Aplicativo para a criação e gerenciamento de rotas

Herick V. Carlsen

[1]Pontifícia Universidade Católica de Campinas (PUC-Campinas)
Campinas – SP – Brasil

[2]Faculdade de Engenharia da Computação – Pontifícia Universidade Católica
Campinas, Brasil.

herick.carlsen@gmail.com

Abstract. *This paper presents the development process of an app that creates and manages routes, in order to deal with complex routes and logistics. Additionally, this paper aims to point out problems which our application features intends to address. The application explores the popularity of smartphones and the performance of wireless networks that we have nowadays, allowing to help all users who need to optimize their routes and logistics of their business.*

Resumo. *Esse artigo apresenta o processo de desenvolvimento de um aplicativo para a criação e gerenciamento de rotas, com o objetivo de lidar com logísticas e rotas complexas. Além disso, esse artigo demonstra quais são os problemas que as funcionalidades do aplicativo têm a intenção de resolver. A aplicação explora a popularidade dos celulares e a performance da internet sem fio nos dias atuais, permitindo qualquer tipo de usuário que precise otimizar suas rotas e logísticas de seus negócios.*

1. Introdução

É natural no mundo corporativo reduzir os custos e aumentar os lucros, tudo que está envolvido com logística apresenta alto custo e, como Edinburgh diz em [EDINBURGH 2003], rotas de veículos não são uma exceção. Criar e gerenciar rotas otimizadas não é uma tarefa simples, gastando tempo e consequentemente dinheiro. Apesar disso, é indispensável para qualquer empresa que tenha a necessidade de uma boa logística de rotas e automatizar partes desse processo é muito útil para qualquer tipo de empresa, pequena ou grande.

Com a popularidade dos celulares, redes sem fio e computação em nuvem, tornou-se possível o desenvolvimento de um aplicativo democrático, o qual qualquer empresa tenha a condição de pagar. Baseado no uso do algoritmo de roteamento de veículos, arquitetura REST, Google Maps SDK e serviços da AWS, o aplicativo proposto tem duas principais funcionalidades: o usuário será capaz de inserir uma lista de endereços e uma lista de veículos, e obter como resposta rotas otimizadas para cada veículo, sem repetir nenhum dos endereços e o usuário será capaz de compartilhar rotas criadas e rastrear os veículos em tempo real. Por exemplo, um gerente de uma empresa de distribuição será capaz de criar rotas para cada um dos caminhões e rastreá-los em tempo real.

Nesse artigo, será definido como rota otimizada o caminho de menor distância. No desenvolvimento de futuras funcionalidades, existe o plano para considerar a rota de

menor tempo. Com isso será possível escolher qual o melhor tipo de rota otimizada para cada situação.

A organização deste artigo é definida da seguinte maneira: a Seção 2 discute os trabalhos relacionados e todas as suas semelhanças e diferenças. Na Seção 3, serão apresentadas todas as funcionalidades do aplicativo e quais são os problemas que elas têm a intenção de resolver. Na Seção 4, serão mostrados diagrama de fluxo de trabalho e diagramas de arquitetura explicando como as funcionalidades são integradas e por que foram desenvolvidas de tal forma. Na Seção 5 será discutido sobre o problema de roteamento de veículo e como o algoritmo funciona. A Seção 6 será sobre a conclusão, dificuldades, resultados e próximos passos.

2. Trabalhos Relacionados

Sempre que pensamos sobre rotas de veículos e aplicativo móveis, a comparação com o Waze e Google Maps é inevitável. Porém, o propósito desses dois aplicativos está relacionado à mobilidade, ou seja, ajudar pessoas comuns a se locomoverem. No Waze o seu foco está nos motoristas de carros, ajudando os usuários com direção, trânsito, sinalização de trânsito e outras informações úteis para motoristas. Google Maps seu foco relacionado à mobilidade urbana em geral, ajudando desde motoristas no trânsito até pedestres e usuários de transporte público.

O problema desses dois aplicativos é que ambos não são úteis em situações de negócio. Com esses aplicativos, os usuários não são capazes de criar rotas otimizadas, sendo então responsabilidade dos próprios usuários criar a logística de suas rotas. E isso pode ser um trabalho árduo, considerando o grande número de endereços que podem existir. Outro problema é que esses aplicativos não resolvem o problema de roteamento de veículo, que é basicamente criar rotas otimizadas considerando múltiplos veículos, e também não são capazes de fazer o rastreamento em tempo real de veículos. Um ponto interessante é que o aplicativo desenvolvido utiliza algumas APIs do Google Maps e, por conta disso, foi possível integrar com o aplicativo da Google sendo, então, possível aproveitar as vantagens das duas aplicações.

Existe um serviço web chamado MapLink que tem como propósito de resolver o problema do roteamento de veículo. Porém ele tem dois problemas: o primeiro é o fato de ser um serviço caro e o outro é o fato de não ser um aplicativo móvel.

Após comparações com os sistemas citados, é possível notar o diferencial do projeto: um aplicativo de baixo custo completamente móvel, no qual é possível criar rotas otimizadas com múltiplos veículos, compartilhar essas rotas e também fazer o rastreamento em tempo real desses veículos. Com isso, o aplicativo não só auxilia na criação das rotas de veículos como também ajuda na questão do gerenciamento da execução dessas rotas.

Não podemos esquecer de mencionar todos os estudos e pesquisas relacionados a otimização de rotas. Como o caminho Hamiltoniano formulado por W.R Hamilton em [Hamilton 1856], os trabalhos relacionados ao Problema do Caixeiro Viajante por H.Whitney e M.M.Flood nos artigos [Whitney 1935] e [Flood 1956] e principalmente, pelos trabalhos relacionados ao Problema do Roteamento de Veículo escrito por G.B.Dantzig, J.H.Ramser [Dantzig and Ramser 1959] e N.Christofides [Christofides 1976].

3. Funcionalidades

3.1. Cadastro

Os usuários devem criar uma conta válida para ter acesso ao aplicativo. Para isso, eles precisam ir na seção de cadastro, inserir nome de usuário, uma senha e um e-mail. Após isso, um código de validação será enviado ao e-mail e com isso será possível confirmar o cadastro. A tela de cadastro é apresentada na Figura 1.

3.2. Acesso

Após o sucesso na criação da conta, os usuários serão capaz de acessar seus respectivo perfis em qualquer celular Android e, com isso, ter acesso a todas as funcionalidades do aplicativo. A tela de acesso está presente na Figura 2.

3.3. Conexão de Usuários

Os usuários são capaz de se conectar entre si, porém isso só é possível entre usuários válidos. Na Figura 3 é possível verificar a tela de conexão de usuários, permitindo inserir outra conta de usuário válida.

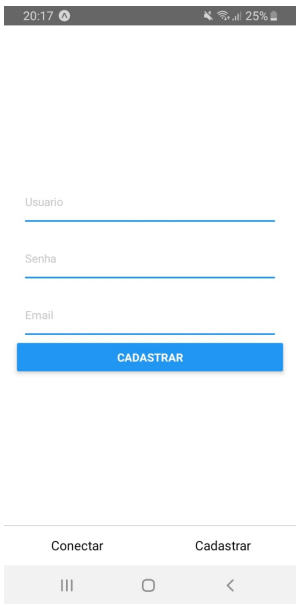

Figura 1. Tela Cadastro

Figura 2. Tela Acesso

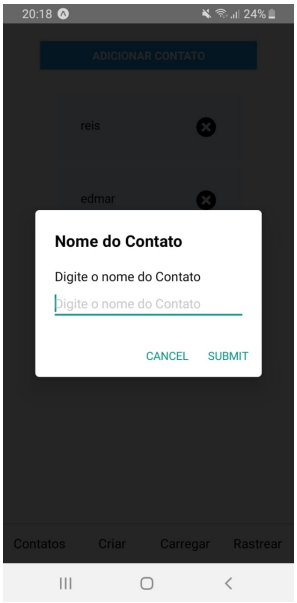

Figura 3. Tela Conexão de Usuários

3.4. Rota Simples

Os usuários devem inserir uma lista de endereços (Figura 4). O primeiro endereço dessa lista será nosso ponto de origem. Teremos como resposta a rota mais curta possível, sem esquecer ou repetir nenhum endereço da lista. Os usuários conseguiram ver suas rotas otimizadas graficamente na Figura 5 e importar para o aplicativo do Google Maps (Figura 6) em apenas um clique.

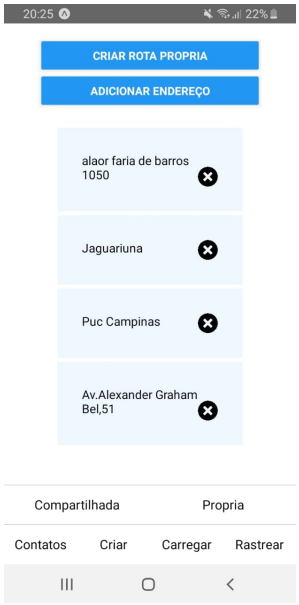

Figura 4. Tela Lista de Endereços (1)

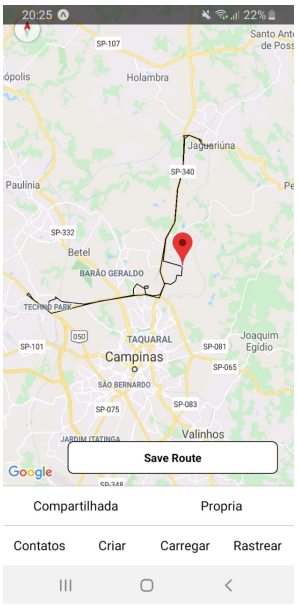

Figura 5. Tela Rota Simple

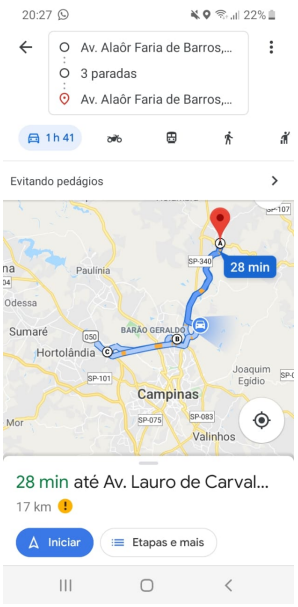

Figura 6. Tela Google Maps

3.5. Salvar Rotas

Após criar rotas com sucesso, os usuários serão capaz de salvar e importá-las para o Google Maps. Para realizar esse procedimento, os usuários precisam selecionar o botão ”Salvar Rota” presente na Figura 5 e escolher um nome para a rota.

3.6. Compartilhar Rota

Os usuários são capazes de compartilhar as rotas salvas com outros usuários conectados. Para essa funcionalidade, o usuário deve escolher uma rota salva e, em seguida, escolher um usuário conectado com o qual tenha a intenção de compartilhar. Após o compartilhamento com sucesso, o usuário escolhido deverá conseguir ver a nova rota na sua lista de rotas salvas.

3.7. Deletar Rotas

Os usuários são capaz de deletar todas as suas rotas criadas e compartilhadas. Como ambos os tipos de rotas estão salvos na mesma lista, apenas o dono das rotas tem permissão para deletá-las.

3.8. Rota Complexa

Semelhante a rota simples, os usuários são capaz de inserir uma lista de endereço (Figura 7) com o acréscimo de uma lista de usuários conectados (Figura 8). A rota obtida pelo algoritmo do problema de roteamento de veículo (Seção 5) será a rota otimizada para cada um dos usuários conectados sem esquecer ou repetir nenhuma dos endereços (Figura 9). Após isso, o usuário consegue compartilhar as rotas específicas para cada um dos usuários conectados em apenas um clique.

Figura 7. Tela Lista de Endereços (2)

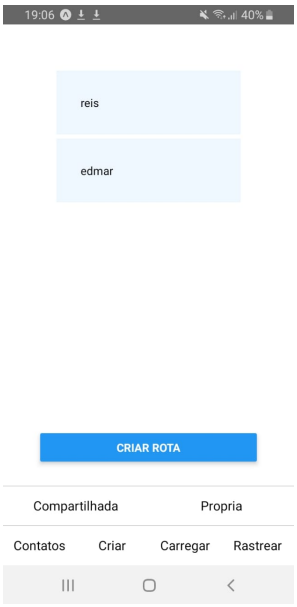

Figura 8. Tela Lista de Veículos

Figura 9. Tela Rota Otimizada

3.9. Compartilhamento e Rastreamento de Localização

Os usuários são capaz de compartilhar em tempo real a sua localização (Figura 10). Com isso, outros usuários conectados são capaz de rastrear todo o percurso da rota (Figura 11). No caso do usuário conectado desabilitar o compartilhamento de localização, o rastreamento é cancelado imediatamente (Figura 12).

Figura 10. Tela de Compartilhamento de Localização

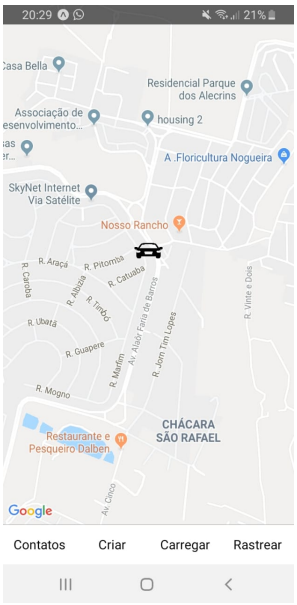

Figura 11. Tela de Rastreamento de Localização

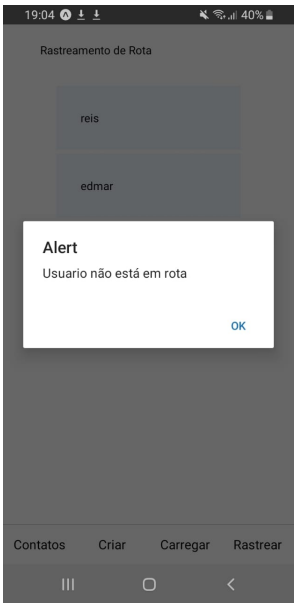

Figura 12. Tela de Rastreamento Cancelado

4. Arquitetura de Software

Neste diagrama (Figura 13) é possível ver como o sistema foi estruturado. Temos três blocos separados. O primeiro (App) é o lado do cliente, que consiste no aplicativo desenvolvido com react-native. Foi utilizada arquitetura REST e, dessa maneira, o aplicativo consegue enviar e receber informação dos outros dois blocos. O bloco dois é o servidor AWS EC2, que contém a API para resolver nossos problemas de otimização de rotas. O bloco três (SERVERLESS) contém API serverless desenvolvida em NODE.JS, que tem como objetivo resolver tudo relacionado ao banco de dados.

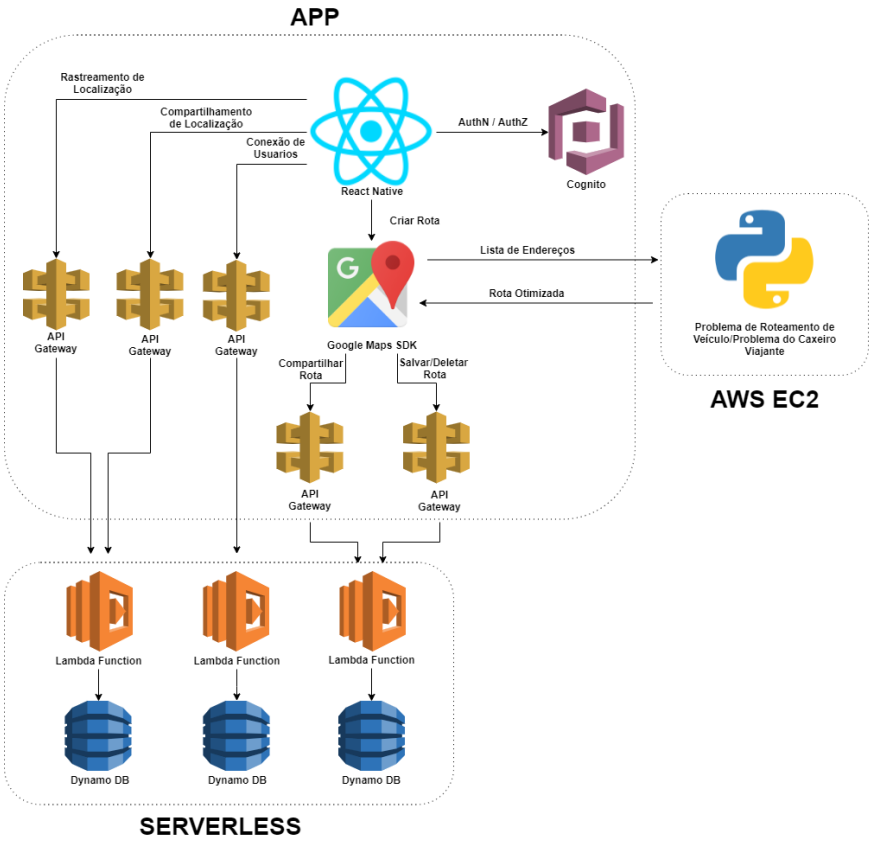

Figura 13. Diagrama de Arquitetura

Amazon Web Services (AWS) e Google Maps SDK são utilizados para o processo de desenvolvimento. Esses recursos contêm um conjunto de bibliotecas e serviços para computação em nuvem. No sistema são utilizado os seguinte serviços:

- React Native: uma biblioteca Javascript usada para desenvolver aplicativos Android e iOS;
- Google Maps SDK: lista de funcionalidades e APIs associadas a mapas, localização, endereços etc;
- Cognito: um serviço serverless usado para a autenticação e autorização de usuários;
- EC2: uma instância de máquina virtual utilizada para fazer a publicação da a API em python;
- DynamoDB: um banco de dados de documento serverless utilizado para armazenar os dados;

- Lambda: um serviço serverless utilizado para desenvolver as APIs em NODE.JS;
- API Gateway: um serviço serverless utilizado para permitir a comunicação entre o front-end e as funções lambda.

É importante explicar o conceito de serverless. São todos os serviços de computação em nuvem nos quais a infraestrutura é gerenciada pelo provedor do serviço (AWS) e não pelo desenvolvedor, permitido-os a apenas focar no desenvolvimento. Com serverless, nosso aplicativo se torna facilmente escalável, dando grandes possibilidades para o crescimento do sistema.

Como já foi explicadas todas as funcionalidades do sistema, como todo o projeto é estruturado e as tecnologias que foram utilizadas, é possível explicar e entender melhor como as funcionalidades interagem entre e si e por que elas foram desenvolvidas dessa maneira. Para auxiliar na explicação de como as funcionalidade foram desenvolvidas e a jornada do usuario, serão utilizados dois diagramas de sequencias presente nas Figuras 14 e 15.

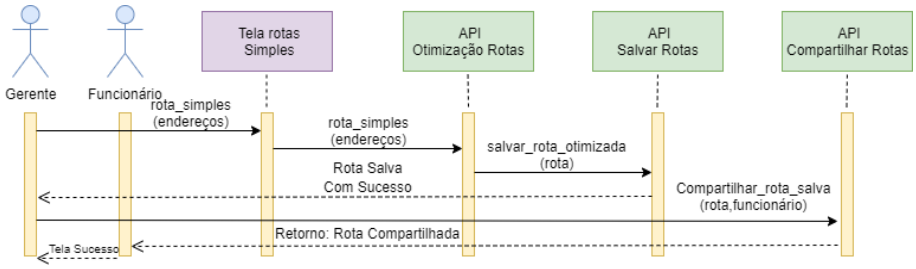

Figura 14. Diagrama de Sequência Rota Simples

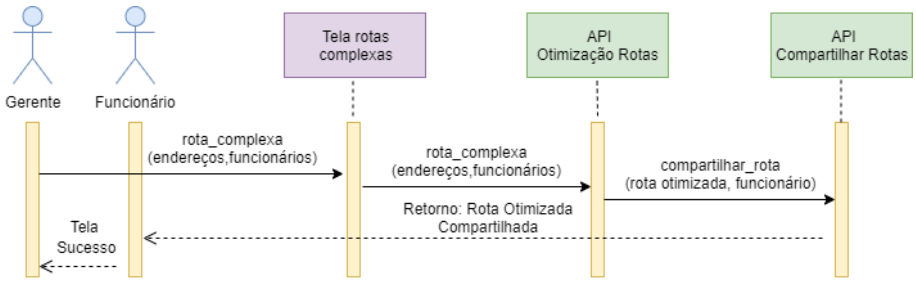

Figura 15. Diagrama de Sequência Rota Complexa

Apesar de não existirem diferenças de usuários no sistema, os diagramas de sequências apresentado contém dois tipos de diferentes (gerente e funcionário), com o intuito de criar um caso de uso hipotético e facilitar o entendimento de todo o fluxo do sistema.

Para ter acesso a todas as funcionalidades do aplicativo, os usuários precisam completar todo o processo da funcionalidade de cadastro. Utilizando o Cognito, foi possível desenvolver todo o serviço de autenticação de autorização de usuários de uma maneira segura e fácil. No próprio console do serviço, no site da Amazon, temos acesso a todos os usuários cadastrados, suas informações e as permissões de acesso que eles possuem no

sistema. Além disso, o serviço nos permite configurar métodos de confirmação de cadastro, no nosso caso, um código de confirmação por e-mail. Com o cadastro confirmado, o acesso do sistema se torna possível.

Como o projeto é um aplicativo de gerenciamento, os usuários devem conseguir conectar-se uns aos outros (um gerente deve conseguir conectar com os seus funcionários). Para essa funcionalidade, foi desenvolvida uma API utilizando os serviços Lambda Function, API Gateway e DynamoDB. Como é necessário garantir que ambos os usuários envolvidos sejam cadastrados e validados, utilizamos a lista de usuários cadastrados que temos acesso graças ao Cognito. Além disso, é preciso persistir a conexão dos usuários no nosso DynamoDB. Dessa maneira, todos os usuários cadastrados possuem uma lista de usuários conectados no nosso banco de dados. Para remover essa conexão, o processo é o mesmo, a lista de ambos os usuários é atualizada, sem a conexão.

Para ambos os tipos de rotas, Rotas Simples e Rota Complexa, existe uma instância na AWS EC2, que contém uma API em Python responsável por todos os algoritmos de otimização (tudo sobre o algoritmo será explicado na íntegra na próxima seção).

No caso da Rota Simples, após a rota ser otimizada, o usuário tem a opção de importar a rota para o Google Maps (utilizamos uma API da próprio Google para essa funcionalidade) ou salvar essa rota para uso futuro. Para salvar as rotas, foi desenvolvidas uma API nos mesmos moldes da conexão entre usuários, porém, nesse caso, existem duas listas no DynamoDB. Na primeira lista, contém os nomes das rotas e os endereços que as definem. A segunda contém uma lista com o nome de todas as rotas salvas referentes ao usuário (facilitando a exibição da lista de rotas salvas para os usuários no aplicativo). Para a funcionalidade de remover as rotas, uma Lambda Function remove a rota no banco de dados de ambas as listas vistas anteriormente. Com as rotas salvas, os usuários podem importá-las para o Google Maps ou podem compartilhar essas rotas com usuários na lista de usuários conectados.

No caso da Rota Complexa, além de inserir a lista de endereços, o usuário deve inserir uma lista de usuários conectados. Após a rota ser otimizada, o usuário terá uma lista de rotas otimizadas para cada um dos usuários escolhidos e, em seguida, as rotas serão compartilhadas em massa.

Para o compartilhamento de rotas, a API desenvolvida é muito semelhante à de salvar as rotas, com apenas duas exceções. A primeira é o fato de que a rota será persistida no banco de dado do outro usuário; a segunda é em relação ao nome da rota compartilhada, que consiste no nome do usuário que a compartilhou junto com um Id gerado pela função uuid, do próprio React Native. No caso do compartilhamento em massa que vimos no Problema do Roteamento de Veículo, o processo é o mesmo com a inclusão de um loop para cada rota otimizada.

Com a API de geolocalização do React-Native, usuários (funcionários) conseguem ter acesso às suas coordenadas atuais. Com o auxílio de uma API desenvolvida em Lambda Function, essas coordenadas são persistidas e atualizadas em um DynamoDB a cada 20 segundos, permitindo apenas usuários (gerentes) receber essa informação e desenhar a posição em tempo real no mapa no aplicativo. Quando o usuário (funcionário) cancela o compartilhamento de localização, uma Lambda Function é acessada e remove

as coordenadas no banco de dados, impossibilitando qualquer tipo de rastreamento indesejado.

5. Problema do Roteamento de Veículo

Agora que é possível entender quais tipos de problemas os algoritmos de otimização tem a intenção de resolver, podemos explicar como a API foi desenvolvida para servir às necessidades do sistema.

Como foi visto anteriormente, o sistema utiliza algumas APIs e bibliotecas do Google Maps. Usamos para criar mapas globais, desenhar rotas e outros objetos no aplicativo. Além disso, foram utilizados duas APIs, a primeira para traduzir um endereço em latitude e longitude, com essas coordenadas, é possível calcular a distância do trajeto entre dois pontos utilizando uma segunda API, ou seja, o cálculo é realizado utilizando valores reais do caminho e não de uma linha reta entre os dois pontos. Com isso, foi possível criar uma matriz de distância, que representa nosso grafo. Vértices são os endereços e as arestas serão as distância entre os pontos.

Com uma lista de veículos, o algoritmo cria um modelo contendo a nossa matriz de distância, números de veículos e o ponto de origem. Com isso foi utilizado uma biblioteca chamada OR-Tool, que tem o propósito de ajudar a executar os algoritmos de otimização baseados no nosso modelo, criando e indexando cada rota para cada veículo. O pseudo código do M.A Mohammed [Mohammed et al. 2017], foi utilizado para ser utilizado na nossa API e pode ser visto abaixo na Figura 16.

Algorithm 1: The VRP model

1. begin;
2. generate the *InitialSolution* (group of routs) with applying a nearest rule between nodes i.e.:
   rout 1: $12 \rightarrow 16 \rightarrow 30 \rightarrow 1 \rightarrow 26 \rightarrow 7 \rightarrow 13 \rightarrow 15$,
   rout 2: $23 \rightarrow 3 \rightarrow 28 \rightarrow 8 \rightarrow 22$,
   rout 3: $11 \rightarrow 4 \rightarrow 18 \rightarrow 2 \rightarrow 9 \rightarrow 24$,
   rout 4: $10 \rightarrow 19 \rightarrow 17 \rightarrow 31 \rightarrow 21 \rightarrow 6 \rightarrow 14 \rightarrow 20$,
   rout 5: $25 \rightarrow 5 \rightarrow 29 \rightarrow 27$;

//d denote to distance, q denote to demands, and s is the concerned solution.

3. calculate the fitness of *InitialSolution* as follow:
   $d_s = d_{Route_1} + d_{Route_2} + \cdots + d_{Route_N}$,
   $q_s = q_{Route_1} + q_{Route_2} + \cdots + q_{Route_N}$,
   $d_{Route_i} = d_{0,1} + d_{1,2} + \cdots + d_{n-1,n}$,
   $q_{Route_i} = q_1 + q_2 + \cdots + q_n$;

//Fitness_s is the fitness of solution

4. $Fitness_s = d_s + q_s$;
5. set best solution (*BestSol*) = *InitialSolution*;
6. loop for 10,000 iterations, in each iteration:
   6.1 generate new random solution (*NewSol*) with applying a nearest rule between nodes;
   6.2 calculate the fitness of a *NewSol*;
   6.3 if *NewSol* is better than saved *BestSol*, then *BestSol* = *NewSol*;
7. end of loop;
8. checking *BestSol* feasibility;
9. write out the path of each route stored inside the *BestSol*;
10. end;

Figura 16. Problema do Roteamento de Veículo

O primeiro passo para nosso algoritmo, é calcular uma solução inicial baseada

na regra do vizinho mais próximo, dessa maneira, vamos ter uma rota inicial para todos os nossos veículos. O segundo passo, será calcular a distância total que será percorrida pela nossa frota. Com a solução inicial obtida, utilizando a Busca Tabu, o algoritmo tenta encontrar novas soluções melhores que a inicial. Por fim, será verificado se a solução encontrada é realmente uma solução viável (verificar se não existem rotas com endereços repetidos ou endereços ausentes).

5.1. Busca Tabu

Busca Tabu é um algoritmo matemático de otimização criado por M.Laguna [Laguna et al. 1991] e está presente no nosso algoritmo do problema de roteamento de veículo. Busca Tabu é um procedimento auxiliar que, a partir de uma solução inicial, tenta encontrar outra e melhor solução até que um determinado critério de parada seja atendido. Resumindo, baseado na regra do vizinho mais próximo, o algoritmo através de uma espécie de força bruta tenta encontrar a melhor solução possível. Nosso critério de parada é quando o algoritmo atinge um número constante de interações sem nenhum ganho relativo ou 10.000 interações para evitar muita latência para dar alguma resposta ao usuário.

5.2. Otimização de rotas utilizando o aplicativo

Considerando um exemplo simples, onde um usuário necessita criar uma rota considerando os seguintes lista de endereços: Campinas (nosso ponto de origem e de chegada), São Paulo, Jaguariúna e Minas Gerais. Como podemos ver na tabela abaixo (Tabela1), temos todas as rotas possíveis considerando os endereços propostos (foi utilizado o Google Maps para calcular as distâncias) e o trajeto resultante de nossa API de otimização de rotas Figura 17.

Tabela 1. Rotas possíveis

Rotas	Distância total
São Paulo - Jaguariuna - Minas Gerais	1.726 km
São Paulo - Minas Gerais - Jaguariuna	1.687 km
Jaguariuna - Minas Gerais - São Paulo	1.635 km
Jaguariuna - São Paulo - Minas Gerais	1.712 km
Minas Gerais - Jaguariuna - São Paulo	1.719 km
Minas Gerais - São Paulo - Jaguariuna	1.709 km


```
{"Distance_of_the_route":[1634143],"Maximum_of_the_route_distances":1634143,"Route":
[["campinas","jaguariuna","minas+gerais","sao+paulo","campinas"]],"Vehicle":[0]}
```


Figura 17. Retorno API de Otimização

É possível ver que a diferença entre a pior e a melhor solução é de 91 km, lembrando que nesse exemplo estamos considerando apenas um exemplo simples, com poucos endereços e apenas um veículo. Em alguns casos, muitas possibilidades de rotas, fazem com que essa análise para encontrar a melhor rota inviável, fazendo com que automatizar esse processo uma tarefa fundamental.

5.3. Benefícios de softwares para otimização de rotas

Como o algoritmo do problema de roteamento de veículo é antigo, conseguimos coletar muitos dados e pesquisas sobre o assunto. Considerando isso, é possível visualizar os principais benefícios que o algoritmo consegue nos oferecer, como é presente nos artigos escritos por Silva Melo [Silva Melo and Ferreira Filho 2001] e César Brandão de Oliveira[César Brandão de Oliveira 2007]:

- O algoritmo consegue reduzir em 25% a frota de carros;
- O algoritmo consegue reduzir em 20% o custo de entregas;
- O algoritmo consegue aumentar o número de entregas por mês;
- O algoritmo consegue aumentar a ocupação da frota;
- O algoritmo consegue reduzir o consumo de gasolina e distância viajada;
- O algoritmo evita a duplicação de entregas;
- O algoritmo consegue reduzir em 10% - 15% custo final dos produtos.

6. Conclusão

Esse artigo apresenta um aplicativo para a criação e gerenciamento de rotas de veículos. A princípio, o objetivo era apenas focado na criação das rotas otimizadas e como algoritmos antigos ainda conseguem resolver muitos problemas modernos. A ideia inicialmente foi proposta por um gerente de vendas e, após algumas conversas, foi descoberto o quão útil pode ser um aplicativo que ajude no planejamento de rotas. Além disso, foram descobertas outras necessidades de um gerente de vendas, sendo uma delas uma maneira de ter controle e instruir vendedores da empresa. E foi assim que toda a ideia do projeto nasceu, onde é permitido a um usuário criar e gerenciar suas rotas.

Foi necessário muita atenção e cuidado no planejamento e desenvolvimento das funcionalidades. Além de serem muitas, precisam ser integradas e funcionar em harmonia para uma boa experiência de usuário. Como é um sistema móvel, os usuários não possuem acesso à mouses e teclados e por isso dependem muito da fluidez do aplicativo. Algumas funcionalidades, como otimização de rotas e rastreamento em tempo real, possuem um alto nível de complexidade, necessitando de muito estudo para o desenvolvimento ocorrer de uma maneira satisfatória.

Existe todo um mundo de possibilidades para futuros desenvolvimentos. O primeiro, como já foi explicado, é a possibilidade do usuário escolher qual tipo de rotas otimizadas ele quer, sendo elas o caminho mais curto ou tempo mais curto. Ainda em rotas otimizadas, usuários devem conseguir criar determinadas prioridades entre os endereços, por exemplo, um veículo deve passar apenas no endereço B após passar pelo endereço A.

Uma melhoria importante em relação ao rastreamento em tempo real, é a possibilidade do usuário (gerente) conseguir ver além da localização do usuário (funcionário) a rota que ele deve seguir. E em caso de desvios de rota, gerar alertas para facilitar o monitoramento e gerenciamento.

Existem melhorias realizadas nos algoritmos do problema do roteamento de veículo, como diz B.L. Golden [Golden et al. 2008]. Alguma dessas melhorias permitem que o algoritmo crie rotas otimizadas utilizando múltiplos pontos de origem, capacidade máxima ou até mesmo tempo de espera por veículo. Devemos ficar atento às necessidades

do Mercado, fazendo novas pesquisas e melhorando o aplicativo com novas funcionalidades e serviços.

Como já foi explicado foi utilizado react-native para o desenvolvimento do aplicativo móvel. Apesar da ferramenta ser usada tanto para ambiente Android como iOS, ainda é necessário fazer refinamento e testes no ambiente iOS. A aplicação tem a intenção de ser acessível para todos os tipos de deficiência.

Referências

César Brandão de Oliveira, H. (2007). Um modelo híbrido estocástico para tratamento do problema de roteamento de veículos com janela de tempo. Master's thesis, Universidade Federal de Pernambuco.

Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group.

Dantzig, G. B. and Ramser, J. H. (1959). The truck dispatching problem. *Management science*, 6(1):80–91.

EDINBURGH, U. (2003). The effects of transport investment on logistical efficiency.

Flood, M. M. (1956). The traveling-salesman problem. *Operations research*, 4(1):61–75.

Golden, B. L., Raghavan, S., and Wasil, E. A. (2008). *The vehicle routing problem: latest advances and new challenges*, volume 43. Springer Science & Business Media.

Hamilton, W. R. (1856). Memorandum respecting a new system of roots of unity. *Philosophical Magazine*, 12(446):1856.

Laguna, M., Barnes, J. W., and Glover, F. W. (1991). Tabu search methods for a single machine scheduling problem. *Journal of Intelligent Manufacturing*, 2(2):63–73.

Mohammed, M. A., Ghani, M. K. A., Hamed, R. I., Mostafa, S. A., Ahmad, M. S., and Ibrahim, D. A. (2017). Solving vehicle routing problem by using improved genetic algorithm for optimal solution. *Journal of computational science*, 21:255–262.

Silva Melo, A. C. d. and Ferreira Filho, V. J. M. (2001). Sistemas de roteirização e programação de veículos. *Pesquisa operacional*, 21(2):223–232.

Whitney, H. (1935). T. johns hopkins. *American Journal of Mathematics*, 57(3):509–533.