Curso de Sistema de Informação
Universidade Estadual de Mato Grosso do Sul

FlexMap: Aplicativo baseado na API Google Maps para permitir flexibilidade na definição das rotas.

Sidnei Ferreira da Silva

Dr. Evandro Cesar Bracht (orientador)

Dourados – MS
2019

Curso de Sistema de Informação
Universidade Estadual de Mato Grosso do Sul

FlexMap: Aplicativo baseado na API Google Maps para permitir flexibilidade na definição das rotas.

Sidnei Ferreira da Silva

Outubro de 2019

Banca Examinadora:

Prof. Dr. Evandro Cesar Bracht (Orientador)
Área de Computação – UEMS

Prof. Dr. Ricardo Luís Lachi
Área de Computação – UEMS

Prof. Dr. Cleber Valgas Gomes Mira
Área de Computação – UEMS

FlexMap: Aplicativo baseado na API Google Maps para permitir flexibilidade na definição das rotas.

Sidnei Ferreira da Silva

Este exemplar corresponde à redação final da monografia da disciplina Projeto Final de Curso, devidamente corrigida e defendida por Sidnei Ferreira da Silva e aprovada pela Banca Examinadora, como parte dos requisitos para obtenção do título de Bacharel em Sistema da Informação.

Dourados, 15 de Outubro de 2019.

Dr. Evandro Cesar Bracht (orientador)

Esse trabalho dedico a meus pais, meus maiores incentivadores.

Dedico esse trabalho à professora Maria de Fátima Oliveira Mattos Grassi (Fatinha) (in memorian).

“Na busca de um objetivo, profissional ou pessoal, a jornada se entrelaça por vários caminhos e nesse contexto tomar decisões é necessário. E o conhecimento é a chave para as melhores decisões”.

AGRADECIMENTO

Agradeço a Deus que me permitiu chegar a esse momento, com muito esforço e dedicação, mesmo nos momentos em que me faltava coragem de seguir.

Aos meus pais Maria Aparecida e Cicero Ferreira que me apoiaram a todo momento aos meus irmãos Sérgio, Fagner, Marcelo e Rosineide por me desafiarem a concluir o curso e tentar me distrair nessa jornada, sem eles talvez tivesse desistido.

Aos meus amados sobrinhos Gustavo, Guilherme e Miguel por sempre quererem jogar no meu computador na hora em que eu queria estudar.

Ao amor da minha vida Thais Fernanda de Lima, minha esposa por me apoiar em todos os meus projetos e ter muita paciência e entender quanto eu tinha que estudar.

A minha maior joia minha filha amada Ana Júlia que foi o fator importante para terminar esse projeto.

Aos meus amigos que tiveram paciência em entender que o chopp, a cervejinha, o churrasco e festas do final de semana tinham que esperar.

Aos meus professores pela persistência em tentar me ensinar algo durante o curso, e as pessoas que tiveram direta ou indiretamente envolvido na construção do meu conhecimento.

Ao professor Evandro Cesar Bracht pela paciência e persistência, em ajudar, orientar e estruturar a melhor proposta para este trabalho e com seu conhecimento otimizar as ideias.

Aos professores presente na banca avaliadora Cleber Valgas Mira Gomes e Ricardo Luís Lachi, por contribuírem com dicas, observações e sugestão, para a melhoria do trabalho.

Quando escrevi o primeiro texto ainda tentando construir uma ideia para o projeto final de curso estava sob a orientação da professora Maria de Fátima Oliveira Mattos Grassi conhecida por (Fatinha), que me ajudou a estruturar melhor a ideia sobre o trabalho.

Em agradecimento dedico este trabalho em memória à professora Fatinha Mattos.

RESUMO

O surgimento de smartphones e do sistema operacional Android favoreceu o desenvolvimento de diversos aplicativos para os mais diversos fins (gerenciamento, rotas, vendas, jogos, etc.). Alguns aplicativos possuem a capacidade de definir uma rota a partir de um endereço inicial e final, um exemplo de aplicativo que utiliza esse recurso é o Google Maps.

Apesar das facilidades proporcionadas pelos smartphones, empresas do setor de serviços (frete, entregas, vendas, etc.) ainda não utilizam essa tecnologia em escala para otimizar seus serviços no gerenciamento do trajeto, desse modo, organizar uma rota manualmente implica usar mais tempo na realização dos serviços, pois depende da capacidade do servidor em definir o trajeto a ser percorrido, através da interpretação dos mapas, do conhecimento prévio das localidades e da organização do itinerário em um mapa físico e mental.

Dessa forma para otimizar o trabalho de criar o trajeto, seria necessário utilizar algum aplicativo que fornecesse o serviço que permitisse manipular a rota, porém os aplicativos existentes (Google Maps e Waze) são utilizados para definir um percurso, mas não permitem a manipulação ou edição das rotas. Permitem somente utilizar as rotas para encontrar o melhor caminho.

Este trabalho propõe o desenvolvimento de um aplicativo que permita a utilização de vários endereços para gerar um trajeto e que possibilite ao usuário inserir, alterar, remover um endereço para corrigir a rota, conforme sua necessidade.

Para desenvolver este aplicativo, serão utilizados ferramentas existentes como APIs Google Maps (Location, Places, Geolocation, Direction), IDE (Android Studio), Banco de dados SQLite e ferramenta de prototipagem e modelagem.

Palavras-chave: Aplicativo móvel, Google Maps, API Location, API Places, API Geolocation, Android, Rotas.

SUMÁRIO

LISTAS DE ABREVIATURAS E SÍMBOLOS

AJAX	*Asynchoronous Javascript and XML*
API	*Application Programming Interface*
APK	*Android Package*
APP	*Application*
AVD	*Android Virtual Device*
CCH	Caminho ou Circuito Hamiltoniano
CRUD	Acrônimo do inglês *Create*, *Read*, *Update and Delete*
CSV	*Comma-separated values*
GPS	*Global Positioning System*
IDE	*Integrated Development Environment*
NDK	*Native Development Kit*
NP	*Nondeterministic Polynomial*
PCV	Problema do Caixeiro Viajante
SDK	*Software Development Kit*
URL	*Uniform Resource Locator*
TXT	Simboliza a identificação de um arquivo-texto.
XML	*eXtensible Markup Language*
XD	*Design Experience Adobe*

LISTAS DE FIGURAS

LISTAS DE TABELAS

Capítulo 1

Introdução

O surgimento dos aplicativos móveis trouxe consigo um grande número de aplicações para os mais diversos fins. Dentre esses aplicativos podemos destacar alguns que possibilitam realizar navegação por mapa utilizando a localização por sistema de posicionamento global (GPS – Global Positioning System). Comumente, aplicativos com estes recursos utilizam a inserção de uma localidade inicial e final para projetar uma rota no mapa.

Segundo Castro (1999, p.1), um dos maiores desafios ao planejar rotas consiste na ponderação de todos os fatores individuais e subjetivos, que fazem com que uma rota seja considerada aceitável ou não.

Os aplicativos Google Maps e Waze cumprem o seu propósito ao gerar rotas com o melhor caminho. Se estiverem *online* apresentam indicativos do que está acontecendo no trajeto em tempo real, mas se considerarmos a forma de como é gerada uma rota nestes aplicativos, torna-se limitado para a execução de serviços que utilizam muitos endereços para visitação, já que para gerar o trajeto, os endereços têm que ser inseridos um a um e o cálculo do melhor caminho se limita no caso do Google Maps a 10 endereços, não permitindo alterações na estrutura do trajeto para adaptá-la ao itinerário definido.

Desta forma verificou-se a necessidade de desenvolver um aplicativo para gerar rotas que permita manipulação em um número maior de endereços. Os endereços serão carregados no mapa do aplicativo através de um arquivo-texto, e serão ligados manualmente ponto a ponto na ordem desejada para visitação, desta forma poderá ser realizado as alterações conforme necessidade.

Para exemplificar o uso de rotas o problema do caixeiro viajante será utilizado. Segundo (Castro apud Hoffman e Wolfe, 1985), o problema do caixeiro viajante consiste em um vendedor que precisa encontrar o menor caminho para visitar uma série de cidades sem repeti-las e retornar ao ponto de origem.

O problema do caixeiro viajante ajuda a contextualizá a importância de se desenvolver um aplicativo para auxiliar a visitação de endereços em um trajeto devido à complexidade exigida para gerar uma rota com o melhor caminho, porém não serão tratados soluções para o problema do caixeiro viajante, mas, será desenvolvido um aplicativo para facilitar a manipulação dos endereços e um mapa.

Para ajudar na "construção" desse aplicativo será preciso estudar conceitos de banco de dados, linguagem de programação, modelagem de sistemas, UX (User Experience Design) e UI (User Interface Design).

1.1 Objetivos

Este trabalho tem como objetivo principal desenvolver um aplicativo que realizará a leitura de endereços contidos em um arquivo-texto e em seguida apresentar tais endereços como pontos distribuídos no mapa, permitindo que o usuário defina a ordem de visitação, conectando os pontos a partir do ponto que deseja.

1.1.1 Objetivos Específicos

Para realizar este projeto será necessário seguir algumas etapas para o desenvolvimento do aplicativo, tais etapas são:

- Estudo da API Google Maps, Location, Places, Directions e Geolocation;
- Instalação e configuração do ambiente de desenvolvimento para Android, IDE Android Studio, SDK;
- Estudo da linguagem Java para desenvolvimento de aplicativos móveis;
- Estudo do banco de dados SQLite para dispositivos móveis;
- Modelagem;
- Prototipagem;
- Técnicas UI/UX;
- Testes e validações de sistemas.

1.2 Justificativa e Motivação

A busca por melhorias e otimizações atinge a maioria dos setores, comercial, industrial e de serviço. Dessa forma fornecer ferramentas que possam otimizar os serviços de logística

destes setores, permitirá que o processo de organização, manutenção de rotas para a visitação, se torne mais ágil.

1.3 Metodologia

A documentação que o Google disponibiliza em seu website (DEVELOPERS, 2018) será utilizada para padronizar o desenvolvimento do aplicativo através das melhores práticas, bem como instruções de como utilizar as APIs. Nesta documentação existem exemplos de códigos, padrões de arquitetura, design etc e também foram utilizados artigos, dissertações, teses, trabalhos acadêmicos e livros existentes em bibliotecas físicas e virtuais.

Para mostrar em sua interface o mapa será utilizado a API Maps para android, esta permitirá utilizar classes e métodos próprios para instanciar objetos, encontrar a localização atual, calcular a distância entre os endereços, buscar informações referentes ao endereço pesquisado, etc.

O banco de dados será criado através da ferramenta SQLite e será abordado com mais detalhes na Seção **5.1.12**.

1.4 Organização do Texto

O **Capítulo 2** apresentará a definição do problema proposto no trabalho, o problema do caixeiro viajante e o caminho hamiltoniano, bem como a definição de grafos e uma demonstração de como os aplicativos Google Maps e Waze funcionam.

No **Capítulo 3** será explicado o que é o sistema Android, a porcentagem de aparelhos utilizando o sistema no ano de 2019 e a versão utilizada e o motivo da escolha desta versão.

O **Capítulo 4** apresentará as ferramentas utilizadas para desenvolver o aplicativo proposto, Android Studio e SQLite.

O **Capítulo 5** apresentará o aplicativo FlexMap e suas funcionalidades já desenvolvidas.

Por fim, o **Capítulo 6** serão apresentadas as conclusões finais e trabalhos futuros.

Capítulo 2

Definição do problema

O problema proposto neste trabalho foi percebido através da experiência profissional e pessoal do autor, onde o mesmo necessitava realizar visitas em muitos endereços no decorrer do período de trabalho e, por muitas vezes, precisava definir uma rota manualmente e corrigi-la toda vez que fosse preciso realizar uma parada para almoçar ou atualizar os endereços do trajeto. Desta forma foi observada essa demanda nos setores onde o serviço de visitação de múltiplos endereços, como entregas, fretes e vendas são utilizados.

Alguns problemas são percebidos no processo de organizar uma rota manualmente:

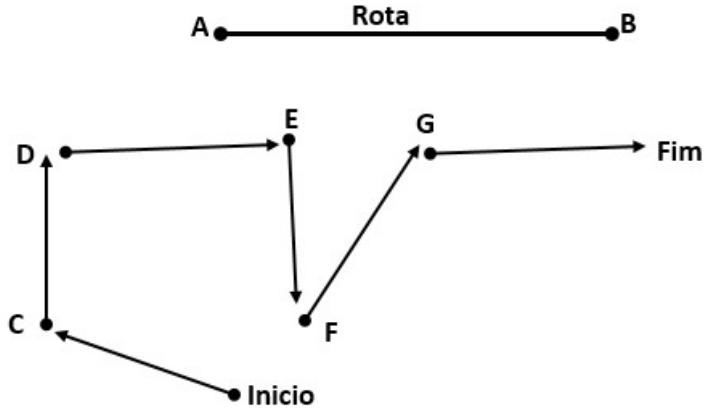
1. Se houver muitos endereços, a dificuldade em memorizá-los é alta;
2. Orientar-se por um mapa físico demanda tempo;
3. Mudar a rota definida acarreta em reorganizar a mesma.

2.1 Rota

Uma rota é um caminho que se escolhe para ir de um ponto a outro, um trajeto. Seguindo essa definição podemos aplicá-la na construção da funcionalidade do aplicativo partindo do princípio que, para se ter uma rota devemos ter dois pontos, inicial e final, que ao serem ligados produzirão segmento de um trajeto.

Para que seja definido rota e trajeto será utilizado a **Figura 1.1** indicando uma reta que se conecta através dos pontos ***A*** e ***B*** formando uma rota. O trajeto será definido por um par de pontos conectados através de uma reta ***[(INÍCIO e C), (C e D), (D e E), (E e F), (Fe G) e (G e FIM)].***

Figura1.1 – Exemplo de rota.



Fonte: o autor (2019)

2.2 Problema Caixeiro Viajante (PCV)

O PCV consiste em um problema no qual o viajante tem que encontrar o menor caminho entre uma série de cidades, visitá-las e retornar a cidade de origem. Esse é um problema de otimização NP-difícil inspirado na necessidade dos viajantes em realizar entregas em diversas localidades, percorrendo o melhor caminho possível, (CERVERI e PY, 2000).

A classe NP-difícil é considerada na teoria da complexidade computacional um conjunto de problemas que acredita-se não possuir solução em tempo polinomial, (SILVA, 2013).

Segundo (CASTRO apud Hoffman e Wolfe, 1985) a importância do problema do caixeiro viajante não está em um grande leque de aplicações derivadas ou semelhantes a este problema, pois não existem muitas, mas sim porque este é um problema típico do seu gênero, de **otimização combinatória**.

> Sendo um grafo G = (N, E) onde N = {1,…, *n*) é um conjunto de nós e E = {1,…., *m*) é o conjunto de arestas de G, e custos, $\boldsymbol{C_{ij,}}$ associados com cada aresta ligando os vértices *i* e *j,* o problema consiste em localizar o menor ciclo Hamiltoniano do grafo G. O tamanho do ciclo é calculado pelo somatório dos custos das arestas que formam o ciclo. Os nós do grafo são, frequentemente, referenciados como “cidades” e, em outras palavras, o objetivo é visitar todas as cidades passando apenas uma vez por cidade, retornando ao ponto de origem. Este

> percurso deve ser feito de forma a minimizar a distância total percorrida.
> Dada a **Formulação Combinatória** um conjunto $C=\{C_1,\ldots,C_n\}$ de n cidades C_j e uma matriz de distâncias **(ρ_{ij}),** onde $\rho_{ij}=\rho(C_i,C_j),(i,j\in(1,\ldots,n),\rho_{ij}=\rho_{ji},\rho_{ii}=0)$, a tarefa passa por encontrar a permutação $\pi\in S_n=(s:(1,\ldots,n)\rightarrow(1,\ldots,n))$ que faça com que a função objetivo (distância do circuito) $f:S_n\rightarrow R$, onde $f(\pi)=\sum_{i=1}^{n-1}\rho\pi(i),\pi(i+1)+\rho\pi(n),\pi(1)$, atinja o seu mínimo. O tamanho do espaço de procura aumenta exponencialmente dependendo de **n**, o número de cidades, uma vez que existem circuitos possíveis (a posição inicial é arbitrária e a ordem do circuito pode ser invertida). (Hoffman e Wolfe, 1985).

Podemos perceber que se a quantidade de cidades aumentar, calcular o melhor caminho por meio de força bruta em um tempo razoável é impossível.

Vejamos o porquê:

1. Temos que reduzir a um problema de enumeração;
2. Achar todas as rotas possíveis;
3. Calcular o comprimento de cada uma delas;
4. Verificar qual é o melhor caminho.

Alguns diriam que, para um computador essa tarefa seria “moleza”, fácil de realizar, porém para encontrar o melhor caminho temos que passar por todas as cidades e retornar à primeira cidade.

Supondo ter um computador muito veloz, capaz de fazer ***1 bilhão*** (***10^9***) de cálculos por segundo. Isso parece uma velocidade imensa, capaz de tudo.

Se ***n*** for muito pequeno ***n <= 10*** um computador levaria milésimos de segundo para realizar o cálculo, assim encontrar o número total de rotas ***R (n)*** levando em consideração a notação fatorial, temos que ***(n-1)!***, neste caso não calcula a primeira cidade, fica ***R(n)=(n-1)!***, por exemplo **(*n=10)!*** seria efetuado ***R(n-1)! = R(9)!=362.880*** combinações possíveis de cidades sem repetições, essas cidades geram uma verificação de ***10^9/9=110*** milhões de rotas por segundo.

Com efeito, no caso de 20 cidades, o computador precisa apenas de ***19*** adições para dizer qual o comprimento de uma rota e então será capaz de calcular ***10^9/ 19 = 53*** milhões de rotas por segundo. Contudo, essa imensa velocidade é um nada diante da imensidão do número ***19!*** de rotas que precisará examinar. Com efeito, acredite se puder, o valor de ***19!*** é

121 645 100 408 832 000 ou aproximadamente, ***1.2 x 10^{17}*** em notação científica. Consequentemente, ele precisará de ***1.2 x 10^{17}/ (53 milhões) = 2.3 x 10^{9} segundos*** para completar sua tarefa, o que equivale há cerca de 73 anos. O problema é que a quantidade ***(n-1)!*** cresce com uma velocidade alarmante, sendo que muito rapidamente o computador torna-se incapaz de executar o que lhe pedimos. (SILVEIRA, 2006).

A **Tabela 1.1** indica como ficaria as possibilidades para encontrar as rotas.

Tabela 1.1 – Exemplo de cálculo por *n* rotas

n	Rotas por segundo **10^{9}/(n-1)**	**(n-1)!**	Cálculo total **(n-1)!/(10^{9}/(n-1))**
5	250 milhões	24	Insignificante
10	110 milhões	362880	0,003 seg.
15	71 milhões	87 bilhões	20 min
20	53 milhões	1.2×10^{17}	73 anos
25	42 milhões	6.2×10^{23}	470 milhões de anos

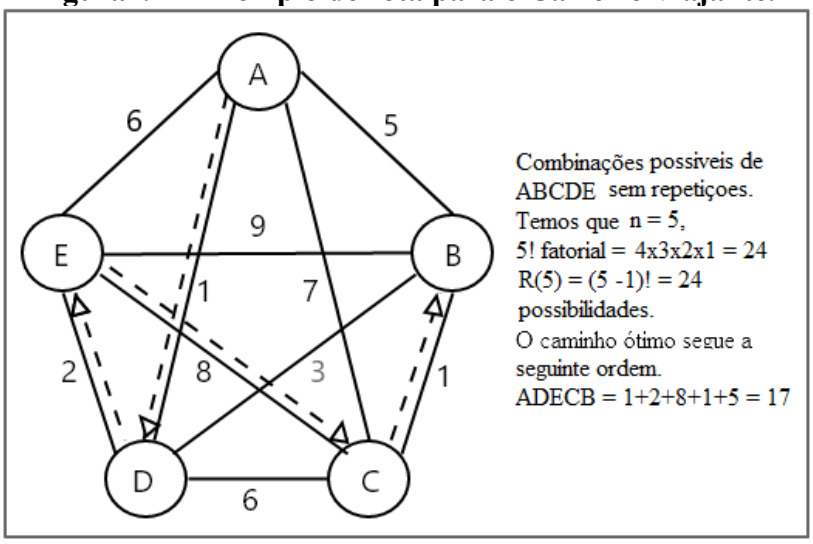
Fonte: SILVEIRA

Segundo (SILVEIRA, 2006), o aumento no valor de ***n*** provoca uma diminuição na velocidade com que o computador computa o tempo de cada rota como mostrado na coluna "rotas por segundo", e logo provoca um enorme aumento no tempo total do cálculo. Em outras palavras a inviabilidade computacional se deve à presença do fatorial na medida do esforço computacional do método de força bruta.

Para exemplificar melhor a viagem do PCV, a **Figura 1.2** demonstra uma viagem (circuito) passando por todas as cidades ***(A, B, C, D*** e ***E),*** em um mapa.

O tamanho de um circuito é o somatório das linhas que fazem parte da viagem, neste exemplo as linhas tracejadas. Um circuito ótimo neste mapa passa pelas cidades ***A-D-E-C-B***, com distância percorrida total de ***d=17***.

Figura1.2 – Exemplo de rota para o Caixeiro Viajante.



Fonte: O autor

O problema de encontrar o melhor circuito, consiste em calcular um número muito grande de combinações possíveis, impossibilitando a análise computacional. Para minimizar o custo computacional existem dois métodos que tratam o PCV:

1. Métodos exatos;
2. Métodos heurísticos;

Métodos exatos são aqueles que têm como característica a capacidade de determinar sempre uma solução ótima para o problema. Exemplo: Força Bruta e Backtracking.

Métodos aproximados (heurísticos) são aqueles que garantem a determinação de soluções que retornam um resultado satisfatório em tempo hábil.

Os algoritmos capazes de produzir um resultado satisfatório são: algoritmos genéticos (Ags), algoritmos ACO (Colônia de Formigas), algoritmos por programação dinâmica, algoritmos de meta-heurística (Algoritmos Gulosos). (Mathforum, 2017).

Estes algoritmos não serão vistos neste trabalho, pois servirão somente para contextualizar a problemática do caixeiro viajante, já que a API do Google fornece os cálculos para encontrar o menor caminho se for necessário.

2.3 Caminho e ciclo Hamiltoniano (CCH)

Segundo (COSTA, 2011), um grafo pode ser definido da seguinte forma: “Um grafo simples ***G*** é formado por um par de arestas e vértices ***(V (G), A (G))*** onde vértices ***V(G)*** é um

conjunto não vazio e arestas ***A (G)*** um conjunto de pares distintos não ordenados de elementos distintos de ***V(G)***".

Um caminho Hamiltoniano é um caminho que permite passar por todos os vértices de um grafo ***G***, não repetindo nenhum vértice.

> Todos os grafos têm vértices e arestas. Agora, imagine viajar do vértice ao longo das bordas. Podemos caracterizar matematicamente esta "viagem ao redor do grafo" com uma lista em ordem dos vértices e arestas que compõem a "viagem". Esta série de vértices e arestas é chamada de **caminhada**. Em uma caminhada, há muito poucas restrições, de modo que as bordas ou vértices podem ser visitados em qualquer número de vezes. Uma caminhada em que um vértice não é repetido é chamado de **caminho**. Um caminho hamiltoniano visita todos os vértices exatamente uma vez.
> Se estendemos um caminho Hamiltoniano para conectar o vértice final ao vértice inicial por meio de uma aresta existente, então criamos algo novo – um ciclo Hamiltoniano. Um ciclo é uma caminhada que se conecta de volta ao seu vértice de partida, enquanto um ciclo hamiltoniano deve atingir todos os vértices exatamente uma vez antes de retornar ao vértice inicial. Se um grafo tiver um ciclo Hamiltoniano, então ele é chamado de **grafo hamiltoniano**. (Mathforum, 2017).

O problema é determinar se um grafo possui um ciclo Hamiltoniano. No problema do caixeiro viajante vimos a complexidade de encontrar caminhos ótimos, caso haja muitas cidades, o mesmo problema se aplica ao caminho Hamiltoniano.

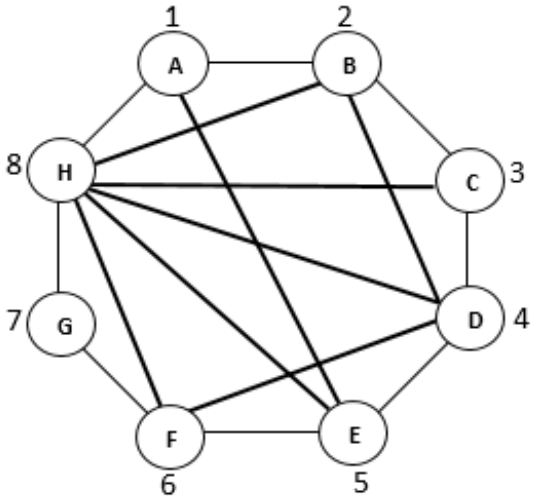
Para verificar se um grafo possui um ciclo Hamiltoniano existem 2 teoremas úteis, Teorema de Dirac e Teorema de Bondy-Chvátal;

- Teorema de Dirac, "Se o grafo simples ***G*** tiver $n \geq 3$ vértices e se $d(v) \geq \frac{n}{2}$, onde ***v*** é o grau de vértice mínimo em ***G***, então ***G*** é um grafo hamiltoniano. (Mathforum, 2017).
- Teorema de Bondy-Chvátal, "Seja ***G*** ser um grafo simples e seja ***n*** o número de vértices em ***G***. Sejam u e v dois vértices não adjacentes em ***G***, de modo que ***d (u) + d (v)n***. Então ***G*** é Hamiltoniano se e somente se ***G + uv*** for Hamiltoniano". (Mathforum, 2017).

A **Figura 1.3** exemplifica o que é um caminho ou ciclo Hamiltoniano.

O caminho a seguir ***H, D, B,*** definir um grafo não-direcionado e representa um ciclo, que se inicia a partir do ***A, B, C, D, E, F, G, H*** e retorna ao ponto inicial ***A***.

Figura1.3 – Exemplo de Caminho Hamiltoniano.



Fonte: O autor.

2.4 Diferenciando PCV e CCH do problema proposto

Vimos a definição do problema do caixeiro viajante e do ciclo Hamiltoniano, que no caso do PCV visita todas as cidades do trajeto e retorna ao ponto inicial encontrando o melhor caminho, já o CCH realiza uma visita a todos os endereços ao menos uma vez e retorna ao ponto inicial sem repetir o mesmo caminho.

A proposta do aplicativo utiliza parte das definições apresentadas do PCV e do CCH, para visitação, no entanto a diferença está em permitir o retorno para o ponto inicial tanto pelo mesmo caminho ou caminhos diferentes desde que satisfaça a necessidade do serviço prestado, desta forma há repetições de pontos (vértices) visitados.

Para este aplicativo não terá regras que impossibilitem utilizar a mesma rota como retorno, porque se levarmos em consideração que os trajetos podem sofrer atualizações, então é preciso adaptá-lo ao itinerário que for novamente fornecido.

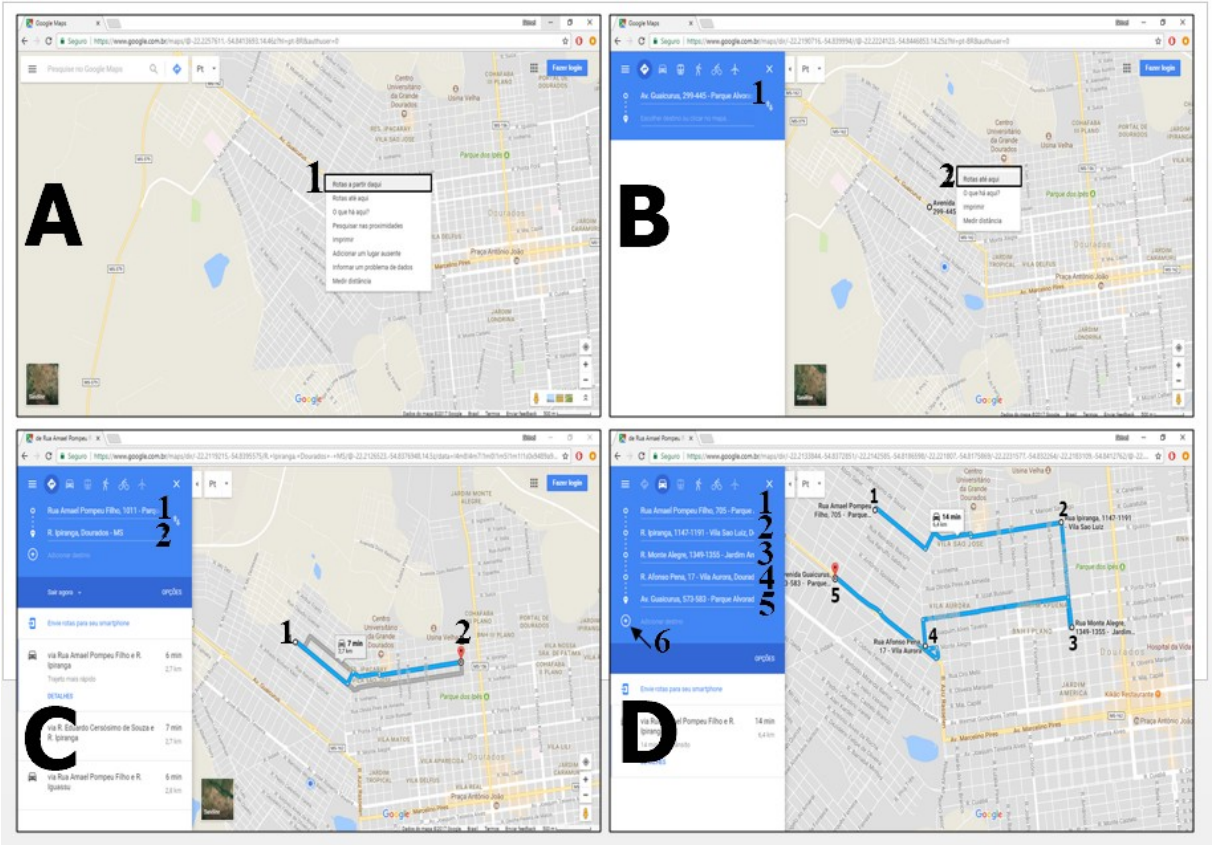
2.5 Google Maps

Tanto a versão web como a versão *mobile* permite definir rotas. Para definir uma rota na versão *web* deverá ser realizado as seguintes etapas:

- No quadro **A**, no campo pesquisar ou a partir do botão direito do mouse definir o primeiro endereço na opção “rota a partir daqui”, indicado com o número 1;
- O quadro **B** indica como colocar o endereço 2, com o botão direito do mouse escolher a opção “rota até aqui”, que está marcado com o número 2;
- Os endereços 1 e 2 foram conectados e foi definido uma rota, que está representado pelo quadro **C**;
- No quadro **D,** um trajeto foi definido com os endereços de 1 a 5 que pode ser visualizado no mapa;
- O número 6 indica como adicionar mais endereços, através do botão com o ícone “+”;
- O máximo de endereços permitido é 10.

A **Figura 1.4** demonstra como realizar a inserção dos endereços.

Figura1.4 – Exemplo de Inserção de endereços no Google Maps.



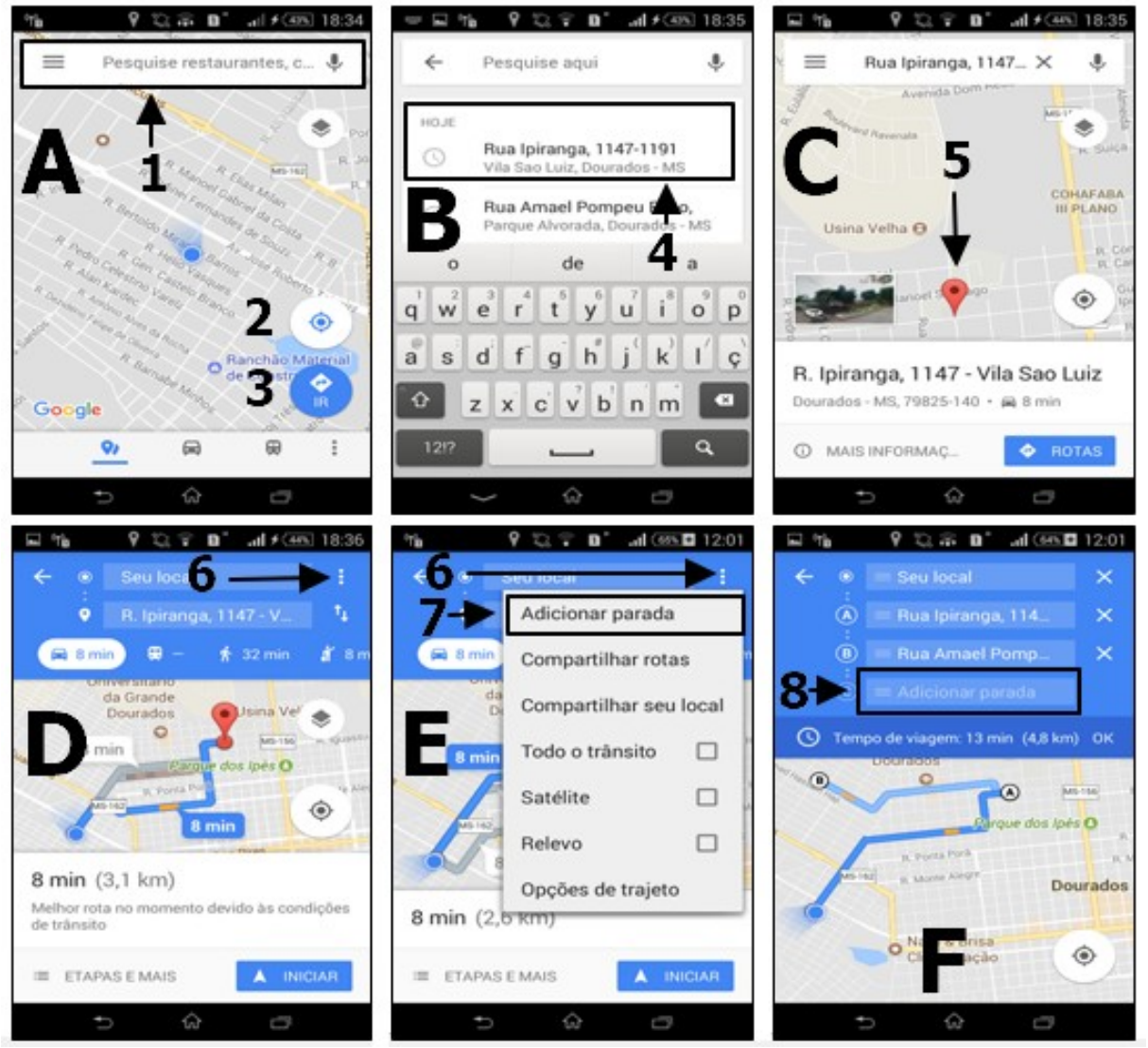
Fonte: Google Maps

Para criar uma rota no aplicativo *mobile* as funcionalidades são semelhantes, porém para definir mais endereços no mapa, tem que acessar um submenu e ir na opção “adicionar parada”, a etapa a seguir indica como adicionar os endereços.

- Utilizar o campo de pesquisa para inserir um endereço número 1 ou o botão que busca a localização por GPS número 2, quadro **A**;
- Clicar no botão “IR” número 3 logo abaixo do botão GPS número 2, quadro **A**;
- No campo “Pesquisar Aqui”, se for clicado aparecerá um histórico com os endereços mais recentes e um teclado para digitar novos endereços número 4, quadro **B**;
- Um ponto será posto no mapa número 5, quadro **C**, no quadro **D** uma rota será definida;
- A opção com 3 pontos em branco quadro **D** número 6, serve para utilizar um submenu de funcionalidades;
- Ao clicar no submenu com três pontos em branco número 6, aparecerá como primeira opção “adicionar parada” número 7, quadro **E**, se clicar nesta opção um campo vazio aparecerá para ser inserido um novo endereço, quadro **F** (número 8);
- Uma rota será definida entre o ponto que representa “Seu local”, **A**, **B** no mapa;
- Será permitido definir o máximo de 10 endereços.

A **Figura 1.5** exemplifica como funciona a inserção de novos endereços do aplicativo Google Maps.

Figura1.5 – Adicionando endereços uma rota Google Maps Android.



Fonte: Google Maps

Como podemos perceber a inserção a partir de 2 endereços depende da iteração com a funcionalidade “Adicionar parada”, no qual é inserido endereços sequencialmente, que fica oculto em um submenu.

2.6 Waze

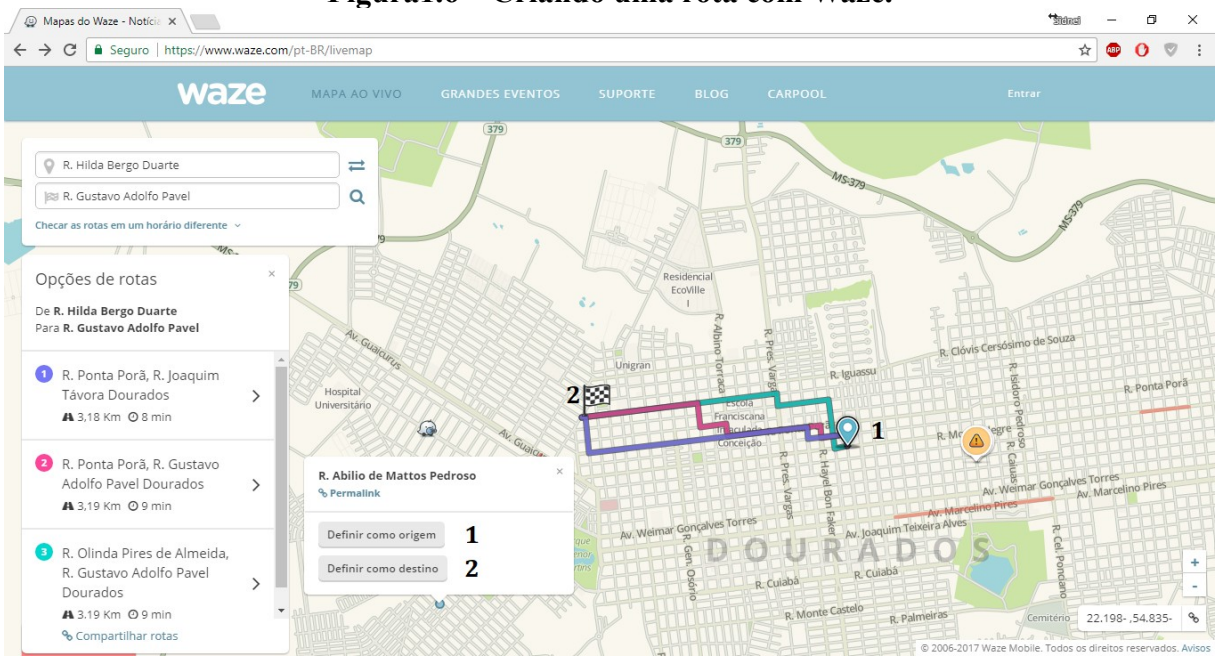
Waze difere do software navegador GPS tradicional, pois é um aplicativo comunidade de direção, que fornece dados complementares do mapa e outras informações de tráfego dos usuários. Como outro software de GPS, ele aprende conforme os usuários dirigem para fornecer rotas e atualizações de tráfego em tempo real.

> A definição no Website do Waze diz: “Depois de digitar um endereço de destino, os usuários apenas dirigem com o aplicativo ligado e passam a contribuir passivamente com informações por onde trafegam. Ativamente, eles contribuem compartilhando alertas sobre acidentes, perigos, polícia e outros eventos ao longo do percurso, ajudando outros usuários da mesma área com informações atualizadas sobre o que está acontecendo ao redor”. (Waze, 2017).

Nos testes realizados com o Waze não foi possível inserir múltiplos endereços, o aplicativo não possui a opção de adicionar mais endereços além de origem (1) e destino (2), o que ocorre que ao tentar adicionar mais endereços de e destino o que é realizado é o recalculo da rota indicando uma nova opção.

A **Figura 1.6** demonstra a funcionalidade.

Figura1.6 – Criando uma rota com Waze.



Fonte: Waze

Capítulo 3

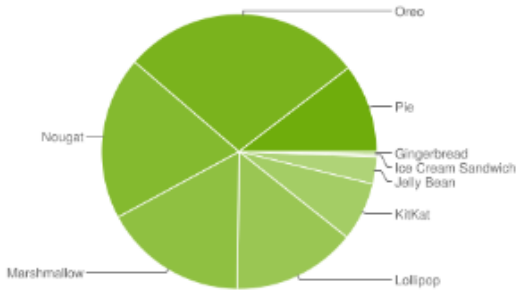
Sistema Android

Para entendermos melhor o funcionamento de um aplicativo, devemos conhecer a estrutura do sistema operacional para qual ele foi desenvolvido, neste trabalho utilizaremos o sistema Android, por este estar presente em 86% dos celulares no mundo, segundo pesquisa realizada em 2016 pela Gartner (*Website ComputerWorld*, 2016). Com a saída do Windows da disputa pelo mercado *mobile* o Android em 2017 possuía 88% do mercado segundo Jornal El País (ELPAIS, 2017). Aqui no Brasil atualmente segundo a revista digital Época Negócios do grupo Globo 230 milhões de celulares android estão ativos (EPOCANEGOCIOS, 2019).

O sistema Android está presente em dispositivos móveis (*smartphones*), televisores (*smart tvs*), relógios (*smartwatch*), automóveis etc. É um sistema de código aberto desenvolvido pela Google, o seu núcleo é baseado no Linux e parte da sua implementação possui trechos de código da linguagem de programação Java.

Figura1. 7 – Porcentagem de sistema Android ativo.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.5%
4.3		18	0.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1		22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1		25	7.8%
8.0	Oreo	26	12.9%
8.1		27	15.4%
9	Pie	28	10.4%



Dados coletados durante um período de sete dias encerrado em May 7, 2019.
As versões com menos de 0,1% de distribuição não são exibidas.

A **Figura 1.7** indica a porcentagem de aparelhos que utilizam determinada versão do sistema. A versão mais utilizada é a versão Oreo com 28% do sistema Android distribuído entre os aparelhos, desta forma utilizaremos esta versão como base para o aplicativo proposto neste trabalho.

Fonte: https://developer.android.com/about/dashboards/index.html

Cada versão do sistema possui uma API que é responsável por carregar todas as funcionalidades do sistema. Quanto mais nova é a versão do sistema maior é o número da

API. Por padrão deve-se indicar a versão do sistema com porcentagem suficiente que contemple um maior número de aparelhos antigos.

A documentação distribuída pela Google serve de parâmetro para criar um padrão para os aplicativos desenvolvidos, caso um desenvolvedor for seguir suas orientações. Desta forma é só seguir as indicações de como criar *layouts*[1], medidas de ícones, cores, animações, imagens, listas, etc, bem como utilizar os exemplos e os códigos disponíveis, para o aplicativo atender um padrão de distribuição mundial.

1 É um esboço ou esquema que mostra a estrutura física do projeto.

Capítulo 4

Ferramentas

Neste capítulo mostraremos as ferramentas utilizadas para desenvolver aplicativos para a plataforma Android, veremos a definição de IDE, SDK, AVD, NDK, APIs e como gerar um mapa em um projeto Android.

As ferramentas são parte importante no desenvolvimento dos aplicativos, que permitem inserir e utilizar funcionalidades, testar e compilar os aplicativos.

4.1 API Google Maps para Android

Neste trabalho vamos focar na API do Google Maps para utilizar suas funcionalidades. Esta biblioteca permite através de métodos a instanciação de um fragmento do mapa navegável simbolizando o planeta Terra.

Segundo a página do Google:

> Com o Google Maps Android API, você pode adicionar mapas baseados em dados do Google Maps ao aplicativo. A API processa automaticamente o acesso aos servidores do Google Maps, o download de dados, a exibição de mapas e a resposta a gestos de mapa. Também é possível usar chamadas a APIs para adicionar marcadores, polígonos e sobreposições a um mapa básico e para alterar a visualização de uma determinada área de mapa pelo usuário. Esses objetos fornecem informações adicionais sobre localizações do mapa e permitem a interação do usuário com o mapa.(GOOGLE, 2017).

Uma API (Application Programming Interface) é fornecida para a integração, que contribuirá para os diversos métodos de desenvolvimento.

Segundo (GOLVEIA, 2016), API é uma ferramenta que realiza comunicação entre aplicações que desejam compartilhar suas rotinas, ferramentas, padrões e protocolos. É uma espécie de mensageiro entre dois ou mais sistemas e tudo isso é possível porque a interação do Google Maps utiliza a tecnologia AJAX (Asynchoronous Javascript + XML).

A documentação da API traz muita informação de como utilizar a ferramenta. Aqui abordaremos as partes mais relevantes da documentação, maiores detalhes na página oficinal (GOOGLE, 2017).

4.1.1 Utilização da API Google Maps

Para utilizar a API do Google Maps para Android é necessário montar um ambiente de desenvolvimento e fazer a instalação das seguintes ferramentas: IDE, SDK e um AVD.

Neste projeto foi utilizado o Android Studio, para desenvolver o aplicativo, junto ao Android SDK Manager que facilita a atualização e instalação de pacotes para desenvolvimento e o AVD (Android Virtual Device) que serve para emular um aparelho com sistema Android.

É necessário obter uma chave para a utilização da API do Google Maps, essa chave é formada por uma string de caracteres alfanuméricos.

Segundo site do Google (2017).

> Para usar a Google Maps Android API, é necessário registrar o projeto do aplicativo no Google API Console e obter uma chave de API do Google para adicioná-la ao aplicativo. Observação: há vários tipos de restrição para as chaves de API. Você precisa de uma chave de API com restrição a aplicativos Android (não uma chave com restrição a navegador).

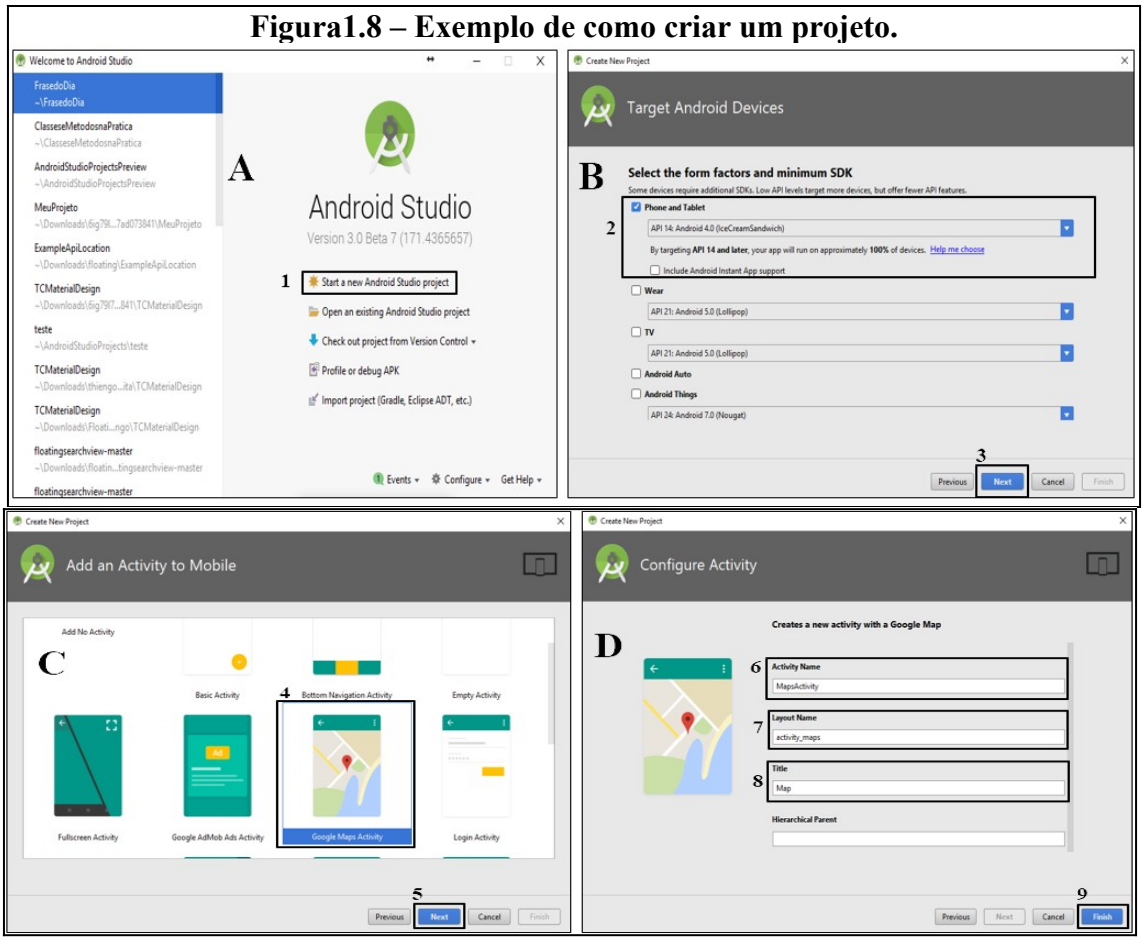
As etapas a seguir descrevem como criar um projeto no Android Studio e como utilizar o fragmento do mapa, através da API Google Maps.

- Quadro **A** deve escolher a opção "*Start a new Android Studio Project*" indicado pelo número 1;
- Quadro **B** escolher a opção para *smartphones* indicado pelo número 2, após a escolha clicar em *next* número 3;
- No quadro **C** deve ser escolhido o *layout* que melhor se adéqua ao projeto, neste caso o *layout* "Google Maps Activity", número 4, após escolhido clicar no botão *next* número 5;
- Quadro **D** e realizado as configurações do projeto, número 6 indica que pode ser configurado um nome para a *Activity*, número 7 permite renomear

o nome do *layout* e o número 8 permite colocar o título da aplicação, realizado estas configurações, clicar no botão *finish*;

A **Figura 1.8** demonstra como criar um projeto.

Figura1.8 – Exemplo de como criar um projeto.



Fonte: O autor

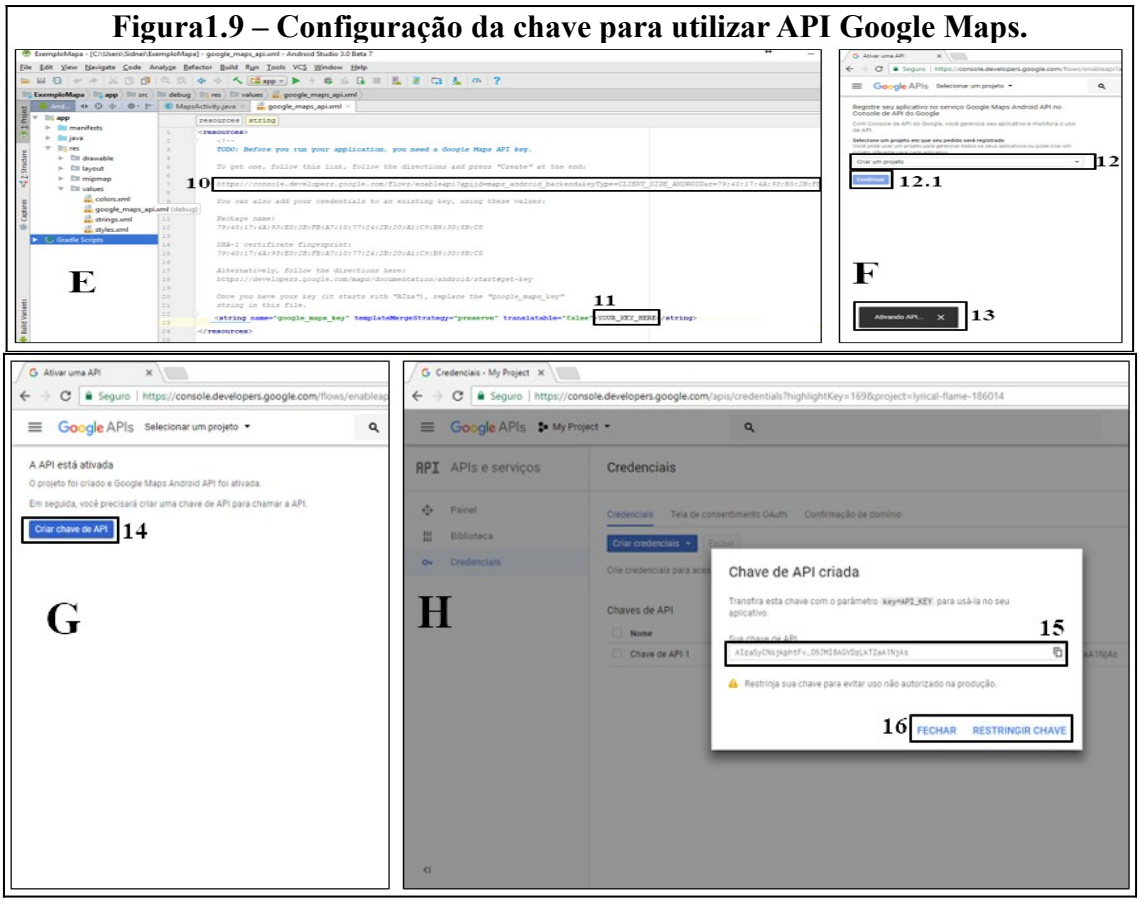
- O quadro **E** demonstra como foi gerado o projeto e como ele está organizado na IDE, para configurar o fragmento do mapa no projeto, será preciso copiar o link gerado pelo *layout* "*google_maps_api.xml*" criado pelo Android Studio, que está indicado pelo número 10, colar o link em um navegador e acessar o *Website* ***Console Developers*** do Google, que permite ativar a chave da API e inseri-la no campo reservado indicado pelo número 11;
- O quadro **F** representa um navegador acessando o Website através do link fornecido após criar um projeto no Android Studio com API Google Maps.

Para conseguir uma chave que ativa a API, deve escolher criar um projeto na opção número 12, e clicar no botão “continuar” número 12.1, feito isso será ativado a chave número 13;

- Após ativado a chave, poderá ser obtido a chave string de caracteres alfanuméricos**,** número 14 do quadro **G** indica esta ação;
- O quadro **H** representa a ação final que deverá ser copiado a chave para o projeto no Android Studio, os números 15 e 16 representam esta ação;

A **Figura 1.9** demonstra como criar um projeto.

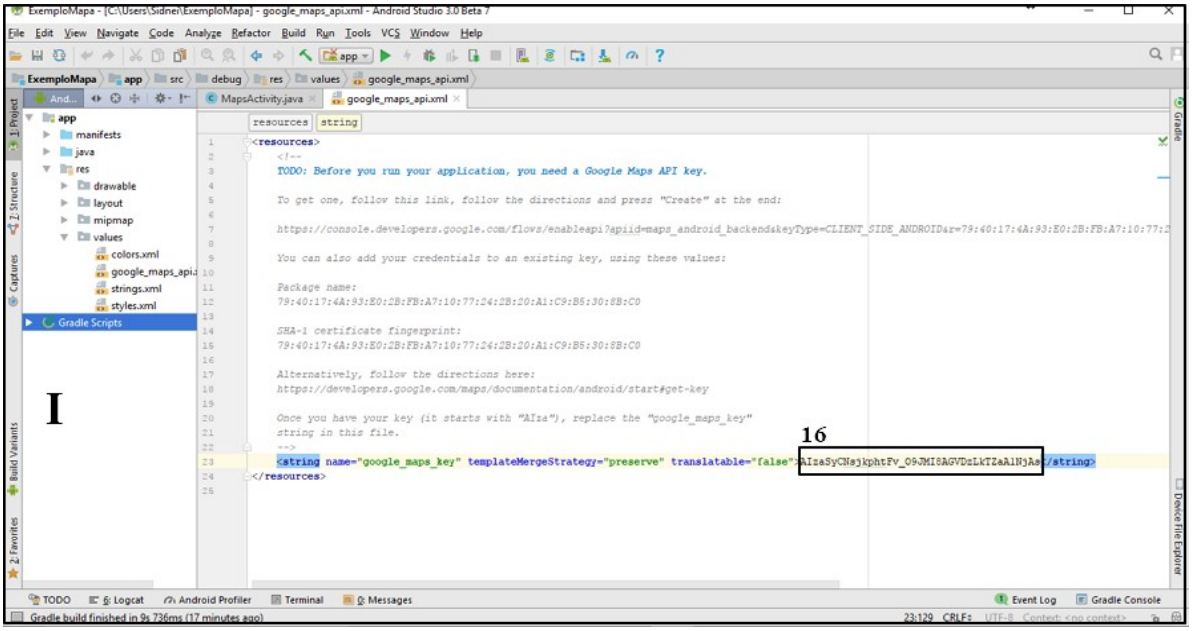
Figura1.9 – Configuração da chave para utilizar API Google Maps.



- No quadro **I** o número 16 indica como ficará a chave no arquivo e onde ela será posta, para instanciar o fragmento do mapa no projeto,

A **Figura 1.10** demonstra como o arquivo contendo a chave da API Google Maps ficará após a configuração.

Figura1.10 – Arquivo XML contendo a chave de ativação da API.



Fonte: O autor

Seguindo estas etapas o ambiente de desenvolvimento estará pronto para ser utilizado em projetos que utilizam API Google Maps.

4.1.2 IDE Android Studio

Android Studio é um ambiente de desenvolvimento integrado (IDE) baseado no Intellij IDEA da desenvolvedora JetBrains, (DEVELOPERS, 2019). O Android Studio foi desenvolvido com o intuito de facilitar o desenvolvimento de aplicativos Android. Desta forma a ferramenta disponibiliza ainda mais recursos para aumentar a produtividade na criação de aplicações Android.

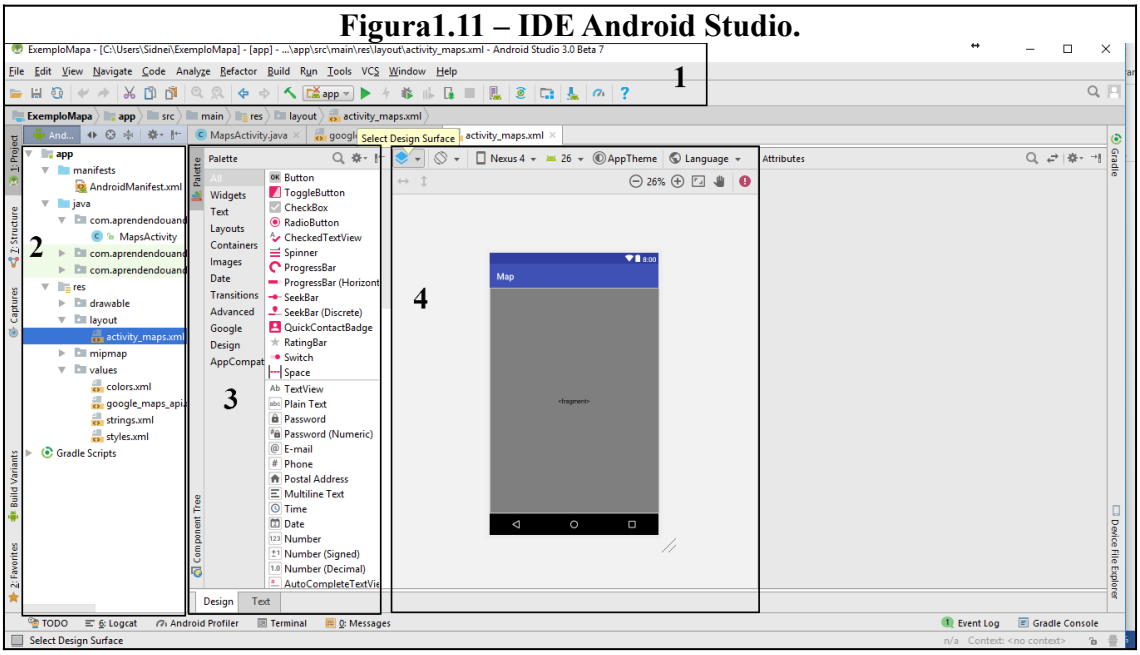
Em seu ambiente existem 4 grupos de funcionalidades que permitem a interação e configuração com os arquivos do projeto. A lista a seguir representa os grupos de funcionalidade que a **Figura 1.11** demonstra.

1. Caixa de opções, serve para a configuração da ferramenta como um todo. Neste grupo pode ser criado um novo projeto, salvo um projeto em uso, podem ser feitos atualizações em componentes necessários para a aplicação, etc;
2. Neste grupo está definido a organização do projeto. Nele pode ser encontrado o arquivo *manifest*, as classes Java própria da IDE ou classes criadas para o projeto, as

imagens, os valores padrão do projeto, os *layouts*, as *strings* padrão do projeto entre outros;

3. Neste grupo está todos os componentes agrupados por categorias que podem ser utilizados em um projeto. Por Exemplo: *Buttons, Texts, Plain Texts, Radio Buttons, ListViews, CheckBoxs*, etc;
4. Este grupo contém a visualização dos componentes que serão inseridos, em uma representação dos aparelhos que simulam os *smartphones, tablets, smartwatchs e samart Tvs.* Neste grupo poderão ser realizados as configurações de largura e altura dos componentes em relação ao limite máximo da representação dos aparelhos.

Figura1.11 – IDE Android Studio.



Fonte: O autor

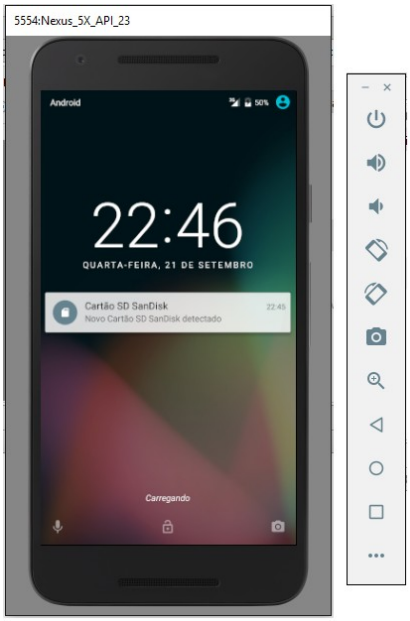
A documentação no site Google Developer descreve como funciona este ambiente de desenvolvimento:

- Um sistema de compilação flexível baseado no *Gradle* (*Gradle* é um sistema de automação de compilação *open source* que se baseia nos conceitos de Apache *Ant e Apache Maven*);
- Um emulador rápido com muitos recursos;
- Um ambiente unificado onde você pode desenvolver para todos os dispositivos Android;

- *Instant Run* (modificações em tempo de execução na *Activity* executada) para enviar alterações a aplicativos em execução sem compilar um novo APK;
- Modelos de códigos e integração com *GitHub* para ajudar a criar recursos comuns de aplicativos e importar exemplos de código;
- Ferramentas e estruturas de teste abrangentes;
- Ferramentas de verificação de código suspeito para detectar problemas de desempenho, usabilidade e compatibilidade de versões, entre outros;
- Compatibilidade com C e C++ e NDK (interface que permite comunicação com C e C++). "O Android Native Development Kit (NDK) é um conjunto de ferramentas que permitem usar código C e C++ em aplicativos Android. É possível usá-lo também para compilar a partir do seu próprio código-fonte ou aproveitar as bibliotecas pré-compiladas." (GOOGLE, 2017);
- Compatibilidade integrada com o Google *Cloud Platform*, facilitando a integração do Google *Cloud Messaging e do App Engine*.

Para testar os aplicativos a IDE conta com um AVD para os testes de desempenho, funcionalidades, armazenamento, fluidez, etc. O AVD serve para emular um aparelho que tenha instalado o sistema Android, como exemplo os *smartphones.* A **Figura 1.12** representa um AVD.

Figura1.12 – Exemplo de AVD Android Studio.



Fonte: O autor.

Após executar o “compilador”, é indicado uma opção para escolher a versão do sistema android configurado para iniciar o AVD, e o APK da aplicação é instalado no aparelho virtual permitindo os testes de funcionalidade.

Capítulo 5

FlexMap

O Aplicativo FlexMap foi idealizado para permitir a inserção de vários pontos de endereço no mapa, permitindo modificações e customizações da rota de forma a fornecer agilidade no decorrer da realização das tarefas, sem o usuário se preocupar em memorizar os endereços.

5.1 Estudo de caso

Para desenvolver o projeto, foi percebido de forma empírica a necessidade de um aplicativo que permitisse utilizar vários endereços para gerar uma rota de tal forma que fosse possível identificar cada endereço visitado, bem como, modificá-la para atender a realização dos serviços. Portanto, o aplicativo foi idealizado para auxiliar na realização dos serviços comerciais, como entradas, cobranças, vendas etc.

Os aplicativos convencionais, Google Maps e Waze, citados neste trabalho fornecem ótimas rotas, considerando diversos fatores, e fornecem uma gama de funcionalidades, como distância percorrida, velocidade, pontos de controle, radares, etc., mas não permitem modificações ou customizações nas rotas, além de ter limitações na geração de rotas para uma quantidade grande de endereços. Em resumo não foram projetados visando a execução comercial de serviços para visitações.

O FlexMap foi idealizado com o objetivo de permitir inserções de até 30 endereços. Existem duas principais escolhas para o máximo de endereços no aplicativo.

Primeiro o controle do volume de endereço para ser carregado no mapa evita à sobrecarga de marcadores para não atrapalhar a visualização do trajeto depois de ligado, segundo controlar a requisição do cálculo das rotas para o servidor do Google que será calculado através da *API Directions.*

O controle de sobrecarga para o cálculo da rota é evitado devido o envio dos pontos de latitude e longitude serem enviados em pares e não todos de uma vez. Assim após o usuário clicar no segundo marcador é feito o cálculo e retornado a distância entre os dois pontos para

serem conectados através de uma *polyline*[2], dessa forma toda vez será feito o envio em pares reiniciado o cálculo, isso evita a sobrecarga ou alguma exceção no servidor.

> Clientes de API mal projetados podem colocar mais carga do que o necessário nos servidores da Internet e do Google. Seguir essas práticas recomendadas na documentação pode ajudar a evitar que seu aplicativo seja bloqueado por abuso inadvertido das APIs […] um grande número de solicitações sincronizadas às APIs do Google pode parecer um ataque de negação de serviço distribuído (DDoS) na infraestrutura do Google e ser tratado de acordo. Para evitar isso, verifique se as solicitações de API não estão sincronizadas entre os clientes. (GOOGLE, 2019).

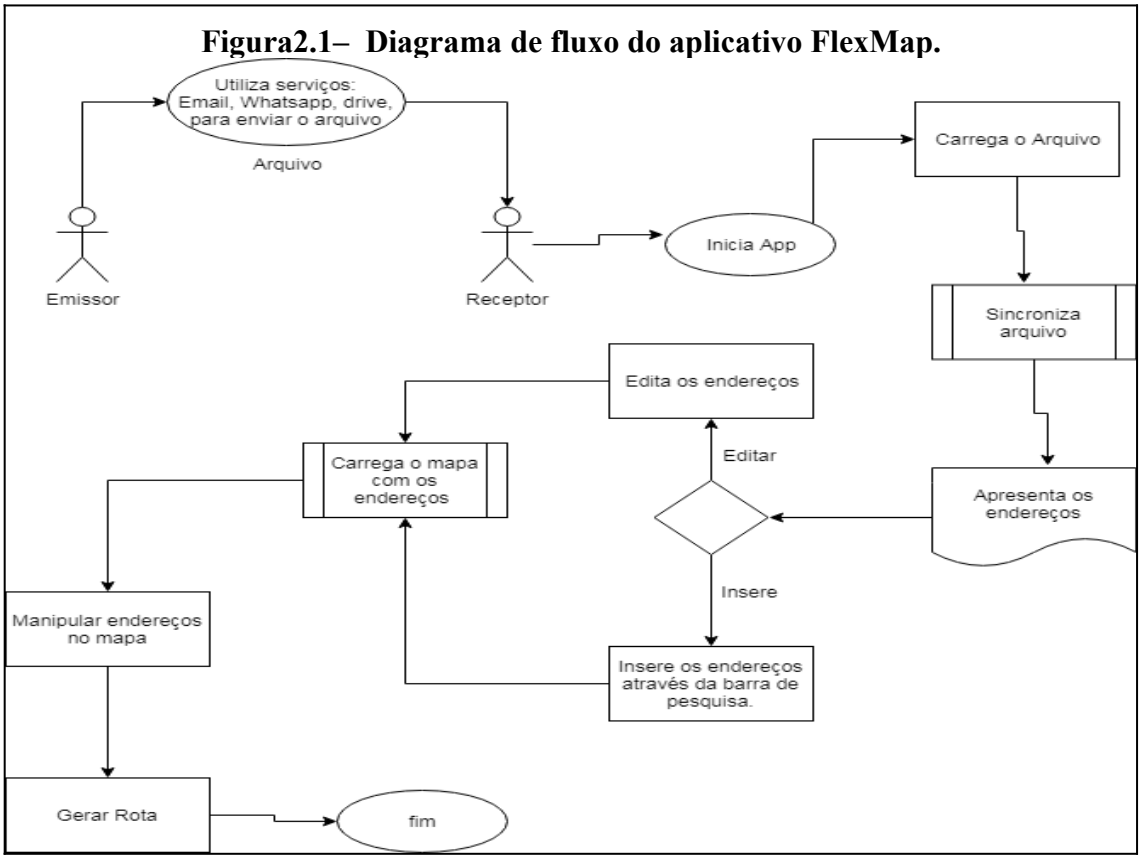
O aplicativo FlexMap deve ler um arquivo-texto contendo os endereços previamente preenchido pelo emissor (mais adiante veremos o conceito de emissor e receptor para o aplicativo), portanto, foi desenvolvido um padrão simples para criar o arquivo, para facilitar a leitura do mesmo.

Como o objetivo de otimizar a usabilidade foram desenvolvidos etapas, denominadas: **Tela Inicial, Carregar arquivo, Sincronizar, Carregar mapa, Editar ou excluir endereços, Inserir endereço por pesquisa, Gerar rota.**

A **Figura 2.1** ilustra o fluxo do aplicativo.

2 Reta gráfica gerada no mapa através do cálculo entre dois pontos de uma coordenada geográfica.

Figura2.1– Diagrama de fluxo do aplicativo FlexMap.



Fonte: O autor (2019)

5.1.2 Emissor

O emissor é responsável por criar o arquivo contendo os endereços no padrão que será apresentado no item 5.1.3, poderá ser utilizado algum sistema que possua um banco de dados com endereços previamente cadastrados, ou poderá ser feito a inserção dos endereços manualmente no arquivo “txt” ou “csv”.

Para disponibilizá-lo ao receptor poderá ser utilizado um provedor de e-mail, aplicativos de mensagens (*whatsapp, messenger facebook, Skype, etc*), drives virtuais (*Google Drive*, *One Drive, etc*), etc.

Na etapa em que o aplicativo está neste trabalho para carregar os endereços no mapa dever receber o arquivo de uma fonte externa como as citadas acima.

5.1.3 Arquivos de endereços

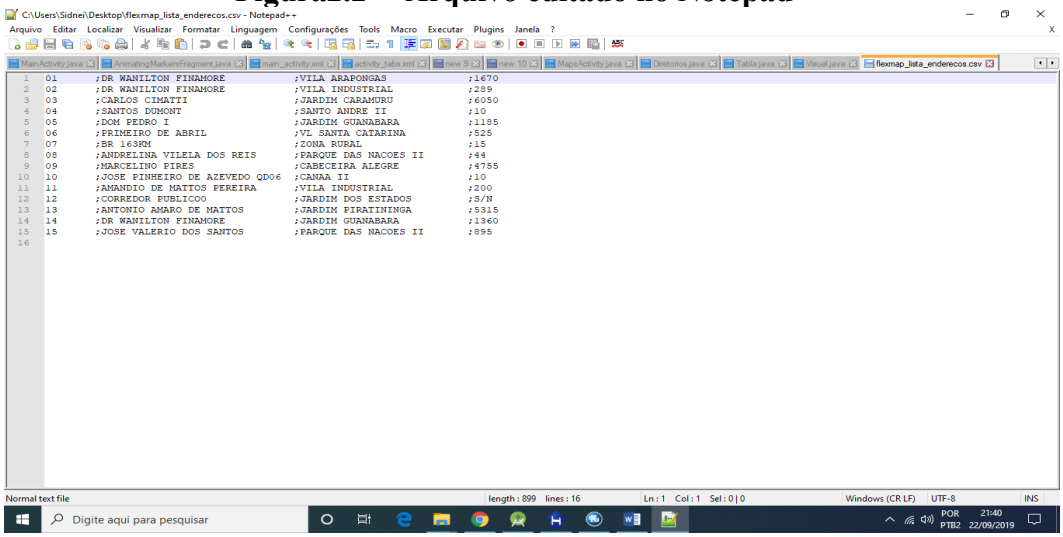
O arquivo gerado pode conter a extensão “txt” ou “csv”, caso o emissor possuir um sistema de gerenciamento e este poder gerar um arquivo contendo os endereços deverá estar

no padrão indicado a seguir desde que esteja em qualquer uma das extensões já citadas, o padrão adotado é bem simples no qual o aplicativo vai ser configurado para ler.

- O arquivo “txt” ou “csv” deverá conter obrigatoriamente um código (número inteiro sequencial), nome da rua, número do local, nome do bairro são prioritários, o nome da cidade e do estado são opcionais, devido o pré-processamento do arquivo-texto completar essas informações através da *API Geolocation* do Google. A separação entre cada informação deverá ser feita por um ponto e vírgula (“;”);
- O arquivo “csv” poderá ser editado pelo Excel que faz parte do pacote Office da Microsoft. Esse arquivo pode ser editado utilizando algum editor de texto como NotePad do próprio Windows ou NotePad++, desde que mantenham os mesmos padrões já mencionado;
- É muito importante que o arquivo “txt”, ou o arquivo “csv”, seja salvo com o nome flexmap, FlexMap ou Flexmap, para a identificação no momento em que o arquivo for lido no aplicativo, isso evita que o aplicativo fique procurando e lendo arquivos desnecessários caso haja outros arquivos “txt” ou “csv” na pasta definida pelo usuário quando o aplicativo é instalado, essa pasta serve para carregar o arquivo para o pré-processamento.
- Exemplo de como o arquivo é preenchido: 01; Rua Marcelino Pires; 3350; Centro; Dourados; MS.

A **Figura 2.2** faz referência de como o arquivo deve ser gerado. Neste caso foi utilizado o programa NotePad++ para editar e salvar o arquivo.

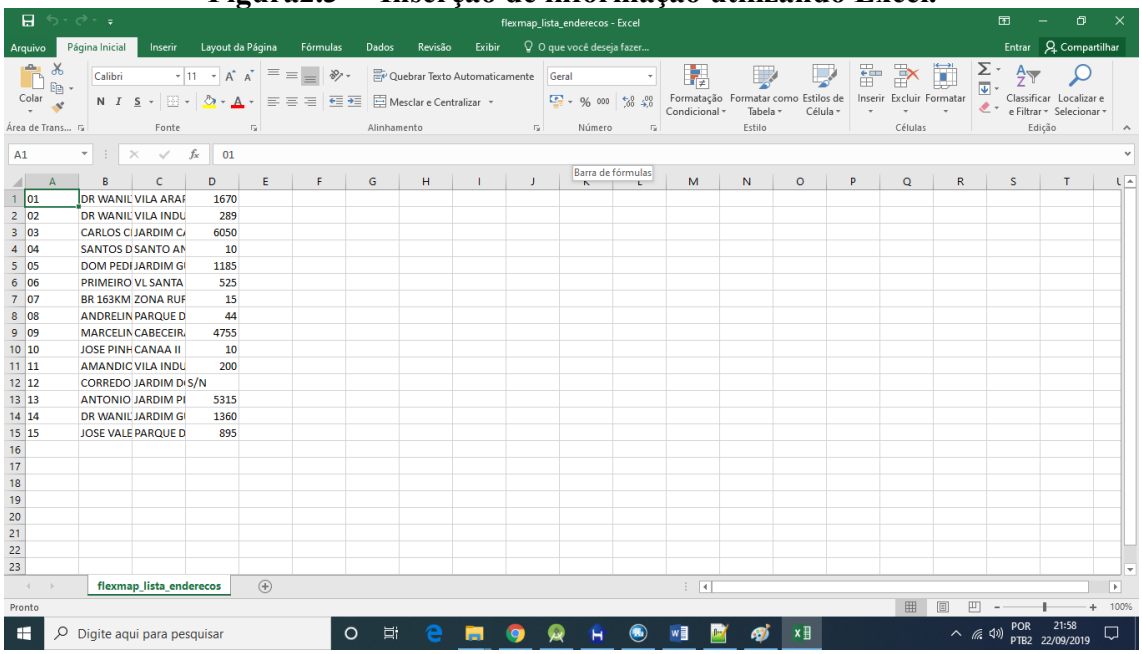
Figura2.2 – Arquivo editado no Notepad++



Fonte: O autor (2019)

A **Figura 2.3** apresenta a inserção de informações utilizando o Excel, ao salvar o documento deverá ser escolhido a opção csv, disponibilizada na aplicação.

Figura2.3 – Inserção de informação utilizando Excel.



Fonte: O autor (2019)

5.1.4 Receptor

O receptor é responsável por utilizar o aplicativo. Este receberá o arquivo contendo os endereços que serão processados para gerar os endereços no mapa, desta forma tais endereços poderão ser manipulados ligando-os ponto a ponto.

5.1.5 Tela Inicial

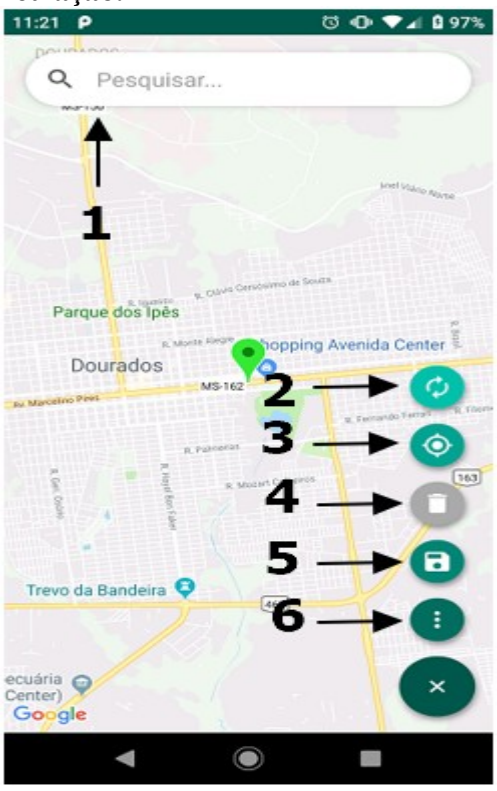
A primeira tela a aparecer no aplicativo após a instalação é um aviso pedindo para o usuário escolher uma pasta padrão onde ele receberá os arquivos contendo os endereços, esta pasta servirá para capturar os arquivos e processá-los no aplicativo. A pasta poderá ser trocada caso o usuário deseje acessando a parte de configurações do aplicativo.

A tela inicial carrega o mapa gerado pela API Google Maps os botões e o campo de pesquisa. Não existe uma tela de *login*, visto que o aplicativo é para ser de uso ágil e nesta etapa não será armazenado dados do usuário, desta forma as únicas informações salvas serão os endereços visitados.

A **Figura 2.4** representa a tela inicial do aplicativo, como os botões e o campo de pesquisa com o evento dos *buttons* de retração e a **Figura 2.5** demonstra a tela inicial com o evento dos objetos de não retração.

Figura2.4 – Tela inicial com retração.

Figura2.5 – Tela inicial sem retração.



Os objetos que compõem a tela inicial e suas funcionalidades serão listadas a seguir:

1. Campo de pesquisa: serve para encontrar um endereço digitado na base do servidor da Google;
2. Botão de sincronização: carregará o arquivo e processará o mesmo para compor outro arquivo com as informações dos endereços e latitude e longitude bem como salvar em banco de dados;
3. Botão para encontrar a localidade do usuário;
4. Botão para excluir objetos no mapa caso haja marcadores ou rotas definidas;
5. Botão para salvar uma rota caso ela for definida. Essa funcionalidade salva todos marcadores existentes no mapa em uma tabela no banco de dados;
6. Botão de configuração, nesta opção o usuário poderá alterar a pasta onde os arquivos são salvos, bem como a opção ajuda para encontrar instruções do aplicativo.

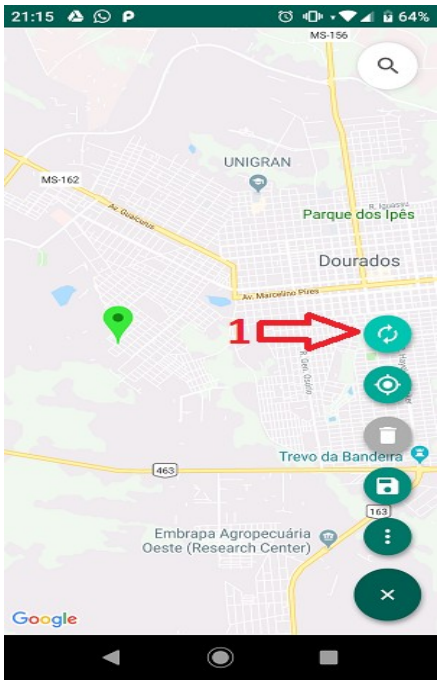
5.1.6 Carregar Arquivo

A próxima etapa ao iniciar o aplicativo é carregar o arquivo-texto. O processo será iniciado ao clicar no ícone indicado na **Figura 2.6** pelo número (1), quando a função for executada, internamente o arquivo-texto será processado, e para cada endereço é realizado a busca da latitude e longitude através da API Location do Google. As informações dos endereços, que estão separadas por ponto e vírgula no arquivo original, serão agrupadas em um novo arquivo com as informações recuperadas da API. O arquivo será renomeado para FM_Rota mais a data e hora sem os pontos de separação e um número com até quatro dígitos que será gerado randomicamente para compor o nome do arquivo, isso evitará nomes duplicados. Por exemplo: (FM_Rota + dia 09102019 + hora 1414 + número randômico 6541) ficará assim: FM_Rota0910201914146541.

Ao renomear o arquivo garantimos um padrão para realizar os processos internos que o aplicativo necessita. Após realizar todo o processo e gerar o novo arquivo contendo o novo nome os endereços, latitude e longitude as informações serão salvas no banco SQLite o nome

do arquivo na (**tabela arquivo**) e os endereços na (**tabela endereco**) para facilitar a realização das atualizações, inserções e remoções.

Figura2.6 – Opção de Sincronizar.



5.1.7 Sincronizar

Para indicar que as informações estão sendo sincronizadas um *dialog*[3] será iniciado a **Figura 2.7** demonstra o *dialog* sendo executado. Após esta etapa o *dialog* com a lista contendo os arquivos processados será executado, com as informações de: quantidade de arquivos, arquivos listados, quantidade de endereços por arquivos. **A Figura 2.8** mostra o *dialog* com essas informações.

Figura2.7 – Dialog Sincronização.

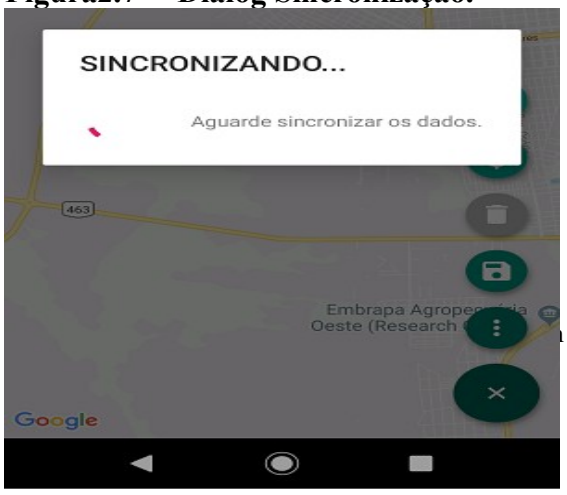
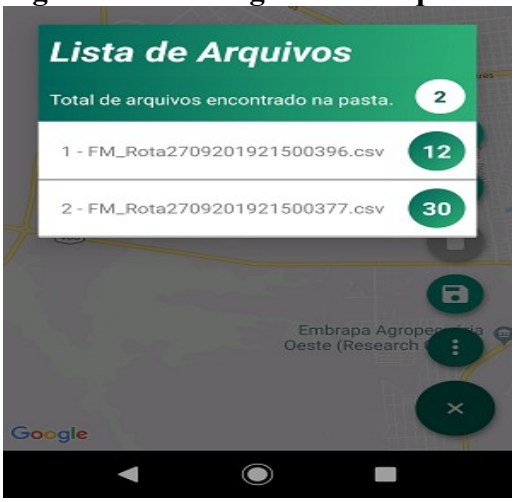


Figura2.8 – Dialog lista de arquivos.



5.1.8 Carregar Mapa

A carga dos endereços no mapa é feita pela interação com o *dialog* que lista os endereços indicado na **Figura 2.8**, onde o mesmo possui a opção enviar para o mapa. Se for escolhido a opção ***SIM*** os endereços aparecerão no mapa e serão indicados com marcadores em verde, assim como ilustrado na **Figura 2.9** o *dialog* e na **Figura 2.10** demonstra os endereços no mapa.

Figura2.9 – Dialog para carregar os endereços.

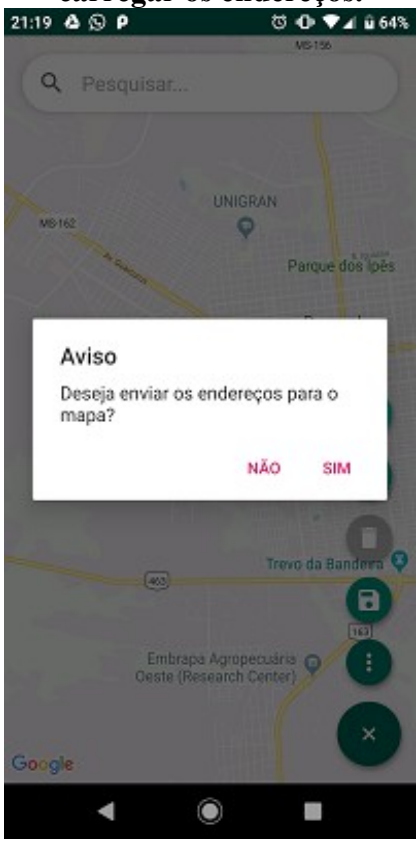
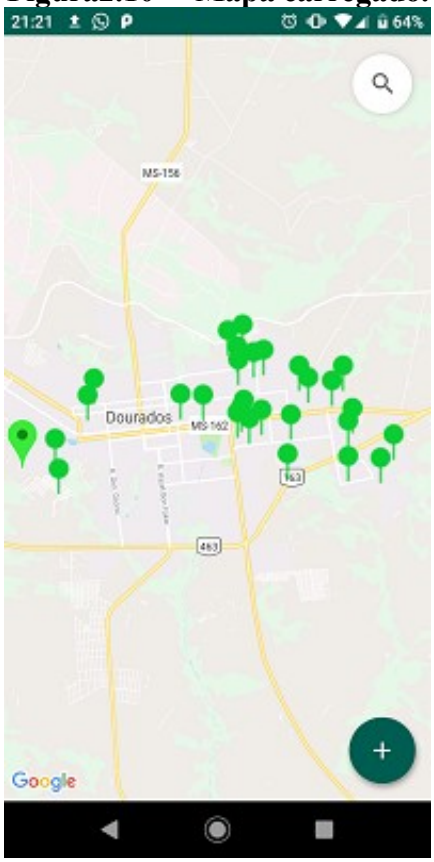


Figura2.10 – Mapa carregado.



5.1.9 Editar e Excluir Endereços

A partir do *dialog* que lista os endereços está funcionalidade só serve exclusivamente para editar ou excluir, poderá ser carregado o mapa ou editar os endereços, caso necessário. No caso escolher a opção ***NÃO*****,** indica a edição e a opção ***SIM*** envia os endereços para o mapa. A **Figura 2.11** e **2.12** demonstram essa funcionalidade de edição. Ao escolher a opção ***NÃO*** no *dialog* que lista os arquivos, o *dialog* de edição abrirá e terá a opção ***EDIÇÃO*** ou

SAIR*,** se a opção ***SAIR for a escolhida, o dialog fechará e a tela inicial será apresentada, já se for escolhida a opção ***EDIÇÃO***, será carregada uma lista com os endereços.

Dentro da lista cada endereço terá dois ícones uma caneta e uma lixeira, que servirão para indicar a edição ou exclusão das informações, a edição e exclusão será feita no nível de banco de dados. Uma vez feita a alteração ou exclusão não é possível retornar às informações originais.

Figura2.11 – Dialog de edição.

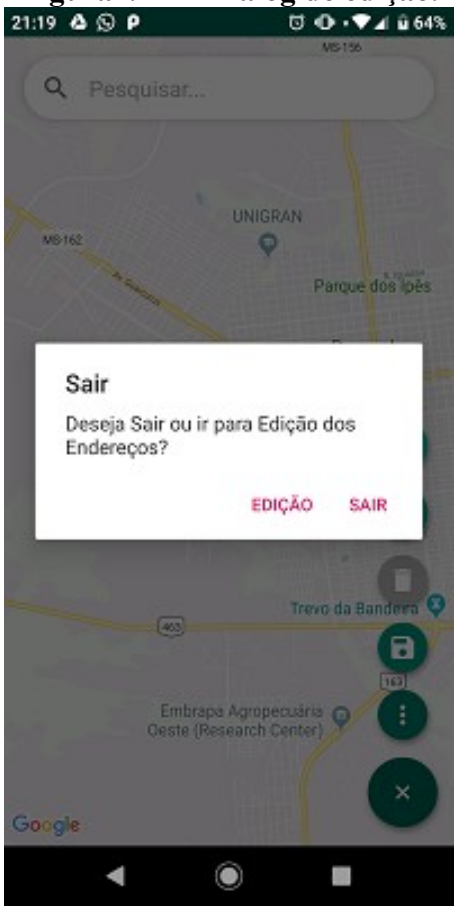


Figura2.12 – Lista de endereços edição.

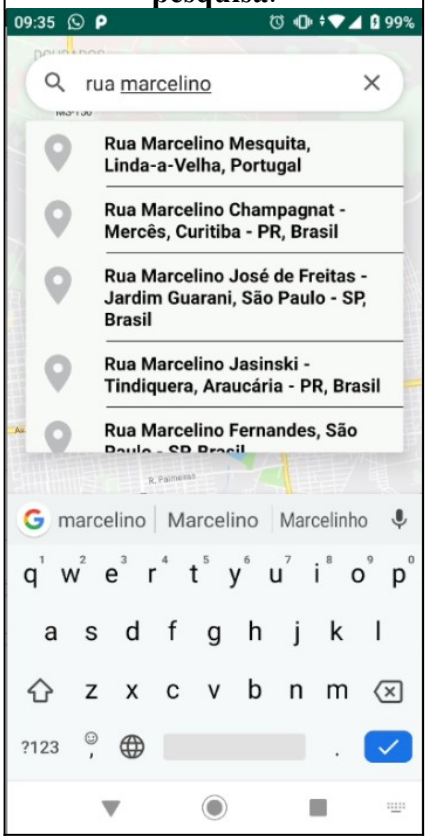
5.1.10 Inserir endereços por pesquisa

Para inserir um marcador no mapa sem a carga do arquivo-texto, o usuário poderá realizar a pesquisa no campo posicionado na parte superior da tela, o campo é retrátil, para possibilitar a melhor visualização do mapa, o campo de pesquisa usa o “motor” de busca dos servidores do Google, através da API Places.

Após realizar a pesquisa e selecionar o endereço apresentado, para enviar para o mapa, deve-se clicar no botão que indica o “Enter” no teclado do celular, dessa forma um marcador em verde será posicionado no mapa. Caso tenha um arquivo para carregar, esse endereço inserido pelo campo de pesquisa, será adicionado com os endereços do arquivo.

A **Figura 2.13** demonstra o campo de pesquisa.

Figura2.13 – Campo de pesquisa.



5.1.11 Gerar rotas

A etapa para gerar a rota consiste em ter, pelo menos, um endereço inserido, já que a posição do usuário será definido automaticamente e representado pelo marcador maior, dessa forma havendo dois pontos de referência assim é possível gerar uma rota entre estes dois pontos.

Quando os pontos forem ligados, a cor dos marcadores será alterado de verde para a cor vermelha e uma linha em azul definirá a rota, indicando a rota.

Ao carregar os endereços de um arquivo, onde o máximo é 30 endereços, a rota sempre será iniciada pelo marcador do usuário. Ele é o centro da rota, dessa forma um *click* no próximo marcador escolhido pelo usuário deverá ligar e alterar a cor do marcador, assim sucessivamente até que todos os marcadores estejam ligados. A **Figura 2.14** e **2.15** indica como a rota é gerada para um endereço ou vários endereços.

Figura2.14 – Iniciando uma rota.

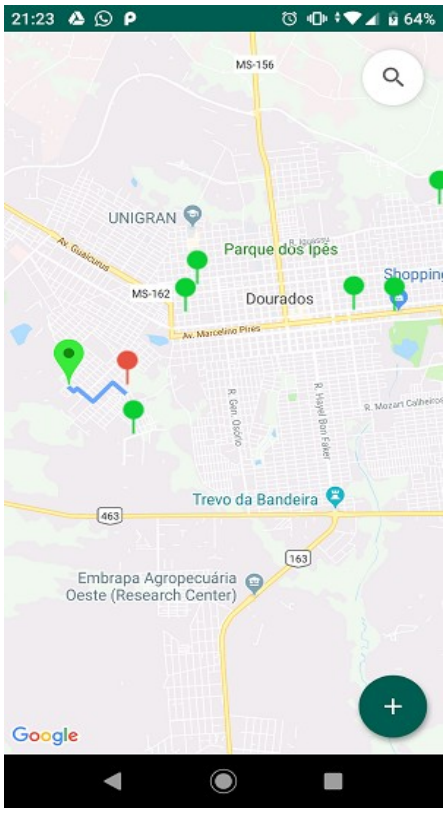
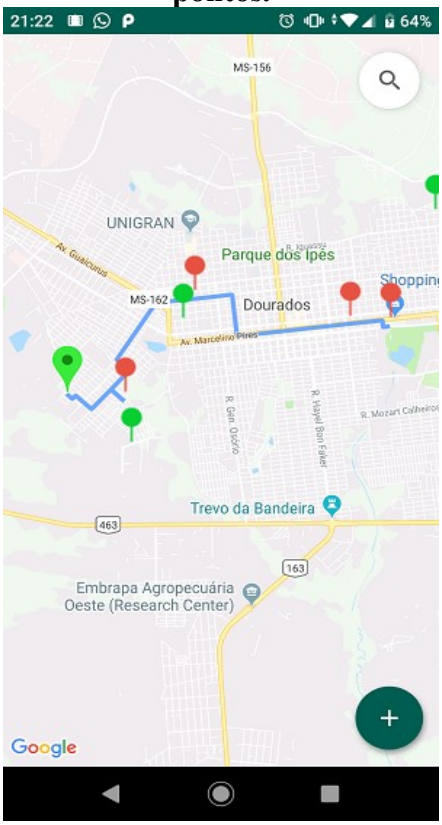


Figura2.15 – Conectando os pontos.



Para interagir com a rota o usuário terá a possibilidade de alterar, excluir e visualizar a informação referente aquele endereço, as etapas a seguir enumeram essa iteração:

1. Exclusão: foi tratada como uma lista no conceito computacional e temos as seguintes propriedades para permitir a exclusão, as **Figuras 2.16, 2.17, 2.18** e **2.19** demonstram como a exclusão funciona;
    - É necessário que exista no mínimo dois marcadores já conectados, a posição do usuário e mais um marcador de endereço, dessa forma ao clicar no marcador do endereço a exclusão da rota é realizada, a

Figura 2.17 demonstra dois marcadores, uma rota e a possibilidade de exclusão;

- Se mais de um marcador estiver conectado, ao clicar no primeiro marcador após o marcador do usuário, toda a rota é apagada, **Figura 2.16 e 2.17**, simbolizam os marcadores da **Figura 2.18** e **2.19** sendo excluídos, caso for clicado qualquer marcador sem ser o marcador do usuário ou o último marcador conectado a exclusão acontecerá sempre após o marcador clicado e apagará todos que estiverem conectados a frente dele, **Figura 2.17**.
- Ao clicar no último marcador nada é realizado já que a rota só é excluída se o marcador clicado estiver ligado a outro marcador;

Figura2.16 – Marcadores não ligados.

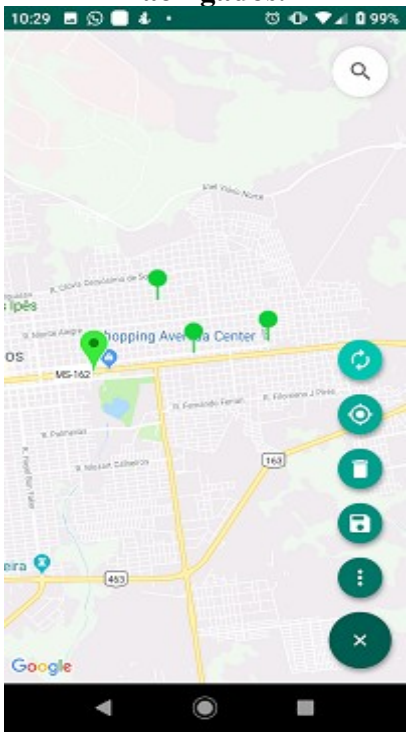
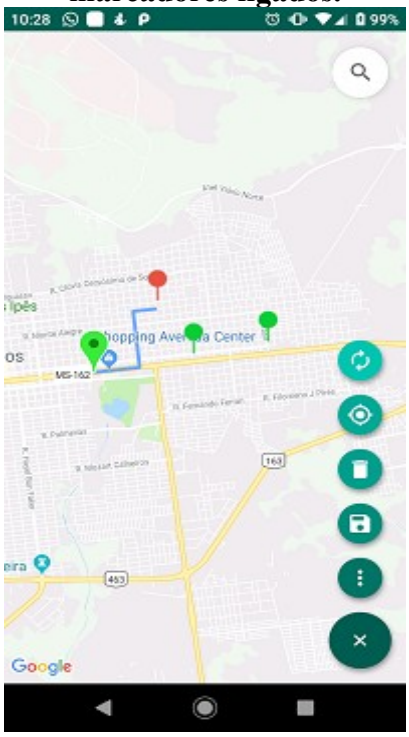


Figura2.17 – Rota 2 marcadores ligados.



As **Figuras 2.16** indica quando o mapa está carregado e a **Figura 2.17** indica o usuário conectando os marcadores, neste exemplo escolhe um marcador por preferência que será considerado o segundo, já que o primeiro marcador é o do usuário.

A **Figurar 2.18** demonstra o terceiro marcador conectado, na figura temos o marcador do usuário (primeiro), o segundo marcador escolhido e por último o terceiro marcador. Na

Figura 2.19 segue sequencialmente escolhendo por preferência os marcadores que deverão ser conectados.

Figura2.18 – Rota 3 marcadores.

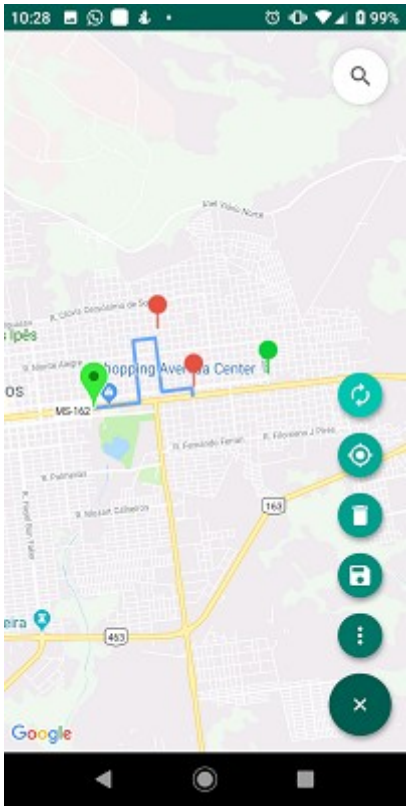
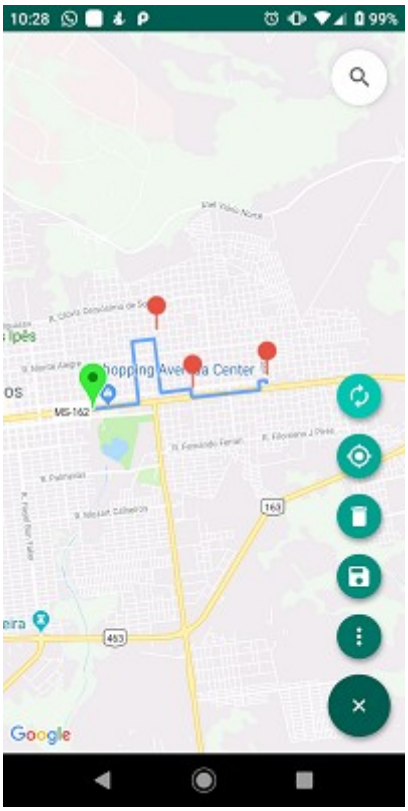
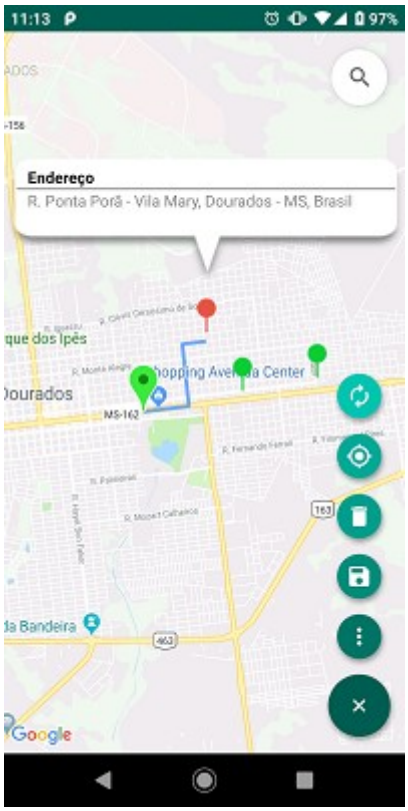


Figura2.19 – Rota 4 marcadores.



2. Inserção: o oposto da exclusão, deverá ocorrer a partir do marcador do usuário, ao escolher o marcador que deseja para gerar a rota uma linha será traçada e gerará a primeira rota, seguindo essa sequência escolher o próximo marcador e assim sucessivamente até gerar a rota desejada, cada marcador escolhido alterará a cor de verde para vermelho, as **Figuras 2.16, 2.17, 2.18** e **2.19** represe**n**tam tanto a exclusão como a inserção.
3. Para visualizar as informações dos marcadores, o marcador deverá ser pressionado por 2 segundos e um *dialog* será apresentado. **Figura 2.20**.

Figura2.20 – Dialog informativo.



5.1.12 Banco de Dados SQLite3

Para facilitar o desenvolvimento do aplicativo foi utilizado um banco de dados com duas tabelas, a tabela endereço e a tabela arquivo que se relacionam através da chave estrangeira fkidarquivo que está na tabela endereço e se liga ao id da tabela arquivo. A tabela arquivo contém os campos (idarquivo, nome_arquivo, processados, quant_arquivos), a tabela

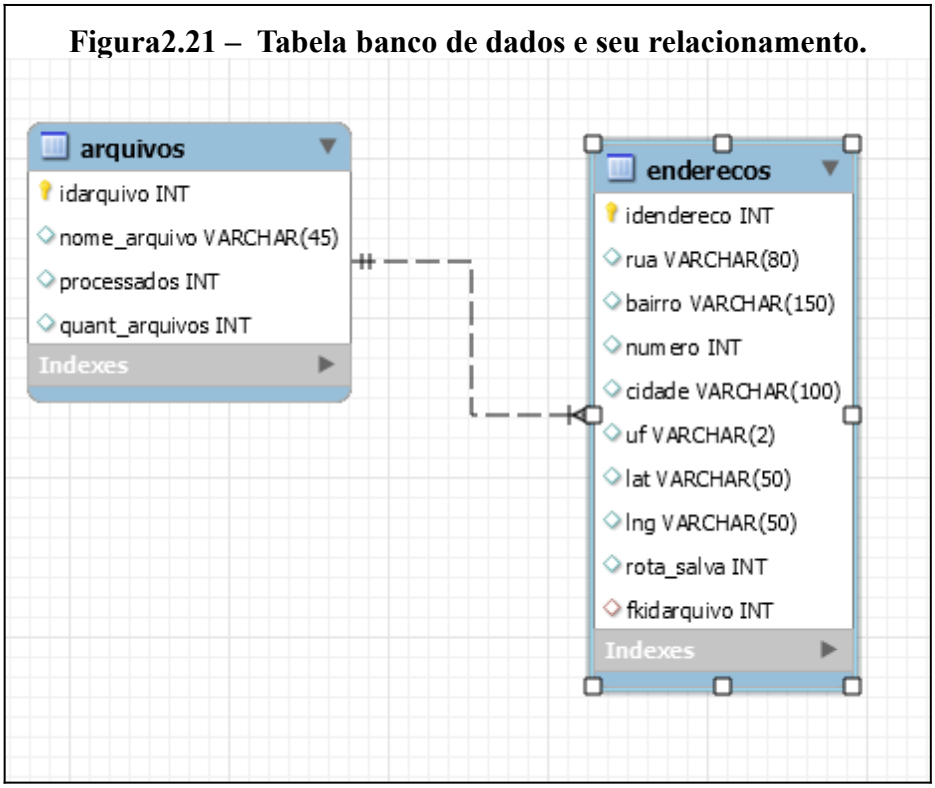
endereços contém os campos (idenderecos, rua, bairro, número, cidade, uf, lat, lng, rota_salva, fkidarquivo).

O relacionamento foi necessário devido em alguns momentos o aplicativo recupera o nome do arquivo e a consulta em uma única tabela poderia trazer muitos valores duplicados, dessa forma o relacionamento entre as tabelas facilita ao criar *SELECTS*[4], UPDATES[5].

O banco de dados permite trabalhar de forma rápida e prática com CRUD[6] e facilita quando é necessário realizar inserções, atualizações, remoções e pesquisas. O arquivo limita o trabalho com CRUD, pois o arquivo tem que ser alterado por completo ou desenvolver métodos para tais fins, o que inviabiliza todo o projeto deixando os processos mais demorados.

A **Figura 2.21** demonstra as tabelas e o seu relacionamento de 1 para N.

Figura2.21 – Tabela banco de dados e seu relacionamento.



4 Expressão que indica consulta em banco de dados.
5 Expressão que indica uma atualização em banco de dados.
6 Acrônimo do Inglês Create, Read, Update e Delete.

Capítulo 6

Conclusão

Neste trabalho vimos definições, conceitos, ferramentas e bibliotecas que foram utilizados para desenvolver o aplicativo. Foi apresentado aplicativos que permitem a utilização de um mapa para navegação por localidade, capazes de mostrar uma rota que se conecta por 2 ou até no máximo 10 endereços, no entanto, ao utilizar tais aplicativos, alterar, reorganizar ou modifica a rota conforme suas necessidades, não é permitido, visto que os aplicativos oferecem como opção a melhor ou menor rota e informações do trânsito em tempo real por compartilhamento de informações.

Definir uma rota com múltiplos endereços sequenciais é importante, pois atende um segmento do setor de serviço, que exige a criação de rotas personalizadas, flexíveis e que atendam uma realidade específica.

Devido às limitações dos aplicativos apresentados para uma determinada demanda, verificou-se a necessidade de implementar um aplicativo utilizando as APIs da Google, portanto foi implementado um aplicativo que agrega a facilidade e funcionalidade de manipular a rota com mais de 10 endereços, quando um serviço de visitação for realizado, assim contribuindo com agilidade no desempenho da tarefa.

Para conseguir desenvolver este projeto contamos com conceitos de rota, grafos, IDE Android Studio, Banco de dados SQLite e pesquisas em livros e na internet, com capacidade de fornecer o conhecimento necessário para atingirmos o nosso objetivo.

6.1 Trabalhos Futuros

Para melhorar o sincronismo das informações envio/recebimento utilizar o banco de dados Firebase NoSql[7], permitirá um sincronismo quando houver conexão com a internet, já que toda informação é armazenada localmente em um arquivo json (FIREBASE, 2019).

Opções como favoritos, históricos, endereços mais visitados, distância percorrida e tempo do trajeto também podem ser implementados.

Um algoritmo como o A* pela formulação de Dijkstra (CORMEN, c.24, p.470), poderá ser implementado para automatizar a criação do trajeto, ou sugerir o melhor caminho, se o usuário desejar.

7 Promove soluções de armazenamento de dados não relacionais.

Referência Bibliográfica

ANDRADE, Brian Castellan et al. Avaliação do Algoritmo A* na Heurística de Path-Planning em Ambientes de Jogos Utilizando o Motor Unity3D. 2016. 9 f. Dissertação (Mestrado) - Curso de Ciência da Computação, – Unidade Acadêmica de Ciências, Engenharias e Tecnologias, Universidade do Extremo Sul Catarinense, Criciúma, 2016. Cap. 31. Disponível em: <http://periodicos.unesc.net/sulcomp/article/view/3126/2856>. Acesso em: 05 nov. 2019.

BORDIN, M. V. Introdução a Arquitetura Android, Faculdade de Sistema de Informação, Sociedade Educacional 03 de Maio, 2012. Disponível em <Http://sites.setrem.com.br/stin/2012/anais/maycon.pdf>. Acessado em 18 setembro 2017.

BRAGA, Edgar Augusto da Silva. Modelagem de otimização do problema do caixeiro viajante com restrições de tempo, distância, confiabilidade via algoritmos genéticos. 2007. 64 f. Dissertação (Mestrado) - Curso de Engenharia de Produção, Universidade Federal do Pernambuco, Recife, 2007. Cap. 5. Disponível em: <http://repositorio.ufpe.br/bitstream/handle/123456789/5672/arquivo7291_1.pdf?sequence=1&isAllowed=y>. Acesso em: 20 jun. 2017.

CASTRO, Josué Pereira de. Um algoritmo evolucionário para a geração de planos de rotas. 1999. 85 f. Dissertação (Mestrado) - Curso de Engenharia de Produção e Sistemas, Universidade Federal de Santa Catariana, Florianópolis, 1999. Cap. 7. Disponível em: <https://repositorio.ufsc.br/bitstream/handle/123456789/80798/150624.pdf?sequence=1&isAllowed=y>. Acesso em: 20 jun. 2017.

CERVIERI, Alexandre; PY, Mónica – *Algoritmo para a resolução do problema do caixeiro viajante* [Em Linha]. Porto Alegre: Instituto de Informática - UFRGS, 2000. [Consult. 15 abr. 2009].

CORREIA, A. G. S. Aplicações e Serviços Baseados em Localização, Pontifícia Universidade Católica do Rio de Janeiro, 2004. Disponível em <http://www-di.inf.pucrio.Br/~endler/courses/Mobile/Monografias/04/AdolfoCorreia-Mono.pdf> acessado 14 agosto 2017.

CORMEN, Thomas H. et al. Algoritmos: Teoria e Prática. 4. ed. São Paulo: Elsevier, 2002. 916 p. Vandeberg D. de Souza. Disponível em: <http://www.inf.ufrgs.br/~tsrodrigues/utilidades/cormem.pdf>. Acesso em: 30 out. 2019.

COSTA, Polyanna Possani. Teoria dos Grafos e suas Aplicações. 2011. 77 f. Dissertação (Mestrado) - Universidade Estadual Paulista “Júlio de Mesquita Filho”, Instituto de Geociências e Ciências Exatas Campus de Rio Claro. Disponível em: <http://www.rc.unesp.br/tmelo/diss-polyanna.pdf. Acessado em 14 de novembro de 2017.

DEVELOPERS . Documentação Google – Disponível em: <https://developers.google.com/maps/documentation/android-sdk/intro?hl=pt-br> Acessado em 10 de outubro de 2019.

DEVELOPERS, Documentação Google sobre Android Studio – Disponível em: <https://developer.android.com/studio/intro/ > Acessado em 04 de novembro de 2019.

ÉPOCA NEGÓCIOS. Brasil tem 230 milhões de smartphone em uso. Disponível em: <https://epocanegocios.globo.com/Tecnologia/noticia/2019/04/brasil-tem-230-milhoes-de-smartphones-em-uso.html> Acessado em 08 de outubro 2019.

EL PAÍS. Android já é o sistema operacional mais usado do mundo. Disponível em: <https://brasil.elpais.com/brasil/2017/04/04/tecnologia/1491296467_396232.html >Acessado em 08 de outrubro de 2019.

FIREBASE. Firebase Real Time. Disponível em: <https://firebase.google.com/docs/database/?hl=pt-br > acessado em 05 de novembro de 2019.

FEOFILOFF, P. Complexidade computacional e problemas NP-completos. Disponível em: https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/NPcompleto2.html.

FORTNOW, Lance (2009). The Status of the P Versus NP Problem. Communications of the ACM, vol.52 (nº9), p.78-96. Acesso em 03 de maio de 2017, disponível em https://cacm.acm.org/magazines/2009/9/38904-the-status-of-the-p-versus-np-problem/fulltext#PageTop

GOLVEIA, Alexandre. O que é uma API? - Universidade Positivo, 2016 Engenharia da Computação. Artigos. Disponível em: < http://www.up.edu.br/blogs/engenharia-da-computacao/2016/07/01/o-que-e-uma-api/. Acessado em 14 de novembro de 2017.

GOOGLE. (24 de março de 2017). Introdução à Google Maps Android API. Fonte: Google Maps APIs: https://developers.google.com/maps/documentation/android-api/intro?hl=pt-br

GOOGLE. Google Mapas Android API V2. Disponível em: <Https://developers.google.com/maps/documentation/android/map?hl=pt-br>. Acessado em 17 de setembro 2017.

GOOGLE. Painéis Google Android. Disponível em: <https://developer.android.com/about/dashboards/index.html>. Acessado em 26 de setembro 2017.

GOOGLE. Primeiros passos com NDK. Disponível em: <https://developer.android.com/ndk/guides/index.html?hl=pt-br. Acessado em 14 de novembro de 2017.

GOOGLE. Introdução ao Android. Disponível em: <Https://developer.android.com/guide/index.html?hl=pt-br. Acessado em 14 de novembro de 2017.

GOOGLE. Uso educado das APIs do Google. Disponível em: <https://developers.google.com/maps/documentation/directions/web-service-best-practices >. Acessado em 09 de outubro de 2019.

JOHNSON David S. e GAREY Michael R. (1979). Computer and Intractability: a Guide to the Theory of NPCompleteness. Freeman. Nova York: Freeman.

OLIVEIRA, S. L. Desenvolvimento de um aplicativo Android utilizando Geolocalização, UEMS Universidade Estadual do Estado do Mato Grosso, 2014. Disponível em <http://www.comp.uems.br/~ricardo/PFCs/PFC%20151.pdf>. Acessado 18 de setembro de 2017.

ROCHA, Ana Maria A.C., Fernandes, Edite M.G.P., SOARES, João Luís C., Aplicação do algoritmo volumétrico a resolução aproximada e exata do problema do caixeiro viajante assimétrico, Investigação Operacional, Vol. 25 (2), 277-294, 2005. ISBN 978-3-540-69840-1. Disponível em:
http://repositorium.sdum.uminho.pt/bitstream/1822/9723/1/ATSP_port_2005_06_13.pdf.
Acessado em 10 de novembro de 2017.

SILVEIRA, J. F. Porto da, Caixeiro Viajante. Disponível em
http://www.mat.ufrgs.br/~portosil/caixeiro.html. Acessado em 10 de novembro de 2017.

SILVA, Mariana Oliveira da. PROBLEMA DE COBERTURA POR VÉRTICES EM REDES COMPLEXAS. 2013. 50 f. Dissertação (Mestrado) - Curso de Computação Aplicada, Departamento Acadêmico de Informática Programa de Pós-graduação em Computacão Aplicada, Universidade Tecnológica do Paraná, Paraná, 2013. Cap. 2. Disponível em: <http://repositorio.utfpr.edu.br/jspui/bitstream/1/734/1/CT_PPGCA_M_Silva%2C %20Mariana%20Oliveira%20da_2013.pdf>. Acesso em: 02 nov. 2018.

TERRA. 3,8 milhões de smartphones são vendidos no mundo diariamente. Disponível em: <http://computerworld.com.br/38-milhoes-de-smartphones-sao-vendidos-no-mundo-diariamente.> Acessado em 14 de novembro de 2017.