UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ANDRÉ TELLES

Rota Certa - suporte computacional a roteirização veicular

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Marcelo Pimenta

Porto Alegre
2017

AGRADECIMENTOS

Agradeço a todos que contribuíram de alguma forma para que este trabalho pudesse ser realizado. Em especial agradeço a minha mãe Sonia, por ser meu refúgio, a meu pai Severiano, pelas palavras de sabedoria, e ao meu irmão Jardel pelo eterno companheirismo. Agradeço aos amigos que estiveram ao meu lado e que suportaram o mau humor típico dos fins de semestre. E finalmente aos colegas de graduação com os quais dividi os melhores e os mais difíceis momentos.

RESUMO

A mobilidade urbana é atualmente uma das maiores preocupações dos gestores nos grandes centros urbanos. O custo dos deslocamentos realizados dentro das cidades pode comprometer desde as mais simples tarefas cotidianas até grandes sistemas municipais. Atividades que dependem do deslocamento diário de veículos sofrem com o planejamento de rotas na tentativa de minimizar seus gastos e ainda assim realizar com eficiência o atendimento de todas as suas demandas. Este trabalho apresenta o desenvolvimento de uma ferramenta computacional para facilitar o planejamento de rotas veiculares, utilizando modernas tecnologias disponíveis, com o intuito de otimizar os custos envolvidos no processo: trata-se de um software de roteirização veicular composto por uma aplicação web para a criação, gerenciamento e acompanhamento das rotas e por um aplicativo móvel para auxiliar na execução das rotas e efetuar o registro das visitas aos destinos.

Palavras-chave: Aplicação Web. Aplicativo Móvel. Roteirização Veicular. Problema do Caixeiro Viajante Coletor de Prêmios.

Rota Certa - computational support for vehicular routing

ABSTRACT

Urban mobility is currently one of the major management concerns in large urban centers. The cost of travel within cities can compromise everything, from the simplest daily tasks to large municipal systems. Activities that depend on the daily displacement of vehicles suffer from the planning of routes in an attempt to minimize their expenses and still efficiently fulfill all their demands. This work presents the development of a computational tool to facilitate the planning of vehicular routes, using modern technologies available, in order to optimize the costs involved in the process. It is a vehicle routing software composed of a web application for the creation, management and monitoring of the routes, and a mobile application used to assist in the execution of the routes and register the visits to the destinations.

Keywords: Web Application, Mobile Application, Vehicular Routing, Prize Collecting Travelling Salesman Problem.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
BAAS	Back-end As A Service
CSS	Cascading Style Sheets
CSV	Comma Separeted Vlues
HTML	HyperText Markup Language
JS	JavaScript
JSON	JavaScript Object Notation
NoSQL	Not Only Structured Query Language
PCV	Problema do Caixeiro Viajante
PCVCP	Problema do Caixeiro Viajante COletor de Prêmios
SQL	Structured Query Language
VRP	Vehicular Routing Problem

SUMÁRIO

1 INTRODUÇÃO

A mobilidade urbana é atualmente uma das maiores preocupações dos gestores nos grandes centros urbanos. Estima-se que entre os anos de 2002 e 2012 a frota automotiva tenha aumentado 138,6% enquanto a população cresceu apenas 12,2% no mesmo período (PENA, 2017). O resultado disso são ruas e avenidas sempre cheias de veículos e os gastos com deslocamento cada vez maiores.

O custo dos deslocamentos realizados dentro das cidades pode comprometer desde as mais simples tarefas cotidianas até grandes sistemas municipais. Inúmeras soluções têm surgido em escala global visando otimizar os custos de deslocamento e escoamento dentro das cidades, seja por compartilhamento de veículos particulares (ex. Uber e Cabify), pelo aluguel de veículos alternativos (ex. BikePoa) ou pela combinação das melhores alternativas de transporte público disponíveis (ex. Moovit).

No município de Novo Hamburgo (RS) não é diferente. Seguindo a tendência global de crescente urbanização, a cidade tem visto seus dias iniciarem e acabarem em numerosas filas de carros pelas avenidas principais. Fato que tem afetado diretamente a equipe de Suporte Técnico de informática da prefeitura municipal, da qual o autor deste trabalho faz parte.

A equipe tem a responsabilidade de atender ocorrências e prestar suporte na área de informática nas mais diversas localidades do município, como postos de saúde, escolas de educação fundamental e infantil, hospitais e centros de atendimento social, assim como órgãos da própria administração, cada qual com um nível de prioridade de atendimento diferente dentro da hierarquia municipal.

Algumas destas localidades situam-se a dezenas de quilômetros de distância da sede administrativa, portanto, planejar o atendimento destas demandas tornou-se uma tarefa bastante desgastante e custosa.

Com o intuito de otimizar o tempo de planejamento, reduzir os custos e tornar o atendimento das demandas mais eficaz, o autor deste trabalho se propõe a desenvolver uma ferramenta computacional para o planejamento de rotas veiculares, utilizando as mais modernas tecnologias disponíveis.

1.1 Objetivo

O objetivo deste trabalho é o projeto e o desenvolvimento de um software de roteirização veicular composto por uma aplicação *web* para a criação, gerenciamento e acompanhamento das rotas e por um aplicativo móvel para auxiliar na execução das rotas e efetuar o registro das visitas aos destinos.

Este trabalho também tem o objetivo de empregar modernas técnicas e tecnologias para a efetivação do projeto, tais como a construção de uma aplicação web leve, responsiva e dinâmica, um aplicativo voltado a dispositivos móveis e a utilização de serviços de banco de dados não relacionais disponíveis na nuvem aliados a ferramentas de geolocalização.

1.2 Estrutura do trabalho

Este trabalho está dividido em 7 capítulos. O capítulo 1 contextualiza o problema que motivou a execução do trabalho, apresentando em seguida seus objetivos e estrutura do texto. O capítulo 2 apresenta as técnicas, conceitos e tecnologias utilizadas neste trabalho. O capítulo 3 apresenta sistemas, aplicações e trabalhos que se assemelham com este em seu objetivo, comparando-os entre si e, sobretudo, apontado as limitações que fomentaram este trabalho. O capítulo 4 fala sobre o desenvolvimento das duas versões do software, detalhando os métodos utilizados em cada uma delas e sua relação com as tecnologias empregadas. O capítulo 5 apresenta guia de uso das aplicações e apresentando suas funcionalidades em detalhes. O capítulo 6 divide-se em duas etapas, na primeira se faz uma avaliação dos algoritmos utilizados e na segunda, uma avaliação qualitativa do trabalho. Por fim, o capítulo 7 apresenta as conclusões fazendo retrospectiva aos resultados obtidos no trabalho, listando funcionalidades que podem ser implementadas em futuras revisões deste trabalho.

2 FUNDAMENTOS, CONCEITOS E TECNOLOGIAS

Neste capítulo será apresentada a base teórica na qual este trabalho foi fundamentado, os algoritmos de que se fizeram uso e as tecnologias que possibilitaram a execução deste projeto.

Dado que o foco deste trabalho é a criação de um sistema computacional que se apresenta em uma aplicação *web* e também em um aplicativo para dispositivos móveis, é fundamental que seja feita uma descrição detalhada destes conceitos, bem como dos elementos que dão suporte a estas tecnologias.

2.1 Metodologia ágil

Antes de detalhes tecnológicos, é necessário esclarecer que a construção deste trabalho se deu através da aplicação de técnicas e ferramentas da metodologia ágil para o desenvolvimento de softwares.

Metodologia ágil é um conjunto de técnicas de desenvolvimento de software que têm o objetivo de acelerar os processos, visando uma melhoria contínua e incremental, de modo a entregar parcelas funcionais do projeto aos usuários, possibilitando uma rápida resposta quando da necessidade de alterações.

Uma das características mais marcantes desta metodologia é a subdivisão dos objetivos em metas menores com janelas de tempo, tipicamente mensais, para sua conclusão. Estas janelas de tempo recebem o nome de *sprints* e tem o objetivo de, além de facilitar o acompanhamento das metas, propiciar a entrega parcial de etapas do projeto ao usuário.

Cada *sprint* é uma iteração do projeto e antes de ser iniciada, é realizado um planejamento prevendo todas as ações a serem realizada durante sua execução. Da mesma maneira ao final de cada iteração é feita uma retrospectiva de avaliação.

Desta maneira, uma iteração sempre começa com suas dependências satisfeitas pela iteração anterior, caso contrário, as tarefas desta deveriam ser adiadas para uma iteração futura, e seriam definidas novas tarefas para a iteração atual. Com isso, garante-se que cada funcionalidade a ser implementada passe por etapas de planejamento, análise de requisitos, projeto, codificação e testes dentro das iterações.

Dentre outros benefícios, a adoção desta metodologia almeja o aumento da comunicação e interação da equipe, organização diária para o alcance da meta definida, evitar falhas na elaboração, respostas rápidas às mudanças do projeto e aumento significativo da

produtividade (HELABS, 2017).

2.2 Aplicação Web

Aplicações *web* são sistemas de informática que preveem sua utilização através de navegadores de internet, fazendo uso de tecnologias como HTML, Javascript e CSS. Diferencia-se de um *website* pela sua finalidade. Enquanto um *site* é um conjunto de páginas *web* geralmente estáticas para informar ou expôr alguma informação, uma aplicação web permite a entrada, tratamento e manipulação de dados em páginas que respondem dinamicamente às solicitações de seu utilizador (SCRIPTCASE, 2017).

Um aplicativo web consiste basicamente de documentos HTML que são interpretados pelo navegador do computador, contudo, para que a aplicação tenha uma aparência mais agradável e uniforme e também para que possa responder dinamicamente às solicitações do usuário, utilizam-se recursos fornecidos por arquivos adicionais em Javascript e folhas de estilo CSS.

2.3 JavaScript e JQuery

O JavaScript é uma linguagem de programação do lado cliente, ou seja, é processada pelo próprio navegador. Com o JavaScript podemos criar efeitos especiais para páginas na *web*, além de proporcionar uma maior interatividade com os usuários. Por ser uma linguagem orientada a objetos ela trata todos os elementos da página como objetos distintos, facilitando a tarefa da programação e desenvolvimento (MOZILLA, 2017b). Em resumo, o JavaScript é uma poderosa linguagem de programação que possibilita criar páginas *web* dinâmicas e interativas.

Esta linguagem é suportada atualmente por todos os grandes navegadores de internet. Foi criada e é mantida pela European Computer Manufacturer's Association (ECMA), e responsabiliza-se pela camada de comportamento dos navegadores, enquanto as duas outras camadas de informação e formatação ficam a cargo, respectivamente, das páginas HTML e folhas de estilo CSS (TABLELESS, 2017a).

Devido a sua popularidade, muitas bibliotecas surgem para aprimorar o uso do Javascript. A mais popular delas é a biblioteca Jquery, disponibilizada em 2006, com o objetivo de fornecer simplificações através de camadas de abstração para facilitar a tarefa

de programação nesta linguagem (TABLELESS, 2017b).

A versão web do software objeto deste trabalho, o Rota Certa, utilizou tanto Javascript como principal linguagem de programação quanto elementos da biblioteca Jquery e assim foi possível criar uma aplicação responsiva e interativa, como poderá ser visto no capítulo 4.

2.4 Bootstrap

Bootstrap é o mais popular *framework* HTML, CSS, e JS para desenvolvimento de projeto para dispositivos móveis na web. Foi criado no Twitter em 2010 e oferece uma série de componentes e comportamentos desenvolvidos com JQuery. Seu foco são aplicativos móveis e sendo assim, oferece diversas classes CSS para manipular o conteúdo estático da página de modo que esta se adapte a tela de qualquer dispositivo de navegação (BOOTSTRAP, 2017).

Além de garantir uma aparência uniforme para a aplicação, este *framework* foi utilizado no trabalho principalmente por oferecer um conjunto de elementos suportados pelos principais navegadores de internet e por ser totalmente compatível com Javascript.

2.5 Bootbox

Para auxiliar na implementação da interface de usuário, utilizou-se a biblioteca Bootbox. Trata-se de uma biblioteca auxiliar que é responsável pela construção de caixas de diálogo utilizando modelos do framework Bootstrap.

Além de criar camadas de abstração para a implementação de caixas de diálogo, a biblioteca fornece apoio para realizar o tratamento das respostas do usuário, podendo encadeá-las diretamente a funções da aplicação *web* (BOOTBOX, 2017).

2.6 Sistema operacional Android e aplicativos móveis

Android é um sistema operacional baseado no *kernel* Linux e atualmente desenvolvido pela Google, sua dona desde 2005. É projetado para dispositivos móveis, ou portáteis, e voltado pra telas sensíveis ao toque, embora também seja implementado em dispositivos que não possuem tal característica (GOOGLE, 2017a).

O sistema é disponibilizado sob licença de código aberto, o que permite que desenvolvedores tenham acesso ao seu código-fonte para manipulação e experimentação. A natureza do software de código aberto do sistema estimulou uma grande comunidade paralela de programadores e entusiastas a desenvolver projetos que adicionam recursos para os usuários mais avançados, e disponibilizar atualizações para dispositivos mais antigos.

Aplicativo móvel é qualquer programa criado com a finalidade de ser executado em um ambiente como o descrito anteriormente, ou seja, em um dispositivo portátil com um sistema operacional dedicado para dar suporte a suas singularidades, como telas sensíveis a toque e elementos de orientação de posição.

A Google estimula oficialmente desenvolvedores interessados em criar aplicativos voltados para seu sistema operacional, oferecendo ferramentas para tal. Dentre estas ferramentas, a de maior destaque é o ambiente de desenvolvimento integrado Android Studio.

Através da utilização de tais ferramentas foi possível criar a aplicação que acompanha o projeto Rota Certa em sua versão para sistemas operacionais Android. Maiores detalhes serão vistos no capítulo 4.

2.7 Geolocalização e geocodificação

Geolocalização é o processo ou técnica de identificação da localização geográfica de uma pessoa ou objeto por meio de informações digitais processadas pela internet (DICTIONARY, 2017). Geralmente é associado aos sistemas de posicionamento global (GPS), porém qualquer outro meio de captura da posição global é válido, como a triangulação através de torres de telefonia celular.

Geocodificação segundo a definição do Google "é o processo de conversão de endereços (como "1600 Amphitheatre Parkway, Mountain View, CA") em coordenadas geográficas (como latitude 37.423021 e longitude -122.083739) que podem ser usadas para inserir marcadores em um mapa ou posicionar o mapa"(GOOGLE, 2017e). Ou seja, transforma endereços no formato tradicionalmente utilizados para coordenadas de localização geográfica sobre a superfícies terrestre, podendo também ser utilizada de maneira reversa traduzindo coordenadas em endereços aproximados.

Ambas técnicas serão empregadas neste trabalho através das ferramentas disponibilizadas pela interface de programação de mapas do Google.

2.8 Interface de Programação de Aplicação

Algumas empresas fornecem junto a um software ou a um serviço um conjunto de rotinas e padrões estabelecidos para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do software, mas apenas usar seus serviços. A este conjunto de funções dá-se o nome de Interface de Programação de Aplicação, ou API na sigla em inglês.

Neste projeto foram utilizadas duas API's do Google. A primeira delas e, sem dúvida, a interface que fundamentou este trabalho, a Google Maps API é responsável por fornecer o suporte a composição de mapas e às marcações com referência em geolocalização. Já a Firebase API fornece um serviço de back-end, de banco de dados e de autenticação de usuários. Cada uma dessas ferramentas será apresentada a seguir.

2.9 Firebase

Firebase é um serviço criado e desenvolvido em 2009 pela empresa Google e fornece hospedagem gratuita de conteúdo estático, como páginas HTML e arquivos javascript, sistema de autenticação através de e-mail ou de contas pré existentes de redes sociais e principalmente armazenamento de dados em formato JSON em um banco de dados não-relacional. Esta ferramenta foi desenvolvida com o intuito de garantir uma estrutura de serviços para desenvolvedores de aplicações web e móveis, antes capaz apenas por meio do aluguel ou aquisição de caros servidores (FIREBASE, 2017a).

Como banco de dados não-relacional, "todos os dados do Firebase Realtime Database são armazenados como objetos JSON. Você pode considerar o banco de dados como uma árvore JSON hospedada na nuvem. Diferentemente de um banco de dados SQL, não há tabelas nem registros. Quando você adiciona dados à árvore JSON, eles se tornam um nó na estrutura JSON existente"(FIREBASE, 2017b).

Este tipo de modelo de serviço é conhecido na bibliografia como *Backend as a service* (BaaS), onde uma empresa garante na nuvem um suporte a funcionalidades acessados através de uma API. Exatamente desta maneira os serviços oferecidos pelo Firebase foram utilizados neste trabalho, como uma forma de suporte para armazenamento de dados e para autenticação de usuários, como será explicado em detalhes na seção 4.6.

2.10 Google Maps

Google Maps é também um serviço fornecido e desenvolvido pela empresa Google. Possibilita a pesquisa e visualização de mapas e imagens de satélite da Terra. Está disponível gratuitamente para uso na internet desde 2005 e possui atualmente suporte a mapas completos de vários países e regiões, como Brasil, Estados Unidos e União Europeia (GOOGLE, 2017c).

O serviço disponibiliza muitas outras ferramentas, como informações de trânsito em tempo real, visualizações panorâmicas em 360 graus e planejamento de rotas simples. Além disso, o Google Maps oferece uma API, ou interface de programação da aplicação, que permite que os mapas sejam incluídos em páginas de terceiros, bem como em aplicativos para dispositivos móveis.

Existem também funções da API de mapas que disponibilizam informações pela geocodificação de uma localização geográfica através da tradução de um endereço informado pelo usuário.

Como será visto na seção 4.5, esta API foi de extrema importância na composição e execução deste projeto por garantir o suporte a mapas e oferecer as funcionalidades necessárias.

2.11 Problema do Caixeiro Viajante e Caixeiro Viajante Coletor de Prêmios

Na bibliografia, o problema do caixeiro viajante (PCV, ou TSP na silga em inglês) representa a dificuldade de um vendedor em percorrer um número `N` de cidades, passando por todas elas, de maneira a realizar o menor caminho possível, reduzindo o tempo necessário para a viagem e os possíveis custos com transporte e combustível (DANTZIG; FULKERSON; JOHNSON, 1954). O problema pode também ser generalizado como uma maneira de encontrar o menor laço fechado que conecta um número de pontos sobre um plano (CROES, 1958).

É um problema classificado como NP-difícil, o que quer dizer que existe um algoritmo determinístico para resolvê-lo cujo tempo de processamento cresce exponencialmente em função do tamanho da instância do problema e é caracterizado pela existência de um algoritmo não-determinístico que os resolve em tempo polinomial (RIBEIRO et al., 1997).

Muitos outros problemas podem ser reduzidos a instâncias do PCV, como por

exemplo a fabricação de circuitos integrados ou a perfuração de placas eletrônicas. Existem também generalizações do problema, onde incluem-se restrições e regras para o atendimento parcial dos destinos.

Uma destas generalizações é o Problema do Caixeiro Viajante Coletor de Prêmios (PCVCP, ou PCTSP, na sigla em inglês), onde cada vértice, ou cidade, possui um valor de prêmio e um valor de penalidade associados e o objetivo é encontrar uma rota que visite um subgrupo de vértices tal que o tamanho total da rota adicionado a soma das penalidades associadas aos vértices que não estão na rota sejam o menor possível (BIENSTOCK et al., 1993). Pode-se acrescentar ao algoritmo uma restrição na qual deve incluir em sua trajetória cidades suficientes para ganhar uma quantidade mínima de prêmio (BALAS, 1989).

Ao analisarmos a essência do problema de roteirização de veículos, também podemos facilmente reduzi-lo a uma instância do PCV e, portanto, lançar mão das mesmas soluções para encontrar as devidas respostas.

2.12 Algoritmo 2-OPT

O algoritmo de busca local 2-Opt foi inicialmente proposto em 1958 para a solução do problema do caixeiro viajante simétrico, de maneira mais eficiente que a busca exaustiva por força bruta (CROES, 1958). Este algoritmo necessita de uma rota inicial como entrada, e então realiza movimentos de troca de arestas. Durante cada movimento, duas arestas da rota são excluídas, quebrando-a em dois trajetos isolados e depois os reconecta na única outra maneira possível para que se crie uma nova rota entre todos os endereços (JOHNSON; MCGEOCH, 1997). Abaixo pode-se ver como o algoritmo realiza as trocas de arestas através de um o pseudo-código:

Figura 2.1: Pseudo-código do algoritmo de troca do 2-Opt

```
troca_2opt(rota, i, k) {
    nova_rota recebe (rota[inicio] até rota[i-1])
    concatena em nova_rota inverso(rota[i] até rota[k])
    concatena em nova_rota (rota[k+1] até rota[fim])
    retorna nova_rota
}
```

As arestas são trocadas enquanto houver uma melhora no tamanho total da rota. Como pode ser visto no pseudo-código a seguir:

Nota-se a presença da variável `numero_de_vertices_trocaveis`, e isto se

Figura 2.2: Pseudo-código do algoritmo 2-Opt

```
2opt(rota_atual){
    enquanto houver melhora
        melhor_distancia recebe distancia_total(rota_atual)
        para i de 0 até (numero_de_vertices_trocaveis - 1)
            para k de (i + 1) até numero_de_vertices_trocaveis
                nova_rota recebe troca_2opt(rota_atual, i, k)
                nova_distancia recebe distancia_total(nova_rota)
                se (nova_distancia < melhor_distancia)
                    rota_atual recebe nova_rota
                fim se
            fim para
        fim para
    fim enquanto
    retorna rota_atual
}
```

deve à existência de vértices que não podem ser trocados de ordem dentro da rota, como os pontos inicial e final, sob a penalidade de descaracterizá-la caso seja uma rota cíclica.

Como será visto na seção 4.7.2, este algoritmo foi de extrema importância para a construção da solução final deste trabalho, que seja a menor rota para os veículos realizarem as visitas aos destinos indicados.

2.13 Algoritmo Insere-Vértice ponderado por prêmios

Trata-se de um algoritmo proposto por (RIBEIRO et al., 1997) para solução do problema do caixeiro viajante coletor de prêmios. Utiliza uma ponderação entre a minimização da distância total da rota, a minimização da soma de penalidades dos locais não atendidos e uma maximização dos prêmios dos locais visitados na rota. Tem complexidade O(n^3) e abaixo podemos ver uma demonstração em pseudo-código:

Neste trabalho, o algoritmo Insere-Vértice foi usado para solucionar a aplicação de prioridades de atendimento dos destinos inseridos em uma rota, como será visto na seção 4.7.3.

Figura 2.3: Pseudo-código do algoritmo Insere-Vértice

```
Seja:
Pi - Penalidade do vértice i
Wi - Prêmio do vértice i
Cij - Distância da aresta i -> j
W0 - Valor mínimo de premiação

Então:
0 - Inicie a rota pela aresta com o maior valor de
    (P0 + Pj) / C0j,
onde 0 e j são os vértices da extremidade da aresta.

1 - Para cada vértice i fora da rota, calcule
    ( min( Cij + Cik - Cjk) - Pi) / Wi,
onde j e k são vértices já na rota e
min( Cij + Cik - Cjk) é o menor valor de acréscimo
de comprimento da rota causado pela inserção de i.

2 - Insira na rota o vértice que apresentar menor valor
calculado acima.

3 - Se W0 não satisfeito, volte ao passo 1.
```

3 APLICAÇÕES E TRABALHOS RELACIONADOS

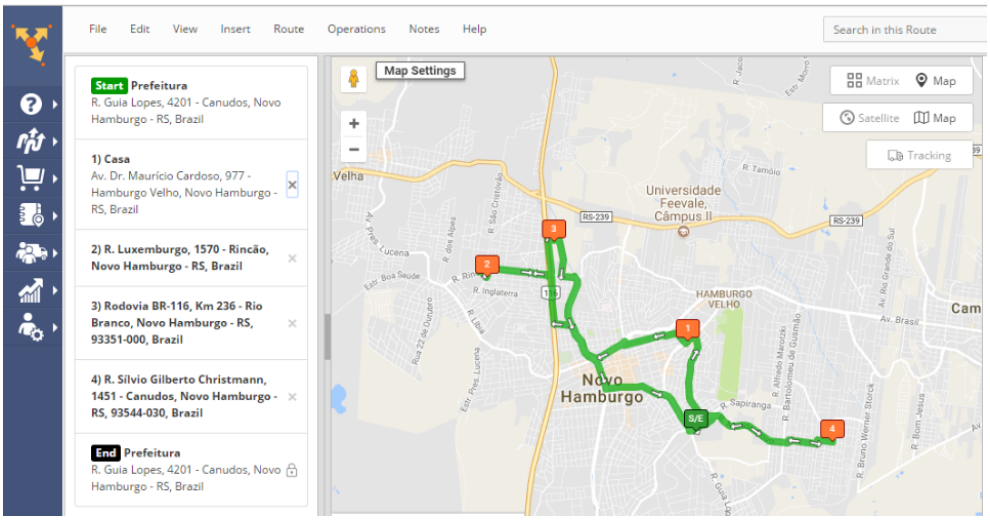
Neste capítulo serão apresentados alguns programas e aplicativos que oferecem aos seus usuários diferentes maneiras de programar rotas veiculares. Embora existam muitas ferramentas disponíveis para realizar tal tarefa, a maioria delas é paga ou oferece funcionalidades limitadas para usuários não assinantes de seu serviço. Será realizada uma breve análise das soluções propostas por esses aplicativos, dentro dos limites da gratuidade de cada um deles, e como podem ser uteis para o usuário.

3.1 Route4Me

Fundada em 2010 em Nova Iorque, Estados Unidos, a empresa Route4Me apresenta muitas soluções para logística. A principal delas é o serviço de planejamento de rotas veiculares que intitula a companhia, mas também oferece outras ferramentas de planejamento de depósitos, mapas, internet das coisas, GPS e telemática.

O software Route4Me é um dos mais completos desta pesquisa, permitindo o cadastramento de destinos e rotas, o compartilhamento de rotas entre usuários e o acompanhamento destas pelos gestores do sistema. Utiliza os mapas fornecidos pelo Google e a inclusão de destinos é possibilitada através de uma busca inteligente de endereços. Autodenomina-se o primeiro sistema de roteamento veicular a ter integração com dispositivos Android e IOS.

Figura 3.1: Route4Me - exemplo de utilização



É também um dos poucos sistemas pesquisados que oferece a opção para definição de prioridade dos destinos, porém, além de não esclarecer como este atributo funciona no

sistema, o impacto sobre a construção da rota foi nulo.

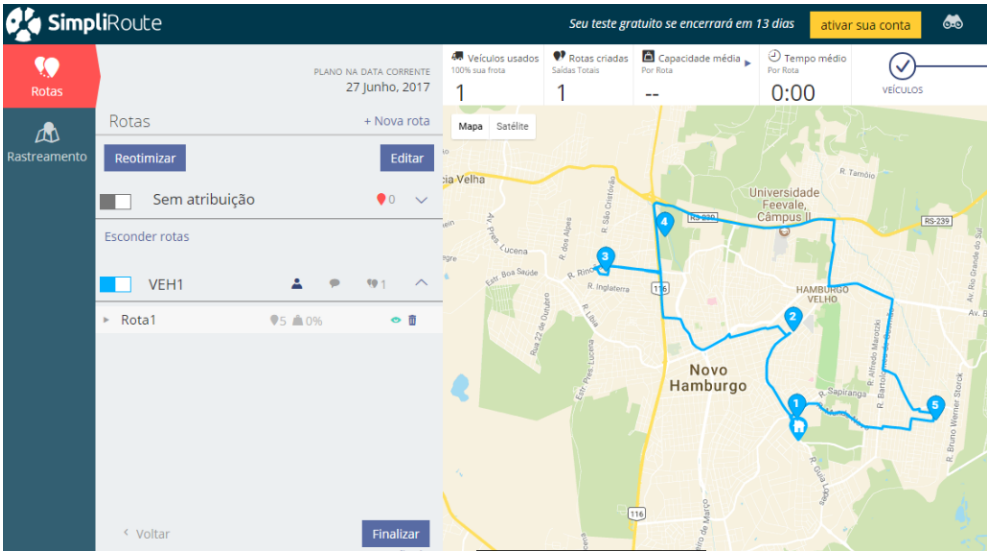
Entretanto, esta ferramenta é a de preço mais alto dentre as pesquisadas com planos que iniciam em US$999 para a administração de 100 veículos cadastrados, sendo US$69 adicionais para cada veículo extra. Fica evidente que o público-alvo são grandes corporações e empresas de logística.

3.2 SimpliRoute

SimpliRoute é uma empresa *startup* chilena que teve seu início em 2016 participando da incubadora de negócios do vale do silício 500 Startups (http://500.co/) e destina-se exclusivamente ao desenvolvimento do software de mesmo nome para roteirização e rastreamento veicular.

Seu produto também localiza-se entre os mais completos que esta pesquisa revelou, possuindo opções de armazenamento e atribuição de rotas aos motoristas e rastreamento da movimentação destes. Porém estranhamente não disponibiliza um cadastros de endereços de destinos ou de clientes, assim a rota deve ser criada sempre com a inclusão manual de todos os endereços, o que torna a tarefa repetitiva e cansativa para o usuário.

Figura 3.2: SimpliRoute - exemplo de utilização



Em seu favor conta o fato de ser o único sistema pesquisado que sustenta uma opção funcional para definição de prioridade de atendimento dos endereços. Contudo, trata-se de um atributo binário, onde pode-se definir se o destino é prioritário ou não. Na prática, por não existir uma escala mais gradual deste atributo, o sistema gera rotas onde os destinos prioritários são obrigatoriamente atendidos antes dos demais, não havendo

uma ponderação quanto à sua localização ou ao custo de inseri-los nesta ordem.

Também trata-se de um software oferecido como um serviço e tem planos de assinatura pelo valor de US$40 por veículo cadastrado.

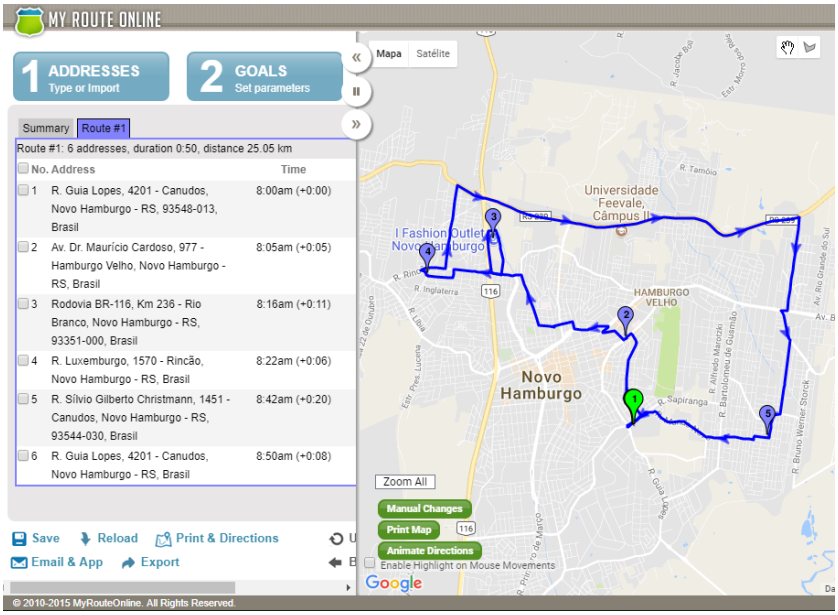
3.3 My Route Online

Projeto iniciado em 2009 por Baruch Axelrod para otimizar as entregas do comercio de flores de sua esposa, My Route Online acabou por tornar-se uma das grandes ferramentas disponíveis na atualidade.

My Route Online disponibiliza uma boa interface para o usuário e tem muitas opções para a construção de rotas, como a limitações de janela de tempo, limitações de distância e ainda subdivisão da rota entre vários veículos. Porém é a ferramenta que apresentou a maior lentidão ao realizar o cálculo da melhor rota.

Tem posição de destaque dentre as aplicações aqui listadas pois, apesar de ter planos de assinatura pagos, permite que o sistema seja utilizado gratuitamente, embora estabeleça algumas limitações como o número de destinos em uma rota ou a impossibilidade de utilizar os serviços de cadastramento de destinos e armazenamento de rotas. Para usuários com necessidades maiores, os planos de assinatura vão de US$20 a US$170.

Figura 3.3: My Route Online - exemplo de utilização



3.4 Rota Fácil

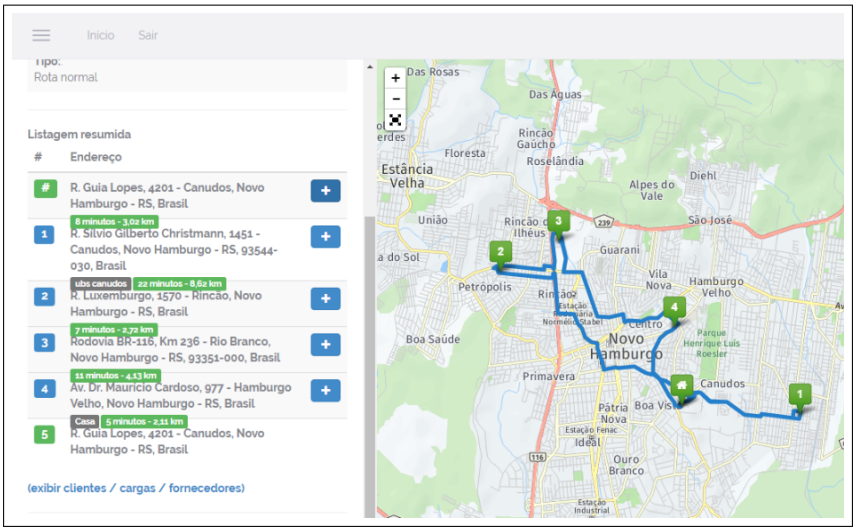
Rota Fácil é o único software de origem brasileira incluído nesta lista. Trata-se de um projeto bastante recente, tendo sido desenvolvido em 2016 e oferece grande atrativo ao usuário por ser inteiramente em português.

Este software apresenta as principais opções para que um sistema de roteamento veicular, comporta cadastramento de diversos tipos de destino, categorizando-os entre fornecedores e clientes, porém é o único que não prevê em seus cálculos janelas de tempo para o atendimento das demandas.

Outra singularidade deste sistema é a utilização dos mapas fornecidos por Leaflet e, talvez por limitação deste *framework*, também não oferece a possibilidade de inclusão de destinos através do apontamento com o *mouse* diretamente no mapa.

Oferece um único plano de assinatura pelo valor de R$ 36,90 por mês, permitindo a utilização de até 95 endereços por rota.

Figura 3.4: Rota Fácil - exemplo de utilização



3.5 Track Road

Track Route é um software da empresa Automated Digital Offices Corporation, que participa do mercado de software desde 1985, e também oferece um serviço de planejamento de rotas para veículos.

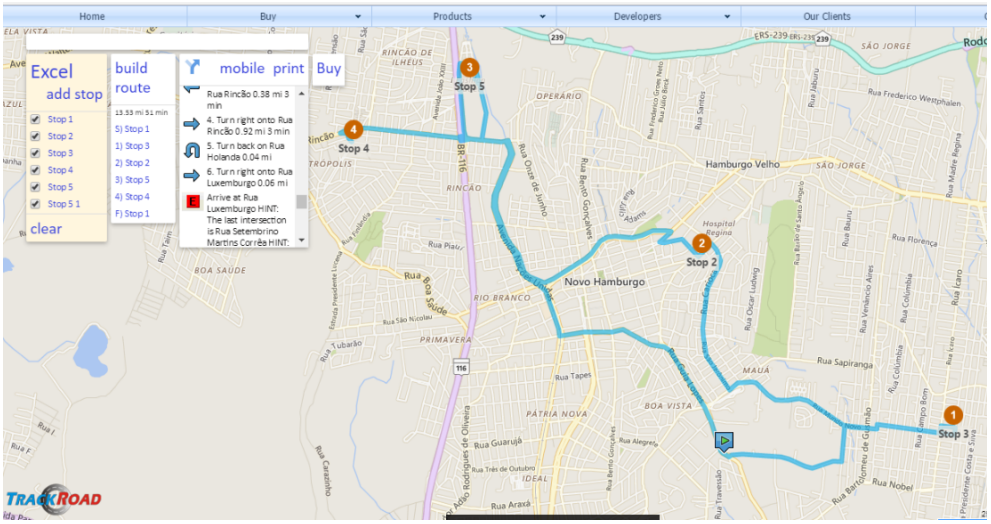
Esta aplicação tem uma interface bastante confusa e carece de uma reestruturação gráfica, mas oferece boas ferramentas para a construção de rotas e permite cálculos com

variáveis de tempo e distância, além da subdivisão de rotas entre vários veículos.

Utiliza mapas fornecidos pela Microsoft (Bing Maps) e não prevê rotas circulares, de modo que o usuário é forçado a indicar um destino extra para retorno ao ponto inicial da rota. Não possibilita o cadastramento de destinos ou de rotas, mas promete o compartilhamento de rotas entre usuários. Não foi possível verificar o funcionamento do compartilhamento de rotas pois esta é uma função disponível apenas para usuários assinantes do serviço.

As assinaturas do serviço variam de US$10 por mês, para o calculo de uma rotapor dia, a até US$2000 para empresas com até 1000 veículos e rotas com até 10.000 destinos.

Figura 3.5: Track Road - exemplo de utilização



3.6 Outras aplicações

Existem outras aplicações para a construção de rotas veiculares, porém não possuem a totalidade de funcionalidades necessárias para realizar o planejamento ou uma tomada de decisão. A seguir, uma breve descrição destas ferramentas:

- **Google Maps**: Oferece a possibilidade de consulta de endereços e encadeamento manual de destinos para a construção de uma rota. Utiliza informações de trânsito em tempo real para apresentar estimativas de tempo de viagem e distância, porém não realiza um cálculo de ordenamento entre os destinos para otimização de qualquer um destes atributos.
- **Plot a Route**: Ferramenta baseada nos mapas do Google que oferece a possibilidade de construção de rotas com as informações fornecidas por este. Também não

realiza cálculos de otimização da rota, mas permite armazená-las e compartilhá-las, além de fornecer ferramentas de manipulação de trechos do trajeto.

3.7 Comparativo de Funcionalidades

A partir dos resultados desta pesquisa é possível extrair um quadro comparativo das funcionalidades de cada aplicação. Estas funcionalidades foram agrupadas em categorias para facilitar sua compreensão e são mostradas na tabela 3.1.

- *Distribuição*: Refere-se ao modo de apresentação de cada aplicação, ou como o usuário pode ter acesso a elas.
- *Construção de rota*: Aqui foram agrupadas as funcionalidades referentes a construção de rotas, como a inclusão de destinos ou a sua ordenação.
- *Armazenamento*: Agrupa as opções de armazenamento de destinos e de rotas oferecidas por cada aplicação, além da possibilidade de compartilhamento com outros usuários.
- *Acompanhamento*: Lista as possibilidades de acompanhar o atendimento dos destinos da rota, atribuindo-a a um motorista, rastreando sua localização e auditando sua realização.
- *Cálculo*: Lista as opções que cada um dos sistemas oferece para a otimização de uma rota, por exemplo: tempo, distância ou prioridade. Também aqui mostra-se um comparativo do tempo de processamento destes cálculos classificados subjetivamente, na impossibilidade de mensurá-los com precisão.

Através deste comparativo, conseguimos perceber que nenhum dos trabalhos avaliados fornece algum tipo de classificação hierárquica para a visita dos destinos ou realiza cálculos de rotas adequados baseando-se em níveis de prioridade.

Além disto, todos os softwares estão disponíveis em sua versão completa apenas mediante alguma forma de assinatura, oferecendo curtos períodos para testes ou severas restrições de uso para não assinantes. Impossibilitando, desta maneira, a adoção de qualquer um deles por uma entidade pública, como a prefeitura de Novo Hamburgo, sem que seja realizado um complexo e burocrático processo de compra.

Com o intuito de preencher tais lacunas, este trabalho apresenta uma solução livre e gratuita para a construção de rotas veiculares com foco na hierarquização dos destinos através do estabelecimento de níveis de prioridade e obrigatoriedade de atendimento.

Tabela 3.1: Comparativo de funcionalidades das aplicações semelhantes

	Funcionalidades / Aplicação	*Route4Me*	*SimpliRoute*	*My Route Online*	*Rota Fácil*	*Track roads*	*Google Maps*	*Plot a Route*
Distrib.	Versão gratuita			✓		✓	✓	✓
	Versão paga	✓	✓	✓	✓	✓		
	Versão web	✓	✓	✓	✓	✓		✓
	Versão móvel	✓	✓		✓	✓	✓	✓
Constr.	Inclusão com o mouse	✓	✓	✓			✓	✓
	Busca por endereço	✓	✓	✓	✓	✓	✓	✓
	Busca por nome comercial	✓	✓	✓	✓		✓	✓
	Ordenação de destinos	✓	✓	✓	✓	✓		
Armaz.	Destinos	✓			✓			
	Rotas	✓	✓	✓	✓		✓	
	Compartilhamento	✓	✓	✓	✓	✓	✓	✓
Acomp.	Atribuição	✓	✓		✓	-		
	Rastreamento	✓	✓		✓	-		
	Auditoria	✓	✓		✓	-		
Cálculo	Prioridade	✓	✓					
	Tempo	✓	✓	✓	✓	✓		
	Distância	✓	✓	✓		✓		
	Subdivisão de rota	✓	✓	✓	✓	✓		
	Tempo de processamento	R	R	L	R	L		

Legenda: R - Rápido / L - Lento

4 DESENVOLVIMENTO DA APLICAÇÃO

O capítulo 4 desenvolve o processo de criação da aplicação, comentando decisões de projeto e as etapas de desenvolvimento do software Rota Certa. Os tópicos serão abordados em ordem semelhante a como foram desenvolvidos, dando prioridade para cobrir os principais elementos do projeto.

4.1 Requisitos e objetivos

Este trabalho foi desenvolvido seguindo a metodologia ágil de produção de software, e para tanto seguiu-se uma agenda de execução de tarefas por meio de iterações periódicas. A cada iteração eram definidas funcionalidades que deveriam ser adicionadas ao software, focando sempre na entrega de versões utilizáveis do programa ao fim do período.

Contudo, afim de esclarecer e identificar os objetivos do software, foi realizado o levantamento dos requisitos junto ao corpo técnico do setor de suporte de informática da prefeitura de Novo Hamburgo.

Definiu-se, portanto que a aplicação Rota Certa deve:

- Possibilitar a construção e edição de rotas veiculares, através da definição de destinos por meio de apontamento no mapa ou pela busca de endereço.
- Permitir a definição de prioridade e/ou obrigatoriedade de atendimento para cada destino, individualmente.
- Permitir a limitação no número total de destinos atendidos em uma rota.
- Selecionar automaticamente os destinos a serem atendidos por uma rota, através de uma ponderação entre o custo do atendimento (distância a ser percorrida) e prioridade.
- Possibilitar que cada usuário armazene e salve destinos de sua preferência.
- Permitir o armazenamento e compartilhamento de rotas completas com outros usuários do sistema.
- Permitir o registro e acompanhamento das visitas aos destinos.
- Ser livre e de uso gratuito.

Assim, ficou ficou estabelecido que aplicação Rota Certa seria dividida em duas

apresentações aos usuários: uma aplicação *web*, e um aplicativo para dispositivos móveis de sistema operacional Android. No primeiro, o usuário realizaria a criação, manutenção, gerenciamento, compartilhamento e edição de rotas veiculares, configurando os endereços, as prioridades, o traçado e a sequência de atendimento que as definem.

Por sua vez, o aplicativo móvel possibilitaria ao usuário fazer o acompanhamento e a confirmação de visitas aos endereços presentes em uma rota previamente configurada e cadastrada no aplicativo web.

4.2 Entidades

Este projeto trabalha com um grupo pequeno de entidades que se resumem às instâncias mais básicas planejadas para a aplicação. A primeira delas representa um destino, ou seja um local ou um endereço a ser visitado no mapa, e portanto deve possuir atributos referentes a sua localização, sua identificação, ao estado e prioridade. Um objeto **Destino** é composto dos seguintes atributos:

- **Nome**: Nome ou título dado ao destino. Pode ser atribuído pelo usuário no momento da criação.
- **Latitude**: Numeral (*float*) referente a latitude da posição geográfica do destino.
- **Longitude**: Numeral (*float*) referente a longitude da posição geográfica do destino.
- **Detalhes**: Endereço completo do destino.
- **Obrigatório**: Campo booleano que indica a obrigatoriedade deste destino pertencer a rota se verdadeiro.
- **Prioridade**: Numeral inteiro que informa o nível de prioridade do destino dentro da rota. Pode ir de 1 a 5.
- **Estado**: Campo de texto que informa o estado do destino quanto à sua visita.

Outra entidade faz a representação de uma rota e para tanto possui os atributos referentes a sua identificação, uma lista de objetos do tipo Destino, representando os locais incluídos na rota, além de informações do usuário autor e do motorista atribuído. Um objeto **Rota** é composto dos seguintes atributos:

- **ID**: Identificador único da rota, gerado no momento de sua criação.
- **Nome**: Nome ou título dado à rota. Pode ser atribuído pelo usuário no momento da criação.

- **Destinos**: Lista de objetos do tipo Destino. Representa os locais a serem visitados pela rota.
- **Melhor Rota**: Arranjo de números inteiros que representa a ordem na qual os destinos devem ser visitados.
- **Proprietário**: Identificador único do usuário que criou a rota.
- **Motorista Atribuído**: Identificador único do usuário que foi atribuído como motorista responsável por executar a rota.
- **Estado**: Campo de texto que informa o estado da rota quanto à sua execução.

Por fim a entidade Usuário que define um usuário do sistema. A maior parte dos dados desta entidade são provenientes do método de autenticação junto ao Firebase, que será mostrado na seção 4.6. São acrescidas então as informações referentes ao sistema Rota Certa, como os vínculos com as respectivas rotas pertencentes a este usuário. Portanto, um objeto **Usuário** é composto dos seguintes atributos:

- **ID**: Identificador único de usuário, gerado no momento de sua vinculação com o Firebase.
- **Nome**: Nome completo do usuário, fornecido pelo métode autenticação do Firebase.
- **E-mail**: Endereço de e-mail, também fornecido através da autenticação.
- **Rotas**: Lista de identificadores de objetos do tipo Rota. Cada identificador faz o vínculo de propriedade entre este usuário e uma rota.
- **Latitude**: Numeral (*float*) referente a latitude da última posição geográfica conhecida do usuário.
- **Longitude**: Numeral (*float*) referente a longitude da última posição geográfica conhecida do usuário..

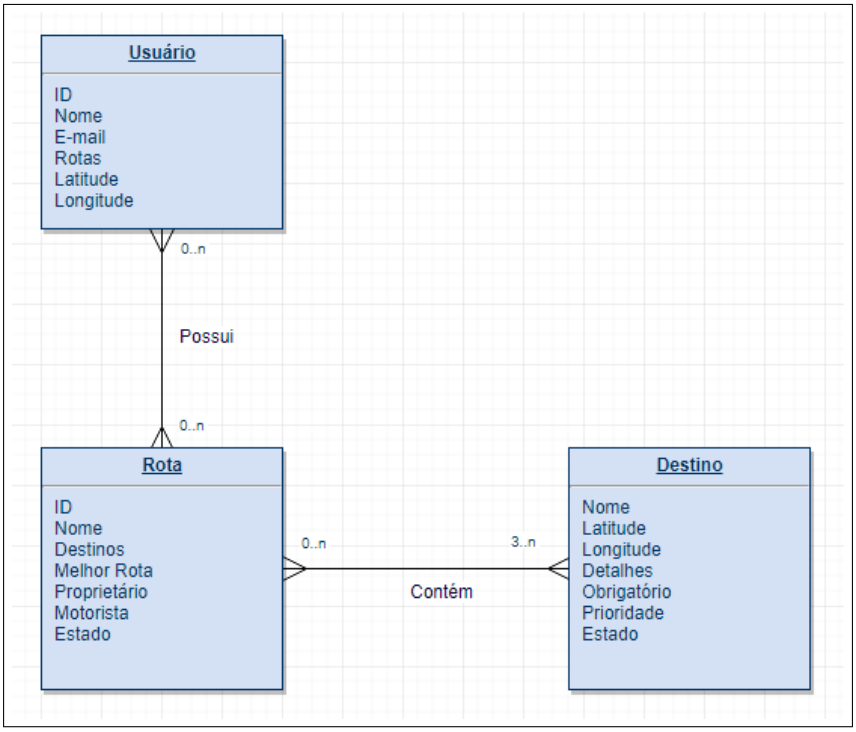
Por possuir poucas entidades definidas, o diagrama de relacionamento entre elas acaba sendo algo bastante simples, contudo ajuda a compreender o funcionamento do software. A figura 4.1 mostra o diagrama de relacionamento entre cada entidade e os demais componentes do sistema.

O relacionamento entre as entidades Usuário e Rota é *N para N*, pois um usuário pode possuir muitas rotas e uma rota pode ser compartilhada entre muitos usuário.

Da mesma maneira, o relacionamento de Rota com Destino é *N para N* pois uma rota deve possuir ao menos 3 destinos e um destinos pode ser incluído em diferentes rotas

planejadas.

Figura 4.1: Diagrama Entidade-Relacionamento



4.3 Aplicação web

Esta aplicação foi construída em uma única página HTML, deste modo cria-se uma grande dependência por recursos adicionais, pois o documento principal é carregado apenas uma vez, exatamente no momento do acesso.

O dinamismo da aplicação e a resposta aos comandos dos usuários é garantido através de documentos JavaScript auxiliares, que são responsáveis pela apresentação, modificação, inclusão e exclusão de elementos gráficos e controles da página, através da manipulação de trechos de código da página quando necessário.

Também foi utilizado o *framework* Bootstrap para a padronização da aparência e comportamento dos elementos da página bem como para garantir que a aplicação Web seja responsiva, ou seja, que funcione corretamente em vários tamanhos de tela de computadores, telefones ou dispositivos portáteis.

4.4 Aplicativo móvel

Para desenvolver o aplicativo móvel foi necessário configurar uma instância do ambiente de desenvolvimento integrado (IDE, na sigla em inglês) Android Studio, uma ferramenta oficialmente fornecida pelo Google para desenvolvedores de aplicações nativas para o sistema operacional Android.

Foi necessário também acrescentar bibliotecas ao Android Studio que permitiram o acesso ao Google Maps e Firebase (figura 4.2). A configuração para cada uma das delas será abordada das sub-sessões 4.5 e 4.6 respectivamente.

Figura 4.2: Configuração do Android Studio

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })

    compile 'com.android.support:appcompat-v7:25.3.1'
    compile 'com.android.support:design:25.3.1'
    compile 'com.android.support:cardview-v7:25.3.1'
    compile 'com.android.support:recyclerview-v7:25.3.1'
    compile 'com.android.support.constraint:constraint-layout:1.0.2'
    compile 'com.google.firebase:firebase-auth:10.2.1'
    compile 'com.google.firebase:firebase-database:10.2.1'
    compile 'com.google.code.gson:gson:2.7'
    compile 'com.google.android.gms:play-services-auth:11.0.2'
    compile 'com.google.android.gms:play-services-maps:11.0.2'
    compile 'com.google.android.gms:play-services:11.0.2'
    compile 'com.google.maps.android:android-maps-utils:0.4'

    testCompile 'junit:junit:4.12'
}
```

4.5 Mapa, marcadores e geolocalização

O ponto principal deste trabalho passa pela criação de uma maneira de identificar visualmente os destinos que devem ser visitados. Portanto, pensou-se imediatamente na utilização de uma aplicação centrada sobre mapa onde o usuário pudesse inserir os endereços desejados e visualizar a rora criada.

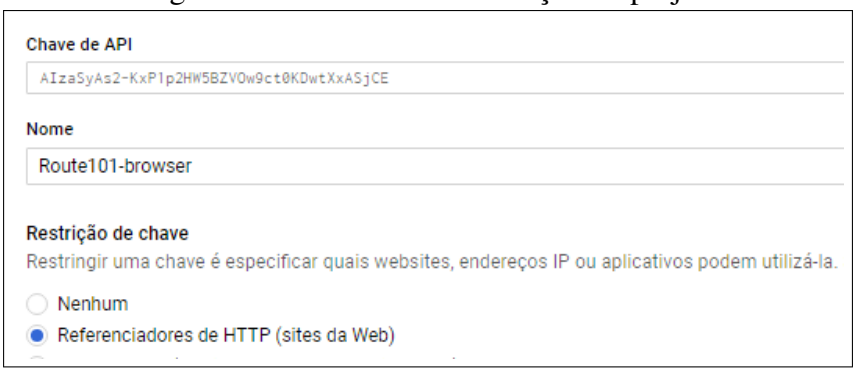
Existem algumas ferramentas que disponibilizam o suporte a mapas em páginas web, dentre elas destacam-se OpenStreetMap API e Google Maps API. A primeira tem em seu favor o fato de ser uma ferramenta de código aberto, de livre colaboração e sem

restrições legais ou técnicas para sua utilização. Porém, a escolha pela segunda se deveu à prévia experiência do autor deste trabalho com a API e à existência de uma base de documentos mais extensa e detalhada, possibilitando que a curva de aprendizagem fosse menor.

A Google Maps API foi utilizada nas duas apresentações do software objeto deste trabalho. Na aplicação Web foi utilizada a versão em JavaScript da API, já para o aplicativo móvel utilizou-se sua versão para Android construída com a linguagem de programação Java. Além diferenciarem-se entre si devido a serem desenvolvidas sob distintas linguagens de programação, cada uma das versões da API possui singularidades e limitações que serão abordadas nesta seção e no item 4.5.3 deste trabalho.

A utilização da API de mapas do Google passa inicialmente pela criação de uma chave de identificação, que é posteriormente vinculada a um projeto. Essa chave é responsável pelo controle de acesso às funcionalidades e identificação junto aos servidores do Google. Foi necessário adicioná-la tanto à aplicação web quanto ao aplicativo móvel (figura 4.3).

Figura 4.3: Chave de identificação do projeto



Uma das grandes vantagens da utilização desta API é o fornecimento de um painel de acompanhamento (figura 4.4) de estatísticas de uso que, graças a unificação sob a mesma chave de identificação, tornou possível administrar os acessos e ocorrências de eventuais erros nas duas versões da aplicação.

Para a apresentação do mapa na aplicação Web foi necessária a criação da um objeto `map` vinculado a um elemento `<div>` HTML pertencente a página (figura 4.5). Desta maneira, a biblioteca do Google Maps se encarrega da exibir o mapa dentro do elemento selecionado, sempre respeitando suas características, como tamanho, localização e profundidade, dentro da página.

Na aplicação Android, a inserção do mapa se dá de maneira semelhante. Cria-se inicialmente um fragmento em uma `activity` e a ele vincula-se a inicialização do mapa

Figura 4.4: Console de APIs do Google

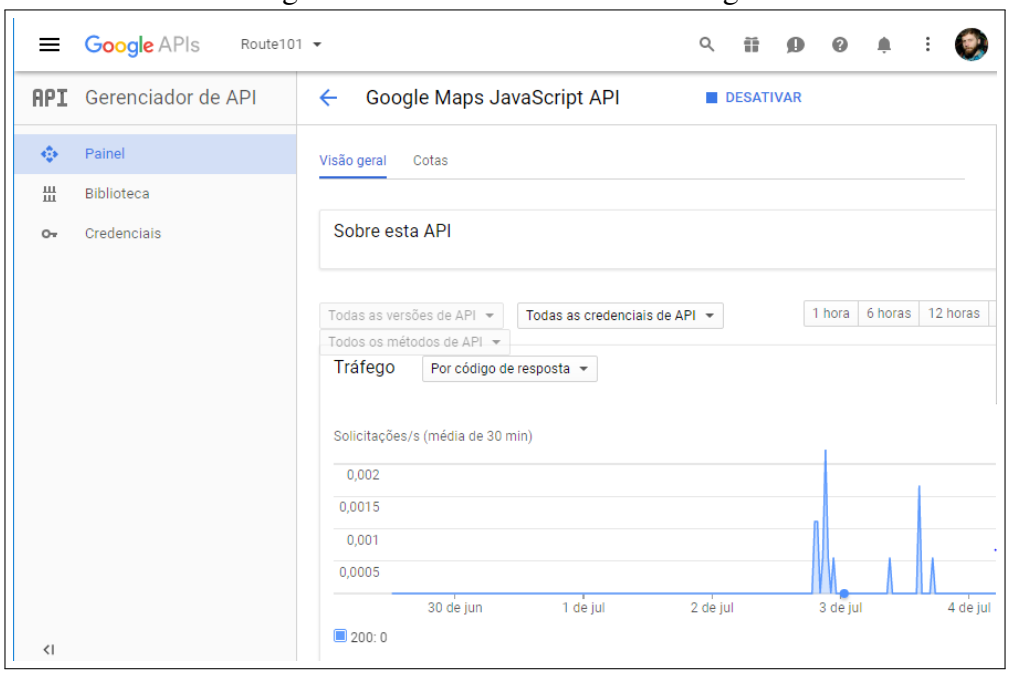


Figura 4.5: Método de inicialização do mapa na API JavaScript

```
prefeitura = new google.maps.LatLng(-29.694774, -51.115820);

var mapCanvas = document.getElementById("map");
var mapOptions = {
        center: prefeitura,
        zoom: 13,
        mapTypeId: google.maps.MapTypeId.ROADMAP,
        mapTypeControl: false
    }
map = new google.maps.Map(mapCanvas, mapOptions);
```

(figura 4.6). Assim como na aplicação Web, a biblioteca de mapas se responsabiliza pela exibição do mapa dentro das dimensões do fragmento indicado.

Figura 4.6: Método de inicialização do mapa no Android (Java)

```
SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
        .findFragmentById(R.id.map);
mapFragment.getMapAsync(this);
```

Tanto na apresentação Web quanto no Android, os mapas são dinamicamente apresentados e respondem a comandos do usuário como aproximação e afastamento do *zoom*, mudança de limites e centralização. Estes eventos são naturalmente suportados pela API que realiza a renderização do mapa conforme recebe tais comandos através de cliques ou gestos na tela, não sendo necessária nenhuma intervenção por parte do programador. Entretanto, algumas ações não são oferecidas nativamente e devem ser antes tratadas pelo programador para ampliar o horizonte de possibilidades ao usuário.

4.5.1 Inclusão de Marcadores

Muitas ações extras são utilizadas neste software, dentre elas a inclusão de marcadores, a busca por endereços e o desenho das rotas sobre as ruas do mapa. Para todas elas a API oferece suporte através de funções e métodos que devem ser implementados de acordo com as necessidades e especificações do projeto. Neste projeto, por exemplo, a inclusão de marcadores pode ser realizada de diversas maneiras, todas elas porém tem como base as mesmas funções.

Nas duas aplicações a abordagem para a inclusão de marcadores é bem semelhante: inicialmente cria-se um objeto `LatLng` com as coordenadas geográficas do endereço no formato `(latitude, longitude)`, posteriormente cria-se um objeto marcador, `marker`, passando como parâmetros obrigatórios, um título identificador do tipo `string`, um objeto mapa, do tipo `map`, no qual ele será exibido e uma localização geográfica do tipo `LatLng`, como a criada anteriormente (figura 4.7).

Figura 4.7: Método de criação de marcadores

```
var position = new google.maps.LatLng(destino.Latitude, destino.Longitude);

var marker = new google.maps.Marker({
    position: position,
    title: destino.nome,
    map: map,
    icon: 'https://maps.google.com/mapfiles/kml/paddle/red-blank_maps.png'
});
```

Contudo, o método de criação de marcadores depende das coordenadas de latitude e longitude de onde o marcador deve ser posicionado, informações estas que dificilmente são conhecidas pelo usuário. Por este motivo, a criação de marcadores não tem utilidade isoladamente, fazendo-se necessária a utilização de outras ferramentas e processos para que este método seja viável.

Tanto a biblioteca da API em JavaScript quanto sua versão em Java suportam eventos (GOOGLE, 2017d), e isto significa permitir a adição de elementos *listeners* ao mapa que aguardam uma determinada interação do usuário, como por exemplo o clique do mouse, ou a alteração de estado da página ou aplicativo. Estes eventos disparam gatilhos no programa e assim algumas informações do atual estado do mapa podem ser capturados por funções.

Utilizando o processo de geocodificação juntamente ao conceito de evento possibilitou-se, enfim, criar maneiras viáveis para que o usuário do sistema tenha controle sobre a inclusão de marcadores no mapa.

Na primeira delas, adiciona-se um elemento *listener* ao mapa, de modo que este verifique quando ocorre um clique do botão direito do mouse e, através da opção de geocodificação da API (figura 4.8), receba as coordenadas da posição apontada e as converta no endereço válido mais próximo ao local.

Figura 4.8: Método de geocodificação

```
function initRightClick(){
    var geocoder = new google.maps.Geocoder();
    google.maps.event.addListener(map, 'rightclick', function (e) {
        var position = e.latLng;
        geocoder.geocode({'location': position}, function (results, status) {
            if (status === google.maps.GeocoderStatus.OK) {
                if (results[0]) {
                    place = results[0];
                    insertTempMarker();
                } else {
                    window.alert('No results found');
                }
            } else {
              window.alert('Geocoder failed due to: ' + status);
            }
        });
    });
}
```

Outra ferramenta da Google Maps JavaScript API que foi empregada juntamente a geocodificação é a função de preenchimento automático para endereços e termos em uma barra de pesquisa (figura 4.9). Esta ferramenta deve ser vinculada a um elemento de entrada de texto do tipo `input` da página, e assim, "é possível usar o preenchimento automático para que os aplicativos ofereçam o comportamento de pesquisa durante a digitação do campo de pesquisa do Google Maps. Quando um usuário começa a digitar um endereço, o preenchimento automático insere o restante"(GOOGLE, 2017f).

Figura 4.9: Método de inicialização da barra de busca de endereços

```
var autocomplete = new google.maps.places.Autocomplete(
      (document.getElementById('address'))
);

autocomplete.addListener('place_changed', function () {
    place = autocomplete.getPlace();
    if(place.geometry.location != undefined){
        insertTempMarker();
    }
});
```

Todo navegador atual de internet possui suporte a geolocalização, isso quer dizer que "disponibiliza acesso de conteúdo Web à localização do dispositivo. Isso permite que um Web site ou aplicativo ofereçam resultados customizados baseado na localização do usuário"(MOZILLA, 2017a).

Assim, utiliza-se o método `getCurrentPosition` (figura 4.10) para capturar a posição geográfica mais recente do usuário, possibilitando a inclusão de um marcador neste local.

Figura 4.10: Método de geolocalização do navegador

```
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(function (position) {
        var geocoder = new google.maps.Geocoder();
        var Lat = position.coords.latitude;
        var Lng = position.coords.longitude;
        var latlng = new google.maps.LatLng(Lat,Lng);

        geocoder.geocode({'location': latlng}, function (results, status) {
            if (status === google.maps.GeocoderStatus.OK) {
                if (results[0]) {
                    place = results[0];
                    insertTempMarker();
                } else {
                    window.alert('No results found');
                }
            } else {
              window.alert('Geocoder failed due to: ' + status);
            }
        });
    });
}
```

4.5.2 Renderização de Rotas

Após a inclusão dos marcadores em um mapa, a biblioteca do Google Maps permite calcular distâncias entre eles e até renderizar o traçado de rotas através do serviço *Directions*. Deste serviço, foram utilizadas dois métodos em conjunto: `route` e `DirectionsRenderer`. O primeiro deles é responsável pelo cálculo da rota entre dois pontos quaisquer de origem e destino, podendo incluir-se também pontos intermediários que devem ser visitados entre eles, e que retorna a distância entre eles e instruções para a renderização da rota entre todos pontos na ordem em que foram informados. Com essas informações armazenadas, pode-se acionar o segundo método para renderização da rota no mapa. A dependência e utilização dessas funções pode ser vista na figura 4.11.

Entretanto, este serviço tem a característica de ser assíncrono (GOOGLE, 2017g), o que quer dizer que, por ter um alto custo de processamento em servidores externos, o resultado de uma solicitação de qualquer das funções pode levar certo tempo para ser retornado ao solicitante, neste caso, as aplicações *web* e Android deste projeto.

Tomando o cuidado para que a assincronia do serviço prestado pelo Google Maps não afetasse o software em nenhuma de suas aplicações, o cálculo de rotas sobre o mapa e a renderização de rotas foram evitados a todo custo e utilizados apenas para o traçado

Figura 4.11: Métodos de consulta e renderização de rotas

```
var request = {
    origin:start,
    destination:end,
    waypoints: waypts,
    travelMode: google.maps.TravelMode.DRIVING
};

directionsService.route(request, function (result, status) {
    if (status == google.maps.DirectionsStatus.OK) {
        drawRoute(result);
        getInstructions(result);
    }
});
```

```
var directionsDisplay = new google.maps.DirectionsRenderer({
    suppressMarkers: true,
    preserveViewport: true,
    polylineOptions: polylineOpt,
});
directionsDisplay.setMap(map);
directionsDisplay.setDirections(route);
```

da rota final, quando as demais funções do software já haviam realizado as otimizações necessárias. Essa e outras limitações da API serão abordadas na seção seguinte.

4.5.3 Limitações da Google Maps API e soluções encontradas

Apesar de apresentar muitas soluções e prestar um enorme serviço à comunidade oferecendo tais serviços, a ferramenta disponibilizada pelo Google possui muitas limitações de uso a usuários com licença de uso padrão (gratuita). Serão listadas aqui algumas das limitações que foram encontradas durante a execução deste projeto e a solução desenvolvida pelo autor deste trabalho para contorná-las.

A primeira grande limitação encontrada foi na utilização do método `route` do serviço *Directions* para cálculo do trajeto entre pontos. Esta função permite uma solicitação aos servidores que inclui os pontos de origem e destino, podendo ser incrementada com até 23 pontos intermediários, não mais que isso (GOOGLE, 2017g).

A solução encontrada, neste caso, foi realizar uma divisão da solicitação pautada pelo limite oferecido, realizando várias consultas simultâneas cada uma com um pedaço da consulta original e posteriormente uma concatenação das *polylines* das respostas em uma única rota íntegra traçada no mapa. As funções responsáveis por esta solução podem ser analisadas na figura 4.12 a seguir.

Figura 4.12: Funções de divisão das consultas e captura de *polyline*.

```
function calcRoute(routeArray){
    var startDestination = markers[routeArray[0]].position;
    var endDestination = markers[routeArray[routeArray.length-1]].position;
    var waypoints = [];

    for(var i=1; i < routeArray.length ; i++){
        if((i % 23) == 0 || (i == routeArray.length-1)){
            endDestination = markers[routeArray[i]].position;

            getPolyline(startDestination, endDestination, waypoints);

            startDestination = markers[routeArray[i]].position;
            waypoints = [];
        }
        else {
            var destination = {
                location: markers[routeArray[i]].position,
                stopover: true
            };
            waypoints.push(destination);
        }
    }
    return true;
}
```

Embora esta solução exija um maior número de consultas aos servidores, a limitação de consultas permitidas por este tipo de licença, que também existe, possui uma margem bem maior de tolerância para as necessidades do projeto, e pôde ser desconsiderada.

Outra limitação enfrentada, e esta com um impacto maior sobre o projeto, foi na tentativa de utilização do serviço de Matriz de Distâncias oferecido pela API. Com este serviço, seria possível enviar uma solicitação com uma lista de locais, e como retorno receber uma matriz contendo a distância e tempo de deslocamento do cruzamento entre todos os locais informados (GOOGLE, 2017b).

Contudo, este serviço limita a resposta a uma matriz de cem resultados de cruzamento entre locais, gerando uma matriz de cruzamento entre no máximo dez locais diferentes. Para o objetivo deste projeto, o limite estabelecido por este serviço delimitaria as consultas a um quinto do tamanho médio previsto.

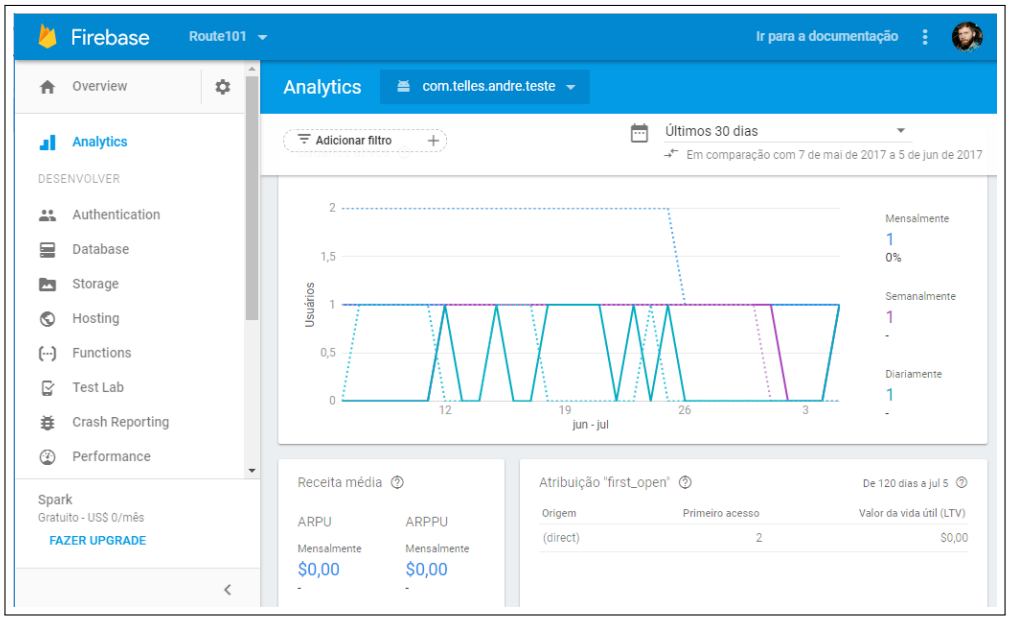
Visto que a matriz de distâncias é um elemento primordial na construção deste projeto, a solução para contornar esta limitação teve se ser muito bem pensada, e a partir disto, decidiu-se abrir mão da precisão em sua construção. Assim, a matriz foi implementada através de funções que fazem o cálculo da distância em linha reta entre coordenadas geográficas, tornando-se na prática uma matriz de distâncias aproximadas e não reais, porém de construção mais prática e rápida. Esta solução será analisada com mais detalhamento no item 4.7.1.

4.6 Banco de Dados

Uma vez solucionado no projeto qual seria a ferramenta utilizada para a utilização de mapas, restava escolher a ferramenta que seria responsável pelo armazenamento das informações permanentes e não voláteis do sistema, como a lista de usuários e os dados de cada um deles. Obviamente, as opções limitavam-se à utilização de um banco de dados e, inicialmente, cogitaram-se as alternativas mais populares, como por exemplo um servidor MySQL com interfaces de escrita e leitura desenvolvidas em PHP. Entretanto, o recente contato com novas tecnologias e a vontade de adaptar-se a elas, fez com que o autor deste trabalho optasse pela utilização de um novo paradigma em bancos de dados. Escolheu-se, deste modo, mais um serviço disponibilizado pelo Google, o Firebase.

Assim como o Google Maps, o serviço do Firebase também é disponibilizado através de uma interface de programação de aplicação (API), que pode ser utilizada tanto por aplicações *web* (biblioteca JavaScript), quanto por aplicativos móveis para Android. Outra semelhança entre as API's é o fornecimento de um console *web* para realização da administração da biblioteca, onde é possível fazer manutenção e visualizar toda a estrutura dos dados armazenados, ademais também é permitido inserir e remover manualmente elementos no banco.

Figura 4.13: Console Firebase



Para utilizar o serviço é necessário criar um projeto junto ao console do Firebase e, de maneira similar ao Ggoogle Maps, criar uma chave de autorização de uso da API que deverá ser vinculada ao projeto. Na aplicação *web* deve-se invocar um método de inicia-

lização com os parâmetros da configuração do projeto contendo tal chave, já na aplicação móvel deve-se vincular um arquivo de configuração em formato JSON ao projeto do Android Studio, como podem ser vistos nas imagens abaixo.

Figura 4.14: Método de inicialização do Firebase em JavaScript

```
var config = {
    apiKey: "AIzaSyB64MdcVIS0rnZcz0GjVm3u8VOysmneHwM",
    authDomain: "route101-141314.firebaseapp.com",
    databaseURL: "https://route101-141314.firebaseio.com",
    storageBucket: "route101-141314.appspot.com",
    messagingSenderId: "371832317001"
};
firebase.initializeApp(config);
```

Figura 4.15: Trecho do arquivo de configuração do Firebase no Android

```
"project_info": {
  "project_number": "371832317001",
  "firebase_url": "https://route101-141314.firebaseio.com",
  "project_id": "route101-141314",
  "storage_bucket": "route101-141314.appspot.com"
},
"client": [
  {
    "client_info": {
      "mobilesdk_app_id": "1:371832317001:android:d2e6f1234ec5ce0a",
      "android_client_info": {
        "package_name": "com.telles.andre.teste"
      }
    },
```

4.6.1 Autenticação

Além de fornecer um serviço de banco de dados, a Firebase API também providencia serviço de autenticação de usuários que pode ser realizado através da vinculação de contas de redes sociais (Facebook e Twitter) ou de contas do próprio desenvolvedor Google. Também é possível construir uma nova base de autenticação através de endereços de e-mail. Para facilitar a implementação do projeto, foi escolhido o método de autenticação através de contas do Google, abstraindo assim as tarefas de controle, criação e manutenção de contas de usuário, além de garantir uma maior compatibilidade com dispositivos Android.

O sistema de autenticação fornecido pelo Firebase funciona pela chamada de métodos específicos fornecendo algumas parametrizações básicas, e que então se tornam responsáveis pelo tratamento das credenciais de cada usuário. Conforme já citado anteriormente, para este software foi implementado o modo de autenticação através de contas

do Google, excluindo-se os demais métodos possíveis. Nas figuras 4.16 e 4.17 abaixo, podem ser visualizados os métodos de autenticação conforme foram usados nas aplicações Web e móvel.

Figura 4.16: Método de autenticação do Firebase em Javacript

```
var authButton = document.getElementById('loginButton');

authButton.addEventListener('click', function () {
    firebase.auth()
        .signInWithPopup(provider)
        .then(function (result) {
            var token = result.credential.accessToken;
            displayName.innerText = result.user.displayName;
            currentUser = result.user;
            registerUser(result.user);
            $("#login").hide();
            $("#logout").show();
            initFirebaseListeners();
            isLoggedIn = true;
            updateUI();
        }).catch(function (error) {
            console.log(error);
        });
});
```

Figura 4.17: Método de autenticação do Firebase para Android

```
GoogleSignInOptions gso = new GoogleSignInOptions
        .Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
        .requestIdToken(getString(R.string.default_web_client_id))
        .requestEmail()
        .build();
mGoogleApiClient = new GoogleApiClient.Builder(this)
        .enableAutoManage(this, this )
        .addApi(Auth.GOOGLE_SIGN_IN_API, gso)
        .build();

mAuth = FirebaseAuth.getInstance();
mAuthListener = new FirebaseAuth.AuthStateListener() {
    @Override
    public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth) {
        user = firebaseAuth.getCurrentUser();
        if (user != null) {
            Log.d(TAG, "onAuthStateChanged:signed_in:" + user.getUid());
        } else {
            Log.d(TAG, "onAuthStateChanged:signed_out");
        }
        updateUI(user);
    }
};
```

Logo após, o software recebe o resultado do processo de autenticação e, em caso de sucesso, armazena as informações do usuário em memória para tratá-las futuramente, bem como para garantir os devidos acessos aos usuários.

4.6.2 Acesso e Gravação de Dados

O armazenamento dos dados em uma banco Firebase é bastante distinta de um banco de dados relacional tradicional, onde ao invés de armazená-los em estruturas tabelares como estes últimos, é criado uma árvore hierárquica em formato de um documento JSON.

Este fato implica que uma consulta aos dados tem que ser realizada de maneira diferente. Enquanto um banco de dados relacional faz este acesso através da Linguagem de Consulta Estruturada (SQL, na sigla em inglês), a API do Firebase oferece métodos para escutar uma porção do documento JSON armazenado. Para isso é necessário criar um elemento de referência à um dos nós do documento e associar a ele um elemento do tipo *listener* ativado pela mudança de estado do nó referido. Ou seja, caso haja uma mudança nos dados armazenados sob o nó referido, dispara-se um evento que ativa uma reação do programa frente aos dados alterados.

Uma referência à raiz do documento JSON é realizada através da chamada do método `firebase.database.ref()`. Para referenciar-se um trecho específico do documento, deve-se realizar a navegação pela árvore acessando os ramos filhos com o método `child`. O resultado da consulta depende de como os dados estão armazenados sob o nó referenciado, contudo as ocorrências retornadas podem ser organizados pelo valor dos atributos que possuem e para isso é necessário utilizar o método `orderByChild`.

Além disso, uma consulta ao banco pode ser realizada através de dois métodos, que diferenciam-se basicamente pela duração de sua atuação. Enquanto o método `on` fica ativo permanentemente desde sua criação até a finalização da conexão com o banco, respondendo a eventos de alteração dos dados, o método `once` realiza uma única consulta a uma referência. Entretanto é possível desligar uma referência de consulta permanente através do método `off`.

A seguir, podemos ver exemplos de consultas ao banco de dados Firebase que são realizadas neste trabalho (figuras 4.18, 4.19 e 4.20).

Figura 4.18: Método para carregar locais favoritos

```
function getFavoriteDestinations(){
    refFavoriteDestinations.child(currentUser.uid).orderByChild("nome")
        .on('value', function(snapshot) {
            illFavoritesTable(snapshot.val());
        }, function (errorObject) {
        console.log("The read failed: " + errorObject.code);
    });
}
```

Figura 4.19: Método para carregar rotas salvas

```
refUsers.child(currentUser.uid).child('routes').on('value', function(snapshot){
    initSavedRoutes();
    snapshot.forEach(function(child){
        refRoutes.child(child.key).once('value', function(snapshot){
            var userRoute = {};
            userRoute[snapshot.key+''] = snapshot.val();
            insertSavedRoute(userRoute);
        }, function (errorObject) {
            console.log("The read  UserRoutes failed: " + errorObject.code);
        });
    });
}, function (errorObject) {
    console.log("The read  Routes failed: " + errorObject.code);
});
```

Figura 4.20: Método para carregar usuários

```
function getUsers(routeId){
    refUsers.once('value', function(snapshot) {
        fillUsersTable(snapshot.val());
    }, function (errorObject) {
        console.log("The read failed: " + errorObject.code);
    });
}
```

A gravação de dados no Firebase também difere bastante se comparada às realizadas em bancos de dados relacionais. A API fornece dois métodos para isto, no primeiro deles o elemento a ser gravado possui identificação e atributos definidos, utiliza-se portanto o método `set`, que grava um objeto em um determinado local apontado pela referência. Se o elemento não possui uma chave identificadora, utiliza-se então o método `push`, que cria um nó com uma identificação única, gerada dinamicamente, sob o nó referenciado e então salva o objeto sob essa chave.É possível utilizar os dois métodos em conjunto quando se quer ter um controle sobre as chaves identificadoras geradas pela API.

Da mesma maneira que acontece com as consultas, para que um dado seja gravado um uma posição específica do documento JSON, deve-se navegar a árvore através dos nós utilizando o método `child`. A seguir, pode se verificar a utilização das funções de gravação neste projeto:

Figura 4.21: Método para salvar destinos favoritos

```
function saveFavorite(destination) {
    refFavoriteDestinations.child(currentUser.uid)
        .push(destination);
}
```

Figura 4.22: Método para salvar rotas

```
function saveRoute(obj, key) {
    console.log(obj, key);
    if(key == "" || key == null || key == undefined){
        obj.owner = currentUser.uid;
        var newKey = refRoutes.push(obj).getKey();
        refUsers.child(currentUser.uid).child('routes')
            .child(newKey).set('routeID');
        return newkey;
    }
    else{
        refRoutes.child(key).set(obj);
        return key;
    }
}
```

Figura 4.23: Método para cadastrar usuários no primeiro acesso

```
function registerUser(user){
    refUsers.child(user.uid).child('userData').set({
        email: user.email,
        displayName: user.displayName,
        photoURL: user.photoURL,
    });
}
```

4.7 Aplicação dos algoritmos e heurística

Após a escolha das principais ferramentas para a implementação deste projeto, coube ao autor tratar de solucionar questões de caráter mais teórico. Algumas limitações encontradas impuseram a necessidade da busca por soluções criativas e de um estudo mais profundo na bibliografia.

Trataremos nesta seção sobre as soluções encontradas para a implementação de uma matriz de distâncias de construção rápida, para otimização do cálculo da menor rota e para acrescentar o cálculo de prioridades através da solução do problema do caixeiro viajante coletor de prêmios (PCVCP).

4.7.1 Matriz de distâncias

Um dos elementos mais importantes para a construção deste projeto é a definição de uma matriz de distâncias, só a partir dela é possível realizar o cálculo de menor rota entre determinados endereços. Conforme comentado anteriormente no item 4.5.3, a API fornecida pelo Google Maps disponibiliza uma ferramenta de obtenção desta matriz,

porém com sérias restrições e limitações de uso, impedindo assim sua utilização neste projeto.

Para contornar tal problema, decidiu-se abrir mão da precisão das distâncias oferecidas pela API e construir manualmente a matriz de distâncias através de formulas matemáticas de aproximação da distância entre duas coordenadas geográficas na superfície terrestre. Obviamente, esta aproximação matemática não considera o trajeto das vias urbanas ou o impacto do trânsito, do relevo e das barreiras geográficas entre dois destinos. Outra característica desta implementação é tornar a matriz de distâncias simétrica sobre sua diagonal principal, ou seja, a distância calculada entre duas coordenadas é a mesma não importando qual delas é considerada o ponto de origem ou destino, sendo também resultado da aproximação matemática.

Entretanto, nos testes realizados, os resultados apresentados foram suficientemente satisfatórios e bastante próximos à aplicação com a matriz fornecida pela API do Google Maps. Além disso, possibilitou uma matriz grande o bastante para cumprir o objetivo deste trabalho, limitada apenas pela memória do dispositivo.

Podemos ver a implementação de cada uma das etapas do cálculo de aproximação matemática da distância entre duas coordenadas geográficas nas figuras 4.24 e 4.25.

Figura 4.24: Método para criação da matriz de distâncias

```
function initDistanceMatrix(markers){
    for(var i=0; i < markers.length; i++){
        for(var j=0; j < markers.length; j++){
            if(i==j){
                setMatrixElement(i,j,0);
            }
            else if(i<j){
                var dist = distanceInKmBetweenEarthCoordinates(
                                markers[i].position.lat(),
                                markers[i].position.lng(),
                                markers[j].position.lat(),
                                markers[j].position.lng()
                            );
                setMatrixElement(i,j,dist);
                setMatrixElement(j,i,dist);
            }
        }
    }
}
```

Figura 4.25: Método para calcular distância entre duas coordenadas

```
function distanceInKmBetweenEarthCoordinates(lat1, lng1, lat2, lng2) {
    var earthRadiusKm = 6371;
    var dLat = degreesToRadians(lat2-lat1);
    var dLng = degreesToRadians(lng2-lng1);

    lat1 = degreesToRadians(lat1);
    lat2 = degreesToRadians(lat2);

    var a = Math.sin(dLat/2) * Math.sin(dLat/2) +
            Math.sin(dLng/2) * Math.sin(dLng/2) * Math.cos(lat1) * Math.cos(lat2);

    var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    return earthRadiusKm * c;
}
```

4.7.2 Solução para caixeiro viajante e heurística 2-Opt

Uma vez construída a matriz de distância chegamos a outro ponto central deste trabalho: a solução do menor caminho a ser percorrido entre todos os endereços escolhidos.

Na bibliografia este problema é conhecido como o problema do caixeiro viajante (PCV), onde um vendedor deve percorrer algumas cidades, passando por todas elas e percorrendo o menor caminho possível, reduzindo o tempo necessário para a viagem e os possíveis custos com transporte e combustível, como pode ser visto em detalhes na seção 2.11.

Inicialmente abordou-se este problema através da solução mais trivial encontrada, um algoritmo de busca pela solução ótima testando todas as soluções possíveis para cada conjunto de cidades.

Esse tipo de algoritmo costuma ser chamado de solução por força bruta e teve desempenho péssimo em casos onde o número de cidades ultrapassou o limite da primeira dezena. Este fato foi observado logo nos primeiros testes, se tornando uma séria limitação à eficiência do software. Esta análise pode ser vista no capítulo de avaliação de desempenho, na seção 6.1.

Existe uma vasta literatura destinada a análise do TSP e uma bibliografia ainda maior na área de otimização combinatória apontando algoritmos de solução para minimizar os custos da solução trivial para o problema. Apoiando-se em um dos estudos desta área, optou-se por construir a solução com o auxílio de uma heurística de trocas, ou heurística de intercâmbio, o algoritmo 2-Opt.

Como apresentado na seção 2.12, o algoritmo de busca local 2-Opt realiza movimentos de troca de arestas em uma rota inicial definida, excluindo duas delas e posterior-

mente reconectando os dois trajetos restantes na única outra maneira possível.

Na definição tradicional do algoritmo, as trocas são feitas enquanto houver uma melhora no tamanho total da rota construída. Porém, a implementação do algoritmo neste projeto estabeleceu um número de tolerância para movimentos sem melhora afim de evitar mínimos locais. Desta maneira, o algoritmo aceita algumas soluções piores que a encontrada, antes de encerrar as buscas, como pode ser visto na figura 4.26.

Figura 4.26: Funções 2-Opt e 2-Opt Swap

```
function twoOptSwap(route, i, k){
    var route1 = route.slice(i,k);
    var route2 = route.slice(k,route.length);
    route = route.slice(0,i);
    return route.concat(route1.reverse()).concat(route2);
}
```

```
function twoOpt(existingRoute){
    var improvement = 0;
    var limit = existingRoute.length * existingRoute.length;

    while(improvement < limit){
        var bestLocalDistance = calculateTotalDistance(existingRoute);

        bestDistance = bestLocalDistance;
        bestRoute = existingRoute;

        for(var i = 1; i < (existingRoute.length - 1); i++) {
            for (j = i + 1; j < (existingRoute.length); j++) {
                var swappedRoute = twoOptSwap(existingRoute, i, j);
                var newDistance = calculateTotalDistance(swappedRoute);

                if (newDistance < bestLocalDistance) {
                    improvement = 0;
                    existingRoute = swappedRoute;
                    bestDistance = bestLocalDistance;
                    bestRoute = existingRoute;
                }
            }
        }
        improvement++;
    }
}
```

No capítulo de avalização dos resultados deste trabalho, na seção 6.1, realizaremos uma comparação confrontando os dois algoritmos expostos nesta seção, busca por força bruta e o algoritmo de trocas 2-Opt.

4.7.3 Solução para o caixeiro viajante coletor de prêmios

Tendo implementado a solução para o caixeiro viajante e garantindo um cálculo eficiente da menor rota, o desafio seguinte seria implementar uma maneira de atribuir ao

cálculo valores relacionados à prioridade de atendimento de cada local.

A necessidade de tal implementação advém de um requisito do projeto: tendo um limite de visitas a se fazer, quais dos locais devem ser atendidos e quais devem ser negligenciados no momento. Este é um problema de decisão que vai além da minimização dos gastos da rota e deve selecionar uma sub-rota dentro da rota principal que maximize o número de locais de alta prioridade.

Mais uma vez, recorreu-se a bibliografia e após algumas pesquisas foi encontrada uma generalização do problema do caixeiro viajante que satisfaria as necessidades deste projeto.

Trata-se do caso especial nomeado Problema do Caixeiro Viajante Coletor de Prêmios (PCVCP), onde cada vértice, ou destino, possui um valor de prêmio e um valor de penalidade associados e o objetivo é encontrar uma rota que visite um subgrupo de vértices tal que minimize-se tanto o tamanho total da rota quanto a soma das penalidades associadas aos vértices que não visitados, maximizando o prêmio adquirido nos vértices pertencentes a rota. Uma explicação mais detalhada pode ser encontrada na seção 2.11.

Tendo identificado o problema, bastou mapeá-lo e adaptá-lo para que o valor relativo às penalidades do algoritmo fosse substituído pelo nível de prioridade, desta maneira quanto maior a prioridade de atendimento de um local, menor será a chance de ser excluído da rota final. Outra adaptação realizada, foi a equiparação dos valores de prêmio, sendo assim todos os locais tem peso igual e o algoritmo, quando restrito a satisfazer uma quantidade mínima de prêmios, na verdade estará restrito a visitar um número mínimo de cidades.

Muitos algoritmos foram analisados para a solução do PCVCP, entretanto foi escolhido o algoritmo Insere-Vértice, apresentado na seção 2.13, por balancear em sua fórmula o impacto dos valores de distância entre os vértices e os atributos penalidade e prêmio. Contudo, cada aplicação deve ser calibrar o algoritmo para as necessidades de cada ambiente, aumentando ou diminuindo a relação entre cada um dos atributos.

Na versão final do software, ambos algoritmos 2-Opt e PC-TSP trabalham em conjunto, pois mesmo após a definição de quais serão os endereços pertencentes a sub-rota que atenda os requisitos de prioridade, é necessário que esta seja otimizada, minimizando custos de deslocamento.

A implementação do algoritmo em JavaScript pode ser visto na imagem 4.27 abaixo.

Figura 4.27: Função Insere-Vértice para PCVCP

```
function pcTSP(gone, toGo, minPrize){
    var bestRelation;
    var bestVertice;

    while (calculateTotalPrize(gone) < minPrize){
        for(var i=0;i<toGo.length; i++){
            var minRelation;

            for(var j=0;j<gone.length; j++){
                for(var k=j+1;k<gone.length; k++){
                    var relation = (matrix[toGo[i]][gone[j]]
                                    + matrix[toGo[i]][gone[k]]
                                    - matrix[gone[j]][gone[k]]);
                }
                if(j == 0){
                    minRelation = relation;
                }
                if(relation < minRelation){
                    minRelation = relation;
                }
            }
            minRelation = minRelation / destinations[toGo[i]].priority;
            if(i == 0){
                bestRelation = relation;
                bestVertice = i;
            }
            if (minRelation < bestRelation){
                bestRelation = minRelation;
                bestVertice = i;
            }
        }
        gone.push(toGo[bestVertice]);
        toGo.splice(bestVertice,1);
    }
    return gone;
}
```

5 GUIA DE USO

Este capítulo faz uma análise mais profunda sobre a aplicação desenvolvida, Rota Certa, focada na utilização, navegação e organização dos elementos. Nas próximas sessões serão demonstradas cada uma das funcionalidades do software em suas duas apresentações, tanto da aplicação Web quanto do aplicativo móvel, esclarecendo o que pode ser realizado e mostrando o comportamento visual em respostas às ações e necessidades do usuário.

5.1 Aplicativo Web

A aplicação *web* apresenta todas as opções ao usuário em uma única página que dinamicamente altera sua aparência e conteúdo conforme recebe as solicitações deste. Nesta página é possível criar e gerenciar rotas veiculares, alterando seu trajeto ao inserir locais de atendimento ou ajustando a prioridade de cada um dos endereços. Ao gerenciar as rotas, o usuário pode salvá-las para carregá-las posteriormente, bem como compartilhá-las com outros usuários do mesmo sistema.

5.1.1 Apresentação geral

De maneira geral, a aplicação gira em torno da apresentação do mapa e da manipulação de marcadores dentro deste, porém a maior parte das ações do usuário se dá através do menu lateral. É através deste menu que o usuário tem acesso as principais funções do software. Na tela inicial (figura 5.1) são apresentadas as opções de criação de uma nova rota, cadastro de novos endereços ou o carregamento de rotas previamente salvas. Além disso, a barra superior oferece ao usuário a possibilidade de realizar a autenticação.

5.1.2 Autenticação

Ao acessar o software Rota Certa, o usuário pode autenticar-se através de uma conta do Google. Para isso, deve-se selecionar a respectiva opção na barra superior e então, será apresentada uma tela com as opções de login gerenciada pela ferramenta de autenticação do Firebase quando ligada ás contas Google, como explicado na seção 4.6 e

Figura 5.1: Tela inicial da aplicação Web Rota Certa

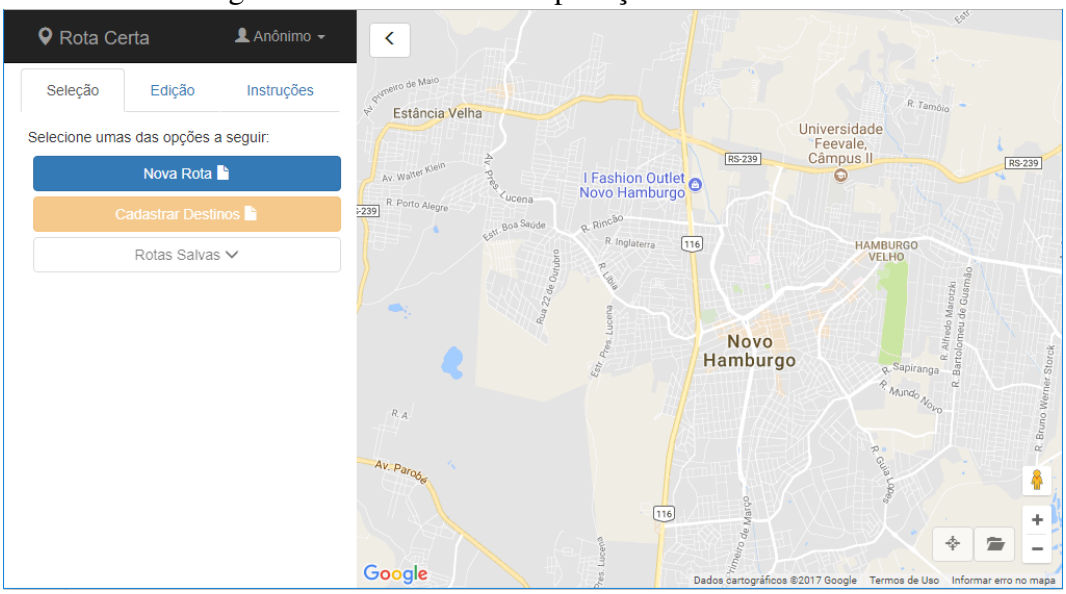
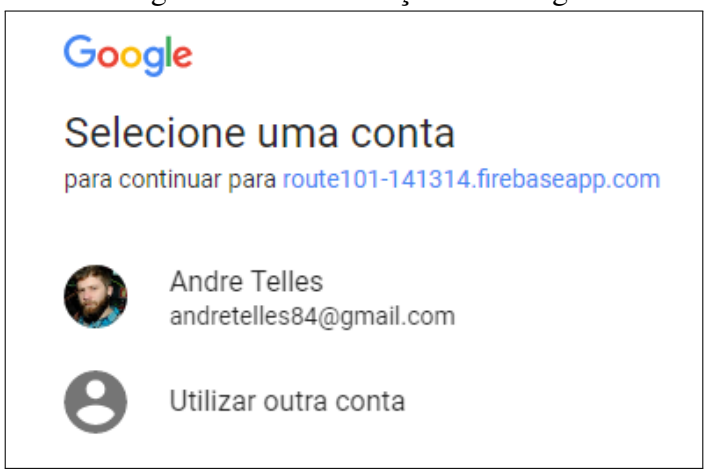


Figura 5.2: Autenticação via Google

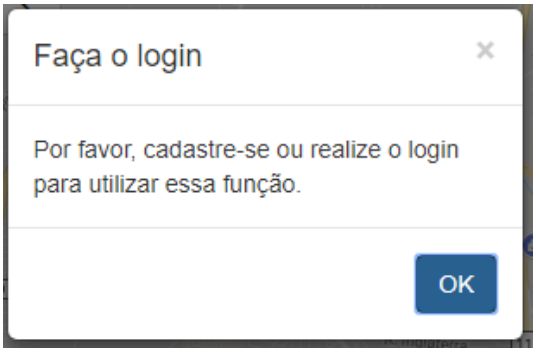


pode ser visto na figura 5.2.

Entretanto, é possível utilizar o sistema mesmo sem realizar a autenticação e algumas funcionalidades básicas ficam disponíveis logo no acesso inicial. O sistema assim foi pensado devido a algumas ações não terem dependência de dados armazenados ou por não necessitarem de controle de acesso de usuário. Assim, é possível realizar ações como construir uma rota com endereços informados manualmente e realizar um cálculo instantâneo de prioridades. Estes dados não são armazenados e são descartados tão logo o usuário deixe o sistema. A cada ação não autorizada pela falta de autenticação, uma janela informativa (figura 5.3) é apresentada solicitando ao usuário autenticar-se.

Após realizar o login, são liberadas todas as funções da página ao usuário e os respectivos botões deixam de ter a aparência de inativos. A partir de então, é autorizada a utilização de algumas funcionalidades como o salvamento e carregamento de endereços e

Figura 5.3: Caixa de diálogo de aviso para autenticação



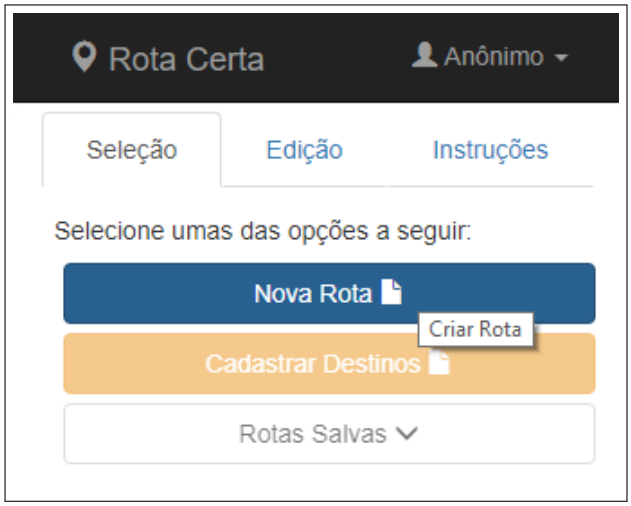
de rotas, bem como o compartilhamento de rotas entre usuários.

5.1.3 Construir rotas

O objetivo principal deste trabalho é a construção de rota veiculares. Para um usuário, essas rotas são compostas basicamente por um ponto inicial, que deve ser o destino final de retorno, uma lista de destinos a serem visitados e um traçado coerente entre todos estes locais.

Logo na tela inicial do software Rota Certa, o menu lateral de navegação apresenta as opções de seleção de rotas. Neste momento, o usuário pode selecionar a opção de criação de uma nova rota, como pode ser visto na figura 5.4, e será, então, levado para a aba de edição do menu.

Figura 5.4: Menu lateral - criação de nova rota



A troca do estado de seleção para o estado de edição, também ocorre se o usuário inserir um destino através da manipulação direta do mapa com o botão direito do *mouse*, como será explicado na seção a seguir 5.1.4.

5.1.4 Inserir destinos

Uma das funções básicas do sistema é a inclusão de destinos pra o cálculo da rota final. Foram elaboradas diversas maneiras para sua implementação, ampliando as possibilidades para o usuário e consequentemente a versatilidade do software.

Uma das maneiras de adicionar novos destinos é através do clique do botão direito do *mouse* em determinado local apontado no mapa. Com isso o sistema dispara a função de geocodificação, explicada anteriormente na seção 4.5.1, e insere um marcador com o endereço válido mais próximo ao local indicado.

Outra forma possível é utilizando a barra de busca de endereços no topo do menu lateral, quando selecionada a aba de edição de rota. Ao inserir um endereço na barra, sugestões são apresentadas através do recurso de autocompletar. As previsões vão sendo apresentadas abaixo da barra de pesquisa e o usuário pode selecioná-las através das teclas direcionais, como pode ser visto na imagem 5.5.

Figura 5.5: Barra de busca de endereços

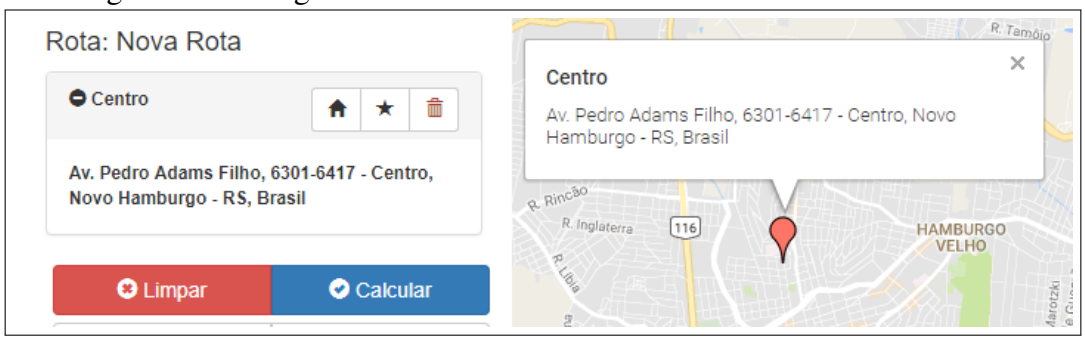
Para qualquer uma dessas opções, um marcador temporário é inserido solicitando uma confirmação da inclusão do destino, onde o usuário deve informar também um título para identificação do local, como pode ser visto na imagem 5.6. Uma vez realizada a confirmação, o marcador é adicionado definitivamente ao mapa possuindo uma janela de informações e o destino é adicionado a listagem no menu lateral (figura 5.7).

5.1.4.1 Opções de edição

Ao ser adicionado á listagem lateral, cada destino possui três botões com opções de edição, como pode ser visto na figura 5.7, que são respectivamente nomeados de Início, Favorito e Remover. Os nomes assim foram escolhidos para que intuitivamente possa-se compreender a função de cada um deles.

Figura 5.6: Marcador temporário

Figura 5.7: Listagem lateral de destinos e infobox do marcador definitivo



A opção Início é utilizada para designar qual será o ponto inicial, e por consequência o destino final de retorno, da rota. Por definição do conceito de rota cíclica, apenas um dos destinos pode ser selecionado como inicial, portanto, ao selecionar-se um segundo destino como tal, automaticamente o primeiro o deixa de ser. Se nenhum destino for selecionado, o primeiro da lista será considerado como inicial.

Através do opção Remover, o usuário pode excluir o destino selecionado, removendo-o da listagem da rota e seu respectivo marcador do mapa do software. Já a opção Favorito será explicada a seguir.

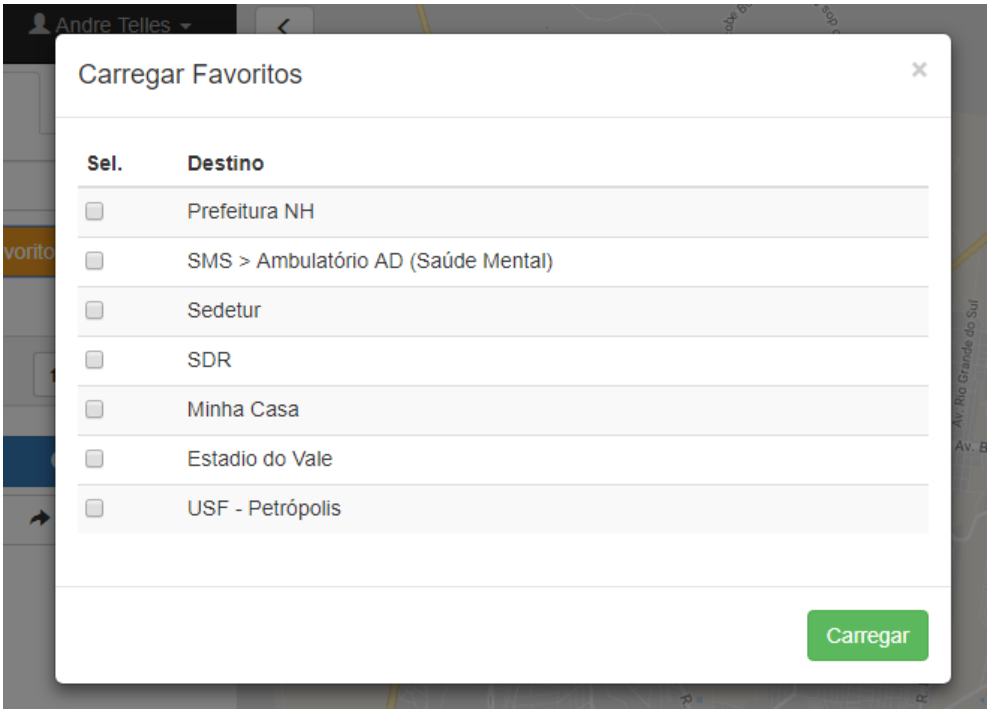
5.1.4.2 Locais favoritos

Durante a construção de nova uma rota, é possível também adicionar destinos previamente salvos, nomeados aqui de Locais Favoritos. Essa listagem pode ser consultada acessando-se a respectiva opção na aba de edição de rota, logo abaixo da barra de buscas (figura 5.5), e é apresentada no formato de uma lista de seleção de itens em uma janela informativa sobre a página principal.

Nesta janela, o usuário deve fazer uma seleção, que pode ser unitária ou múltipla, de destinos a serem adicionados ao mapa dentre seus Destinos Favoritos, (5.8). Quando

confirmada a escolha, cada um dos destinos recebe um marcador no mapa referente à sua localização e uma entrada na listagem apresentada no menu lateral, assim como acontece quando um destino é adicionado manualmente.

Figura 5.8: Janela de seleção dos destinos favoritos



Existem duas maneiras para que um destino pertença à lista de Locais Favoritos, a primeira delas é através da opção de edição Favorito, identificada pela estrela. Quando selecionada esta opção, uma janela de confirmação é apresentada ao usuário que, se confirmada, insere o destino entre os Locais Favoritos assim como este foi incluído na rota, ou seja, com o título e coordenadas geográficas atuais.

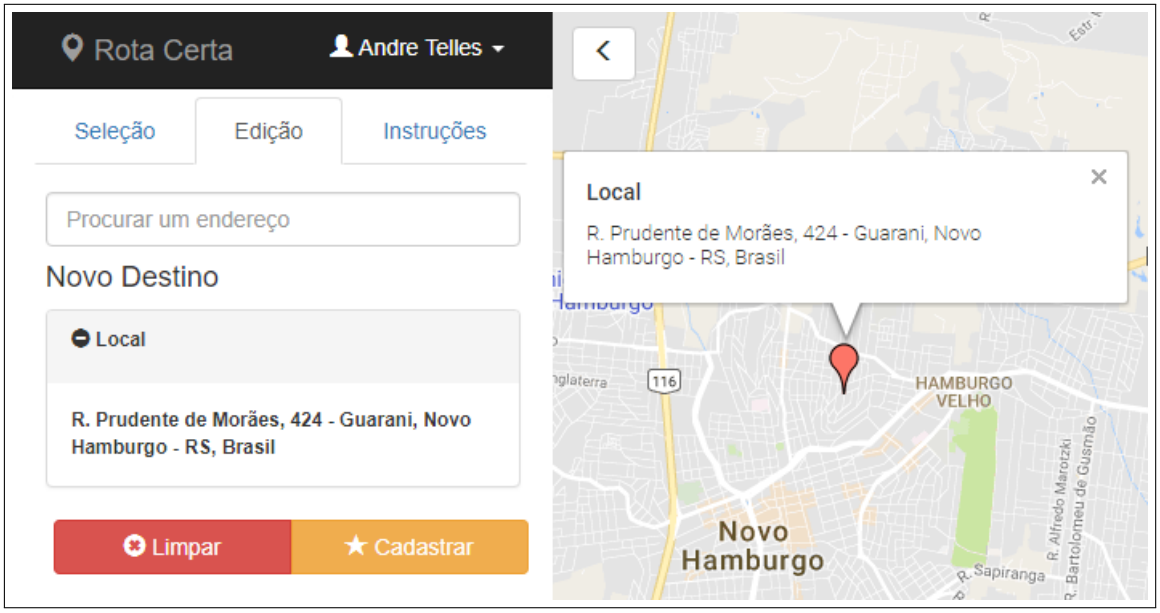
A segunda maneira, se dá através de uma função específica apresentada na tela inicial do software, na aba de seleção de atividade do menu lateral, apresentada na figura 5.9. Com ela, o usuário é levado ao estado de edição, onde pode localizar um endereço através das mesmas opções de inclusão de destinos apresentadas anteriormente (figura 5.10), salvando-o posteriormente.

5.1.5 Cálculo de rotas

A etapa final de criação de uma rota é o cálculo de seu traçado, ou seja, a definição de quais destinos serão visitados e em que ordem isso ocorrerá. Estas definições são o resultado dos cálculos que o sistema produz ao passo que renderiza sobre o mapa a

Figura 5.9: Cadastrar destino favorito

Figura 5.10: Selecionar e salvar um destino



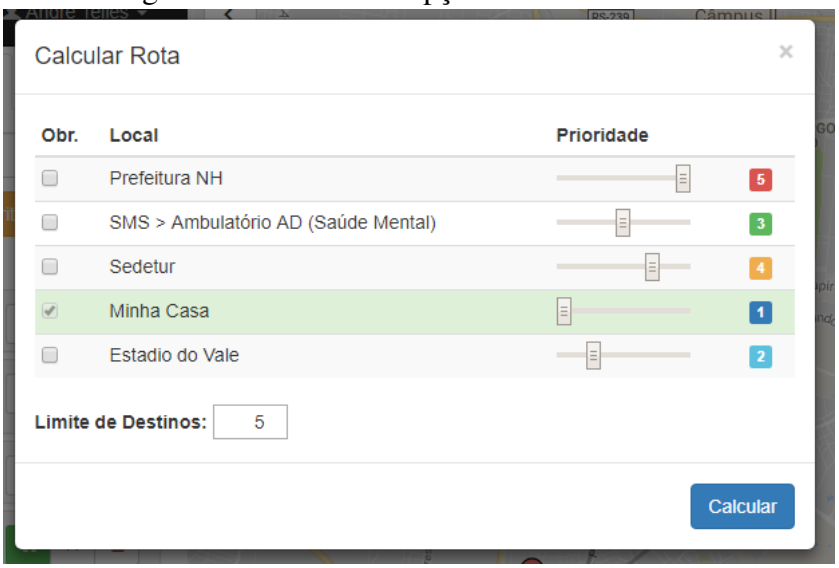
representação gráfica delas.

Pode-se realizar o cálculo da melhor rota sobre todos os destinos inseridos na listagem. Desta maneira o sistema executa o algoritmo para solucionar o Problema do Caixeiro Viajante, como explanado na seção 4.7.2, e para isso basta ao usuário não definir nenhuma restrição na janela de cálculo, (figura 5.11). Com a utilização deste algoritmo, o resultado será uma trajetória que envolve todos os destinos (figura 5.12).

5.1.5.1 Cálculo de rota com restrição

Outra opção que o usuário tem ao calcular uma rota envolve um problema de solução de prioridades. Após a inclusão de todos os destinos que necessitam ser visitados,

Figura 5.11: Janela de opções de cálculo de rota

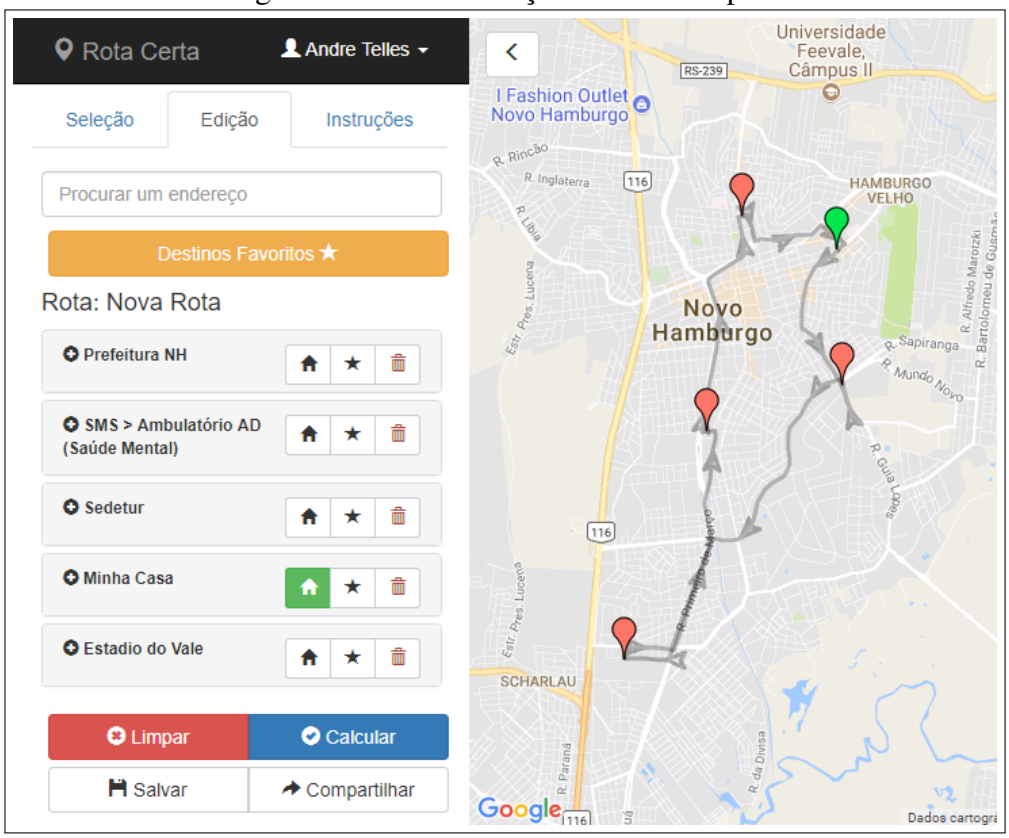


o usuário define uma restrição referente ao número máximo de locais que a rota pode ter e então a prioridade de cada um deles através de um controle deslizante, pode ser visto também na figura 5.11. A escala de prioridade pode variar de 1 (um) a 5 (cinco), onde 5 (cinco) é a prioridade máxima, informando ao sistema que este destino deve ser atendido antes dos demais com prioridade mais baixa.

Contudo, ao calcular uma rota com restrição de atendimento, o software realiza uma ponderação entre a distância a ser percorrida e a prioridade de atendimento de cada destino utilizando o algoritmo para solucionar o Problema do Caixeiro Viajante Coletor de Prêmios, como visto na seção 4.7.3. Usamos como exemplo a mesma rota da figura 5.12, porém com restrição ao atendimento de apenas quatro destinos. O resultado é uma rota que abrange um subconjunto dos destinos (figura 5.13), que não necessariamente serão os de maior prioridade, e sim os de melhor relação entre distância e prioridade.

Quando definidas restrições de atendimento, é possível informar ao sistema se a visita a um destino é obrigatória, através da seleção do respectivo item na janela de cálculo. Desta maneira, o destino é inserido no trajeto final não importando qual sua relação entre prioridade e distância. Por definição, o destino marcado como inicial é considerado obrigatório. Não é possível restringir o sistema, para atender menos destinos do que o total de destinos marcados como obrigatórios.

Figura 5.12: Renderização da rota completa



5.1.6 Gerenciamento de rotas

Assim que finalizada, uma rota deve possuir todos os seus elementos básicos definidos, que sejam, uma lista de destinos e um trajeto que os ligue da melhor forma possível. Com isso estabelecido, é esperado que se possa salvá-la para consulta posterior ou para compartilhá-la com outros usuários. Esta tarefa pode ser realizada através do respectivo botão Salvar na parte inferior da aba de edição de rota do menu lateral (como pode ser visto na figura 5.12).

Caso esta seja uma rota recém construída, será solicitado ao usuário informar um nome para que esta rota seja armazenada e identificada futuramente (figura 5.14).

Tendo presente a possibilidade salvar rotas, agora podemos apresentar uma das opções iniciais do sistema que havia sido deixada de lado. Na aba de seleção de atividades é possível realizar o carregamento de uma rota completa, trazendo consigo todos os destinos e trajeto como quando fora calculado. Ao selecionar a opção Rotas Salvas, o sistema mostra uma lista com todas as rotas pertencentes àquele usuário, (figura 5.15), e pode ser carregada ao utilizar-se a respectiva opção.

Ao realizar o carregamento de uma rota, pode-se acompanhar informações pro-

Figura 5.13: Renderização da rota com restrição de atendimento

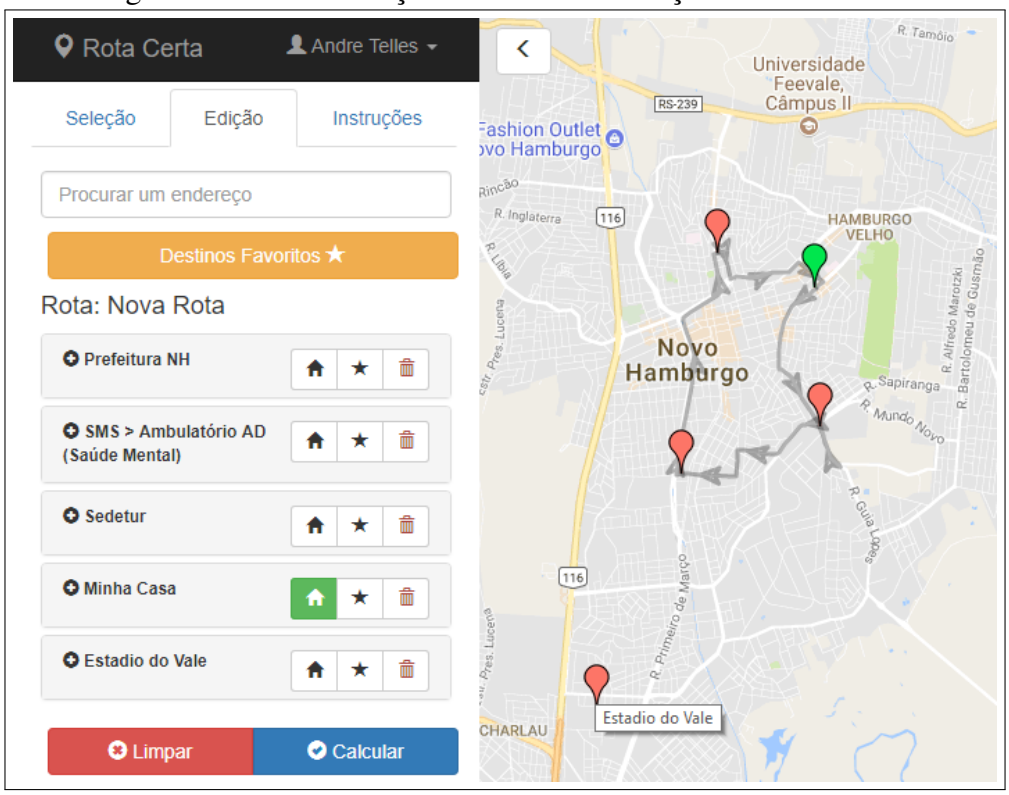
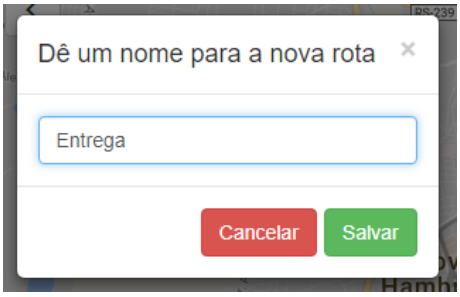


Figura 5.14: Caixa de diálogo de confirmação de salvar a rota

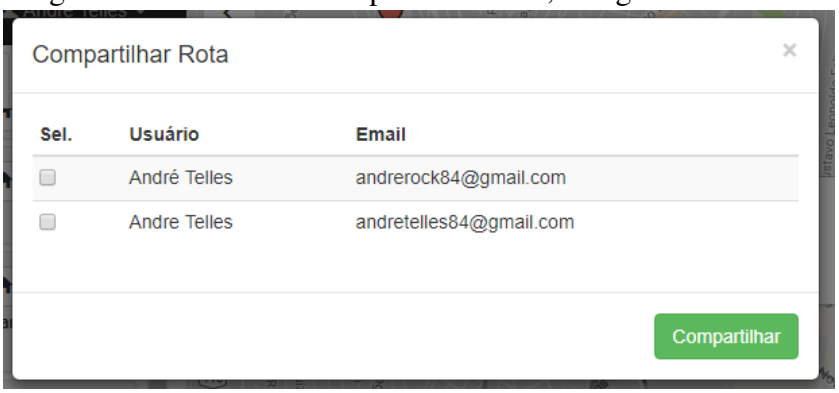


cedentes da utilização do aplicativo móvel, como a posição do motorista atribuído e os destinos por este visitados. Mais detalhes sobre como esses dados são gerados serão vistos na seção 5.2.3.

Após carregar uma rota, habilita-se a opção de compartilhamento que pode ser utilizada através do respectivo botão no fim da listagem lateral, pode ser visto também na figura 5.12. Esta função permite ao usuário dar acesso a outros usuários a uma rota criada e salva por ele. Ao ser selecionada esta opção mostra uma janela que lista os todos os usuários do sistema com quem é possível realizar o compartilhamento, como pode ser visto na figura 5.16.

Figura 5.15: Botão carregar rotas salvas e listagem de rotas

Figura 5.16: Janela de compartilhamento, listagem de usuários



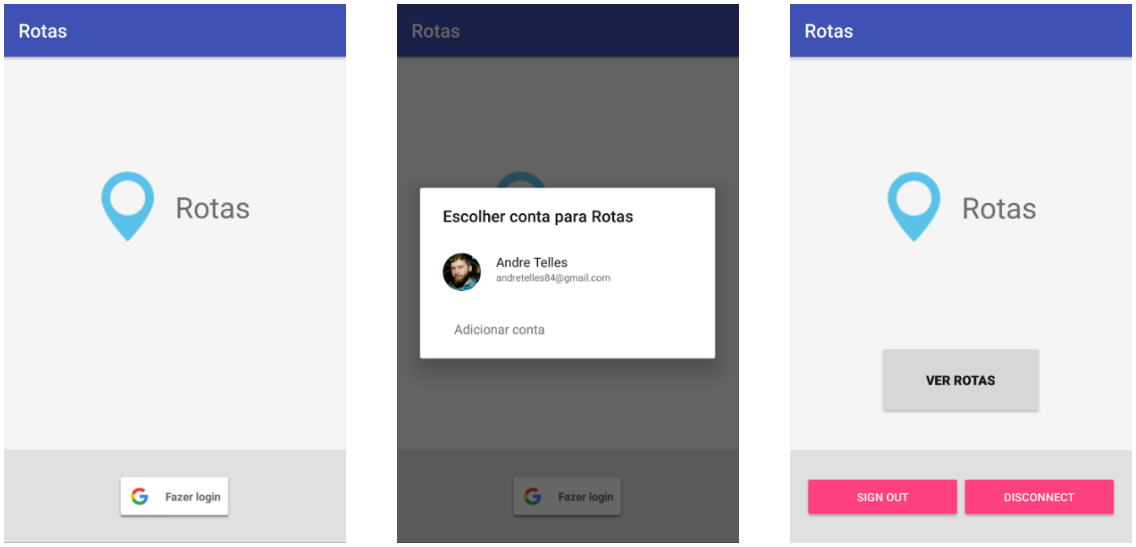
5.2 Aplicativo móvel Android

O aplicativo móvel do software Rota Certa tem funções mais limitadas que sua apresentação *web*, porém não menos importantes. Com o aplicativo é permitido ao usuário consultar sua lista de rotas salvas, acompanhá-las no mapa tendo sua posição como referência enquanto realiza seu deslocamento e fazer o registro da visita dos destinos conforme forem ocorrendo.

5.2.1 Autenticação

Diferentemente do que ocorre com a aplicação *web*, é obrigatório realizar a autenticação para utilizar o aplicativo Android do software. Isso se deve ao fato de que todas as funções disponíveis envolvem dados específicos de cada usuário, sendo portanto necessário realizar um controle do acesso a estes dados. Sendo assim, a tela inicial (figura 5.17) exige as credenciais do usuário para que as próximas funções sejam liberadas.

Figura 5.17: Transição da tela inicial ao realizar a autenticação



5.2.2 Carregar rota

Após autenticar-se, o usuário pode visualizar a listagem das rotas de sua autoria ou compartilhadas por outros usuários (figura 5.18). Estas rotas devem necessariamente serem criadas e salvas na aplicação Web conforme explicado na seção anterior 5.1.3, pois a função de criação de rotas não é suportada no aplicativo móvel.

Ao selecionar um item da lista, o usuário é levado ao mapa do aplicativo onde ocorre o carregamento da rota, ou seja, a inclusão dos marcadores referentes a cada destino e a renderização do trajeto calculado, como pode ser visto na figura 5.19.

5.2.3 Registro de visita

Uma vez que o usuário tenha carregado a rota selecionada, ele pode realizar o registro da visita de cada destino utilizando o botão localização de sua posição atual e de

Figura 5.18: Listagem de rotas Android

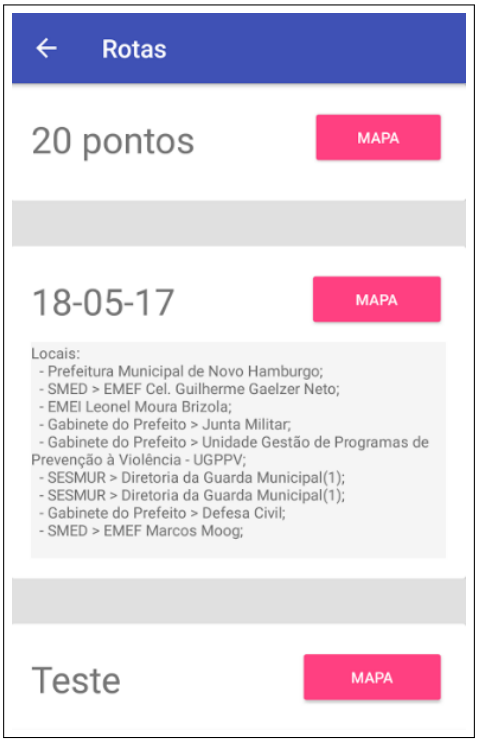
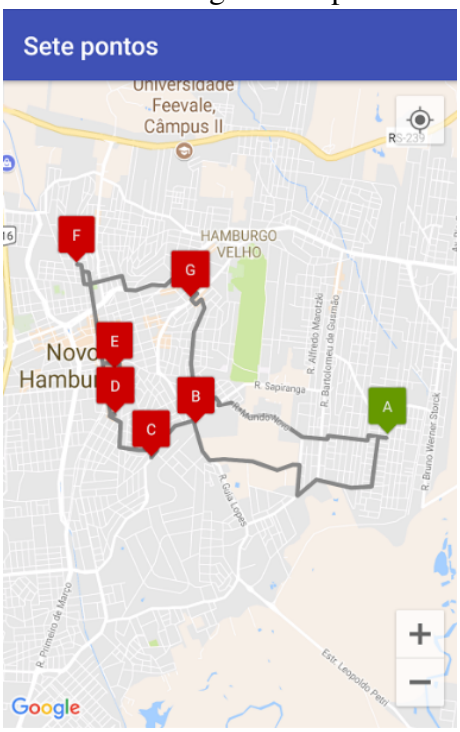


Figura 5.19: Rota carregada no aplicativo Android



centralização do mapa. O sistema então verifica a localização atual do usuário e confere qual o destino mais próximo da rota, se ambas localizações estiverem próximas o bastante é apresentada uma janela de confirmação do registro da visita (figura 5.20). Cada destino que tiver sua visita registrada, tem seu marcador alterado para a cor azul, diferenciando-se do padrão vermelho dos demais para facilitar a leitura (figura 5.21).

Figura 5.20: Janela de registro da visita ao destino

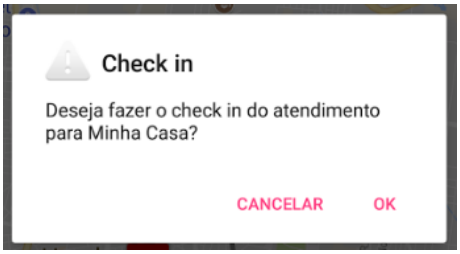
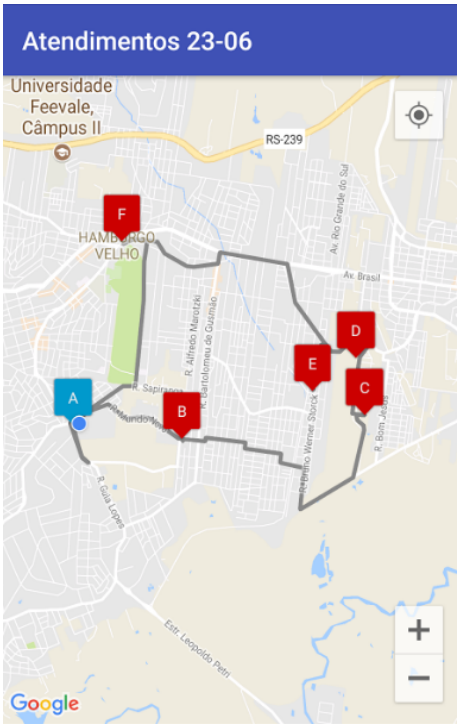


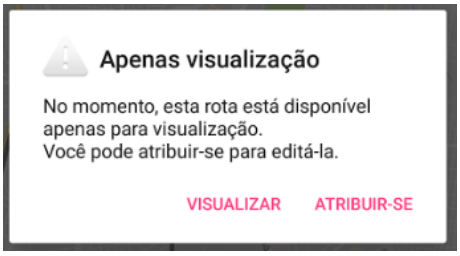
Figura 5.21: Rota carregada com registros de visitas



Entretanto, para evitar conflitos, o sistema de registros de visitas foi desenvolvido permitindo que apenas um usuário tenha permissões para realizá-los, independentemente se ser o criador da rota ou não, bastando ter acesso à rota através de compartilhamento. Assim, quando uma rota é carregada, verifica-se a existência de um usuário atribuído à função de registro, o qual nomeia-se motorista atribuído.

Caso ainda não exita um motorista atribuído para esta rota, é apresentada uma janela de confirmação para que o usuário possa delegar-se essa função (figura 5.22) e, a partir de então, realizar o registro de visitas dos destinos.

Figura 5.22: Janela de confirmação de atribuição da rota



Caso o motorista atribuído seja diferente do usuário atual, ou caso o usuário não queria atribuir-se tal tarefa, a rota é carregada em modo Apenas Visualização, onde é possível verificar todos seus detalhes e acompanhar o progresso das visitas, porém sem interferir em seu registro.

As informações de destinos visitados e de motorista atribuído, podem ser visualizadas também na aplicação Web, quando uma rota é carregada e ajudam no acompanhamento e gerenciamento das rotas.

6 AVALIAÇÃO E PESQUISA

Este capítulo tem a finalidade de realizar a avaliação do projeto Rota Certa em sua atual implementação e para isso foi dividido em duas partes. Na primeira etapa será realizada uma análise de desempenho da implementação do algoritmo 2-Opt na solução do problema do caixeiro viajante. Na outra etapa será apresentada uma pesquisa qualitativa aplicada junto aos usuários deste software.

6.1 Análise de desempenho

Nesta seção realizaremos uma análise de desempenho do algoritmo 2-Opt aplicado na solução do Problema do Caixeiro Viajante no software Rota Certa. Utilizando instâncias reais criadas na própria aplicação, será realizada uma comparação contra o algoritmo de solução por força bruta que, se por um lado garante a solução ótima, é conhecidamente o algoritmo de maior custo computacional.

6.1.1 Ambiente de testes

Para a realização da comparação, foram utilizadas duas máquinas com as seguintes configurações:

- **Máquina A**: Processador: Intel Core 2 Duo E7500 2,94 GHz; Memória RAM: 4GB; Sistema Operacional: Ubuntu 16.04 64bits;
- **Máquina B**: Processador: AMD Phenom(tm) II X4 B95 3,0 GHz; Memória RAM: 6GB; Sistema Operacional: Windows 10 Pro 64bits;

Em cada um das máquinas, foram realizados testes com instâncias de cinco (5), sete (7), dez (10), onze (11), doze (12), vinte (20) e quarenta (40) destinos diferentes. Para cada combinação possível entre máquina e instância, foram executadas três tomadas de tempo e então utilizada a média destas execuções na construção da tabela comparativa.

A tabela 6.1 mostra os resultados da aplicação dos testes, onde as colunas denominadas Distância mostram a distância total da melhor rota encontrada pelo respectivo algoritmo, em metros. As colunas Tempo são o tempo de execução de cada algoritmo e representadas em milissegundos. Já as colunas sob o título Relação mostram a relação

dos respectivos atributos entre o algoritmo de força bruta sobre o 2-Opt.

Tabela 6.1: Comparativo de desempenho: Força Bruta x 2-Opt

Instância	Máq.	***Força Bruta***		***2-Opt***		***Relação***	
		Dist. (m)	Tempo (ms)	Dist. (m)	Tempo (ms)	Dist.	Tempo
5 Destinos	A	8.440	1,00	8.440	1,00	1,0000	1,0000
	B	8.440	1,00	8.440	1,00	1,0000	1,0000
7 Destinos	A	10.742	6,00	10.742	8,00	1,0000	1,3330
	B	10.742	2,33	10.742	2,33	1,0000	0,9993
10 Destinos	A	36.630	1378,34	36.630	18,67	1,0000	0,0135
	B	36.630	678,34	36.630	9,67	1,0000	0,0142
11 Destinos	A	37.293	12092,33	39.140	45,00	1,0495	0,0037
	B	37.293	6102,33	39.140	11,67	1,0495	0,0019
12 Destinos	A	39.161	180505,33	39.831	45,67	1,0171	0,0003
	B	39.161	74118,33	39.831	17,00	1,0171	0,0002
20 Destinos	A	-	-	45.996	232,56	-	-
	B	-	-	45.996	149,99	-	-
40 Destinos	A	-	-	51.522	4505,72	-	-
	B	-	-	51.522	3148,99	-	-

6.1.2 Análise dos resultados

Com a ajuda da tabela comparativa 6.1, podemos verificar o ganho em performance possibilitado pela utilização do algoritmo 2-Opt para a solução do problema do caixeiro viajante.

Para instâncias realmente muito pequenas, com até sete destinos, percebemos que a utilização dos algoritmos se equivale. Porém a partir deste limite, o tempo de execução do algoritmo que utiliza força bruta cresce exponencialmente, chegando a três minutos de espera para a instância com doze destinos. Este fato inviabilizou testes de instâncias com maior número de endereços.

Enquanto isso, a utilização do algoritmo 2-Opt possibilita realizar cálculos em instâncias muitas vezes maior sem comprometer a experiência do usuário. Pode-se ver na mesma tabela que os resultados encontrados por este algoritmo não ultrapassaram um desvio maior que cinco por cento (5%) em relação a solução ótima encontrada pelo anterior, em execuções que chegam a ser até centenas de vezes mais rápidas.

6.2 Avaliação qualitativa

Nesta seção, será apresentada a pesquisa realizada com os usuários do software Rota Certa afim de obter dados qualitativos a respeito das principais características da aplicação, como apresentação da interface de usuário e validação das funcionalidades. A pesquisa foi aplicada através de um formulário produzido no Google Forms, diretamente com oito (8) usuários interessados na utilização do software, todos eles pertencentes a equipe do setor de suporte técnico em informática da prefeitura da Novo Hamburgo.

A avaliação foi dividida em três partes. Na primeira delas buscou-se identificar o perfil dos usuários com questões que mapeassem seus conhecimentos em informática, em logística, e sua experiência com sistemas baseados em mapas e de planejamento de rotas.

Na segunda etapa da pesquisa, os usuários foram instruídos a realizar tarefas que abrangessem as principais operações e funcionalidades do sistema Rota Certa. Pediu-se que estes tomassem nota de suas impressões durante a prática das tarefas.

Os usuário foram submetidos a um último questionário relativo a terceira etapa da pesquisa. Neste, foram indagados sobre as principais características do programa afim de que pudessem realizar um parecer final sobre a aplicação. Ao final, também foi aberto um espaço livre onde poderiam ser realizadas críticas e sugestões.

6.2.1 Identificação de Perfil

Na primeira parte da pesquisa coletamos informações afim de estabelecer o perfil dos usuários do sistema. Como os testes foram feitos em um ambiente controlado, ou seja, com funcionários do setor de suporte técnico da prefeitura de Novo Hamburgo, é natural supor que o resultado seja um perfil bastante homogêneo.

Analisando os gráficos da pesquisa, confirmamos nossa hipótese. Vemos que os usuários do sistema tem sua grande maioria entre 25 e 35 anos com alto conhecimento na área de tecnologia da informação, porém com conhecimento raso na área de roteirização veicular e logística.

A seguir notamos que os usuários tem experiência na utilização de sistemas web, estão adaptados ao uso de aplicativos móveis e já fizeram uso de programas baseados em mapas. Porém a grande maioria não fez uso de sistemas de planejamento de rotas.

Figura 6.1: Idade e área de atuação profissional dos pesquisados

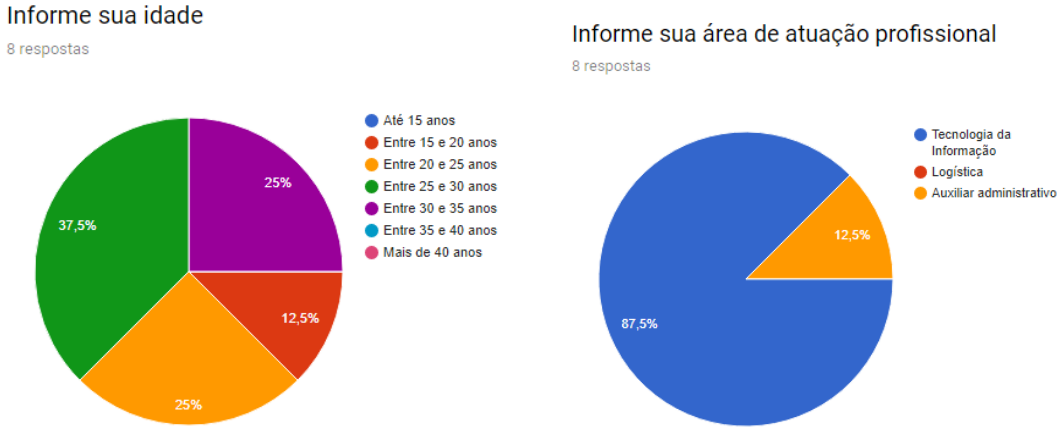
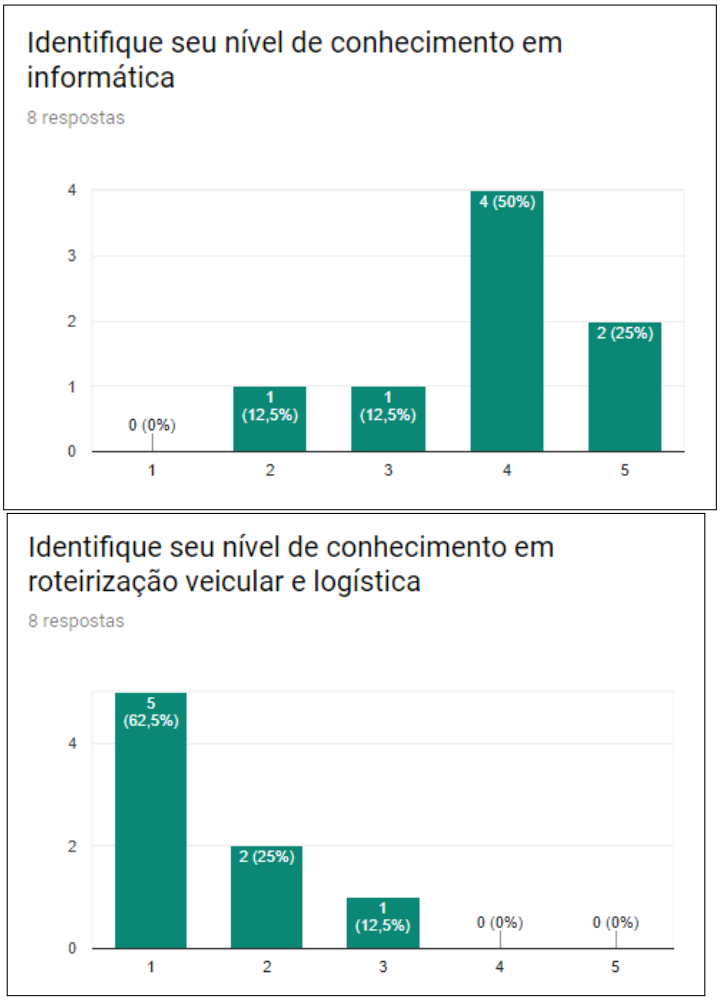


Figura 6.2: Nível de conhecimento em informática e em roteirização veicular dos pesquisados, em uma escala de *Básico* (1) até *Avançado* (5)



6.2.2 Tarefas

Na segunda etapa, foram apresentadas quatro tarefas a serem realizadas pelos usuários do sistema, cada uma delas prevendo a utilização de funções específicas e or-

Figura 6.3: Uso de aplicações web e aplicações móveis

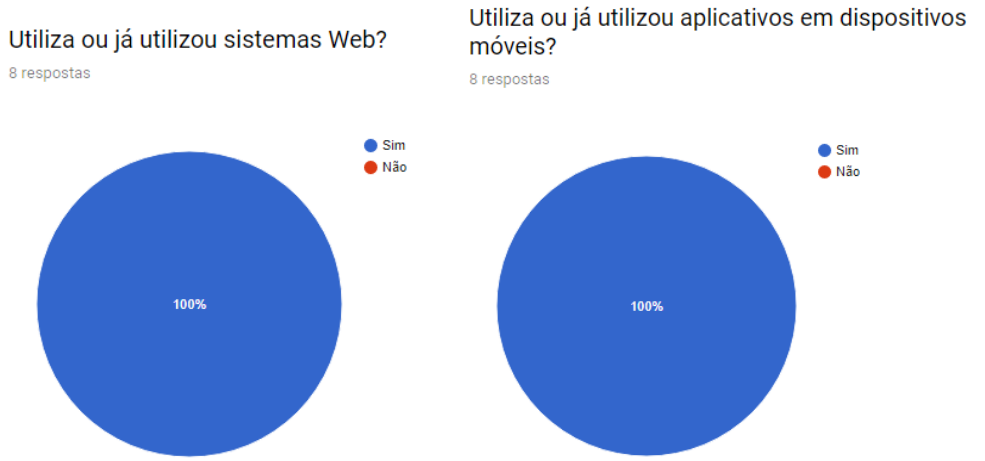
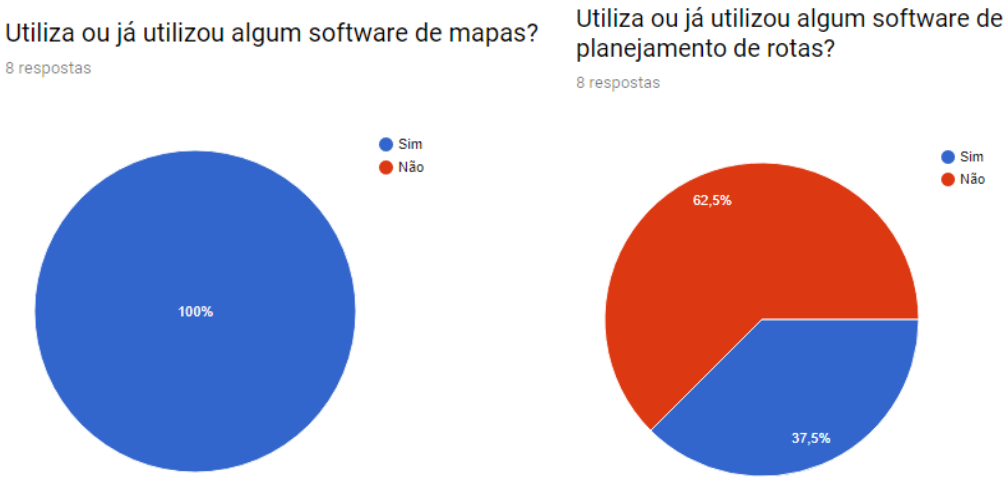


Figura 6.4: Uso de aplicações com mapas e aplicações de roteirização



denadas de maneira que ao final pudessem ter alcançado a totalidade das funcionalidades da aplicação.

- **Tarefa 1**: Utilizando a versão web do Rota Certa, realizar a autenticação no sistema e cadastrar ao menos cinco (5) destinos utilizando os diversos métodos apresentados pelo sistema.
- **Tarefa 2**: Também na versão web, construir uma nova rota carregando ao menos os destinos cadastrados na tarefa 1, utilizando as opções de edição de destinos, como a exclusão destes. Após isso, salvar a rota e compartilhá-la com outros usuários.
- **Tarefa 3**: Utilizando a versão móvel para Android, realizar sua autenticação, carregar a rota criada anteriormente na tarefa 2, atribuir-se como motorista desta e realizar o registro de suas visitas.
- **Tarefa 4**: Novamente no sistema web, carregar a rota percorrida na tarefa 3 e verificar as informações sobre a execução desta.

6.2.3 Índices qualitativos e avaliação

Após a realização das quatro tarefas, os usuários responderam um questionário referente a última parte da pesquisa. Neste, as questões eram direcionadas para ter um retorno sobre sua experiência ao utilizar o sistema Rota Certa, focando em atributos como usabilidade, utilidade e eficiência.

Podemos verificar que a interface de ambas as versões do programa foi aprovada por todos os usuários e este fato fica claro tanto quando vemos os gráficos referentes ao questionamento direto deste atributo (figuras 6.5 e 6.6), quanto pela análise do sucesso na execução de cada uma das quatro tarefas em seus respectivos diagramas (figuras 6.7, 6.8, 6.9 e 6.10).

Figura 6.5: Avaliação da interface Web, em uma escala de *Péssima* (1) até *Ótima* (5)

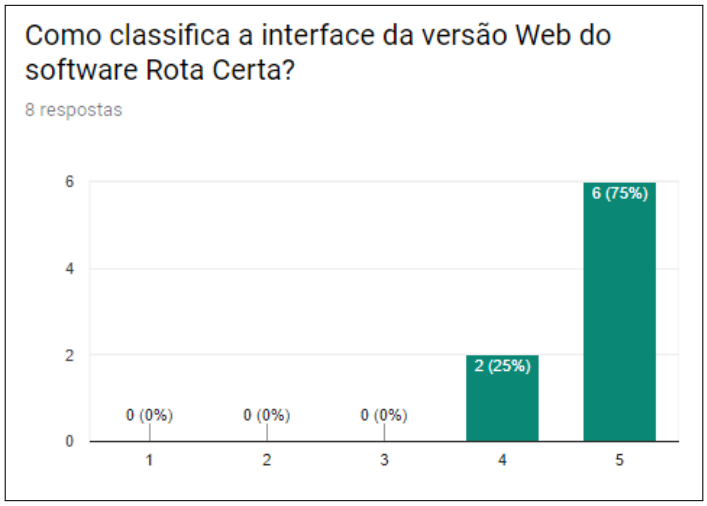
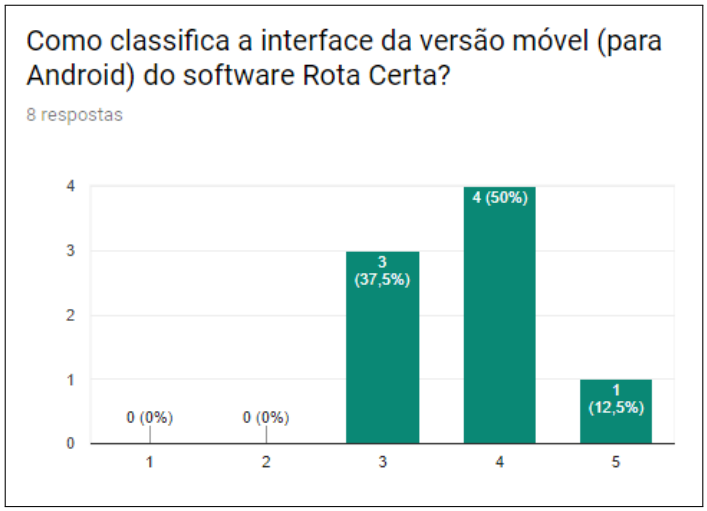


Figura 6.6: Avaliação da interface Android, em uma escala de *Péssima* (1) até *Ótima* (5)



Contudo, estes gráficos também revelam que tanto a interface do aplicativo móvel

Figura 6.7: Avaliação da facilidade de execução da Tarefa 1, em uma escala de *Com muita dificuldade* (1) até *Com muita facilidade* (5)

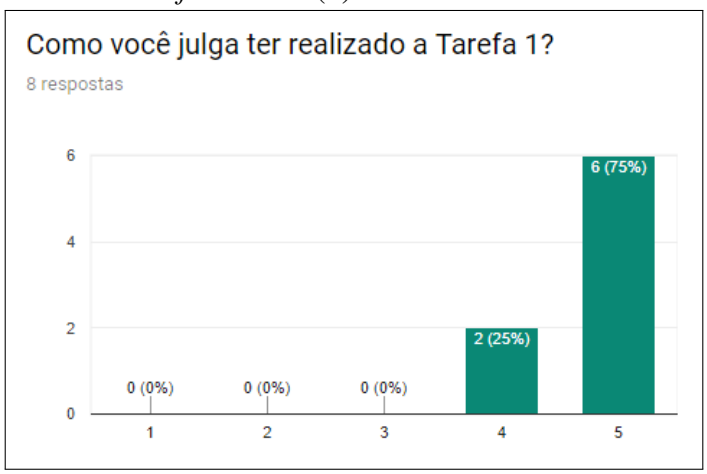


Figura 6.8: Avaliação da facilidade de execução da Tarefa 2, em uma escala de *Com muita dificuldade* (1) até *Com muita facilidade* (5)

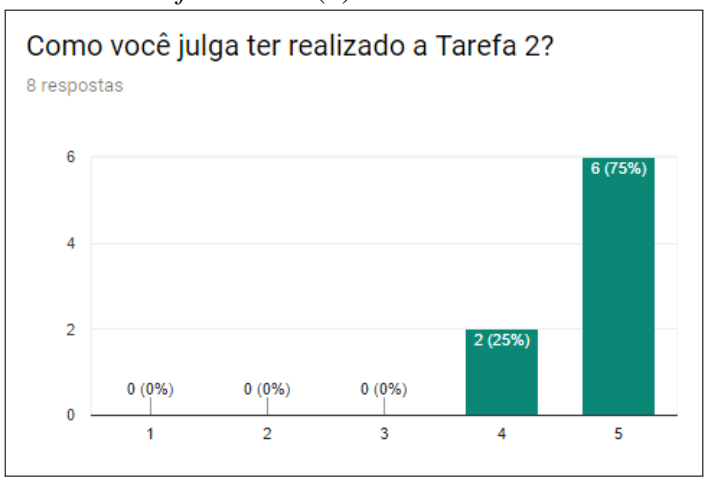
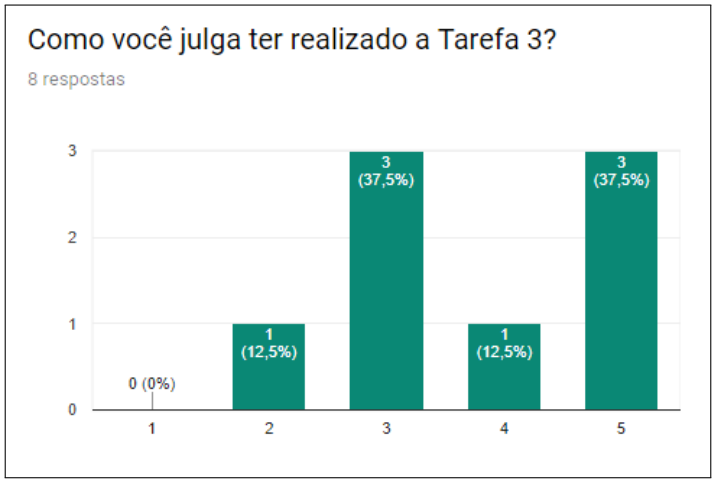
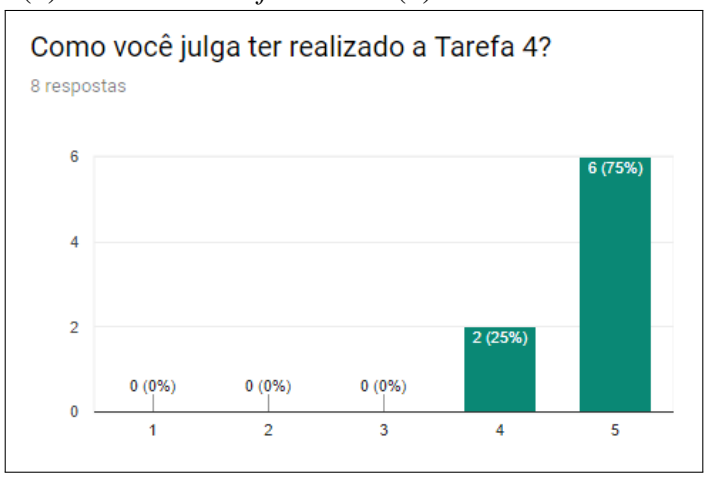


Figura 6.9: Avaliação da facilidade de execução da Tarefa 3, em uma escala de *Com muita dificuldade* (1) até *Com muita facilidade* (5)



quanto a Tarefa 3 associada a essa versão tiveram uma taxa menor de aprovação que as demais. Este fato se confirma ao verificarmos que esta versão foi alvo da maioria das

Figura 6.10: Avaliação da facilidade de execução da Tarefa 4, em uma escala de *Com muita dificuldade* (1) até *Com muita facilidade* (5)



críticas no espaço destinado para isso como: "Seria interessante adicionar as funções de criação e edição de rotas no aplicativo para celular.", "O aplicativo para Android deveria ter uma forma de ver os locais da rota depois de carregar ela e também poderia ter uma apresentação um pouco mais organizada"e por fim "A versão Android ainda carece de uma atenção no design e organização dos elementos, parece um protótipo incompleto".

Adicionalmente, devemos levar em consideração o fato de que todos os usuários de alguma maneira já tiveram contato com sistemas e aplicativos baseados em mapas, e neste quesito cabe apontar a grande popularidade dos softwares de navegação como Google Maps, Waze e equipamentos de posicionamento global (GPS). Desta maneira, pode-se traçar uma ligação com a facilidade na realização das demais tarefas.

Entretanto também verificamos que, apesar de julgarem o sistema necessário e reconhecerem sua importância (figura 6.11), poucos usuários utilizam, ou utilizaram em algum momento, outros sistemas cuja finalidade era a roteirização veicular ou o planejamento de rotas. Fato que pode ser explicado pela comum necessidade de realizar assinaturas ou pagamentos para a utilização destes sistemas, como visto no capítulo 3.

De maneira geral, o software foi aprovado com níveis de satisfação bastante altos (figura 6.13)e sua eficiência foi comprovada através da confirmação do cumprimento de seu objetivo (figura 6.12).

Entretanto, cabe ter atenção com as outras observações feitas pelos usuários que, dentre as mais relevantes, solicitaram a adição do cálculo das visitas através de janelas de tempo, a impressão da rota e das instruções de navegação e a possibilidade de inclusão de destinos a partir da leitura de arquivos em formato CSV, por exemplo.

Figura 6.11: Avaliação quanto a importância de um software de roteirização veicular, em uma escala de *Sem utilidade* (1) até *Muito útil* (5)

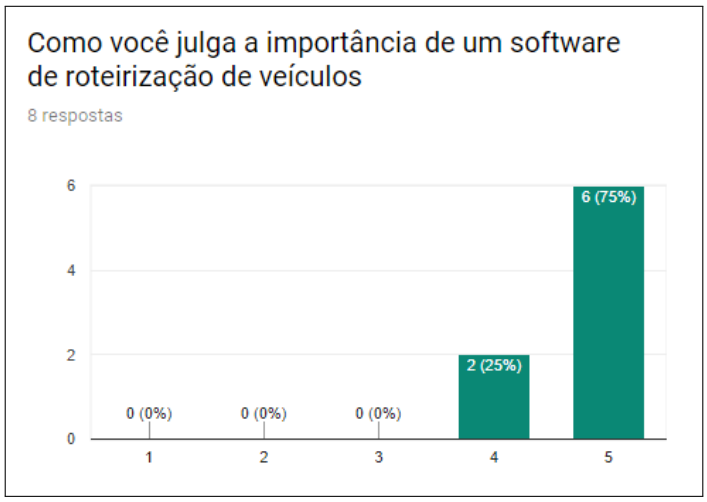


Figura 6.12: Avaliação quanto ao cumprimento do objetivo final do software, em uma escala de *Não cumpre* (1) até *Cumpre plenamente* (5)

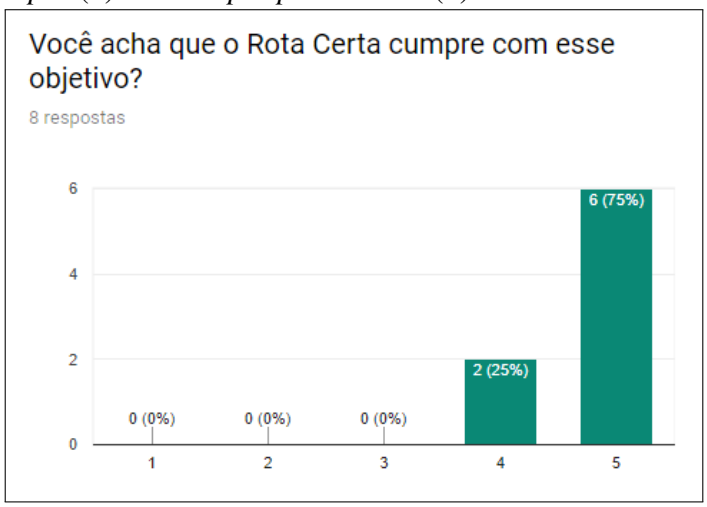
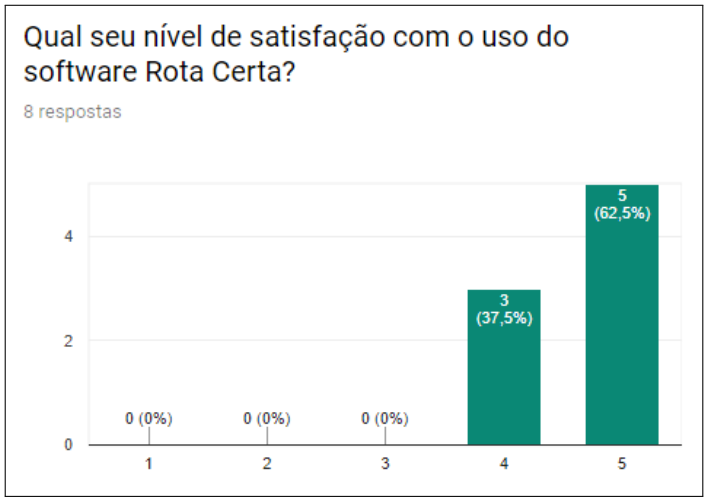


Figura 6.13: Avaliação quanto a satisfação dos usuários, em uma escala de *Plenamente insatisfeito* (1) até *Plenamente satisfeito* (5)



7 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou o software Rota Certa na tentativa de solucionar as principais dificuldades da roteirização de veículos através de um sistema computacional simples e de fácil usabilidade que permite realizar o planejamento, a construção, o gerenciamento e o acompanhamento de rotas em uma frota de veículos.

Ao longo deste trabalho foram apresentadas algumas das soluções atualmente disponíveis para o problema, apontando sobretudo suas limitações, como por exemplo a falta de regras para o tratamento da prioridade de visitação dos destinos. A construção do Rota Certa foi encaminhada a partir destas lacunas e, como pode ser visto na tabela de comparação com as demais 7.1, verificamos que os objetivos estabelecidos foram atingidos em sua completude. Juntamente a isso, a avaliação qualitativa aplicada junto aos usuários do sistema revelou que o sistema apresentou-se como uma excelente ferramenta.

Tabela 7.1: Comparativo de funcionalidades entre Rota Certa e demais aplicações

	Funcionalidades / Aplicação	*Route4Me*	*SimpliRoute*	*My Route Online*	*Rota Fácil*	*Track roads*	*Google Maps*	*Plot a Route*	*Rota Certa*
Distrib.	Versão gratuita			✓		✓	✓	✓	✓
	Versão paga	✓	✓	✓	✓	✓			
	Versão web	✓	✓	✓	✓	✓		✓	✓
	Versão móvel	✓	✓		✓	✓	✓	✓	✓
Constr.	Inclusão com o mouse	✓	✓	✓			✓	✓	✓
	Busca por endereço	✓	✓	✓	✓	✓	✓	✓	✓
	Busca por nome comercial	✓	✓	✓	✓		✓	✓	✓
	Ordenação de destinos	✓	✓	✓	✓	✓			✓
Armaz.	Destinos	✓			✓				✓
	Rotas	✓	✓	✓	✓		✓		✓
	Compartilhamento	✓	✓	✓	✓	✓	✓	✓	✓
Acomp.	Atribuição	✓	✓		✓	-			✓
	Rastreamento	✓	✓		✓	-			✓
	Auditoria	✓	✓		✓	-			✓
Cálculo	Prioridade	✓	✓						✓
	Tempo	✓	✓	✓	✓	✓			
	Distância	✓	✓	✓		✓			✓
	Subdivisão de rota	✓	✓	✓	✓	✓			
	Tempo de processamento	R	R	L	R	L			R

Legenda: R - Rápido / L - Lento

7.1 Aspectos Técnicos

As tecnologias adotadas para a construção deste trabalho apresentaram um resultado extremamente satisfatório, apesar de representarem um desafio ao autor, exigindo deste uma profunda pesquisa e dedicação para seu aprendizado.

Algumas das tenologias já eram familiares ao autor deste trabalho, e assim foram selecionadas afim de minimizar as dificuldades na execução do projeto, como por exemplo a opção por um ambiente *web* utilizando HTML e Javascript.

A utilização das API's dos serviços do Google Maps e Firebase, com certeza facilitaram a implementação deste projeto ao abstraírem muitas camadas da programação, dando suporte a geolocalização, geocodificação, armazenamento de dados e autenticação de usuários. Contudo, pode-se dizer que a curva de aprendizado de ambas ferramentas foi bastante difícil ao representarem novos paradigmas de programação e, principalmente, uma nova tecnologia em banco de dados. Como apresentado anteriormente, contornar as limitações oferecidas por essas ferramentas para usuários de suas versões gratuitas foi uma tarefa desgastante.

O contato com tecnologias de desenvolvimento de aplicativos para dispositivos móveis também foi algo que exigiu muita pesquisa e tem uma curva de aprendizado ainda mais complexa. Aprofundar-se no paradigma de orientação a objetos da linguagem de programação Java associado a manipulação dos leiautes e visualizações demanda abandonar os vícios da programação estruturada tradicional. Entretanto, esta tarefa foi facilitada com o auxílio da ferramenta Android Studio, que fornece suporte para programação voltada para o sistema operacional Android.

7.2 Trabalhos Futuros

Mesmo considerando que a construção do Rota Certa atingiu importante sucesso, existem outras funcionalidades que podem ser implementadas em versões futuras do software. Muitas das quais, sugeridas pelos próprios usuários do sistema.

A primeira funcionalidade que poderia ser adicionada ao software seria a leitura de arquivos, por exemplo planilhas no formato CSV, para a inclusão de dados no sistema, principalmente para a importação de endereços oriundos de outros sistemas e bases de dados legadas. Esta possibilidade já fora estudada pelo autor e poderia ser facilmente implementada utilizando alguma ferramenta auxiliar para realizar a conversão de dados.

A biblioteca Papaparse, disponibilizada em Javascript, poderia ser utilizada para tal finalidade, pois realiza a conversão de documentos CSV em objetos JSON (PAPAPARSE, 2017).

Outro possível aprimoramento seria incluir opções de cálculo com janelas de tempo para realização das visitas. Desta maneira, um usuário poderia definir o momento em que um determinado destino estaria disponível e a rota deveria se adequar a estas restrições. Contudo, esta funcionalidade descaracterizaria o problema solucionado neste trabalho, incluindo-o em outro caso especial do PCV, o problema do caixeiro viajante com coleta de prêmios e janelas de tempo e demandaria uma alteração profunda dos algoritmos utilizados.

Por fim, outra sugestão dos usuários poderia ser atendida ao realizar a migração das funções da aplicação *web*, como a criação de rotas e o cadastro de destinos, para o aplicativo móvel de maneira que o sistema pudesse ser gerenciado integralmente tanto pela página quanto através de dispositivos móveis. A realização desta tarefa depende mais do planejamento da interface do que propriamente da migração dos métodos e funções do sistema devido, principalmente, às limitações apresentadas pelo tamanho diminuto da tela destes dispositivos.

REFERÊNCIAS

BALAS, E. The prize collecting traveling salesman problem. **Networks**, Wiley Online Library, v. 19, n. 6, p. 621–636, 1989.

BIENSTOCK, D. et al. A note on the prize collecting traveling salesman problem. **Mathematical programming**, Springer, v. 59, n. 1, p. 413–420, 1993.

BOOTBOX. **Bootbox.js**. 2017. <http://bootboxjs.com>. [Online: Acessado 05 de Julho de 2017].

BOOTSTRAP. **Get Bootstrap**. 2017. <http://getbootstrap.com.br/>. [Online: Acessado 05 de Julho de 2017].

CROES, G. A. A method for solving traveling-salesman problems. **Operations research**, INFORMS, v. 6, n. 6, p. 791–812, 1958.

DANTZIG, G.; FULKERSON, R.; JOHNSON, S. Solution of a large-scale traveling-salesman problem. **Journal of the operations research society of America**, INFORMS, v. 2, n. 4, p. 393–410, 1954.

DICTIONARY, O. **Geolocation**. 2017. <https://en.oxforddictionaries.com/definition/geolocation>. [Online: Acessado 04 de Julho de 2017].

FIREBASE. **About Firebase**. 2017. <https://firebase.google.com/docs/>. [Online: Acessado 04 de Julho de 2017].

FIREBASE. **Estruturar seu banco de dados**. 2017. <https://firebase.google.com/docs/database/web/structure-data?hl=pt>. [Online: Acessado 04 de Julho de 2017].

GOOGLE. **Android**. 2017. <https://www.android.com/>. [Online: Acessado 05 de Julho de 2017].

GOOGLE. **Distance Matrix API**. 2017. <https://developers.google.com/maps/documentation/distance-matrix>. [Online: Acessado 04 de Julho de 2017].

GOOGLE. **Google Maps API**. 2017. <https://www.google.com/maps/about/>. [Online: Acessado 04 de Julho de 2017].

GOOGLE. **Google Maps API's - Eventos**. 2017. <https://developers.google.com/maps/documentation/javascript/events?hl=pt-br>. [Online: Acessado 04 de Julho de 2017].

GOOGLE. **Google Maps API's - O que é Geocodificação?** 2017. <https://developers.google.com/maps/documentation/geocoding/intro?hl=pt-br>. [Online: Acessado 04 de Julho de 2017].

GOOGLE. **Google Maps API's - Preenchimento automático para endereços e termos de pesquisa**. 2017. <https://developers.google.com/maps/documentation/javascript/places-autocomplete?hl=pt-br>. [Online: Acessado 04 de Julho de 2017].

GOOGLE. **Serviços Directions**. 2017. <https://developers.google.com/maps/documentation/javascript/directions?hl=pt-br>. [Online: Acessado 04 de Julho de 2017].

HELABS. **Metodologia Ágil - Scrum**. 2017. <http://www.desenvolvimentoagil.com.br/>. [Online: Acessado 05 de Julho de 2017].

JOHNSON, D. S.; MCGEOCH, L. A. The traveling salesman problem: A case study in local optimization. **Local search in combinatorial optimization**, Chichester, UK, v. 1, p. 215–310, 1997.

MOZILLA. **Geolocation**. 2017. <https://developer.mozilla.org/pt-BR/docs/Web/API/Navigator/geolocation>. [Online: Acessado 04 de Julho de 2017].

MOZILLA. **O que é o JavaScript**. 2017. <https://developer.mozilla.org/pt-PT/docs/Web/JavaScript/O_que_%C3%A9_o_JavaScript>. [Online: Acessado 05 de Julho de 2017].

PAPAPARSE. **Papaparse.js**. 2017. <http://papaparse.com/>. [Online: Acessado 07 de Julho de 2017].

PENA, R. A. **Mobilidade urbana no Brasil**. 2017. <http://brasilescola.uol.com.br/geografia/mobilidade-urbana-no-brasil.htm>. [Online: Acessado 07 de Julho de 2017].

RIBEIRO, W. E. et al. Algoritmos heuristicos para o prize collecting traveling salesman problem. [sn], 1997.

SCRIPTCASE. **A diferença entre um Site e uma Aplicação WEB**. 2017. <http://scriptcaseblog.com.br/diferenca-site-aplicacao-web/>. [Online: Acessado 05 de Julho de 2017].

TABLELESS. **O que é o JavaScript?** 2017. <http://tableless.github.io/iniciantes/manual/js/>. [Online: Acessado 05 de Julho de 2017].

TABLELESS. **O que é o JQuery?** 2017. <http://tableless.github.io/iniciantes/manual/js/o-que-jquery.html>. [Online: Acessado 05 de Julho de 2017].