelas de conversas entre motoristas e passageiros. Essas funções ajustar detalhes da carona, além de oferecer rotas alternativas para reduzir tempo e custo de viagem. O produto final será um aplicativo gratuito, com a implementação em Java e utilizando a tecnologia do React Native para criação do mesmo.

Referências

ASSUNÇÃO, L. d. F. d. O. *Sistema de carona usando rede social e coordenadas geográficas*. Dissertação (Mestrado) — Universidade Federal do Maranhão. Centro de Ciências Exatas e Tecnologias, São Luís, 2019. Bacharelado Interdisciplinar em Ciência e Tecnologia. Citado na página 14.

BlaBlaCar. *Sobre nós*. 2024. Acesso em 20 de agosto de 2024. Disponível em: <https://blog.blablacar.com.br/about-us?_gl=1*qelh4n*_gcl_au*MTU4MjU0MjA4Ni4xNzI0MjAxNTg4>. Citado na página 12.

BROWN, S. O modelo c4 de documentação para arquitetura de software. agosto 2018. Acesso em 26 ago 2024. Disponível em: <https://www.infoq.com/br/articles/C4-architecture-model/>. Citado na página 18.

CABIFY. *What moves us*. 2024. Acesso em 20 de agosto de 2024. Disponível em: <https://cabify.com/en/about-us>. Citado na página 12.

CASTRO, H. A. d.; GOUVEIA, N.; ESCAMILLA-CEJUDO, J. A. Questões metodológicas para a investigação dos efeitos da poluição do ar na saúde. *Revista Brasileira de Epidemiologia*, SciELO Public Health, v. 6, p. 135–149, 2003. Citado na página 10.

CODD, E. F. A relational model of data for large shared data banks. *Communications of the ACM*, v. 13, n. 6, p. 377–387, 1970. Citado na página 19.

Confederação Nacional de Municípios. *Análise do Impacto da Frota de Veículos nos Municípios Brasileiros*. 2023. Acesso em: 20 ago. 2024. Disponível em: <https://cnm.org.br/storage/biblioteca/2023/Estudos_tecnicos/202309_ET_MOB_Impacto_frota_veiculos.pdf>. Citado na página 10.

COSTA, L. d. S.; TONIDANDEL, F. Análise de algoritmos de path-planning. Centro Universitário FEI, 2018. Citado na página 23.

COSTA, S. S. *Banco de Dados Geográficos*. São Luís: Instituto Federal de Educação, Ciência e Tecnologia do Maranhão (IFMA), 2021. Citado na página 20.

DIAS, F. F. et al. Exposure to pollution during pregnancy and occurrence of miscarriage. *Ciência Saúde Coletiva*, v. 27, n. 6, p. 2087–2096, 2022. Acesso em: 20 ago. 2024. Disponível em: <https://www.scielo.br/j/cflo/a/CCWQsx3MZcTDpBSMJD4pq5S/?lang=pt>. Citado na página 10.

Facebook. *React Native*. 2020. Acesso em: Agosto 2024. Disponível em: <https://reactnative.dev>. Citado na página 23.

FIELDING, R. T.; RESCHKE, J. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. [S.l.], 2014. Citado 2 vezes nas páginas 17 e 18.

GADELHA, M. *Aplicativo de caronas solidárias da UFRN registra mil downloads em uma semana*. 2017. Acesso em 20 de agosto de 2024. Disponível em: <https://www.ufrn.br/imprensa/materias-especiais/2872/

aplicativo-de-caronas-solidarias-da-ufrn-registra-mil-downloads-em-uma-semana>. Citado na página 14.

GOMES, T. S. *Fundamentos de GPS: Conceitos, Operação e Configuração*. [S.l.]: Brasília (DF), 2010. Não publicado. Citado 2 vezes nas páginas 21 e 22.

GRISA, F.; DOMINGUES, F. O trânsito no brasil na última década: dados para subsídio da educação para o trânsito. *Revista Escola DetranRS*, v. 3, n. 1, p. 22–37, 2023. Citado na página 10.

IBGE. *Frota de veiculos*. 2023. Acesso em: 26 set. 2024. Disponível em: <https://cidades.ibge.gov.br/brasil/pesquisa/22/28120>. Citado na página 10.

JÚNIOR, S. E. P. R. *Verificação de Conformidade entre Diagramas de Sequência UML e Código Java*. Dissertação de Mestrado — Universidade Federal de Campina Grande, Campina Grande, Paraíba, Brasil, 2012. Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação. Citado na página 17.

MONTEZI, J. C. J. Algoritmo do caminho mínimo de dijkstra aplicado à linha de manufatura enxuta. *TAS Journal*, v. 1, n. 1, p. 30–39, 2017. Instituto Federal de Educação, Ciência e Tecnologia de São Paulo. Citado na página 23.

OAuth. *OAuth 2.0 Authorization Framework*. OAuth, 2024. Online. Acesso em: Agosto 2024. Disponível em: <https://oauth.net/2/>. Citado na página 19.

Object Management Group. *UML 2.0 Superstructure Specification*. 2005. Disponível em: <https://www.omg.org/spec/UML/2.0/Superstructure/>. Citado na página 17.

OLIVEIRA, S. C. d. *Educação ambiental para promoção da saúde com trânsito solidário*. Tese (Doutorado) — Universidade de São Paulo, 2013. Citado na página 10.

OPENLAYERS. 2023. Acessado em: 20 ago. 2024. Disponível em: <https://openlayers.org>. Citado na página 23.

ORENSTEIN, D. E. *Spatial Databases: A Tour*. [S.l.]: ACM Press, 1999. Citado na página 19.

OSRM. 2023. Acessado em: 20 ago. 2024. Disponível em: <https://github.com/Project-OSRM/osrm-backend>. Citado na página 23.

RESENDE, P. R. d.; CUZZUOL, V. M. A. Rideuff: desenvolvimento de aplicativo de carona solidária na universidade federal fluminense. *Revista Brasileira de Informática na Educação*, 2019. Citado na página 13.

SANTOS, D. M. dos. Bacharelado em Ciência da Computação. *Arquitetura Orientada a Serviços: Da Teoria à Prática*. Marília: FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA", 2008. Citado na página 17.

SOFTWARE, P. *Spring Boot*. 2024. Disponível em: <https://spring.io/projects/spring-boot>. Citado na página 19.

SOLÓRZANO, A. L. V.; CHARAO, A. S. Explorando a plataforma de computação em nuvem heroku para execução de programas paralelos com openmp. In: SOCIEDADE BRASILEIRA DE COMPUTAçãO (SBC). *Anais da XVII Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul*. [S.l.], 2017. Citado na página 17.

SOMMERVILLE, I. *Software Engineering*. 9. ed. [S.l.]: Addison-Wesley, 2011. Citado por: "Um aplicativo é um programa de computador que fornece uma funcionalidade específica para o usuário final.". Citado na página 22.

SOUZA, A. M.; GUIDONI, D.; BOTEGA, L. C.; VILLAS, L. A. Co-op: Uma solução para a detecção, classificação e minimização de congestionamentos de veículos utilizando roteamento cooperativo. In: *Anais do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*. [S.l.: s.n.], 2015. Citado na página 12.

Swagger. *Swagger Documentation*. Swagger, 2024. Online. Acesso em: Agosto 2024. Disponível em: <https://swagger.io/>. Citado na página 19.

TANENBAUM, A. S.; WETHERALL, D. J. *Redes de Computadores*. 5ª. ed. [S.l.]: Pearson, 2011. Citado na página 16.

Uber. *Quem somos*. 2024. Acesso em 20 de agosto de 2024. Disponível em: <https://www.uber.com/br/pt-br/about/?uclick_id=db89db3d-9df6-4e55-841d-dd941007b4be&utm_source=AdWords_Brand&utm_campaign=CM2341793-search-google-brand_25_-99_BR-National_o-d_web_acq_cpc_pt_T2_Generic_BM_uber_kwd-12633382_689375954776_164044604879_b_c&ad_id=689375954776&campaign_id=21469908197&kwid=kwd-12633382&kw=uber&gclid=CjwKCAjw_ZC2BhAQEiwAXSgClrPkS7Cnr-6A0LLJrVoEbV5wBqLGvqQs_4W0d9PZTZXWpcYZs0-6txoCSpYQAvD_BwE>. Citado na página 13.

Anexo 1: Código SQL- Função verifica_intersecao_rota

```
-- DROP FUNCTION public.verifica_intersecao_rota(int4, int4);

CREATE OR REPLACE FUNCTION public.verifica_intersecao_rota
(id_rota_coordenada integer, id_rota_linha integer)
 RETURNS boolean
 LANGUAGE plpgsql
AS $function$
DECLARE
    result BOOLEAN;
BEGIN
    WITH
        rota_data AS (
            SELECT
                ST_SetSRID(r1.coordenada_partida_geom, 3857)
                AS point_geom,
                ST_SetSRID(r1.coordenada_destino_geom, 3857)
                AS destino_geom,
                COALESCE(r2.raio, 100) AS raio_metros,
                ST_SetSRID(r2.rota, 3857) AS line_geom
            FROM rota r1
            JOIN rota r2 ON r2.id_rota = id_rota_linha
            WHERE r1.id_rota = id_rota_coordenada
        ),
        inicio_rota AS (
            SELECT
                ST_Length(
                    ST_Transform(
                        ST_LineSubstring(
                            rd.line_geom,
                            0,
                            ST_LineLocatePoint
                            (rd.line_geom, rd.point_geom)
                        ),
                        3857
                    )
                ) AS distancia_partida_ao_inicio
            FROM
                rota_data rd
        ),
        fim_rota AS (
            SELECT
                ST_Length(
                    ST_Transform(
                        ST_LineSubstring(
                            rd.line_geom,
                            0,
```

```
47                              ST_LineLocatePoint
48                              (rd.line_geom, rd.destino_geom)
49                          ),
50                          3857
51                      )
52                  ) AS distancia_destino_ao_final
53              FROM
54                  rota_data rd
55          )
56      SELECT
57          CASE WHEN
58              ST_DWithin(
59                  ST_Transform(rd.line_geom, 4326),
60                  ST_Transform(rd.point_geom, 4326),
61                  rd.raio_metros,
62                  FALSE
63              ) AND
64              ST_DWithin(
65                  ST_Transform(rd.line_geom, 4326),
66                  ST_Transform(rd.destino_geom, 4326),
67                  rd.raio_metros,
68                  FALSE
69              ) AND
70              fim.distancia_destino_ao_final >=
71              inicio.distancia_partida_ao_inicio
72          THEN
73              true
74          ELSE
75              false
76          END
77      INTO result
78      FROM
79          rota_data rd
80          CROSS JOIN inicio_rota inicio
81          CROSS JOIN fim_rota fim;
82
83      RETURN result;
84  END;
85  $function$
```

Listing 5.1 – Função para verificar interseção de rotas